Discover Your Data : Basic Programming I

Afif Akbar Iskandar,S.Si

Universitas Indonesia

afifai@sci.ui.ac.id

November 21, 2015

Overview

1. Introduction to Python
   - Interactive Shell
   - Variables, Expressions and Statements
   - Input & Output
   - Exercises

Why Python ?

- Code Readability
- Obvious simplicity
- Open extensibility
- Cross-platform runability
- Humanity

Python vs Java

Java

```
public class HelloWorld
{
    public static void main (String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

Python

```
print "Hello World!"
```

About "Shell"

- Great for Learning a Programming Language
- Great for Experimenting with a library
- Two variations : IDLE (GUI) / Python on Command Line

Interactive Shell

Try This

```
>>> print "Hello World !"
Hello World !
>>> a = 3
>>> b = input("Enter an Integer : )
Enter an Integer : 5
>>> a + b
8
>>> x = 11**2
>>> x - a
118
```

Variables

- Are not declared, just assigned
- The variable is created the first time you assign it a value
- Are references to objects
- Type information is with the object, not the reference
- Everything in Python is an object

Simple Data Types

- int : 7
- float : 7.0
- list : [1,2,3,4,5]
- tuple : (1,2,3,4,5)
- set : {1,2,3,4,5}
- string : "python"
- boolean : True, False

Compound Data Types

- list : [1,2,3,4,5]
- tuple : (1,2,3,4,5)
- set : {1,2,3,4,5}
- dict : {"name" : "Afif", "Age" : 15}

Math Operation

shell

```
>>> a = 100
>>> b = 10
>>> a + b #add
110
>>> a - b #substract
90
1000
>>> a / b #divide
10
>>> a**b #power
100000000000000000000L
```

String Operation

shell

```
>>> a = "LabMaKA
>>> "Lab"+"MaKA" #concatenation
"LabMaKA"
>>> a * 3 #repetition
"LabMaKALabMaKALabMaKA"
>>> a[2:3] #cutting
"bM"
>>> len(a) #String Size
7
>>> a[4] #Indexing
"a"
>>> "x" in a #Search
False
```

Lists

- Ordered collection of data
- Data can be of different types
- Lists are *mutable*
- Issues with shared references and mutability
- Same indexing operations as Strings

String Operation

shell

```
>>> a = [1,2,3]
>>> b = [4,5]
>>> a + b #concatenation
[1,2,3,4,5]
>>> a * 3 #repetition
[1,2,3,1,2,3,1,2,3]
>>> a[1:2] #cutting
[2,3]
>>> len(a) #String Size
3
>>> a[2] #Indexing
3
>>> 5 in a #Search
False
```

Lists : Modifying Content

- x[i] = a reassigns the ith element to the value a
- Since x and y point to the same list object, both are changed
- The method append also modifies the list

Example (Modifiying Content)

```
>>> x = [1,2,3]
>>> y = x
>>> x[1] = 100
>>> x
[1,100,3]
>>> y
[1,100,3]
>>> x.append(12)
>>> y
[1,100,3,12]
```

Lists : Modifying Content

shell

```
>>> a = [1,2,3]
>>> a.count(2)
1
>>> a.extend([1,2,3,4,5])
>>> a
[1,2,3,1,2,3,4,5]
>>> a.sort()
[1,1,2,2,3,3,4,5]
>>> a.pop()
5
>>> a
[1,1,2,2,3,3,4]
```

Tuple

- Tuples are immutable versions of lists
- Faster than list

Example (Tuple)

```
>>> x = (1,2,3)
>>> x.count(1)
1
>>> x.index(3)
2
>>> x.append(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute
 'append'
```

Dictionaries

- A set of key-value pairs
- Dictionaries are *mutable*

Example (Dictionaries)

```
>>> d = {1 : "tes", "two" : 2, "blah" : [1,2,3]}
>>> d
{1:"tes","two":2,"blah":[1,2,3]}
>>> d["two"]
2
```

Dictionaries : Add/Modify

- Entries can be changed by assigning to that entry
- Assigning to a key that does not exist adds an entry

Example (Add/Modify)

```
>>> d = {1 : "tes", "two" : 2, "blah" : [1,2,3]}
>>> d["two"] = 99
>>> d
{1:"tes","two":99,"blah":[1,2,3]}
>>> d["two"]
2
>>> d[7] = "new entry"
>>> d
{1:"tes",7:"new entry","two":99,"blah":[1,2,3]}
```

Dictionaries : Deleting Elements

- The **del** method deletes an element from a dictionary

Example (Deleting Elements)

```
>>> d = {1 : "tes", "two" : 2, "blah" : [1,2,3]}
>>> del(d[1])
>>> d
{"two":2,"blah":[1,2,3]}
```

Input & Output

- The **raw_input(string)** method returns a line of user input as a string
- The parameter is used as a prompt
- The string can be converted by using the conversion methods **int(string)**, **float(string)**, etc.

Example (Input & Output)

```
>>> a = raw_input("Input : ")
>>> a
Input : LabMaKA
>>> a
"LabMaKA"
```

Try This!

inputoutput.py

```
print "What's your name?"
name = raw_input("> ")
print "What year were you born?"
birthyear = int(raw_input("> "))
print "Hi %s!" %(name)
print "You are %d years old!" %(2015 - birthyear)
```

File Input

fileinput.py

```
f = open("data.txt","r") #open file "data" for input
S1 = f.read() #read whole data into one string
f.close() #close a file
f = open("data.txt","r")
N=10 #define N
S2 = f.read(N) #Reads N bytes (N >= 1)
f.close()
f = open("data.txt","r")
S3 = f.readline() #Returns a list of line strings
f.close()

print S1
print S2
print S3
```

File Output

fileoutput.py

```
S1 = "Hello"
S2 = "World"
L = [S1,S2]
f = open("testout.txt","w")
f.write(S1) #Write the string S1 to file
f.write(S2) #Write the string S2 to file
f.write("\n") #write a new line notation to file
f.writelines(L) #Write each of the strings in list L
f.close()
```

If Statements

Syntax :

```
if (boolean):
    statement
elif (boolean):
    statement
else:
    statement
```

Example (ifstatements.py)

```
a = 3
b = 7
if a < b:
    print "b is greater than a"
else:
    print "a is greater than b"
```

While Loops

Syntax :

```
while (boolean):
    statement
```

Example (whileloops.py)

```
a = 0
b = 10
while a < b:
    print "a = %d" %(a)
    a = a+1 #update a value
```

For Loops

Syntax :

```
for itervar in list:
    statement
```

Example (forloops1.py)

```
a = [5,4,2,6,2]
for i in a:
    print i**2
```

Example (forloops2.py)

```
a = [5,4,2,6,2]
b = [0,0,0,0,0]
for i in range(len(a)):
  b[i] = a[i]**2
print b
```

Loop Control Statement

- break — Jumps out of the closest enclosing loop
- pass — Jumps to the top of the closest enclosing loop
- continue — Does nothing, empty statement placeholder

Break Example

Example (break1.py)

```
for letter in 'Python':
   if letter == 'h':
      break
   print 'Current Letter :', letter
```

Example (break2.py)

```
var = 10
while var > 0:
   print 'Current variable value :', var
   var = var -1
   if var == 5:
      break
```

Pass Example

Example (passexample.py)

```
for letter in 'Python':
   if letter == 'h':
      pass
      print 'This is pass block'
   print 'Current Letter :', letter
```

Continue Example

Example (continue1.py)

```
for letter in 'Python':
   if letter == 'h':
      continue
   print 'Current Letter :', letter
```

Example (continue2.py)

```
var = 10
while var > 0:
   var = var -1
   if var == 5:
      continue
   print 'Current variable value :', var
```

Exercise

Write a program able to play the "Guess the number"-game, where the number to be guessed is randomly chosen between 1 and 20. (Source: http://inventwithpython.com).

guessthenumber.py

```
from random import randint

rand = randint(1,20)

#write your code here
```

Exercise

output example

```
Hai! Siapa namamu?
>> Afif
Nah, Afif, saya memikirkan angka dari 1 sampai 20.
Ayo tebak!
>> 7
Tebakanmu terlalu besar.
Ayo tebak lagi!
>> 2
Tebakanmu terlalu kecil.
Ayo tebak lagi!
>> 3
Bagus, Afif! Kamu bisa menebak dalam 3 kali tebakan!
```

Solution

tebakangka.py

```
from random import randint
rand = randint(1,20)
i = 0
print "Hai! Siapa namamu?"
nama = raw_input(">> ")
print "Nah, %s, Sekarang saya memikirkan angka \
dari 1 sampai 20." %(nama)
print "Ayo tebak!"
tebakan = input(">> ")
i = i+1
```

Solution (cont.)

tebakangka.py

```
while tebakan != rand:
  if tebakan < rand:
    print "Tebakanmu terlalu kecil. "
    print "Ayo tebak lagi!"
    tebakan = input(">> ")
    i = i+1
  else :
    print "Tebakanmu terlalu besar. "
    print "Ayo tebak lagi!"
    tebakan = input(">> ")
    i = i+1
print "Bagus, %s! Kamu bisa menebak \
angkaku dalam %d kali tebakan!" %(nama,i)
```