Lecture 5 – Python

Mr. Yousif G. Arshak

University of Zakho

Computer Science Department

yousif.arshak@uoz.edu.krd

OUTLINES

- Python Functions
- Python Lambda

Python Functions

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

Creating a Function

- In Python a function is defined using the def keyword:

Example

```
def my_function():
  print("Hello from a function")
```

Calling a Function

- To call a function, use the function name followed by parenthesis:

  Example

```
def my_function():
  print("Hello from a function")

my_function() #Hello from a function
```

Arguments

- Information can be passed into functions as arguments.
- You can add as many arguments as you want, just separate them with a comma.
- Example

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

```
Output:
Emil Refsnes
Tobias Refsnes
Linus Refsnes
```

Note: Arguments are often shortened to *args* in Python documentations.

Parameters or Arguments?

- The terms *parameter* and *argument* can be used for the same thing: information that are passed into a function.
- From a function's perspective:
  - A parameter is the variable listed inside the parentheses in the function definition.
  - An argument is the value that is sent to the function when it is called.

Number of Arguments

- By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.
- Example: This function expects 2 arguments, and gets 2 arguments:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")   #Emil Refsnes
```

Arbitrary Arguments, *args

- If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
- This way the function will receive a tuple of arguments, and can access the items accordingly:

  Example: If the number of arguments is unknown, add a * before the parameter name:

- ```
  def my_function(*kids):
    print("The youngest child is " + kids[2])
  my_function("Emil", "Tobias", "Linus") # The youngest child is Linus
  ```

Arbitrary Arguments are often shortened to **args* in Python documentations.
  ```

Default Parameter Value

- If we call the function without argument, it uses the default value:
  - ```
    def my_function(country = "Norway"):
      print("I am from " + country)

    my_function("Sweden")   I am from Sweden
    my_function("Belgium")  I am from Belgium
    my_function()           I am from Norway
    my_function("Brazil")   I am from Brazil
    ```
    ```

Return Values

- To let a function return a value, use the return statement:

  Example

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

```
15
25
45
```

Python Lambda

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.
- The power of lambda is better shown when you use them as an anonymous function inside another function.
- Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

Syntax

lambda arguments : expression

The expression is executed and the result is returned:

Example

- Add 10 to argument a, and return the result:
  - ```
    x = lambda a : a + 10
    print(x(5)) #15
    ```
- Multiply argument a with argument b and return the result:
  - ```
    x = lambda a, b : a * b
    print(x(5, 6)) #30
    ```
- Summarize argument a, b, and c and return the result:
  - ```
    x = lambda a, b, c : a + b + c
    print(x(5, 6, 2)) #13
    ```
  -
    ```

Lambda with anonymous function

- ```
  def myfunc(n):
    return lambda a : a * n
  mydoubler = myfunc(2)
  print(mydoubler(11)) #22
  ```

- ```
  def myfunc(n):
    return lambda a : a * n

  mydoubler = myfunc(2)
  mytripler = myfunc(3)

  print(mydoubler(11)) #22
  print(mytripler(11)) #33
  ```