Department of Computer Science and Engineering
PES University, Bangalore, India

Lecture Notes
Python for Computational Problem Solving
UE23CS151A

Lecture #51

Functions: Definition and Call

By,
Prof. Sindhu R Pai,
Anchor, PCPS - 2023
Assistant Professor
Dept. of CSE, PESU

Verified by,
PCPS Team - 2023

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Centers, Former Chairperson, CSE, PES University)
Prof. Chitra G M(Asst. Prof, Dept. of CSE, PCPS Anchor – 2022)

Introduction to Functions

Function is a set of statements written inside a block and given a name. These statements will be executed when the block is called with its name. It is a subprogram which helps in accomplishing a given task. All functions constitute into a big enterprise and results in million lines of code.
Example: YouTube, Facebook, Whatsapp, etc.

If all the code is in a single sequence, we will not be able to manage it. So parts of code are required and thus **modularity** was brought into picture. It is good and easier for people to develop the modules and finally integrate those to meet the requirements.

Advantages of using functions in python:

- Brings modularity
- Increases readability
- Memory is conserved i.e., repeated invocations of the same function
- Promotes reuse of code
- Debugging and Maintenance becomes easier

Python supports context switching method to support functions rather than monolithic way of handling the functions. This avoids repetition and hence the readability is increased.

NOTE: Every function is associated with a function definition, a function call(invocation) and the return statement

Categories of functions:

Based on who built it:

System defined functions and **User defined functions**

Based on the return value:

Value returning functions and **Non-Value returning functions**

System defined functions: These are the built-in functions which we have just **called/used without writing any definitions.** Example: input(), print(), list.append(), list.extend(), sorted(), set.add(), dict.update() etc.

User defined functions: These are the functions for which **we need to write the definitions and use/call it as per the requirement.**

Function Definition

Has two parts – **leader and suite**

The leader starts with the keyword **def**, then the **function name** and then **a pair of parentheses followed by a colon(:).** Function name must follow the rules of identifiers. The suite can have any valid statement/(s) including another function definition, with an indentation for each statement in a suite.

When the function is defined, the function name becomes an interface to refer to the function. The function entity is stored with the name used in the definition along with the suite. Each entity in Python has a reference count. When the interpreter encounters the leader of the function definition, **leader is processed.** It means, **entry is made in symbol table** with the name of the function/address and the parameters (Parameters in detail is discussed in Lecture #52). At this point of translation, the suite or body of the function is not processed.

Function Call

Name of the function followed by round parentheses causes a function call. No : to be used in function call. A pair of round parenthesis () is called a **function call operator.**

When the interpreter encounters the function call statement, it checks whether the entry is available in the symbol table. **If entry available**, it results in transfer of control to the leader of the function and then **the suite is executed**. After this, control comes back to the point after the function call in the user's code. **If entry is not available** in the symbol table, it results in an Exception - **Name Error.** A called function returns the control to the caller when one of the following occurs:

> the end of the function body is reached

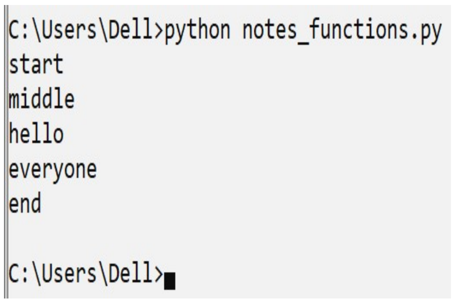
> the return statement is executed

Note: Above discussion on functions prove that there is **no Function Hoisting supported by Python**

Consider the below examples in python tutor and observe the control flow.

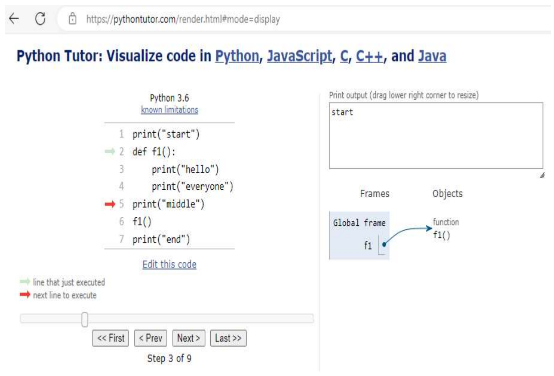
Example code 1:

```
print("start")
def f1():
   print("hello")
   print("everyone")
print("middle")
f1()
print("end")
```



```
C:\Users\Dell>python notes_functions.py
start
middle
hello
everyone
end

C:\Users\Dell>
```



https://pythontutor.com/render.html#mode=display

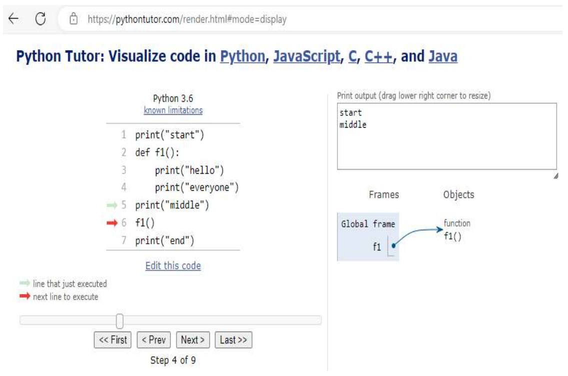
Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6

```
print("start")
def f1():
    print("hello")
    print("everyone")
print("middle")
f1()
print("end")
```

Print output (drag lower right corner to resize)

```
start
```



https://pythontutor.com/render.html#mode=display

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6

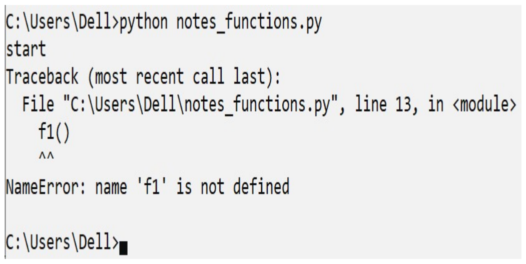
```
print("start")
def f1():
    print("hello")
    print("everyone")
print("middle")
f1()
print("end")
```

Print output (drag lower right corner to resize)

```
start
middle
```

Example code 2: Function call is before the function definition. Results in NameError

```
print("start")
f1()
def f1():
   print("hello")
   print("everyone")
print("middle")
f1()
print("end")
```

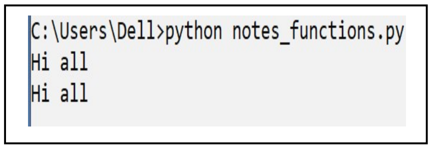


```
C:\Users\Dell>python notes_functions.py
start
Traceback (most recent call last):
  File "C:\Users\Dell\notes_functions.py", line 13, in <module>
    f1()
    ^^
NameError: name 'f1' is not defined

C:\Users\Dell>
```

Example code 3: Function name is same. Only the leader of the latest function is available in the symbol table during both the function calls

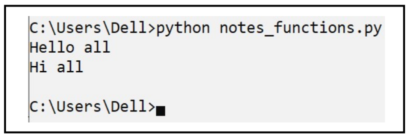
```
def f1():
   print("Hello all")
def f1():
   print("Hi all")
f1()
f1()
```



```
C:\Users\Dell>python notes_functions.py
Hi all
Hi all
```

Example code 4:

```
def f1():
    print("Hello all")
f1()  #during this call, first f1() in symbol table
def f1():
    print("Hi all")
f1() # during this call, second f1() has replaced first one
```



```
C:\Users\Dell>python notes_functions.py
Hello all
Hi all

C:\Users\Dell>
```

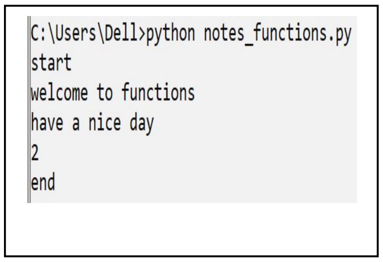
Value returning functions: Function which returns the value explicitly from the function to the calling program using a return statement at the end of the function definition. Example: list.pop(), dict.popitem(),input(), etc.

Non-Value returning functions: Function which do not return the value explicitly from the function by default **returns the None value implicitly** to the calling program at the end of the function definition. Example: list.append(), list.extend(), sorted(), set.add(), dict.update(), print(), etc.

Let us write few User defined Value returning and Non- value returning functions

Example code 5: Value returning function f1

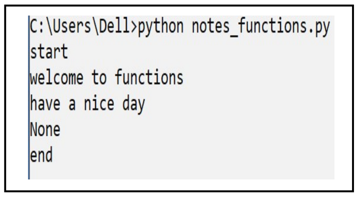
```
print("start")
def f1():
    print("welcome to functions")
    print("have a nice day")
    return 2
print(f1())
print("end")
```



```
C:\Users\Dell>python notes_functions.py
start
welcome to functions
have a nice day
2
end
```

Example code 6: Non-value returning function by default returns None. F2() returns None. So print prints None

```
print("start")
def f2():
    print("welcome to functions")
    print("have a nice day")
print(f2())
print("end")
```



```
C:\Users\Dell>python notes_functions.py
start
welcome to functions
have a nice day
None
end
```

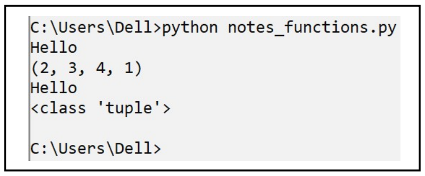
Note: None is a value in python whose type is NoneType.

```
>>> a = None
>>> type(a)
<class 'NoneType'>
>>>
```

Returning multiple values: When multiple values are returned in a function using the same return statement, the interpreter puts it together into a tuple and returns it to the calling program.

Example code 7: Returning multiple elements in the same return statement

```
def f3():
    print("Hello")
    return 2,3,4,1
print(f3())
print(type(f3()))
```



```
C:\Users\Dell>python notes_functions.py
Hello
(2, 3, 4, 1)
Hello
<class 'tuple'>

C:\Users\Dell>
```

Try it!!

- If there are multiple return statements inside the function, what is the result of function execution?
- Can you define a function inside another function?
- If yes, how do you call it?
- Li = [2,4,6,8]; Li = Li.append(0); print(Li) – What this code prints?
- What is the output of below code?

  ```
  def f1():
          print("in f1")
          return f2()
  def f2():
          print("in f2")
  print(f1())
  ```

- If you interchange the f1() call and f2() definitions, what is the output?

-END-