Imam Abdulrahman Bin Faisal University
Deanship of Preparatory Year and Supporting Studies

Basics of Programming through Python

User-Defined Functions

Introduction to Programming COMP102

Term 3 2022-2023

User-Defined Functions

Learning Outcomes

- Create your own function.
- Distinguish the difference between Fruitful Functions & Void Functions.
- Distinguish between arguments and parameters.

Functions in Python

- A **function** is a block of organized, **reusable** code that is used to perform a single, related action.
- Functions **reduce** code duplication, **increase** program modularity and provides a high degree of code **reusing**.
- As you already know, Python gives you many built-in functions like print() etc. but you can also create your own functions. These

Functions in Python

- A function is like a ***subprogram***, a small program inside of a program.
- The basic idea – we write a sequence of statements and then give that sequence a **name**. We can then execute this sequence at any time by **referring** to the name

Defining a function

Here are simple rules to define a function in Python:

- Function blocks begin with the **keyword def** followed by the function name and parentheses (( )).
- Any input **parameters or arguments** should be placed within these parentheses. You can also define parameters inside these parentheses.
- The code block within every function starts with a colon (:) and is indented.
- **Syntax:**

```
def function_name( parameters ):
        Body
```

Function Definition

def marks the start of the function
To define a and declare a function

Function name to uniquely identify a function

```
def my_function(parameter):
```

Argument to pass a value in function

Colon(:) to mark end of function header

Example 1: print your name and age

```
1
2  def print_info( name, age ):
3      print ("Name: ", name)
4      print ("Age ", age)
5
6  #call of the function
7  print_info( age=50, name="Ahmad" );
```

OR

```
age=50
name="Ahmad"
print_info( age, name);
```

- **Output:**

Name: Ahmad

Age 50

Example 2

```
def my_name():
    print("I am Nora.")
```

- "**def**" is a keyword.
- Indicates that this is a function definition.

Example 2

```
def my_name():
    print("I am Nora.")
```

- “my_name” is the function name.
- Indicates that this is a function definition.

Example 2

```
def my_name():
    print("I am Nora.")
```

- “()” empty parentheses.
- Indicate that this function doesn’t take any parameters.

Example 2

```
def my_name():
    print("I am Nora.")
```

- The first line of the function definition is called the header.
- Header end with Colon :

Example 2

```
def my_name():
    print("I am Nora.")
```

- The rest called the Body.
- Body has to be intended (4 spaces).
- Body can contain any number of statements.
- End the function with an empty line.

Function Call

```
def my_name():
    print("I am Nora.")


my_name()
```

We **call**/invoke the function by using the function name, parentheses, and arguments in an expression .

Naming rules

The rules for function names are the same as for variable names:

- Letters & numbers & some punctuation marks are legal.
- The first character can’t be a number.
- You can’t use a keyword as a function name.
- Avoid having a variable and a function with the same name.
- **Two different functions** can’t have the same name, even if they have different arguments.

Example 3

```
def repeat_name():
    my_name()
    my_name()
```

```
I am Nora.
I am Nora.
```

- Once you have defined a function, you can ***use it inside another function.***
- Define a new function “repeat_name”.
- Call the old function inside it “my_name() ”.
- And then call the new function

Example 3

```python
def my_name():
    print("I am Nora.")


def repeat_name():
    my_name()
    my_name()


repeat_name()
```

Definitions & Uses

- The statements inside the function do not get executed **until the function is called**.
- you have to create a function before you can execute it.
- Define the Function then Call the Function.

Try (1)

- Move the last line of this program (Example 3) to the top, so the function call appears before the definitions.
- Run the program and see what error message you get.

Try (2)

- Move the function call back to the bottom and move the definition of my_name() after the definition of repeat_name(). What happens when you run this program?

Flow of Execution

- The order in which statements are executed is called *flow of execution.*
- Execution always begins at the first statement of the program.
- Statements are executed one at a time, in order from top to bottom.

Flow of Execution

- *Function Definitions* do not alter the flow of execution of the program.
- *Function Call* is like a detour in the flow of execution. Instead of going to the next statement, the flow jumps to the body of the function, executes all the statements there, and then comes back to pick up where it left off.

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 parentheses after the name of the function

Example: Total=sum(x,y)

Arguments

Example 4

This function works with any value that can be printed.

Import math

```
print_twice('spam ')
print_twice(17)
print_twice(math.pi)
```

```
spam
spam
17
17
3.141592653589793
3.141592653589793
```

Call with an argument

Example 5

Import math

```
print_twice('spam ' *4)
print_twice(math.cos(math.pi))
```

```
spam spam spam spam
spam spam spam spam
-1.0
-1.0
```

Variable as an argument

You can use variable as an argument:

```
myName = "Nora "
print_twice(myName)
```

```
Nora
Nora
```

Parameters

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

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

“

The name of the variables we pass as arguments (x,y) can be different from the name of the parameters (a,b).

```
def sum(a=3,b=6):
        return a+b
x=7
y=4
print(sum(x,y))
```

Mathematical Built-in Function

- **you should add import math before using math functions or values like :**
- **sin()**
- **cos()**
- **pow(,)**
- **sqrt()**
- **pi** **value = 3.14**

Constant variable

Import math
math. --------

Parameters & Arguments

- Some *built-in function* requires an *arguments*.
- Some takes one argument such as *math.sin* and some takes two arguments such as *math.pow* (base & exponents)

Import math

```
math.sin(2)
math.pow(3, 2)
```

number power

0.9092

9 like: print(3**2)

Fruitful Functions & Void Functions

- Fruitful function:
  - Such as *math functions.*
  - Yield results.
  - If you call *math.sqrt(5)* in the **interactive mode**, Python will display the results.
  - In the script mode, it will compute the square root of 5, but since it **doesn't store** the result in a variable or display the result, it is not very useful.

Example 6

You need to assign it to a variable or use it as part of an expression:

radians=float(input(‘enter radians’))

```
x = math.cos(radians)
golden = (math.sqrt(5) + 1) / 2
```

print(x)
print(golden)

Fruitful Functions & Void Functions

- Void Functions
  - Such as *print_twice.*
  - Perform an action but don’t return a value.
  - Void functions might display something on the screen or have some other effect, but they don’t have a return value. If you try to assign the result to a variable, you get a special value called None.

Example 7

```
result = print_twice('Bing')
print(result)
```

```
Bing
Bing
None
```

Return Statement

- To return a result from a function, we use the return statement in our function.
  - For example, we could make a very simple function called *add_two* that adds two numbers together and **returns** a result.

Example

```
def add_two(a, b):
    added = a + b
    return added


x = add_two(3, 5)
print(x)
```

```
8
```

Note: The return statement within a function does not print the value being returned to the caller

Return Statement

- When this script executes, the print statement will print out “8” because the *addtwo* function was called with 3 and 5 as arguments. Within the function, the **parameters** a and b were 3 and 5 respectively.

Return Statement

- The function computed the sum of the two numbers and placed it in the local function variable named added. Then it used the return statement to send the computed value back to the calling code as the function result, which was assigned to the **variable x** and printed out.

Functions That Return Values

- Sometimes a function needs to return more than one value.
- To do this, simply list more than one expression in the `return` statement separated by comma such as: **return a,b,c**
- When calling this function, use simultaneous assignment**(i.e.**

Functions That Return Values

```
1  def sumDiff(x, y):
2      sum = x+y
3      diff = x-y
4      return sum, diff
5
6  num1, num2 = eval(input("Please enter two numbers (num1, num2) "))
7  s,d = sumDiff(num1, num2)
8  print(s)
9  print(d)
10 print ("The sum is", s, "and the difference is", d)
```

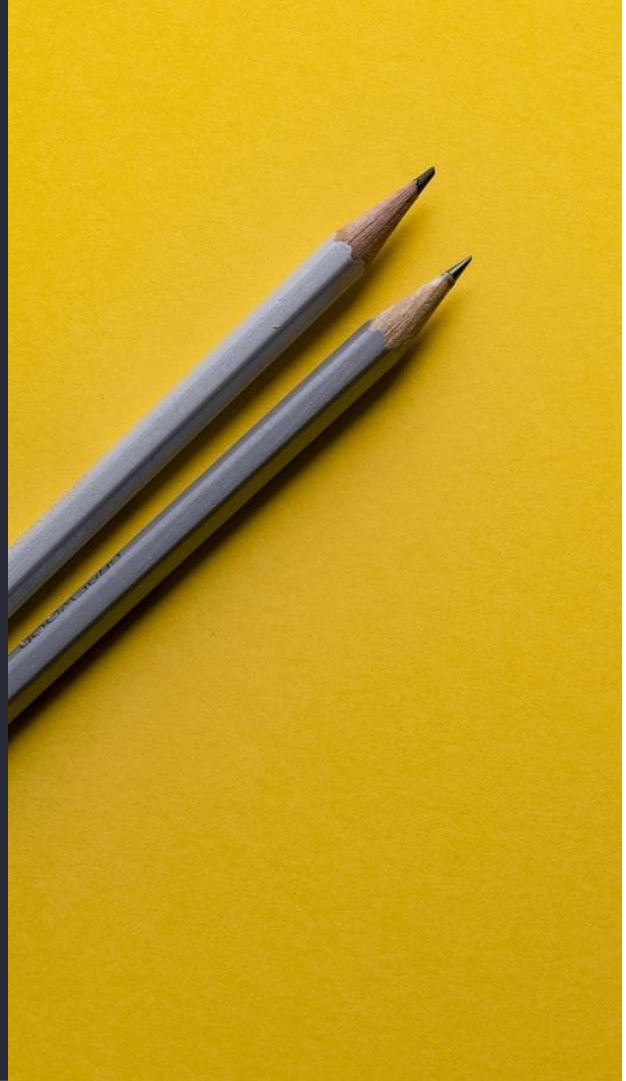
- the values are assigned based on position, so s gets the first value returned (the sum), and d gets the second (the difference)

Why Functions?

- Creating a new function gives you an opportunity to name a group of statements, which makes your program easier to read, understand, and debug.
- Functions can make a program smaller by eliminating repetitive code. Later, if you make a change, you only have to make it in one place.

Why Functions?

- Dividing a long program into functions allows you to debug the parts one at a time and then assemble them into a working whole.
- Well-designed functions are often useful for many programs. Once you write and debug one, you can reuse it.



Thanks!