LESSON 03 – METHODS THAT RETURN A VALUE

In this lesson we will examine how to create methods that return a value. We can pass arguments to a method, process those arguments, and have the method return a value to be used throughout the rest of the program.

Sections:

I. RETURNING A VALUE .... PAGE 1
II. RETURN TYPES .... PAGE 3
III. METHODS WITH A RETURN VALUE & PARAMETERS .... PAGE 5

I. RETURNING A VALUE:

So far, our methods have been somewhat static. At the most, we have passed in arguments to a method, that method processed those arguments, then that method outputted something to the screen. Now we are going to have our method **return a value.** Let's look at an example:

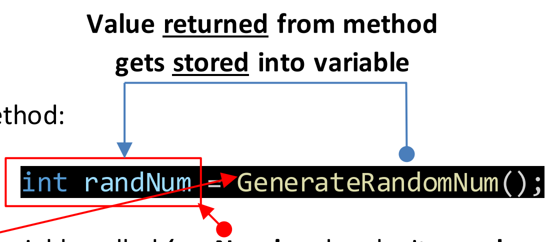
```
int GenerateRandomNum()
{
    Random rnd = new Random();
    int num  = rnd.Next(1, 11);
    return num;
}
int randNum = GenerateRandomNum();
Console.WriteLine(randNum);
```

Sample **Output:**

```
6
```

Notice our method is no longer being declared with 'void', but rather with **'int'.** This indicates that this method must **return** an **integer value.** We return a value from a method using the keyword **'return'** followed by a value that is of the proper **return type.** In this example, we are returning the integer variable **'num'** which is a random number generated between 1 and 10.

Notice how we **call** this method:



```
int randNum = GenerateRandomNum();
```

We create an integer variable called **'ranNum'** and make it **equal** our method call **GenerateRandumNum().** After our **GenerateRandumNum()** executes, it **returns** an **integer value** (i.e., a random number between 1 and 10) which then gets stored in the **'randNum'** variable. This number is then outputted to the screen:

```
Console.WriteLine(randNum);
```

Alternatively, we could **output** the return value **directly** without having to first store it into a variable. For example:

```
Console.WriteLine(GenerateRandomNum());
```

The sequence here is straightforward: our GenerateRandomNum() method is called first, it returns a value, then that value is outputted to the console with Console.WriteLine(). **Note:** What we have here is a **nested method call.** When nesting our methods calls, the innermost method always gets called first (in our case GenerateRandomNum() gets called first before Console.WriteLine()).

II. RETURN TYPES:

Our last example in the previous section demonstrated a method that returned an integer value. We can in fact return a value of any type we wish. For example:

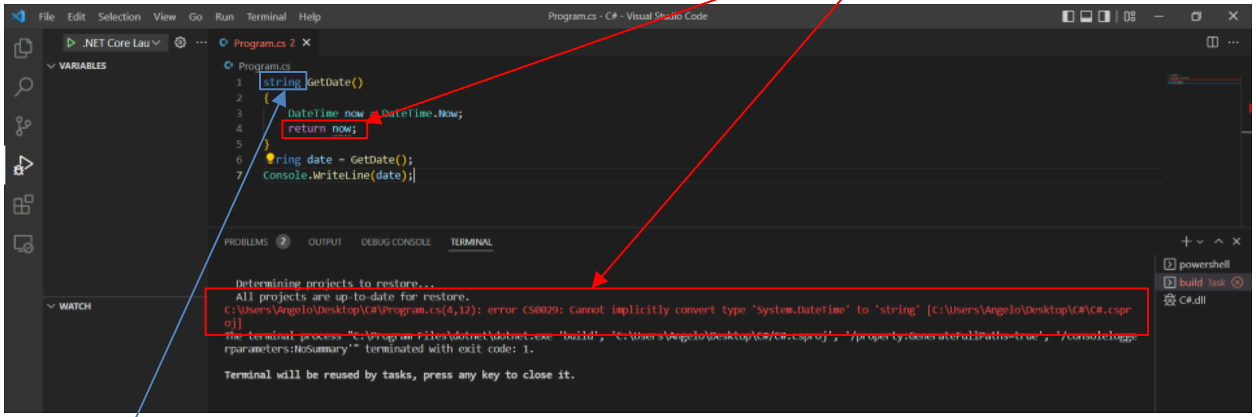
```
string GetDate()
{
    DateTime now = DateTime.Now;
    return now.ToString();
}
string date = GetDate();
Console.WriteLine(date);
```

Sample **Output:**

```
2022-06-08 1:44:03 PM
```

In the above example, our method **GetDate()** returns a **string data type.**

It is important that you return the proper data type, or you will get a **syntax/build error,** for example:



```
string GetDate()
{
    DateTime now = DateTime.Now;
    return now;
}
string date = GetDate();
Console.WriteLine(date);
```

```
Determining projects to restore...
All projects are up-to-date for restore.
C:\Users\Angelo\Desktop\C#\Program.cs(4,12): error CS0029: Cannot implicitly convert type 'System.DateTime' to 'string' [C:\Users\Angelo\Desktop\C#\C#.cspr
oj]
The terminal process "C:\Program Files\dotnet\dotnet.exe 'build', 'C:\Users\Angelo\Desktop\C#/C#.csproj', '/property:GenerateFullPaths=true', '/consolelogge
rparameters:NoSummary'" terminated with exit code: 1.

Terminal will be reused by tasks, press any key to close it.
```

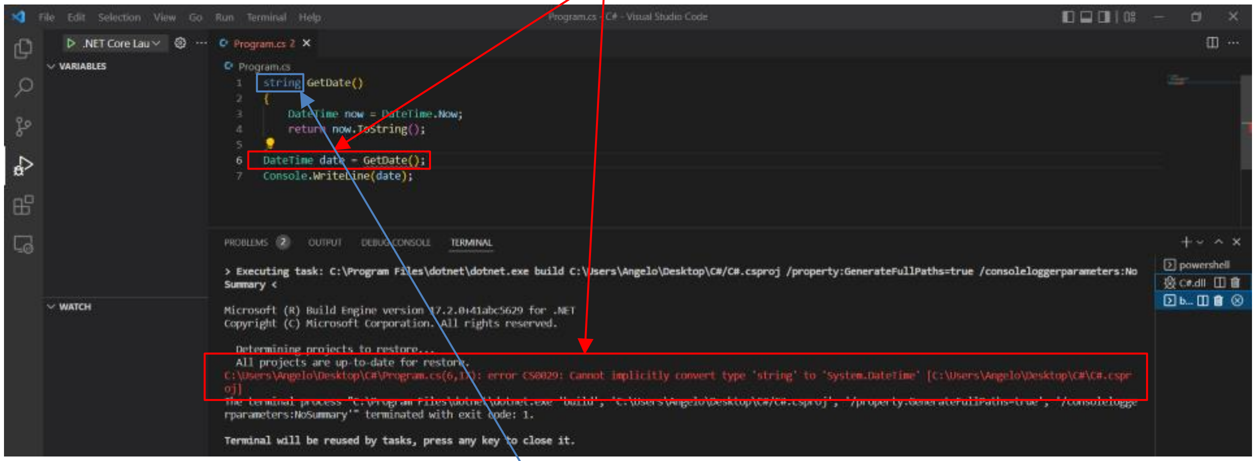
The variable **'now'** is of the special data type **DateTime.** Since the method declaration has a **'string' return type,** then we must return a **string** data type **not** a **DateTime** data type. Therefore, with our example above we must do:

```
return now.ToString();
```

not:

```
~~return now;~~
```

Similarly, our variable we use to store the value returned by a method must be of the same data type of the returned value or we will get a **syntax/build error.** For example:



```
string GetDate()
{
    DateTime now = DateTime.Now;
    return now.ToString();

DateTime date = GetDate();
Console.WriteLine(date);
```

```
> Executing task: C:\Program Files\dotnet\dotnet.exe build C:\Users\Angelo\Desktop\C#/C#.csproj /property:GenerateFullPaths=true /consoleloggerparameters:No
Summary <

Microsoft (R) Build Engine version 17.2.0+41abc5629 for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

  Determining projects to restore...
  All projects are up-to-date for restore.
C:\Users\Angelo\Desktop\C#\Program.cs(6,17): error CS0029: Cannot implicitly convert type 'string' to 'System.DateTime' [C:\Users\Angelo\Desktop\C#\C#.cspr
oj]
The terminal process "C:\Program Files\dotnet\dotnet.exe 'build', 'C:\Users\Angelo\Desktop\C#/C#.csproj', '/property:GenerateFullPaths=true', '/consolelogge
rparameters:NoSummary'" terminated with exit code: 1.

Terminal will be reused by tasks, press any key to close it.
```

The method declaration has a **'string' return type.** Therefore, our variable **'date'** should be declared as a **string** and **not** as a **DateTime.** Therefore, with our example above we must do:

```
string date = GetDate();
```

not:

~~`DateTime date = GetDate();`~~

To summarize:

- When you do not wish to return a value, then declare your method with **'void'.** For example:
    - `void OutputDate()`
    - `void WelcomeMessage()`

- You can return a value of any data type from a method. For example, here are some methods with different return types:
    - `double GetTax()`
    - `DateTime GetDate()`
    - `string GetDate()`
    - `int GetRandomNumber()`

III. METHODS WITH A RETURN VALUE & PARAMETERS:

You can have a method that **returns a value** and **receives arguments.** For example:

```
int GenerateRandomNum(int start, int end)
{
    Random rnd = new Random();
    int num = rnd.Next(start, end + 1);
    return num;
}
int randNum = GenerateRandomNum(1, 10);
Console.WriteLine(randNum);
```

Sample **Output:**

```
4
```

Our method above has been declared to **return** an **int,** and has been declared with two integer **parameters 'start'** and **'end'.** The method uses these parameters as the range for the random number to be generated (we add **'1'** to **'end'** since the **.Next()** method generates a number up to but not including the end parameter). Once an appropriate number is generated, it is **returned.**

When we call this method, we pass arguments **'1'** and **'10'** to the parameter's **'start'** and **'end'** respectively. The value **returned** from this method is then stored into the integer variable **'randNum'**:

```
int randNum = GenerateRandomNum(1, 10);
```

Value returned from method
gets stored into variable

To summarize:

- Your method can contain as **many parameters** of **any data type** as you wish
- Your method can **return** only **one value** of any **data type**