Programming Constructs in C#

Variables and Data Types

Variables are used to store data, and data types define the kind of data that can be stored in a variable.

```csharp
int number = 10;
string message = "Hello, World!";
bool isActive = true;
```

Constants

Constants are immutable values which are known at compile time and do not change for the life of the program.

```csharp
const double PI = 3.14159;
```

Operators

Operators are symbols that specify which operations to perform on variables and values.

```csharp
int sum = 5 + 3;
int product = 5 * 3;
bool isEqual = (5 == 3);
```

Control Flow Statements

Control flow statements are used to control the flow of execution in a program.

if-else

```csharp
if (number > 0)
{
   Console.WriteLine("Positive number");
}
else
{
   Console.WriteLine("Non-positive number");
}
```

switch

```csharp
```

```csharp
switch (dayOfWeek)
{
   case "Monday":
      Console.WriteLine("Start of work week");
      break;
   case "Friday":
      Console.WriteLine("End of work week");
      break;
   default:
      Console.WriteLine("Midweek");
      break;
}
```
for Loop
```csharp
for (int i = 0; i < 5; i++)
{
   Console.WriteLine(i);
}
```
while Loop
```csharp
int i = 0;
while (i < 5)
{
   Console.WriteLine(i);
   i++;
}
```
do-while Loop
```csharp
int i = 0;
do
{
   Console.WriteLine(i);
   i++;
} while (i < 5);
```
foreach Loop
```csharp
string[] fruits = { "Apple", "Banana", "Cherry" };
foreach (string fruit in fruits)
{
```

```
    Console.WriteLine(fruit);
}
```

Methods

Methods are blocks of code that perform a specific task and can be called upon as needed.

```csharp
void Greet(string name)
{
    Console.WriteLine($"Hello, {name}!");
}

Greet("Alice");
```

Classes and Objects

Classes are blueprints for objects. Objects are instances of classes.

```csharp
class Person
{
    public string Name { get; set; }
    public int Age { get; set; }

    public void Introduce()
    {
        Console.WriteLine($"Hello, my name is {Name} and I am {Age} years old.");
    }
}

Person person = new Person();
person.Name = "John";
person.Age = 30;
person.Introduce();
```

Inheritance

Inheritance is a way to form new classes using classes that have already been defined.

```csharp
class Animal
{
    public void Eat()
    {
        Console.WriteLine("Eating...");
    }
```

```
}

class Dog : Animal
{
    public void Bark()
    {
        Console.WriteLine("Barking...");
    }
}

Dog dog = new Dog();
dog.Eat();
dog.Bark();
```

Interfaces

Interfaces define a contract that implementing classes must fulfill.

```csharp
interface IFlyable
{
    void Fly();
}

class Bird : IFlyable
{
    public void Fly()
    {
        Console.WriteLine("Flying...");
    }
}

Bird bird = new Bird();
bird.Fly();
```

Enumerations

Enumerations provide a way to define a set of named integral constants.

```csharp
enum DaysOfWeek
{
    Sunday,
    Monday,
    Tuesday,
    Wednesday,
```

```
    Thursday,
    Friday,
    Saturday
}

DaysOfWeek today = DaysOfWeek.Monday;
Console.WriteLine(today);
```

Exception Handling

Exception handling provides a way to handle runtime errors in a controlled fashion.

```csharp
try
{
    int result = 10 / 0;
}
catch (DivideByZeroException ex)
{
    Console.WriteLine("Cannot divide by zero.");
}
finally
{
    Console.WriteLine("This will always execute.");
}
```