



The Islamic University of Gaza
Faculty of Engineering
Computer Engineering Department



Introduction to Computers

LAB #6

Functions

Eng. Hashem M. Hejazy

Eng. Mohammed O. AlHabbash

Fall 2026

Introduction

Suppose that you need to find the sum of integers from 1 to 10, 20 to 37, and 35 to 49. If you create a program to add these three sets of numbers, your code might look like this:

```
sum = 0
for i in range(1, 11):
    sum += i
print("Sum from 1 to 10 is", sum)
sum = 0
for i in range(20, 38):
    sum += i
print("Sum from 20 to 37 is", sum)
sum = 0
for i in range(35, 50):
    sum += i
print("Sum from 35 to 49 is", sum)
```

You may have observed that the code for computing these sums is very similar, except that the starting and ending integers are different. Wouldn't it be nice to be able to write commonly used code once and then reuse it? You can do this by defining a function, which enables you to create reusable code.

Defining a Function

A function definition consists of the **function's name**, **parameters**, **body**, and **return expression (Optional)**.

The syntax for defining a function is as follows:

```
def functionName(list of parameters):

    # Function body

    return expression
```

For example, the preceding code can be simplified by using functions, as follows:

```
def sum(i1, i2):  
    result = 0  
    for i in range(i1, i2 + 1):  
        result += i  
    return result
```

Calling a Python Function

In a function's definition, you define what it is to do. To use a function, you have to call or invoke it. The program that calls the function is called a caller. There are two ways to call a function, depending on whether or not it returns a value.

If the function returns a value, a call to that function is usually treated as a value. For example:

```
sumResult = sum(1, 10)
```

calls `sum(1, 10)` and assigns the result of the function to the variable `sumResult`.

Another example of a call that is treated as a value is

```
print(f'Sum of Two number using Function is:  
{sum(1, 10)}')
```

If a function does not return a value, you can call the function just by its name and parameter as follow:

```
sum(1, 10)
```

Example #1

Write a Python program that asks the user to enter two numbers and then define a function to find the max number.

```
1 def max(num1,num2):
2     if num1>num2:
3         print('The first number the user entered is the largest number ')
4     elif num1==num2:
5         print('Both number are equal')
6     else:
7         print(f'The second number the user entered is the largest number ')
8
9 firstNumber=float(input('Enter First Value '))
10 secondNumber=float(input('Enter Second Value '))
11 max(firstNumber,secondNumber)
12
```

Run: main ×

C:\Users\HP\PycharmProjects\Lab\venv\Scripts\python.exe C:\Users\HP\PycharmProjects\Lab\main.py

Enter First Value 20

Enter Second Value 5

The first number the user entered is the largest number

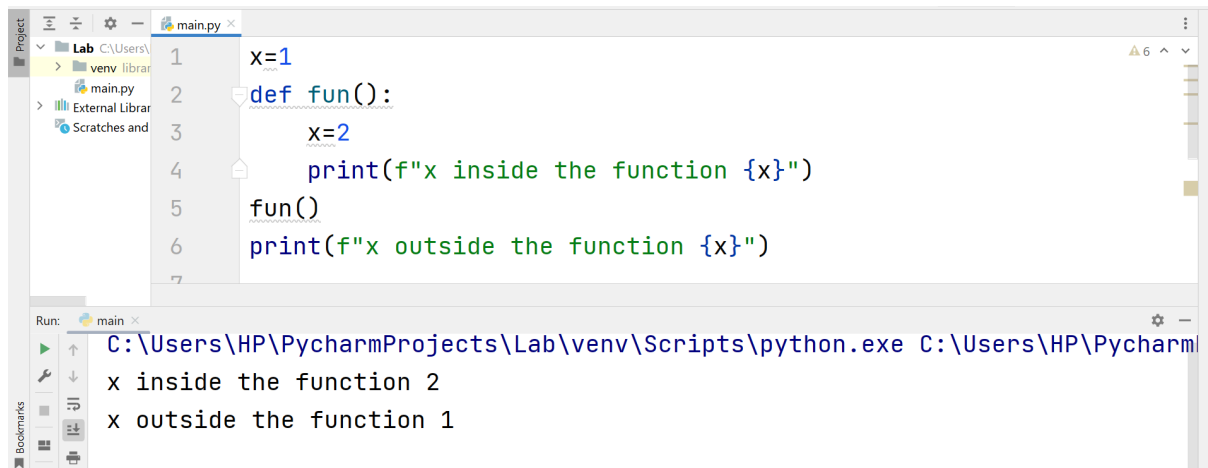
The Scope of Variables

The scope of a variable is the part of the program where the variable can be referenced.

A variable created inside a function is referred to as a local variable. Local variables can only be accessed within a function. The scope of a local variable starts from its creation and continues to the end of the function that contains the variable. In Python, you can also use global variables. They are created outside all functions and are accessible to all functions in their scope.

Example #2

Here there are two variables with the same name (x). Inside the function, it will take a different value from the original value.



The screenshot shows the PyCharm IDE with a file named `main.py`. The code in the editor is as follows:

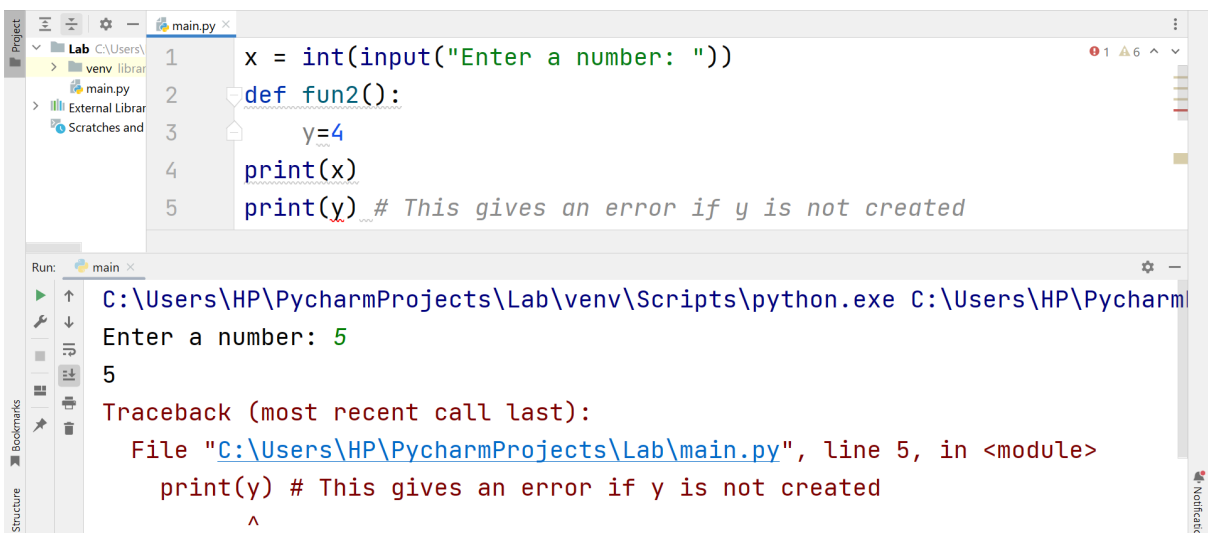
```
1 x=1
2 def fun():
3     x=2
4     print(f"x inside the function {x}")
5     fun()
6     print(f"x outside the function {x}")
7
```

The Run window at the bottom shows the execution output:

```
Run: main
C:\Users\HP\PycharmProjects\Lab\venv\Scripts\python.exe C:\Users\HP\Pycharm
x inside the function 2
x outside the function 1
```

Example #3

Here there are two variables with different names (x and y). x will print fine because it is a global variable. but y is a local variable and prints outside the function, so it's out of scope and this error will occur.



The screenshot shows the PyCharm IDE with a file named `main.py`. The code in the editor is as follows:

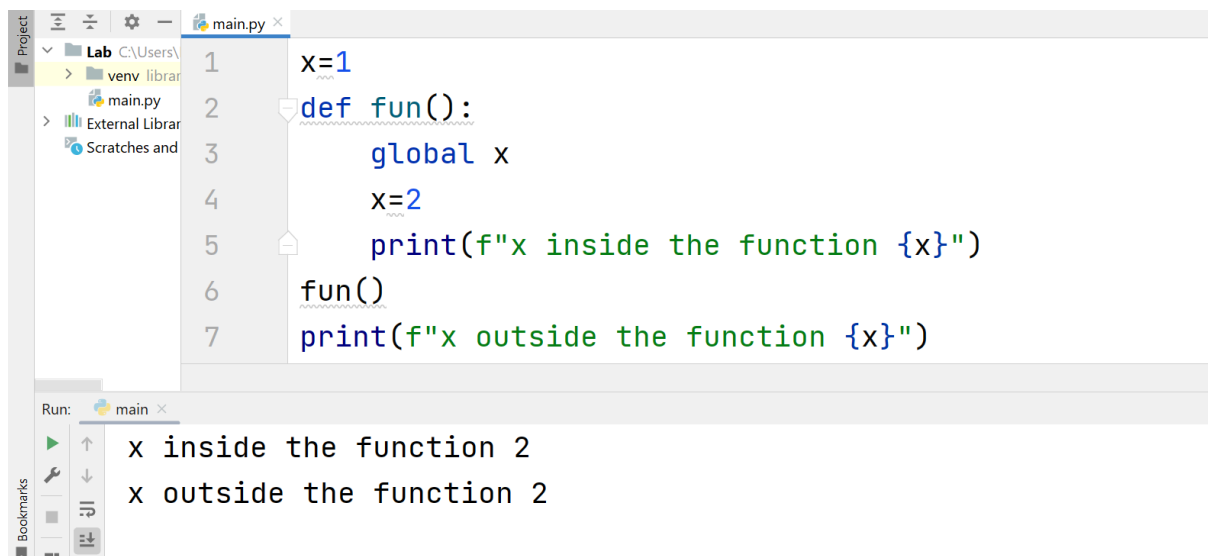
```
1 x = int(input("Enter a number: "))
2 def fun2():
3     y=4
4     print(x)
5     print(y) # This gives an error if y is not created
```

The Run window at the bottom shows the execution output:

```
Run: main
C:\Users\HP\PycharmProjects\Lab\venv\Scripts\python.exe C:\Users\HP\Pycharm
Enter a number: 5
5
Traceback (most recent call last):
  File "C:\Users\HP\PycharmProjects\Lab\main.py", line 5, in <module>
    print(y) # This gives an error if y is not created
    ^
```

Example #4

Here a global variable `x` is created in line 1 and `x` is bound in the function in line 3, **which means that `x` in the function is the same as `x` outside of the function**, so the program prints 2 in line 5 and line 8.



```
1 x=1
2 def fun():
3     global x
4     x=2
5     print(f"x inside the function {x}")
6 fun()
7 print(f"x outside the function {x}")
```

Run: main

```
x inside the function 2
x outside the function 2
```

Types of Python Function Arguments

Python supports various types of arguments that can be passed at the time of the function call. In Python, we have the following 4 types of function arguments.

- ❖ Default argument
- ❖ Keyword arguments (named arguments)
- ❖ Positional arguments
- ❖ Arbitrary arguments (variable-length arguments `*args` and `**kwargs`)

Default Arguments

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument.

```
def myFun(y=50):
    print("y: ", y)
```

Keyword Arguments

The idea is to allow the caller to specify the argument name with values so that the caller does not need to remember the order of parameters.

```
def student(x, y):  
    print(f'The Values are: {x, y}')
```



```
student(y=5, x=10)
```

Output:

```
C:\Users\HP\PycharmProjects\Lab\venv\Scripts\python.exe C:\Users\HP\PycharmProjects\Lab\main.py  
The Values are: (10, 5)
```

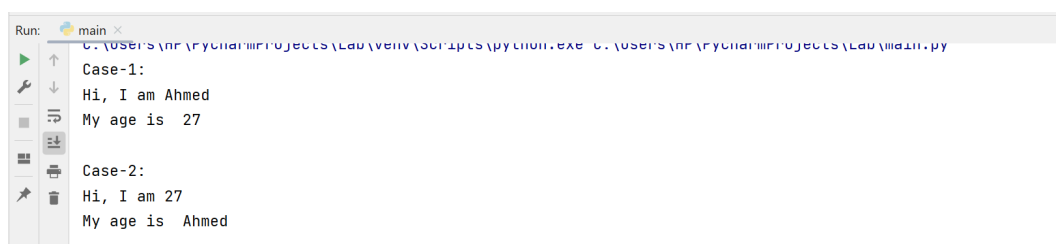
```
Process finished with exit code 0
```

Positional Arguments

We used the Position argument during the function call so that the first argument (or value) is assigned to name and the second argument (or value) is assigned to age. By changing the position, or if you forget the order of the positions, the values can be used in the wrong places.

```
def nameAge(name, age):  
    print("Hi, I am", name)  
    print("My age is ", age)  
# You will get the correct output  
print("Case-1:")  
nameAge("Ahmed", 27)  
# You will get incorrect output  
print("\nCase-2:")  
nameAge(27, "Ahmed")
```

Output:



More Function

The random() Function

In Python, the `random()` function is used to generate a random floating-point number between 0.0 (inclusive) and 1.0 (exclusive). This function is commonly used in simulations, games, or any application where randomization is needed.

```
from random import random

print(f"This is a random number between 0.0 and 0.9  
( {random()} ) ")
```

Output:

```
"C:\Users\HP\PycharmProjects\First project\.venv\Scripts\python.exe"  
This is a random number between 0.0 and 0.9 (0.733255649017607)
```

```
Process finished with exit code 0
```

The randrange() Function

The `randrange()` function is used to generate a random integer within a specific range. Unlike `random()`, it can generate whole numbers and allows for custom step sizes. **the syntax is:**

`randrange(start, stop, step)`

```
from random import randrange

print(f"This is a random number ( {randrange(-100, 100, 5)} ) ")
```

Output:

```
"C:\Users\HP\PycharmProjects\First project\.venv\Scripts\python.exe"  
This is a random number between 0.0 and 0.9 (0.733255649017607)
```

```
Process finished with exit code 0
```

round() Function

In Python, we can control the number of decimal places when displaying numbers using the `round()`

```
num = 3.14159265
rounded_num = round(num, 2)
print("Rounded to 2 decimal places:", rounded_num)
```

Output:

```
"C:\Users\HP\PycharmProjects\First project\.venv\Scripts\python.exe"
This is a random number (-65)
```

```
Process finished with exit code 0
```

Also, can use formatted string literals (`f-strings`).

```
num = 3.14159265
print(f"Rounded to 3 decimal places: {num:.3f}")
```

Output:

```
"C:\Users\HP\PycharmProjects\First project\.venv\Scripts\python.exe"
This is a random number between 0.0 and 0.9 (0.733255649017607)
```

```
Process finished with exit code 0
```

