Arithmetic Operations

Modulo Operations

```
In [1]: 0%3, 1%3, 2%3, 3%3, 4%3, 5%3, 6%3, 7%3, 8%3, 9%3, 10%3
Out[1]: (0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1)
```

Observe that there are 3 elements repeating (0, N-1) where N is modulo divisor.

Complex Numbers ¶

Complex Number = Real Number +/- Imaginary Number

```
In [2]: 2+3j
Out[2]: (2+3j)

In [3]: n1 = 2+3j

In [4]: type(n1)
Out[4]: complex

In [5]: n2 = 3-2j

In [6]: type(n2)
Out[6]: complex

In [7]: print n1, n1 + n2, n1-n2, n1*n1, pow(n1,2), n1/n1
        (2+3j) (5+1j) (-1+5j) (-5+12j) (-5+12j) (1+0j)

In [8]: print n1, n1.conjugate()
        (2+3j) (2-3j)

In [9]: print n1, n1.real, n1.imag
        (2+3j) 2.0 3.0

In [10]: n2 = 2.0-3.45j
```

In [11]:
```
print n2, n2.real, n2.imag
```
```
(2-3.45j) 2.0 -3.45
```

In [12]:
```
4j
```
Out[12]:
```
4j
```

4j, j4, j4 are not possible. In these cases, interpreter treats 'j' as a variable.

In [13]:
```
print n1+n2, n1-n2, -n1+n2, -n1-n2
```
```
(4-0.45j) 6.45j -6.45j (-4+0.45j)
```

In [15]:
```
print n1*n2.real, (n1*n2).real
```
```
(4+6j) 14.35
```

In [16]:
```
print n1, n2, n1.real+n2.imag   # n2.imag is resulting in a real value
```
```
(2+3j) (2-3.45j) -1.45
```

In [17]:
```
complex(2,-3.456)  # Builtin function
```
Out[17]:
```
(2-3.456j)
```

.* (of Matlab) operator is not valid in python. Element-wise multiplication is possible with numpy module. floor division is also not valid on complex type data.

In [18]:
```
(3+4j) == (4j+3)  # == checks value equivalence
```
Out[18]:
```
True
```

In [19]:
```
(3+4j) is (4j+3)  # is - checks object level (both value and address)
```
Out[19]:
```
False
```

In [20]:
```
print n1, abs(n1)  # abs - absolute function, Builtin function.
```
```
(2+3j) 3.60555127546
```

abs((a+bj)) is equal to math.sqrt(pow(a,2), pow(b,2))

In [23]:
```
print abs(3+4j)
```
```
5.0
```

In [24]:
```
import math; print math.sqrt(3**2+4**2)
```
```
5.0
```

In [25]:
```
divmod(12+2j, 2-3j) # divmod(x,y) returns x//y, x%y
```

```
c:\python27\lib\site-packages\ipykernel\__main__.py:1: DeprecationWarning: co
mplex divmod(), // and % are deprecated
  if __name__ == '__main__':
```

Out[25]: ((1+0j), (10+5j))

Compound(mixed) Operations

+=, -=, *=, **=

In [28]:
```
a = 12; print a, type(a)
```

```
12 <type 'int'>
```

In [29]:
```
a = a + 1 ; print "a = ", a
```

```
a =  13
```

In [30]:
```
a += 1 ; print "a = ", a
```

```
a =  14
```

In [31]:
```
a -= 1 ; print "a = ", a   # a = a -1
```

```
a =  13
```

In [32]:
```
a *= 2 ; print "a = ", a  # a = a*2
```

```
a =  26
```

In [33]:
```
a /= 2 ; print "a = ", a  # a = a / 2
```

```
a =  13
```

In [34]:
```
a **= 2 ; print "a = ", a  # a = a ** 2
```

```
a =  169
```

In [35]:
```
a %= 100 ; print "a = ", a
```

```
a =  69
```

In [36]:
```
a = 23; a//=2; print "a = ", a
```

```
a =  11
```

In [37]:
```
a<<=1; print "a = ", a   #left-shift
```

```
a =  22
```

```
at binary level 16 8 4 2 1
                   1 0 1 1
       <<1      1  0 1 1 0
```

In [39]:
```
a>>=1; print "a = ", a   #right-shift
```

```
a =  11
```

In [40]:
```
a^=2; print "a = ", a   # bitwise XOR operation
```

```
a =  9
```

In [41]:
```
a|=2; print "a = ", a   # bitwise OR operation
```

```
a =  11
```

Pre- and Post- increment/ decrements (++a, a++, --a, a--) are not valid in Python

IO Operations

In python 2.x, raw_input() and input() are two builtin functions used for getting runtime input.

```
raw_input()  - takes any type of runtime input as a string.

input()      - takes any type of runtime input originally without any type conversi
on.


NOTE: working with raw_input() requires us to use type converters to convert the da
ta into the required data type.
```

In Python 3.x, there is only input() function; but not raw_input(). The Job of raw_input() in python 2.x is done by input() in python 3.x

In [137]:
```
#!/usr/bin/python

# class3_io.py

'''
    Purpose : demonstration of input() and raw_input()
'''

dataRI = raw_input('Enter Something: ')

dataI = input('Enter something: ')

print dataRI, type(dataRI)

print dataI, type(dataI)
```

```
Enter Something: 123
Enter something: 123
123 <type 'str'>
123 <type 'int'>
```

Analyzed outputs for various demonstrated cases:

```
>>>
========== RESTART: C:/pyExercises/class3_io.py ==========
Enter Something: 123
Enter something: 123
123 <type 'str'>
123 <type 'int'>
>>>
========== RESTART: C:/pyExercises/class3_io.py ==========
Enter Something: 'Yash'
Enter something: 'Yash'
'Yash' <type 'str'>
Yash <type 'str'>
>>>
========== RESTART: C:/pyExercises/class3_io.py ==========
Enter Something: True
Enter something: True
True <type 'str'>
True <type 'bool'>
>>>
========== RESTART: C:/pyExercises/class3_io.py ==========
Enter Something: Yash
Enter something: Yash

Traceback (most recent call last):
  File "C:/pyExercises/class3_io.py", line 12, in <module>
    dataI = input('Enter something: ')
  File "<string>", line 1, in <module>
NameError: name 'Yash' is not defined
>>> dataRI
'Yash'
```

input() takes only qualified data as runtime input. Whereas **raw_input()** will qualify any data as a 'str' type

In [138]:
```
#!/usr/bin/python

# class3_io1.py

'''
    Purpose : demonstration of input() and raw_input()
'''

dataRI = int(raw_input('Enter a number: '))

dataI = input('Enter a number: ')


print dataRI, type(dataRI)

print dataI, type(dataI)

print "Sum of numbers is ", dataRI+dataI
```

```
Enter a number: 123
Enter a number: 123
123 <type 'int'>
123 <type 'int'>
Sum of numbers is  246
```

Analyzed outputs for various demonstrated cases:

```
========== RESTART: C:/pyExercises/class3_io1.py ==========
Enter a number: 123
Enter a number: 123
123 <type 'str'>
123 <type 'int'>
>>>
========== RESTART: C:/pyExercises/class3_io1.py ==========
Enter a number: 123
Enter a number: 123
123 <type 'str'>
123 <type 'int'>
Sum of numbers is

Traceback (most recent call last):
  File "C:/pyExercises/class3_io1.py", line 19, in <module>
    print "Sum of numbers is ", dateRI+dataI
NameError: name 'dateRI' is not defined
>>>
========== RESTART: C:/pyExercises/class3_io1.py ==========
Enter a number: 123
Enter a number: 123
123 <type 'str'>
123 <type 'int'>
Sum of numbers is

Traceback (most recent call last):
  File "C:/pyExercises/class3_io1.py", line 19, in <module>
    print "Sum of numbers is ", dataRI+dataI
TypeError: cannot concatenate 'str' and 'int' objects
>>>
========== RESTART: C:/pyExercises/class3_io1.py ==========
Enter a number: 123
Enter a number: 123
123 <type 'int'>
123 <type 'int'>
Sum of numbers is  246
>>>
```

String Operations

string data type can be representing using either single or double quotes

Creating a string

In [42]:
```
s1 = 'Python Programming'
```

In [43]:
```
print s1
```
```
Python Programming
```

In [44]:
```
print type(s1)
```
```
<type 'str'>
```

In [46]:
```
s2 = "Django"
```

In [47]:
```
print s2, type(s2)
```
```
Django <type 'str'>
```

In [48]:
```
s3 = ''' python programming with Django '''
```

In [49]:
```
print s3
```
```
 python programming with Django
```

In [50]:
```
print type(s3)
```
```
<type 'str'>
```

In [51]:
```
s4 = """ python programming with Django  """
```

In [52]:
```
print s4
```

```
 python programming with Django
```

In [53]:
```
type(s4)
```

Out[53]:
```
str
```

In [54]:
```
django1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-54-220f950d4f1f> in <module>()
----> 1 django1

NameError: name 'django1' is not defined
```

In [55]:
```
'django1'
```

Out[55]:
```
'django1'
```

In [56]:
```
type('django1')
```

Out[56]:
```
str
```

In [59]:
```
s5 = '~!@#$%^&*()1232425'
```

In [60]:
```
print s5, type(s5)
```

```
~!@#$%^&*()1232425 <type 'str'>
```

In [61]:
```
s6 = str(123.34)  # str() is a builtin function to convert to string
```

In [62]:
```
print s6, type(s6)
```

```
123.34 <type 'str'>
```

In [63]:
```
s7 = str(True)
```

In [64]:
```
print s7, type(s7)
```

```
True <type 'str'>
```

Indexing

In [65]:
```
s1
```

Out[65]:
```
'Python Programming'
```

In [66]:
```
len(s1)   # len() is a bultin function to return the length of object
```

Out[66]: 18

```
P y t h o n   P r o g  r  a  m  m  i  n  g
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17  -> forward indexing
                          -5 -4 -3 -2 -1  -> Reverse indexing
```

In [68]:
```
s1[0]
```

Out[68]: 'P'

In [69]:
```
s1[6]
```

Out[69]: ' '

In [70]:
```
s1[17]
```

Out[70]: 'g'

In [71]:
```
s1[18]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-71-8486803f392a> in <module>()
----> 1 s1[18]

IndexError: string index out of range
```

In [72]:
```
s1[-1]
```

Out[72]: 'g'

In [73]:
```
s1[-5]
```

Out[73]: 'm'

In [74]:
```
s1[-16]
```

Out[74]: 't'

In [75]:
```
s1[-18] == s1[0]
```

Out[75]: True

Interview Question 1: what is string[-0]

In [76]:
```
s1[-0]  # is equal to s1[0]
```

Out[76]: 'P'

String Slicing

```
In [77]: s1[2:6]
```

```
Out[77]: 'thon'
```

```
In [78]: s1[2:8]
```

```
Out[78]: 'thon P'
```

```
In [79]: s1[:]
```

```
Out[79]: 'Python Programming'
```

```
In [80]: s1[:-1]
```

```
Out[80]: 'Python Programmin'
```

```
In [83]: s1[-5:-1]
```

```
Out[83]: 'mmin'
```

```
In [84]: s1[-5:17]    # complex indexing
```

```
Out[84]: 'mmin'
```

```
In [81]: s1[::-1]
```

```
Out[81]: 'gnimmargorP nohtyP'
```

```
In [82]: s1[::1]
```

```
Out[82]: 'Python Programming'
```

```
In [85]: s1[::2]
```

```
Out[85]: 'Pto rgamn'
```

```
In [86]: s1[::3]
```

```
Out[86]: 'Ph oai'
```

```
In [87]: s1[::4]
```

```
Out[87]: 'Poran'
```

```
In [88]: s1[4:9]
```

```
Out[88]: 'on Pr'
```

In [89]:
```
s1[4:9:1]     # string[initialBound, finalBound, increment/decrement]
```

Out[89]:
```
'on Pr'
```

In [90]:
```
s1[4:9:-1]        # 4-1 = 3  index 3 is not represented in this object
```

Out[90]:
```
''
```

NOTE: After all these alterations, the original string object will not change, until it is overwrited.

Mutability of Strings

In [91]:
```
print s1
```

```
Python Programming
```

In [92]:
```
s1[3]
```

Out[92]:
```
'h'
```

In [93]:
```
s1[3] = 'H'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-93-a32a26674f0e> in <module>()
----> 1 s1[3] = 'H'

TypeError: 'str' object does not support item assignment
```

String objects are Immutable. They, can't be edited. Only way is to overwrite it

In [94]:
```
s1 = "PyTHON PROGRAMMING"
```

In [95]:
```
s1         # object overwriting taken place
```

Out[95]:
```
'PyTHON PROGRAMMING'
```

String attributes

In [97]:
```
print dir(s1)
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
 '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
 '__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt
__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex_
_', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str_
_', '__subclasshook__', '_formatter_field_name_split', '_formatter_parser',
 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtab
s', 'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'i
sspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partitio
n', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translat
e', 'upper', 'zfill']
```

In [98]:
```
s1
```

Out[98]: 'PyTHON PROGRAMMING'

In [99]:
```
s1.capitalize()
```

Out[99]: 'Python programming'

In [100]:
```
s1.count('m')
```

Out[100]: 0

In [102]:
```
s1.count('M')   # python is case- sensitive, and capitalize() created new obje
ct, without disturbing the original object
```

Out[102]: 2

In [103]:
```
s1.endswith('ing')  # endswith() returns the boolen result
```

Out[103]: False

In [104]:
```
s1.endswith('ING')
```

Out[104]: True

In [136]:
```
s1.startswith('Py')
```

Out[136]: True

In [105]:
```
s1.find('P')
```

Out[105]: 0

In [106]:
```
s1
```

Out[106]: 'PyTHON PROGRAMMING'

In [107]:
```
s1.find('THON')
```

Out[107]: 2

In [108]:
```
s1.find('MM')
```

Out[108]: 13

In [109]:
```
s1.find('M')
```

Out[109]: 13

In [111]:
```
s1.index('THON')
```

Out[111]: 2

Assignment 2: difference between s1.find and s1. index? Also, 'rfind', 'rindex'

In [112]:
```
s1
```

Out[112]: 'PyTHON PROGRAMMING'

In [113]:
```
s1.capitalize()
```

Out[113]: 'Python programming'

In [114]:
```
s1.lower()
```

Out[114]: 'python programming'

In [115]:
```
s1.upper()
```

Out[115]: 'PYTHON PROGRAMMING'

In [116]:
```
s1.title()
```

Out[116]: 'Python Programming'

In [117]:
```
s1.swapcase()
```

Out[117]: 'pYthon programming'

Interview Question : what is the data type of result of string.split()

In [118]:
```
s1.split(' ')
```

Out[118]: ['PyTHON', 'PROGRAMMING']

In [119]:
```
s1.split('O')
```

Out[119]: ['PyTH', 'N PR', 'GRAMMING']

In [120]:
```
s1
```

Out[120]:
```
'PyTHON PROGRAMMING'
```

In [121]:
```
s1.split('N')         # string to list conversion
```

Out[121]:
```
['PyTHO', ' PROGRAMMI', 'G']
```

In [122]:
```
s1.split('r')    # no splitting as there is no 'r' character, but 'R' characte
r in string s1
```

Out[122]:
```
['PyTHON PROGRAMMING']
```

In [123]:
```
s1.split('y')
```

Out[123]:
```
['P', 'THON PROGRAMMING']
```

In [124]:
```
len(s1.split('y'))
```

Out[124]:
```
2
```

In [125]:
```
''.join(s1.split('y'))    # list to strinsg conversion
```

Out[125]:
```
'PTHON PROGRAMMING'
```

In [126]:
```
'@'.join(s1.split('y'))   # delimiter can be placed
```

Out[126]:
```
'P@THON PROGRAMMING'
```

In [127]:
```
s1.split('O')
```

Out[127]:
```
['PyTH', 'N PR', 'GRAMMING']
```

In [129]:
```
'@'.join(s1.split('O'))   # Observe that 'O' is replaced by '@'.   This is one
 example of duck-typing
```

Out[129]:
```
'PyTH@N PR@GRAMMING'
```

In [131]:
```
s9  = '''
        This is a good day!
        Fall 7 times, raise 8!
        This is a famous japanese quote.
            '''
```

In [132]:
```
print len(s9), s9
```

```
114
        This is a good day!
        Fall 7 times, raise 8!
        This is a famous japanese quote.
```

In [135]:
```
print 'IS'.join(s9.split('is'))
```

```
ThIS IS a good day!
Fall 7 times, raISe 8!
ThIS IS a famous japanese quote.
```

In [139]:
```
s1
```

Out[139]: 'PyTHON PROGRAMMING'

In [140]:
```
s1.isalpha()
```

Out[140]: False

In [141]:
```
'python'.isalpha()
```

Out[141]: True

In [142]:
```
'python programming'.isalpha()  # As there is a space character also
```

Out[142]: False

In [143]:
```
'python'.isalnum()
```

Out[143]: True

In [144]:
```
'python123'.isalnum()
```

Out[144]: True

In [145]:
```
'python123 '.isalnum() # There is a white space character also
```

Out[145]: False

In [146]:
```
'python123'.isdigit()
```

Out[146]: False

In [147]:
```
'123'.isdigit()
```

Out[147]: True

In [148]:
```
'python'.islower()
```

Out[148]: True

In [149]:
```
'python123$'.islower()   # It ensures that there is no capital letter, only
```

Out[149]: True

In [150]:
```
'python123$ '.isspace()
```

Out[150]: False

In [152]:
```
' '.isspace()
```

Out[152]: True

In [153]:
```
''.isspace()
```

Out[153]: False

In [154]:
```
'python programming'.isupper()
```

Out[154]: False

In [155]:
```
'PYTHON'.isupper()
```

Out[155]: True

In [156]:
```
'PyTHoN'.isupper()
```

Out[156]: False

In [157]:
```
'PyTHoN'.upper().isupper()    # operator precedence rules applies. Left to rig
ht operation takes place
```

Out[157]: True

In [158]:
```
s1
```

Out[158]: 'PyTHON PROGRAMMING'

In [159]:
```
s1.istitle()
```

Out[159]: False

In [160]:
```
(s1.title()).istitle()
```

Out[160]: True

In [161]:
```
'   Python  Programming  '.rstrip()
```

Out[161]: '   Python Programming'

In [162]:
```
'   Python  Programming  '.lstrip()
```

Out[162]: 'Python Programming '

In [163]:
```
'   Python  Programming  '.strip()   # removes whitespaces
```

Out[163]: 'Python Programming'

In [164]:
```
'   Python  Programming  '.strip('ing')
```

Out[164]: '   Python Programming '

In [166]:
```
'   Python  Programming  '.strip().strip('ing')
```
Out[166]: 'Python Programm'

In [167]:
```
'ad123da'.strip('a')
```
Out[167]: 'd123d'

In [168]:
```
'ad1a2a3ada'.strip('a')  # middle characters will retain
```
Out[168]: 'd1a2a3ad'

In [169]:
```
'01\t012\t0123\t01234'.expandtabs()  # recognizes \t espace character
```
Out[169]: '01   012   0123   01234'

In [170]:
```
'01012012301234'.expandtabs()
```
Out[170]: '01012012301234'

To get the website name, excluding the domain

In [175]:
```
'www.python.org'.lstrip('www.').rstrip('.org')
```
Out[175]: 'python'

In [176]:
```
'www.python.org'.split('.')[1]
```
Out[176]: 'python'

In [177]:
```
word = 'Python'
```

In [178]:
```
len(word)
```
Out[178]: 6

In [179]:
```
word.zfill(6)    # numbers less than len(word) doesn't show any affect
```
Out[179]: 'Python'

In [180]:
```
word.zfill(7)     # zeros will be prepended correspondingly
```
Out[180]: '0Python'

In [181]:
```
word.zfill(len(word)+4)
```
Out[181]: '0000Python'

In [184]:
```
'@'.join((word.zfill(len(word)+4).split('0')))    # filling with character '@'
```
Out[184]: '@@@@Python'

In [185]:
```
'Python\tProgramming\t'.expandtabs()   # recognizes '\t', '\r' and '\r\t'; and
 expands correspondingly whenever it finds '\t'
```

Out[185]:
```
'Python  Programming     '
```

In [186]:
```
r'Python\tProgramming'.expandtabs()
```

Out[186]:
```
'Python\\tProgramming'
```

In [187]:
```
'Python\r\tProgramming'.expandtabs()
```

Out[187]:
```
'Python\r        Programming'
```

In [188]:
```
print 'Python\r\tProgramming'.expandtabs()
```

```
        Programming
```

In [189]:
```
'Python\t\rProgramming'.expandtabs()
```

Out[189]:
```
'Python  \rProgramming'
```

In [190]:
```
print 'Python\t\rProgramming'.expandtabs()
```

```
Programming
```

Assignment : Try to practice the remaining attributes, in this order.

split splitlines rsplit

partition rpartition

find rfind

index rindex

center ljust rjust

encode decode translate