1 - Python Basics

December 4, 2015

1 Global operators

Operators that work on multiple Python objects (Python enables operator overloading)

1.1 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_
```

1.2 Concatenation operator " + "

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

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

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

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

1.3 Repetition operator " * "

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

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

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

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

1.4 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"
```

1.5 Other operators

```
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
```

```
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)
```

Following works only for containers (list, dict, set)

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

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

```
In [ ]: # object existence
        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"
```

1.6 Loops

1.6.1 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
```

Enumerables (string, list, dictionaries)

```
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
```

Iterables (dictionaries)

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

1.6.2 WHILE

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

1.7 Cast

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

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

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

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

2 Numbers

Integers, floats.

2.1 Assignment =

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

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

2.2 Operations

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

In [ ]: # Substraction
        a - b

In [ ]: # Product
        a * b

In [ ]: # Quotient
        a / b

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

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

In [ ]: # Floored quotient
        a // b

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

```
In [ ]: # Power
        a ** b

In [ ]: # 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
```

2.3 Bitwise operators

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

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

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

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

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

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

3 Strings “”

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

3.1 Assignement =

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

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

3.2 Concatenation +

```
In [ ]: string1 + string2

In [ ]: string1 + ' world'

In [ ]: 'hello' + ' world'

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

3.3 Indexing [ ]

```
In [ ]: string1[0]

In [ ]: string1[2]

In [ ]: string1[-1]

In [ ]: string1[-2]

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

3.4 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]
```

3.5 String Methods

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

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

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

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

In [ ]: # split() --> converts string to list
        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

4.1 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
```

4.2 Concatenation +

```
In [ ]: list1 + list2

In [ ]: list1 + list3

In [ ]: list2 + list3
```

4.3 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
```

4.4 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]
```

4.5 List Methods

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

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

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

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

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',
```

4.6 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

5.1 Assignment =

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

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

5.2 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
```

5.3 Dictionary Methods

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

In [ ]: # Values
        dict1.values()

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

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

6.1 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
```

6.2 Concatenation |=

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

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

6.3 Indexing

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

6.4 Set Methods

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

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

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

```
In [ ]: # pop() --> POPS ARBITRARY ELEMENT FROM SET (UNORDERED)
        set1.pop()
        set1
```

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

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

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

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

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

7 Tuples ( )

An arbitrary group of elements.

7.1 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
```

7.2 Concatenation +

```
In [ ]: tuple1 + tuple2

In [ ]: tuple1 + tuple3

In [ ]: tuple2 + tuple3
```

7.3 Indexing [ ]

```
In [ ]: tuple1[0]

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

7.4 Slicing [:]

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

In [ ]: tuple2[-2:]

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

7.5 Tuple unwrapping

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

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