Basics of Programming

L06: Functions

Mar 2020

Dr. Ram P Rustagi
Professor, CSE Dept
KRP, KSGI
rprustagi@ksit.edu.in

Resources and Acknowledgements

- https://www.python-course.eu/passing_arguments.php
- A first course in programming
  - https://introcs.cs.princeton.edu/python/20functions/
- Python for everybody
  - https://www.py4e.com
-

Function: Parameter Passing

- Two common strategies
  - Call by value (aka pass by value)
  - Call by reference (aka pass by reference)
- Call by value
  - Used in C/C++
  - Used in Java/python for primitive variables
  - A local copy of variable/expression is passed
  - Changes can be made locally,
    - Does not affect original variable on return

Function: Parameter Passing

- Call by reference
  - The invoked function gets an implicit reference to the argument.
    - Does not get the copy of the argument.
    - Any changes made locally
      - Affect the original variable on return
    - Thus, funtion can modify the argument and it is reflected back on the argument in the calling function
  - Advantages & disadvantages
    - Time and space efficiency - no copying of arguments
      - Accidental changes in the function can be catastrophic
- Java: automatic passes a copy of reference to the object.

Parameter Passing in Python

- Call by value
  - Used for primitive types
- Call by object reference (similar to java)
  - Used for non-primtive types
  - A copy of reference is provided,
  - Immutable objects can't be changed.
    - Will result in error on changing
    - e.g. Tuples, strings
  - Mutable objects can be changed
    - Lists, dictionary etc.

Example: Pass by Obj Reference

```
def ops(obj):
   // could be any object e.g. list
   obj.append("new")  #modifies orig obj
   obj = [1,2,3]  # replaces the obj ref.
   // orig obj remain as before including modification

list=["a","b","c"]
ops(obj)
print(obj)  # print orig list + "new"
```

Pass by Obj Reference: string

```
def ops(str):
   // str in an immutable object.
   str.append("new")  #modifies orig obj

list=["a","b","c"]
ops(list) # works fine
print(list) #a, b, c, new
mystr="cse"
ops(mystr)   # gives error, string is immutable.
```

Function as Arguments

- function name is no different than other argument

```
def mops(fn,a,b):
  return fn(a,b)

def add(a,b):
  return a+b

def mult(a,b):
  return a*b

mops(add,5,10)
mops(mult,5,10)
```

Scope-1

- Scope of a variable within the namespace
  - local by default
  - variable does not exist outside the namespace
  - Namespace of a function is its code definition
- Example

```
def add(a,b):
   x = a + b
   return x

#main code
 y=add(5,7)
 print(y)
 print(x) # gives error, x is unknown
```

Scope-2

- Local declaration of variable within the function with same name as global
  - treated as local, does not affect global
- Example

```
def add(a,b):
   x = a + b # x is a local, different from global x
   return x

#main code
 x=10
 y=add(5,7)
 print(y)
 print(x)
```

Scope-3

- Scope of a variable within the namespace
  - Can use a global variablle inside a function.
- Example

```
def add(a,b):
   print(y) # y becomes global
   x = a + b
   return x
#main code
 x=10; y=20
 y=add(5,7)
 print(y)
 print(x)
```

Scope-4

- Scope of a variable within the namespace
  - Conflict of local vs global
- Example

```
def add(a,b):
   print(x) # x becomes global
   x = a + b #  confusion? is it local x? error
   return x
#main code
 x=10; y=20
 y=add(5,7)
 print(y)
 print(x)
```

Scope-5

- Scope of a variable within the namespace
  - Conflict of local vs global
- Example

```
def add(a,b):
   global x # x is declared as global
   print(x)
   x = a + b
   return x
#main code
 x=10; y=20
 y=add(5,7)
 print(y)
 print(x)
```

Scope-6

- Scope of a variable within the namespace
  - Conflict of local vs global
- Example

```
def add(a,b):
   global x # x is declared as global
   print(x)
   x = a + b
   return x
#main code
 y=20
 y=add(5,7)
 print(y)
 print(x)
```

Ex 01: Circular Prime

- Given an input positive integer M, check if it is circular prime i.e. when digits of this number are rotated by any number of positions, it is still a prime number.
- For example, number 1193 is circular prime because all of its rotations 1931, 9311, 3119 are prime numbers

Ex 02:

- Given a number N, compute its all binomial coefficients i.e.
  - $^{N}C_0$, $^{N}C_1$, $^{N}C_2$,…, $^{N}C_k$,…, $^{N}C_N$
  - The coefficient $^{N}C_k$ is defined as

    $^{N}C_k = n!/(k!*(n-k)!)$

Ex 03:

- Write a function that takes 3 arguments and returns the max of 3. Do not build into a list and then use sort() inbuilt cuntion.
  _

Ex 04:

- Write a function that takes a list as its arguments and returns the list that contains unique elements of the given list
  –

Ex 05:

- Write a function that checks if a given number is perfect or not. The program takes N as input and prints all perfect numbers <= N.
  - A perfect number is defined as a number which is equal to sum of all its factors (excluding itself) but including 1. for example
    - `6=1+2+3`
    - `28=1+2+4+7+14`

Questions