Arguments & Functions

Olarik Surinta, PhD.

Outline

- Arguments - argparse
- Functions

Argparse

```
import argparse

parser = argparse.ArgumentParser(description='Arguments')
parser.add_argument('-f', '--first', dest='int1', required=True, help='first value')
parser.add_argument('-s', '--second', dest='int2', required=True, help='second value')
parser.add_argument('--version', action='version', version='%(prog)s version 0.1')
args = vars(parser.parse_args())

print(args['int1'])
print(args['int2'])
```

Argparse – cont.

```
val1 = args['int1']
val2 = args['int2']


print ''val1 + val2 = '', val1+val2
```

Argparse – cont.

$ python pass-argu.py -h

```
usage: pass-argu.py [-h] -f INT1 -s INT2 [--version]

Arguments

optional arguments:
  -h, --help            show this help message and exit
  -f INT1, --first INT1
                        first value
  -s INT2, --second INT2
                        second value
  --version             show program's version number and exit
```

Functions - Syntax

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

docstring

- The first string after the function header is called the docstring and is short for documentation string.
- It is used to explain in brief, what a function does.

docstring

```
def greet(name):
    """This function greets to
    the person passed in as
    parameter"""
    print("Hello, " + name + ". Good morning!")
```

docstring

```
>>> print(greet.__doc__)
This function greets to
        the person passed into the
         name parameter
>>> greet("Paul")
Hello, Paul. Good morning!
```

Example – helloWorld.py

```
#!/usr/bin/python
def printHelloWorld():
    "This function prints --> Hello World!!!"
    print "Hello World!!!"
    return
```

Calling a Function

- You can execute a function by calling it from another function or directly from the Python prompt.

```
#!/usr/bin/python
def printHelloWorld():
    print "Hello World!!!"
    return


printHelloWorld()
```

$ python helloWorld.py
Hello World!!!

Pass by reference

- All parameters (arguments) in the Python language are passed by reference.
- It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

Pass by reference

```
#!/usr/bin/python

# Function definition is here
def changeme(mylist):
   "This changes a passed list into this function"
   mylist.append([1,2,3,4])
   print "Values inside the function: ", mylist
   return

# Now you can call changeme function
mylist = [10,20,30]
changeme(mylist)
print "Values outside the function: ", mylist
```

Pass by reference

- Here, we are maintaining reference of the passed object and appending values in the same object. So, this would produce the following result −

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

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

Pass by reference

```
#!/usr/bin/python

# Function definition is here
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist = [1,2,3,4] # This would assig new reference in mylist
   print "Values inside the function: ", mylist
   return

# Now you can call changeme function
mylist = [10,20,30]
changeme( mylist )
print "Values outside the function: ", mylist
```

Pass by reference

- The parameter *mylist* is local to the function **changeme**.
- Changing *mylist* within the function does not affect *mylist*.

Pass by reference

- The function accomplishes nothing and finally this would produce the following result:

Values inside the function: [1, 2, 3, 4]

Values outside the function: [10, 20, 30]

Function Arguments

- You can call a function by using following types of formal arguments:
  - Required arguments
  - Keyword arguments
  - Default arguments
  - Variable-length arguments

Required arguments

- Required arguments are the arguments passed to a function in correct positional order.
- The number of arguments in the function call should match exactly with the function definition.

Required arguments

- To call the cunftion printme(), you definitely need to pass one argument, otherwise it gives a syntax error.

```
#!/usr/bin/python


def printme(str):
    print str
    return


printme()
```

```
Traceback (most recent call last):
  File "test.py", line 11, in <module>
    printme();
TypeError: printme() takes exactly 1 argument (0
given)
```

Keyword arguments

- Keyword arguments are related to the function calls.
- When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.
- This allows you to skip arguments or place them out of order.

Keyword arguments

```
#!/usr/bin/python

# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
   print str
   return;

# Now you can call printme function
printme( str = "My string")
```

Keyword arguments

```
#!/usr/bin/python

def printinfo( name, age ):
   "This prints a passed info into this function"
   print "Name: ", name
   print "Age ", age
   return

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

Name: miki
Age 50

Default arguments

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

Default arguments

```
#!/usr/bin/python

def printinfo( name, age = 35 ):
   "This prints a passed info into this function"
   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

- You may need to process a function for more arguments than you specified while defining the function.
- These arguments are called variable-length arguments and are not named in the function definition.

Variable-length arguments - Syntax

```
def functionname([formal_args,] *var_args_tuple ):
    "function_docstring"
    function_suite
    return [expression]
```

- 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.

Variable-length arguments - Syntax

```
def printinfo( arg1, *vartuple ):
    print "Output is: "
    print arg1
    for var in vartuple:
       print var
    return

printinfo(10)
printinfo(70, 60, 50)
```

Output is:
10
Output is:
70
60
50

The *return* statement

- 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.

The *return* statement

Inside the function : 30
Outside the function : 30

```
def sum(arg1, arg2):
    # Add both the parameters and return them."
    total = arg1 + arg2
    print "Inside the function : ", total
    return total


total = sum(10, 20);
print "Outside the function : ", total
```

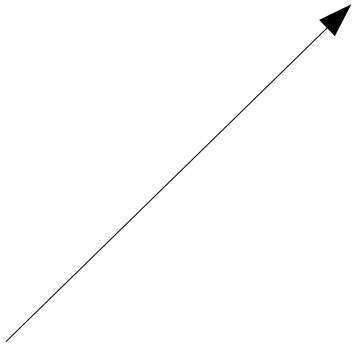
Multiple Functions

```
def functionA():
    statement1
    statementN

def functionB():
    statement1
    statementN

functionA()
functionB()
```

Multiple Functions

- It is a good convention to have the main action of a program be in a function for easy reference.

```
def functionA():
    statement1
    statementN

def functionB():
    statement1
    statementN
```



```
def main():
    functionA()
    functionB()

main()
```

Multiple Functions

```
import <modules>

def functionA():
    statement1
    statementN
    return

def functionB():
    statement1
    statementN
    return
```

```
def main():
    # your code here
    functionA()
    functionB()

if __name__ == "__main__":
    main()
```

Import Other Python scripts

$ sudo nano convert.py

```
NB_CM_IN_INCH = 2.54

def cm2inch(val):
    return val / NB_CM_IN_INCH

def inch2cm(val):
    return val * NB_CM_IN_INCH
```

Example 1 – **import-script.py**

```
import convert


cm = 10
inch = 3.93700787402
print cm, "cm. equal to", convert.cm2inch(cm), "inch"
print inch, "inch equal to", convert.inch2cm(inch), "cm"
```

Example 2 – **import-script.py**

```
from convert import *
# from convert import cm2inch, inch2cm


cm = 10
inch = 3.93700787402
print cm, "cm. equal to", cm2inch(cm), "inch"
print inch, "inch equal to", inch2cm(inch), "cm"
```

Example 3 – **import-script.py**

```
from convert import cm2inch as ci
from convert import inch2cm as ic


cm = 10
inch = 3.93700787402
print cm, "cm. equal to", ci(cm), "inch"
print inch, "inch equal to", ic(inch), "cm"
```

Argparse – calculator.py

```
import argparse

# def yourfunction():

def main():
    parser = argparse.ArgumentParser(description='Arguments')
    parser.add_argument('-f', '--first', dest='int1', required=True, help='first value')
    parser.add_argument('-s', '--second', dest='int2', required=True, help='second value')
    parser.add_argument('--version', action='version', version='%(prog)s version 0.1')
    args = vars(parser.parse_args())

   # your code here

if __name__ == "__main__":
main()
```

Lab1 – calculator.py

- สร้างโปรแกรมสำหรับทำการคำนวณ เช่น บวก ลบ คูณ หาร
- ในโปรแกรมประกอบด้วย 4 ฟังก์ชันหลัก คือ บวก ลบ คูณ และหาร
- ในแต่ละฟังก์ชันจะรับค่าตัวแปรจำนวน 2 ตัวแปร เช่น
  - def functionA(val1, val2)
- ในแต่ละฟังก์ชันสามารถ return ค่าที่ได้จากการคำนวณกลับออกไปยัง main function ได้
  - val3 = functionA(val1, val2)
- การเรียกใช้โปรแกรม
  - **./calculator -f 10 -s 15.2**
  - 10 + 15.2 = 25.2

Lab2 – calculator.py

- ต่อเนื่องจาก Lab1 หลังจากคำนวณเสร็จเรียบร้อย ให้เก็บผลลัพธ์ที่ได้จากการคำนวณลง File
- การเรียกใช้โปรแกรม
  - **$ ./calculator -f 10 -s 15.2 -o cal.txt**
  - 10 + 15.2 = 25.2
- และเมื่อเปิดไฟล์ขึ้นมาดูจะปรากฏข้อความดังนี้
  - **$ cat cal.txt**
  - **10 + 15.2 = 25.2**

References

- https://www.programiz.com/python-programming/function
- https://www.tutorialspoint.com/python/python_functions.htm
- http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/functions.html
- d