

# ECTTP: Functions

Valentijn Muijrs  
<https://github.com/vmuijrs/ECTTP>

•

•

# Course Overview

- Week One: Course overview
- Week One: Variables
- Week Two: Conditions
- Week Three: Loops
- **Week Four: Functions ←**
- Week Five:
- Week Six:
- Week Seven: (Files, Exceptions, IO)
- **First Test!**
- Week Eight: Lists
- Week Nine: Classes and Objects
- Week Ten:
- Week Eleven:
- **Second Test!**

# Our Super Powers so far...

- Variables! (Int, String, Boolean and Float)
- Mathematical Operators (+, \*, -, /)
- Boolean Operators (and or not, >, <, ==, >=, <=, !=)
- If- statements!
- Modulo %
- Random( x, y )
- For-loops
- While-loops



# While-loop

- While some condition is true, execute the code

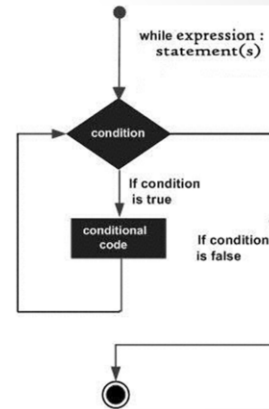
```
x = 0
```

```
while x < 5 :
```

```
    x = x + 1
```

```
    print("I am looping!")
```

```
print("Aaaaand we'r e done")
```



# For-loop

The for-loop let's the code run within a range, in this way the code is executed a limited amount of times

```
for i in range ( 3, 6 ):
    print ("I am printed 3 times")
```

i gets the value 3  
then prints  
i gets the value 4  
then prints  
i gets the value 5  
then prints  
then the for-loop exits

•

•

# Function

A function stores code which can be reused.

```
def thing():
```

```
    print "Hello"
```

```
    print "Fun"
```

```
thing()
```

```
print "Zip"
```

```
thing()
```

Output:

Hello  
Fun

Zip  
Hello  
Fun



# Python Functions

- There are two kinds of functions in Python
  - **Built-in** functions that are provided as part of Python:
    - `raw_input()`, `type()`, `float()`, `int()`, `print()`...
  - Functions that we **define ourselves** and then use
- We treat the built-in functions and our own function names as **reserved words** (don't use them as variable names)

# Function Definition

- In Python a functions is a piece of reusable code that can take **argument(s)** as **input**, does some computation and then **returns** a result or results
- We define a **function** using the **def** reserved word
- We call/invoke the function by using the function name, parenthesis and arguments in an expression



# Defining a function

```
def addTwoValues( x , y ):
    return x + y
```

- 'def' shows that we are defining a new function
- "addTwoValues" is the name of our function
- x and y are the parameters for this function
- The "return" keyword returns the value to the original place from where the function was called, if the function does not return a value, the return keyword can be omitted

# Defining a function

```
def addTwoValues( x , y ):
    return x + y
```

- The body of the function is **indented** in Python
- The definition of the function does **not execute** the body of the function
- A **function call** (or invoke) executes the body of the function
- Do not forget the colon at the end ':'

Example of a function call:

addTwoValues ( 3, 6 ) >>> this will return the value 9

•

•

# Arguments

- An **argument** is a value we pass into the function as its **input** when we call the function
- We use **arguments** so we can direct the function to do **different kinds of work** when we call it at different times
- We put the arguments in **parenthesis** after the name of the function

# Using a Function

`big = max("Hello World")`

Function call

argument

`big = "W"` → the result of the function call is assigned to the variable

# Parameters

- A **parameter** is a variable which we use in the function **definition** that is a “handle” that allows the code in the function to access the arguments for a particular function invocation.

```
def greet( lang ):
    if lang == 'es':
        print 'Hola'
    elif lang == 'fr':
        print 'Bonjour'
    else:
        print 'Hello'
```

```
greet('en') → Hello
greet('es') → Hola
greet('fr') → Bonjour
```

# Multiple Parameters

```
def addTwo( a, b):  
    added = a + b  
    return added  
x = addTwo ( 3, 5)  
print x → 8
```

- We can define more than one parameter in the function definition
- We simply add more arguments when we call the function
- We **match** the **number** and **order** of arguments and parameters

# Return Values

- Often a function will take its arguments, do some computation and return a value to be used as the value of the function call in the calling expression. The **return** keyword is used for this.

```
def greet():  
    return "Hello"
```

```
print ( greet()+ "Glenn") → Hello Glenn  
print ( greet()+ "Sally" ) → Hello Sally
```

•

•

# Return Value

- A “fruitful” function is one that **produces** a result (or **returns** a value)
- The **return** statement ends the function execution and “**sends back**” the result of the function

•

•



# Void Functions

- When a function does not return a value, we call it “void”
- Functions that return values are “fruitful” functions
- Void functions are “not fruitful”



# When to use functions

- **Organize** your code into "paragraphs" - capture a complete thought and "name it"
- **Don't repeat yourself** - make it work once and then reuse it as a function
- If something gets too long or **complex**, break up logical chunks and put those chunks in functions
- Make a library of **common code** that you do over and over - perhaps share this with your friends...

•

•

# Exercise

- Rewrite your pay computation with time-and-a-half for overtime ( you work 40 hours a week, the rest is overtime) and create a function called **computePay** which takes two parameters ( hours and rate).
- Enter Hours: 45
- Enter Rate: 10
- Pay: 475.0
- Result:  $40 * 10 + 5 * 15 = 475$

# Solution

```
def computepay ( hours, rate )  
    overtime = 0  
    if hours > 40:  
        overtime = hours - 40  
    return (hours - overtime) * rate + 1.5 *rate *overtime
```

Test your function:

computepay ( 0, 15 ) → 0

computepay ( 45 , 10 ) → 475

computepay ( 30 , 20 ) → 600

•

•

# Fifth lab is online

[https://github.com/vmuijters/ECTP/blob/master/Labs/Lab\\_5.md](https://github.com/vmuijters/ECTP/blob/master/Labs/Lab_5.md)

#For examples/tutorials and references!  
[py.processing.org](http://py.processing.org)

#For more practise with python!  
[codecademy.com](http://codecademy.com)

#Now let's practise some more with codingbat:  
<http://codingbat.com/python>

•

•