5.1 - Functions

Data Science Practicum 2021/22, Lesson 5.1

Marko Tkalčič

Univerza na Primorskem

Table of Contents

Functions

Function Exercises

Declaration

- 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 first statement of a function can be an optional statement - the documentation string of the function or *docstring*
- The code block within every function starts with a colon (:) and is indented
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None

Declaration

- 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 first statement of a function can be an optional statement - the documentation string of the function or *docstring*
- The code block within every function starts with a colon (:) and is indented
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

Function

```
def fun1(a):
    """"multiline comment
    second line"""
    print(a)
    return


help(fun1)

fun1(1)
```

```
Help on function fun1 in module __main__:

fun1(a)
    "multiline comment
    second line

1
```

Calling functions

PASS BY REFERENCE

Calling functions

PASS BY REFERENCE

```
def changeme( mylist ):
   mylist.append([1,2,3,4]);
   print( "Values inside the function: ", mylist)
   return

mylist = [10,20,30];
print("Values outside the function: ", mylist)
changeme( mylist );
print("Values outside the function: ", mylist)
```

Calling functions

PASS BY REFERENCE

```
def changeme( mylist ):
   mylist.append([1,2,3,4]);
   print( "Values inside the function: ", mylist)
   return

mylist = [10,20,30];
print("Values outside the function: ", mylist)
changeme( mylist );
print("Values outside the function: ", mylist)
```

```
Values outside the function:  [10, 20, 30]
Values inside the function:  [10, 20, 30, [1, 2, 3, 4]]
Values outside the function:  [10, 20, 30, [1, 2, 3, 4]]
```

PAssing variables

- If the value passed in a function is immutable, the function does not modify the caller's variable
- If the value is mutable, the function may modify the caller's variable in-place

```python
def try_to_modify(x, y, z):
    x = 23
    y.append(42)
    z = [99] # new reference
    print(x)
    print(y)
    print(z)

a = 77    # immutable variable
b = [99]  # mutable variable
c = [28]

try_to_modify(a, b, c)
print("---")
print(a)
print(b)
print(c)
```

```
23
[99, 42]
[99]
---
77
[99, 42]
[28]
```

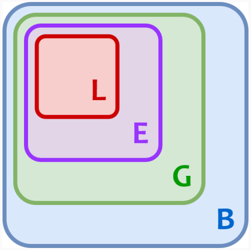
Namespaces - Variable scope

- Suppose the same name is given to variables in various parts of your code
- How does the interpreter know which value to use?

Namespaces - Variable scope

- Suppose the same name is given to variables in various parts of your code
- How does the interpreter know which value to use?

- Order:
  1. Local: If you refer to x inside a function, then the interpreter first searches for it in the innermost scope that's local to that function.
  2. Enclosing: If x isn't in the local scope but appears in a function that resides inside another function, then the interpreter searches in the enclosing function's scope.
  3. Global: If neither of the above searches is fruitful, then the interpreter looks in the global scope next.
  4. Built-in: If it can't find x anywhere else, then the interpreter tries the built-in scope.



Namespaces - Variable scope

```
x = 5
def printx():
    x = 10
    print(x)
printx()
```

```
10
```

```
x = 5
def printx():
    print(x)
printx()
```

```
5
```

Namespaces - Variable scope

```
x = 5
def printx():
    x = 10
    print(x)
printx()
```

```
10
```

```
x = 5
def printx():
    print(x)
printx()
```

```
5
```

```
x = 'global'

def f():
    x = 'enclosing'
    def g():
        x = 'local'
        print(x)
    g()

f()
```

Arguments

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Required arguments

```python
def printme(a):
    print(a)

printme()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-633999144120> in <module>
      2     print(a)
      3 
----> 4 printme()

TypeError: printme() missing 1 required positional argument: 'a'
```

Keyword arguments

```
def printinfo( name, age ):
   print("Name: ", name)
   print( "Age ", age)
   return;

printinfo( age=50, name="miki" )
```

```
Name:  miki
Age  50
```

Default arguments

```
def printinfo( name, age=35 ):
   print("Name: ", name)
   print( "Age ", age)
   return;

printinfo( age=50, name="miki" )
printinfo( name="miki" )
```

```
Name:  miki
Age  50
Name:  miki
Age  35
```

Variable-length arguments

- Number of parameters could be unknown at write-time
- An asterisk (*) is placed before the variable name that holds the values of all non-keyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call.

```
def printinfo( arg1, *vartuple ):
   print(arg1)
   for var in vartuple:
      print(var)
   return;

# Now you can call printinfo function
printinfo( 10 )
print("---")
printinfo( 70, 60, 50 )
```

```
10
---
70
60
50
```

Pass

- Function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

```
def myfunction():
  pass
```

Table of Contents

Functions

Function Exercises

Exercise

- Create a function showEmployee() in such a way that it should accept employee name, and it's salary and display both, and if the salary is missing in function call it should show it as 9000

Exercise

- Create a function showEmployee() in such a way that it should accept employee name, and it's salary and display both, and if the salary is missing in function call it should show it as 9000

```python
def showEmployee(name, salary=9000):
    print("Employee", name, "salary is:", salary)

showEmployee("Ben", 9000)
showEmployee("Ben")
```

Exercise

- Create an inner function to calculate the addition in the following way
  - Create an outer function that will accept two parameters a and b
  - Create an inner function inside an outer function that will calculate the addition of a and b
  - At last, an outer function will add 5 into addition and return it

Exercise

- Create an inner function to calculate the addition in the following way
  - Create an outer function that will accept two parameters a and b
  - Create an inner function inside an outer function that will calculate the addition of a and b
  - At last, an outer function will add 5 into addition and return it

```python
def outerFun(a, b):
    def innerFun(a,b):
        return a+b
    add = innerFun(a, b)
    return add+5

result = outerFun(5, 10)
print(result)
```

Exercise

- write a function that takes as input up to five integers and returns the sum

Exercise

- write a function that takes as input up to five integers and returns the sum

```python
def mysum(*ints):
    s = 0
    for i in ints:
        s +=i
    return s

print(mysum())
```

Exercise

- write a function that takes as input up to five variables:
    - two ints
    - two strings
    - one list
- and prints out all the values of these variables

Exercise

- write a function that takes as input up to five variables:
  - two ints
  - two strings
  - one list
- and prints out all the values of these variables

```
i1 = 5
i2 = 10
s1 = "abc"
s2 = "def"
l1 = [1,2,3]

def myprint(i1 = 0, i2 = 0, s1 = "", s2 = "", l1 = []):
    print(i1,i2,s1,s2,l1)

myprint()
myprint(l1=l1)
myprint(l1=l1, s2=s2)
```

```
0 0   []
0 0   [1, 2, 3]
0 0  def [1, 2, 3]
```

References

Part of the material has been taken from the following sources. The usage of the referenced copyrighted work is in line with fair use since it is for nonprofit educational purposes.

- https://realpython.com/python-namespaces-scope/
- https://www.tutorialspoint.com/python/python_functions.htm
- https://www.w3schools.com/python/python_functions.asp
- https://pynative.com/python-functions-exercise-with-solutions/
- https://www.w3schools.com/python/python_classes.asp
- https://stackoverflow.com/questions/625083/what-init-and-self-do-on-python