.NET Programming Concepts

1. Define variable and constant.

A variable can be defined as a meaningful name that is given to a data storage location in the computer memory that contains a value. Every variable associated with a data type determines what type of value can be stored in the variable, for example an integer, such as 100, a decimal, such as 30.05, or a character, such as 'A'.

You can declare variables by using the following syntax:

<Data_type> <variable_name> ;

A constant is similar to a variable except that the value, which you assign to a constant, cannot be changed, as in case of a variable. Constants must be initialized at the same time they are declared. You can declare constants by using the following syntax:

const int interestRate = 10;

2. What is a data type? How many types of data types are there in .NET ?

A data type is a data storage format that can contain a specific type or range of values. Whenever you declare variables, each variable must be assigned a specific data type. Some common data types include integers, floating point, characters, and strings. The following are the two types of data types available in .NET:

- **Value type** - Refers to the data type that contains the data. In other words, the exact value or the data is directly stored in this data type. It means that when you assign a value type variable to another variable, then it copies the value rather than copying the reference of that variable. When you create a value type variable, a single space in memory is allocated to store the value (stack memory). Primitive data types, such as int, float, and char are examples of value type variables.
- **Reference type** - Refers to a data type that can access data by reference. Reference is a value or an address that accesses a particular data by address, which is stored elsewhere in memory (heap memory). You can say that reference is the physical address of data, where the data is stored in memory or in the storage device. Some built-in reference types variables in .Net are string, array, and object.

3. Mention the two major categories that distinctly classify the variables of C# programs.

Variables that are defined in a C# program belong to two major categories: **value type** and **reference type**. The variables that are based on value type contain a value that is either allocated on a stack or allocated in-line in a structure. The variables that are based on reference types store the memory address of a variable, which in turn stores the value and are allocated on the heap. The variables that are based on value types have their own copy of data and therefore operations done on one variable do not affect other variables. The reference-type variables reflect the changes made in the referring variables.

Predict the output of the following code segment:

```
int x = 42;
int y = 12;
int w;
object o;
o = x;
w = y * (int)o;
Console.WriteLine(w);

/* The output of the code is 504. */
```

4. Which statement is used to replace multiple if-else statements in code.

In Visual Basic, the **Select-Case** statement is used to replace multiple **If - Else** statements and in C#, the **switch-case** statement is used to replace multiple **if-else** statements.

5. What is the syntax to declare a namespace in .NET?

In .NET, the namespace keyword is used to declare a namespace in the code.

The syntax for declaring a namespace in C# is:
namespace UserNameSpace;

The syntax for declaring a namespace in VB is:
Namespace UserNameSpace

6. What is the difference between constants and read-only variables that are used in programs?

Constants perform the same tasks as read-only variables with some differences. The differences between constants and read-only are

Constants:

1. Constants are dealt with at compile-time.
2. Constants supports value-type variables.
3. Constants should be used when it is very unlikely that the value will ever change.

Read-only:

1. Read-only variables are evaluated at runtime.
2. Read-only variables can hold reference type variables.
3. Read-only variables should be used when run-time calculation is required.

7. Differentiate between the *while* and *for* loop in C#.

The *while* and *for* loops are used to execute those units of code that need to be repeatedly executed, unless the result of the specified condition evaluates to false. The only difference between the two is in their syntax. The *for* loop is distinguished by setting an explicit loop variable.

8. What is an identifier?

Identifiers are northing but names given to various entities uniquely identified in a program. The name of identifiers must differ in spelling or casing. For example, *MyProg* and *myProg* are two different identifiers. Programming languages, such as C# and Visual Basic, strictly restrict the programmers from using any keyword as identifiers. Programmers cannot develop a class whose name is *public*, because, *public* is a keyword used to specify the accessibility of data in programs.

9. What does a break statement do in the switch statement?

The *switch* statement is a selection control statement that is used to handle multiple choices and transfer control to the *case* statements within its body. The following code snippet shows an example of the use of the *switch* statement in C#:

```
switch(choice)
{
```

```
 case 1:
 console.WriteLine("First");
 break;
 case 2:
 console.WriteLine("Second");
 break;
 default:
 console.WriteLine("Wrong choice");
 break;
}
```

In *switch* statements, the *break* statement is used at the end of a *case* statement. The *break* statement is mandatory in C# and it avoids the fall through of one *case* statement to another.

10. Explain keywords with example.

Keywords are those words that are reserved to be used for a specific task. These words cannot be used as identifiers. You cannot use a keyword to define the name of a variable or method. Keywords are used in programs to use the features of object-oriented programming.

For example, the *abstract* keyword is used to implement abstraction and the inherits keyword is used to implement inheritance by deriving subclasses in C# and Visual Basic, respectively.

The *new* keyword is universally used in C# and Visual Basic to implement encapsulation by creating objects.

11. Briefly explain the characteristics of value-type variables that are supported in the C# programming language.

The variables that are based on value types directly contain values. The characteristics of value-type variables that are supported in C# programming language are as follows:

- All value-type variables derive implicitly from the *System.ValueType* class
- You cannot derive any new type from a value type
- Value types have an implicit default constructor that initializes the default value of that type
- The value type consists of two main categories:
    - Structs - Summarizes small groups of related variables.
    - Enumerations - Consists of a set of named constants.

12. Give the syntax of using the *while* loop in a C# program.

The syntax of using the *while* loop in C# is:

```
while(condition) //condition
{
 //statements
}
```

You can find an example of using the *while* loop in C#:

```
int i = 0;
while(i < 5)
{
 Console.WriteLine("{0} ", i);
 i++;
}
```

The output of the preceding code is: *0 1 2 3 4* .

13. What is a parameter? Explain the new types of parameters introduced in C# 4.0.

A parameter is a special kind of variable, which is used in a function to provide a piece of information or input to a caller function. These inputs are called arguments. In C#, the different types of parameters are as follows:

- **Value type** - Refers that you do not need to provide any keyword with a parameter.
- **Reference type** - Refers that you need to mention the *ref* keyword with a parameter.
- **Output type** - Refers that you need to mention the *out* keyword with a parameter.
- **Optional parameter** - Refers to the new parameter introduced in C# 4.0. It allows you to neglect the parameters that have some predefined default values. The example of optional parameter is as follows:
- `public int Sum(int a, int b, int c = 0, int d = 0); /* c and d is optional */`
- `Sum(10, 20); //10 + 20 + 0 + 0`
- `Sum(10, 20, 30); //10 + 20 + 30 + 0`
- `Sum(10, 20, 30, 40); //10 + 20 + 30 + 40`
- **Named parameter** - Refers to the new parameter introduced in C# 4.0. Now you can provide arguments by name rather than position. The example of the named parameter is as follows:
- `public void CreateAccount(string name, string address = "unknown", int age = 0);`
- `CreateAccount("Sara", age: 30);`
- `CreateAccount(address: "India", name: "Sara");`

14. Briefly explain the characteristics of reference-type variables that are supported in the C# programming language.

The variables that are based on reference types store references to the actual data. The keywords that are used to declare reference types are:

1. **Class** - Refers to the primary building block for the programs, which is used to encapsulate variables and methods into a single unit.
2. **Interface** - Contains only the signatures of methods, properties, events, or indexers.
3. **Delegate** - Refers to a reference type that is used to encapsulate a named or anonymous method.

15. What are the different types of literals?

A literal is a textual representation of a particular value of a type.

The different types of literals in Visual Basic are:

- Boolean Literals - Refers to the True and False literals that map to the true and false state, respectively.
- Integer Literals - Refers to literals that can be decimal (base 10), hexadecimal (base 16), or octal (base 8).
- Floating-Point Literals - Refers to an integer literal followed by an optional decimal point By default, a floating-point literal is of type Double.
- String Literals - Refers to a sequence of zero or more Unicode characters beginning and ending with an ASCII double-quote character.
- Character Literals - Represents a single Unicode character of the Char type.
- Date Literals - Represents time expressed as a value of the Date type.
- Nothing - Refers to a literal that does not have a type and is convertible to all types in the type system.

The different types of literals in C# are:

- Boolean literals - Refers to the True and False literals that map to the true and false states, respectively.
- Integer literals - Refers to literals that are used to write values of types int, uint, long, and ulong.
- Real literals - Refers to literals that are used to write values of types float, double, and decimal.

- Character literals - Represents a single character that usually consists of a character in quotes, such as 'a'.
- String literals - Refers to string literals, which can be of two types in C#:
    - A regular string literal consists of zero or more characters enclosed in double quotes, such as "hello".
    - A verbatim string literal consists of the @ character followed by a double-quote character, such as @"hello".

- The Null literal - Represents the null-type.

16. What is the main difference between sub-procedure and function?

The sub-procedure is a block of multiple visual basic statements within Sub and End Sub statements. It is used to perform certain tasks, such as changing properties of objects, receiving or processing data, and displaying an output. You can define a sub-procedure anywhere in a program, such as in modules, structures, and classes.

We can also provide arguments in a sub-procedure; however, it does not return a new value.
The function is also a set of statements within the Function and End Function statements. It is similar to sub-procedure and performs the same task. The main difference between a function and a sub-procedure is that sub-procedures do not return a value while functions do.

17. Determine the output of the code snippet.

```
int a = 29;
a--;
a -= ++a;
Console.WriteLine("The value of a is: {0}", a);

/* The output of the code is -1. */
```

18. Differentiate between Boxing and Unboxing.

When a value type is converted to an object type, the process is known as boxing; whereas, when an object type is converted to a value type, the process is known as unboxing.

Boxing and unboxing enable value types to be treated as objects. Boxing a value type packages it inside an instance of the Object reference type. This allows the value type to be stored on the garbage collected heap. Unboxing extracts the value type from the object. In this example, the integer variable *i* is boxed and assigned to object *obj*.

Example:

```
int i = 123;
object obj = i; /* Thi line boxes i. */

/* The object obj can then be unboxed and assigned to integer variable i: */
i = (int)obj; // unboxing
```

19. Give the syntax of using the *for* loop in C# code?

The syntax of using the *for* loop in C# code is given as follows:

```
for(initializer; condition; loop expression)
{
 //statements
```

```
}
```

In the preceding syntax, initializer is the initial value of the variable, condition is the expression that is checked before the execution of the *for* loop, and loop expression either increments or decrements the loop counter.

The example of using the *for* loop in C# is shown in the following code snippet:

```
for(int i = 0; i < 5; i++)
 Console.WriteLine("Hello");
```

In the preceding code snippet, the word Hello will be displayed for five times in the output window.