

# M110: Python Programming

## Meeting #7

### Functions



**AOU**

الجامعة العربية المفتوحة  
Arab Open University  
Lebanon



Prepared by Dr. Ahmad Mikati

# Content

- Introduction to Functions
- Defining and Calling a Void Function
- Designing a Program to Use Functions
- Local Variables
- Passing Arguments to Functions
- Global Variables and Global Constants
- Reusing Code

# Modular Programming

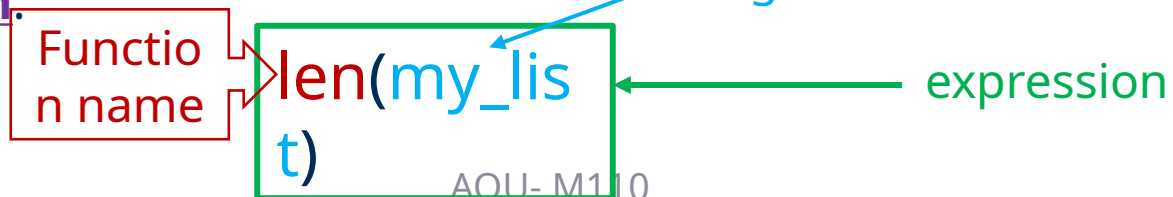
- Modular programming is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules.
- Modular Programming helps develop programs that are much easier to read since they can be enabled as user-defined functions.
- A program that carries multiple functions benefits in the following ways:
  - Simpler Code
  - Code Reuse
  - Faster Development
  - Better Testing
  - Better Teamwork

# Introduction to Functions

- Most programs perform tasks that are large enough to be broken down into several subtasks.
- For this reason, programmers usually break down their programs into small manageable pieces known as functions.
- A function is a group of statements that exist within a program for the purpose of performing a specific task.
- Instead of writing a large program as one long sequence of statements, it can be written as several small functions, each one performing a specific part of the task.
- These small functions can then be executed in the desired order to perform the overall task.

# Functions in Python

- You may recall being told that you can find the size of a list `my_list` by using `len(my_list)`. This involved a Python function, `len()`.
- The `len()` function is a built-in Python function: the Python interpreter will know how to handle it, without you needing to add any additional code to your program.
- When we talk about a function, we use its **name** – in this case, `len` – followed by a pair of parentheses: `len()`. This way, you can see at once that we are talking about a function rather than, for instance, a variable.
- In addition to a name, a function can have **arguments** (though not all functions have arguments). In the example `len(my_list)`, the variable ***my\_list*** is the argument of the function.
- When a function name is combined with its argument, the result is an **expression**.



# Functions in Python

## Creating a Function

In Python a function is defined using the **def** keyword. Here is the general format of a function definition in Python:

```
def function_name():  
    block (a set of statements)
```

The first line is known as the function header. It marks the beginning of the function definition.

The function header begins with the key word **def**, followed by the name of the function, followed by a set of parentheses, followed by a colon.

```
def printHello():  
    print("Hello! ")
```

Beginning at the next line is a set of indented statements known as a block. A block is simply a set of statements that belong together as a group. These statements are performed any time the function is executed.

# Functions in Python

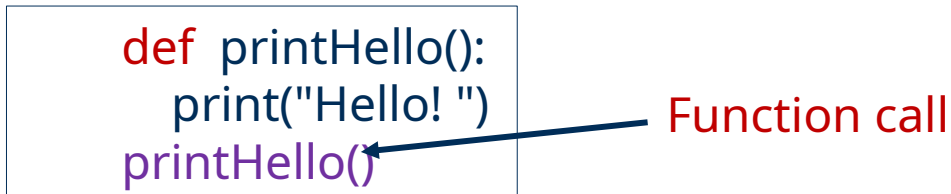
## Calling a Function

A function definition specifies what a function does, but it does not cause the function to execute.

To execute a function, you should call it.

```
def printHello():  
    print("Hello! ")  
printHello()
```

Function call



When a function is called, the interpreter jumps to that function and executes the statements in its block.

When the end of the block is reached, the interpreter jumps back to the part of the program that called the function, and the program resumes execution at that point.

# Functions

## Four parts of function's definition:

1. The reserved word **def** which indicates that what follows is a function definition.
2. The **name** of the function.
3. List of **arguments** enclosed in parentheses.
4. The **body** of the function, which is the set of statements that carry out the work of the function, noting that:
  - Use colon ( : ) to start the body of the function.
  - Use indentation for the body of the function .

```
def printHello():  
    print("Hello! ")
```



# Functions in Python

## Argument

- S** Information can be passed into functions as arguments. **Arguments** are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (*aname*).

```
def my_function(aname):  
    print('Hello, ' + aname )
```

```
name=input('Enter a  
name: ')  
my_function(name)
```



```
Enter a name:  
Jamal  
Hello, Jamal
```

## Number of Arguments

By default, a function must be called with the correct number of arguments. This means that if your function expects 2 arguments, you should call the function with 2 arguments, not more, and not

# Functions in Python

## Passing a List as an Argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

```
def my_function(names):  
    for x in names:  
        print(x)  
  
families = ["Badr", "Jomaa", "Hamad"]  
my_function(families)
```

```
1  def my_function(names):  
2      for x in names:  
3          print(x)  
4  
5  families = ["Badr", "Jomaa", "Hamad"]  
6  my_function(families)  
7
```

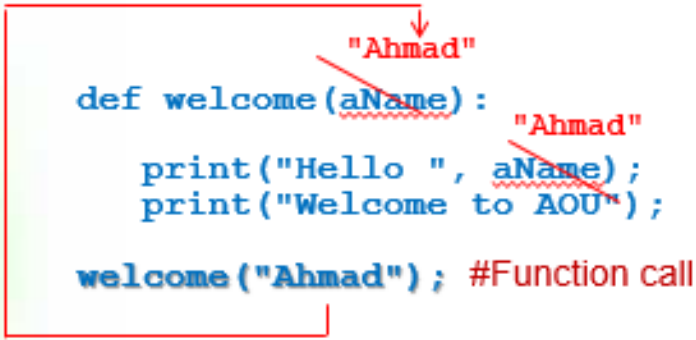
```
Badr  
Jomaa  
Hamad
```

# Functions without return

- Some functions perform simple procedural tasks (specified in their bodies) but do not return any information when they are called.
- Example: Write a function that will display a welcome message to a student with his/her name. Then, use the function in your program.

```
def welcome(aName):  
    print("Hello" , aName)  
    print("Welcome to AOU")  
  
welcome("Ahmad")  #Function call
```

- When the function is called, an actual value for the argument must be used.
- For example, when the function call `welcome('Ahmad')` is executed, the actual string 'Ahmad' replaces the argument `aName` resulting in the following output:



```
def welcome(aName):  
    print("Hello ", aName);  
    print("Welcome to AOU");  
  
welcome("Ahmad"); #Function call
```

The diagram illustrates the function call process. A red arrow points from the string "Ahmad" in the function call `welcome("Ahmad");` to the parameter `aName` in the function definition `def welcome(aName):`. Another red arrow points from the same `aName` parameter to the `aName` argument in the `print("Hello ", aName);` statement within the function body.

```
-----output-----  
Hello Ahmad  
Welcome to AOU
```

# Functions with return

- Some functions provide a value as the result of some calculations made in the function's body.
- Python provides us with the *return* statement:  
*return* followed by the value it needs to return
- In order not to lose the returned value , you need to:
  - assign it to a variable (for further calculations, if needed)OR
  - print it immediately.

# Functions with return

- Example : Write a function that takes the height and width as arguments, calculates the area of a rectangle, and returns it. Then display the area in the output window. Use the function in your program.

```
def recArea(aHeight, aWidth) :  
    area = aHeight * aWidth  
    return area  
  
h = eval(input("Enter the height: " ))  
w = eval(input("Enter the width: "))  
print("Area = ", recArea(h,w)) #Function call inside print()
```

-----output-----

```
Enter the height: 4  
Enter the width: 6  
Area = 24
```

# Functions with return

- Problem: Write a function that accepts two numbers as arguments and returns the sum of these two numbers.

The  
function:

```
def sum_of_numbers(a,  
b):  
    return a + b
```

The complete  
program:

```
def sum_of_numbers(a, b):  
    return a + b  
  
num1 = int(input("Enter first number: "))  
num2 = int(input("Enter second number:  
"))  
  
print("Sum of two numbers: ",  
sum_of_numbers(num1, num2))
```

# Functions with Multiple return values

So, what if you want to return two variables from a function instead of one? There are a couple of approaches which new programmers take.

Let's take a look at a simple example:

```
# example on multiple return values
```

```
def profile():
```

```
    name = input('Enter your name: ')
```

```
    YOB = int(input('Enter your year of  
    birth: '))
```

```
    age= 2022- YOB
```

```
    return [name, age]
```

```
profile_data = profile()
```

```
print('Name: ',profile_data[0])
```

```
print('Age: ',profile_data[1])
```

The returned values  
are now in a list

Capturing the  
returned values in  
the list `profile_data`

Retrieving each  
item by its index

```
Enter your name: Naji  
Enter your year of birth: 1999  
Name: Naji  
Age: 23
```

```
# example on multiple return values
```

```
def profile():
```

```
    name = input('Enter your name: ')
```

```
    YOB = int(input('Enter your year of  
    birth: '))
```

```
    age= 2022- YOB
```

```
    return name, age
```

```
name, age = profile()
```

```
print('Name: ',name)
```

```
print('Age: ',age)
```

The returned  
values are  
assigned to  
variables

```
Enter your name: Naji  
Enter your year of birth:  
1999  
Name: Naji  
Age: 23
```

# Functions in Python

## Local Variables

A local variable is created inside a function and cannot be accessed by statements that are outside the function. Different functions can have local variables with the same names because the functions cannot see each other's local variables.

Anytime you assign a value to a variable inside a function, you create a local variable.

A local variable belongs to the function in which it is created, and only statements inside that function can access the variable.

```
def printHello():  
    aname = input('Enter your  
name: ')  
    print("Hello,", aname)  
printHello()
```

Enter your name:  
Fadi  
Hello, Fadi



# Functions in Python

## Local Variables

An error will occur if a statement in one function tries to access a local variable that belongs to another function or any part other than

```
1 def printHello():
2     aname = input('Enter your name: ')
3     print("Hello,", aname)
4     print(aaname)
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-9-bba0841c5ec6> in <module>()
      2     aname = input('Enter your name: ')
      3     print("Hello,", aname)
----> 4     print(aaname)
```

**NameError: name 'aname' is not defined**

The statement in line 4 tries to access the *aname* variable. This results in an error because the *aname* variable is local to the *printHello()* function, and statement in line 4 cannot access it.

# Functions in Python

## Making Changes to Parameters

When an argument is passed to a function in Python, the function parameter variable will reference the argument's value.

However, any changes that are made to the parameter variable will not affect the argument.

```
1 name="Ahmad"
2 def printHello(name):
3     print("Hello,", name)
4     name= "Fadi"
5     print("The name is now,", name)
6 printHello(name)
7 print("Outside the function, the name is still",name)
```

Hello, Ahmad

The name is now, Fadi

Outside the function, the name is still Ahmad

# Functions in Python

## Global Variables

A **global variable** can be accessed by any statement in the program file, including the statements in any function.

When a variable is created by an assignment statement that is written outside all the functions in a program file, the variable is ***global***.

```
name="Ahmad"
def printHello():
    print("Hello",name)
printHello()
```

Hello Ahmad

```
1 x = "awesome"
2 def myfunc():
3     x = "fantastic"
4     print("Python is " + x)
5
6 myfunc()
7 print("Python is " + x)
```

Python is fantastic  
Python is awesome

# Functions in Python

## Global Variables

An additional step is required if you want a statement in a function to assign a value to a **global** variable. In the function, you must declare the global variable.

```
1 def myfunc():
2     global x
3     x = "fantastic"
4
5 myfunc()
6
7 print("Python is " + x)
```

Python is fantastic

# Functions in Python

## Global Variables

Also, use the **global** keyword if you want to change a global variable inside a function.

To change the value of a global variable inside a function, refer to the variable by using the **global** keyword:

```
1 name="Ahmad"
2 def main():
3     global name
4     name="Nizar"
5     printHello()
6
7 def printHello():
8     print("Hello",name)
9
10 main()
```

Hello Nizar

```
x = "awesome"
def myfunc():
    global x
    x = "fantastic"
    print("Python is " +
x)
```

```
myfunc()
print("Python is " + x)
```

Python is  
fantastic  
Python is  
fantastic

# Hiding Complexity: Interfaces and Implementations

- When a function is called, the resulting value is referred to as the return value of the function.
- To use a function, all you need to know are:
  - the name of the function.
  - what kind of argument(s) it needs.
  - what it gives back as a return value.
  - what other effects calling the function has.
- There is a name for the items you need to know: the interface of the function. To use a function, knowing the interface is enough. There's no need to know what goes on under the bonnet-its implementation.
- When the implementation is separated from the interface, thereby hiding some of the complexity, we also say that an abstraction has been created.

# The Python Interpreter and Functions

- The Python interpreter deals with a program one line after the other, starting with the very first line. A normal program line is executed when the interpreter gets to it. However, functions do receive special treatment, which we examine in this section.
- When the Python interpreter encounters a function definition, it reads this definition into memory for later use.
- The interpreter only executes the lines of a function definition, when the function is called.

# The Python Interpreter and Functions

- On those occasions where a function is called, the parameters are initialized with the actual arguments and then the statements in its body are executed.
- On encountering the keyword **return**, Python does two things:
  1. the function stops executing (control is handed back to the point where the function was called originally)
  2. the value of the expression following return is passed back. In other words, the evaluation of the function expression is complete, and the result is the return value of the function.



# Using Functions- The Benefits

- Replacing duplicate code with a function can make a program shorter and more readable.
- A further advantage is that it makes it much easier to modify the code, if needed.
- **In short, functions can help us write code that:**
  - is shorter and consequently more readable.
  - is easier to change, making it less likely that errors are introduced.
  - explicitly mirrors the decomposition of a problem and consequently is easier to understand.

# Functions in Python

## The **math** Module

The **math** module in the Python standard library contains several functions that are useful for performing mathematical operations.

Some math Module functions:

function	Description
<b>ceil(x)</b>	Returns the smallest integer that is greater than or equal to x.
<b>exp(x)</b>	Returns $e^x$
<b>floor(x)</b>	Returns the largest integer that is less than or equal to x.
<b>hypot(x, y)</b>	Returns the length of a hypotenuse that extends from (0, 0) to (x, y).
<b>log(x)</b>	Returns the natural logarithm of x.
<b>log10(x)</b>	Returns the base-10 logarithm of x.
<b>radians(x)</b>	Assuming x is an angle in degrees, the function returns the angle converted to radians.
<b>degrees(x)</b>	Assuming x is an angle in radians, the function returns the angle converted to degrees.
<b>sqrt(x)</b>	Returns the square root of x.

# Functions in Python

## The **math** Module

Some math Module functions

examples:

```
1 from math import *
2 x=2.6
3 y=10000
4 z=1
5 print (ceil(x))
6 print (floor(x))
7 print(log(y))
8 print(log10(y))
9 print(exp(z))
```

3

2

9.210340371976184

4.0

2.718281828459045

# Exercise 1

1. Write a program that asks the user to enter a distance in kilometers, then using a function **showMiles** that accepts the entered distance in kilometers and displays the converted distance in miles rounded to 2 decimal places. The conversion formula is as follows:

$$\text{Miles} = \text{Kilometers} * 0.6214$$

## Solution

```
def showMiles(kilometers):  
    miles = kilometers * 0.6214  
    print(f"{kilometers} kilometers is equal to {round(miles,2)} miles.")  
  
kilometers = float(input("Enter distance in kilometers: "))  
showMiles(kilometers)
```

# Exercise 2

Python allows you to repeat a string by multiplying it by an integer, e.g. 'Hi' \* 3 will give 'HiHiHi'.

Pretend that this feature does not exist, and instead write a function named `repeat` that accepts a string and an integer as arguments.

The function should return a string of the original string repeated the specified number of times, e.g. `repeat('Hi', 3)` should return 'HiHiHi'.

Use a main function that enters the string to be repeated and the number of repetitions, then calls the function and prints the result.

## Ex. 2 Solution

```
def repeat(string, times):  
    repeated_string = ''  
    for i in range(times):  
        repeated_string += string  
    return repeated_string
```

```
def main():  
    string = input("Enter a string: ")  
    times = int(input("Enter the number of times to repeat: "))  
    result = repeat(string, times)  
    print("Result:", result)
```

```
main()
```

# Exercise 3

Write a program that generates a random number in the range of 1 through 100 and asks the user to guess what the number is.

If the user's guess is higher than the random number, the program should display "Too high, try again."

If the user's guess is lower than the random number, the program should display "Too low, try again."

If the user guesses the number, the application should congratulate the user and generate a new random number so the game can start over.

# Exercise 3- Solution

```
import random

# main function
def main():

    # Initializing Local variables
    number = 0
    play = 1

    while(play > 0):
        number = random.randint(1, 100)
        play = playGuessingGame(number)

    print('Thanks for playing!')

def playGuessingGame(number):
    # Get the user's guess.
    userGuess = int(input('Enter a number between 1 and 100, ' \
                          'or 0 to quit: '))

    # As long as user doesn't want to quit
    while userGuess > 0:
        if userGuess > number:
            print('Too high, try again')
            userGuess = int(input('Enter a number between 1 ' \
                                  'and 100, or 0 to quit: '))

        elif userGuess < number:
            print('Too low, try again')
            userGuess = int(input('Enter a number between 1 ' \
                                  'and 100, or 0 to quit: '))

        else:
            print('Congratulations! You guessed the right number!')
            return userGuess # Start the game again

    return userGuess # UserGuess is 0 and user wants to quit.

# Call the main function.
main()
```

```
Enter a number between 1 and 100, or 0 to quit: 20
Too low, try again
Enter a number between 1 and 100, or 0 to quit: 50
Too low, try again
Enter a number between 1 and 100, or 0 to quit: 90
Too high, try again
Enter a number between 1 and 100, or 0 to quit: 70
Too high, try again
Enter a number between 1 and 100, or 0 to quit: 60
Too high, try again
Enter a number between 1 and 100, or 0 to quit: 55
Too high, try again
Enter a number between 1 and 100, or 0 to quit: 52
Too low, try again
Enter a number between 1 and 100, or 0 to quit: 53
Too low, try again
Enter a number between 1 and 100, or 0 to quit: 54
Congratulations! You guessed the right number!
Enter a number between 1 and 100, or 0 to quit: 0
Thanks for playing!
```

*Try to modify the program so that the user can have only 5 guesses.  
If the user fails to guess after the allowed trials, a proper message should be displayed, and the program terminates.*