Tutorial – Functions And Intro To Classes

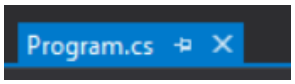
This tutorial is going to look at functions, how we create them and their many uses.

Before We Begin

Create a new C# Console project called **Functions.** Examine your new file:

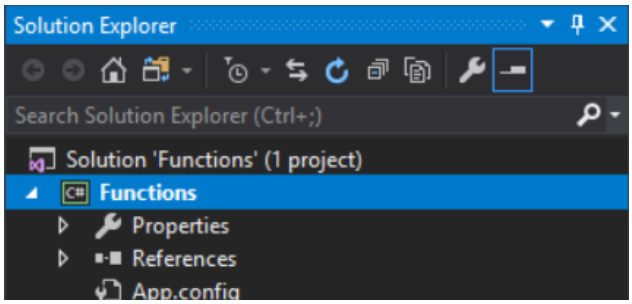
```
namespace Functions
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Notice how we already have a class called **Program.** If you look at the tabs at the top of your screen, you'll see the code file you're inside is called **Program.cs.**



Every C# file in your project will be a class, and when you have different files / classes, they can talk to each other.

Go to the **Solution Explorer** on the right hand side of your screen and **right mouse click** on the name of your project, which should be **Functions.** You'll see a pop-up menu. Go to:



1. Add
2. Class
3. Name your class **Game.cs**
4. Click **Add**

It will now create a new class for you called **Game**:

```
namespace Functions
{
    class Game
    {
    }
}
```

Creating A Function

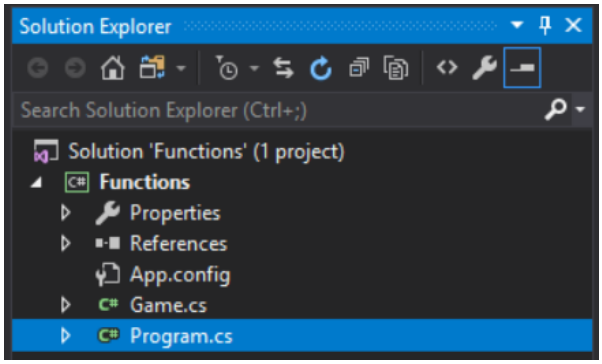
Now we will create our first function. Add the following code:

```
class Game
{
    public void Start()
    {
        Console.WriteLine("My game has begun!");
        Console.ReadKey();
    }
}
```

Start is the function's name. The **parentheses** are for **arguments** (more on that later).
void means it doesn't return any values (more on that later).
public means other classes / files can use this function.
The code between the **curly braces** for printing text is what will happen when we use our function.

When you create a function like this, it's called the **function definition.** If you run the game, it won't print text yet. We need our other class to talk to our new class.

Go back to our **Program.cs** file (double click on the file in the **Solution Explorer**):



Add the following code:

```
class Program
{
    static void Main(string[] args)
    {
        Game game = new Game();
        game.Start();
    }
}
```

Our **Program** class already has a function. It's called **Main**, and we just added some code to it. **Program** is the default class every C# console app has. **Main** is the default function and the first piece of code that will run.

static is a special function type you don't need to understand right now, but it limits what we can do. Because of these limitations, we need to create other classes if we want to use other functions.

When we create a class, think of it as a template. Our **Game class** is a template from which we can make **Game objects**, also known as **instances**.

Remember how we declared variables? **int someVariable = 0;** We do something similar when we use our classes.

```
Game game = new Game();
```

The above code says, "Make a new **object** from the **Game class template.**" We are create a new object from our **Game class** that has all of the variables and functions inside **Game.** By using the classes **name** as the **type** when we declare it, that's how we know what class to use.

To use a public function from our **Game** class, we have to go through our new **game** variable:

```
game.Start();
```

We use an **instance** of a **class** just like we do with other variables, by calling the name of the **object**, which in this case is **game**. Then we put **.Start()** after it.

When you use a functions name in your code, it accesses it's special information and properties. When we type **game.Start(),** we're saying, **"I want to use the Start() function inside my instance of the Game class".**

Run your code and see what happens. You'll see the text appear now. Let's leave classes alone for now and focus on **functions.**

What Are Functions

Functions are chunks of code you can write to do a specific task. Rather than copy and pasting code, when you call the the function, you can re-use that code as much as you like.

Your First Function

Go back to the **Game** class and look at your function called **Start**:

```
game.Start();
```

In the above code, we told our program to use all of the code inside our **Start** function of the **Game** class. A function goes in the pattern of:

privacy return value Name(arguments)
{
Run the code here
}

Our **Start** function is **public** which means other classes can access the function, like we said earlier. If we change it to private:

```
        private void Start()
```

Everything looks fine at first, but go back to your **Program** class. You'll see this:

```
game.Start();
```

Because **Start** is now **Private,** other classes can't access the function.
Go back to your **Game** class and make your function public again:

```
        public void Start()
```

Don't worry about **void** yet**.**

When you call the **name** of your function, it will run all of the code that goes between the **curly braces** of that **function.** Modify your **Game** class as follows:

```
    class Game
    {
        public void Start()
        {
            Console.WriteLine("My game has begun!");
            Console.ReadKey();
            MonsterEncounter();
        }

        private void MonsterEncounter()
        {
            Console.WriteLine("A vicious monster appears!");
            Console.ReadKey();
        }
    }
```

Having **function definitions** between the **curly braces** of your class **defines** that they belong to that class. You'll get errors if you try and put them outside of the classes curly braces.

We create the function **MonsterEncounter** and set it to **private** and **void.**
Inside the function is code to display text for a new battle.
In our **Start** function we added a new line: **MonsterEncounter ();**
Run your program and see what happens.

The original text appears, then the battle text from the **MonsterEncounter** function appears.

We call the **Start** function from our **Program** class with **game.Start().**
At the end of our **Start** function we use the **MonsterEncounter()** function by calling it's name.

Let's add more code to show how functions can be re-used:

```
    class Game
    {
        int playerHealth = 100;
        int monsterDamage = 30;

        public void Start()
        {
            Console.WriteLine("My game has begun!");
            Console.ReadKey();

            MonsterEncounter();
            MonsterAttacks();
            MonsterAttacks();
```

```
            MonsterAttacks();
        }

        private void MonsterEncounter()
        {
            Console.WriteLine("A vicious monster appears!");
            Console.ReadKey();
        }

        private void MonsterAttacks()
        {
            playerHealth -= monsterDamage;
            Console.WriteLine("The monster attacks you for " + monsterDamage + "
points of damage. You have " + playerHealth + " health left.");
            Console.ReadKey();
        }
    }
```

We added variables and a new function called **MonsterAttacks.** Remember to put the variables and the new function inside the **curly braces** of the class.

Our **Start** function calls **MonsterAttacks** 3 times. Run your game and see what that does.

The monster damages the player 3 times, leaving him with **10 health**. This is how you re-use functions. It saves time and makes code easier to read. If you showed the **Start** function to someone who didn't know how to program, they would have some idea what was going on.

It is also quicker to fix errors and make changes because you only have to change code in one place. Change the **MonsterAttacks** function as follows:

```
private void MonsterAttacks()
        {
            playerHealth -= monsterDamage * 2;
            Console.WriteLine("The monster attacks you for " + monsterDamage * 2 + "
points of damage. You have " + playerHealth + " health left.");
            Console.ReadKey();
        }
```

Monster attacks now do double damage. If we hadn't used a function, we'd have to find every time the monster did damage to the player in our code and change it. Not only is this a waste of time, we'd more likely make errors.

Arguments

You can pass variables into functions. Modify your **MonsterAttacks** function:

```
        ~~private void MonsterAttacks()~~
        private void MonsterAttacks(int damage)
        {
            playerHealth -= damage;
            Console.WriteLine("The monster attacks you for " + damage + " points of
damage. You have " + playerHealth + " health left.");
            Console.ReadKey();
        }
```

The **parentheses (brackets)** after the function's name now have a variable inside them: **int damage.** We then use that variable to subtract from our players health and also display it in the text.
Before you test your game, modify your **Start** function as follows:

```
public void Start()
        {
            Console.WriteLine("My game has begun!");
            Console.ReadKey();

            MonsterEncounter();
            MonsterAttacks(monsterDamage * 2);
            MonsterAttacks(monsterDamage);
            MonsterAttacks(monsterDamage * 3);
        }
```

Notice how we put something between the **parentheses** when we use the **MonsterAttacks** function? Run your game. What happens?

When we **define** a **function,** variables that we put inside the **parentheses** are called **arguments.** They are temporary variables that we create to be used **only** inside that function. They do not exist outside the function.

When we type **MonsterAttacks(monsterDamage * 2),** the variable **int damage** in our function becomes 60, or double the value of **monsterDamage**.

The value of the variable **monsterDamage** never changes. That's because **int damage** is a temporary variable that exists just inside that function. Once we've used the function, the variable disappears.

To illustrate the point, try modifying your start function as follows:

```
  public void Start()
        {
            Console.WriteLine("My game has begun!");
            Console.ReadKey();

            MonsterEncounter();
            MonsterAttacks(monsterDamage * 2);
            MonsterAttacks(monsterDamage);
            MonsterAttacks(monsterDamage * 3);

            damage = 5;
        }
```

You can try and do it, but **damage** will be underlined in red because we are in the **Start** function and **int damage** only exists in the **MonsterAttacks** function. The same happens if we do this to our **MonsterAttacks** function:

```
private void MonsterAttacks(int damage)
        {
            int combo = 0;
            combo += 1;
            playerHealth -= damage;
            Console.WriteLine("The monster attacks you for " + damage + " points of
damage. You have " + playerHealth + " health left.");
            Console.ReadKey();
        }
```

Try using the variable **combo** in your **Start** function and you will get is a red underline. That's because a variable you create in this way inside a function also only exists inside that function.

That's why when we create a variable we want to use anywhere in our class, we declare it outside of the functions, but inside the class, like this:

```
class Game
    {
        int playerHealth = 100;
```

Returning Values

Delete your functions in the **Game** class then modify as follows:

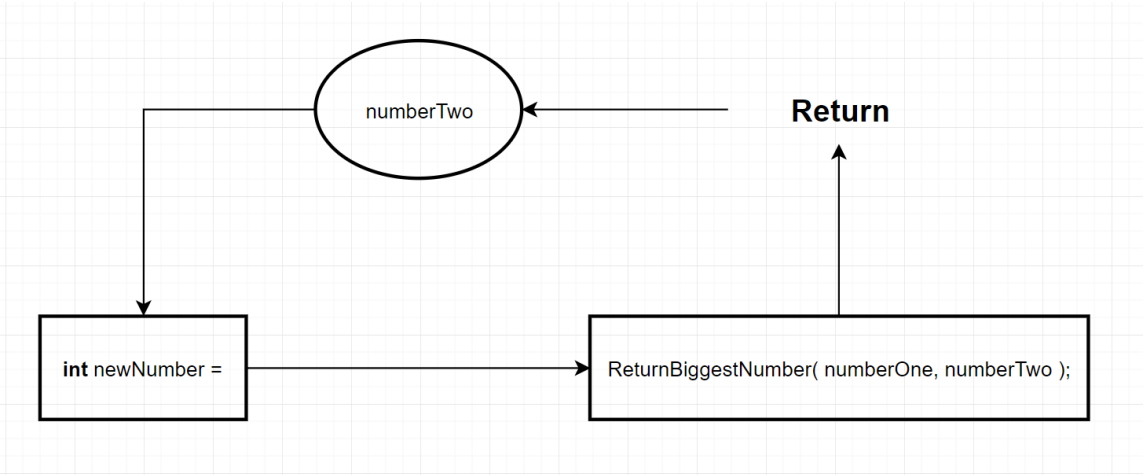
```
   class Game
    {
        int numberOne = 10;
        int numberTwo = 200;

        public void Start()
        {
            int biggestNumber = ReturnBiggestNumber(numberOne, numberTwo);

            Console.WriteLine("Of " + numberOne + " and " + numberTwo + " the biggest
number is " + biggestNumber);
            Console.ReadKey();
        }

        int ReturnBiggestNumber(int first, int second)
        {
            if(first > second)
            {
                return first;
            }
            else
            {
                return second;
            }
        }
    }
```

We created some new variables. We then create a temporary variable (also known as a **local variable**) inside our **Start** function, but we set it's value to a function.

Our new function is not **public** or **private** but blank. That means the privacy is **private** by default. Then instead of **void** as a **return type,** we choose **int,** which means the function returns an **int.** Run the program, look what happens and carefully read the function.

First, we make two temporary variables, **int first** and **int second.**
Then we have an **if / else statement.** We check to see which of the two numbers are bigger.
If **first** is bigger, we **return first.**
Else, we **return second.**

Here's a diagram to illustrate what happens:



Which do you think is bigger?
numberOne = 10;
numberTwo = 200;

Clearly numberTwo is bigger. So our function would **return** numberTwo, which means it spits that variable back out. But then what does it do with it? Look at the line that uses it:

```
int biggestNumber = ReturnBiggestNumber(numberOne, numberTwo);
```

Since we know **numberTwo** is bigger, we could rewrite that as:

```
int biggestNumber = numberTwo;
```

Whatever value a function returns, it spits it back out so that something else can use it.

Naming Conventions

It's also important to know that when we name functions and classes, we use something called **PascalCase.** It's almost exactly the same as **camelCase**, except that **camelCase** has the first letter **not capitalised. PascalCase** always has the first letter **capitalised.** When you use **PascalCase** on your **functions** and **classes**, and **camelCase** on your **variables**, it's easier to skim read your code and understand at a quick glance, what's a variable, and what's a function.

PascalCase example: **SwampMonster** or **PlayerController**
camelCase example: **swampMonster** or **playerController**

Conclusion

There were a lot of concepts to take in from this tutorial. Take a look at the Review Sheet and take some time to let it soak in.

Try making some functions of your own and experimenting. Don't worry if classes still seem a little weird, we only talked about them briefly and they will be covered again later in more detail.