Introduction to Computer Science:

Python programming 2

April 2020

Honguk Woo

Contents

Question : variables in collection

- In general, assignment statements in Python do not copy objects, they create **bindings between a variable and an object**
- We can think that each element of a collection (e.g., d[0]) is a variable
  - E.g., a list is not so much a sequence of elements, as it is a sequence of references to elements

```
a = 3
b = 4
c = 5
d1 = [a, b, c]
d2 = d1
d1[0] = -1

print(a, d1[0], d2[0])
?

print(id(a), id(d1[0]), id(d2[0]))
?
```

Question : variables in collection

- tuple **d** is immutable. What if its element is a list ?

```
d = (3, 4, 5)
d[0] = -1
?

d = ([3, 4], [5, 8])
print(id(d[0]), id(d[1]))

d[0][1] = -2
print(d)
?
print(id(d[0]), id(d[1]))

a = d[0]
a[1] = -3
print(d)
?
print(id(d[0]), id(d[1]))
```

Discussion : *id of immutable, mutable*

- Different id values or not ?

```
>>> a = ["abc", "abc"]
>>> b = ["abc", "abc"]
>>> print(id(a), id(b))
?

>>> print(id(a[0]), id(b[0]))
?
```

Variable : global or local

- Global : a variable declared outside of the function
- Local : a variable declared inside the function's body

```
x = "global“

def foo():
   print("x inside :", x)

foo()
print("x outside:", x)
```

Question : global, local variable

- Global variable
- Local variable

```
def f():
        print(s)

s = “happy thanksgiving"
f()
```

```
def f():
        s = "happy valentine"
        print(s)

s = "happy thanksgiving"
f()
print(s)
```

```
def f():
        s = s + “and happy valentine"
        print(s)

s = "happy thanksgiving"
f()
print(s)
```

Discussion

- What's the ouput ?

```
x = 5
def my_function():
        print(x)
        x += 5
        print(x)

my_function()
```

Global and nonlocal

global keyword is to read and write a global variable inside a function.
nonlocal keyword is used in nested function whose local scope is not defined

```
def scope_test():
        def do_local():
                spam = "local spam"

        def do_nonlocal():
                nonlocal spam
                spam = "nonlocal spam"

        def do_global():
                global spam
                spam = "global spam"

        spam = "test spam"
        do_local()
        print("After local assignment:", spam)
        do_nonlocal()
        print("After nonlocal assignment:", spam)
        do_global()
        print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

Recursion : factorial

- Recursion is the process of defining something in terms of itself
  - Consisting of end condition (base condition) and recursive call

```
def factorial(n):
    if n == 1:
        return 1
    elif n > 1:
        return factorial(n-1)*n

>>> factorial(5)
120
```

Recursion : factorial

```
def factorial(n):
  print("factorial has been called with n = " + str(n))
  if n == 1:
    return 1
  elif n > 1:
    res = factorial(n-1)*n
    print("intermediate result for ", str(n), " * factorial(" , str(n-1) , "): ", str(res))
    return res
```

- **factorial(5)**

factorial has been called with n = 5 # factorial(5)

factorial has been called with n = 4 # factorial(4) * 5

factorial has been called with n = 3 # factorial(3) * 4 * 5

factorial has been called with n = 2 # factorial(2) * 3 * 4 * 5

factorial has been called with n = 1 # factorial(1) * 2 * 3 * 4 * 5

intermediate result for 2 * factorial(1): 2

intermediate result for 3 * factorial(2): 6

intermediate result for 4 * factorial(3): 24

intermediate result for 5 * factorial(4): 120

=> 120

Recursion : Fibonacci

- **Fibonacci**

  0,1,1,2,3,5,8,13,21,34,55,89, …

  $F_n = F_{n-1} + F_{n-2}$

  with $F_0 = 0$ and $F_1 = 1$

- **def fib(n):**

Recursion : Fibonacci

```
def fib(n):
        if n == 0:
                return 0
        elif n == 1:
                return 1
        else:
                return fib(n-1) + fib(n-2)
```

Argument passing : mutable, immutable

```
def dataCalc1(data):
  data = data + 1

def dataCalc2(data):
  data[0] = data[0] + 1

data1 = 1
data2 = [1]

dataCalc1(data1)
print(data1)
# ?

dataCalc2(data2)
print(data2)
# ?
```

Argument passing – mutable, immutable

- Immutable object parameter : similar to “call by value” in C
- Mutable object parameter : similar to “call by reference” in C

```
def dataCalc1(data):
    print(id(data1), id(data))
    data = data + 1
    print(id(data1), id(data))

def dataCalc2(data):
    print(id(data2), id(data))
    data[0] = data[0] + 1
    print(id(data2), id(data))

data1 = 1
data2 = [1]

dataCalc1(data1)
print(data1)

dataCalc2(data2)
print(data2)
```

```
// same object id for global, local
140723474576208  140723474576208
// local assignment create a new object (immutable)
140723474576208  140723474576240

// same object id for global, local
1765984910984  1765984910984
// local assignment updates the object (mutable)
1765984910984  1765984910984




1


[2]
```

Module

- Module : a file containing a python code

Import module	from module import function

`import calculator`
 	
`print(calculator.plus(10, 5))`
`print(calculator.minus(10, 5))`
`print(calculator.multiply(10, 5))`
`print(calculator.divide(10, 5))`
	`from calculator import plus`
	`from calculator import minus`
	`from calculator import multiply`
	`from calculator import divide`
	
	`print(plus(10, 5))`
	`print(minus(10, 5))`
	`print(multiply(10, 5))`
	`print(divide(10, 5))`

Module

- Advantages to **modularizing** code in a large application:
  - Simplicity
  - Maintainability
  - Reusability
  - Scoping

- Function, module, and class are all relevant to being modular

Module

- To create a module
  - Write code
  - Save as mymod.py

```
s = “do you like programming in python? "
a = [100, 200, 300]

def foo(arg):
    print(arg)
```

- Import the module and use it
  - import mymod
  - Then you can use, e.g.,
    - mymod.s
    - mymod.foo(“test”)

pygame : Module for game programming

- Install pygame on your environment
  - **Getting started**
    - https://www.pygame.org/wiki/GettingStarted
  - Tutorials
    - https://realpython.com/pygame-a-primer/

- Play with some sample codes
  - http://programarcadegames.com/index.php?chapter=example_code
  - https://www.pygame.org/tags/all

Quiz : power (r, n) = r^n

Implement **def power(r, n) :**

- Using for-statement, define powerF
- Using Recursion, define powerR

- Write code with the following test, e.g.,

```
print(powerF(2, 5))
print(powerR(4, 6))
```