



# More Functions and Returns!

# Multiple Returns

What will be printed by the following program?

```
def mystery(x):  
    return x * 2  
    return x / 2  
  
print(mystery(5))
```



# Multiple Returns

What will be printed by the following program?

```
def mystery(x):  
    return x * 2  
    return x / 2  
  
print(mystery(5))
```

10



# Multiple Returns

What will be printed by the following program?

```
def mystery(x):  
    return x * 2  
    return x / 2  
  
print(mystery(5))
```

10

This happens because when we `return` a value, the **Function** is instantly going to end!  
Is there a way to actually `return` multiple values in Python?



# Multiple Returns

What will be printed by the following program?

```
def mystery(x):  
    return x * 2, x / 2  
  
print(mystery(5))
```

```
(10, 2.5)
```



# Saving the Multiple Returns

If we want to save the values that are `returned` from a `Function`, we can **assign** the `Function` call to multiple variables!

```
def mystery(x):  
    return x * 2, x / 2  
  
num1, num2 = mystery(5)  
print(str(num1) + " " + str(num2))
```

```
10, 2.5
```





and Namespaces

# Namespace

According to CodeHS:

A namespace is the collection of variable names that exist at a certain point in your code. Names don't exist throughout the entire program, they only exist within a certain *namespace*.





# Scope

A variable's scope refers to where the variable exists within a program. If a variable doesn't exist at a certain place, then it is “out of scope”.

We could also say that a variable's scope determines which *namespace* a variable exists inside.



# Please explain that in real english

Think of namespace as a box where a variable's name exists. The biggest box is **everywhere** - the variable can be seen anywhere in your program. Any variable in the **everywhere** namespace is also called a *global variable*.

Anytime you create a **Function**, you make another box, and any variables created inside the **Function** go into that box. Any variable inside a **Function**'s namespace a *local variable* to that **Function**.



# An example!

```
x = 5
```

```
def change_x():
```

```
    x = 10
```

```
    print(x)
```

```
print(x)
```

```
change_x()
```

```
print(x)
```



# An example!

```
x = 5
```

```
def change_x():  
    x = 10  
    print(x)  
print(x)  
change_x()  
print(x)
```

Global	
x	5



# An example!

```
x = 5
```

```
def change_x():  
    x = 10  
    print(x)
```

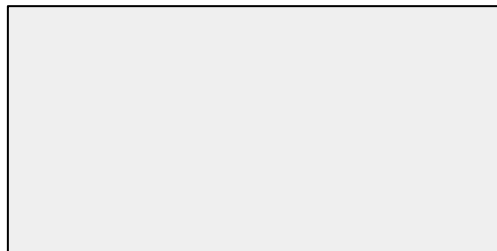
```
print(x)
```

```
change_x()
```

```
print(x)
```

Global	
x	5

change_x()	
x	



# An example!

```
x = 5
```

```
def change_x():
```

```
    x = 10
```

```
    print(x)
```

```
print(x)
```

```
change_x()
```

```
print(x)
```

Global	
x	5

change_x()	
x	

5
---



# An example!

```
x = 5
```

```
def change_x():
```

```
    x = 10
```

```
    print(x)
```

```
print(x)
```

```
change_x()
```

```
print(x)
```

Global	
x	5

change_x()	
x	

5
---



# An example!

```
x = 5
```

```
def change_x():
```

```
    x = 10
```

```
    print(x)
```

```
print(x)
```

```
change_x()
```

```
print(x)
```

Global	
x	5

change_x()	
x	10

5
---





# An example!

```
x = 5
```

```
def change_x():
```

```
    x = 10
```

```
    print(x)
```

```
print(x)
```

```
change_x()
```

```
print(x)
```

Global	
x	5

change_x()	
x	10

5
10



# An example!

```
x = 5
```

```
def change_x():
```

```
    x = 10
```

```
    print(x)
```

```
print(x)
```

```
change_x()
```

```
print(x)
```

Global	
x	5

change_x()	
x	

5
10
5



# Note about Scope

Any variable that is in the **global** namespace can be **seen** inside any **Function**. Only when the variable is **edited** does a new variable get created inside the **Function's** namespace.

```
x = 5
def print_x():
    print(x)

print_x()
```



# Note about Scope II

```
def print_x():  
    print(x)  
    x = 10  
    print(x)
```

```
x = 5  
print_x()
```



# Note about Scope II

```
def print_x():  
    print(x)  
    x = 10  
    print(x)
```

UnboundLocalError: local variable 'x' referenced before assignment

```
x = 5  
print_x()
```



# Note about Scope II

```
def print_x():  
    print(x)  
    x = 10  
    print(x)
```

```
x = 5  
print_x()
```

UnboundLocalError: local variable 'x' referenced before assignment

This happens because when the interpreter reads through the **Function definition**, it knows that there is going to be a **local** variable `x` defined *somewhere* within. If we try to reference a variable before it's been **initialized**, we'll get an error!

