1 - Python Basics

December 7, 2015

1 Global concepts & operators

Operators that work on multiple Python objects (Python enables operator overloading). Essential python concepts.

1.1 Operators

Assignment operator =

```
In [ ]: int_    = 5                                    # int
        float_  = 4.0                                  # float
        string_ = "hello"                              # string
        list_   = [1, 2, 3]                            # list
        dict_   = {'Boss': 'Pat', 'Engineer': 'Harry'} # dict
        set_    = {1, 2}                               # set
        tuple_  = (1, 2, 3)                            # tuple
```

```
In [ ]: # print operator --> Python 3: print() function
        print int_
        print float_
        print string_
        print list_
        print dict_
        print set_
        print tuple_
```

Concatenation operator +

```
In [ ]: int_ + float_
```

```
In [ ]: string_ + " world!"
```

```
In [ ]: list_ + [3, 4, 5, 6]
```

```
In [ ]: tuple_ + (4, 5, 6)
```

Repetition operator *

```
In [ ]: int_ * float_
```

```
In [ ]: string_ * 3
```

```
In [ ]: list_ * 3
```

```
In [ ]: tuple_ * 3
```

Comparison operator ==

```
In [ ]: # equal ==
        print int_    == 6
        print float_  == 3.0
        print string_ == "hell"
        print list_   == [1, 3, 4]
        print dict_   == {'Boss': 'Pat', 'Engineer': 'Bob'}
        print set_    == {1, 2, 3}
        print tuple_  == (1, 2, 3, 4)
```

```
In [ ]: # superior
        print int_    > 10
        print float_  > 10.0
        print string_ > "jello"
```

```
In [ ]: # inferior
        print int_    < 10
        print float_  < 10.0
        print string_ < "jello"
```

Other operators

`type(obj)`

```
In [ ]: # type(obj)
        print type(int_)
        print type(float_)
        print type(string_)
        print type(dict_)
        print type(set_)
        print type(tuple_)
```

```
In [ ]: # check type
        type(int_) == float
```

`isinstance(obj, type)`

```
In [ ]: # isinstance(obj, type)
        print isinstance(int_, int)
        print isinstance(float_, float)
        print isinstance(string_, str)
        print isinstance(list_, list)
        print isinstance(dict_, dict)
        print isinstance(set_, set)
        print isinstance(tuple_, tuple)
```

`len(obj)`

```
In [ ]: # len(obj)
        len(list_), len(dict_), len(set_), len(tuple_)
```

`obj in other_obj`

```
In [ ]: # in
        'hello' in string_, 1 in list_, 2 in set_, 1 in tuple_
```

Object existence or non-emptyness

```
In [ ]: # object existence or non-emptyness
        if string_:
            print "String is defined and is not empty !"
        if list_:
            print "List is defined and not empty !"
        if set_:
            print "Set is defined and not empty"
        if tuple_:
            print "Tuple is defined and not empty"
```

1.2 Loops

FOR

```
In [ ]: # loop through string
        for char in string_:
            print char
```

```
In [ ]: # loop through list
        for obj in list_:
            print obj
```

```
In [ ]: # loop through dictionary keys
        for key in dict_:
            print key
```

```
In [ ]: # loop through set values
        for val in set_:
            print val
```

Following works only on enumerables (string, list, dictionaries, set)

```
In [ ]: # loop through string WITH INDEX
        for index, obj in enumerate(string_):
            print index, obj
```

```
In [ ]: # loop through list WITH INDEX
        for index, obj in enumerate(list_):
            print index, obj
```

```
In [ ]: # loop through dict WITH INDEX
        for index, value in enumerate(dict_):
            print index, value
```

```
In [ ]: # loop through set WITH INDEX
        for index, value in enumerate(set_):
                print index, value
```

Following works only on iterables (dictionaries)

```
In [ ]: # loop through dict WITH KEY AND VALUES
        for key, value in dict_.iteritems():
            print key, value
```

WHILE

```
In [ ]: i = 1
        while len(list_) != 0:
            print "Popping element %d" % i
            list_.pop()
            i+=1
        print "List empty !"
```

1.3 Cast

int –> str

```
In [ ]: # int --> string
        str(5)
```

str –> int

```
In [ ]: # string --> int
        int("4096")
```

list –> set

```
In [ ]: # list --> set
        set([1, 2, 2, 3])
```

dict –> list

```
In [ ]: # dict --> list
        dict_ = {'John': 22, 'Frank': 35}
        print list(dict_.keys())
        print list(dict_.values())
        print list(dict_.items())
```

tuple –> str

```
In [ ]: # tuple --> string
        tuple_ = ('Paul', 42)
        print list(tuple_)
        print str(tuple_)
```

1.4 Print

Python 2.7

```
In [3]: print "Hello World !"
```

```
Hello World !
```

with arguments

```
In [2]: string = "World !"
        for i in range(10):
            print "%d. Hello %s" % (i, string)
```

```
0. Hello World !
1. Hello World !
2. Hello World !
3. Hello World !
4. Hello World !
5. Hello World !
6. Hello World !
7. Hello World !
8. Hello World !
9. Hello World !
```

with a comma

```
In [5]: string = "World !"
        print "Hello", string
```

```
Hello World !
```

with +

```
In [6]: print "Hello" + string
```

```
HelloWorld !
```

Python 3

In Python 3, the print statement has been replaced by a function. This function has more features that the print statement.

To get it in Python 2.7, we can import it with:

```
In [7]: from __future__ import print_function
```

```
In [8]: print("Hello, world !")
```

```
Hello, world !
```

2 Numbers

Integers, floats.

Assignment =

```
In [9]: # int
        a = 5
        c = 2047
```

```
In [10]: # float
         b = 4.0
         d = 3.0
```

Operations

```
In [11]: # Addition
         a + b
```

```
Out[11]: 9.0
```

```
In [12]: # Substraction
         a - b
```

```
Out[12]: 1.0
```

```
In [13]: # Product
         a * b
```

```
Out[13]: 20.0
```

```
In [14]: # Quotient
         a / b
```

```
Out[14]: 1.25
```

```
In [15]: # Quotient (integer)
         a / c

Out[15]: 0

In [16]: # Quotient (float)
         float(a)/c

Out[16]: 0.002442598925256473

In [17]: # Floored quotient
         a // b

Out[17]: 1.0

In [18]: # Modulo (remainder)
         a % b

Out[18]: 1.0

In [19]: # Power
         a ** b

Out[19]: 625.0

In [20]: # In place (+=, -=, *=, **=)
         print a
         print b
         print

         a += b  #add
         print a

         a -=b   #substract
         print a

         a /= b  #divide
         print a

         a *= b  #product
         print a

         a **=b  #power
         print a

         a = int(a)
         print a


          File "<ipython-input-20-e5571d409d0a>", line 2
        print a
              ^
    SyntaxError: invalid syntax
```

Bitwise operators

```
In [21]: # Binary shift (left)
         a << 1
```

```
Out[21]: 10
```

```
In [22]: # Binary shift (right)
         a >> 1
```

```
Out[22]: 2
```

```
In [23]: # Binary 'and'
         a & c
```

```
Out[23]: 5
```

```
In [24]: # Binary 'or'
         a | c
```

```
Out[24]: 2047
```

```
In [25]: # Binary 'or exclusive'
         a ^ c
```

```
Out[25]: 2042
```

```
In [26]: # Binary complement
         ~a
```

```
Out[26]: -6
```

3 Strings “”

A string is a set of characters - ordered, immutable.

Assignement =

```
In [27]: string1 = 'hello'
```

```
In [28]: string2 = 'world'
```

Concatenation +

```
In [29]: string1 + string2
```

```
Out[29]: 'helloworld'
```

```
In [30]: string1 + ' world'
```

```
Out[30]: 'hello world'
```

```
In [31]: 'hello' + ' world'
```

```
Out[31]: 'hello world'
```

```
In [32]: print "hello" + ',' + "\n" + "\t world !"
```

```
  File "<ipython-input-32-0647411aac7a>", line 1
    print "hello" + ',' + "\n" + "\t world !"
                ^
SyntaxError: invalid syntax
```

Indexing [ ]

```
In [33]: string1[0]

Out[33]: 'h'

In [34]: string1[2]

Out[34]: 'l'

In [35]: string1[-1]

Out[35]: 'o'

In [ ]: string1[-2]

In [ ]: # Assignemnt on index: TYPERROR EXCEPTION --> strings are NOT mutable
        string1[-2]='a'
```

Slicing [:]

```
In [ ]: # Slicing interval - outter bound excluded: [a, b[
        string1[1:4]

In [ ]: string1[0:3]

In [ ]: string1[-3:]

In [ ]: # with step [a:b:step]
        string1[1:5:2]

In [ ]: # with step [a:b:step]
        string1[0::2]

In [ ]: # reverse a string
        string1[::-1]
```

String Methods

`upper()`

```
In [ ]: 'helLO'.upper()
```

`lower()`

```
In [ ]: 'HEllO'.lower()
```

`capitalize()`

```
In [ ]: 'hello world'.capitalize()
```

`isdigit()`

```
In [ ]: '2015'.isdigit()
```

`split()` - converts string to list

```
In [ ]: split1 = 'hello world'.split()       #delimiter is space (default)
        split2 = 'hello, world'.split(',')  #delimiter is ','
        print split1
        print split2
```

4 Lists [ ]

A list contains objects of any kind - ordered, mutable

Assignment =

```
In [ ]: # Empty list
        list1 = []
        list1

In [ ]: # List of integers
        list1 = [1, 2, 3, 4, 5]
        list1

In [ ]: # List of strings
        list2 = ['a', 'b', 'c', 'd']
        list2

In [ ]: # List of mixed types
        list3 = [1.0, 'a', 2, dict()]
        list3

In [ ]: # Multidimensional
        list4 = [[1, 2, 3], ['a', 'b', 'c']]
        list4
```

Concatenation +

```
In [ ]: list1 + list2

In [ ]: list1 + list3

In [ ]: list2 + list3
```

Indexing [ ]

```
In [ ]: # Normal indexing
        list1[3]

In [ ]: # Normal indexing
        list4[0]

In [ ]: # Reverse indexing
        list1[-2]

In [ ]: # Reverse indexing
        list4[-1]

In [ ]: # Assignment on index
        print list2
        list2[0] = "new_value"
        list2
```

Slicing [:]

```
In [ ]: list1[2:4]

In [ ]: # from index -2 to end
        list1[-2:]
```

```
In [ ]: list1[1:5]

In [ ]: # with step
        # from index -4 to end with step = 2
        list1[-4::2]

In [ ]: # reverse a list
        list1[::-1]
```

List Methods

`insert(position, value)`

```
In [ ]: # insert(position, value)
        list1.insert(2, 4)
        list1
```

`append(value)`

```
In [ ]: # append(value)
        list2.append("at_the_end")
        list2
```

`remove(value)`

```
In [ ]: # remove(value)
        list2.remove('new_value')
        list2
```

`pop()`

```
In [ ]: # pop()
        list2.pop()
```

`extend(other_list)`

```
In [ ]: # extend(other_list)
        list3.extend(list4)
        list3

In [ ]: # 'delimiter'.join(list) --> converts string to list
        ' '.join(['C', 'C', 'C', '', 'I', 'N', 'F','O','R','M','A','T','I','O','N', '', 'S', 'E', 'R',
```

List comprehension

```
In [ ]: # Don't type this ...
        list2 = []
        for a in list1:
            list2.append(a+1)

        print list2

In [ ]: # ... Type this !
        list2 = [a+1 for a in list1]

        print list2
```

More examples of list comprehension

```
In [ ]: # Following sounds familiar ? Get all even numbers in list
        list2 = []
        for e in list1:
            if e % 2 == 0:
                list2.append(e)

        print list2
```

```
In [ ]: # Why not do that instead ? List comprehension \O/
        list2 = [e for e in list1 if e % 2 == 0]

        print list2
```

```
In [ ]: # Don't type this ...
        list2 = []
        for sub_list in list4:
            for sub in sub_list:
                if type(sub) is int:
                    list2.append(sub)

        print list2
```

```
In [ ]: # ... Type this !
        list2 = [sub for sub_list in list4 for sub in sub_list if type(sub) is int]

        print list2
```

```
In [ ]: # ... intellectually the same
        [sub for sub_list in list4
                 for sub in sub_list
                     if type(sub) is int]

        print list2
```

```
In [ ]: # We can also reassign in-place
        list4 = [s for e in list4 for s in e if type(s) is str]

        print list4
```

5 Dictionaries

A dictionary is a (key, value) store - mutable, unordered, unique

Assignment =

```
In [ ]: # empty dict
        dict1 = {}
        dict1 = dict()
        print dict1
```

```
In [ ]: # initialize dict
        dict1 = {'John': [22, 'architect'],
                 'Marc': [35, 'boss']
                }
        print dict1
```

Indexing [ ]

```
In [ ]: dict1['John']

In [ ]: dict1['Marc']

In [ ]: dict1['Frank']

In [ ]: # Add new key, value
        dict1['Frank'] = [20, 'intern']
        dict1

In [ ]: # NO INDEXING BY NUMBER
        dict1[0] # --> KeyError Exception
```

Dictionary Methods

keys()

```
In [ ]: # Keys
        dict1.keys()
```

values()

```
In [ ]: # Values
        dict1.values()
```

items()

```
In [ ]: # Items (keys and values)
        dict1.items()
```

Loops

```
In [ ]: # Loop through keys
        for key in dict1:
            print key

In [ ]: # Loop through keys (old way)
        for key in dict1.keys():
            print key

In [ ]: # Loop through values
        for val in dict1.values():
            print val

In [ ]: # Loop through both keys AND values
        for key, value in dict1.iteritems(): #PYTHON 3: iteritems() --> items()
            print key, value
```

6 Sets

A dictionary with no values, only keys - immutable, unordered, unique

Assignment =

```
In [ ]: # empty set
        set0 = set()
        set0
```

```
In [ ]: # initialize set
        set1 = {1, 2, 3, 4, 5}
        set1

In [ ]: # intialize set
        set2 = set(['a','b','c','d'])
        set2

In [ ]: # initialize set from list
        list1 = [1, 2, 3, 4, 5, 5]
        set3 = set(list1)
        set3
```

Concatenation |=

```
In [ ]: # Concatenate two sets
        set1 |= set2
        set1

In [ ]: # Concatenate two sets
        set1.update(set2)
        set1
```

Indexing

```
In [ ]: # TYPEERROR EXCEPTION : NO INDEXING FOR SETS
        set1[1]
```

Set Methods

`add(element)`

```
In [ ]: set1.add(6)
        set1
```

`remove(element)`

```
In [ ]: set1.remove(6)
        set1
```

`discard(element)`

```
In [ ]: set1.discard(1)
        set1
```

`pop()` - pops **arbitrary** element from set.

```
In [ ]: set1.pop()
        set1
```

`union(other_set)`

```
In [ ]: set1.union(set3)
```

`intersection(other_set)`

```
In [ ]: set1.intersection(set3)
```

`difference(other_set)`

```
In [ ]: set1.difference(set3)
```

`symmetric_difference(other_set)`

```
In [ ]: set1.symmetric_difference(set2)
```

`clear()`

```
In [ ]: set1.clear()
        set1
```

7 Tuples ( )

An arbitrary group of elements.

Assignment =

```
In [ ]: tuple1 = (1, 2, 3)
        tuple1
```

```
In [ ]: tuple2 = tuple([1, 2, 3, 4])
        tuple2
```

```
In [ ]: tuple3 = tuple("Hello world !")
        tuple3
```

```
In [ ]: tuple4 = ("Hello world !",)
        tuple4
```

```
In [ ]: tuple5 = tuple({'John': 14, 'Marc': 22})
        tuple5
```

```
In [ ]: tuple6 = tuple(list1)
        tuple6
```

Concatenation +

```
In [ ]: tuple1 + tuple2
```

```
In [ ]: tuple1 + tuple3
```

```
In [ ]: tuple2 + tuple3
```

Indexing [ ]

```
In [ ]: tuple1[0]
```

```
In [ ]: tuple1[-1]
```

Slicing [:]

```
In [ ]: tuple1[0:2]
```

```
In [ ]: tuple2[-2:]
```

```
In [ ]: tuple3[1::2]
```

Tuple unwrapping

```
In [ ]: a, b, c, d, e, f = tuple6
```

```
        print a, b, c, d, e, f
```