Return

The basic way to *return*values from a method is to use a `return` statement! (A well-constructed programming language shouldn't have a lot of surprises.)

Let's start with an example in the below code. It contains a definition of the `Yell()` method, which returns a string, and it calls that method in `Main()`.

When we execute this program, the code in `Main()` runs first:

> `Yell()` is called.
>
> In `Yell()`, the the uppercase version of `"who's there?"` is created and returned.
>
> Back in `Main()`, that returned value is stored in the variable `output` and then printed to the console:

```
static string Yell(string phrase)
{
  return phrase.ToUpper();
}

public static void Main()
{
  string output = Yell("who's there?");
  Console.WriteLine(output); // Prints WHO'S THERE?
}
```

Here's a more generic definition: the keyword `return` tells the computer to exit the method and return a value to wherever the method was called.

When a method is declared, it must announce the type of value it will return. In this case, `Yell()` returns a string, so it has the `string` modifier (right before the name `Yell`).

That first line of the method is called a *method declaration*, so we can say that the method declaration must contain the type of the return value.

Generally, the method declaration is a combination of details including: the access modifiers, return type, method name, and parameter types. This lesson will not cover access modifiers, like `static`, so that we can focus on the return type, like `string`.

☑**Instructions**

1.
Let's define a method `DecoratePlanet()` that takes a planet name as input and returns a fancy welcome to the planet.

First, write the method declaration. It should have a `string` parameter and return a `string`.

Hint

The method should return a `string` and have one `string` parameter.

2.
Write the method body so that it returns a fancy welcome to the planet. For example, calling

```
DecoratePlanet("Mars");
```

returns:

```
"*.*.* Welcome to Mars *.*.*"
```

Hint

You'll need to use string interpolation in the body of the method, which requires $"{}" syntax. In this example, multiplier is a variable:

$"He had a heart {multiplier} sizes too small"

3.

Call the method with the argument `"Jupiter"` and print its output to the console.

Hint

This step asks you to nest methods:

```
OneMethod(AnotherMethod(arg));
```

Or store the returned value of one method and use it as input to the next:

```
string result = AnotherMethod(arg);
OneMethod(result);
```