Advance Diploma in Information Technology Application

ADITA

Study Material

Youth Computer Training Centre

Topics Covered:

- **C++, Visual C++ with OOPS**
- **VB.Net**
- **JavaScript**
- **VB Script**

Disclaimer

This study material is designed to provide information about 'Advance Diploma in Information Technology Application'. While every effort has been made to make this study material as complete, comprehensive and as accurate as possible, no warranty or fitness is to be implied. The information is provided on an "as is" basis. Hewlett Packard Enterprise shall not have any liability or responsibility to any person or entity with respect to any loss or damages arising from the information contained in this study material. Internet websites offered as citations and/ or sources for further information may have changed or disappeared between the time this study material is written and the time when it may be read by the reader.

Contents

Chapter 1: C++, Visual C++ with OOPS

Introduction

C++ is a high-level, general-purpose programming language that is widely used in software development. It combines the power of low-level programming with the flexibility of high-level languages, making it suitable for a wide range of applications, including system software, game development, real-time applications, and more.

A Brief History of C++

C++ has a rich history that dates back to the late 1970s when it was developed as an extension of the C programming language. Here's a timeline of its evolution:

1. Origins (1979–1983)
   - **Bjarne Stroustrup**, a Danish computer scientist, began working on C++ in **1979** at Bell Labs in New Jersey.
   - Stroustrup aimed to create a language that combined the efficiency of **C** with additional features for managing large software projects.
   - Initially called **"C with Classes"**, it introduced the concept of **classes**, inspired by Simula (an early object-oriented programming language).
2. Birth of C++ (1983)
   - The language was officially renamed **C++** in **1983**.
     The name reflects its evolutionary nature: "++" is the increment operator in C, symbolizing an enhancement of C.
   - During this period, features like **virtual functions**, **function overloading**, and **default arguments** were introduced.
3. Early Standardization (1985–1998)
   - **1985:** The first edition of *The C++ Programming Language* by Bjarne Stroustrup was published, serving as an informal specification of the language.
   - **1989:** C++ 2.0 was released, introducing significant features like **multiple inheritance**, **abstract classes**, and **static member functions**.
   - **1990s:** The **Standard Template Library (STL)** was developed by Alexander Stepanov, introducing generic programming concepts with templates and collections like vectors, lists, and maps.
   - **1998:** The first official **C++ standard (ISO/IEC 14882:1998)**, often referred to as **C++98**, was published, consolidating the language and STL.
4. Modernization Era (2011–Present)
   - **2011 (C++11):** A major update introduced features like:
     - **Smart pointers** (memory management),
     - **Lambda expressions** (anonymous functions),
     - **Auto keyword** (type inference),
     - **Move semantics** (efficient resource handling).

- **2014 (C++14):** A minor revision added enhancements like **generic lambdas** and bug fixes.
- **2017 (C++17):** Introduced features like:
  - **std::optional**,
  - **std::variant**,
  - **Structured bindings**.
- **2020 (C++20):** Focused on simplification and powerful tools:
  - **Concepts** (constraints for templates),
  - **Ranges** (more expressive algorithms),
  - **Coroutines** (asynchronous programming).

5. Current and Future Trends
   - **C++23 (expected in 2023):** Focused on refining existing features and improving usability.
   - C++ continues to evolve, maintaining its relevance in high-performance computing, system programming, and application development.

C++ has consistently adapted to meet the demands of modern programming while maintaining backward compatibility, making it one of the most enduring and versatile programming languages in use today.

Difference Between C and C++

C and C++ are closely related programming languages, but they have significant differences in their features, use cases, and programming paradigms. Here's a comparison:

1. Programming Paradigm
   - **C:** Procedural programming language.
     - Focuses on **functions** and **procedures**.
     - Programs are written as a sequence of instructions.
     - Example: Functional decomposition.
   - **C++:** Multi-paradigm language.
     - Supports **object-oriented programming (OOP)**, **procedural programming**, and **generic programming**.
     - Encourages modularity and code reusability through classes and objects.

2. Object-Oriented Programming
   - **C:** Does not support OOP.
     - Lacks features like **classes**, **inheritance**, and **polymorphism**.
   - **C++:** Fully supports OOP.
     - Includes **classes**, **objects**, **inheritance**, **encapsulation**, **abstraction**, and **polymorphism**.

3. Standard Library
   - **C:** Provides a limited standard library.

- Includes functions for input/output (printf, scanf), string manipulation, memory management, etc.

- **C++:** Includes a rich **Standard Template Library (STL)**.
  - Provides ready-to-use templates for **data structures** (e.g., vector, map) and **algorithms** (e.g., sort, search).

4. Memory Management
   - **C:** Manual memory management using functions like malloc, calloc, free.
   - **C++:** Provides **manual** and **automatic memory management**.
     - Introduced **new** and **delete** operators for dynamic memory allocation.
     - Includes **smart pointers** (from C++11) for safer memory management.

5. Compatibility
   - **C:** Highly compatible across platforms.
   - **C++:** Backward-compatible with C.
     - Most C code can be compiled in a C++ compiler with minimal changes.

6. Features and Syntax

Feature	C	C++
Namespaces	Not supported	Supported (avoids naming conflicts).
Function Overloading	Not supported	Supported.
Operator Overloading	Not supported	Supported.
Inline Functions	Limited	Fully supported.
Error Handling	Uses setjmp and longjmp	Uses **try-catch** blocks.

7. Performance
   - **C:** Slightly faster for low-level programming due to minimal runtime overhead.
   - **C++:** Slightly slower than C in some cases because of features like OOP and abstraction, but still highly efficient.

8. Use Cases

Language	Primary Use Cases
C	System programming, embedded systems, device drivers, operating systems (e.g., Linux).
C++	Game development, real-time simulations, GUI applications, high-performance software (e.g., Adobe, Unreal Engine).

Key Features of OOP

1. Classes and Objects:
    - **Class:** Blueprint for objects.
    - **Object:** Instance of a class.
2. Encapsulation:
    - Bundles data and methods into a single unit.
    - Protects data using access specifiers (private, public, protected).
3. Abstraction:
    - Hides complex details; shows only essential features.
    - Achieved using abstract classes or interfaces.
4. Inheritance:
    - Enables a class to inherit attributes and methods from another class.
    - Promotes code reuse.
5. Polymorphism:
    - Functions or methods behave differently based on context.
    - Types: Compile-time (function overloading) and Run-time (virtual functions).

Benefits of OOP

- Code reusability through inheritance.
- Easier to debug, modify, and extend programs.
- Better organization of large software projects.
- Enhances modularity by dividing the program into smaller, manageable parts.

By mastering these features, developers can create robust, efficient, and scalable software solutions using OOP principles.

Data Types in C++

Data types define the type of data a variable can hold. They are categorized as:

1. Basic Data Types
    - **Integer Types:** int, short, long, long long
    - **Floating-point Types:** float, double, long double
    - **Character Type:** char
    - **Boolean Type:** bool
2. Derived Data Types
    - **Arrays:** Collection of elements of the same type.
    - **Pointers:** Variables that store memory addresses.
    - **References:** Alternative names for existing variables.

3. User-defined Data Types
   - **Structures (struct)**
   - **Classes**
   - **Enumerations (enum)**
4. Void
   - Represents no type. Used for functions that do not return a value.

Operators in C++

Operators perform operations on variables and values. Categories include:

1. Arithmetic Operators

   +, -, *, /, %

   Example: int sum = a + b;
2. Relational (Comparison) Operators

   ==, !=, <, >, <=, >=

   Example: if (a > b)
3. Logical Operators

   && (AND), || (OR), ! (NOT)

   Example: if (a > 0 && b < 10)
4. Assignment Operators

   =, +=, -=, *=, /=, %=

   Example: a += 5; (equivalent to a = a + 5;)
5. Bitwise Operators

   & (AND), | (OR), ^ (XOR), ~ (NOT), << (Left Shift), >> (Right Shift)

   Example: int c = a & b;
6. Increment/Decrement Operators

   ++ (increment), -- (decrement)

   Example: a++; (post-increment)
7. Conditional (Ternary) Operator

   ? :

   Example: int max = (a > b) ? a : b;
8. Other Operators
   - **Sizeof:** sizeof(type) returns the size of a type.
   - **Typecast:** (type) converts a variable to a specific type.
   - **Comma:** , separates expressions.

Statements in C++

Statements are instructions executed by the compiler.

1. Declaration Statement
   - Defines variables.
     Example: int a = 5;
2. Assignment Statement
   - Assigns values to variables.
     Example: a = 10;
3. Input/Output Statements
   - **Input:** cin >> variable;
   - **Output:** cout << "Text";
     Example:

```
int a;
cin >> a;
cout << "Value is: " << a;
```

4. Control Statements
   - **Conditional:** if, if-else, switch.
     Example:

```
if (a > 0) { cout << "Positive"; }
```

   - **Loops:** for, while, do-while.
     Example:

```
for (int i = 0; i < 5; i++) { cout << i; }
```

5. Jump Statements
   - **Break:** Exits a loop or switch.
   - **Continue:** Skips the rest of the loop iteration.
   - **Return:** Exits a function and optionally returns a value.

Variables

What are Variables?

In C++, a **variable** is a named location in memory used to store data that can change during the execution of a program. Variables make it easier to work with data in programs by providing a way to access and manipulate that data.

Declaring Variables

To declare a variable, specify its type followed by its name. You can also assign a value to it during the declaration.

Syntax:

```
type variableName = value; // Optional initialization
```

Example:

```
int age;          // Declaring an integer variable
float price = 9.99; // Declaring and initializing a float variable
```

Common Data Types for Variables in C++

Data Type	Description	Example Values
int	Integer type for whole numbers	-10, 0, 100
float	Floating-point type for decimal numbers	3.14, -0.01
double	Double-precision floating-point numbers	3.14159265359
char	Character type for single characters	'A', 'z'
bool	Boolean type for true or false values	true, false
string	Stores a sequence of characters (requires <string> library)	"Hello"

Examples of Declaring and Using Variables

```
#include <iostream>
#include <string> // Required for string type
using namespace std;
int main() {
    int age = 25;          // Integer variable
    float height = 5.9;    // Float variable
    char grade = 'A';      // Character variable
    bool isStudent = true; // Boolean variable
    string name = "Alice"; // String variable
    // Output the variables
    cout << "Name: " << name << endl;
    cout << "Age: " << age << endl;
    cout << "Height: " << height << " feet" << endl;
    cout << "Grade: " << grade << endl;
    cout << "Is Student: " << isStudent << endl;
    return 0;
}
```

Output:

```
Name: Alice
Age: 25
Height: 5.9 feet
Grade: A
Is Student: 1
```

(Note: Boolean true is printed as 1 and false as 0.)

Rules for Naming Variables

1. Must start with a letter or an underscore (_).
2. Cannot contain spaces or special characters.
3. Cannot use C++ reserved keywords (int, return, etc.).
4. Variable names are case-sensitive (age and Age are different).

Types of Variable Initialization

1. **Default Initialization**:

   int x; // No value assigned; contains garbage data
2. **Direct Initialization**:

   int x = 10;
3. **Uniform Initialization (C++11 and later)**:

   int x {10};

Understanding variables is the foundation of programming as they hold the data you manipulate in your programs!

First Program Using C++

Writing your first program in C++ is exciting! Here's a classic example: the "Hello, World!" program, which is often used as a beginner's introduction to programming.

Code Example:

```
#include <iostream> // Include the library for input and output
int main() {
   cout << "Hello, World!" << endl; // Output a message to the console
   return 0; // Indicate that the program ended successfully
}
```

Explanation:

1. **#include <iostream>**:
   - This includes the standard input/output stream library, which allows the program to perform input and output operations.
2. **int main()**:
   - This is the entry point of the program. Every C++ program starts execution from the main() function.
3. **cout << "Hello, World!" << endl;**:
   - cout is used to print to the console.
   - << is the insertion operator, used to send the string "Hello, World!" to the output stream.
   - endl is used to end the line and flush the output buffer.
4. **return 0;**:
   - This tells the operating system that the program finished successfully.

Control Statement

In C++, **control statements** are used to control the flow of program execution based on conditions or repetition. These include decision-making statements, loops, and jump statements.

1. Decision-Making Statements

a) if Statement

Executes a block of code if a condition is true.

Syntax:

```
if (condition) {
        // Code to execute if condition is true
}
```

Example:

```
int x = 10;
if (x > 5) {
    cout << "x is greater than 5" << endl;
}
```

b) if-else Statement

Executes one block of code if a condition is true, otherwise executes another.

Syntax:

```
if (condition) {
    // Code to execute if condition is true
} else {
    // Code to execute if condition is false
}
```

Example:

```
int x = 3;
if (x > 5) {
    cout << "x is greater than 5" << endl;
} else {
    cout << "x is less than or equal to 5" << endl;
}
```

c) if-else if Ladder

Checks multiple conditions in sequence.

Syntax:

```
if (condition1) {
    // Code to execute if condition1 is true
} else if (condition2) {
```

```
    // Code to execute if condition2 is true
} else {
    // Code to execute if no conditions are true
}
```

Example:

```
int x = 10;
if (x > 15) {
    cout << "x is greater than 15" << endl;
} else if (x == 10) {
    cout << "x is equal to 10" << endl;
} else {
    cout << "x is less than 10" << endl;
}
```

d) switch Statement

Tests a variable against multiple cases.

Syntax:

```
switch (variable) {
    case value1:
        // Code for case value1
        break;
    case value2:
        // Code for case value2
        break;
    default:
        // Code if no case matches
}
```

Example:

```
int day = 3;
switch (day) {
    case 1:
        cout << "Monday" << endl;
        break;
    case 2:
        cout << "Tuesday" << endl;
        break;
    case 3:
        cout << "Wednesday" << endl;
        break;
    default:
        cout << "Invalid day" << endl;
}
```

2. Loop Statements

a) for Loop

Repeats a block of code a specific number of times.

Syntax:

```
for (initialization; condition; update) {
    // Code to execute
}
```

Example:

```
for (int i = 1; i <= 5; i++) {
    cout << i << " ";
}
```

Output: 1 2 3 4 5

b) while Loop

Repeats a block of code while a condition is true.

Syntax:

```
while (condition) {
    // Code to execute
}
```

Example:

```
int i = 1;
while (i <= 5) {
    cout << i << " ";
    i++;
}
```

Output: 1 2 3 4 5

c) do-while Loop

Executes a block of code at least once, then repeats while the condition is true.

Syntax:

```
do {
    // Code to execute
} while (condition);
```

Example:

```
int i = 1;
do {
    cout << i << " ";
    i++;
} while (i <= 5);
```

Output: 1 2 3 4 5

3. Jump Statements

a) break

Exits the nearest enclosing loop or switch.

Example:

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) break;
    cout << i << " ";
}
```

Output: 1 2

b) continue

Skips the current iteration of the loop and moves to the next iteration.

Example:

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) continue;
    cout << i << " ";
}
```

Output: 1 2 4 5

c) return

Exits a function and optionally returns a value.

Example:

```
int add(int a, int b) {
    return a + b;
}
```

Array

What is an Array?

In C++, an **array** is a collection of elements of the same data type, stored at contiguous memory locations. It is used to store multiple values in a single variable, instead of declaring separate variables for each value.

Declaring an Array

Syntax:

type arrayName[size];

- **type**: The data type of the elements (e.g., int, float, char).
- **arrayName**: The name of the array.
- **size**: The number of elements the array can hold.

Example:

```
int numbers[5]; // Declares an array of integers with 5 elements
```

Initializing an Array

1. **During Declaration:**

   int numbers[5] = {10, 20, 30, 40, 50};

2. **Partially Initialized:**

   Uninitialized elements are set to 0.

   int numbers[5] = {10, 20}; // Remaining elements will be 0

3. **Implicit Size:**

   Let the compiler deduce the size.

   int numbers[] = {10, 20, 30, 40, 50};

Accessing Array Elements

Array elements are accessed using **indices** (starting from 0).

Syntax:

```
arrayName[index];
```

Example:

```
#include <iostream>
using namespace std;

int main() {
   int numbers[5] = {10, 20, 30, 40, 50};
   // Accessing elements
   cout << "First element: " << numbers[0] << endl;
   cout << "Second element: " << numbers[1] << endl;

   return 0;
}
```

Output:

```
First element: 10
Second element: 20
```

Iterating Over an Array

Use loops to iterate through arrays.

Example:

```
#include <iostream>
using namespace std;
int main() {
   int numbers[5] = {10, 20, 30, 40, 50};
```

```
    // Using a for loop
    for (int i = 0; i < 5; i++) {
        cout << "Element " << i << ": " << numbers[i] << endl;
    }
    return 0;
}
```

Output:

```
Element 0: 10
Element 1: 20
Element 2: 30
Element 3: 40
Element 4: 50
```

Types of Arrays

1. Single-Dimensional Array:
   Example:
   ```
   int arr[3] = {1, 2, 3};
   ```
2. Multi-Dimensional Array:
   Example:
   ```
   int matrix[2][3] = {
       {1, 2, 3},
       {4, 5, 6}
   };
   ```

Advantages of Arrays

1. Allows storing multiple elements under a single name.
2. Efficient memory usage for large data sets.
3. Easy to traverse and manipulate using loops.

Limitations of Arrays

1. Fixed size: The size must be known at compile-time.
2. All elements must be of the same data type.
3. No built-in methods for dynamic resizing or searching.

Functions

What is a Function?

In C++, a **function** is a block of code that performs a specific task. Functions make programs modular, reusable, and easier to understand. You can call a function as many times as needed within a program.

Types of Functions

1. **Built-in Functions**: Predefined functions in libraries (e.g., sqrt() from <cmath>).
2. **User-Defined Functions**: Functions created by the programmer.

Defining a Function

A function consists of:

1. **Return type**: Specifies the data type of the value returned by the function.
2. **Function name**: Identifies the function.
3. **Parameters (optional)**: Inputs to the function.
4. **Body**: The code executed when the function is called.

Syntax:

```
returnType functionName(parameter1, parameter2, ...) {
   // Function body
   return value; // Optional, based on return type
}
```

Example: User-Defined Function

```
#include <iostream>
using namespace std;
// Function to add two numbers
int add(int a, int b) {
   return a + b; // Return the sum
}
int main() {
   int num1 = 5, num2 = 10;
   int result = add(num1, num2); // Call the function
   cout << "The sum is: " << result << endl;
   return 0;
}
```

Output:

```
The sum is: 15
```

Function Declaration, Definition, and Call

1. Declaration (Prototype):
    - Informs the compiler about the function's existence.
    - Syntax: returnType functionName(parameterList);
    - Example:

```
int add(int, int);
```

2. Definition:
    - Provides the actual implementation.
    - Example:

```
int add(int a, int b) {
 return a + b;
}
```

3. Call:
    - Executes the function.
    - Example:

```
int result = add(5, 10);
```

Void Functions

Functions that do not return a value use the void return type.

Example:

```
#include <iostream>
using namespace std;
// Void function
void greet() {
   cout << "Hello, World!" << endl;
}
int main() {
   greet(); // Call the function
   return 0;
}
```

Output:

Hello, World!

Parameter Passing

1. Pass by Value:
    - Copies the value of arguments.
    - Changes made inside the function do not affect the original variables.

    Example:

```
void increment(int x) {
        x++;
}
```

2. Pass by Reference:
    - Passes the address of arguments.
    - Changes made inside the function affect the original variables.

    Example:

```
void increment(int &x) {
        x++;
}
```

Default Arguments

Functions can have default values for parameters.

Example:

```
int add(int a, int b = 5) { // b has a default value
    return a + b;
}
int main() {
    cout << add(10) << endl;    // Uses default b = 5
    cout << add(10, 20) << endl; // Overrides default
    return 0;
}
```

Output:

15

30

Inline Functions

Small functions can be declared as inline to reduce function call overhead. The compiler replaces the function call with the function code.

Example:

```
inline int square(int x) {
    return x * x;
}
```

Advantages of Functions

1. Code reusability.
2. Improved readability and modularity.
3. Easier debugging and maintenance.
4. Avoids code repetition.

Functions are a fundamental concept in programming that help in structuring and organizing your code effectively.

Class & Object

Class and Object in C++

In C++, **classes** and **objects** are the foundation of Object-Oriented Programming (OOP). They allow you to model real-world entities in your program.

What is a Class?

A **class** is a blueprint or template for creating objects. It defines properties (data members) and behaviors (member functions) that the objects will have.

Syntax:

```
class ClassName {
public:
    // Data members (variables)
    // Member functions (methods)
};
```

What is an Object?

An **object** is an instance of a class. It represents a real-world entity with specific properties and behaviors.

Example: Class and Object

```
#include <iostream>
using namespace std;
// Define a class
class Car {
public:
    // Data members
    string brand;
    int year;
    // Member function
    void displayInfo() {
        cout << "Brand: " << brand << endl;
        cout << "Year: " << year << endl;
    }
};
int main() {
    // Create an object of the class
    Car car1;
    // Access and set data members
    car1.brand = "Toyota";
    car1.year = 2020;
    // Call the member function
    car1.displayInfo();
    return 0;
}
```

Output:

```
Brand: Toyota
Year: 2020
```

Access Specifiers

Access specifiers determine the visibility of class members. The three main types are:

1. public:
   - Members are accessible from outside the class.
2. private:
   - Members are only accessible within the class.
3. protected:
   - Members are accessible within the class and by derived classes.

Example:

```
class Example {
private:
   int privateVar; // Not accessible outside
public:
   int publicVar; // Accessible from outside
};
```

Features of Classes and Objects

1. Encapsulation:
   - Wrapping data (variables) and functions into a single unit (class).
   - Example:

```
class Example {
private:
   int data; // Encapsulated data
public:
   void setData(int value) {
      data = value;
   }
   int getData() {
      return data;
   }
};
```

2. Abstraction:
   - Hiding implementation details from the user and showing only the essential features.
3. Inheritance:
   - Reuse and extend the functionality of existing classes.
4. Polymorphism:
   - Using the same function name for different purposes (e.g., function overloading, overriding).

Constructor & Destructor

Constructor

A **constructor** is a special function in a class that is automatically called when an object of the class is created. It is typically used to initialize data members of the class.

Characteristics of Constructors:

1. The constructor name is the same as the class name.
2. It has no return type, not even void.
3. It is called automatically when an object is instantiated.
4. It can be overloaded (i.e., multiple constructors can be defined with different parameter lists).

Types of Constructors

1. **Default Constructor**:
   - A constructor with no parameters.
   - Automatically provided by the compiler if no constructor is defined by the programmer.

   Example:

```
class Car {
public:
        Car() { // Default constructor
        cout << "Default constructor called" << endl;
         }
};
int main() {
   Car car1; // Calls the default constructor
   return 0;
}
```

   Output:

```
Default constructor called
```

2. **Parameterized Constructor**:
   - A constructor that accepts arguments to initialize an object with specific values.

   Example:

```
class Car {
public:
   string brand;
   Car(string b) { // Parameterized constructor
      brand = b;
   }
```

```
    void display() {
        cout << "Brand: " << brand << endl;
    }
};
int main() {
    Car car1("Toyota"); // Calls the parameterized constructor
    car1.display();
    return 0;
}
```

Output:

```
Brand: Toyota
```

3. **Copy Constructor**:
   - A constructor that initializes an object by copying another object of the same class.
   - The compiler provides a default copy constructor, but you can define your own.

Example:

```
class Car {
public:
    string brand;
    Car(string b) { // Parameterized constructor
        brand = b;
    }
    Car(const Car &c) { // Copy constructor
        brand = c.brand;
    }
    void display() {
        cout << "Brand: " << brand << endl;
    }
};
int main() {
    Car car1("Tesla");  // Calls the parameterized constructor
    Car car2 = car1;    // Calls the copy constructor
    car2.display();
    return 0;
}
```

Output:

```
Brand: Tesla
```

Destructor

A **destructor** is a special member function that is automatically called when an object goes out of scope or is explicitly deleted. It is used to clean up resources (e.g., memory) that were allocated by the object.

Characteristics of Destructors:

1. The destructor name is the same as the class name, preceded by a tilde (~).
2. It has no parameters and no return type.
3. It is automatically invoked when the object is destroyed.
4. There can only be one destructor in a class (no overloading).

Example: Destructor

```
#include <iostream>
using namespace std;

class Car {
public:
    Car() {
        cout << "Constructor called" << endl;
    }
    ~Car() {
        cout << "Destructor called" << endl;
    }
};
int main() {
    Car car1; // Constructor is called here
    return 0; // Destructor is called here
}
```

Output:

```
Constructor called
Destructor called
```

Constructor and Destructor in Dynamic Memory Allocation

When dynamically allocating memory, constructors can initialize the memory, and destructors ensure it is properly deallocated.

Example:

```
class Car {
private:
    string* brand;
```

```
public:
    // Constructor
    Car(string b) {
        brand = new string(b); // Allocate memory dynamically
        cout << "Memory allocated for " << *brand << endl;
    }

    // Destructor
    ~Car() {
        delete brand; // Free the allocated memory
        cout << "Memory deallocated" << endl;
    }
};

int main() {
    Car car1("BMW"); // Constructor allocates memory
    return 0;        // Destructor deallocates memory
}
```

Output:

```
Memory allocated for BMW
Memory deallocated
```

Key Differences Between Constructor and Destructor

Feature	Constructor	Destructor
Purpose	Initializes an object.	Cleans up resources used by the object.
Name	Same as the class name.	Same as the class name, preceded by ~.
Parameters	Can have parameters (supports overloading).	Cannot have parameters (no overloading).
Call	Called automatically when an object is created.	Called automatically when an object is destroyed.
Count	Can have multiple constructors (overloading).	Only one destructor is allowed.

Inheritance

Inheritance in **C++** is a core concept of Object-Oriented Programming (OOP) that enables a class to inherit properties and behaviors (methods and attributes) from another class.

Types of Inheritance in C++

1. **Single Inheritance**: A class inherits from one parent class.
2. **Multiple Inheritance**: A class inherits from two or more parent classes.
3. **Multilevel Inheritance**: A class inherits from a derived class (creating a chain).
4. **Hierarchical Inheritance**: Multiple classes inherit from a single base class.
5. **Hybrid Inheritance**: A combination of two or more types of inheritance.

Syntax

Basic Syntax for Inheritance

```
class BaseClass {
   // Members of the base class
};
class DerivedClass : access_specifier BaseClass {
   // Members of the derived class
};
```

Access Specifier

The **access_specifier** determines the visibility of the base class members in the derived class:

- **public**: Public and protected members of the base class remain public/protected in the derived class.
- **protected**: Public and protected members of the base class become protected in the derived class.
- **private**: Public and protected members of the base class become private in the derived class.

Examples of Inheritance

1. Single Inheritance

```
#include <iostream>
using namespace std;
class Animal {
public:
   void eat() {
      cout << "I can eat." << endl;
   }
};
class Dog : public Animal {
public:
   void bark() {
      cout << "I can bark." << endl;
   }
};
```

```
int main() {
    Dog dog;
    dog.eat();  // Inherited from Animal
    dog.bark(); // Defined in Dog
    return 0;
}
```

2. Multiple Inheritance

```
#include <iostream>
using namespace std;
class Flyer {
public:
    void fly() {
        cout << "I can fly." << endl;
    }
};
class Swimmer {
public:
    void swim() {
        cout << "I can swim." << endl;
    }
};
class Duck : public Flyer, public Swimmer {
public:
    void quack() {
        cout << "I can quack." << endl;
    }
};

int main() {
    Duck duck;
    duck.fly();
    duck.swim();
    duck.quack();
    return 0;
}
```

3. Multilevel Inheritance

```
#include <iostream>
using namespace std;
class Animal {
public:
   void breathe() {
      cout << "I can breathe." << endl;
   }
};
class Mammal : public Animal {
public:
   void walk() {
      cout << "I can walk." << endl;
   }
};
class Human : public Mammal {
public:
   void speak() {
      cout << "I can speak." << endl;
   }
};
int main() {
   Human human;
   human.breathe();
   human.walk();
   human.speak();
   return 0;
}
```

4. Hierarchical Inheritance

```
#include <iostream>
using namespace std;
class Shape {
public:
   void description() {
      cout << "I am a shape." << endl;
   }
};
class Circle : public Shape {
public:
```

```
    void area(int radius) {
        cout << "Area of circle: " << 3.14 * radius * radius << endl;
    }
};
class Square : public Shape {
public:
    void area(int side) {
        cout << "Area of square: " << side * side << endl;
    }
};
int main() {
    Circle circle;
    Square square;
    circle.description();
    circle.area(5);
    square.description();
    square.area(4);
    return 0;
}
```

5. Hybrid Inheritance

```
#include <iostream>
using namespace std;
class Vehicle {
public:
    void move() {
        cout << "I can move." << endl;
    }
};
class Engine {
public:
    void power() {
        cout << "I have an engine." << endl;
    }
};
class Car : public Vehicle, public Engine
{
public:
```

```
    void wheels() {
        cout << "I have four wheels." << endl;
    }
};

int main() {
    Car car;
    car.move();
    car.power();
    car.wheels();
    return 0;
}
```

Advantages of Inheritance

- **Code Reusability**: Shared behavior is implemented once and inherited.
- **Extensibility**: Easily extend functionality by adding derived classes.
- **Maintainability**: Centralized updates in the base class affect derived classes.

Data File Handling

Data File Handling refers to the process of creating, reading, writing, and manipulating files on a disk using a programming language. It is essential for storing persistent data, such as configuration files, logs, or user-generated data.

In this context, a **file** is a collection of data stored on a disk. Files can be classified into two main types:

1. **Text Files**: Store data in plain text format (e.g., .txt, .csv).
2. **Binary Files**: Store data in binary format, which is not human-readable (e.g., .exe, .jpg).

Data File Handling in C++

C++ provides tools for file handling through the <fstream> library, which includes three main classes:

1. **ofstream**: Used to write data to files.
2. **ifstream**: Used to read data from files.
3. **fstream**: Used for both reading and writing.

Steps in File Handling

1. Include the <fstream> header file.
2. Declare an object of ofstream, ifstream, or fstream.
3. Open a file using the .open() function or constructor.
4. Perform the desired operations (read/write).
5. Close the file using .close() to free resources.

Examples

1. Writing to a File

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ofstream outFile("example.txt");  // Create and open a file
    if (outFile.is_open()) {
        outFile << "Hello, this is a test file.\n";
        outFile << "File handling in C++ is easy.\n";
        outFile.close();  // Close the file
        cout << "Data written to file successfully!" << endl;
    } else {
        cout << "Error opening file!" << endl;
    }
    return 0;
}
```

2. Reading from a File

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ifstream inFile("example.txt");  // Open the file for reading
    if (inFile.is_open()) {
        string line;
        while (getline(inFile, line)) {  // Read line by line
            cout << line << endl;       // Display the content
        }
        inFile.close();  // Close the file
    } else {
        cout << "Error opening file!" << endl;
    }
    return 0;
}
```

3. Appending Data to a File

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ofstream outFile("example.txt", ios::app);  // Open in append mode
    if (outFile.is_open()) {
        outFile << "This line is appended.\n";
        outFile.close();
        cout << "Data appended successfully!" << endl;
    } else {
        cout << "Error opening file!" << endl;
    }
    return 0;
}
```

4. Reading and Writing (Using fstream)

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    fstream file("example.txt", ios::in | ios::out);  // Open for both reading and writing
    if (file.is_open()) {
        file << "Adding new content with fstream.\n";  // Writing to the file
        file.seekg(0);  // Move the file pointer to the beginning
        string line;
        while (getline(file, line)) {  // Reading from the file
            cout << line << endl;
        }
        file.close();
    } else {
        cout << "Error opening file!" << endl;
    }
    return 0;
}
```

Modes for File Opening

File opening modes are specified as flags when opening a file:

Mode	Description
ios::in	Open for reading.
ios::out	Open for writing.
ios::app	Append to the file.
ios::trunc	Truncate the file (delete content if exists).
ios::binary	Open in binary mode.

Working with Binary Files

Binary files store data in binary format, which is efficient for non-text data (e.g., images, compiled files).

Writing Binary Data

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ofstream outFile("data.bin", ios::binary);  // Open in binary mode
    int number = 42;
    if (outFile.is_open()) {
        outFile.write(reinterpret_cast<char*>(&number), sizeof(number));
        outFile.close();
        cout << "Binary data written successfully!" << endl;
    } else {
        cout << "Error opening file!" << endl;
    }
    return 0;
}
```

Reading Binary Data

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ifstream inFile("data.bin", ios::binary);  // Open in binary mode
    int number;
```

```
    if (inFile.is_open()) {
        inFile.read(reinterpret_cast<char*>(&number), sizeof(number));
        inFile.close();
        cout << "Binary data read: " << number << endl;
    } else {
        cout << "Error opening file!" << endl;
    }
    return 0;
}
```

Advantages of File Handling

- Data persistence between program executions.
- Efficient data storage for large datasets.
- Ability to process structured data like CSV files.

Summary:

C++ is a high-level, general-purpose programming language that is widely used for system and application software, game development, and real-time systems. It was developed by Bjarne Stroustrup in 1979 as an extension of the C programming language, adding object-oriented features.

It is a versatile language that balances high-level abstractions with low-level system control. It is essential for developers who need to optimize performance and work on complex systems while still taking advantage of object-oriented programming techniques.

Check your Understanding:

Question 1:

Which feature in OOP allows reusing code?

a) Polymorphism
b) Inheritance
c) Encapsulation
d) Data hiding

Question 2:

Which of the following concept of oops allows compiler to insert arguments in a function call if it is not specified?

a) Call by value
b) Call by reference
c) Default arguments
d) Call by pointer

Question 3:

Which of the following concepts of OOPS means exposing only necessary information to client?

a) Encapsulation
b) Abstraction
c) Data hiding
d) Data binding

Question 4:

Which of the following statement is correct?

a) A constructor is called at the time of declaration of an object.
b) A constructor is called at the time of use of an object.
c) A constructor is called at the time of declaration of a class.
d) A constructor is called at the time of use of a class.

Question 5:

Which of the following will not valid expressions in C++?

a) a=2+(b=5);
b) a=b=c=5;
c) a=11%3
d) b+5=2

Question 6:

The escape sequence '\b" is a

a) back space
b) next line
c) tab
d) none of the above

Question 7:

Which of the following is correct about function overloading?

a) The types of arguments are different
b) The order of argument is different
c) The number of arguments is same
d) Both A and B.

Question 8:

Which of the following concepts means wrapping up of data and functions together?

a) Abstraction
b) Inheritance
c) Polymorphism
d) Encapsulation

Question 9:

Which of the following header file includes definition of cin and cout?

a) istream.h
b) ostream.h
c) iomanip.h
d) iostream.h

Question 10:

Which of the following is an invalid visibility label while inheriting a class?

a) public
b) private
c) protected
d) friend

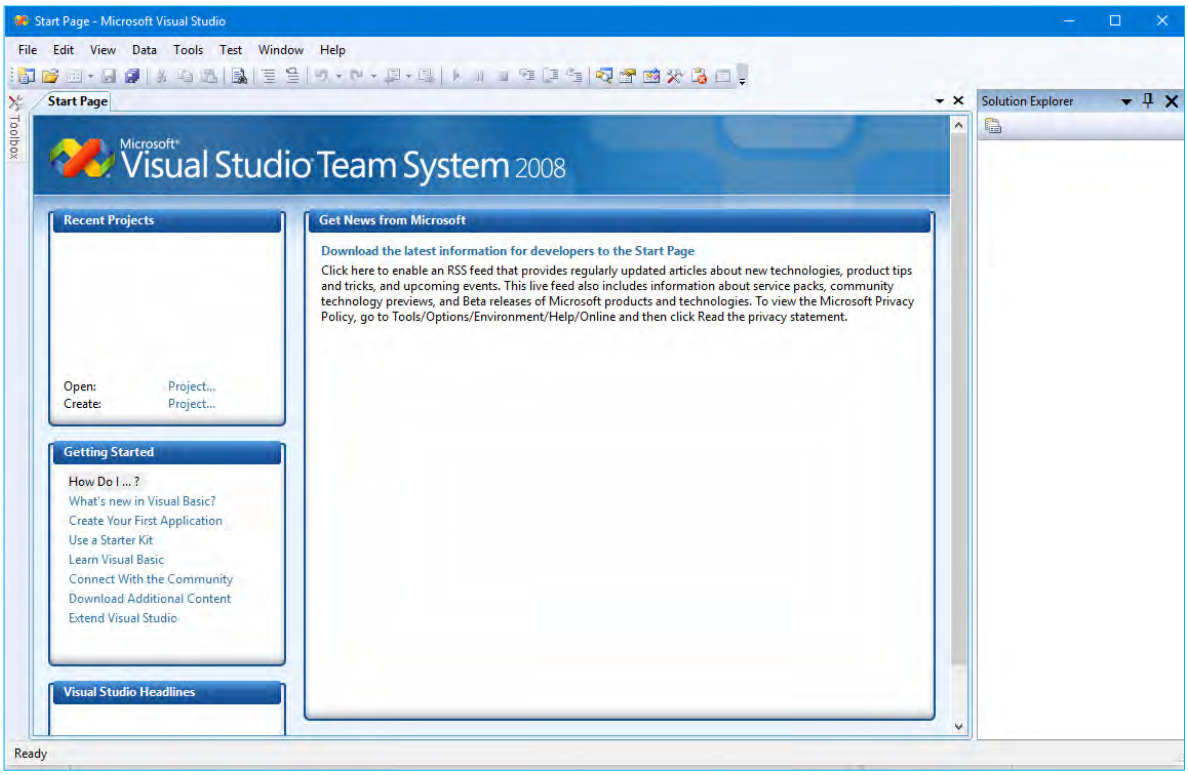
Chapter 2: VB.NET 2008

Introduction

Visual Basic 2008 is the latest version of Visual Basic launched by Microsoft. It is almost similar to Visual Basic 2005 but it has included many features. Visual Basic 2008 is a full-fledged Object-Oriented Programming (OOP) Language, so it has caught up with other OOP languages such as C++, Java, C# and others. However, you don't have to know OOP to learn VB2008. In fact, if you are familiar with Visual Basic 6, you can learn VB2008 effortlessly because the syntax and interface are similar.

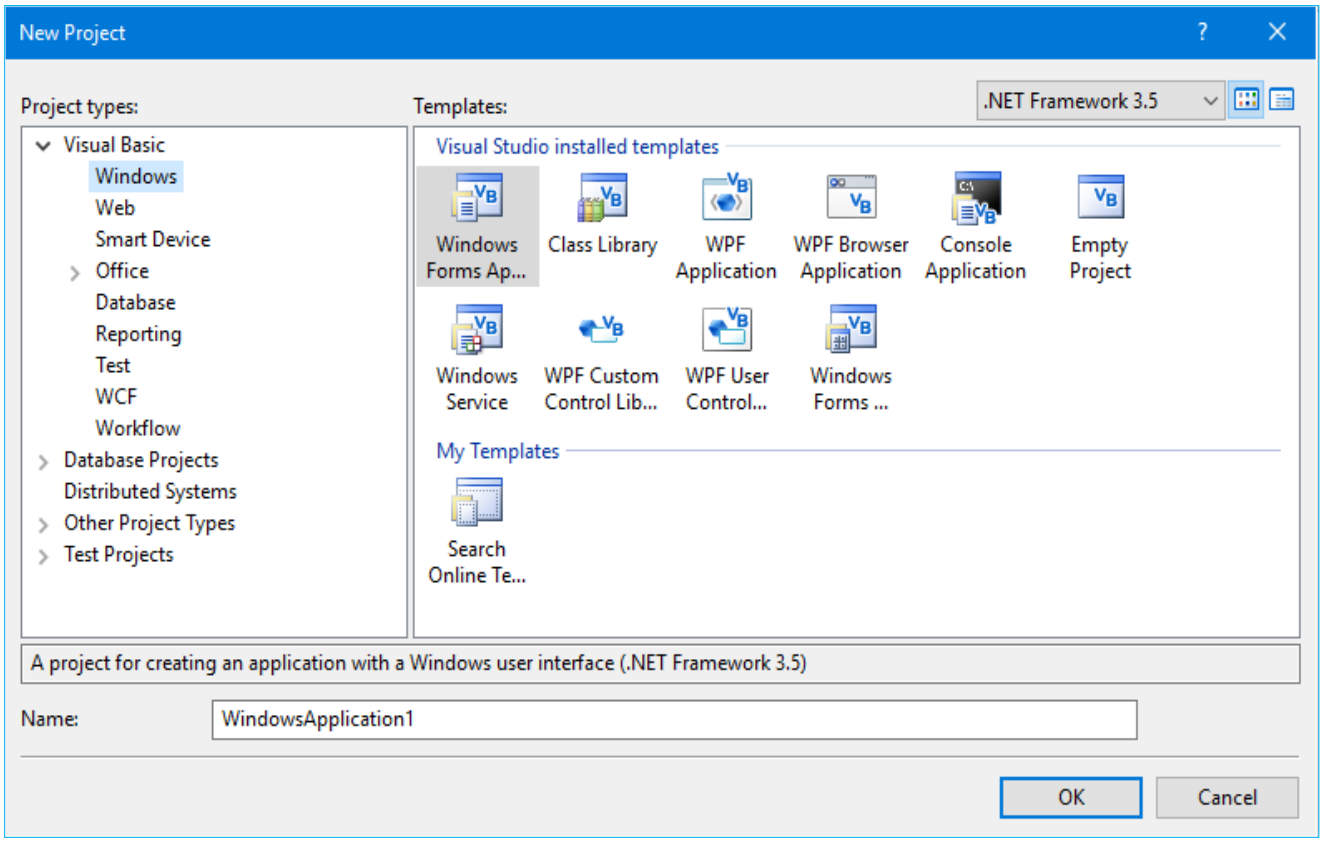
The Integrated Development Environment when you launch VB2008 Ex-press is shown in the diagram below. The IDE consists of a few panes, namely:

A) The Recent Projects Pane- it shows the list of projects that have been created by you recently.
B) The Getting Started Pane- It provides some helpful tips to quickly develop your applications.
C) The VB Express Headlines pane- It provides latest online news about Visual Basic 2008 Express. It will announce new releases and updates



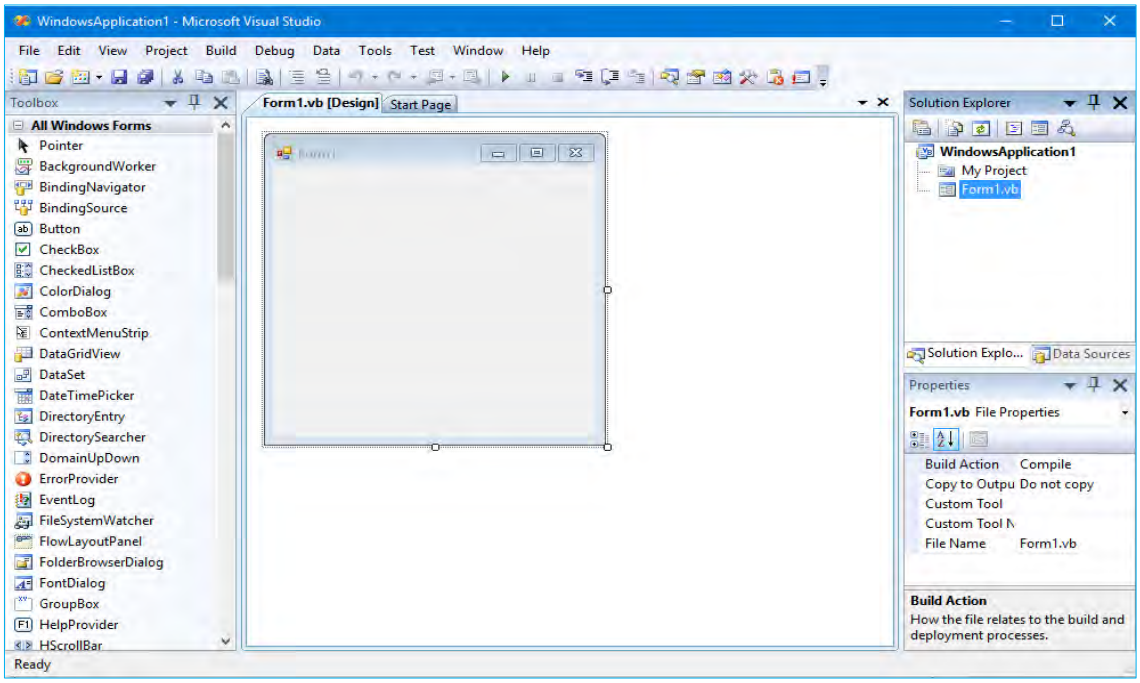
Creating your first application

To start creating your first application, you need to click on file and select new project. The following VB2008 New Project dialog box will appear.



The dialog box offers you five types of projects that you can create. As we are going to learn to create windows Applications, we will select Windows Forms Application.

At the bottom of this dialog box, you can change the default project name WindowsApplication1 to some other name you like, for example, MyFirstProgram. After you have renamed the project, click OK to continue. The following IDE Windows will appear, it is almost similar to Visual Basic 6. It consists of an empty form, the common controls toolbox, the solution explorer and the properties.



Creating the first program

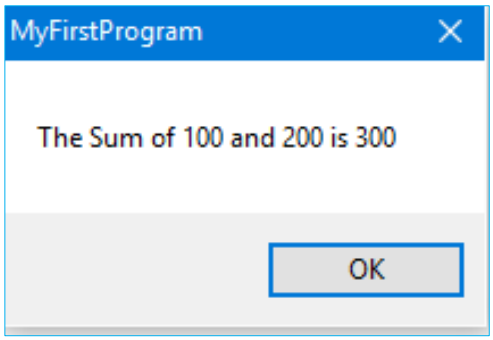
Create your first program. First of all, drag one common button into the form and change its default name to calculate.

First of all, drag one common button into the form and change its default name to calculate,

Next, click on the calculate button and key in the following code at the source code window as shown below.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim num1, num2, sum As Single
num1 = 100
num2 = 200
sum = num1 + num2
MsgBox(“ The Sum of “ & num1 & “ and “ & num2 & “ is “ & sum,,”MyFirstProgram”)
End Sub
```

Now run your first application! And you can see the follow message box showing the sum of two numbers.



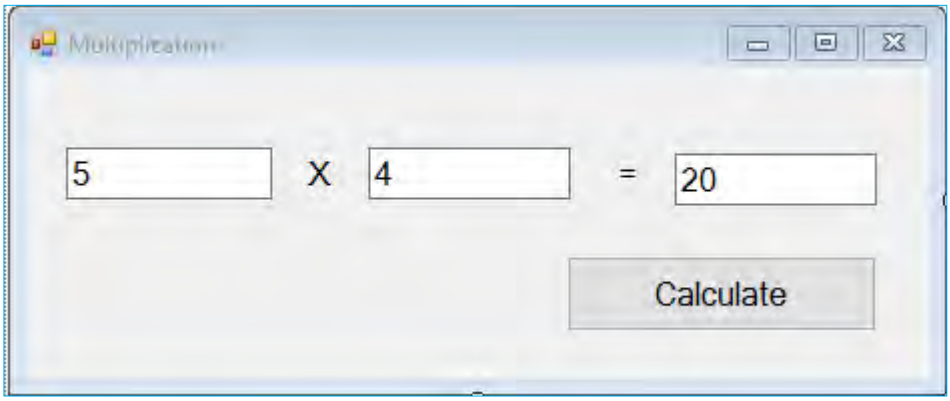
Working with Controls

Introduction of Controls

Controls in VB2008 are useful tools that can be placed in the form to per-form various tasks. They are used to create many kinds of Windows applications. The diagram on the right is the Toolbox that contains the controls of VB2008. They are categorized into Common Controls, Containers, Menus, Toolbars, Data, Components, Printings and Dialogs. At the moment, we will focus on the common controls. Some of the most used common controls are Button, Label, ComboBox, ListBox, PictureBox, TextBox etc. To insert a control into your form, you just need to drag the control and drop it into the form. You can reposition and resize it as you like. Let's examine a few programs that made use of Button, Label, TextBox, ListBox and PictureBox.

Using Text Box-A multiplication program

In this program, you insert two textboxes, three labels and one button. The two textboxes are for the users to enter two numbers, one label is to display the multiplication operator and the other label is to display the equal sign. The last label is to display the answer



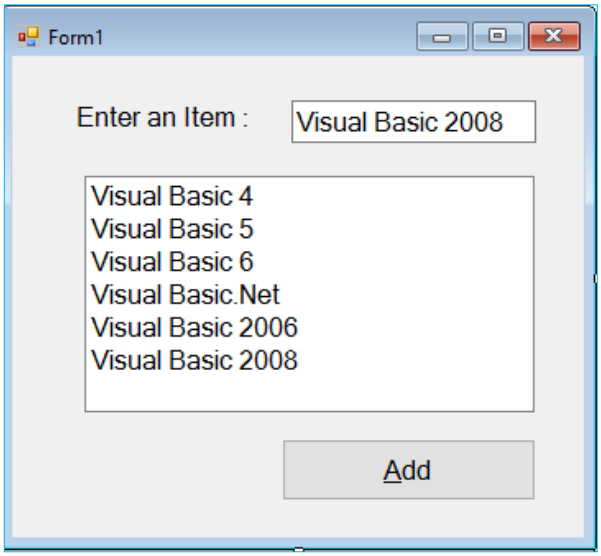
The Code:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

Dim num1, num2, product As Single

num1 = TextBox1.Text

num2 = TextBox2.Text

product = num1 * num2

Label3.Text = product

End Sub
```

Using the ListBox-A program to add items to a list box

This program will add one item at a time as the user enter an item into the TextBox and click the Add button

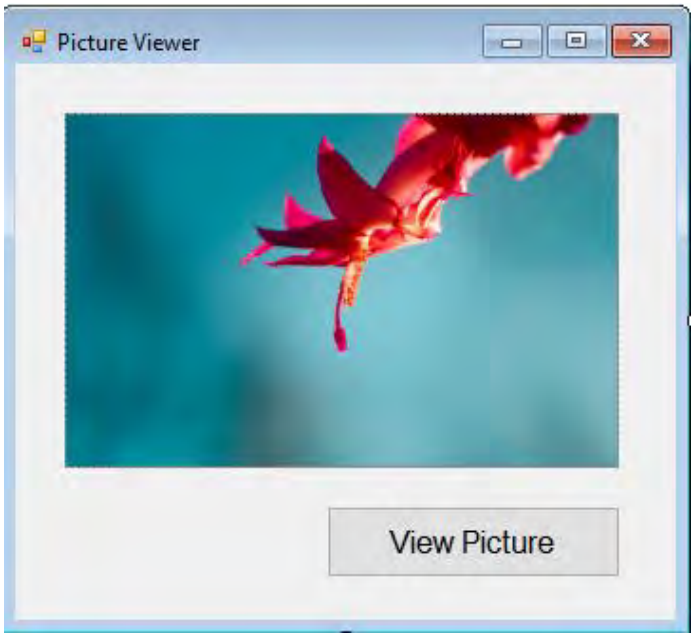
```
Public Class Frm1
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim item As String
item = TextBox1.Text
'To add items to a listbox
ListBox1.Items.Add(item)
End Sub
End Class
```



Using the PictureBox

In this program, we insert a PictureBox and a Button into the form. Make sure to set the SizeMode property of the PictureBox to StretchImage so that the whole picture can be viewed in the picture box. Key in the code as shown below and you can load an image from a certain image file into the PictureBox

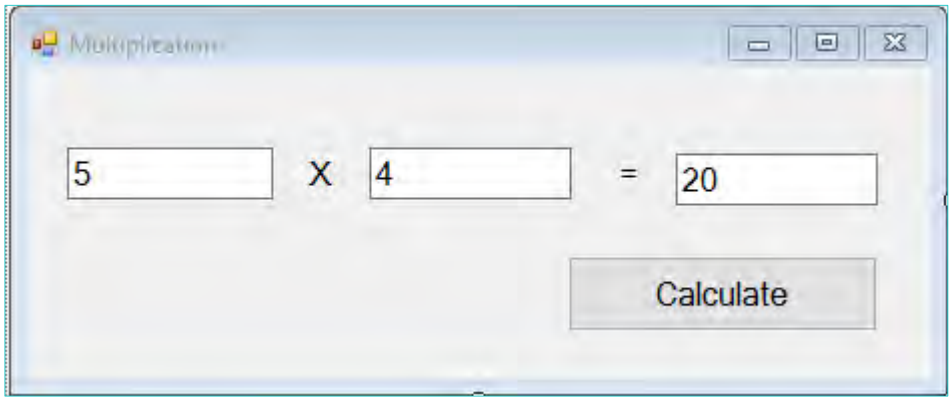
```
Public Class Form1
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
'To load an image into the PictureBox from an image file
PictureBox1.Image = Image. From File ("c:\ Users \ Public \ Pictures \ Sample Pictures \ Frangipani Flowers.jpg")
End Sub
End Class
```



Working with Control Properties

The Control Properties

Before writing an event procedure for the control to response to a user's input, you have to set certain properties for the control to determine its appearance and how it will work with the event procedure. You can set the properties of the controls in the properties window at design time or at runtime.



The title of the form is defined by the Text property and its default name is Form 1. To change the form's title to any name that you like, simple click in the box on the right of the Text property and type in the new name, in this example, the title is Multiplication. Notice that this title will appear on top of the windows. In the properties window, the item appears at the top part is the object currently selected. At the bottom part, the items listed in the left column represent the names of various properties associated with the selected object while the items listed in the right column represent the states of the properties. Properties can be set by highlighting the items in the right column then change them by typing or selecting the options available. You may also alter other properties of the form such as font, location, size, foreground color, background color, MaximizeBox, MinimizeBox etc.

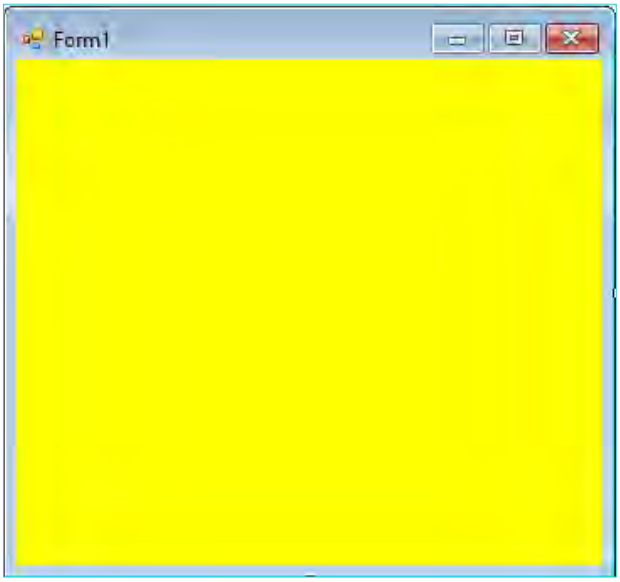
You can also change the properties of the object at runtime to give special effects such as change of color, shape, animation effect and so on. For example the following code will change the form color to yellow every time the form is loaded. VB2008 uses RGB (Red, Green, Blue) to determine the colors. The RGB code for yellow is 255, 255, 0. **Me** in the code refer to the current form and Backcolor is the property of the form's background color. The formula to assign the RGB color to the form is **Color.FormArbg(RGB codes)**.

```
Public Class Form1
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
Me.BackColor = Color.FromArgb(255, 255, 0)
End Sub
End Class
```

You may also use the follow procedure to assign the color at run time.

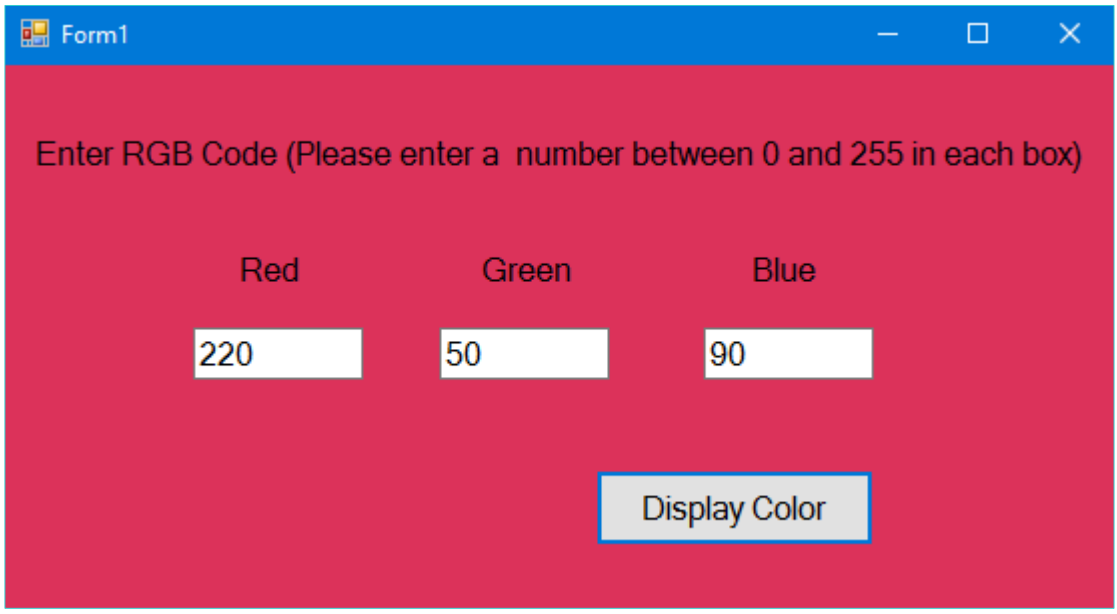
```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
Me.BackColor = Color.Yellow
End Sub
End Class
```

Both procedures above will load the form with a yellow background as follows:



Color	RGB Color	Color	RGB Color	Color	RGB Color
	0, 0, 255		0, 0, 255		139, 0, 0
	255, 0, 255		255, 0, 0		255, 255, 255
	255, 255, 0		0, 255, 0		0, 0, 0

The following is another program that allows the user to enter the RGB codes into three different textboxes and when he/she clicks the display color but-ton, the background color of the form will change according to the RGB codes. So, this program allows users to change the color properties of the form at run time



The code

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim rgb1, rgb2, rgb3 As Integer
rgb1 = TextBox1.Text
rgb2 = TextBox2.Text
rgb3 = TextBox3.Text
Me.BackColor = Color.FromArgb(rgb1, rgb2, rgb3)
End Sub
```

Object Oriented Programming

OOPS Concepts

First of all, VB2008 is very much similar to VB6 in terms of Interface and program structure, their underlying concepts are quite different. The main different is that VB2008 is a full Object-Oriented Programming Language while VB6 may have OOP capabilities, it is not fully object oriented. In order to qualify as a fully object-oriented programming language, it must have three core technologies namely encapsulation, inheritance and polymorphism.

These three terms are explained below

Encapsulation refers to the creation of self-contained modules that bind processing functions to the data. These user-defined data types are called classes. Each class contains data as well as a set of methods which manipulate the data. The data components of a class are called instance variables and one instance of a class is an object. For example, in

a library system, a class could be member, and John and Sharon could be two instances (two objects) of the library class.

Inheritance Classes are created according to hierarchies, and inheritance allows the structure and methods in one class to be passed down the hierarchy. That means less programming is required when adding functions to complex systems. If a step is added at the bottom of a hierarchy, then only the processing and data associated with that unique step needs to be added. Everything else about that step is inherited. The ability to reuse existing objects is considered a major advantage of object technology.

Polymorphism Object-oriented programming allows procedures about objects to be created whose exact type is not known until runtime. For example, a screen cursor may change its shape from an arrow to a line depending on the program mode. The routine to move the cursor on screen in response to mouse movement would be written for "cursor," and polymorphism allows that cursor to take on whatever shape is required at runtime. It also allows new shapes to be easily integrated.

VB2008 allows users to write programs that break down into modules. These modules will represent the real-world objects and are knows as classes or types. An object can be created out of a class and it is known as an instance of the class. A class can also comprise subclass. For example, apple tree is a *subclass* of the *plant* class and the apple in your backyard is an instance of the apple tree class. Another example is student class is a subclass of the human class while your son John is an instance of the student class.

Basic Class Creation

A class consists of data members as well as methods. In VB2008, the program structure to define a Human class can be written as follows:

```
Public Class Human
'Data Members
Private Name As String
Private Birthdate As String
Private Gender As String
Private Age As Integer
'Methods
Overridable Sub ShowInfo( )
MessageBox.Show(Name)
MessageBox.Show(Birthdate)
MessageBox.Show(Gender)
MessageBox.Show(Age)
End Sub
End Class
```

After you have created the human class, you can create a subclass that inherits the attributes or data from the human class. For example, you can create a student's class that is a subclass of the human class. Under the student class,

you don't have to define any data fields that are already defined under the human class, you only have to define the data fields that are different from an instance of the human class. For example, you may want to include StudentID and Address in the student class.

The program code for the Student Class is as follows:

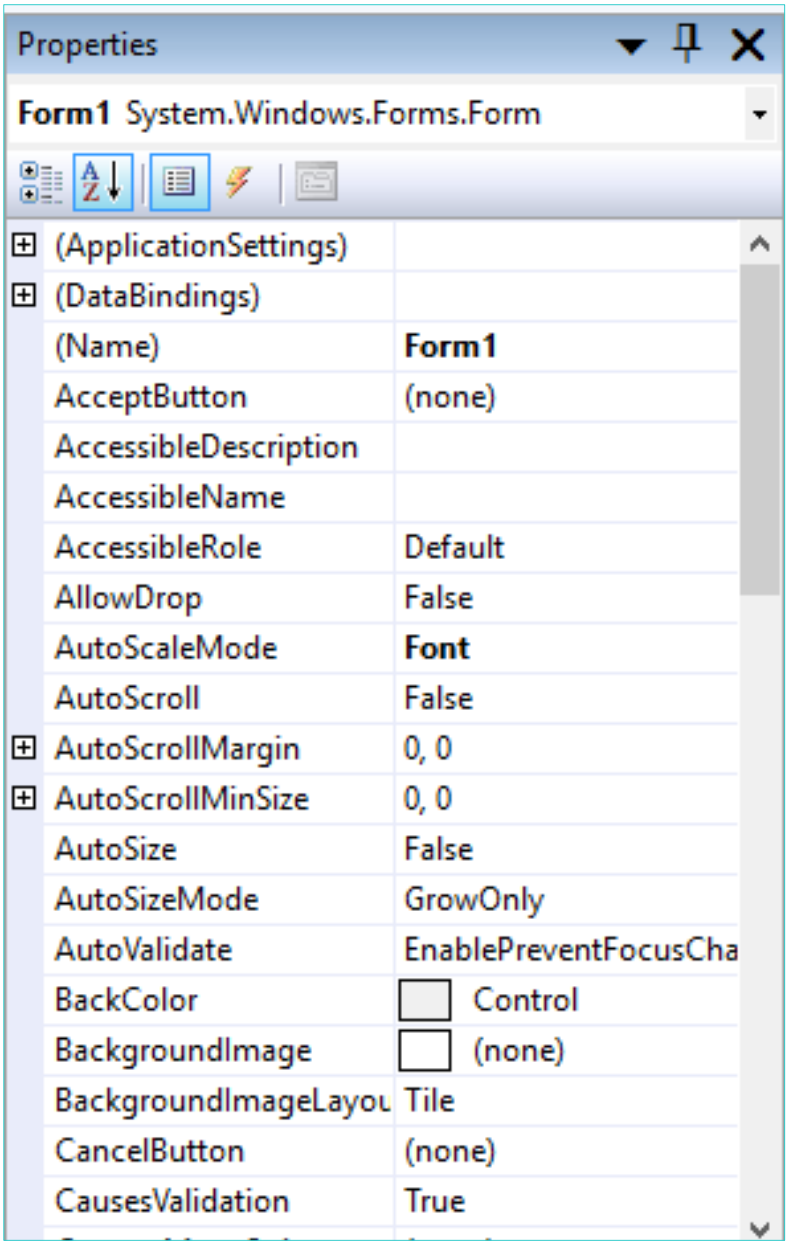
```
Public Class Students
Inherits Human
Public StudentID as String
Public Address As String
Overrides  Sub ShowInfo( )
MessageBox.Show(Name)
MessageBox.Show(StudentID)
MessageBox.Show(Birthdate)
MessageBox.Show(Gender)
MessageBox.Show(Age)
MessageBox.Show(Address)
End Sub
```

Writing Code

You have understood the meanings of class, object, encapsulation inheritance as well as polymorphism. You have also learned to write some simple programs without much understanding some underlying foundations and theories. In this chapter, you will learn some basic theories about VB2008 programming but we will focus more on learning by doing, i.e. learning by writing programs.

The event Procedure

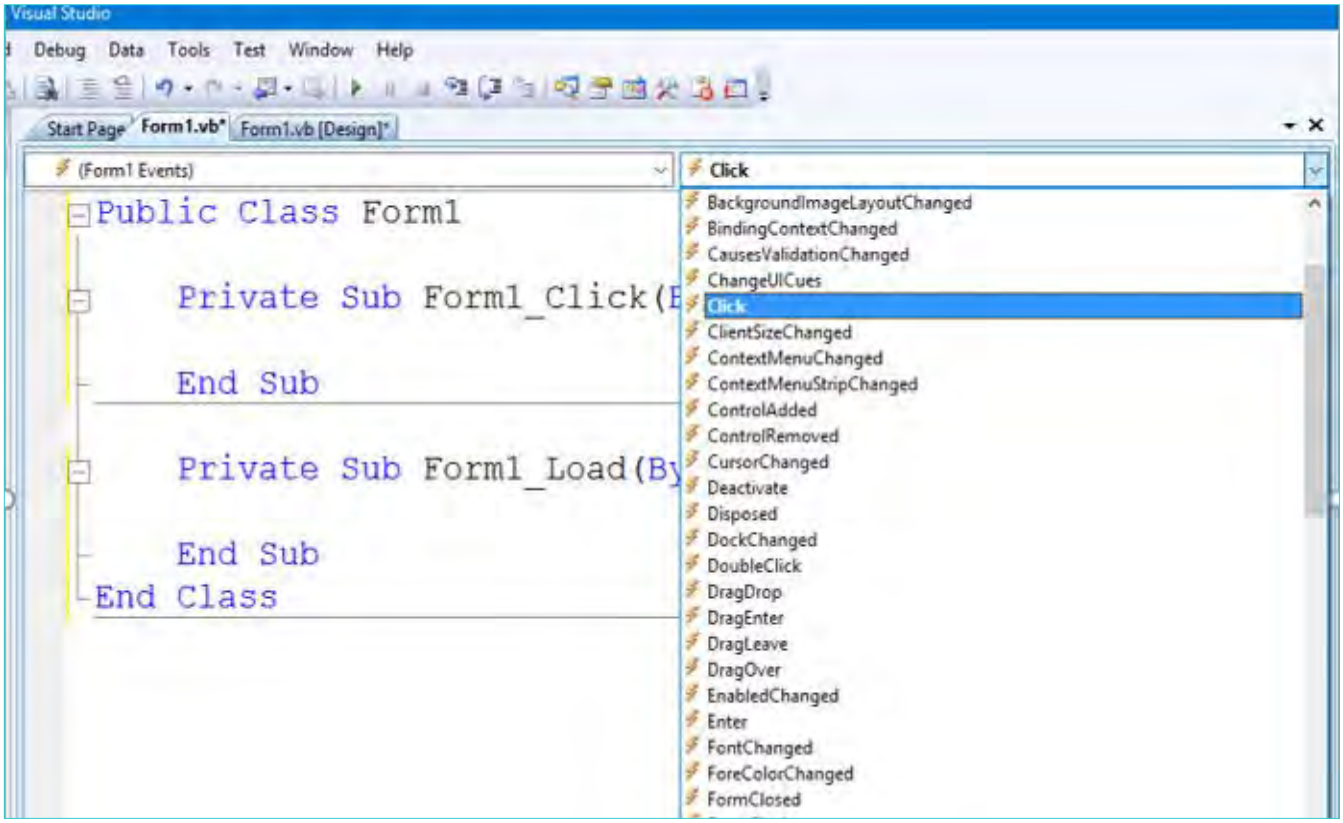
VB2008 is an object oriented and event driven programming language. In fact, all windows applications are event driven. Event driven means the user will decide what to do with the program, whether he/she wants to click the command button, or he/she wants to enter text in a text box, or he/she might want to close the application and etc. An event is related to an object, it is an incident that happens to the object due to the action of the user, such as a click or pressing a key on the keyboard. A class has events as it creates instant of a class or an object. When we start a windows application in VB2008 in previous chapters, we will see a default form with the Form1 appears in the IDE, it is actually the Form1 Class that inherits from the Form class System.Windows.Forms.Form, as shown in the Form1 properties windows.



When we click on any part of the form, we will see the code window as shown below. This the structure of an event procedure. In this case, the event procedure is to load Form1 and it starts with Private Sub and end with End Sub. This procedure includes the Form1 class and the event Load, and they are bind together with an underscore, i.e. Form_Load. It does nothing other than loading an empty form.

```
Public Class Form1
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)HandlesMyBase.Load


End Sub
End Class
```

The are other events associated with the Form1 class, such as click, DoubleClick, DragDrop, Enter as so on, as shown in the diagram below (It appears when you click on the upper right pane of the code window)

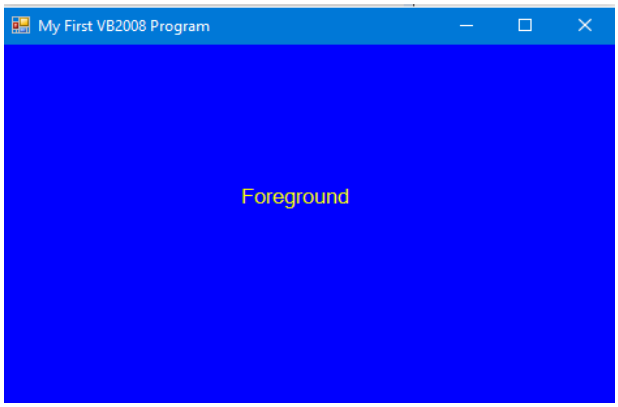


Writing the code

Now you are ready to write the code for the event procedure so that it will do something more than loading a blank form. The code must be entered between Private Sub.....End Sub. Let's enter the following code:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
Me.Text="My First VB2008 Program"
Me.ForeColor = Color.Yellow
Me.BackColor = Color.Blue
End Sub
```

The output is shown in the windows below:

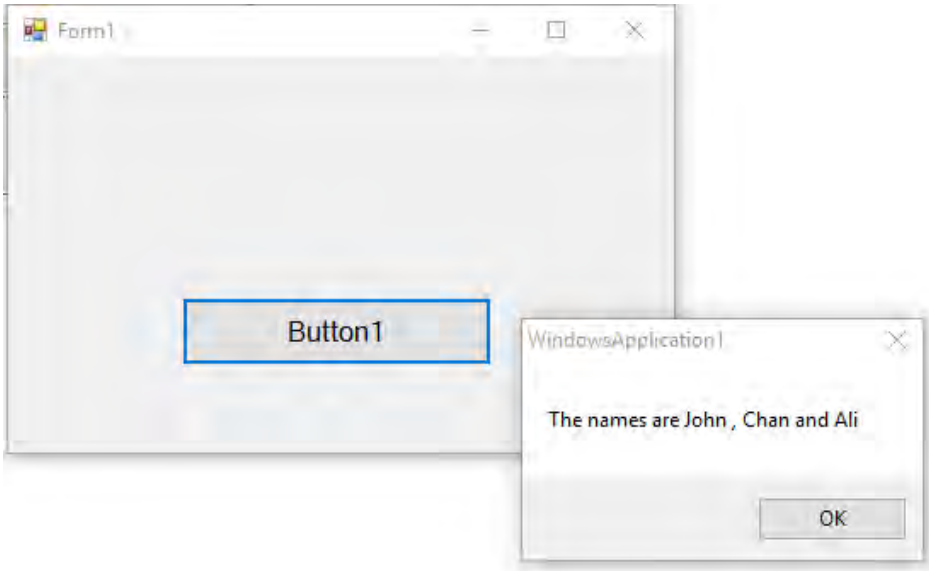


The first line of the code will change the title of the form to My First VB2008 Program, the second line will change the foreground object to yellow(in this case, it is a label that you insert into the form and change its name to Foreground) and the last line changes the background to blue color. The equal in the code actually is used to assign something to the object, like assigning yellow color to the foreground of the Form1 object (or an instance of Form1).

Here is another example.

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim name1, name2, name3 As String
name1 = “John”
name2 = “Chan”
name3 = “Ali”
MsgBox(“The names are “ & name1 & “ , “ & name2 & “ and “ & name3)
End Sub
```

In this example, you insert one command button into the form and rename its caption as Show Hidden Names. The keyword Dim is to declare variables name1, name2 and name3 as string, which means they can only handle text. The function MsgBox is to display the names in a message box that are joined together by the “&” signs.



Managing VB 2008 Data

There are many types of data that we come across in our daily life. For example, we need to handle data such as names, addresses, money, date, stock quotes, statistics and etc. everyday. Similarly in Visual Basic 2008, we have to deal with all sorts of data, some can be mathematically calculated while some are in the form of text or other forms. VB2008 divides data into different types so that it is easier to manage when we need to write the code involving those data.

Visual Basic 2008 Data Types

Visual Basic 2008 classifies the information mentioned above into two major data types, they are the numeric data types and the non-numeric data types.

Numeric Data Types

Numeric data types are types of data that consist of numbers, which can be computed mathematically with various standard operators such as add, minus, multiply, divide and so on. Examples of numeric data types are your examination marks, your height, your weight, the number of students in a class, share values, price of goods, monthly bills, fees and etc. In Visual Basic 2008, numeric data are divided into 7 types, depending on the range of values they can store.

Calculations that only involve round figures or data that don't need precision can use Integer or Long integer in the computation. Programs that require high precision calculation need to use Single and Double precision data types, they are also called floating point numbers. For currency calculation, you can use the currency data types. Lastly, if even more precision is requires to perform calculations that involve a many decimal points, we can use the decimal data types. These data types summarized in Table 1

Table 1: Numeric Data Types

Data Type	Storage	Range
Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	- 2,147,483,648 to 2,147,483,648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values.
Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places).

Non-numeric Data Types

Nonnumeric data types are data that cannot be manipulated mathematically using standard arithmetic operators. The non-numeric data comprises text or string data types, the Date data types, the Boolean data types that store only two values (true or false), Object data type and Variant data type .They are summarized in Table 2

Table 2: Nonnumeric Data Types

Data Type	Storage	Range
String(fixed length)	Length of string	1 to 65,400 characters
String(variable length)	Length + 10 bytes	0 to 2 billion characters
Date	8 bytes	January 1, 100 to December 31, 9999

Boolean	2 bytes	True or False
Object	4 bytes	Any embedded object
Variant(numeric)	16 bytes	Any value as large as Double
Variant(text)	Length+22 bytes	Same as variable-length string

Suffixes for Literals

Literals are values that you assign to a data. In some cases, we need to add a suffix behind a literal so that VB2008 can handle the calculation more accurately. For example, we can use num = 1.3089# for a Double type data. Some of the suffixes are displayed in Table 3

Table 3:

Suffix	Data Type
&	Long
!	Single
#	Double
@	Currency

In addition, we need to enclose string literals within two quotations and date and time literals within two # sign. Strings can contain any characters, including numbers. The following are few examples:

memberName=”Turban, John.”

TelNumber=”1800-900-888-777"

LastDay=#31-Dec-00#

ExpTime=#12:00 am#

Managing Variables

Variables are like mail boxes in the post office. The contents of the variables changes every now and then, just like the mail boxes. In term of VB2008, variables are areas allocated by the computer memory to hold data. Like the mail boxes, each variable must be given a name. To name a variable in Visual Basic 2008, you have to follow a set of rules.

Variable Names

The following are the rules when naming the variables in Visual Basic 2008 ·

- It must be less than 255 characters
- No spacing is allowed
- It must not begin with a number
- Period is not permitted

Examples of valid and invalid variable names are displayed in Table 4

Table 4:

Valid Name	Invalid Name
My_Car	My.Car
This_year	This year
Newbot1	1NewBoy

Declaring Variables

In Visual Basic 2008, one needs to declare the variables before using them by assigning names and data types. If you fail to do so, the program will show an error. They are normally declared in the general section of the codes' windows using the **Dim** statement.

The format is as follows:

Dim *Variable Name* As *Data Type*

Example:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
Dim password As String
Dim yourName As String
Dim firstnum As Integer
Dim secondnum As Integer
Dim total As Integer
Dim doDate As Date
End Sub
```

You may also combine them in one line, separating each variable with a comma, as follows:

```
Dim password As String, yourName As String, firstnum As Integer,.....
```

For string declaration, there are two possible formats, one for the variable-length string and another for the fixed-length string. For the variable-length string, just use the same format as example 6.1 above. However, for the fixed-length string, you have to use the format as shown below:

*Dim VariableName as String * n*, where n defines the number of characters the string can hold.

Example: Dim yourName as String * 10 'yourName can holds no more than 10 Characters.

Assigning Values to Variables

After declaring various variables using the Dim statements, we can assign values to those variables. The general format of an assignment is Variable=Expression The variable can be a declared variable or a control property value. The expression could be a mathematical expression, a number, a string, a Boolean value (true or false) and etc.

The following are some examples:

firstNumber = 100

secondNumber = firstNumber-99

username = ”John Lyan”

userpass.Text = password

Label1.Visible = True

Command1.Visible = false

Label4.Caption = textbox1.Text

ThirdNumber = Val (usernum1.Text)

total = firstNumber + secondNumber +ThirdNumber

Constants

Constants are different from variables in the sense that their values do not change during the running of the program.

The format to declare a constant is –

Const *Constant Name* As *Data Type* = *Value*

Example:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
Const Pi As Single = 3.142
Const Temp As Single = 37
Const Score As Single = 100
End Sub
```

Mathematical Operations

Computer can perform mathematical calculations much faster than human beings. However, computer itself will not be able to perform any mathematical calculations without receiving instructions from the user. In VB2008, we can write code to instruct the computer to perform mathematical calculations such as addition, subtraction, multiplication, division and other kinds of arithmetic operations. In order for VB2008 to carry out arithmetic calculations, we need to write code that involve the use of various arithmetic operators. The VB2008 arithmetic operators are very similar to the normal arithmetic operators, only with slight variations. The plus and minus operators are the same while the multiplication operator use the * symbol and the division operator use the / symbol. The list of VB2008 arithmetic operators are shown in table below:

Operator	Mathematical function	Example
+	Addition	1 + 2 = 3
—	Subtraction	4 – 1 = 3
^	Exponential	2 ^ 4 = 16
*	Multiplication	4 * 3 = 12
/	Division	12 / 4 = 3
Mod	Modulus (return the remainder from an integer division	15 Mod 4 = 3
\	Integer Division (discards the decimal places)	22 \ 7 = 3

Example

In this program, you need to insert two Textboxes, four labels and one but-ton. Click the button and key in the code as shown below. Note how the various arithmetic operators are being used. When you run the program, it will perform the four basic arithmetic operations and display the results on the four labels.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim num1, num2, difference, product, quotient As Single
num1 = TextBox1.Text
num2 = TextBox2.Text
sum = num1+num2
difference = num1-num2
product = num1 * num2
quotient = num1/num2
Label1.Text = sum
Label2.Text = difference
Label3.Text = product
Label4.Text = quotient
End Sub
```

Example

The program can use Pythagoras Theorem to calculate the length of hypotenuse c given the length of the adjacent side a and the opposite side b. In case you have forgotten the formula for the Pythagoras Theorem, it is written as

$$c^2 = a^2 + b^2$$

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim a, b, c  As Single
a = TextBox1.Text
b = TextBox2.Text
c=(a^2+b^2)^(1/2)
Label3.Text=c
End Sub
```

String Manipulation

String manipulation is an important part of programming because it helps to process data that come in the form of non-numeric types such as name, address, city, book title and etc.

String Manipulation Using + and & signs.

Strings can be manipulated using the & sign and the + sign, both perform the string concatenation which means combining two or more smaller strings into larger strings. For example, we can join “Visual” and “Basic” into “Visual Basic” using “Visual”&”Basic” or “Visual “+”Basic”, as shown in the example below:

Example

```
Public Class Form1
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim text1, text2, text3 As String
text1 = “Visual”
text2 = “Basic”
text3 = text1 + text2
Label1.Text = text3
End Sub
End Class
```

The line text3=text1+ text2 can be replaced by text3=text1 & text2 and produced the same output. However, if one of the variables is declared as numeric data type, you cannot use the + sign, you can only use the & sign.

Example:

```
Dim text1, text3 as string
Dim Text2 As Integer
text1 = “Visual”
text2=22
text3=text1+text2
Label1.Text = text3
```

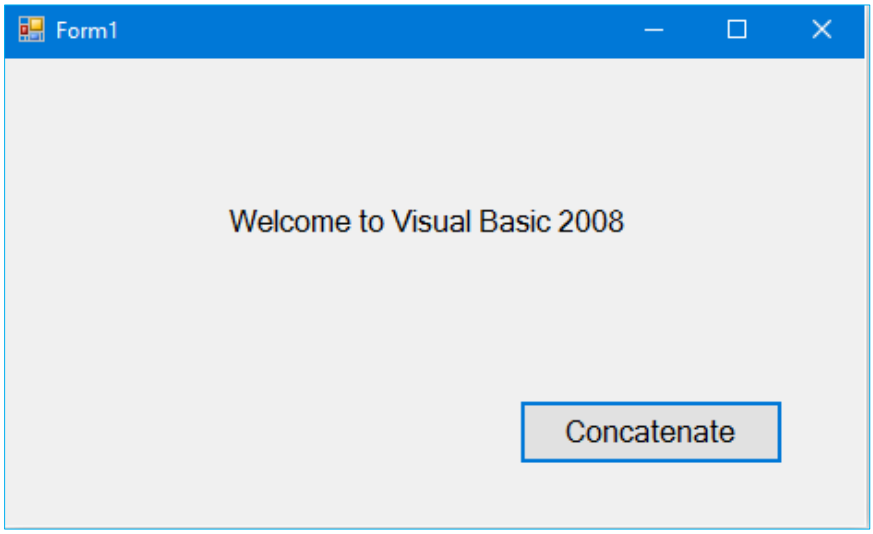
The code will produce an error because of data mismatch. However, using & instead of + will be all right.

```
Dim text1, text3 as string
Dim Text2 As Integer
text1 = “Visual”
text2=22
text3=text1 & text2
Label1.Text = text3
```

You can combine more than two strings to form a larger strings, like the following example:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim text1, text2, text3, text4, text5, text6 As String
text1 = “Welcome ”
text2 = “to ”
text3 = “Visual ”
text4 = “Basic ”
text5 = “2008”
text6 = text1 + text2 + text3 + text4 + text5
Label1.Text = text6
End Sub
```

Running the above program will produce the following screen shot.



String Manipulation Using VB2008 Built-in Functions

A function is similar to a normal procedure but the main purpose of the function is to accept a certain input and return a value which is passed on to the main program to finish the execution. There are numerous string manipulation functions built into VB2008 .

The Len Function

The length function returns an integer value which is the length of a phrase or a sentence, including the empty spaces. The format is:

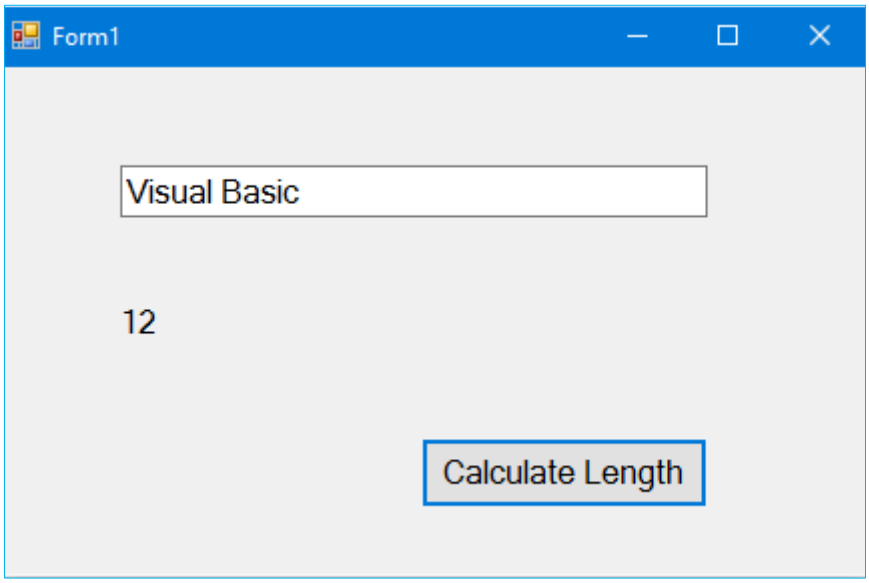
Len ("Phrase")

For example, Len (Visual Basic) = 12 and Len (welcome to VB tutorial) = 22

Example:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Label1.Text = Len(TextBox1.Text)
End Sub
```

The output:



The Right Function

The Right function extracts the right portion of a phrase. The format for Visual Basic 6 is

Right ("Phrase", n)

Where n is the starting position from the right of the phase where the portion of the phrase is going to be extracted. For example,

Right("Visual Basic", 4) = asic

However, this format is not applicable in VB2008. In VB2008, we need use the following format

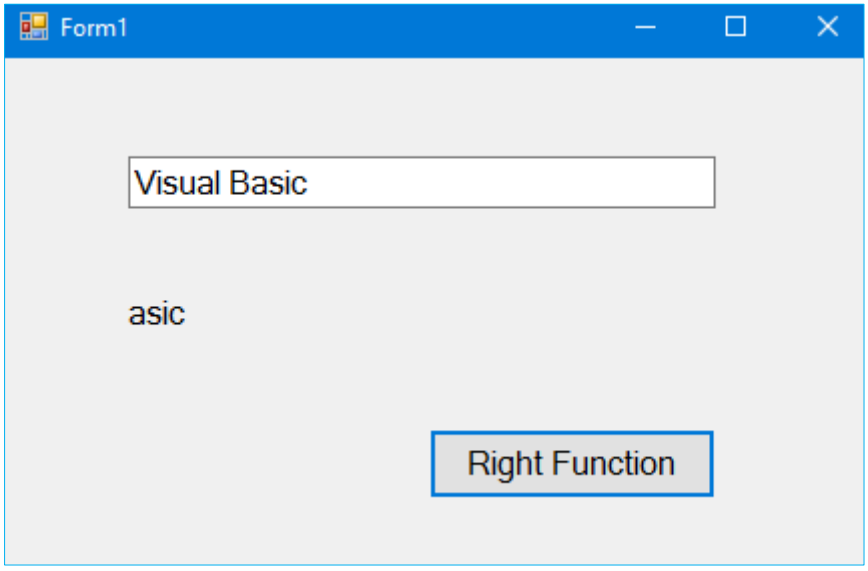
Microsoft.VisualBasic.Right("Phrase",n)

Example:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim text1 As String
text1 = TextBox1.Text
Label1.Text = Microsoft.VisualBasic.Right(text1, 4)
End Sub
```

The above program will return four right most characters of the phrase entered into the textbox.

The Output:



The reason of using the full reference is because many objects have the Right properties so using Right on its own will make it ambiguous to VB2008.

The Left Function

The Left function extract the left portion of a phrase. The format is

Microsoft.VisualBasic.Left(“Phrase”,n)

Where n is the starting position from the left of the phase where the portion of the phrase is going to be extracted.

For example,

Microsoft.VisualBasic.Left (“Visual Basic”, 4) = Visu

Controlling Program Flow

In the previous lessons, we have learned how to program code that accept input from the users and display the output without controlling the program flow. In this chapter, you will learn how to program VB2008 code that can make decision when it process input from the users, and control the pro-gram flow in the process. Decision making process is an important part of programming because it will help solve practical problems intelligently so that it can provide useful output or feedback to the user. For example, we can write a VB2008 program that can ask the computer to perform

certain task until a certain condition is met, or a program that will reject non-numeric data. In order to control the program flow and to make decisions, we need to use the conditional operators and the logical operators together with the If control structure.

Conditional Operators

The conditional operators are powerful tool that resemble mathematical operators that let the VB2008 program compare data values and then decide what actions to take, whether to execute a program or terminate the program and etc. They are also known as numerical comparison operators. Normally they are used to compare two values to see whether they are equal or one value is greater or less than the other value. The comparison will return true or false result. These operators are shown in Table.

Operator	Meaning
And	Both sides must be true
Or	One side or other must be true
Xor	One side or other must be true but not both
Not	Negates truth

Logical Operators

Sometimes we might need to make more than one comparison before a decision can be made and an action taken. In this case, using numerical comparison operators alone is not sufficient, we need to use additional operators, and they are the logical operators. These logical operators are shown in Table.

Operator Meaning	Logical Operators
=	Equal to
>	Greater than
<	Less Than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not Equal to

Normally the above operators are used to compare numerical data. However, you can also compare strings with the above operators. In making strings comparison, there are certain rules to follows: Upper case letters are less than lowercase letters, “A”<“B”<“C”<“D”.....<“Z” and number are less than letters.

Using the If control structure with the Comparison Operators

To effectively control the VB program flow, we shall use the If control structure together with the conditional operators and logical operators. There are basically three types of If control structure, namely **If....Then** statement, **If....Then... Else** statement and **If....Then....ElseIf** statement.

If....Then Statement

This is the simplest control structure which ask the computer to perform a certain action specified by the VB expression if the condition is true. However, when the condition is false, no action will be performed. The general format for the if...then.. statement is

If condition Then

VB expression

End If

Example

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Buttn1.Click
Dim myNumber As Integer
myNumber = TextBox1.Text
If myNumber > 100 Then
Label2.Text = “You win a lucky prize”
End If
End Sub
```

When you run the program and enter a number that is greater than 100, you will see the “You win a lucky prize” statement. On the other hand, if the number entered is less than or equal to 100, you don’t see any display.

If....Then...Else Statement

Using jus If....Then statement is not very useful in programming and it does not provides choices for the users. In order to provide a choice, we can use the **If....Then...Else Statement.** This control structure will ask the computer to perform a certain action specified by the VB expression if the condition is true. And when the condition is false, an alternative action will be executed. The general format for the if...then.. Else statement is

If condition Then

VB expression

Else

VB expression

End If

Example

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim myNumber As Integer
myNumber = TextBox1.Text
If myNumber > 100 Then
Label2.Text = “Congratulation! You win a lucky prize”
Else
Label2.Text = “Sorry, You did not win any prize”
End If
End Sub
```

When you run the program and enter a number that is greater than 100, the statement “Congratulation! You win a lucky prize” will be shown. On the other hand, if the number entered is less than or equal to 100, you will see the “Sorry, You did not win any prize” statement.

Example

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim myNumber, MyAge As Integer
myNumber = TextBox1.Text
MyAge = TextBox2.Text
If myNumber > 100 And myAge > 60 Then
Label2.Text = “Congratulation! You win a lucky prize”
Else
Label2.Text = “Sorry, You did not win any prize”
End If
End Sub
```

This program use the logical And operator beside the conditional operators. This means that both the conditions must be fulfilled in order for the conditions to be true, otherwise the second block of code will be executed. In this example, the number entered must be more than 100 and the age must be more than 60 in order to win a lucky prize, any one of the above conditions not fulfilled will disqualify the user from winning a prize.

If....Then...ElseIf Statement

If there are more than two alternative choices, using jus **If....Then....Else** statement will not be enough. In order to provide more choices, we can use the **If....Then...ElseIf Statement** executed. The general format for the if...then.. Else statement is

If condition Then

VB expression

ElseIf condition **Then**

VB expression

ElseIf condition **Then**

VB expression

.**Else**

VB expression

End If

Example

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim Mark As Integer
Dim Grade as String
Mark = TextBox1.Text
If mark >=80 Then
Grade=”A”
ElseIf  Mark>=60 and Mark<80 then
Grade=”B”
ElseIf  Mark>=40 and Mark<60 then
Grade=”C”
Else
Grade=”D”
End If
End Sub
```

Looping

Visual Basic 2008 allows a procedure to be repeated as many times as long as the processor could support. This is generally called looping. Looping is required when we need to process something repetitively until a certain condition is met. For example, we can design a program that adds a series of numbers until it exceed a certain value, or a program that asks the user to enter data repeatedly until he/she keys in the word ‘Finish’. In Visual Basic 2008, we have three types of Loops, they are the For.....Next loop, the Do loop. and the While.....End while loop

For....Next Loop

The format is:

```
For counter = startNumber to endNumber (Step increment)
   One or more VB statements
Next
```

Sometimes the user might want to get out from the loop before the whole repetitive process is executed, the command to use is Exit For. To exit a For….Next Loop, you can place the Exit For statement within the loop; and it is normally used together with the If…..Then… statement.

Example 1

```
Dim counter as Integer
For  counter = 1 to 10
ListBox1.Items.Add (counter)
Next
```

* The program will enter number 1 to 10 into the list box.

Example 2

```
Dim counter , sum As Integer
For counter=1  to 100 step 10
sum+=counter
ListBox1.Items.Add (sum)
Next
```

* The program will calculate the sum of the numbers as follows: sum=0+10+20+30+40+...

Example 3

```
Dim counter, sum As Integer
sum = 1000
For counter = 100 To 5 Step -5
sum - = counter
ListBox1.Items.Add(sum)
Next
```

* Notice that increment can be negative. The program will compute the subtraction as follow:

1000-100-95-90-.....

Example 4

```
Dim n as Integer
For n=1 to 10
If n>6 then
Exit For
End If
Else
ListBox1.Items.Add (n)
Next
End If
Next
```

* The process will stop when n is greater than 6.

Do Loop

The formats are

a) Do While condition

Block of one or more VB statements

Loop

b) Do

Block of one or more VB statements

Loop While condition

c) Do Until condition

Block of one or more VB statements

Loop

d) Do

Block of one or more VB statements

Loop Until condition

* Exiting the Loop

Sometime we need exit to exit a loop prematurely because of a certain condition is fulfilled. The syntax to use is known as Exit Do. Let's examine the following example

Example 1

```
Do while counter <=1000
    TextBox1.Text=counter
    counter +=1
Loop
```

* The above example will keep on adding until counter >1000.

The above example can be rewritten as

```
Do
   TextBox1.Text=counter
    counter+=1
 Loop until counter>1000
```

Example 2

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim sum, n As Integer
Do
n += 1
sum += n
ListBox1.Items.Add(n & vbTab & sum)
If n = 100 Then
Exit Do
End If
Loop
End Sub
```

Example 3

```
Dim sum, n  As Integer
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim sum, n As Integer
While n <> 100
n += 1
sum = sum + n
ListBox1.Items.Add(n & vbTab & sum)
End While
End Sub
```

While ...End While Loop

The structure of a While….End While is very similar to the Do Loop. it takes the following format:

While *condition*

  Statements

End While

The above loop means that while the condition is not met, the loop will go on. The loop will end when the condition is met.

Functions

A function is similar to a normal procedure but the main purpose of the function is to accept a certain input and return a value which is passed on to the main program to finish the execution. There are two types of functions, the built-in functions (or internal functions) and the functions created by the programmers.

The general format of a function is

FunctionName (arguments)

The arguments are values that are passed on to the function.

In this lesson, we are going to learn two very basic but useful internal functions of Visual basic, i.e. The MsgBox( ) and InputBox ( ) functions.

MsgBox ( ) Function

The objective of MsgBox is to produce a pop-up message box and prompt the user to click on a command button before he /she can continues. This format is as follows:

yourMsg = MsgBox(Prompt, Style Value, Title)

The first argument, Prompt, will display the message in the message box. The Style Value will determine what type of command buttons appear on the message box, please refer to the following Table for types of command button displayed. The Title argument will display the title of the message board.

Style Value	Named Constant	Buttons Displayed
0	vbOkOnly	Ok button
1	vbOkCancel	Ok and Cancel buttons
2	vbAbortRetryIgnore	Abort, Retry and Ignore buttons.
3	vbYesNoCancel	Yes, No and Cancel buttons
4	vbYesNo	Yes and No buttons
5	vbRetryCancel	Retry and Cancel buttons

We can use named constant in place of integers for the second argument to make the programs more readable. In fact, VB6 will automatically shows up a list of names constant where you can select one of them.

Example:

yourMsg = MsgBox(“Click OK to Proceed”, 1, “Startup Menu”)

and

yourMsg = MsgBox“Click OK to Proceed”. vbOkCancel,”Startup Menu”)

Are the same. yourMsg is a variable that holds values that are returned by the MsgBox ( ) function. The values are determined by the type of buttons being clicked by the users. It has to be declared as Integer data type in the procedure or in the general declaration section. The following table shows the values, the corresponding named constant and buttons.

Value	Named Constant	Button Clicked
1	vbOk	Ok button
2	vbCancel	Cancel button
3	vbAbort	Abort button
4	vbRetry	Retry button
5	vbIgnore	Ignore button
6	vbYes	Yes button
7	vbNo	No button

Example 1

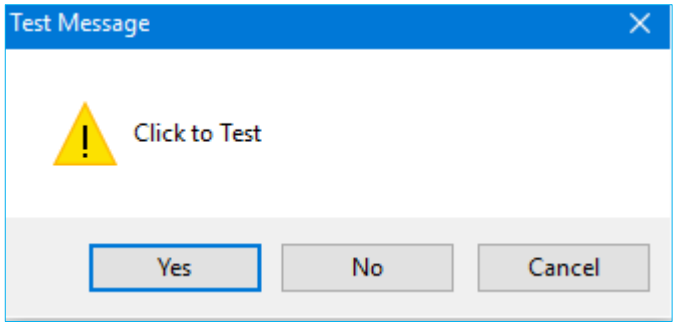
```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim testmsg As Integer
testmsg = MsgBox("Click to test", 1, "Test message")
If testmsg = 1 Then
MessageBox.Show("You have clicked the OK button")
Else
MessageBox.Show("You have clicked the Cancel button")
End If
End Sub
```

To make the message box looks more sophisticated, you can add an icon besides the message. There are four types of icons available in VB2008 as shown in the following table.

Value	Named Constant	Icon
16	vbCritical	
32	vbQuestion	
48	vbExclamation	
64	vbInformation	

Example 2

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim testMsg As Integer
testMsg = MsgBox("Click to Test", vbYesNoCancel + vbExclamation, "Test Message")
If testMsg = 6 Then
MessageBox.Show("You have clicked the yes button")
ElseIf testMsg = 7 Then
MessageBox.Show("You have clicked the NO button")
Else
MessageBox.Show("You have clicked the Cancel button")
End If
End Sub
```



The InputBox( ) Function

An InputBox( ) function will display a message box where the user can enter a value or a message in the form of text. In VB2005, you can use the following format:

myMessage=InputBox(Prompt, Title, default_text, x-position, y-position)

myMessage is a variant data type but typically it is declared as string, which accept the message input by the users. The arguments are explained as follows:

- Prompt - The message displayed normally as a question asked.
- Title - The title of the Input Box.
- default-text - The default text that appears in the input field where users can use it as his intended input or he may change to the message he wish to enter.
- x-position and y-position - the position or tthe coordinates of the input box.

However, the format won't work in VB2008 because InputBox is considered a namespace.

So, you need to key in the full reference to the Inputbox namespace, which is

Microsoft.VisualBasic.InputBox(Prompt, Title, default_text, x-position, y-position)

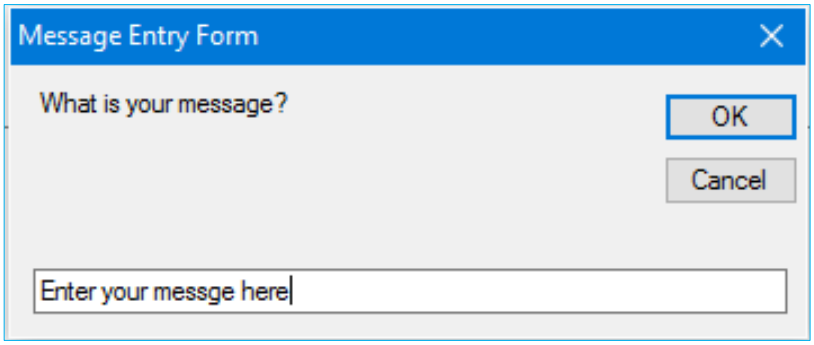
The parameters remain the same.

Example

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim userMsg As String
```

```
userMsg = Microsoft.VisualBasic.InputBox("What is your message?", "Message Entry Form", "Enter your message here", 500, 700)
If userMsg <> "" Then
MessageBox.Show(userMsg)
Else
MessageBox.Show("No Message")
End If
End Sub
```

The inputbox will appear as shown in the figure below when you press the command button



String Functions

The Mid Function

The Mid function is used to retrieve a part of text form a given phrase. The format of the Mid Function is

Mid(phrase, position, n)

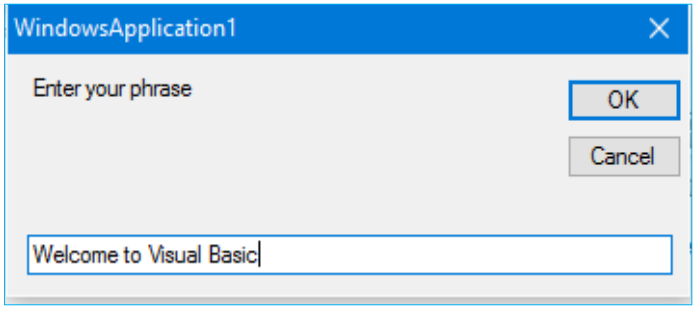
Where

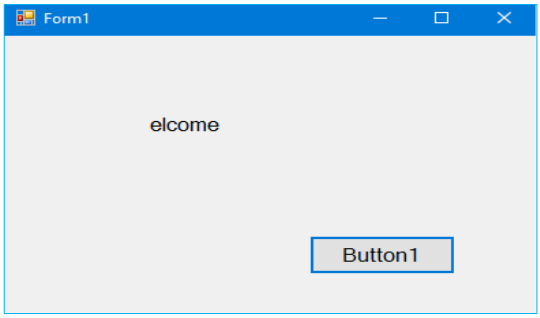
- phrase is the string from which a part of text is to be retrieved
- position is the starting position of the phrase from which the retrieving process begins
- n is the number of characters to retrieve.

Example:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim myPhrase As String
myPhrase = Microsoft.VisualBasic.InputBox("Enter your phrase")
Label1.Text = Mid(myPhrase, 2, 6)
End Sub
```

when the user clicks the command button, an inputbox will pop up asking the user to input a phrase. After a phrase is entered and the OK button is pressed, the label will show the extracted text starting from position 2 of the phrase and the number of characters extracted is 6. The diagrams are shown below:





The Right

Function

The Right function extracts the right portion of a phrase. The format is

Microsoft.Visualbasic.Right ("Phrase", n)

Where n is the starting position from the right of the phase where the portion of the phrase is going to be extracted.

Microsoft.Visualbasic.Right ("Visual Basic", 4) = asic

Example: The following code extracts the right portion any phrase entered by the user.

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim myword As String
myword = TextBox1.Text
Label1.Text = Microsoft.VisualBasic.Right (myword, 4)
End Sub
```

The Trim Function

The Trim function trims the empty spaces on both side of the phrase. The format is

Trim("Phrase")

For example, Trim (" Visual Basic ") = Visual basic

Example

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim myPhrase As String
myPhrase = Microsoft.VisualBasic.InputBox("Enter your phrase")
Label1.Text = Trim(myPhrase)
End Sub
```

The Ltrim Function

The Ltrim function trims the empty spaces of the left portion of the phrase.

The format is **Ltrim("Phrase")**

For example, Ltrim (" Visual Basic")= Visual basic

The Rtrim Function

The Rtrim function trims the empty spaces of the right portion of the phrase. The format is

Rtrim("Phrase")

For example, Rtrim ("Visual Basic ") = Visual Basic

The InStr function

The **InStr** function looks for a phrase that is embedded within the original phrase and returns the starting position of the embedded phrase. The format is

Instr (n, original phase, embedded phrase)

Where n is the position where the Instr function will begin to look for the embedded phrase. For example

Instr(1, "Visual Basic"," Basic") = 8

*The function returns a numeric value.

You can write a program code as shown below:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Label1.Text = InStr(1, "Visual Basic", "Basic")
End Sub
```

The Ucase and the Lcase Functions

The **Ucase** function converts all the characters of a string to capital letters. On the other hand, the **Lcase** function converts all the characters of a string to small letters.

The format is

```
Microsoft.VisualBasic.UCase(Phrase)
Microsoft.VisualBasic.LCase(Phrase)
```

For example,

Microsoft.VisualBasic.Ucase("Visual Basic") =VISUAL BASIC
Microsoft.VisualBasic.Lcase("Visual Basic") =visual basic

The Chr and the Asc functions

The **Chr** function returns the string that corresponds to an ASCII code while the **Asc** function converts an ASCII character or symbol to the corresponding ASCII code. ASCII stands for "American Standard Code for Information Interchange". Altogether there are 255 ASCII codes and as many ASCII characters. Some of the characters may not be displayed as they may rep-resent some actions such as the pressing of a key or produce a beep sound. The format of the Chr function is

Chr(charcode)

and the format of the Asc function is

Asc(Character)

The following are some examples:

Chr(65)=A, Chr(122)=z, Chr(37)=% ,

Asc("B")=66, Asc("&")=38

Math Functions

We have learned how to Vb2008 can perform arithmetic functions using standard mathematical operators. However, for more complex mathematical calculations, we need to use the built-in math functions in VB2008. There are numerous built-in mathematical functions in Visual Basic which we will intro-duce them one by one.

The Abs function

The Abs return the absolute value of a given number.

The syntax is

Math. Abs (number)

* The Math keyword here indicates that the Abs function belong to the Math class. However, not all mathematical functions belong to the Math class.

The Fix Function

The Fix function truncates the decimal part of a positive number and returns the largest integer smaller than the number. However, when the number is negative, it will return smallest integer larger than the number. For ex-ample, Fix(9.2)=9 but Fix(-9.4)=-9

Example:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim num1, num2 As Single
num1 = TextBox1.Text
num2 = Fix(num1)
Label1.Text = num2
End Sub
```

The Int Function

The Int is a function that converts a number into an integer by truncating its decimal part and the resulting integer is the largest integer that is smaller than the number. For example

Int(2.4)=2, Int(6.9)=6 , Int(-5.7)=-6, Int(-99.8)=-100

The Log Function

The Log function is the function that returns the natural logarithm of a number. For example, Log(10)=2.302585

Example:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim num1, num2 As Single
num1 = TextBox1.Text
num2 = Math.Log(num1)
Label1.Text = num2
End Sub
```

* The logarithm of num1 will be displayed on label1

The Rnd( ) Function

The Rnd is very useful when we deal with the concept of chance and probability. The Rnd function returns a random value between 0 and 1. Random numbers in their original form are not very useful in programming until we convert them to integers. For example, if we need to obtain a random output of 6 integers ranging from 1 to 6, which makes the program behave like a virtual dice, we need to convert the random numbers to integers using the formula **Int(Rnd*6)+1.**

Example:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim num as integer
Randomize( )
Num=Int(Rnd()*6)+1
Label1.Text=Num
End Sub
```

The Round Function

The **Round** function is the function that rounds up a number to a certain number of decimal places. The Format is Round (n, m) which means to round a number n to m decimal places. For example, Math.Round (7.2567, 2) =7.26

Example

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Dim num1, num2 As Single
num1 = TextBox1.Text
num2 = Math.Round(num1, 2)
Label1.Text = num2
End Sub
```

* The Math keyword here indicates that the Round function belong to the Math class.

Formatting Functions

The **Format** function is a very powerful formatting function which can display the numeric values in various forms. There are two types of Format function, one of them is the built-in or predefined format while another one can be defined by the users.

Predefined Format

Format (n, "style argument")

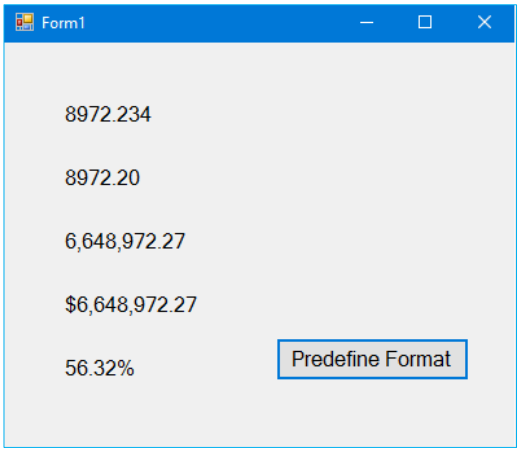
where n is a number and the list of style arguments is given in table below:

Style argument	Explanation	Example
General Number	To display the number without having separators between thousands.	Format(8972.234, "General Number")=8972.234
Fixed	To display the number without having separators between thousands and rounds it up to two decimal places.	Format(8972.2, "Fixed")=8972.23
Standard	To display the number with separators or separators between thousands and rounds it up to two decimal places.	Format(6648972.265, "Standard")= 6,648,972.27
Currency	To display the number with the dollar sign in front, has separators between thousands as well as rounding it up to two decimal places.	Format(6648972.265, "Currency")= $6,648,972.27
Percent	Converts the number to the percentage form and displays a % sign and rounds it up to two decimal places.	Format(0.56324, "Percent")=56.32%

Example

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Label1.Text = Format (8972.234, “General Number”)\
Label2.Text = Format (8972.2, “Fixed”)
Label3.Text = Format (6648972.265, “Standard”)
Label4.Text = Format (6648972.265, “Currency”)
Label5.Text = Format (0.56324, “Percent”)
End Sub
```

The Output window is shown below:



User-defined Format

Format (n, “user’s format”)

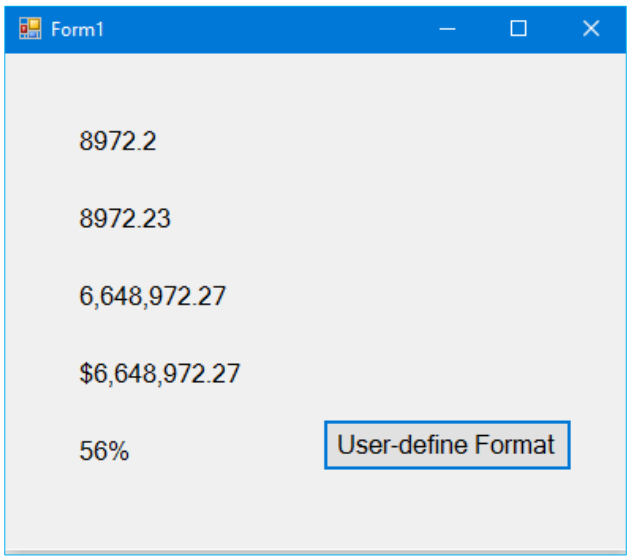
Although it is known as user-defined format, we still need to follows certain formatting styles. Examples of user-defined formatting style are listed in the table below:

Example	Explanation	Output
Format (781234.57,”0”)	Rounds to whole number without separators between thousands.	781235
Format (781234.57,”0.0”)	Rounds to 1 decimal place without separators between thousands.	781234.6
Format (781234.576,”0.00”)	Rounds to 2 decimal places without separators between thousands.	781234.58
Format (781234.576,”#,##0.00”)	Rounds to 2 decimal places with separators between thousands.	781,234.58
Format (781234.576,”$#,##0.00”)	Shows dollar sign and rounds to 2 decimal places with separators between thousands.	$781,234.58
Format (0.576,”0%”)	Converts to percentage form without decimal places.	58%
Format (0.5768,”0.00%”)	Converts to percentage form with 2 decimal places	57.68%

Example

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Label1.Text = Format (8972.234, “0.0”)
Label2.Text = Format (8972.2345, “0.00”)
Label3.Text = Format (6648972.265, “#,##0.00”)
Label4.Text = Format (6648972.265, “$#,##0.00”)
Label5.Text = Format (0.56324, “0%”)
End Sub
```

The Output window is shown below:



Formatting Date & Time

Formatting Date and time using predefined formats

Date and time can be formatted using predefined formats and also user-defined formats. The predefined formats of date and time are shown in the table below:

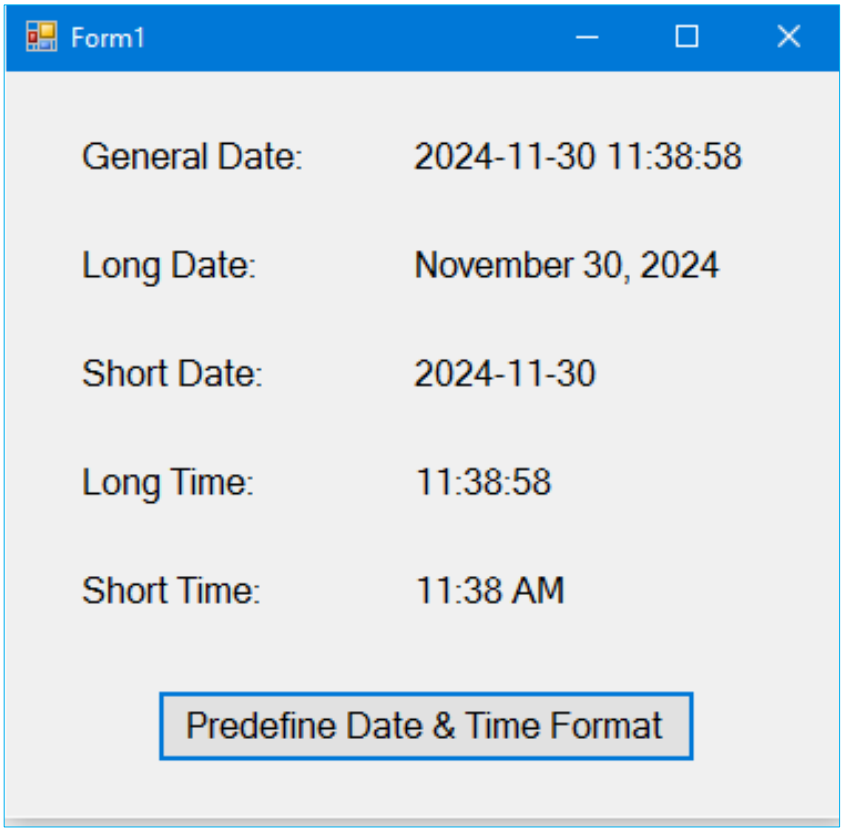
Format	Explanation
Format (Now, “General date”)	Formats the current date and time.
Format (Now, “Long Date”)	Displays the current date in long format.
Format (Now, “Short date”)	Displays current date in short format.
Format (Now, “Long Time”)	Display the current time in long format.
Format (Now, “Short Time”)	Display the current time in short format.

Instead of “General date”, you can also use the abbreviated format “G”, i.e. Format (Now, “G”). And for “Long Time”, you can use the abbreviated format “T”. As for “Short Time”, you may use the abbreviated format “t”

Example

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Label1.Text = Format (Now, “General Date”)
Label2.Text = Format (Now, “Long Date”)
Label3.Text = Format (Now, “short Date”)
Label4.Text = Format (Now, “Long Time”)
Label5.Text = Format (Now, “Short Time”)
End Sub
```

The output is shown in the diagram below:



Formatting Date and time using user-defined formats

Beside using the predefined formats, you can also use the user-defined formatting functions. The general format of a user-defined for date/time is

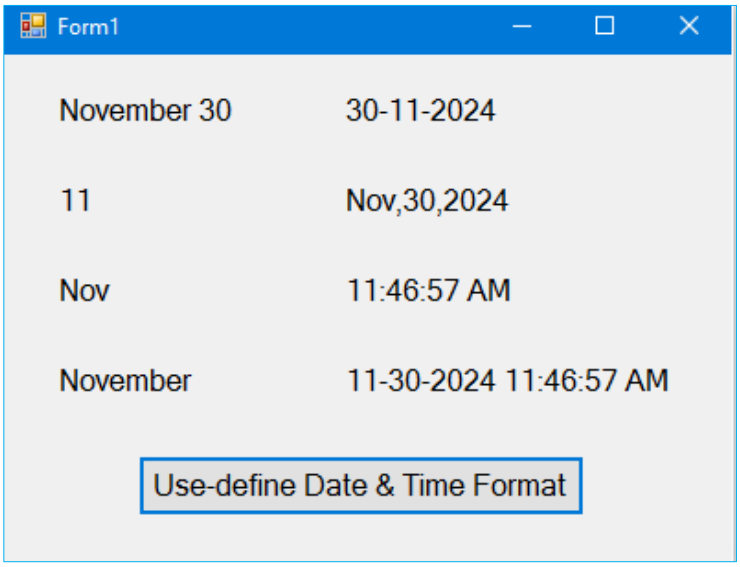
Format (expression,style)

Some of the user-defined formats of date and time are shown in the table below:

Format	Explanation
Format (Now, “M”)	Displays current month and date
Format (Now, “MM”)	Displays current month in double digits.
Format (Now, “MMM”)	Displays abbreviated name of the current month
Format (Now, “MMMM”)	Displays full name of the current month.
Format (Now, “dd/MM/yyyy”)	Displays current date in the day/ month/year format.
Format (Now, “MMM,d,yyyy”)	Displays current date in the Month, Day, Year Format
Format (Now, “h:mm:ss tt”)	Dispalys current time in hour:minute:second format and show am/pm
Format (Now, “MM/dd/yyyy h:mm:ss)	Dispalys current date and time in hour:minute:second format

Example

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Label1.Text = Format (Now, “M”)
Label2.Text = Format (Now, “MM”)
Label3.Text = Format (Now, “MMM”)
Label4.Text = Format (Now, “MMMM”)
Label5.Text = Format (Now, “dd/MM/yyyy”)
Label6.Text = Format (Now, “MMM,d,yyyy”)
Label7.Text = Format (Now, “h:mm:ss tt”)
Label8.Text = Format (Now, “MM/dd/yyyy h:mm:ss tt”)
End Sub
```

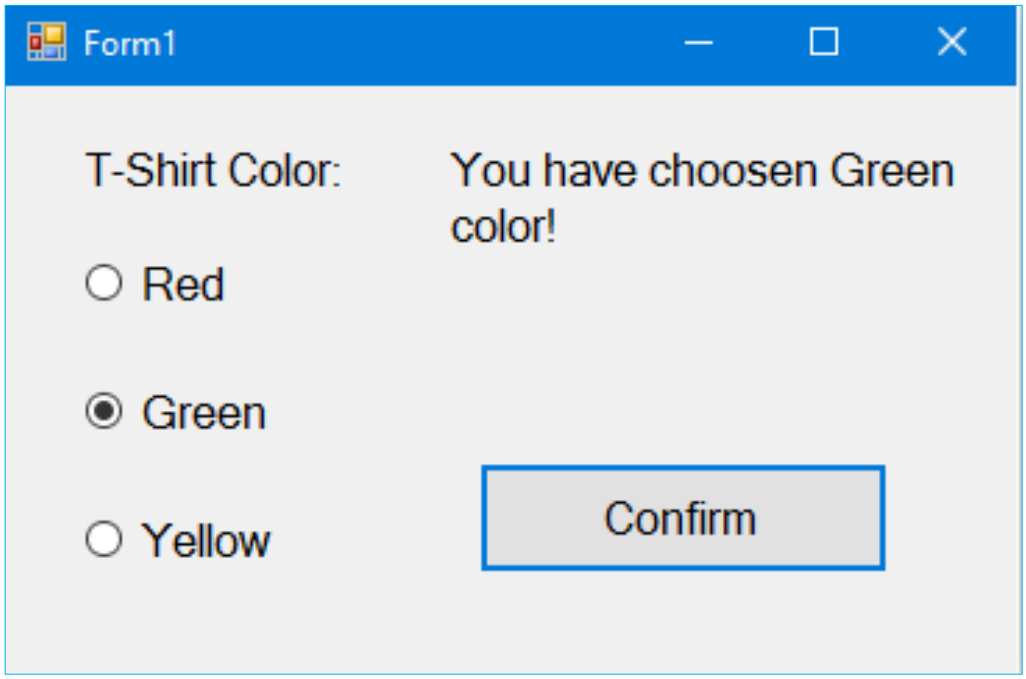


Using Radio Buttons

The radio button is also a very useful control in Visual Basic 2008. However, it operates differently from the check boxes. While the checkboxes work independently and allows the user to select one or more items, radio buttons are mutually exclusive, which means the user can only choose one item only out of a number of choices. Here is an example which allows the users to select one color only.

Example 1:

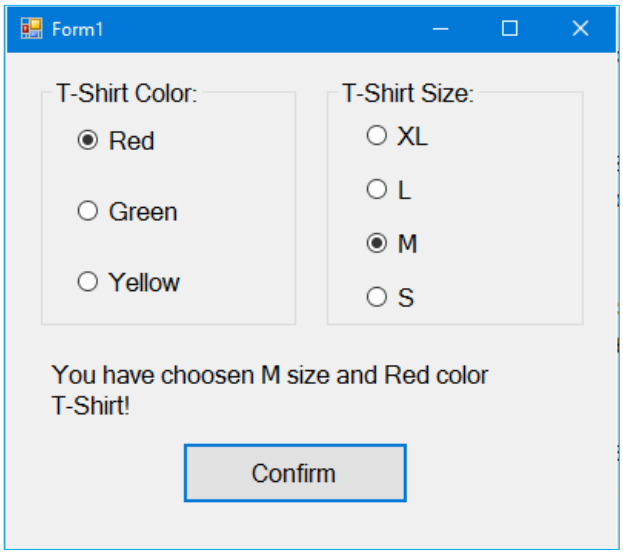
```
Dim strColor As String
Private Sub RadioButton1_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles
RadioButton1.CheckedChanged
strColor = “Red”
End Sub
Private Sub RadioButton2_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
HandlesRadioButton2.CheckedChanged
strColor = “Green”
End Sub
Private Sub RadioButton3_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
HandlesRadioButton3.CheckedChanged
strColor = “Yellow”
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Label2.Text = "You have choosen " & strColor & " color!"
End Sub
```



Example 2

```
Dim strColor As String
Dim strSize As String
Private Sub RadioButton1_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles RadioButton1.CheckedChanged
strColor = “Red”
End Sub
Private Sub RadioButton2_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles RadioButton2.CheckedChanged
strColor = “Green”
End Sub
Private Sub RadioButton3_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles RadioButton3.CheckedChanged
strColor = “Yellow”
End Sub
Private Sub RadioButton4_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles RadioButton4.CheckedChanged
strSize = “XL”
End Sub
Private Sub RadioButton5_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles RadioButton5.CheckedChanged
strSize = “L”
End Sub
Private Sub RadioButton6_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles RadioButton6.CheckedChanged
strSize = “M”
End Sub
Private Sub RadioButton7_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles RadioButton7.CheckedChanged
strSize = “S”
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
Label2.Text = "You have choosen " & strsize & " size and " & strColor & " color T-Shirt!"
End Sub
```

:

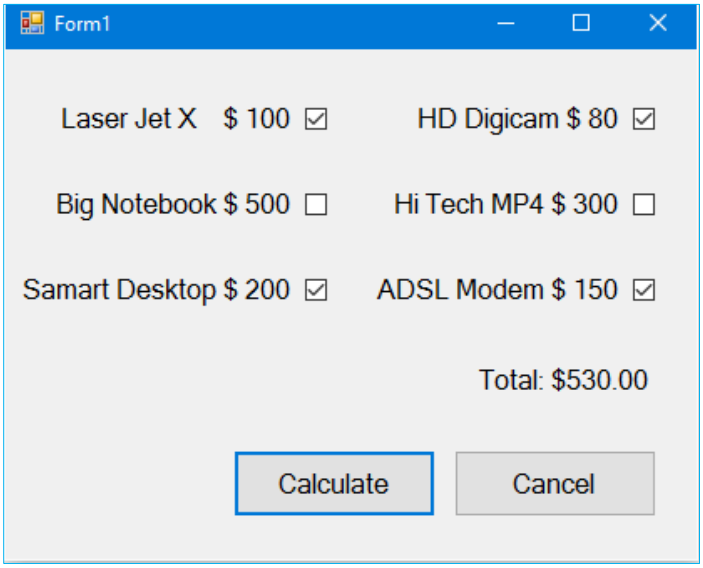


Using Check Box

The Check box is a very useful control in Visual Basic 2008. It allows the user to select one or more items by checking the checkbox/checkboxes concerned. For example, in the Font dialog box of any Microsoft Text editor like FrontPage, there are many checkboxes under the Effects section such as that shown in the diagram below.

Example 1:

Private Sub BtnCalculate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles BtnCalculate.Click
Const LX As Integer = 100
Const BN As Integer = 500
Const SD As Integer = 200
Const HD As Integer = 80
Const HM As Integer = 300
Const AM As Integer = 150
Dim sum As Integer
If CheckBox1.Checked = True Then
sum += LX
End If
If CheckBox2.Checked = True Then
sum += BN
End If
If CheckBox3.Checked = True Then
sum += SD
End If
If CheckBox4.Checked = True Then
sum += HD
End If
If CheckBox5.Checked = True Then



```
sum += HM
End If
If CheckBox6.Checked = True Then
sum += AM
End If
Label5.Text = sum.ToString("c")
End Sub
```

Example 2:

In this example, the user can enter text into a textbox and format the font using the three checkboxes that represent bold, italic and underline.

The code is as follow:

```
Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged
If CheckBox1.Checked Then
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style Or FontStyle.Bold)
Else
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style And Not FontStyle.Bold)
End If
End Sub
Private Sub CheckBox2_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CheckBox2.CheckedChanged
If CheckBox2.Checked Then
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style Or FontStyle.Italic)
Else
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style And Not FontStyle.Italic)
End If
End Sub
Private Sub CheckBox3_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CheckBox3.CheckedChanged
If CheckBox2.Checked Then
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style Or FontStyle.Underline)
Else
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style And Not FontStyle.Underline)
End If
End Sub
```

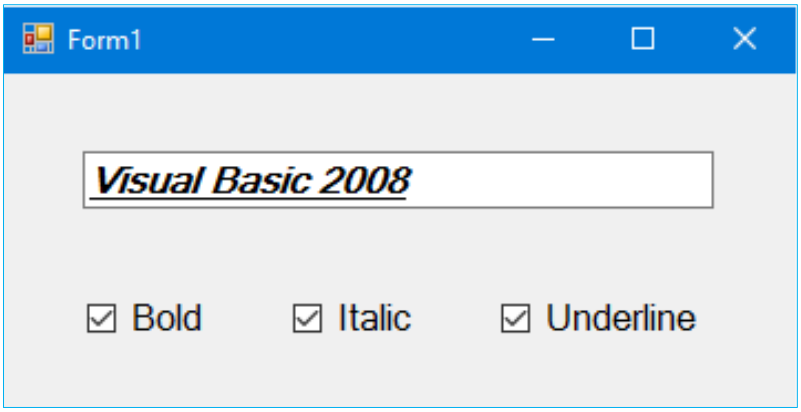
* The above program uses the CheckedChanged event to respond to the user selection by checking a particular checkbox, it is similar to the click event. The statement

```
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style Or FontStyle.Italic)
```

will retain the original font type but change it to italic font style.

TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style And Not FontStyle.Italic)

will also retain the original font type but change it to regular font style. (The other statements emply the same logic)



Error Handling

Error handling is an essential procedure in Visual Basic 2008 programming because it can help make the program error-free. An error-free program can run smoothly and efficiently, and the user does not have to face all sorts of problems such as program crash or system hang.

Errors often occur due to incorrect input from the user. For example, the user might make the mistake of attempting to enter a text (string) to a box that is designed to handle only numeric values such as the weight of a person, the computer will not be able to perform arithmetic calculation for text therefore will create an error. These errors are known as synchronous errors.

Therefore, a good programmer should be more alert to the parts of program that could trigger errors and should write errors handling code to help the user in managing the errors. Writing errors handling code should be considered a good practice for Visual Basic programmers, so do try to finish a program fast by omitting the errors handling code. However, there should not be too many errors handling code in the program as it creates problems for the programmer to maintain and troubleshoot the program later.

VB2008 has improved a lot in built-in errors handling compared to Visual Basic 6. For example, when the user attempts to divide a number by zero, Vb2008 will not return an error message but gives the **'infinity'** as the answer (although this is mathematically incorrect, because it should be undefined)

Using On Error GoTo Syantax

Visual Basic 2008 still supports the VB6 errors handling syntax that is the **On Error GoTo *program_label*** structure. Although it has a more advanced error handling method, we shall deal with that later. We shall now learn how to write errors handling code in VB2008. The syntax for errors handling is

On Error GoTo *program_label*

where ***program_label*** is the section of code that is designed by the programmer to handle the error committed by the user. Once an error is detected, the program will jump to the ***program_label*** section for error handling.

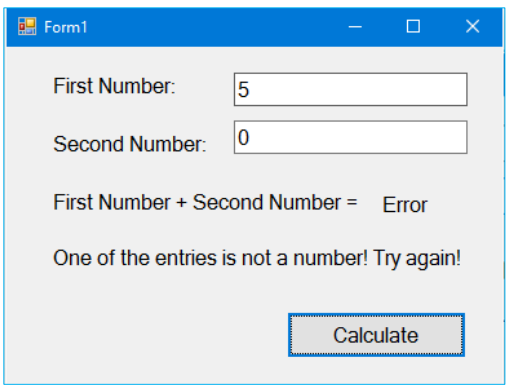
Example 1:

In this example, we will deal with the error of entering non-numeric data into the textboxes that supposed to hold numeric values. The program_label here is error_hanldler. when the user enter a non-numeric values into the textboxes, the error message will display the the text"One of the entries is not a number! Try again!". If no error occur, it will display the correct answer. Try it out yourself.

The Code

```
Private Sub CmdCalculate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CmdCalculate.Click
Lbl_ErrorMsg.Visible = False
Dim firstNum, secondNum As Double
On Error GoTo error_handler
firstNum = Txt_FirstNumber.Text
secondNum = Txt_SecondNumber.Text
Lbl_Answer.Text = firstNum / secondNum
Exit Sub 'To prevent error handling even the inputs are valid
error_handler:
Lbl_Answer.Text = "Error"
Lbl_ErrorMsg.Visible = True
Lbl_ErrorMsg.Text = "One of the entries is not a number! Try again!"
End Sub
```

The Output



Errors Handling using Try.....Catch....End Try Structure

VB2008 has adopted a new approach in handling errors, or rather exceptions handling. It is supposed to be more efficient than the old **On Error Goto** method, where it can handle various types of errors within the **Try...Catch...End Try** structure.

The structure looks like this

Try
    Statements
Catch *exception*_variable as Exception
    Statements to deal with exceptions
End Try

Example 2

Instead of using On Error GoTo method, we can use the **Try...Catch...End Try** method. In this example, the **Catch** statement will catch the exception when the user enters a non-numeric data and returns the error message. If there is no exception, there will not any action from the Catch statement and the program returns the correct answer.

The code

```
Private Sub CmdCalculate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CmdCalculate.Click
Lbl_ErrorMsg.Visible = False
Dim firstNum, secondNum, answer As Double
Try
firstNum = Txt_FirstNumber.Text
secondNum = Txt_SecondNumber.Text
answer = firstNum / secondNum
Lbl_Answer.Text = answer
Catch ex As Exception
Lbl_Answer.Text = “Error”
Lbl_ErrorMsg.Visible = True
Lbl_ErrorMsg.Text = “One of the entries is not a number! Try again!”
End Try
End Sub
```

w

ADO.NET and Database Handling in VB2008

ADO.NET Overview

ADO.NET is an evolution of the ADO data access model that directly addresses user requirements for developing scalable applications. It was designed specifically for the web with scalability, statelessness, and XML in mind.

ADO.NET uses some ADO objects, such as the Connection and Command objects, and also introduces new objects. Key new ADO.NET objects include the DataSet, DataReader, and DataAdapter.

A DataAdapter is the object that connects to the database to fill the DataSet. Then, it connects back to the database to update the data there, based on operations performed while the DataSet held the data. In the past, data processing has been primarily connection-based. Now, in an effort to make multi-tiered apps more efficient, data processing is turning to a message-based approach that revolves around chunks of information. At the center of this approach is the DataAdapter, which provides a bridge to retrieve and save data between a DataSet and its source data store. It accomplishes this by means of requests to the appropriate SQL commands made against the data store.

While the DataSet has no knowledge of the source of its data, the managed provider has detailed and specific information. The role of the managed provider is to connect, fill, and persist the DataSet to and from data stores. The OLE DB and SQL Server .NET Data Providers (System.Data.OleDb and System.Data.SqlClient) that are part of the .Net Framework provide four basic objects: the Command, Connection, DataReader and DataAdapter. In the remaining sections of this document, we'll walk through each part of the DataSet and the OLE DB/SQL Server .NET Data Providers explaining what they are, and how to program against them.

The following sections will introduce you to some objects that have evolved, and some that are new. These objects are:

- **Connections:** For connection to and managing transactions against a database.
- **Commands:** For issuing SQL commands against a database.
- **DataReaders:** For reading a forward-only stream of data records from a SQL Server data source.
- **DataSets:** For storing, remoting and programming against flat data, XML data and relational data.
- **DataAdapters:** For pushing data into a DataSet, and reconciling data against a database.

Note When dealing with connections to a database, there are two different options: SQL Server .NET Data Provider (System.Data.SqlClient) and OLE DB .NET Data Provider (System.Data.OleDb). In these samples we will use the SQL Server .NET Data Provider. These are written to talk directly to Microsoft SQL Server. The OLE DB .NET Data Provider is used to talk to any OLE DB provider (as it uses OLE DB underneath).

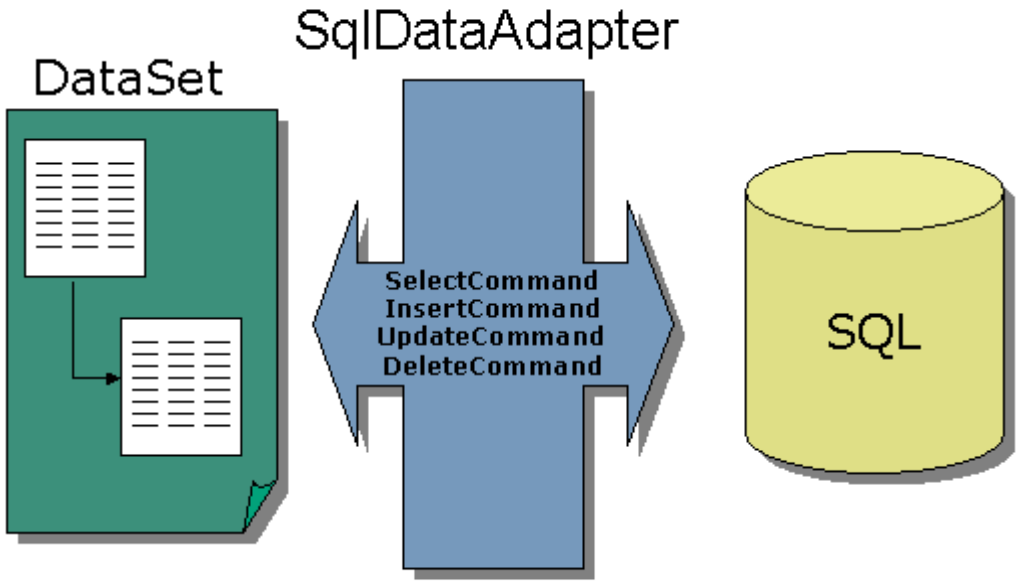
Connections Object

Connections are used to 'talk to' databases, and are respresented by provider-specific classes such as **SQLConnection**. Commands travel over connections and resultsets are returned in the form of streams which can be read by a **DataReader** object, or pushed into a **DataSet** object.

Commands Object

Commands contain the information that is submitted to a database, and are represented by provider-specific classes such as **SQLCommand**. A command can be a stored procedure call, an UPDATE statement, or a statement that returns results. You can also use input and output parameters, and return values as part of your command syntax. The example below shows how to issue an INSERT statement against the **Northwind** database.

The records are appropriately mapped to the given commands accordingly.



Connecting an MS-Access Database to VB2008 Application

There are two approaches to binding MS Access database table to DataGrid or DataGridView in VB.NET.

I presume D:\myDB.mdb database file has one table named User.

Approach 1: Using Data Wizard

Data menu -> Add New Data Source

During wizard, select D:\myDB.mdb and User table, finishing it will create a StaffDBDataSet.xsd file in

Solution Explorer.

Data menu -> Show Data Sources

In Data Sources panel, drag & drop the entire **User** table to your Form, then **UserDataGridView**, UserDBDataSet, UserTableAdapter, UserBindingNavigator and UserBindingSource controls will be added automatically to Form, and **UserDataGridView** has been bound.

Approach 2: Using pure code

Please check the following code sample:

Code Snippet

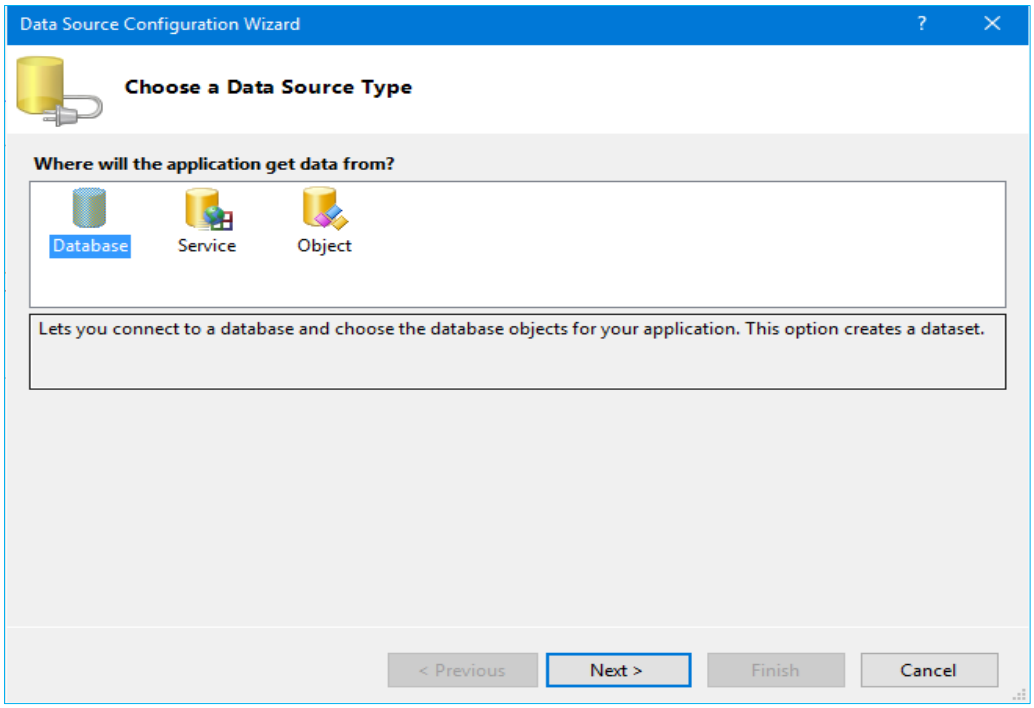
```
Imports System.Data.OleDb
Public Class Form1
' Binding database table to DataGridView
```

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
    Dim con As OleDbConnection = New
    OleDbConnection("Provider=Microsoft.jet.oledb.4.0;datasource=D:\myDB.mdb")
    Dim cmd As OleDbCommand = New OleDbCommand("Select * FROM User", con)
    con.Open()
    Dim myDA As OleDbDataAdapter = New OleDbDataAdapter(cmd)
    Dim myDataSet As DataSet = New DataSet()
    myDA.Fill(myDataSet, "MyTable")
    DataGridView1.DataSource = myDataSet.Tables("MyTable").DefaultView
    con.Close()
    con = Nothing
End Sub

' Retrieve table records into DataReader object
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
    Dim con As New
    OleDbConnection("Provider=Microsoft.jet.oledb.4.0;data source=D:\myDB.mdb")
    Dim cmd As OleDbCommand = New OleDbCommand("SELECT * FROM User", con)
    con.Open()
    Dim sdr As OleDbDataReader = cmd.ExecuteReader()
    While sdr.Read = True
        MessageBox.Show(sdr.Item("username"))
    End While
    Sdr.Close()
    con.Close()
    con = Nothing
    End Sub
End Class
```

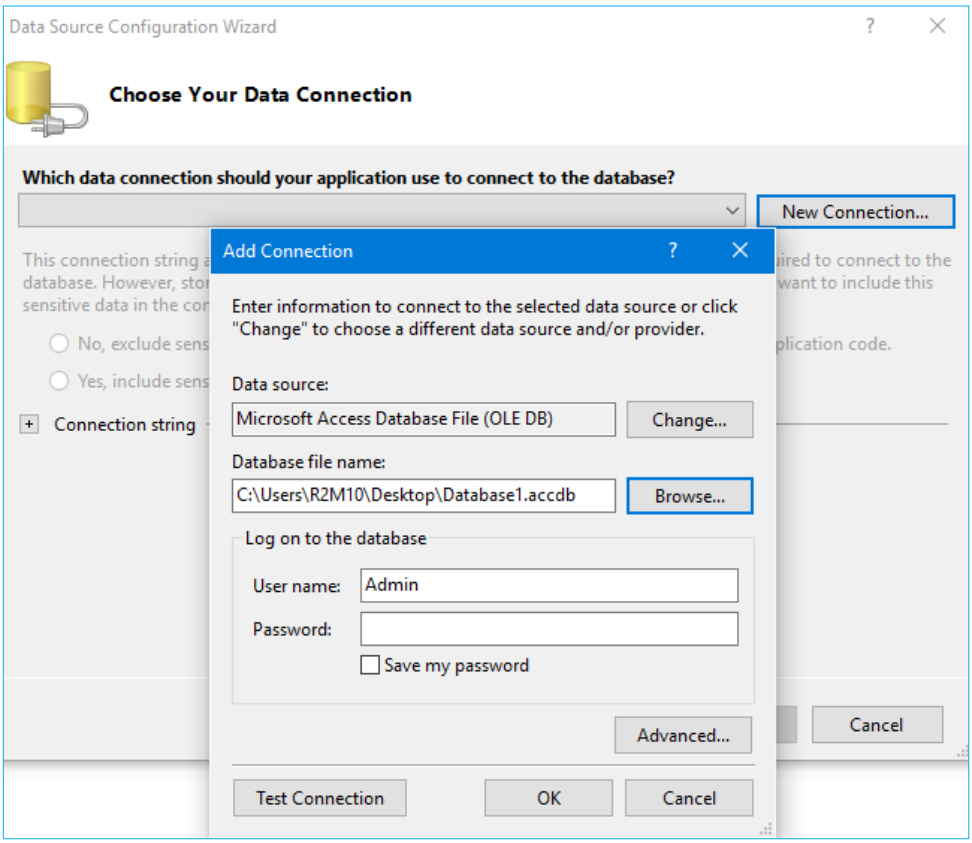
Exploring General Database Operations in VB2008

Database connectivity using Data Wizard

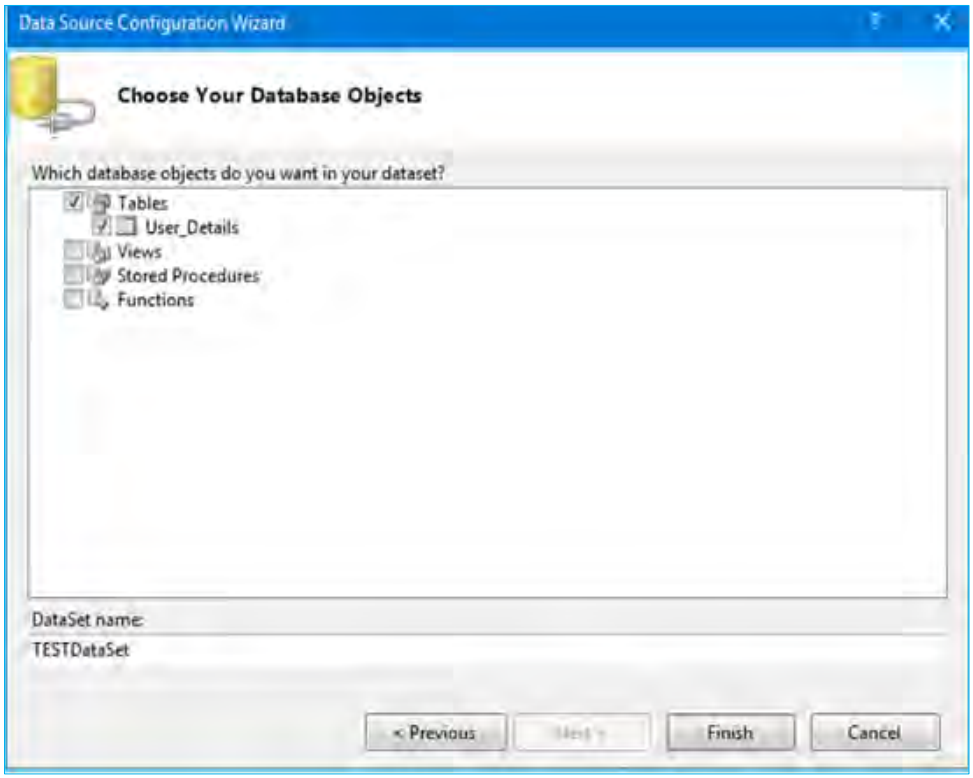
1) Open a new project/Add a new form to your existing project
2) Data-Add new data source
3) Following screen appears



4) Select the Database option and click next
5) Click New Connection option and select the MS-Access database file name by clicking the Browse button from the following screens



6) Click Next and select your table from the screen displayed



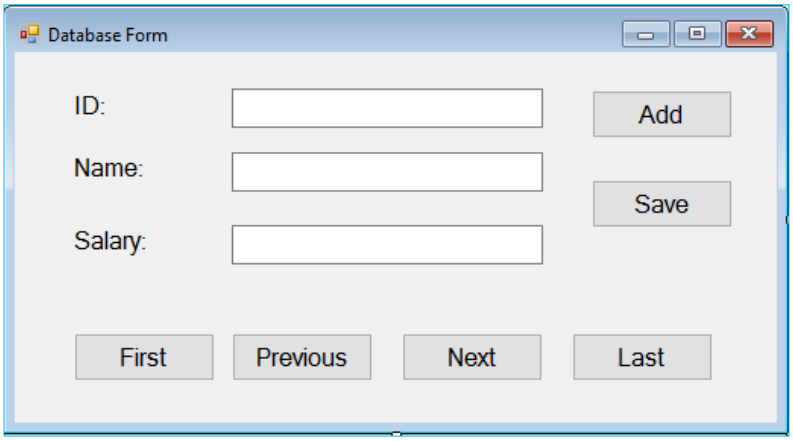
7) Click Finish.
8) Click the Data Sources button and drag each field into the form.
9) Your database manipulation form is ready. Run the project.
10) The code that .NET adds in the background

```
Public Class mydbproj
Private Sub Table1BindingNavigatorSaveItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Table1BindingNavigatorSaveItem.Click
Me.Validate()
Me.Table1BindingSource.EndEdit()
Me.TableAdapterManager.UpdateAll(Me.Database1DataSet4)
End Sub

Private Sub dbform_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
'TODO: This line of code loads data into the 'Database1DataSet4.Table1' table. You can move, or remove it, as needed.
Me.Table1TableAdapter.Fill(Me.Database1DataSet4.Table1)
End Sub
End Class
```

Writing pure Code

We are assuming an Ms-Access 2007 database with a table with fields ID, Name and Basic

Form Design



Code

```
Imports System.Data.OleDb
Public Class frmdb
Dim con As New OleDbConnection
Dim ds As New DataSet
Dim count As Integer
Dim da As OleDbDataAdapter
Dim tot As Integer
Private Sub Navigaterecords()
If (count >= 0 And count < tot) Then
txtId.Text = ds.Tables("Table1").Rows(count).Item(0)
txtName.Text = ds.Tables("Table1").Rows(count).Item(1)
txtBasic.Text = ds.Tables("Table1").Rows(count).Item(2)
End If
End Sub

Private Sub cmdPrev_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdPrev.Click
If count > 0 And tot > 0 Then
count -= 1
ElseIf count >= 0 Then
Navigaterecords()
End If
End Sub

Private Sub cmdNext_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdNext.Click
If count < tot And tot > 0 Then
count += 1
ElseIf count < tot Then
Navigaterecords()
End If
End Sub
```

```
Private Sub cmdSave_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSave.Click
Dim pr As New OleDbParameter
da.InsertCommand = New OleDbCommand("Insert into Table1(Name,Basic) values (@nm,@bas)", con)
pr = da.InsertCommand.Parameters.Add("nm", OleDbType.VarChar)
pr.SourceVersion = DataRowVersion.Current
pr.SourceColumn = "Name"
pr = da.InsertCommand.Parameters.Add("bas", OleDbType.Double)
pr.SourceVersion = DataRowVersion.Current
pr.SourceColumn = "Basic"
Dim dr As DataRow
dr = ds.Tables("Table1").NewRow()
dr(1) = txtName.Text
dr(2) = txtBasic.Text
ds.Tables("Table1").Rows.Add(dr)
Try
da.Update(ds, "Table1")
MsgBox("1 Record Added...")
da = New OleDbDataAdapter("Select * from Table1", con)
da.Fill(ds, "Table1")
tot = ds.Tables(0).Rows.Count
Navigaterecords()
count = ds.Tables(0).Rows.Count – 1
Navigaterecords()
Catch ex As Exception
End Try
End Sub

Private Sub cmdFirst_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdFirst.Click
If tot > 0 Then
count = 0
Navigaterecords()
End If
End Sub

Private Sub cmdLast_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdLast.Click
If tot > 0 Then
count = tot – 1
Navigaterecords()
End If
End Sub

Private Sub cmdAdd_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdAdd.Click
txtId.Text = "" txtBasic.Text = ""
```

```
txtName.Text = “”
End Sub

Private Sub frmdb_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
txtId.Enabled = False
con.ConnectionString =
”Provider=Microsoft.ACE.OLEDB.12.0;DataSource=C:\Users\ADMIN\Documents\Database1.accdb”
con.Open()
da = New OleDbDataAdapter(“Select * from Table1”, con) da.Fill(ds, “Table1”)
count = 0
tot = ds.Tables(0).Rows.Count
Navigaterecords()
End Sub
End Class
```

Summary:

VB.NET is a powerful and versatile programming language that combines simplicity with the capabilities of modern software development. Its integration with the .NET framework allows developers to build a wide range of applications, from desktop to web to mobile solutions. VB.NET’s object-oriented approach, rich UI capabilities, and strong support for modern programming features make it a strong choice for experienced developers.

Check your Understanding:

Question 1:

Visual Studio .NET provides which feature:

a) debugging
b) application deployment
c) syntax checking
d) All of the above

Question 2:

Which does the solution explorer not display?

a) Form Properties
b) Reference Folder
c) Form File
d) Assemble File

Question 3:

The Tick event is found only in which object?

a) Form
b) Button
c) TextBox
d) Timer

Question 4:

Which is a valid statement for declaring a variable?

a) Const Form As Integer
b) Const myForm As Integer
c) Dim Form As Integer
d) Dim myForm As Integer

Question 5:

The Items property of a ComboBox:

a) is a collection of items
b) is the same as the Items property of a ListBox
c) contains methods and properties
d) All of the above

Question 6:

Which method of a ListBox will remove just one item at a time?

a) Items.RemoveAt
b) Item.RemoveAt
c) Items.ClearAt
d) Item.ClearAt

Question 7:

Which event is activated when a RadioButton is selected?

a) Checked
b) CheckedChanged
c) Selected
d) SelectedChanged

Question 8:

Which of the following data type is not related to VB.NET?

a) Boolean
b) Single
c) Double
d) Float

Question 9:

Which is not a valid Exit statement?

a) Exit Do
b) Exit For
c) Exit Form
d) Exit Select

Question 10:

Which of the following statements is guaranteed to pass the variable numVar by value to the Sub procedure VB?

a) VB(numVar)
b) VB (ByVal numVar)
c) VB ((numVar))
d) VB (ByVal numVar As Double)

Chapter 3: JavaScript

Introduction

A Script is a segment of code that manipulates the browser and its contents in ways that is not possible with ordinary HTML or Cascading Style Sheets. By using a script in your web pages, you can gain more control of how the page looks and behaves. Dates and times can be added to the page, form elements validated before the contents are sent, browser details checked, cookies set, even simple games can be added to a web page - all with a scripting language.
The learning curve for scripting is a lot a steeper than HTML and Style Sheets. But you can learn the basics, and use scripts on your own pages without it causing you too much trouble. The scripting language covered in these pages is meant to get you started on the subject, and is not intended as an in-depth study. We're going to study the JavaScript programming language, because it is a widely-used scripting language for web pages. All the scripts in these pages have been tested with modern versions of a wide variety of browsers.

Let's start with a small code of JavaScript

```
<HTML>
<HEAD>
<TITLE>My First Script</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE = JavaScript>
document.write("Hello World")
</SCRIPT>
</BODY>
</HTML>
```

Rules for writing codes

When you're writing your scripts, you enclose them between two <SCRIPT> tags, an opening one and a closing one. The opening one should tell the browser what language the script is written in:
<SCRIPT LANGUAGE = JavaScript>
The closing Script tag is just the word SCRIPT in between two angle brackets with a forward slash:
</SCRIPT> Most of your JavaScript will go between these two tags. So what's all that "document dot write" bit?
document.write("Hello World")

Script Tag and HTML

At the moment, we have our script between the two BODY tag and it works perfectly well here.

You're not cluttering up your HTML code with lots of JavaScript.
So, cut your script out of the BODY section, and paste it into the HEAD section.

```
<HTML>
<HEAD>
<TITLE>A First Script</TITLE>
<SCRIPT LANGUAGE = JavaScript>
document.write("Hello World")
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Save your work, and then view the results in your browser. Did it make a difference? No, it did not. But rest assured that your script was dealt with before anything in the BODY section.

You can also put your scripts into HTML tags. Here's the document.write() code inserted into a Form's Button code:

```
<BODY>
<INPUT TYPE = Button VALUE = "Click Me" OnClick ="document.write('Hello World')">
</BODY>
```

We can use the Button as :-

OnClick = "document.write('Hello World')"

OnClick is an event that can be applied to buttons (amongst other things.) .

The Pop-up message box we've seen one way to display text - with the write() method of document. Another way to display text (and numbers) is with a little pop-up box. These are called Alert boxes in JavaScript, and they are ideal for nagging your users when they don't fill in your forms correctly. Here's a picture of one:

The code for an Alert box is quite simple. It's this:

```
<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  alert("Hallo Users");
}
</script>
```

The Document Object

As was mentioned, document is a part of the Document Object Model.We saw a method that can be used with document, but here's a couple more (Properties and Methods).

Properties

- bgColor
- fgColor
- title
- location
- images
- forms

Methods

- · open()
- · close()
- · write()
- · writeln()

There are quite a few more Properties and Methods you can use with document, but these are enough for us. Let's see how you can use them in your own scripts.

Start a new web page in your HTML Editor (Or use an old one, if you like - waste not, want not!) Then add this to the BODY section of your page:

```
<html>
<body>
<INPUT TYPE = Button VALUE = "Colour One" Onclick ="document.bgColor = 'Red'">
<INPUT TYPE = Button VALUE = "Colour Two" Onclick ="document.bgColor = 'Blue'">
<INPUT TYPE = Button VALUE = "Colour Three" Onclick ="document.bgColor = 'Green'">
</BODY>
</html>
```

Remember, when you're typing the OnClick code that this bit **"document.bgColor = "** is surrounded by double quotes, and this bit: **'Green'**is surrounded by single quotes. And the whole thing is a confusion of single and double quotes:
Onclick = "document.bgColor = 'Green'"
When you're done, save your work and view the results in your browser.
Click the button and see what happens.

The background color of your web page should have changed color when you clicked a button. It did this because of the bgColor poperty of the document object.

document.bgColor =
After the equals sign, you can either type a name of a color, or better use an Hexadecimal value:
document.bgColor = #FF0000
document.bgColor = #0000FF
document.bgColor = #00FF00
The fgColor property is similar to the bgColor property. Except it will change the colour of any foreground text on your page. Try it out and seehow it works. You only need to change the "b" of "bgColor" to an "f" and type some text.

The document Title Property
The Title of a web page appears at the very top of the browser window.You set it by typing some text between the two <TITLE> tags. You can use JavaScript to either get the Title or to set a new one. Here's a script that pops up the Title in an alert box. Try it out and see it in action:

```
<HTML>
<HEAD>
<TITLE>The title of this web page</TITLE>
<SCRIPT LANGUAGE = JavaScript>
alert(document.title)
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

IMAGE Property
The images property tells you if any images are loaded
Examine this HTML code for an image:

```
<BODY>
<IMG SRC = "img1.jpg" NAME = "but1">
</BODY>
```

Simple enough, because it's just standard HTML Image code.

We can turn the image into a link by surrounding it with an Anchor tag:

<BODY>

<A HREF = "page2.html">

<IMG SRC = "img1.jpg" NAME = "but1" Border = 0>

</A>

</BODY>

However, we can add a bit of JavaScript to the Anchor link.

This bit of JavaScript uses the images property of document.

Like this:

<BODY>

<A HREF = "page2.html" OnMouseOver = "document.images.but1.src=

'pointer2.jpg'">

<IMG SRC = "pointer1.jpg" NAME = "but1" Border = 0>

</A>

</BODY>

JavaScript Methods and Events

Chapter Contents

1. **Document Methods**
2. **The Navigator object**
3. **The Window Object**

Methods

Let's have a look at those document Methods. They were: **open( ), close( ),write( ), writeln( )**.

We've already done write( ), and writeln( ) is the same as write( ) except that you get a line break.

So we'll look (briefly) at open( ) and close( )

open( )

The default for document.write( ) is to write stuff in the current document window - the browser window.

You could send the HTML code with document.write( )

document.open( )

document.write("<H1>Hello</H1>")

close( )

This method is used to close the document you created with the open( ) method.

Events

Two useful document Events you can use are these:

onLoad

onUnload

These are typically used in the BODY section of the HTML code.

Here are two pieces of code that will annoy visitors to your site:

<BODY onLoad = "alert('Welcome to my web page')">

Or when they leave your site, there's this:

<BODY onUnoad = "alert('Goodbye - Have a nice day')">

The document object dealt with code between the BODY tags. The navigator object is used to reveal information about the browser your visitors are using. Because some browsers can't deal with certain things (like the document images code we wrote), the navigator object is typically used to help divert these users away from any code that they can't handle.

Two Properties of the navigator object are:

appName

userAgent

Let's use an alert box to test them out. Like this:

// document.open()

// document.write("<H1>Hello</H1>")

The two forward slashes mean "please ignore this code". Whether you decide to create a new web page, or use an old one, type this code in the HEAD section of the page:

<SCRIPT LANGUAGE = JavaScript>

alert(navigator.appName)

</SCRIPT>

Try changing the alert box to this:

<SCRIPT LANGUAGE = JavaScript>

alert("Your Browser is: " + navigator.appName)

</SCRIPT>

Screen Size

Copy this code into your editor. Place it in the HEAD section of your HTML. Save your work and test the script in your browser:

```
<Script language = JavaScript>
UserWidth = window.screen.width
UserHeight = window.screen.height
UserWidth = "Screen Width = " + UserWidth
UserHeight = " Screen Height = " + UserHeight
alert(UserWidth + UserHeight)
</Script>
```

UserWidth = window.screen.width

A variable is a little storage area. Think of it as a little cupboard or drawer.Instead of putting socks or underwear in the drawer, you're putting text or number into it. Your little storage area needs a name (else how could you tell which drawer held the socks and which your underwear?). The variable name is UserWidth. This is a name we made up ourselves. You can use just about anything you want as a variable name. (But there are a number of words JavaScript likes to keep for itself.) To put something in your drawer (variable) you type an assignment operator (the equals sign), then type whatever you want to put into the drawer (variable). If you putting text into the variable, surround the text with double quotes:

UserWidth = "This will hold a width"

If you're putting numbers into the variable, don't surround the numbers with double quotes:

UserWidth = 600

Or you can combine the two with the plus sign (+):

UserWidth = "This will hold a width of " + 600

Coding in JavaScript

You have just learnt what a variable is, and that it is a little storage area for text and numbers.

Let's explore variables a bit more.

Variables

Variables can hold two types of information: Numbers,and Strings.

We've seen how strings work (strings are text),

Number variables in JavaScript can be integers, floating point, or Boolean.An integer is a number without a decimal point.

To store one of those number types into a variable, just use the assignment operator (=). The number goes on the right hand side of the assignment operator, and your variable name goes on the left. Here's an example of how to store number types into variables:

Number1 = 3

Number2 = 3.5

Number3 = true

Number1, Number2, and Number3 are our variable names. We could have used anything we liked here. As long as we're not using the handful of words JavaScript reserves for itself, and not starting with a number, and not putting any spaces in the variable name, we're all right. Here's some valid and invalid variable names

BobsNumber = 3 (valid variable name)

Bobs Number = 3 (invalid variable name - space)

3bobsnumber = 3 (invalid variable name - starts with a number)

A good idea when declaring variables, and putting values into them, is to use the word **var** in front of them.

Like this:

var Number1 = 3

The word **var** is short for variable. It ensures that any variable with the same name elsewhere in your code doesn't overwrite it. Which is a pain, believe me. For clarity's sake, however, we'll leave var out of our declarations.

You can store the value in one variable name in another variable. Take this as an example:

A = 2

B = A

Here, the variable "A" is assigned a value of 2. The value inside "A" is then assigned to the variable "B". So what's inside "B"? Not the letter "A", but the number 2, of course. If you want to add, subtract, multiply, and divide your values, you'll need the operators. The four basic operators are these :

+ (Add)

- (Subtract)

***** (Multiply)

/ (Divide)

Actually, the last two should be the first two. That's because Multiply (*)and Divide (/) are done first when JavaScript is doing its sums. But, anyway, here's the operators in action. Try these scripts out in a web page, and see how you get on:

```
<SCRIPT Language = JavaScript>
A = 2
B = A
C = B + A
alert(C)
</SCRIPT>
<SCRIPT Language = JavaScript>
A = 2
B = 4
C = B * A
alert(C)
</SCRIPT>
```

Next one:

<SCRIPT Language = JavaScript>
A = 2
B = 4
B = B - A
alert(B)
</SCRIPT>

<SCRIPT Language = JavaScript>
A = 2
B = 4
B = B / A
alert(B)
</SCRIPT>
<SCRIPT Language = JavaScript>
A = 2
B = 4
C = 6
D = (B * C) / A
alert(D)
</SCRIPT>

So, how did you get on? The brackets in the last script are for sectioning off code that you want handled separately. Anything in brackets gets dealt with first. So B is multiplied by C first, then the total is divided by A. Adding numbers in textboxes We'll design a web page with three text boxes and a command button. When the command button is clicked, any numbers in the first two boxes will be added together. The answer will appear in the third text box. Here's what your form will look like when it's finished (The button doesn't do anything yet, if you click it):

Number One: Number Two

Total:

So you type a number into the Number One text box, and type a number into the Number Two text box. When the "Add Numbers" button is clicked, the total will appear in the "Total" box. Let's design the HTML Form first (You can copy and paste this).

<FORM NAME = frmOne>
Number One: <INPUT TYPE = Text NAME = txtFirstNumber SIZE = 5 value ="">
Number Two: <INPUT TYPE = Text NAME = txtSecondNumber SIZE = 5 value ="">
<P>
Total: <INPUT TYPE = Text NAME = txtThirdNumber SIZE = 5 value = "">

```
<P>
<Input Type = Button NAME = b1 VALUE = “Add Numbers” onClick = calculate()>
</FORM>
```

When we’re writing the code, we need to tell the browser which form we’re talking about. Our form is called “frmOne”. For the very same reason, all three text boxes have names

txtFirstNumber

txtSecondNumber

txtThirdNumber

Note, too, that all three text boxes have a VALUE attribute. The button has an onClick event that is crucial to the operation:

onClick = calculate( )

Remember that name - calculate( ). We’re going to be seeing that in the SCRIPT section. So, in the HEAD section of your web page, write the following script:

```
<SCRIPT language = JavaScript>
function calculate()
        {
                A = document.frmOne.txtFirstNumber.value
                B = document.frmOne.txtSecondNumber.value
                C = (A + B)
                document.frmOne.txtThirdNumber.value = C
        }
</SCRIPT>
```

A function is a way to group together a section of code. This function will calculate the two numbers from our text boxes, and pop the answer in the third text box.

The syntax for a JavaScript function is this:

```
function FunctionName()
        {

        }
```

Our Function Name was calculate (), which was the one from the command button. When the button is clicked, the browser will try to find a function called calculate (). Note that when setting up a function you need a lowercase “f” and not a capital “F”. After the name of your function, type a pair of round brackets. You can put something called an argument in between the round brackets. But our function has no arguments, so we just leave them empty. After the round brackets, type a space, then type a left curly bracket { . The code for the function itself goes next (on a new line), and then you type a right curly bracket }.

Now the function be like:

```
function calculate()
        {
        A = document.frmOne.txtFirstNumber.value
        B = document.frmOne.txtSecondNumber.value
        C = (A + B)
        document.frmOne.txtThirdNumber.value = C
        }
```

The first line of the code is this:

A = document.frmOne.txtFirstNumber.value. The value we're putting in the variable called "A" is coming from the first text box. The part on the right of the equals sign (=) might look a bit long and complicated, but all we're doing is telling the browser which text box we're referring to. We're saying this:

"In the document there is a form called frmOne. This form has a text box called txtFirstNumber. Whatever value has been typed into this text box, store it in the variable called A."

We do the same for the second text box, storing its value in the variable called B. The third line of code adds together the two values A and B. The total is then stored in the variable called C. Once we have our answer in C, we then need to store it in the third text box on the form. So we have to say "Assign the value in the variable called C to the text box called txtThirdNumber."

document.frmOne.txtThirdNumber.value = C

The problem lies in those text boxes. Not surprisingly, what you get from a text box is text. So when we typed in the number 2, the program thought it was text. When we told the program to add the numbers we did it like this:

C = (A + B)

Because the program thinks there is text in the variables A and B, it will join them together instead of adding them up. When you tried to add up 2 + 2, you got 22 and not 4. So what's the solution?

One solution is to force the program to recognise the text as a number. We can use the JavaScript function eval(). Like this:

A = eval(document.frmOne.txtFirstNumber.value)

B = eval(document.frmOne.txtSecondNumber.value)

In other words, surround your document code with the round brackets of the eval() function. Just the first two, A and B.

Another, neater, solution is to use JavaScript's in-built Number() function:

A = document.frmOne.txtFirstNumber.value

B = document.frmOne.txtSecondNumber.value

A = Number(A)

B = Number(B)

Make sure Number has a capital “N”, and not a lowercase “n”. This will force JavaScript to recognize the value from a text box as a number. Using this method, however, will get you a “NaN” in your answer text box, if your user enters a string (two + two instead of 2 + 2. NaN stands for Not a Number). So you need to do some error checking for this, which we’ll get to later.

Control Flow

If Statements

If..Else Statements

To write more complex programs in any language, you need something called control flow. This is just controlling the direction of the program so that it’s not so linear. Instead of reading your code line by line, and every single line, you can skip lines of code depending on certain conditions. For example, if you’re using an OK and Cancel alert box and the user clicks OK, you can execute one piece of code. If, however, the user clicks Cancel, you can have a different segment of code executed.

This is called Control Flow. Don’t worry if none of that makes sense - it will become clearer when you follow the examples.

If Statement

If statements are the backbone of programming. Some people even go so far as to say that in a sense, all programming is a series of If statements. To get you started on just what an If statement is, type this code into the HEAD section of a HTML page, then test it out:

```
<SCRIPT language = JavaScript>
ConfirmStatus = confirm(“Format Your Hard Drive?”)
        if (ConfirmStatus == true)
         {
        alert(“Formatting Hard Drive Now “)
        }
</SCRIPT>
```

The first line of code is what gives you a new type of message box – confirm.

The programme then waits until you click either the OK button or the Cancel button.

ConfirmStatus = confirm(“Format Your Hard Drive?”)

Whichever button you click, the answer will be stored in the variable ConfirmStatus. The answer, in this case, is a Boolean value: it’s either True or False. If you click OK on a confirm box, the answer will be true; if you click Cancel on a confirm box, the answer will be false. What you as a programmer have to do next is to check which button the user clicked. Was OK clicked or was Cancel clicked? If the OK button was clicked, you’ll want to do one thing; if the Cancel button was clicked, you’ll want to do another thing. So you need to test what is inside the variable ConfirmStatus. You do the testing with an if statement.

Our if statement was as simple as if statement get. It was this:

```
if (ConfirmStatus == true)
        {
        alert("Formatting Hard Drive Now ")
        }
```

Note the format for an if statement:

```
if (condition to test)
        {
        Your statement here


        }
```

Points to be noted:

1. First you type the word if (in lowercase letters)
2. Next, type a space, then your condition inside round brackets
3. Type another space, then type a left curly bracket {
4. On a new line, type your statement (what you want to happen)
5. Finish with a right curly bracket }

If you're checking what's inside a variable, JavaScript requires a double equals sign (==). It means "has a value of". In our condition to test, we wanted to check whether the variable ConfirmStatus was true or false. So we did this:

ConfirmStatus == true

We're saying "If ConfirmStatus has a value of true" then execute the next line of code. The next line of code was an alert box:

alert("Formatting Hard Drive Now ")

The main point to bear in mind here is that you would only get the alert message IF you clicked the OK button. Because then the variable ConfirmStatus would be true. But what if the user clicked Cancel? What happens then?

If the Cancel button is clicked, the ConfirmStatus variable will be false. In that case, our alert box statement doesn't get executed - the program will just ignore that line and move on to the next line of code.

We can, however, add another statement to check for a false condition. Change your code to this:

```
<SCRIPT language = JavaScript>
ConfirmStatus = confirm("Format Your Hard Drive?")
        if (ConfirmStatus == true)
        {
        alert("Formatting Hard Drive Now ")
        }
        if (ConfirmStatus == false)
        {
        alert("Tough! Formatting Hard Drive anyway")
        }
</SCRIPT>
```

All we've done is add a second if statement to check for a false condition. If ConfirmStatus is false (the Cancel button was clicked), then the second alert box is displayed.

All you're saying with if statement is this:

"If it's true, do one thing; if it's false do another thing"

You can add as many if statements as you need, and you can nest one if statement inside another. For example, you could add another confirm box inside the second statement, and nest another if condition. Like this:

```
if (ConfirmStatus == false)
        {
        nested = confirm("Ah, go on! Let me format your hard
        drive")
if (nested == true)
        {
        alert("Thanks!")
        }
if (nested == false)
        {
        alert("Spoil Sport!")
        }
        }
```

When you're nesting if statement like that, things can get a little complicated. We saw in the last part of this Control Flow section that to test two conditions (true and false) two if statements were needed. But you don't have to use separate if statements. Instead, you can use the else word.

This is ideal for Boolean values. Here's our code again, this time with an

if … else statement.

```
ConfirmStatus = confirm("Format Your Hard Drive?")
if (ConfirmStatus == true)
        {
        alert("Formatting Hard Drive Now")
        }
else
        {
        alert("Tough! Formatting Hard Drive anyway")
        }
```

The first if statement is exactly the same. Now, though, instead of our second if statement to test for a false condition, we've used the word else. The format for an if … else statement is this:

```
if (condition to test)
        {
        Statement 1
        }
else
        {
        Statement 2
        }
```

So if the first statement isn't true, the code beneath else gets executed.

if … else if

If you need multiple if statements, then if … else if is the one to use. In the code below, we'll check to see what number is in a text box. Depending on which number was typed in, we'll display an appropriate (or inappropriate) message. First, you're going to need a few more symbols to work with. You've already met the double equals symbol (==), and know that this means "has a value of". Here's some more.

> Greater than

< Less than

<= Less than or equal to

>= Greater than or equal to

!= Not equal to

&& Evaluation And || Evaluation Or

We'll see how these operators work with our next little program.

So design an HTML page with the following form on it.

<FORM **NAME = "AgeTest**">

Type in your age, then click the button:

<INPUT TYPE = Text **NAME = txtAge** Size = 15>

<INPUT TYPE = Button NAME = "b1" VALUE = "Age Tester" **onClick = message()**>

</FORM>

Once you've typed the HTML code for the form, type this JavaScript code in the HEAD section of your page:

```
<SCRIPT language = Javascript>
function message()
        {
        age = document.AgeTest.txtAge.value
        if (age < 18)
                {
                alert("How's school these days?")
                }
        else if (age > 18)
                {
                alert("It's tough being an adult")
                }
        }
</SCRIPT>
```

We had another onClick event for our command button, which called the **message( )** function. When the message( ) is called, it first stores the value in the text box into a variable called **age**.

age = document.AgeTest.txtAge.value

Next, we need to test what is inside the age variable. We started the test with an if statement:

```
if (age < 18)
        {
        alert("How's school these days?")
        }
```

Notice one of our new symbols in the "condition to test":

age < 18

If the value stored in the variable **age** is less than 18, then the condition will be true. If the condition is true then our statement will get executed. The statement was an alert box:

alert("How's school these days?")

Next, because we're checking for multiple values, we have the else if condition. Ours was this:

```
if (age < 18)
        {
        alert("How's school these days?")
        }
        else if(age > 18)
        {
        alert("It's tough being an adult")
        }
```

The format for the else if part is this:

```
if (condition to test)
        {
        statement 1
        }
else if(condition to test)
        {
        statement 1
        }
```

It's just another if statement with the word else in front of it (type a space after the word else). You can add as many else if conditions as you want.

```
if (age < 18)
        {
        alert("How's school these days?")
        }
else if(age > 18)
        {
        alert("It's tough being an adult")
        }
else
        {
        alert("Please try again")
        }
```

Notice in that last code that the second age test uses the Greater Than symbol (>). Try entering the number 18 in your text box, and then click your button. What happened? If you haven't added the final else statement, then nothing will happen. This is because we're testing for Less Than 18 in the first if statement, and then testing for Greater Than 18 in the first else statement. We're not testing for exactly 18. So none of our statements will be true.

What we need is another of our symbols. Either **>=** (Greater Than or Equal to) or **<=** (Less Than or Equal to).

So, change your code.

Take out either the Less Than symbol (**<**) or the Greater Than symbol (**>**). In it's place, insert either **<=** or **>=** to see what happens. Play about with the symbols to see how they work.

What happens if you put both in? Like this:

```
if (age <= 18)
        {
        alert("How's school these days?")
        }
else if(age >= 18)
        {
        alert("It's tough being an adult")
        }
else    {
        alert("Please try again")

        }
```

AND and OR

These two operators will return a Boolean value. You use them when you want to test two or more conditions. For example, if you wanted to test if someone was over 65 and had a buss pass, use AND; if you wanted to test if someone was over 65 or had a buss pass, use OR. Like this:

```
if (Age >= 65 && BusPass == false)
         {
        alert("Pensioners - Get your free bus pass now!")
        }
```

If BOTH conditions are true then the IF Statement is true. If just one of them is false, then the entire IF Statement is false. Note the format (and where the round brackets are):

```
if (condition1 && condition2)
         {
        Code if true here
        }
```

Contrast that with the OR operator:

```
if (Age >= 65 || BusPass == false)
        {
        alert("Pensioners - Get your free bus pass now!")
        }
```

This time, if just one of your conditions is true then the entire IF statement is true. They both need to be false for the entire IF Statement to be false. The format (syntax) is the same, except for the two pipe characters (||) in place of the two ampersands (&&).

Loops in JavaScript

1. For Loop
2. While and Do Loops
3. Break and Switch Statements in JavaScript

What's loop ?

A loop is something that goes round and round. Except a programming loop will go round and round until you tell it to stop. You also need to tell the program two other things - where to start your loop, and what to do after it's finished one lap (known as the update expression). There are three types of loops in JavaScript: **for** loops, **while** loops, and **do … while** loops. We'll start with the most common type - the for loop.

For Loops

Here's a JavaScript for loop in a little script. Type, or copy and paste, it into the HEAD section of web page (along with the script tags) and test it out.

```
counter = 0
for(start = 1; start < 10; start++) {
counter = counter + 1
document.write("start = " + start + " counter = " + counter + "<BR>")
}
```

How did you get on? You should have this printed on your page:

start = 1 counter = 1
start = 2 counter = 2
start = 3 counter = 3
start = 4 counter = 4
start = 5 counter = 5
start = 6 counter = 6
start = 7 counter = 7
start = 8 counter = 8
start = 9 counter = 9
start = 10 counter = 10

The format for a for loop is this:

```
for (start value; end value; update expression)
{
}
```

The first thing you need to do is type the name of the loop you're using, in this case **for** (in lower case letters). In between round brackets, you then type your three conditions:

Start Value

The first condition is where you tell JavaScript the initial value of your loop. In other words, start the loop at what number? We used this:

start = 1

We're assigning a value of 1 to a variable called **start**. Like all variables, you can make up your own name. A popular name for the initial variable is the letter.

```
start = 1
for(start; start < 11; start++)
        {
        The result is the same - the start number for the loop is 1
        End Value
```

Next, you have to tell JavaScript when to end your loop. This can be a number, a Boolean value, a string, etc. Here, we're telling JavaScript tobail out of the loop when the value of the variable start is Less Than 11.

Update Expression

Loops need a way of getting the next number in a series. If the loop couldn't update the starting value, it would be stuck on the starting value.If we didn't update our start value, our loop would get stuck on 1. In other words, you need to tell the loop how it is to go round and round. We used this:

start++

In the java programming language the double plus symbol (++) means increment (increase the value by one). It's just a short way of saying this:

start = start + 1

You can go down by one (decrement) by using the double minus symbol(—), but we won't go into that.

So our whole loop reads **"Starting at a value of 1, keep going round and round until the start value is less than 11. Increase the starting value by one each time round the loop."**

Every time the loop goes round, the code between our two curly brackets { } gets executed:

```
counter = counter + 1
document.write("start = " + start + " counter = " + counter + "<BR>")
```

Notice that we're just incrementing the counter variable by 1 each time round the loop, exactly the same as what we're doing with the start variable. So we could have put this instead:

```
counter++
```

The effect would be the same. As an experiment, try setting the value of the counter to 11 outside the loop (it's currently **counter = 0**). Then inside the loop, use **counter- -** (the double minus sign).

While Loops

Instead of using a for loop, you have the option to use a while loop. The structure of a while loop is more simple than a for loop, because you're only evaluating the one condition. The loop goes round and round while the condition is true. When the condition is false, the program breaks out of the while loop. Here's the syntax for a while loop:

```
while (condition)
        {
        statement
        }
```

And here's some code to try. All it does is increment a variable called counter:

```
counter = 1
while (counter < 11)
        {
        document.write(" counter = " + counter + "<BR>")
        counter++
        }
```

The condition to test for is counter < 11. Each time round the while loop, that condition is checked. If counter is less than eleven then the condition is true. When counter is greater than eleven then the condition is false. A while loop will stop going round and round when a condition is false. If you use a while loop, be careful that you don't create an infinite loop. You'd create one of these if you didn't provide a way for you condition to be evaluated as true. We can create an infinite loop with the while loop above. All we have to do is comment out the line where the counter variable is incremented. Like this:

```
counter = 1
while (counter < 11)
        {
        document.write(" counter = " + counter + "<BR>")
        //counter++
        }
```

Notice the two forward slashes before counter++. This line will now beignored. Because the loop is going round and round while counter is lessthan 11, the loop will never end - counter will always be 1.Here's the times table program again, only now we've used a while loop instead of a for loop (the lines that write the result to the text area have been left out):

```
start = 1
end = 1
times = 2
answer = 0
while (end < 11)
        {
        answer = start * times
        start++
        end++
        }
```

The while loop calculates the 2 times tables, up to a ten times 2. Can you see what's going on? Make sure you understand the code. If not, it's a good idea to go back and read this section again. You won't be considered a failure.

Do ... While loops

This type is loop is almost identical to the while loop, except that the condition comes at the end:

do

statement

while (condition)

The difference is that your statement gets executed at least once. In a normal while loop, the condition could be met before your statement gets executed. Don't worry too much about do while loops.

The Break Statement

There are times when you need to break out of a loop before the whole thing gets executed. Or, you want to break out of the loop because of an error your user made. In which case, you can use the break statement. Fortunately, this involves nothing more than typing the word break. Here's some not very useful code that demonstrates the use of the break statement:

```
TeacherInterrupts = true
counter = 1
while (counter < 11)
        {
        document.write(" counter = " + counter + "<BR>")
        if (TeacherInterrupts == true)
        break
        counter++
        }
```

Switch Statements

These types of statements can be very useful. A switch statement is used when you want to choose just one of several possible outcomes. Our age tester was a good example. We had several ages groups that we were testing for, and a person could be in only one of them. We used a series of if statements to test which one our user belonged to.

A better way to do the testing is with a switch statement.

Here, the value in Age is coming from a drop-down combo box on a form called **frmOne**:

```
Age = document.frmOne.cmbAge.value
switch (Age)
        {
        case "1":
        alert("still at school")
        break
        case "2":
        alert("still young")
        break
        case "3":
        alert("start lying")
```

```
break
case “4”:
alert(“start saving”)
break
default:
alert(“undetermined”)
}
```

The switch statement starts with the word switch. What you are testing for comes next, in round brackets. In the code above, we’re testing what is inside the variable Age. Next, type a curly bracket {, then on a new line,you set a series of values that could be in the variable Age. Each value that could be in your variable is preceded by the word case. A colon comes after the value. If the value is going to be text, then use double quotes; if the value is going to be a number, don’t use the quotes:

case “1”:

All we’re saying is, “If it is the case that the variable Age holds the value “1”, then we’ve found a winner.” Once a match has been found, any code after the colon (:) will be executed. For us, that was an alert box: **alert(“still at school”)**

You have to tell JavaScript to break out of the switch statement. Just use the break statement, otherwise the next case down will get executed as well:

```
case “1”:
alert(“still at school”)
break
```

It was mentioned that the value in Age was coming from a drop-down box on a form. Here’s the HTML code for that:

```
<FORM NAME = frmOne>
<select name = cmbAge>
<option value = 1>5 to 16</option>
<option value = 2>17 to 30</option>
<option value = 3>31 to 45</option>
<option value = 4>46+</option>
</select>
</Form>
```

We have four options in the drop down box, each with a value 1 to 4. It’s this value that is going into the variable Age. Because the value from a HTML drop-down box will be a string (text) and not a number, each value for the case was surrounded by double quotes.

Switch statements are an ideal way to check the value returned from a drop down box on a form. Instead of having an alert box in the case statement, you could direct your users to another web page.

Like this:

```
page = document.frmOne.cmbAge.value
switch (page) {
case “1”:
document.URL = “page1.html”
break
case “2”:
```

```
document.URL = “page2.html”
break
case “3”:
document.URL = “page3.html”
break
case “4”:
document.URL = “page4.html”
break
default:
alert(“An error occurred so we’re staying here”)
}
```

The default is what you get when none of your case statements are true.

Arrays in Javascript

We will learn:

1. Arrays
2. Array Methods

Arrays

You know what a variable is - just a little drawer in your computer’s memory. Instead of holding socks or underwear, variables hold text and numbers. Except, a variable will hold only one value. You can store a single number in a variable, or a single string. An array is a variable that holds more than one number, or more than one string. Let’s make a start on explaining how arrays work.

Setting up an Array

To set up an array in JavaScript you have to create an array object. This, for us, involves nothing more complicated that using the word new. Like this:

Seasons = new Array(3)

The above code would set up an array that could hold four different values in it (arrays start at zero). To put something into each “slot” in the array, you’d just do this:

Seasons[0] = “Winter”
Seasons[1] = “Spring”
Seasons[2] = “Summer”
Seasons[3] = “Autumn”

Our array is called Seasons. In between round brackets, we had an upperlimit for the array - 3. (The number in round brackets is called the Index).

When we’re assigning values to each position in the array, we just type the name of the array, and then one of the numbers in square brackets. After typing the assignment operator (=), we type the value we want to go into that “slot” of the array. Another way to set up an array is to have all the values inside the round brackets of the word Array. Like this:

Seasons = new Array(“Winter”, “Spring”, “Summer”, “Autumn”)

We still have an array with four values in it, but we’ve fixed the values inside of the round brackets.

To get at the value inside an array, just use its index number. Like this:

So putting that altogether in a script would give us this:

<SCRIPT language=JavaScript>

Seasons = new Array(3)

Seasons[0] = “Winter”

Seasons[1] = “Spring”

Seasons[2] = “Summer”

Seasons[3] = “Autumn”

alert(Seasons[2])

</SCRIPT>

Test out the script in a web page and see how it works. Change the value of the index from 2 to one of the other numbers and see what happens. Change the value of the index to a number higher than 3 and see what happens.

Array Methods

Here’s another thing you can try. Add this line just before the alert box (on a line of its own):

Seasons.sort()

Now type in 3 as the index number in your alert box. The alert box will now display “Winter”. Despite the fact that our array has Winter at position 0. The reason why it does this is because we used a method for on our array - sort(). Because Array is now an object, we only need the array name followed by a full stop. After the full stop type the method you want to use.

The sort() method will sort your array alphabetically. You can sort your array in reverse order with this:

Seasons.reverse()

In which case, the value in the last position will become the first value, the second last position will become the second value, etc. We can use another method of arrays to see this more clearly. Add this line after the sort() method:

Seasons.join()

So you should have this as your code:

Seasons = new Array(3)

Seasons[0] = “Winter”

Seasons[1] = “Spring”

Seasons[2] = “Summer”

Seasons[3] = “Autumn”

Seasons.sort()

Seasons.join()

alert(Seasons)

The index number of an array

Arrays really come in to their own when you access their index numbers.You can use a variable in place of the index number. Then, if you increment the variable, you can access the elements in your array. As an example, try this script in a web page. For the web page, create a form called frmOne. Put a button on the form, and a text box called txtSeason. If you prefer, copy and paste this one:

```
<FORM name = frmOne>
<INPUT Type = text name = txtData>
<INPUT Type = button value = " Seasons " onClick = GetSeasons()>
</FORM>
```

When you've got the form in place, add this script to the HEAD section of your code:

```
<SCRIPT language=JavaScript>
inc = 0
function GetSeasons () {
Seasons = new Array(3)
Seasons[0] = "Winter"
Seasons[1] = "Spring"
Seasons[2] = "Summer"
Seasons[3] = "Autumn"
document.frmOne.txtData.value = Seasons[inc]
inc++
}
</SCRIPT>
```

We're setting up the array in exactly the same way. Now, though, we'recalling a function with the onClick event of the button in the Form:

onClick = GetSeasons()

The function will set up the array. It also puts a value in our text box. But note what the value is - it's one of our elements in the array. But which element is it?

document.frmOne.txtData.value = Seasons[inc]

The inc is a variable. It was set up outside the function, making it global (all your functions can see a global variable):

inc = 0

So the first value put into our text box is this:

document.frmOne.txtData.value = Seasons[0]

In the final line of the code, we then increment the variable:

inc++

So every time the button is clicked, 1 will get added to the inc variable. The next time you click the button, the code is really this:

document.frmOne.txtData.value = Seasons[1]

When the value in the inc variable goes above 3, you'll get this in the text box:

undefined

That's because we don't have a position number 4 in our array. You can reset the inc variable like this:

if (inc > 3) inc = 0

That should prevent "undefined" from displaying in the text box. Arrays are used a lot inside loops. We're going to see how that works now by creating a lottery number generator. But before continuing you should ensure that you have a good understanding of how for loops work. If not, go back and revise.

Arrays and Loops

If you want to assign a value to a position in an array, you can just do this:

Array[9] = "Hello World"

To display the value, just access its index number again:

alert(Array[9])

The fact that you can access the index number of an array to assign values makes them very handy in loops. To start our lottery number generator (UK lottery), we're going to create an array of 49 numbers. Number 1 will be at position 1 in the array, number 2 at position 2, etc. We could do this for all 49 numbers:

```
Lottery = new Array(49)
Lottery[1] = 1
Lottery[2] = 2
Lottery[3] = 3
Lottery[4] = 4
```

But we'd have a laborious job typing all the array positions out, and all the values for them. Instead, we can use a loop to fill our array with the numbers 1 to 49. Here's the loop we can use:

```
for (i = 1; i < 50; i++)
        {
        Lottery[i] = i
        }
```

Isn't that easier? But how does it work? Well, first we set a start position for our for loop by assigning 1 to the variable called i (i = 1). The end position for our loop is when the variable i is not less than 50 (i < 50).Every time round the loop 1 gets added to i (i++).

Then we assign the value of i to a position in our array:

Lottery[i] = i

At first, the value in i is 1. Which gives us this in reality:

Lottery[1] = 1

The next time round the loop, 1 gets add to the variable i. So we'd then have this:

Lottery[2] = 2

By the time the loop ends, all 49 positions in our array would be filled withthe numbers 1 to 49. And that's all there is to it - a couple of lines of codehas saved us reams of typing! To demonstrate that it does indeed work, create this form on a new web page:

```
<Form name = frmOne>
<textarea name = taAll rows = 10 cols = 15></textarea>
<INPUT Type = button Value = " Get Lottery Numbers" onClick = getNumbers()>
</form>
```

Make sure your form has the NAME frmOne, and that your TEXTAREA has the NAME taAll. For the button, you're adding a onClick event which calls the function getNumbers().

When you have your form in place, add this script to the HEAD section of your web page:

```
<SCRIPT Language = JavaScript>
function getNumbers()
        {
        TA = document.frmOne.taAll
        lottery = new Array(49)
        for (i = 1; i < 50; i++)
         {
        lottery[i] = i
        TA.value = TA.value + lottery[i] + "\n"
        }
        }
</SCRIPT>
```

To see what your programme should do, click this link (opens in a new window):

Click here for the Script in action We've started the script with a function:

```
function getNumbers()
        {
        }
```

The first line inside the function is just assigning the text area on our form to a variable call TA. This will make things easier for us, and we won't have to keep typing it all out when we want to add things to the text area:

```
TA = document.frmOne.taAll
```

On the next line, we set up our array:

```
lottery = new Array(49)
```

Next comes our for loop:

```
for (i = 1; i < 50; i++) {
lottery[i] = i
TA.value = TA.value + lottery[i] + "\n"
}
```

Notice the line that has been added to the for loop:

```
TA.value = TA.value + lottery[i] + "\n"
```

All that line is doing is adding the values in our array to the text area. We want to keep whatever is inside the text area, so we have to say:

TA.value = TA.value +

Then we add the value from the array, with a new line character at the end:

lottery[i] + "\n"

When you've finished adding the code to the HEAD section, test it out.

When you click the button, the numbers 1 to 49 will be added to your text area.

Events in JavaScript

1. onMouseDown Event
2. onClick Event
3. onDblClick
4. onKeyDown
5. onMouseOver
6. onBlur
7. onSubmit

onMouseDown

What we'll do in the pages that follow is to detect the position of the mouse pointer when the user clicks on the screen. To do that, you need to find the X coordinate and the Y coordinate.

· **The X coordinate tells you how far left of the screen you are**

· **The Y coordinate tells you how far down the screen you are.**

Both Netscape and Internet Explorer use the screenX and screen property. If you're sure that your users will all have Internet Explorer, then the task is fairly easy. Here's the script:

```
<HEAD>
<TITLE>XY Coordinates</TITLE>
<SCRIPT Language = javascript>
function XYpos()
        {
        xPos = event.screenX
        yPos = event.screenY
        alert(xPos + " left " + yPos + " down")
        }
</Script>
</HEAD>
<BODY onMouseDown = XYpos()>
```

The BODY section of the HTML is where we put the event handler onMouseDown. Now, every time the mouse is clicked anywhere in the Body of the web page, the function XYpos() gets called. Inside the function, the property screenX and the property screenY (which are both properties of event) are put inside two variables:

xPos = event.screenX

yPos = event.screenY

So xPos will hold how far left the mouse pointer is when the page is clicked; and yPos will hold far down the mouse pointer is when the page is clicked.

Both coordinates are then displayed in an alert box:

```
alert(xPos + " left " + yPos + " down")
```

If you have Netscape, however, the code will point blank refuse to work!If we add our Browser detector code to the script above, though, we can get Netscape to display an error message. Try changing you code to this:

```
<Script language = javascript>
Browser = navigator.appName
Net = Browser.indexOf("Netscape")
Micro = Browser.indexOf("Microsoft")
Netscape = false
IE = false
if(Net >= 0)
        {
        Netscape = true
        }
if(Micro >= 0)
        {
        IE = true
        }
function XYpos()
 {
if (IE == true)
        {
        xPos = event.screenX
        yPos = event.screenY
        alert(xPos + " left " + yPos + " down")
        }
else if (Netscape == true)
{
alert("Script won't work: " + "\n" + "You're using Netscape")
}
}
</script>
```

After the browser detection code, we've added our XYpos() function. This time, we've put some if statements in it. If the user has Internet Explorer, we're all right; if the user has Netscape, display the message:

```
function XYpos()
{
if (IE == true)
        {
        xPos = event.screenX
```

```
        yPos = event.screenY
        alert(xPos + " left " + yPos + " down")
        }
else if (Netscape == true)
        {
        alert("Script only works with Internet Explorer - sorry!")
        }
}
```

So Internet Explorer's way to get the coordinates of the mouse pointer is not too difficult. The reason why the script is so long is simply because Netscape refuses to cooperate. We therefore have to add browser detection code and do one thing for Internet Explorer and do another thing for Netscape. If you want to detect the mousedown event with Netscape then the process is quite complicated. We won't go into here.

onClick

We've already used this event quite a lot. But you can use it with buttons, images, links, radio buttons, checkboxes. Here's a simple onClick event used with an image:

<IMG SRC = red.jpg onClick = "alert('No stealing the images!')">

onDblClick

Same as above, really. Try inserting Dbl into onClick and see what happens.

onKeyDown

This events fires when a key on your keyboard is pressed. It applies to buttons, textboxes, text areas, and links. If you had a search box on a form, for example, you might want to activate the search when the user presses the Return/Enter key on the keyboard. For Netscape users, though, you need to use this:

window.captureEvents(Event.KEYPRESS)

And that's only the start of your problems! An easier example of the KeyDown event that works in both browsers is this:

<INPUT type = text onKeyDown = "alert('Key pressed')">

onMouseOver

You've seen this in action already when we wrote the script for the"dancing hand". Associated events are onMouseOut and onMouseDown.They are mainly used for links and images. A typical use of onMouseOver is to swap images around when creating a rollover effect.

Onblur

This event takes places when objects lose focus. If you click inside one text box then click outside it the textbox has lost focus. That's when the onBlur event fires.

<INPUT TYPE = text onBlur = "alert('Lost focus')">

When you move away from the textbox, you should see the alert box. You can write code to check if something like an email address is correct. If not, you can reset the text box to a blank string.

onSubmit

This is a very useful event that can be used for data validation on a form. You can use this event with a Submit button. The details on the form can then be checked for errors. If you catch any errors, you can then stop the form's details from being submitted. Here's an example. Note the use of the word **return**. This is set to a Boolean value. If return is false then the form doesn't get submitted; if it's true then the form is submitted.

```
<FORM name = f1 onSubmit = "return Validate()">
<INPUT Type = Submit Value = " Send ">
</FORM>
```

The form is not much good because it only has a submit button. But notice where the onSubmit event is - in the FORM tag, and not with the Submit button.

Here's the Validate() function. It doesn't do any checking. All it does is to display an alert message.

```
function Validate()
        {
        Details = false
        if (Details == false)
                {
                alert("errors detected")
                return false
                }
        if (Details == true)
                {
        alert("Form Submitted")
        return true
                }
        }
```

In the function, we've set a variable (Details) to false. This is just for testing purposes, and means the user has filled out the form incorrectly. Look at the two return values, though. If the Details are false then return gets set to false, and the form isn't submitted; if the Details are true then return is true, and the form is submitted.

If you want, add a Method and an Action to your Form. Like this one:

```
<form name = f1 onSubmit = "return Validate()"
Method = post Action=Mailto:MyEmailAddress@MyISP.com Enctype="text/plain">
```

(The Enctype will send the details in plain text that's much easier to read.)

Test out the code on a web page. Change the Details variable to true and click the button.

String Manipulation

1. Substring
2. Split
3. Join

Substring & Split

We're going to learn about substring() and split() first. To get the hang of these functions, we'll create a simple little game. Off we go. A Name Swapper Game The game we'll create is this: A name is entered in a text box. When a button is clicked, a function will swap the first two letters of the surname with the first two letters of the first name, and vice versa.

So, create a web page with a button and a text box on it, just like above. Put the textbox and button inside of FORM tags. Call the form f1 and the text box t1. Add an onClick() event to the button. The onClick event will call our function:

onClick = jumble()

function jumble()

{

}

The first thing we need to do in our function is get the name from the text box.

function jumble()

{

fname = document.f1.t1.value

}

split()

Next, we need to split the name entered into a first name and a second name. We can use the split() function for that:

function jumble()

{

fname = document.f1.t1.value

SplitName = fname.**split(" ")**

}

The split() function splits text and puts the pieces into an array, with each piece taking up one slot in the array.

The syntax for the split() function is this:

String.split([separator])

String is the text you want to split. You then type a full stop followed by split(). Inside the round brackets you type the character that you want the function to use when it's doing the splitting (the separator). In our function we typed a space surrounded by double quotes:

SplitName = fname.split(“ “)

So we’re telling the function to look for a single space in the variable fname. When it finds a single space, use that to add an item to the array. When it finishes splitting, our variable SplitName will be an array. You can access all the “pieces” in the array by referring to the index number. You can do it very simply, like this:

SplitName[0]

SplitName[1]

Or you can use a loop, like this:

```
for (num = 0; num < SplitName.length; num++)
 {
document.write(SplitName[num])
}
```

The length of SplitName is how many slots in the array - the length of the array, in other words.

We’ll do it the simple way, because we know we only want to manipulate a first name and a second name.

```
fname = document.f1.t1.value
sname = fname.split(“ “)
first = sname[0]
second = sname[1]
```

So first will now hold the first name from the text box, and second will hold the surname from the text box.

Join

Before we go any further, a function related to split() is join(). The join function will join together strings in an array. The syntax is this: **arrayname.join(separator)**

Here’s an example of how to use it:

```
aryText = new Array(“This”, “is”, “some”, “text”)
t1 = artText.join()
t2 = artText.join(“ “)
t3 = artText.join(“ - “)
```

The empty round brackets of t1 gets you a comma as a separator (the default); the t2 puts in spaces as a separator; the t3 puts in a minus sign as a separator. So, the three variables will now be this:

t1 = This, is, some, text

t2 = This is some text

t3 = This-is-some-text

But back to our little program. We have just got the first and surnames and put them into variables. We’ll do them one at a time. Let’s chop the first name up.

substring()

We need to grab the first two characters of our first name, and separate them from the rest of the name. We can use the function substring() to do this.

first1 = first.**substring(0, 2)**

first2 = first.**substring(2, first.length)**

The syntax for substring() is this:

stringname.substring(indexA, indexB)

After typing the string you want to manipulate, you type a full stop.

first1 = first.substring(0, 2)

So we're saying, "**Search the variable called first. Start the search at position zero (zero is the first character of the string). End the search at position 2.**" The function will then strip the characters specified and put them into the variable first1.

first2 = first.substring(2, first.length)

Now, we're starting the search from position 2. We're ending at the length of the variable first. Again, if we started with the first name of "Bill", So if we did this:

first1 = first.substring(2, 0)

the function substring() would start the search on the right hand side, at character zero. It would then grab characters up to indexA, which is set at two above. In other words, it would grab two characters starting from the right. We can do the same

for the surname:

second1 = second.substring(0, 2)

second2 = second.substring(2, second.length)

The only thing left to do now is joined the jumbled pieces together and

display the result in the text box:

fname = second1 + first2 + " " + first1 + second2

document.f1.t1.value = fname

The entire code for our jumble() function is then this:

```
function jumble() {
fname = document.f1.t1.value
sname = fname.split(" ")
first = sname[0]
second = sname[1]
first1 = first.substring(0, 2)
first2 = first.substring(2, first.length)
second1 = second.substring(0, 2)
second2 = second.substring(2, second.length)
fname = second1 + first2 + " " + first1 + second2
document.f1.t1.value = fname
}
```

Summary:

JavaScript is a high-level, dynamic, and interpreted programming language that is primarily used for creating interactive and dynamic content on the web. It was developed by Netscape Communications in the mid-1990s and has since become one of the core technologies of web development, alongside HTML and CSS.

It is a powerful and flexible language that plays a crucial role in modern web development. Its ability to create dynamic, interactive, and real-time web applications has made it an essential tool for developers. With its growing ecosystem and versatility across different platforms, JavaScript continues to be a dominant force in the world of programming.

Check your understanding:

Question 1:

Which of the following is NOT a valid data type in JavaScript?

a) String
b) Number
c) Boolean
d) Character

Question 2

What does the this keyword refer to in JavaScript?

a) The current function
b) The global object
c) The object that is executing the current piece of code
d) The first argument passed to the function

Question: 3

Which of the following statements is used to create a new object in JavaScript?

a) new Object();
b) Object.create();
c) new Object;
d) All of the above

Question: 4

Which of the following is NOT a valid JavaScript loop?

a) For
b) While
c) Foreach
d) do-while

Question: 5

What is the purpose of the break statement in JavaScript?

a) Exit from a loop or switch statement
b) Pause a function execution
c) Declare a variable
d) Return a value from a function

Question: 6

How do you declare a function in JavaScript?

a) function myFunction[] {}
b) function myFunction() {}
c) function: myFunction()
d) myFunction function() {}

Question: 7

Which operator is used to compare both value and type in JavaScript?

a) ==
b) ===
c) =
d) !==

Question: 8

Which method is used to remove the last element from an array in JavaScript?

a) shift()
b) pop()
c) splice()
d) unshift()

Question: 9

Which of the following methods can be used to find the index of an element in an array?

a) indexOf()
b) findIndex()
c) searchIndex()
d) Both A and B

Question: 10

What is the default value of an array element that is not initialized in JavaScript?

a) Null
b) Undefined
c) NaN
d) 0

Chapter 4: VBScript

Introduction :

VBScript (Visual Basic Scripting Edition) is a lightweight scripting language developed by Microsoft. It is a variant of the Visual Basic programming language, specifically designed to be embedded in applications or web pages for scripting purposes. Introduced in 1996, VBScript was primarily used for automating tasks in the Windows environment and for client-side scripting in web development.

Key Features of VBScript:

1. **Simplicity**: VBScript is easy to learn and use, especially for those familiar with Visual Basic or basic programming concepts.
2. **Integration with Windows**: VBScript works seamlessly with Windows Script Host (WSH), enabling automation of administrative tasks and interacting with Windows components.
3. **Interoperability with COM Objects**: It can use Component Object Model (COM) objects to access various system resources and functionalities.
4. **Web Scripting**: Early web pages often used VBScript for dynamic client-side scripting, although this has been largely replaced by JavaScript.
5. **Embedding**: VBScript can be embedded in HTML documents using <script> tags.

Typical Uses of VBScript:

- **System Administration**: Automating repetitive tasks, like managing files, configuring systems, or running scripts on a schedule.
- **Data Processing**: Simple data analysis and manipulation tasks.
- **Prototyping**: Rapid prototyping for simple applications or logic.

Example VBScript Code:

```
' A simple VBScript to display a message box

    MsgBox "Hello, World!"
```

Decline and Deprecation:

VBScript's use declined over time due to security vulnerabilities and the rise of more robust and cross-platform scripting languages like JavaScript and Python. Microsoft officially deprecated VBScript in Internet Explorer 11 in 2019, further marking its obsolescence. Despite this, VBScript remains in use for legacy systems and applications.

Script Fundamentals in VBScript

VBScript is based on simple scripting principles. Below are the core elements you need to understand to write and execute VBScript programs:

1. Basic Syntax:

- **Case Insensitivity**: VBScript is not case-sensitive (MsgBox and msgbox are treated the same).
- **Lines of Code**: Each statement typically occupies one line, but the underscore (_) can be used for line continuation.
- **Comments**: Use an apostrophe (') to add a comment.

```
' This is a comment
MsgBox "Hello, World!"  ' This displays a message box
```

2. Variables:

- Variables in VBScript are declared using the Dim statement.
- All variables are of type Variant, and their type is determined dynamically based on the value assigned.

```
Dim myVar
myVar = "VBScript"
MsgBox "Learning " & myVar
```

3. Data Types:

VBScript supports the Variant data type, which can hold different types of data like:

- Numbers (Integer, Double)
- Strings
- Dates
- Boolean values (True/False)
- Empty and Null values

```
Dim number, text, flag
number = 42          ' Numeric
text = "Hello"       ' String
flag = True          ' Boolean
```

4. Operators:

- **Arithmetic**: +, -, *, /, ^ (exponentiation)
- **Comparison**: =, <, >, <=, >=, <>
- **Logical**: And, Or, Not

```
Dim a, b, result
a = 10
b = 5
result = a + b    ' Addition
MsgBox result
```

5. Control Structures:

Conditional Statements:

- **If...Then...Else**:

```
If Hour(Now) < 12 Then
   MsgBox "Good Morning"
Else
   MsgBox "Good Afternoon"
End If
```

- **Select Case**:

```
Dim day
day = Weekday(Now)
Select Case day
   Case 1: MsgBox "Sunday"
   Case 2: MsgBox "Monday"
   ' Additional cases
   Case Else: MsgBox "Other day"
End Select
```

Loops:

- **For...Next**:

```
Dim i
For i = 1 To 5
   MsgBox "Number: " & i
Next
```

- **Do...Loop**:

```
Dim count
count = 1
Do While count <= 3
   MsgBox "Count is: " & count
   count = count + 1
Loop
```

- **While...Wend**:

```
Dim j
j = 1
While j <= 3
   MsgBox "Value: " & j
   j = j + 1
Wend
```

6. Functions and Subroutines:

- **Functions**: Used to return values.

```
Function AddNumbers(a, b)
    AddNumbers = a + b
End Function
MsgBox AddNumbers(3, 7)
```

- **Subroutines**: Perform tasks without returning values.

```
Sub ShowMessage(msg)
    MsgBox msg
End Sub
Call ShowMessage("Hello from Subroutine!")
```

7. Error Handling:

VBScript provides error handling using On Error statements.

```
On Error Resume Next   ' Ignore errors
Dim result
result = 1 / 0         ' This would normally cause an error
If Err.Number <> 0 Then
    MsgBox "An error occurred: " & Err.Description
    Err.Clear          ' Clear the error
End If
```

8. Interacting with Files and System:

VBScript can manipulate files and folders using the FileSystemObject.

```
Dim fso, file
Set fso = CreateObject("Scripting.FileSystemObject")
Set file = fso.CreateTextFile("example.txt", True)
file.WriteLine "Hello, File!"
file.Close
```

9. Running VBScript:

- Save the script as a .vbs file (e.g., script.vbs).
- Double-click the file to execute it in Windows, or run it using **Windows Script Host** in the command prompt:

```
cscript script.vbs
```

The form and its Web Controls

In VBScript, web controls are typically used within an HTML form to create interactive web pages. VBScript, embedded in HTML, works with **HTML forms and their controls** to manage user input and execute scripts. Below is an overview of forms and common web controls, along with their interaction using VBScript.

1. HTML Form Basics

An HTML form serves as a container for various input controls that users interact with, such as text boxes, buttons, checkboxes, etc. VBScript can be used to validate, manipulate, and process these controls.

```
<form id="myForm">
   <label for="name">Name:</label>
   <input type="text" id="name" name="name">
   <input type="button" value="Submit" onclick="SubmitForm()">
</form>
```

2. Common Web Controls

a. Text Box (<input type="text">)

Used to capture single-line text input.

```
<input type="text" id="name" name="name">
```

b. Password Box (<input type="password">)

Used for masked text input.

```
<input type="password" id="password" name="password">
```

c. Buttons

- **Submit Button (<input type="submit">)**: Submits the form.
- **Button (<input type="button">)**: Triggers a custom script.

```
<input type="button" value="Click Me" onclick="ShowMessage()">
```

d. Checkboxes (<input type="checkbox">)

Allows the selection of multiple options.

```
<input type="checkbox" id="option1" name="option1" value="Option 1">
```

e. Radio Buttons (<input type="radio">)

Allows the selection of a single option from a group.

```
<input type="radio" id="choice1" name="choice" value="Choice 1">
<input type="radio" id="choice2" name="choice" value="Choice 2">
```

f. Drop-Down List (<select>)

Used for selecting an option from a dropdown menu.

```
<select id="dropdown">
  <option value="1">Option 1</option>
  <option value="2">Option 2</option>
</select>
```

g. Text Area (<textarea>)

Used for multi-line text input.

```
<textarea id="comments"></textarea>
```

h. Labels (<label>)

Provides text descriptions for input elements.

```
<label for="name">Name:</label>
<input type="text" id="name">
```

3. VBScript with Web Controls

VBScript can interact with these controls using their IDs or names to read, validate, and manipulate their values.

Example: Simple Form with VBScript

```
<!DOCTYPE html>
<html>
<head>
  <title>Form with VBScript</title>
  <script language="VBScript">
    Sub SubmitForm()
      Dim name, age, isChecked
      name = Document.getElementById("name").Value
      age = Document.getElementById("age").Value
      isChecked = Document.getElementById("agree").Checked
      If name = "" Or age = "" Then
        MsgBox "Please fill in all fields."
      ElseIf Not isChecked Then
        MsgBox "You must agree to the terms."
      Else
        MsgBox "Form submitted successfully! Name: " & name & ", Age: " & age
      End If
    End Sub
  </script>
```

```
</head>
<body>
   <form id="myForm">
      <label for="name">Name:</label>
      <input type="text" id="name" name="name"><br><br>
       <label for="age">Age:</label>
      <input type="number" id="age" name="age"><br><br>
            <input type="checkbox" id="agree" name="agree">
      <label for="agree">I agree to the terms</label><br><br>
            <input type="button" value="Submit" onclick="SubmitForm()">
   </form>
</body>
</html>
```

4. How VBScript Interacts with Controls

1. **Access Control Values**: Use Document.getElementById("controlID").Value for retrieving or modifying control values.

```
Dim inputValue
inputValue = Document.getElementById("name").Value
```

2. **Validate Inputs**: Use If...Then statements to check for valid inputs.

```
If inputValue = "" Then
MsgBox "Input is required."
End If
```

3. **Set or Modify Control Properties**: For example, enabling/disabling buttons.

```
Document.getElementById("submitBtn").Disabled = True
```

5. Running VBScript with Forms

1. Save the HTML code with a .html extension (e.g., form_example.html).
2. Open the file in **Internet Explorer** (VBScript is only supported in IE).
3. Interact with the form, and the VBScript will execute in response to user actions.

6. Limitations

- **Browser Support**: VBScript is supported only in older versions of Internet Explorer and is not recommended for modern web development.
- **Security**: VBScript in web pages is often disabled due to security risks.
- **Cross-Browser Compatibility**: Other browsers like Chrome, Firefox, and Edge do not support VBScript.

VBScript - Variables & Operations

In VBScript, variables and operations are fundamental for storing and manipulating data. Below is a detailed explanation:

1. Variables in VBScript

Declaring Variables

Variables in VBScript are declared using the Dim keyword. You don't need to specify the data type because all variables are of type Variant.

```
Dim myVar
```

Assigning Values

You can assign a value to a variable using the assignment operator =.

```
myVar = "Hello, VBScript!"
```

Dynamic Typing

VBScript automatically assigns a data type to a variable based on the value. Common types include:

- String: "Text"
- Integer: 10
- Double: 10.5
- Boolean: True or False
- Date: #MM/DD/YYYY#

Example:

```
Dim strName, intAge, isStudent
strName = "Alice"      ' String
intAge = 25            ' Integer
isStudent = True       ' Boolean
```

2. Variable Scope

- Procedure-Level Variables: Declared inside a subroutine or function and accessible only within that block.
- Global Variables: Declared outside any procedure and accessible throughout the script.

Example:

```
Dim globalVar   ' Global variable
Sub TestProcedure()
    Dim localVar
    localVar = "I'm local"
    MsgBox localVar
End Sub
```

3. Constants

VBScript allows the creation of constants using the Const keyword. Unlike variables, constants cannot be modified after being defined.

```
Const Pi = 3.14159
Const AppName = "My Application"
```

4. Operations in VBScript

a. Arithmetic Operations

VBScript supports common arithmetic operators for numeric data.

Operator	Description	Example	Result
+	Addition	5 + 3	8
-	Subtraction	10 - 4	6
*	Multiplication	6 * 2	12
/	Division	8 / 2	4
^	Exponentiation	2 ^ 3	8
Mod	Modulus (Remainder)	10 Mod 3	1

Example:

```
Dim a, b, result
a = 10
b = 3
result = a Mod b  ' Remainder of 10 divided by 3
MsgBox result     ' Outputs: 1
```

b. String Operations

VBScript provides operations to manipulate strings.

Operator	Description	Example	Result
&	String concatenation	"Hello" & " World"	"Hello World"
+	String concatenation (also works)	"VB" + "Script"	"VBScript"

Example:

```
Dim greeting, name
greeting = "Hello"
name = "Alice"
MsgBox greeting & ", " & name & "!"  ' Outputs: Hello, Alice!
```

c. Comparison Operators

Comparison operators return True or False.

Operator	Description	Example	Result
=	Equal to	5 = 5	True
<	Less than	3 < 5	True
>	Greater than	8 > 10	False
<=	Less than or equal	7 <= 7	True
>=	Greater or equal	5 >= 6	False
<>	Not equal	4 <> 2	True

Example:

```
Dim x, y
x = 10
y = 20
If x < y Then
   MsgBox "x is less than y"
Else
   MsgBox "x is not less than y"
End If
```

d. Logical Operations

Logical operators combine or modify Boolean values.

Operator	Description	Example	Result
And	Logical AND	True And False	False
Or	Logical OR	True Or False	True
Not	Logical NOT	Not True	False

Example:

```
Dim a, b
a = True
b = False
If a And b Then
   MsgBox "Both are True"
Else
   MsgBox "At least one is False"
End If
```

e. Assignment Operators

Operator	Description	Example	Result
=	Assign value	x = 5	x becomes 5
+=	Add and assign	x += 3	x = x + 3
-=	Subtract and assign	x -= 2	x = x - 2
*=	Multiply and assign	x *= 2	x = x * 2
/=	Divide and assign	x /= 4	x = x / 4

Example:

```
Dim num
num = 10
num += 5  ' num is now 15
MsgBox num
```

5. Example Script: Combining Variables and Operations

```
Dim num1, num2, sum, product, fullName

' Assigning values
num1 = 10
num2 = 20
fullName = "Alice" & " " & "Smith"

' Performing operations
sum = num1 + num2
product = num1 * num2

' Displaying results
MsgBox "The sum of " & num1 & " and " & num2 & " is: " & sum
MsgBox "The product of " & num1 & " and " & num2 & " is: " & product
MsgBox "Full Name: " & fullName
```

Procedures and Functions

In VBScript, procedures and functions are reusable blocks of code designed to perform specific tasks. They enhance modularity, readability, and reusability in scripts.

1. Subroutines (Procedures)

Definition

A **Subroutine** (or Procedure) performs a task but does not return a value. It is declared using the Sub keyword and executed using the Call keyword (optional).

Syntax

```
Sub SubroutineName(arguments)
    ' Code block
End Sub
```

Calling a Subroutine

You can call a subroutine with or without the Call keyword. When using Call, arguments must be enclosed in parentheses. Without Call, omit parentheses.

```
Call SubroutineName(argument)  ' Using Call
SubroutineName argument        ' Without Call
```

Example

```
Sub GreetUser(name)
    MsgBox "Hello, " & name & "!"
End Sub
' Calling the subroutine
GreetUser "Alice"
Call GreetUser("Bob")
```

2. Functions

Definition

A **Function** performs a task and returns a value to the calling code. Functions are used when you need to process data and get a result.

Syntax

```
Function FunctionName(arguments)
    ' Code block
    FunctionName = return_value
End Function
```

Calling a Function

Functions are called directly and must return a value.

```
result = FunctionName(argument)
```

Example

```
Function AddNumbers(a, b)
```

```
    AddNumbers = a + b
End Function
' Calling the function
Dim sum
sum = AddNumbers(5, 10)
MsgBox "The sum is: " & sum
```

3. Key Differences Between Subroutines and Functions

Aspect	Subroutine	Function
Return Value	Does not return a value	Returns a value
Usage	Used for performing actions	Used for calculations/tasks
Declaration	Sub SubroutineName	Function FunctionName
Return Statement	Not required	Must return a value

4. Procedures with Arguments

Both subroutines and functions can accept arguments. Arguments are passed **by value** or **by reference**.

- By Value (ByVal)

  The argument's value is passed, and changes within the procedure do not affect the original variable.

```
Sub IncrementByVal(ByVal x)
    x = x + 1
    MsgBox "Inside Sub: " & x
End Sub

Dim num
num = 10
IncrementByVal num
MsgBox "Outside Sub: " & num  ' Output: 10
```

- By Reference (ByRef)

  The argument's reference is passed, and changes within the procedure affect the original variable.

```
Sub IncrementByRef(ByRef x)
    x = x + 1
    MsgBox "Inside Sub: " & x
End Sub

Dim num
num = 10
IncrementByRef num
MsgBox "Outside Sub: " & num  ' Output: 11
```

5. Example: Subroutine vs Function

```
' Subroutine Example
Sub DisplayMessage(msg)
    MsgBox msg
End Sub
Call DisplayMessage("Hello, this is a Subroutine!")

' Function Example
Function SquareNumber(num)
    SquareNumber = num * num
End Function

Dim result
result = SquareNumber(4)
MsgBox "The square is: " & result
```

6. Advanced Concepts

- Default Parameters

VBScript does not natively support default parameter values. However, you can simulate it by checking if a parameter is empty.

```
Sub GreetUser(Optional name)
    If IsEmpty(name) Then
        name = "Guest"
    End If
    MsgBox "Hello, " & name & "!"
End Sub

Call GreetUser()       ' Outputs: Hello, Guest!
Call GreetUser("Alice") ' Outputs: Hello, Alice!
```

- Recursive Functions

A function can call itself (recursion) to solve problems like factorial calculation.

```
Function Factorial(n)
    If n = 0 Then
        Factorial = 1
    Else
        Factorial = n * Factorial(n - 1)
```

```
    End If
End Function

MsgBox "Factorial of 5 is: " & Factorial(5)  ' Output: 120
```

7. Error Handling in Procedures/Functions

Use On Error Resume Next to handle errors gracefully within a procedure or function.

```
Function DivideNumbers(a, b)
    On Error Resume Next
    If b = 0 Then
        DivideNumbers = "Error: Division by zero"
    Else
        DivideNumbers = a / b
    End If
End Function

Dim result
result = DivideNumbers(10, 0)
MsgBox result  ' Output: Error: Division by zero
```

8. Modular Programming with Procedures and Functions

You can create modular scripts by dividing tasks into subroutines and functions. For example:

```
Sub Main()
    Dim num1, num2, result
    num1 = 10
    num2 = 20

    Call DisplayMessage("Welcome to the Calculator!")

    result = AddNumbers(num1, num2)
    MsgBox "The result is: " & result
End Sub

Function AddNumbers(a, b)
    AddNumbers = a + b
End Function

Sub DisplayMessage(msg)
    MsgBox msg
End Sub
```

Built in Functions

VBScript provides a variety of **built-in functions** that perform common tasks like data type conversion, string manipulation, date and time operations, mathematical calculations, and more. Here's a comprehensive guide to VBScript's built-in functions, categorized by their purposes:

1. String Functions

Function	Description	Example	Output
Len	Returns the length of a string or number of characters.	Len("Hello")	5
Left	Returns a specified number of characters from the start.	Left("Hello", 2)	"He"
Right	Returns a specified number of characters from the end.	Right("Hello", 2)	"lo"
Mid	Extracts a substring from a string.	Mid("Hello", 2, 3)	"ell"
Trim	Removes spaces from both ends of a string.	Trim(" Hello ")	"Hello"
LTrim	Removes leading spaces.	LTrim(" Hello")	"Hello"
RTrim	Removes trailing spaces.	RTrim("Hello ")	"Hello"
Instr	Returns the position of the first occurrence of a substring.	Instr(1, "Hello World", "World")	7
Replace	Replaces occurrences of a substring with another.	Replace("Hello World", "World", "VBScript")	"Hello VBScript"
UCase	Converts a string to uppercase.	UCase("Hello")	"HELLO"
LCase	Converts a string to lowercase.	LCase("Hello")	"hello"
Space	Returns a string of spaces.	Space(5)	" "
StrReverse	Reverses a string.	StrReverse("Hello")	"olleH"

2. Mathematical Functions

Function	Description	Example	Output
Abs	Returns the absolute value of a number.	Abs(-5)	5
Sqr	Returns the square root of a number.	Sqr(16)	4
Rnd	Returns a random number between 0 and 1.	Rnd	0.12345 (example)
Int	Returns the integer portion of a number.	Int(10.75)	10

Function	Description	Example	Output
Fix	Returns the integer portion, ignoring sign.	Fix(-10.75)	-10
Round	Rounds a number to a specified number of decimals.	Round(10.567, 2)	10.57
Log	Returns the natural logarithm of a number.	Log(10)	2.302585
Sin	Returns the sine of an angle (in radians).	Sin(3.14159/2)	1
Cos	Returns the cosine of an angle (in radians).	Cos(0)	1
Tan	Returns the tangent of an angle (in radians).	Tan(0)	0

3. Conversion Functions

Function	Description	Example	Output
CInt	Converts to an integer.	CInt(10.75)	11
CDbl	Converts to a double.	CDbl("10.5")	10.5
CSng	Converts to a single-precision number.	CSng("10.5")	10.5
CStr	Converts to a string.	CStr(10.5)	"10.5"
CLng	Converts to a long integer.	CLng(12345.67)	12346
CDate	Converts to a date.	CDate("11/29/2024")	11/29/2024
IsDate	Checks if a value is a valid date.	IsDate("11/29/2024")	True
IsNumeric	Checks if a value is numeric.	IsNumeric("123")	True
IsEmpty	Checks if a variable is uninitialized.	IsEmpty(myVar)	True

4. Date and Time Functions

Function	Description	Example	Output
Now	Returns the current date and time.	Now	11/29/2024 10:30:00 AM
Date	Returns the current date.	Date	11/29/2024
Time	Returns the current time.	Time	10:30:00 AM
Year	Returns the year of a date.	Year(Now)	2024
Month	Returns the month of a date.	Month(Now)	11

Function	Description	Example	Output
Day	Returns the day of a date.	Day(Now)	29
Hour	Returns the hour of a time.	Hour(Now)	10
Minute	Returns the minute of a time.	Minute(Now)	30
Second	Returns the second of a time.	Second(Now)	0
DateAdd	Adds a time interval to a date.	DateAdd("d", 7, Now)	12/6/2024
DateDiff	Returns the difference between two dates.	DateDiff("d", #11/1/2024#, Now)	28
Weekday	Returns the day of the week for a date (as a number).	Weekday(Now)	6
MonthName	Returns the full month name of a month number.	MonthName(11)	November
WeekdayName	Returns the full weekday name of a day number.	WeekdayName(6)	Friday

5. Miscellaneous Functions

Function	Description	Example	Output
MsgBox	Displays a message box.	MsgBox "Hello!"	Displays a message.
InputBox	Displays an input box for user input.	InputBox("Enter your name:")	User input result.
IsArray	Checks if a variable is an array.	IsArray(myArray)	True or False
UBound	Returns the upper bound of an array.	UBound(myArray)	4 (for a 5-element array)

6. Example Script Using Built-in Functions

```
Dim name, greeting, today, square, randomNum

' String Manipulation
name = "Alice"
greeting = "Hello, " & UCase(name) & "!"
MsgBox greeting  ' Outputs: "Hello, ALICE!"

' Date and Time
today = Date
MsgBox "Today's date is: " & today

' Mathematical Calculations
square = Sqr(16)
```

```
MsgBox "The square root of 16 is: " & square

randomNum = Int(Rnd * 100)  ' Random number between 0 and 99
MsgBox "Random number: " & randomNum
```

Conditional Statements

Conditional statements allow you to control the flow of execution in your script based on conditions. VBScript supports several types of conditional statements:

1. If...Then Statement

Executes a block of code if a specified condition is True.

Syntax

```
If condition Then
   ' Code to execute if the condition is True
End If
```

Example

```
Dim age
age = 18

If age >= 18 Then
   MsgBox "You are eligible to vote."
End If
```

2. If...Then...Else Statement

Adds an Else block to execute code when the condition is False.

Syntax

```
If condition Then
   ' Code to execute if the condition is True
Else
   ' Code to execute if the condition is False
End If
```

Example

```
Dim age
age = 16

If age >= 18 Then
   MsgBox "You are eligible to vote."
Else
   MsgBox "You are not eligible to vote."
End If
```

3. If...Then...ElseIf...Else Statement

Allows multiple conditions to be evaluated in sequence.

Syntax

```
If condition1 Then
   ' Code to execute if condition1 is True
ElseIf condition2 Then
   ' Code to execute if condition2 is True
Else
   ' Code to execute if all conditions are False
End If
```

Example

```
Dim score
score = 75

If score >= 90 Then
   MsgBox "Grade: A"
ElseIf score >= 80 Then
   MsgBox "Grade: B"
ElseIf score >= 70 Then
   MsgBox "Grade: C"
Else
   MsgBox "Grade: F"
End If
```

4. Single-Line If...Then Statement

You can write a simple conditional statement in a single line if it contains only one action.

Syntax

```
If condition Then statement
```

Example

```
Dim age
age = 20

If age >= 18 Then MsgBox "You are an adult."
```

5. Select Case Statement

The Select Case statement is an alternative to If...Then...ElseIf. It is useful when multiple conditions depend on the value of a single variable or expression.

Syntax

```
Select Case expression
   Case value1
      ' Code to execute if expression = value1
   Case value2
      ' Code to execute if expression = value2
   Case Else
      ' Code to execute if no cases match
End Select
```

Example

```
Dim day
day = "Monday"

Select Case day
   Case "Monday", "Tuesday", "Wednesday", "Thursday", "Friday"
      MsgBox "It's a weekday."
   Case "Saturday", "Sunday"
      MsgBox "It's the weekend."
   Case Else
      MsgBox "Invalid day."
End Select
```

6. Nested Conditional Statements

You can nest If or Select Case statements inside each other for more complex conditions.

Example

```
Dim score, attendance
score = 85
attendance = 90

If score >= 70 Then
   If attendance >= 75 Then
      MsgBox "You passed the course."
   Else
      MsgBox "You failed due to low attendance."
   End If
Else
   MsgBox "You failed due to low score."
End If
```

7. Combining Conditions

You can combine multiple conditions using logical operators:

- **And**: All conditions must be True.
- **Or**: At least one condition must be True.
- **Not**: Negates a condition.

Example

```
Dim age, citizen
age = 20
citizen = True

If age >= 18 And citizen Then
   MsgBox "You are eligible to vote."
Else
   MsgBox "You are not eligible to vote."
End If
```

8. Error Handling in Conditional Statements

Use conditional statements with error handling to ensure smooth execution.

Example

```
On Error Resume Next

Dim number
number = "ABC"

If IsNumeric(number) Then
   MsgBox "The number is valid: " & number
Else
   MsgBox "Invalid number!"
End If

On Error GoTo 0
```

Example Script with Multiple Conditional Statements

```
Dim age, income
age = 25
income = 50000

If age >= 18 Then
   If income >= 40000 Then
      MsgBox "You qualify for the premium plan."
```

```
    Else
        MsgBox "You qualify for the basic plan."
    End If
Else
    MsgBox "You are not eligible for any plan."
End If
```

Counting & Looping

In VBScript, loops are used to execute a block of code multiple times based on certain conditions. There are various types of loops available to manage repetitive tasks. Below, we'll discuss how to use **counting loops** and **conditional loops** for different situations.

1. For...Next Loop (Counting Loop)

The For...Next loop is used when you know the number of times the loop should execute. It counts through a set number of iterations and is ideal for repetitive tasks where the starting and ending points are known.

Syntax

```
For counter = start To end [Step step]
    ' Code to be executed
Next
```

- **counter**: The loop variable that will increment or decrement.
- **start**: The starting value of the counter.
- **end**: The value at which the loop will stop.
- **Step** (optional): The increment/decrement between each iteration (defaults to 1).

Example

```
For i = 1 To 5
    MsgBox "Iteration " & i
Next
```

This loop will display a message box for each iteration, from 1 to 5.

Example with Step

```
For i = 1 To 10 Step 2
    MsgBox "Current value: " & i
Next
```

This loop will output the values 1, 3, 5, 7, 9 as Step increments by 2.

2. For Each...Next Loop (For Collections)

The For Each...Next loop is used when you want to loop through elements of a collection or array. This is ideal for arrays, dictionaries, or other collections.

Syntax

```
For Each element In collection
    ' Code to be executed
Next
```

- **element**: Represents each item in the collection.
- **collection**: The array or collection being looped through.

Example

```
Dim fruits(3)
fruits(0) = "Apple"
fruits(1) = "Banana"
fruits(2) = "Orange"

For Each fruit In fruits
    MsgBox "Fruit: " & fruit
Next
```

This will display a message box for each fruit in the array, showing "Apple", "Banana", and "Orange".

3. Do...Loop (Conditional Loop)

The Do...Loop is a flexible loop that allows you to repeat a block of code while a condition is true or until a condition becomes true.

Syntax 1: Do While Loop

```
Do While condition
    ' Code to be executed
Loop
```

- The loop continues as long as the condition evaluates to True.

Example (Do While Loop)

```
Dim counter
counter = 1
Do While counter <= 5
    MsgBox "Counter: " & counter
    counter = counter + 1
Loop
```

This will display a message box with the value of counter from 1 to 5.

Syntax 2: Do Until Loop

```
Do Until condition
   ' Code to be executed
Loop
```

- The loop continues until the condition becomes True.

Example (Do Until Loop)

```
Dim counter
counter = 1

Do Until counter > 5
   MsgBox "Counter: " & counter
   counter = counter + 1
Loop
```

This will also output the values from 1 to 5, but the loop continues **until** counter > 5.

Syntax 3: Do...Loop with Exit Condition

You can use Exit Do to exit the loop prematurely when a condition is met.

```
Dim counter
counter = 1
Do
   If counter = 3 Then
      Exit Do
   End If
   MsgBox "Counter: " & counter
   counter = counter + 1
Loop
```

In this example, the loop will terminate after displaying the message for counter = 3.

4. While...Wend Loop (Another Conditional Loop)

The While...Wend loop is an older form of conditional looping in VBScript, functioning similarly to Do...Loop.

Syntax

```
While condition
   ' Code to be executed
Wend
```

- The loop continues as long as the condition is True.

Example

```
Dim counter
counter = 1

While counter <= 5
   MsgBox "Counter: " & counter
   counter = counter + 1
Wend
```

This will output values from 1 to 5, just like the Do While loop.

5. Nested Loops

You can nest loops within each other for more complex tasks, like processing multidimensional arrays or iterating through combinations.

Example

```
Dim i, j
For i = 1 To 3
   For j = 1 To 3
      MsgBox "i = " & i & ", j = " & j
   Next
Next
```

This will display the combinations of i and j, such as i = 1, j = 1, i = 1, j = 2, and so on.

6. Exiting Loops Prematurely

If you want to exit a loop before the condition is met, use the Exit For, Exit Do, or Exit While statement, depending on the type of loop.

Example: Exiting For Loop

```
For i = 1 To 10
   If i = 5 Then
      Exit For  ' Exits the loop when i = 5
   End If
   MsgBox "i = " & i
Next
```

Example: Exiting Do Loop

```
Dim counter
counter = 1
```

```
Do
   If counter > 3 Then
      Exit Do  ' Exits the loop when counter > 3
   End If
   MsgBox "Counter: " & counter
   counter = counter + 1
Loop
```

Best Practices for Using Loops

1. **Avoid Infinite Loops**: Make sure that your loops have a clear exit condition to prevent the script from running indefinitely.
2. **Limit Nested Loops**: Too many nested loops can make the code harder to maintain and may impact performance. Refactor if possible.
3. **Use Step in For Loops**: Specify a Step if you want to increment/decrement the counter by more than 1, which can make your loops more efficient.
4. **Break Early if Needed**: Use Exit statements wisely to break out of loops when a condition is met, avoiding unnecessary iterations.

Example: A Complete Script Using Loops

```
Dim i, total
total = 0

For i = 1 To 5
   total = total + i
Next

MsgBox "The sum of numbers 1 to 5 is: " & total
```

This script will sum the numbers from 1 to 5 and display the result (15) in a message box.

Summary:

VBScript (Visual Basic Scripting Edition) is a lightweight, interpreted scripting language developed by Microsoft. It is primarily used for automating tasks in Windows environments, web development in Internet Explorer, and manipulating system functions.It is a simple, lightweight scripting language primarily used for automating tasks within Windows environments and creating dynamic content in legacy web applications.

Check your Understanding:

Question 1:

Which of the following is NOT a valid data type in VBScript?

a) String
b) Integer
c) Boolean
d) Array

Question 2:

Which statement is used to exit a loop prematurely in VBScript?

a) Exit
b) Break
c) Exit For
d) Exit Loop

Question 3:

Which function is used to read input from the user in VBScript?

a) InputBox
b) MsgBox
c) Read
d) Console.ReadLine

Question 4:

Which of the following is a valid VBScript function to get the length of a string?

a) GetLength
b) Len
c) Length
d) StrLength

Question 5:

Which of the following is a valid VBScript loop structure?

a) While-Do
b) Do-While
c) Until-Loop
d) Loop-Until

Question 6:

Which function is used to convert a string to lowercase in VBScript?

a) Lower
b) ToLower
c) LCase
d) ConvertToLower

Question 7:

What is the correct syntax for a “For Each” loop in VBScript?

a) For i = 1 to 10
b) For Each i in 1 to 10
c) For Each i in Array(1 to 10)
d) For Each i in Range(1 to 10)

Question 8:

Which function is used to convert a value to a string in VBScript?

a) CStr
b) ToString
c) Convert.ToString
d) Str

Question 9:

What is the output of the following VBScript code?

```
a = 10
b = 3
c = a / b
MsgBox c
```

a) 3.33
b) 3.0
c) 3
d) 3.3333

Question 10:

What is the correct syntax for a multi-line comment in VBScript?

a) /* … */
b) ‘…….’
c) { … }
d) Rem …

THE END

Great! Practice Regularly.