Python for JNTUK Students

Sreekanth Kolamala

M.tech Computer Science, IIT Kanpur

Types

Python has 5 data types-Numbers, Strings, Lists, Tuple, Dictionary. In this chapter, we will discuss Numbers and Strings. We will see the other 3 in the next chapter.

Numbers

Numbers are further divided into 4:int, long, float, complex. int and long are used to represent whole numbers. float is used for numbers that have a decimal part. Languages like C have double type for really big floating point numbers. But python only has float.

```python
a = 12
print type(a)
#type is a built-in function that returns the type of a variable
b = 12.3
print type(b)
c = 2323231211142342424
print type(c)
d = complex(2,3)
print d
print type(d)
```

```
## <type 'int'>
## <type 'float'>
## <type 'long'>
## (2+3j)
## <type 'complex'>
```

The data type complex represent mathematical complex numbers that have a real and an imaginary part.

Strings

Python doesn't differete between single quotes and double quotes. Unlike in C, the is not difference between char ans string-both are same.

```python
name = "sreekanth"
occupation = 'software engineer'
print type(name)
qoute = "I'am good" #observe that a single qoute can be used within double quotes
print "qoute:" + qoute
qoute = "qoute:" + 'He said "I am not going" to her'
print qoute
```

```
## <type 'str'>
## qoute:I'am good
## qoute:He said "I am not going" to her
```

```
qoute = "He said "I am not going" to her" #this will not work.
```

Booleans

Boolean variables can be assigned either `True` or `False`

```
flag = True
print flag
flag = False
print flag
```

```
## True
## False
```

Operators

Variables are assigned values using the `=` operator. Variables are written when they are first assigned a value.

```
number = 10
```

The variable `number` is assigned 10. Note that the type of variable `number` is not declared. This is not required because of the python type inference. Based on the operations we do on a variable, the python will automatic guess its type.

```
#multiple assignments can be done in the same statement simultaneously.
a, b = 10,20
#a is assigned 10 and b is assigned 20.
print a,b
a,b = b,a
#a is assigned the value in b and b is assigned the value in a simultaneously.
#this is a popular way of swapping numbers in python.
print a,b
```

```
## 10 20
## 20 10
```

Arithmetic Operators

They are + - * / % **. `a%b` is gives the reminder of when a is divided by b. `a**b` gives a to power of b.

operator	example	result
+	2 + 3	5
-	2.3 - 3.5	-1.2
*	2*1.5	3
/	5/2	2
%	5%2	1
**	5**2	25

Logical Operators

operator	example	result
>	20 > 40	False
	20 > 20	False
>=	20 >= 20	True
<	20 < 40	True
	20 < 20	False
<=	20 <= 20	True
==	20 == 20	True
	20 == 40	False
!=	20 != 20	False
	20 != 40	True

Logical Operators

There are three logical operators: not, or, and. Also, >, >=, <, <=, ==, !=

```
condition = True
print (not condition)
print (condition or False)
print (condition and False)
```

```
## False
## True
## False
```

`not a` evaluates to the opposite of what `a` is. `a and b` evaluates to True only if both a, b are True. a or b evaluates to False if both a,b are False. They behave exactly like !, ||, && operators in C.

a	not a
True	False
False	True

a	b	a and b
True	True	True
True	False	False
False	True	False
False	False	False

a	b	a or b
True	True	True
True	False	True
False	True	True
False	False	False

Membership Operator

The `in` operator is used to determine variable consists of other.

```python
isPresent = "sree" in "sreekanth"
print isPresent
```

True

We will discuss this operator in more detail in unit-3 once we complete lists.

```python
fruits = ["apple", "banana", "orange"]
print "apple" in fruits
#the 'in' operator can be used with the 'not' operator
#to find out if a variable is not present in another
print "banana" not in fruits
```

True
False

Terary operator

    a if condition else b

```python
number = 24
print "even" if number % 2 else "odd"
```

odd

Control flow

```python
a,b = 3,2
if(a > b):
  print "a is greater than b" #Notice this print stmt is indented
  #this means that the print stmt is within the if block
```

a is greater than b

Python seperates blocks based on there indentation. Notice the priNotice unlike other langauges, python doesn't use curly brackets{} to block. The above if statement can also be written as if-else statement.

```python
a,b = 10,20
if a > b:
  print "a is greater than b"
else:
  pass
```

pass is a keyword that represents an empty statement. It's is like an empty ; in C.

```python
a,b = 3,2
if a > b:
  print "this statement is within the if block"
  print "this statement aswell"
else:
  print "this statement is in the else block"
  print "this statement is also in the else block"
print "this statement is not in the if block. it's in the global block and will always print"
```

this statement is within the if block
this statement aswell

```
## this statement is not in the if block. it's in the global block and will always print
```

Loops

The for loop

```python
numbers = range(1,10)
for number in numbers:
  print number
```

```
## 1
## 2
## 3
## 4
## 5
## 6
## 7
## 8
## 9
```

```python
for aphlabet in "sreekanth":
  print aphlabet
```

```
## s
## r
## e
## e
## k
## a
## n
## t
## h
```

```python
fruits = ["apple", "banana", "orange"]
while len(fruits) != 0:
  print fruits.pop()
print fruits
```

```
## orange
## banana
## apple
## []
```

```python
numbers = range(0,10) #remember range() creates a list [0,1,2...,9]
#program to print even numbers
for i in numbers:
  if i%2 != 0:
    continue
  print i #remember indentation rules: this stmt is in for block but not in if block
print "program end"
```

```
## 0
## 2
## 4
## 6
```

```
## 8
## program end
```

Each time the keyword `continue` is hit, the remaining part of the for loop is not executed

```python
numbers = range(0,10)
for i in numbers:
  if i%2 != 0:
    break
  print i
print "program end"
```

```
## 0
## program end
```

Notice that this program produces nothing. When `i=1`, the `if` condition is True, and break statement is hit. When a break statement is hit, the loop, the break statement is in, is immediately exited.

Lists

lists are similar to C arrays but more powerful. Lists store multiple values. unlike C arrays, elements in a python list don't have be of the same type. lists are indexed from 0.

```python
fruits = ['apple', 'mango', 'pineapple']
numbers = [10, 20, 30, 40, 50]
#elements in a list can be accessed using index.
print numbers[1]
#even negative index is allowed, in which case length of list is added
#to the index.
print numbers[-1]
fruitsandnumbers =  ['apple', 'mango', 'pineapple', 10, 20, 30, True]
#mixing elements of different types is fine
print fruitsandnumbers
print len(numbers)
#len is a built-in function used to find the length of lists.
integers = range(1,10)
print integers
#range(start,last) is built-in function that creates a list [start, start+1, ..., last-1]
```

```
## 20
## 50
## ['apple', 'mango', 'pineapple', 10, 20, 30, True]
## 5
## [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Slicing

Slicing is a simple way of creating a new list from (part of) an old list.

```
    newList = oldList[start:end]
```

```python
numbers = [10, 20, 30, 40, 50, 60, 70]
sublist = numbers[1:4]
print sublist
print numbers[0:3]
```

```
print numbers[:3] #missing starting number, with replaced with 0
print numbers[4:7]
print numbers[4:] #missing ending number is replaced with length of the list.
#negative indices can also be given.
#In which case, the length of the list is added
print numbers[-2:7]
print numbers[:-1]  #this will print all elements except the last one.
```

```
## [20, 30, 40]
## [10, 20, 30]
## [10, 20, 30]
## [50, 60, 70]
## [50, 60, 70]
## [60, 70]
## [10, 20, 30, 40, 50, 60]
```

As shown in the example, sublist is a new list consisting of elements 1 to 3(remember, 'end' is not inclusive) in the numbers list. you can also specify negative numbers for indexes in which case, the length of list is automatically added. For example `numbers[-2:7]` is equivalent to `numbers[5:7]`.

List methods

Lists are not fixed in length. They increase and decrease when elements are added and removed.

```
numbers = [10,20,30,40,50]  #creating a list
numbers.append(60)  #adds 60 at the end of the list
print numbers
numbers.insert(2, 25) #you can even add an element at a specific index
print numbers
```

```
## [10, 20, 30, 40, 50, 60]
## [10, 20, 25, 30, 40, 50, 60]
```

Notice how the list gets re-adjusted. All the elements after index 1 are moved 1 place right the new element is insert at index 2. you can delete an element from a list by using the keyword `del`.

```
numbers = [10,20,30,40,50]
del numbers[1] #this will remove element at index 1
print numbers
```

```
## [10, 30, 40, 50]
```

Notice how the list is re-adjusted. All the elements after index 1 are moved 1 place left to full the list. The length of list is also reduced by 1.

```
numbers = [10,20,30,40,50]  #creating a list
print numbers.pop() #removes and returns the last element in the list
print numbers
# More methods
print numbers.index(40) #returns the index of the element 40 if present
numbers = [30,20,40,10,50]
numbers.sort()  #sorts list
print numbers
```

```
## 50
## [10, 20, 30, 40]
```

7

```
## 3
## [10, 20, 30, 40, 50]
```

Comprehensions

```
[expression(x) for x in some_list if condition]
```

expression(x) is the formula used for creating the elements. The elements of some_list is are used in expression(x). The `if condition` is also optional used for filtering. Some examples will solidify these concepts.

```
squares = [x**2 for x in range(10)]
print squares
#squares of even numbers:
squares = [x**2 for x in range(10) if x % 2 == 0]
print squares
squares = [x**2 for x in range(10) if not x % 2]
print squares
```

```
## [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
## [0, 4, 16, 36, 64]
## [0, 4, 16, 36, 64]
```

Sets

Sets are like lists but duplicates are not allowed.

```
presentees = {1,5,6,7,10,1,6,6,5}
print presentees
presentees.add(10)
print presentees
```

```
## set([1, 10, 5, 6, 7])
## set([1, 10, 5, 6, 7])
```

Common mathematical set operations like union, intersection can be easily done.

```
setA = {1,2,3,4,5}
setB = {4,5,6,7,8}
print setA.union(setB)
print setA.intersection(setB)
```

```
## set([1, 2, 3, 4, 5, 6, 7, 8])
## set([4, 5])
```

Dictionaries

Dictionaries(commmonly shortened to dict) are {key,value} pair data structures. Each element in a dict is a pair of key and value. Keys have to be uquine. Keys can be thought as indices with names. Dicts are similar to java maps.

```python
wordFrequency = {'a':1, 'b':5, 'c':10, 'e':15}
#Here all the strings('a', 'b', 'c', 'e' are keys)
#all the numbers are values(1,5,10,15)
#each key has a associated value.
#For example, wordFrequency can be used to store frequency of alphabets in a string
print wordFrequency['a']
#key can be used like an index of a list.
del wordFrequency['a']
#deleting is similar to list.
print wordFrequency
print 'd' in wordFrequency  #find out if a key exists in dict using the 'in' keyword
```

```
## 1
## {'c': 10, 'b': 5, 'e': 15}
## False
```

Accessing the key that isn't present in the dict can result in an exception. This can be avoided by checking if that key is present before accessing it.

```python
wordFrequency = {'a':1, 'b':5, 'c':10, 'e':15}
if('d' in wordFrequency):
  print wordFrequency['d']
else:
  print "not present"
```

```
## not present
```

Functions

Functions are small pieces of code that we can reuse by calling them instead of writing code multiple times. Their syntax:

```python
def function_name(parameters):
  stmt
  stmt
  stmt
```

Few examples as usual will make things clear:

```python
#function defination
def add(a,b):
  c = a + b
  print c
```

We define a function by using the keyword `def`. Notice that all the stmts after the first statement are indented. This is python's way to say that these statements are part of the function defination and not in the global context. Not indenting would throw an error. We can call this function like this:

```python
#function calling
add(2,3)
add(2.3, 4.5)
add(a=2.3, b=4.5) #we can explicitly use the names of the parameter
                  #as defined in function defination
add(b=4.5, a=2.3) #we can also change the order of parameters.
```

```
## 5
```

```
## 6.8
## 6.8
## 6.8
```

Also notice that we were able to use the same add() function to add both ints and floats. Something we can't do in languages like C or Java.

Default parameters

We could give default values to function parameters. In which case, if the function is called without the parameter, the default value is used. If an actual parameter is given then the default one is ignored.

```python
def print_series(n=10):
  for i in range(n):
    print i
print_series(5) #this will print till 5 #mental note: write this into 2 programs and eval, echo blocks
print_series()  #since no parameter is given, the default value 10 is assigned to n
                #and 0 till n are printed.
```

```
## 0
## 1
## 2
## 3
## 4
## 0
## 1
## 2
## 3
## 4
## 5
## 6
## 7
## 8
## 9
```

Fruitful Functions

Instead of printing, a function can also return a value.

```python
#function that returns the square of its parameter
def square(number):
  return number**2
#function calling
a = square(4) #the value returned by the function can be assigned to a variable.
print a
```

```
## 16
```

```python
def square(number):
  return number**2
print map(square, range(10))
```

```
## [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Instead of defining a function, we can use anonymous function. Anonymous function are like regular functions except they don't have a name. They defined using the keyword, `lambda`.

```
print map(lambda x: x**2, range(10))

## [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Strings as lists

In python, strings are lists. So, all the list operations can be done on strings.

```
name = "sreekanth"
print name[0]
print name[1:5] #slicing
print name + "kolamala" # concatination
#

## s
## reek
## sreekanthkolamala
```

Lab programs

Exercise 2 - Operations

a) Write a program to compute distance between two points taking input from the user (Pythagorean Theorem)

```
import math
x1 = int(raw_input("enter x1:"))
y1 = int(raw_input("enter y1:"))
x2 = int(raw_input("enter x2:"))
y2 = int(raw_input("enter y2:"))
distance = math.sqrt(math.pow(x2-x1, 2) + math.pow(y2-y1, 2))
print distance
```

b) Write a program add.py that takes 2 numbers as command line arguments and prints its sum.

Exercise 8 - Functions

a) Write a function ball_collide that takes two balls as parameters and computes if they are colliding. Your function should return a Boolean representing whether or not the balls are colliding. Hint: Represent a ball on a plane as a tuple of (x, y, r), r being the radius If (distance between two balls centers) <= (sum of their radii) then (they are colliding)

b) Find mean, median, mode for the given set of numbers in a list.

```
def mean(numbers):
    return float(sum(numbers)) / max(len(numbers), 1)

def median(numbers):
    if not numbers:
        return 0
    sortedNumbers = sorted(numbers)
    length = len(sortedNumbers)
    return (sortedNumbers[length/2] + sortedNumbers[(length-1)/2])/2.0
```

```
#if you are going to sort numbers, there is simpler way to find mode
def mode(numbers):
    frequencies = countWordFrequency(numbers)
    maximum = None
    result = None
    for number, frequency in frequencies.items():
        if maximum < frequency:
            maximum = frequency
            result = number
    return result


numbers = [2,3,5,2,10,25]
print mean(numbers), median(numbers), mode(numbers)
```

My text!

R Markdown

```
1  > print("hello world")
```

[1] "hello world"

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

Including Plots

You can also embed plots, for example:

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

https://stackoverflow.com/questions/36808263/separate-columns-for-text-and-code-output-in-markdown?noredirect=1&lq=1