Step0 - Introduction and Operators

May 8, 2018

My First Python program

```
In [9]: print "Hello, Python World!"

Hello, Python World!
```

Script Mode Programming: You can put the above code into a notepad, save the file with extension .py (say script.py) and run it from command line interface using below command.
python script.py

Python Identifier: Identifier is the name given to entities like class, functions, variables etc. in Python. It helps differentiating one entity from another.
Rules for writing identifiers in Python:
Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_). Names like myClass, var_1 and print_this_to_screen, all are valid example. An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine. Keywords cannot be used as identifiers. We cannot use special symbols like !, @, #, $, % etc. in our identifier

```
In [10]: # variable name can not start with digit in front, 1variable will through syntax erro
         1variable = 'Identifier can not start with number'


          File "<ipython-input-10-ca4ded31a0f4>", line 2
        1variable = 'Identifier can not start with number'
                ^
    SyntaxError: invalid syntax
```

```
In [ ]: # using keywords as variable name will through error
        class = 1
```

```
In [ ]: # using special symbols as variable name will through error
        x@gmail = 'm.s@gmail.com'
```

Basic Objec Types

```
In [19]: none = None # singleton null object
         boolean = bool(True)
         integer = 1
         Long = 3.14

         # float
         Float = 3.14
         Float_inf = float('inf')
         Float_nan = float('nan')

         # complex object type, note the usage of letter j
         Complex = 2+8j

         # string can be enclosed in single or double quote
         string = 'this is a string'
         me_also_string = "also me"

         List = [1, True, 'ML'] # Values can be changed

         Tuple = (1, True, 'ML') # Values can not be changed

         Set = set([1,2,2,2,3,4,5,5]) # Duplicates will not be stored

         # Use a dictionary when you have a set of unique keys that map to values
         Dictionary = {'a':'A', 2:'AA', True:1, False:0}

         # lets print the object type and the value
         print type(none), none
         print type(boolean), boolean
         print type(integer), integer
         print type(Long), Long
         print type(Float), Float
         print type(Float_inf), Float_inf
         print type(Float_nan), Float_nan
         print type(Complex), Complex
         print type(string), string
         print type(me_also_string), me_also_string
         print type(Tuple), Tuple
         print type(List), List
         print type(Set), Set
         print type(Dictionary), Dictionary
```

```
<type 'NoneType'> None
<type 'bool'> True
<type 'int'> 1
<type 'float'> 3.14
<type 'float'> 3.14
```

```
<type 'float'> inf
<type 'float'> nan
<type 'complex'> (2+8j)
<type 'str'> this is a string
<type 'str'> also me
<type 'tuple'> (1, True, 'ML')
<type 'list'> [1, True, 'ML']
<type 'set'> set([1, 2, 3, 4, 5])
<type 'dict'> {'a': 'A', True: 1, 2: 'AA', False: 0}
```

Comments in Python All characters after the # and up to the physical line end are part of the comment and the Python interpreter ignores them Comments spanning more than one line are achieved by inserting a multi-line string (with """ as the delimiter one each end) that is not used in assignment or otherwise evaluated, but sits in between other statements

```
In [ ]: # This is a comment in Python
        print "Hello Python World" # This is also a comment in Python

        """ This is an example of a multi line
        comment that spans multiple lines.
        Everything that is in between is considered as comments
        """
```

Multiple statementson a single line

```
In [ ]: import sys; x = 'foo'; print x
```

Code Blocks

```
In [ ]: # Correct indentation
        print ("Programming is an important skill for Data Science")
        print ("Statistics is a imporant skill for Data Science")
        print ("Business domain knowledge is a important skill for Data Science")

        # Correct indentation, note that if statement here is an example of suites
        x = 1
        if x == 1:
            print ('x has a value of 1')
        else:
            print ('x does NOT have a value of 1')
```

```
In [ ]: # incorrect indentation, program will generate a syntax error
        # due to the space character inserted at the beginning of second line
        print ("Programming is an important skill for Data Science")
         print ("Statistics is a important skill for Data Science")
        print ("Business domain knowledge is a important skill for Data Science")
```

```
# incorrect indentation, program will generate a syntax error
# due to the wrong indentation in the else statement
x = 1
if x == 1:
    print ('x has a value of 1')
else:
 print ('x does NOT have a value of 1')
```

Milti-line statements

```
In [ ]: # Example of implicit line continuation
        x = ('1' + '2' +
            '3' + '4')

        # Example of explicit line continuation
        y = '1' + '2' + \
            '11' + '12'

        weekdays = ['Monday', 'Tuesday', 'Wednesday',
                    'Thursday', 'Friday']

        weekend = {'Saturday',
                   'Sunday'}

        print ('x has a value of', x)
        print ('y has a value of', y)
        print days
        print weekend
```

Basic Operators

Arithmetic operators Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

```
In [ ]: # Variable x holds 10 and variable y holds 5
        x = 10
        y = 5

        # Addition
        print "Addition, x(10) + y(5) = ", x + y

        # Subtraction
        print "Subtraction, x(10) - y(5) = ", x - y

        # Multiplication
        print "Multiplication, x(10) * y(5) = ", x * y

        # Division
        print "Division, x(10) / y(5) = ", x / y

        # Modulus
        print "Modulus, x(10) % y(5) = ", x % y

        # Exponent
        print "Exponent, x(10)**y(5) = ", x**y

        # Integer division rounded towards minus infinity
        print "Floor Division, x(10)//y(5) = ", x//y
```

Comparison operators Comparison operators are used to compare values. It either returns True or False according to the condition.

```
In [ ]: # Variable x holds 10 and variable b holds 5
        # Equal check operation
        print "Equal check, x(10) == y(5) ", x == y

        # Not Equal check operation
        print "Not Equal check, x(10) != y(5) ", x != y

        # Not Equal check operation
        print "Not Equal check, x(10) <> y(5) ", x <> y

        # Less than check operation
        print "Less than check, x(10) < y(5) ", x < y

        # Greater check operation
        print "Greater than check, x(10) > y(5) ", x > y

        # Less than or equal check operation
        print "Less than or equal to check, x(10) <= y(5) ", x <= y

        # Greater than or equal to check operation
        print "Greater than or equal to check, x(10) >= y(5) ", x >= y
```

Assignment operations

```
In [ ]: # Variable x holds 10 and variable y holds 5
        x = 5
        y = 10

        x += y
        print "Value of a post x+=y is ", x

        x *= y
        print "Value of a post x*=y is ", x
        x /= y
        print "Value of a post x/=y is ", x

        x %= y
        print "Value of a post x%=y is ", x

        x **= y
        print "Value of x post x**=y is ", x

        x //= y
        print "Value of a post x//=y is ", x
```

Bitwise operations Bitwise operators act on operands as if they were string of binary digits. As the name suggest, it operates bit by bit.

```
In [ ]: # Basic six bitwise operations
        # Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)
        x = 10
        y = 4

        print x >> y   # Right Shift
        print x << y   # Left Shift
        print x & y    # Bitwise AND
        print x | y    # Bitwise OR
        print x ^ y # Bitwise XOR
        print ~x    # Bitwise NOT
```

Logical operators Logical operators are the and, or, not operators.

```
In [ ]: x = True
        y = False
        print('x and y is',x and y)
        print('x or y is',x or y)
        print('not x is',not x)
```

Membership operators in and not in are the membership operators in Python. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary). In a dictionary we can only test for presence of key, not the value

```
In [ ]: var1 = 'Hello world'          # string
        var1 = {1:'a',2:'b'}          # dictionary
        print('H' in var1)
        print('hello' not in var2)
        print(1 in var2)
        print('a' in var2)
```

Identity operatros is and is not are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal do not imply that they are identical

```
In [ ]: var1 = 5
        var1 = 5
        var2 = 'Hello'
        var2 = 'Hello'
        var3 = [1,2,3]
        var3 = [1,2,3]
        print(var1 is not var1)
        print(var2 is var2)
        print(var3 is var3)
```

0.1 Control structuresű

A control structure is the basic decision-making process in computing. It is a block of programming that analyzes variables and chooses a direction in which to go based on given parameters

```
In [14]: var = -1
         if var < 0:
             print var
             print("the value of var is negative")

         # If the suite of an if clause consists only of a single line, it may go on the same
         if ( var  == -1 ) : print "the value of var is negative"
```

```
-1
the value of var is negative
the value of var is negative
```

```
In [ ]: var = 1

        if var < 0:
            print "the value of var is negative"
            print var
        else:
            print "the value of var is positive"
            print var
```

```
In [ ]: score = 95

        if score >= 99:
            print('A')
        elif score >=75:
            print('B')
        elif score >= 60:
            print('C')
        elif score >= 35:
            print('D')
        else:
            print('F')
```

```
In [18]: # First Example
         print "First Example"
         for item in [1,2,3,4,5]:
             print 'item :', item

         # Second Example
         print "Second Example"
         letters = ['A', 'B', 'C']
         for letter in letters:
             print (' First loop letter :', letter)

         # Third Example - Iterating by sequency index
         print "Third Example"
         for index in range(len(letters)):
             print 'First loop letter :', letters[index]

         # Fourth Example - Using else statement
         print "Fourth Example"
         for item in [1,2,3,4,5]:
             print 'item :', item
         else:
             print 'looping over item complete!'
```

```
First Example
item : 1
item : 2
item : 3
item : 4
item : 5
Second Example
Current letter : A
Current letter : B
Current letter : C
Third Example
Current letter : A
Current letter : B
Current letter : C
Fourth Example
item : 1
item : 2
item : 3
item : 4
item : 5
```

```
looping over item complete!
```

```
In [12]: count = 0
         while (count < 5):
             print 'The count is:', count
             count = count + 1
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
```

```
In [13]: count = 0
         while count < 5:
             print count, " is  less than 5"
             count = count + 1
         else:
             print count, " is not less than 5"
```

```
0  is  less than 5
1  is  less than 5
2  is  less than 5
3  is  less than 5
4  is  less than 5
5  is not less than 5
```

0.1.1 Lists

```
In [85]: list_1 = ['Statistics', 'Programming', 2016, 2017, 2018];
         list_2 = ['a', 'b', 1, 2, 3, 4, 5, 6, 7 ];

         # Accessing values in lists
         print "list_1[0]: ", list_1[0]
         print "list2_[1:5]: ", list_2[1:5]

         # Adding new value to list
         list_1.append(2019)
         print list_1

         # Updating existing values of list
         print "Value available at index 2 : ", list_1[2]
         list_1[2] = 2015;
         print "New value available at index 2 : ", list_1[2]

         # Deleting list elements
```

```
        del list_1[5];
        print "After deleting value at index 2 : ", list_1

        # Basic Operations
        print "Length: ", len(list_1)
        print "Concatenation: ", [1,2,3] + [4, 5, 6]
        print "Repetition :", ['Hello'] * 4
        print "Membership :", 3 in [1,2,3]
        print "Iteration :"
        for x in [1,2,3]: print x

        # Negative sign will count from the right
        print "slicing :", list_1[-2]
        # If you dont specify the end explicitly, all elements from the specified start index
        print "slicing range: ", list_1[1:]

        # Comparing elements of lists
        print "Compare two lists: ", cmp([1,2,3, 4], [1,2,3])
        print "Max of list: ", max([1,2,3,4,5])
        print "Min of list: ", min([1,2,3,4,5])
        print "Count number of 1 in list: ", [1,1,2,3,4,5,].count(1)
        list_1.extend(list_2)
        print "Extended :", list_1
        print "Index for Programming : ", list_1.index( 'Programming')
        print list_1
        print "pop last item in list: ", list_1.pop()
        print "pop the item with index 2: ", list_1.pop(2)
        list_1.remove('b')
        print "removed b from list: ", list_1
        list_1.reverse()
        print "Reverse: ", list_1
        list_1 = ['a', 'b','c',1,2,3]
        list_1.sort()
        print "Sort ascending: ", list_1
        list_1.sort(reverse = True)
        print "Sort descending: ", list_1
```

```
list_1[0]:  Statistics
list2_[1:5]:  ['b', 1, 2, 3]
['Statistics', 'Programming', 2016, 2017, 2018, 2019]
Value available at index 2 :  2016
New value available at index 2 :  2015
After deleting value at index 2 :  ['Statistics', 'Programming', 2015, 2017, 2018]
Length:  5
Concatenation:  [1, 2, 3, 4, 5, 6]
Repetition : ['Hello', 'Hello', 'Hello', 'Hello']
Membership : True
Iteration :
```

```
1
2
3
slicing : 2017
slicing range:  ['Programming', 2015, 2017, 2018]
Compare two lists:  1
Max of list:  5
Min of list:  1
Count number of 1 in list:  2
Extended : ['Statistics', 'Programming', 2015, 2017, 2018, 'a', 'b', 1, 2, 3, 4, 5, 6, 7]
Index for Programming :  1
['Statistics', 'Programming', 2015, 2017, 2018, 'a', 'b', 1, 2, 3, 4, 5, 6, 7]
pop last item in list:  7
pop the item with index 2:  2015
removed b from list:  ['Statistics', 'Programming', 2017, 2018, 'a', 1, 2, 3, 4, 5, 6]
Reverse:  [6, 5, 4, 3, 2, 1, 'a', 2018, 2017, 'Programming', 'Statistics']
Sort ascending:  [1, 2, 3, 'a', 'b', 'c']
Sort descending:  ['c', 'b', 'a', 3, 2, 1]
```

0.1.2 Tuple

```
In [91]: tuple = ('a','b','c','d',1,2,3)
         print tuple
         tuple = ()
         print tuple
         tuple = (1,)
         print tuple[0]
```

```
('a', 'b', 'c', 'd', 1, 2, 3)
()
1
```

```
In [2]: dict1 = {'Name': 'Jivin', 'Age': 6};
        dict2 = {'Name': 'Pratham', 'Age': 7};
        dict3 = {'Name': 'Pranuth', 'Age': 7};
        dict4 = {'Name': 'Jivin', 'Age': 6};
        print "Return Value : %d" %  cmp (dict1, dict2)
        print "Return Value : %d" %  cmp (dict2, dict3)
        print "Return Value : %d" %  cmp (dict1, dict4)
```

```
Return Value : -1
Return Value : 1
Return Value : 0
```

```
In [5]: # simple function to add two numbers
        def sum_two_numbers(a, b):
            return a + b

        # after this line x will hold the value 3!
        x = sum_two_numbers(1,2)
        print x
```

```
3
```

```
In [9]: def sum_two_numbers(a, b = 10):
            return a + b

        print sum_two_numbers(10)
        print sum_two_numbers(10, 5)
```

```
20
15
```

```
In [10]: def foo(*args):
             for a in args:
                 print a

         foo(1,2,3)
```

```
1
2
3
```

```
In [11]: def bar(**kwargs):
             for a in kwargs:
                 print a, kwargs[a]

         bar(name='one', age=27)
```

```
age 27
name one
```

```
In [13]: import os

         content = dir(os)

         print content
```

```
['F_OK', 'O_APPEND', 'O_BINARY', 'O_CREAT', 'O_EXCL', 'O_NOINHERIT', 'O_RANDOM', 'O_RDONLY', '
```

Reference: mastering machine learning with python in six-steps