Return & Output

Getting something out of a function

Return keyword

Using the **return** keyword, you can end a Method Execution right away:

```
void SayManyThings() {
        Console.WriteLine(“I can say one thing.”);
        return;
        Console.WriteLine(“But can I say many things?”);
}
```

Output: I can say one thing.

Useful, with conditions

This can be very useful, if you have any exit conditions:

```
void StoryTime() {
        Console.WriteLine(“Do you want to hear a story?);
        if(Console.ReadLine() == “Nope”) {
                return;
        }
        Console.WriteLine(“Glad to hear that. Once upon a [...]”);
}
```

Or…

For a game?

```
void NimTurn() {
	PlayerTurn();
	if(matches <= 0) {
		// AI can not draw matches anymore, if the Game is over already.
		return;
	}
	AITurn();
}
```

Return types

If we change the **return type** of our function, it **ALLOWS** us and at the same time **FORCES** us to **return** a value from our function.

This way, whoever called this function is guaranteed that a value will be received:

Remember the syntax of a function?

```
RETURN_TYPE  FUNCTION_NAME (PARAMETER LIST)
{ // Function body / scope start
  // <- Put the code of the function here..
} // Function body / scope end
```

Anything but void

If we use anything but **void** as a **Return Type**, it will force us to use the **return** keyword to return something:

```
int GetFive() {
   // no return...
} // Error: Not all code paths return a value
```

To fix this, we need to make sure that we return a value of the correct type: **int**, from this function:

```
int GetFive() {
   return 5;
}
```

Which allows us..

Cool, now that it returns something, it allows us to use the function where ever we like:

```
int five = GetFive();
Console.WriteLine(five);           // 5
Console.WriteLine(GetFive());      // also 5
```

Note: **All** code paths have to have a return value:

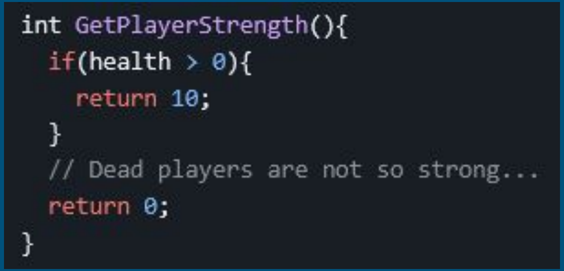
```
int GetPlayerStrength() {
    if(health > 0) {
        return 10;
    }
} // Error: Not all code paths return a value
```

Avoiding unexpected behavior

This is to avoid unexpected behavior when assigning the result:

int health = 0;
int strength = GetPlayerStrength();

What would be returned from the function in this case?
When the health is 0, nothing is returned after all.. since this is uncontrolled and unintended, C# forces us to return a value on all code paths:

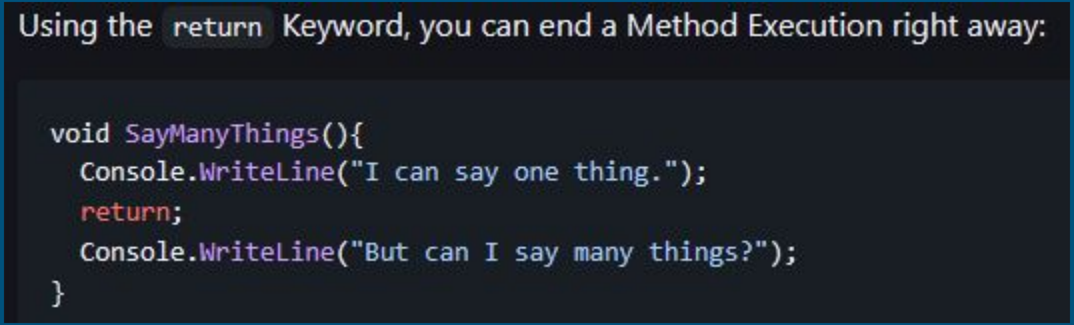


```
int GetPlayerStrength(){
  if(health > 0){
    return 10;
  }
  // Dead players are not so strong...
  return 0;
}
```

But wait??

Won´t the code sample return two values, if the player has a health of 100? Won´t it return 10 first and then 0?

No, it won´t, since the return keyword causes the execution of the method to be stopped immediately!


Using the `return` Keyword, you can end a Method Execution right away:

```
void SayManyThings(){
  Console.WriteLine("I can say one thing.");
  return;
  Console.WriteLine("But can I say many things?");
}
```

Goal 1 - BuyMyGamePlix()

- Write a function that asks the user to buy a game
- If the user enters “Yes” the function says “Thank You!” and returns
- Otherwise the function asks again

Goal 2 - CountDown again..

- Write the countdown goal again, which recursively invokes itself
- But instead of checking if the remaining timer is > 0, and if true, invoking itself, implement it the other way around:
    - if the timer is not > 0, have it return

Example, before:

```
if (health < 3) {
   HealAgain();
}
```

After:

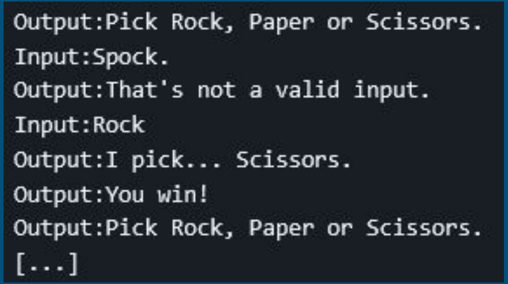
```
if (health == 3) {
   return;
}
HealAgain();
```

Goal 3 - Make the message appear

Look at the code sample below. Fix it by replacing the comment with code to make the magic message appear.

```
void MagicMessage() {
        Console.WriteLine(“You’re trying to find the magic message.”);
        // replace this comment with code…
        return;
        Magic:
        Console.WriteLine(“You found the magic message.”);
}
MagicMessage();
```

Goal 4 - Return Value


```
Output:Pick Rock, Paper or Scissors.
Input:Spock.
Output:That's not a valid input.
Input:Rock
Output:I pick... Scissors.
Output:You win!
Output:Pick Rock, Paper or Scissors.
[...]
```

- Implement a Rock-Paper-Scissors mini game
- Write one function that returns the player's choice
  - but only after the player has made a choice
- Write one function that returns the AI's choice
- Use those functions in you core game loop

Goal 5 - MyFunction

- Create a function and call it **MyFunction**
    - return type **void**
    - no parameters
    - when called
        - ask the user for his name, only continue if his name is not “Neo”
        - ask the user for his age, only continue if it is >= 18
        - ask the user if he wants to enter, only continue if he says “Yes”
        - ask the user if he wants to turn back, only continue if he says “No”
        - say: “Congratulations, you made it in!”
- Make sure to call the function **MyFunction**
- **Remember to use the return keyword**