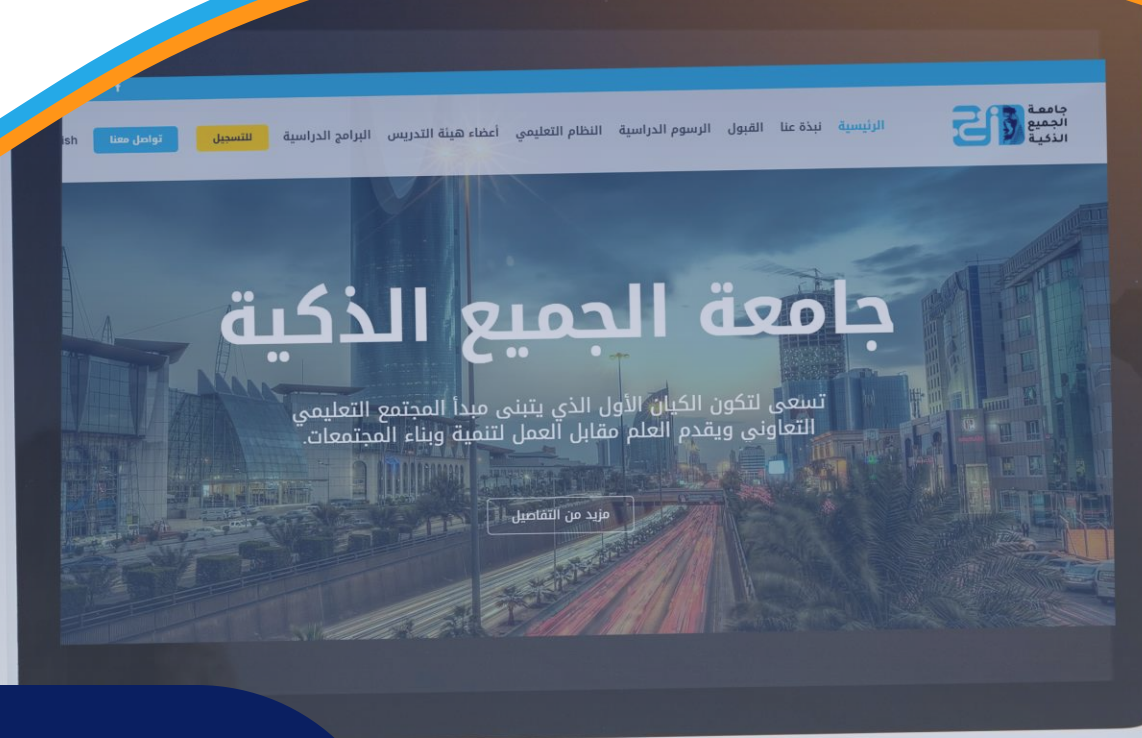




جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY



Functions and Modules
Prepared by Dr. Hamada I. AbdulWakel
Lecture 7

CYB550
Programming Foundations In
Python

- Know the function with default arguments and when to use it.
- Deal with inner and high-order functions.
- Explain the use of anonymous functions
- Explain what is the file system and how to access it.
- Use the concept of modules.

- Function names are references to functions and that we can assign multiple names to the same function

```
def increment(x):  
    return x + 1
```

```
increase = increment  
print(increase(5))
```

- Example1:

```
def greet(name, msg):  
    """This function greets to  
    the person with the provided message"""  
    print("Hello", name + ', ' + msg)
```

```
greet("Monica", "Good morning!")
```

```
greet() #TypeError: greet() missing 2 required positional arguments
```

- Arguments provide the function's caller with the means of transmitting information to the function.
- Programmer can specify optional arguments with default values in any function definition:

```
def function_name(<required args>, <key1> = <val1>, .....)
```

- Following the required arguments are **one or more default or keyword arguments**.
- When function is called with these arguments, default values are overridden by caller's values.

- Example2:

```
def greet(firstname="Mr.", lastname="Who"):
    print("Hello,", firstname, lastname)
```

```
greet()
```

```
greet("Ahmed")
```

```
greet(lastname="Mohamed")
```

```
greet("Ahmed", "Aly")
```



- Any number of arguments in a function can have a default value. But once we have a default argument, **all the arguments to its right must also have default values.**

```
def greet(msg = "Good morning!", name): #SyntaxError: non-default  
argument follows default argument
```

- Python allows functions to be called using **keyword arguments**. When we call functions in this way, the order (position) of the arguments can be **changed**.

```
greet(name = "Ahmed", msg = "How do you do?")
```

```
greet(msg = "How do you do?", name = "Aly") # (out of order)
```

```
greet("Mahmoud", msg = "How do you do?")
```

```
greet(name="Zeyad", "How do you do?") #SyntaxError: non-keyword arg  
after keyword arg
```



- Example3:

```
def accessArguments(x, y = 1, z = 2):  
    Print("X= ", x, "Y= ", y, "Z= ", z)
```

```
accessArguments(3)
```

```
accessArguments(3, 10)
```

```
accessArguments(3, z = 5)
```

```
accessArguments(3, z = 4, y = 6)
```

- Sometimes, we **do not know** in advance the number of arguments that will be passed into a function.
- In the function definition, we use an asterisk (*) before the parameter name to denote this kind of argument.

- Example4:

```
def greet(*names):  
    for name in names:  
        print("Hello," , name)  
  
greet("Ahmed", "Mohamed", "Aly")
```

- A function can be defined and called inside another function.
- This kind of function has direct access to variables and names defined in the enclosing function.

- Example5:

```
def outer_func(who):  
    def inner_func():  
        print("Hello,", who)  
    inner_func()
```

```
outer_func("Hamada")
```

- Example6: calculate factorial to a number using inner function

```
def factorial(number):  
    def inner_factorial(number):  
        if number <= 1:  
            return 1  
        return number * inner_factorial(number - 1)  
    return inner_factorial(number)
```

- A function can take **one or more functions** as arguments.
- A function can be **returned** as a result of another function.

- Example7:

```
def loud(text):  
    return text.upper()  
def quiet(text):  
    return text.lower()  
def hello(func):  
    text = func("Hello")  
    print(text)  
hello(loud)
```



- Example8: 10 (dividend) / 2 (divisor) = 5 (quotient)

```
def divisor(x):  
    def dividend(y):  
        return y / x  
    return dividend
```

```
divide = divisor(2)  
print(divide(10))
```



- Example9:

```
def add_two_nums(x, y):  
    return x + y  
def add_three_nums(x, y, z):  
    return x + y + z  
def get_appropriate_function(num_len):  
    if num_len == 3:  
        return add_three_nums  
    else:  
        return add_two_nums
```

```
args = [1, 2, 3]
```

```
num_len = len(args)
```

```
res_function =  
get_appropriate_function(num_len)
```

```
print(res_function)
```

```
print(res_function(*args))
```



- An anonymous function is a function that is defined **without a name**.
- Anonymous functions are defined using the **lambda** keyword.
- Lambda functions can have any number of arguments but **only one expression**.

lambda arguments: expression

- A lambda function is used when we require a nameless function for a short period of time.
- It is used as an argument to a higher-order function.

- Example10:

```
double = lambda x: x * 2
```

```
print(double(5))
```

This function has no name. It returns a **function object** which is assigned to the identifier **double**. We can now call it as a normal function.

- Example10:

```
double = lambda x: x * 2
```

Is equal to

```
def double(x):  
    return x * 2
```



- Example11: write a lambda function to filter out only the even items from a list.

```
my_list = [1, 5, 4, 6, 8, 11, 3, 12]
```

```
new_list = list( filter(lambda x: (x%2 == 0) , my_list) )
```

```
print(new_list)
```

- Example12: write a lambda function to double each item in a list using map()
-
- `my_list = [1, 5, 4, 6, 8, 11, 3, 12]`
-
- `new_list = list( map(lambda x: x * 2 , my_list) )`
-
- `print(new_list)`

- Module is a file contain a set of functions.
- Add module to your program to help doing some tasks to save time and effort.
- Your program can add/call multiple modules.

- Calling modules in a program:
`import module_name1, module_name2, .....`

```
import random
```

```
print(random)
```

```
print(random.random())
```

```
print(dir(random))
```

- Calling a specific function from a module:
`from module_name import function_name1, function_name2, ....`

`from random import randint`

`print(random.randint(100, 500))` #get an error

`print(randint(100, 500))`

- This module provides functions for:
 - Creating and removing a directory (folder),
 - Fetching its contents, and
 - Changing and identifying the current directory.

- Getting current working directory:

```
import os  
print(os.getcwd())
```

- Creating a directory:

```
os.mkdir("/mnt/FCI/NewFolder")  
os.mkdir("NewFolder")
```

- Changing the current working directory:
`os.chdir("/mnt/FCI")`
`os.getcwd()`
- Removing a directory:
`os.rmdir("/mnt/FCI/NewFolder")` #you can't remove the current working directory

- Listing files and sub-directories in the current working directory:

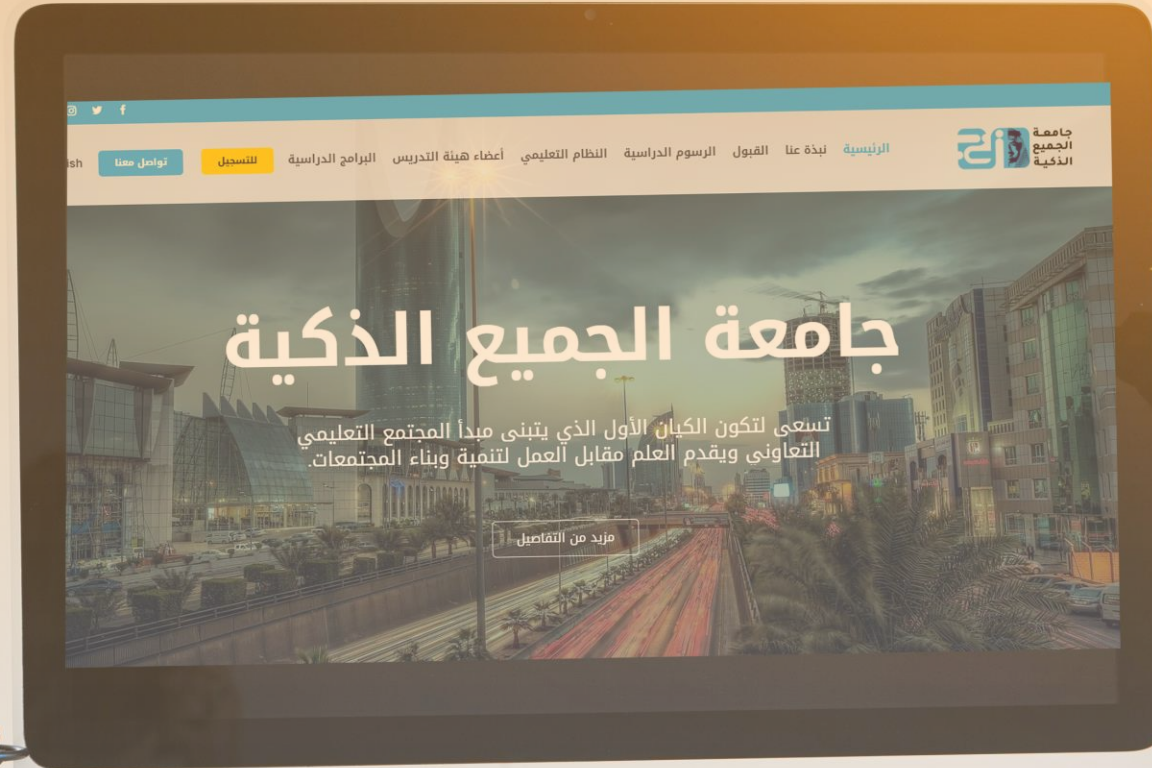
```
os.listdir("/mnt/FCI")
```

```
os.listdir()
```

- Removing a directory:

```
os.rmdir("/mnt/FCI/NewFolder")
```

#you can't remove the current working directory



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

THANK YOU