01_Basics1

April 11, 2020

1 1. The Zen Of Python

[1]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

2 2. Variables

A name that is used to denote something or a value is called a variable. In python, variables can be declared and values can be assigned to it as follows,

[2]:
```
x = 2
y = 5
xy = 'Hey'
```

[3]:
```
print (x+y, xy)
```

```
7 Hey
```

[4]:
```
type(x)
```

[4]:
```
int
```

[5]:
```
type(xy)
```

[5]:
```
str
```

Multiple variables can be assigned with the same value.

[6]:
```
x = y = 1
```

[7]:
```
print (x,y)
```

```
1 1
```

3 3. Operators

3.1 3.1 Arithmetic Operators

Symbol	Task Performed
+	Addition
-	Subtraction
/	division
%	mod
	multiplication
//	floor division
	to the power of

[8]:
```
1+2
```

[8]:
```
3
```

[9]:
```
2-1
```

[9]:
```
1
```

[10]:
```
1*2
```

[10]:
```
2
```

[11]:
```
1/2
```

[11]:
```
0.5
```

[12]:
```
1/2.0
```

[12]: 0.5

[13]:
```
15%10
```

[13]: 5

[14]:
```
3%2
```

[14]: 1

Floor division is nothing but converting the result so obtained to the nearest integer.

[15]:
```
2.8//2.0
```

[15]: 1.0

[16]:
```
3//1.7
```

[16]: 1.0

[17]:
```
3.8//2
```

[17]: 1.0

[18]:
```
2**2
```

[18]: 4

3.2 3.2 Relational Operators

Symbol	Task Performed
==	True, if left operand is equal to right
!=	True, if left operand is not equal to the right
<	True, if left operand is less than right
>	True, if left operand is greater than right
<=	True, if left operand is less than or equal to right
>=	True, if left operand is greater than or equal to right

[19]:
```
3 > 2
```

[19]: True

[20]:
```
3 < 2
```

[20]: False

[21]:
```
z = 1
```

[22]:
```
z == 1
```

[22]:
```
True
```

[23]:
```
z > 1
```

[23]:
```
False
```

3.3 3.3 Logical Operators

Symbol	Task Performed
and	Returns x if x is True, y otherwise
or	Returns y if x is True, x otherwise
not	Returns True if x is True, False otherwise

[24]:
```
x = True
y = False
```

[25]:
```
x and y
```

[25]:
```
False
```

[26]:
```
x or y
```

[26]:
```
True
```

[27]:
```
not x
```

[27]:
```
False
```

3.4 3.4 Bitwise Operators

Symbol	Task Performed
&	Bitwise AND
l	Bitwise OR
^	XOR
>>	Right shift
<<	Left shift

[28]:
```
a = 2 #10
b = 3 #11
```

[29]:
```
print(a & b)
print(bin(a & b))
```

```
2
0b10
```

[30]:
```
a|b
```

[30]: 3

[31]:
```
a^b
```

[31]: 1

[32]:
```
5 >> 1
```

[32]: 2

0000 0101 -> 5

Shifting the digits by 1 to the right and zero padding

0000 0010 -> 2

[33]:
```
5 << 1
```

[33]: 10

0000 0101 -> 5

Shifting the digits by 1 to the left and zero padding

0000 1010 -> 10

3.5 3.5 Identity Operators

Symbol	Task Performed
is	True if the operands are identical
is not	True if the operands are not identical

[34]:
```
x = 5
```

[35]:
```
x is 5
```

[35]: True

[36]:
```
x is not 5
```

[36]: False

3.6 3.6 Membership Operators

Symbol	Task Performed
in	True if it finds element in specified sequence
not in	True if it doesn't finds element in specified sequence

[37]:
```
m = [1,2,3,4,5]
```

[38]:
```
3 in m
```

[38]:
```
True
```

[39]:
```
3 not in m
```

[39]:
```
False
```

4 4 Built-in Functions

Python comes loaded with pre-built functions

4.1 Conversion from one system to another

Conversion from hexadecimal to decimal is done by adding prefix **0x** to the hexadecimal value or vice versa by using built in **hex( )**, Octal to decimal by adding prefix **0** to the octal value or vice versa by using built in function **oct( )**.

[40]:
```
hex(170)
```

[40]:
```
'0xaa'
```

[41]:
```
0xAA
```

[41]:
```
170
```

[42]:
```
oct(9)
```

[42]:
```
'0o11'
```

[43]:
```
print (0o11)
```

```
9
```

[44]:
```
bin(5)
```

[44]:
```
'0b101'
```

int( ) accepts two values when used for conversion, **one is the value in a different number system** and **the other is its base**. Note that input number in the different number system should be of string type.

[45]:
```
print (int('010',8))
print (int('0xaa',16))
print (int('1010',2))
```

```
8
170
10
```

int( ) can also be used to get only the integer value of a float number or can be used to convert a number which is of type string to integer format. Similarly, the function **str( )** can be used to convert the integer back to string format

[46]:
```
print (int(7.7))
print (str(7))
```

```
7
7
```

Also note that function **bin( )** is used for binary and **float( )** for decimal/float values. **chr( )** is used for converting ASCII to its alphabet equivalent, **ord( )** is used for the other way round.

[47]:
```
chr(97) # converting  ASCII to alphabet
```

[47]:
```
'a'
```

[48]:
```
ord('A')
```

[48]:
```
65
```

4.2 Simplifying Arithmetic Operations

round( ) function rounds the input value to a specified number of places or to the nearest integer.

[49]:
```
print (round(5.6231))
print (round(4.55892, 2))
```

```
6
4.56
```

[50]:
```
print (round(5.5292,3))
```

```
5.529
```

complex( ) is used to define a complex number and **abs( )** outputs the absolute value of the same.

[51]:
```
c =complex('3+2j')
print (abs(c))
```

```
3.605551275463989
```

divmod(x,y) outputs the quotient and the remainder in a tuple in the format (quotient, remainder).

[52]:
```
divmod(9,2)
```

[52]:
```
(4, 1)
```

isinstance( ) returns True, if the first argument is an instance of that class. Multiple classes can also be checked at once.

[53]:
```
print (isinstance(1, int))
print (isinstance(1.0,int))
print (isinstance(1.0,(int,float)))
```

```
True
False
True
```

[54]:
```
print (isinstance('hello',str))
```

```
True
```

[55]:
```
isinstance?
```

[56]:
```
help(isinstance)
```

```
Help on built-in function isinstance in module builtins:

isinstance(obj, class_or_tuple, /)
    Return whether an object is an instance of a class or of a subclass thereof.

    A tuple, as in ``isinstance(x, (A, B, …))``, may be given as the target to
    check against. This is equivalent to ``isinstance(x, A) or isinstance(x, B)
    or …`` etc.
```

pow(x,y,z) can be used to find the power x^y also the mod of the resulting value with the third specified number can be found i.e. : (x^y % z).

[57]:
```
print (pow(3,3))
print (pow(3,3,5))
```

```
27
2
```

range( ) function outputs the integers of the specified range. It can also be used to generate a series by specifying the difference between the two numbers within a particular range. The elements are returned in a list (will be discussing in detail later.)

[58]:
```
print (list(range(3)))
print (list(range(2,9)))
print (list(range(2,30,8)))
```

```
[0, 1, 2]
[2, 3, 4, 5, 6, 7, 8]
[2, 10, 18, 26]
```

4.3 Accepting User Inputs

input( ) accepts input and stores it as a string. Hence, if the user inputs a integer, the code should convert the string to an integer and then proceed.

[59]:
```
abc = input("Type something here:")
```

```
Type something here:Kumar
```

[60]:
```
type(abc)
```

[60]:
```
str
```

[61]:
```
print(abc)
```

```
Kumar
```

[62]:
```
abc1 =  int(input("Type integer here: \t"))
```

```
Type integer here:      25
```

[63]:
```
type(abc1)
```

[63]:
```
int
```

[64]:
```
print(abc1)
```

```
25
```

[ ]:
```

```