Basics

January 12, 2021

1 Module 0 - Python Basics

A PDF version of this notebook is available in Module 0 - Python Basics

1.1 Identifier Names

are references to memory locations. Identifier names are case-sensitive, e.g., my_var is different than my_Var. Identifier names follow rules:

- must start with underscores (_) or letters (a - z A - Z), followed by any number of underscores (_), letters (a - z A - Z), or digits (0 - 9), e.g.,
  1. `var`
  2. `my_var`
  3. `__index__`
- cannot be any reserved word (None, True, False, and, or, not, if, else, elif, for, while, break, continue, pass, def, lambda, global, nonlocal, return, del, in, is, assert, class, try, except, finally, raise, import, from, with, as)

1.2 Variables

A variable is created the moment you first assign a value to an identifier name. e.g.,

[1]:

```
x = 5
y = "John"
print(x)
print(y)
```

```
5
John
```

The identifier name `x` is assigned the value 5, which is an integer. For Python `x` is only an alias to a memory location in the computer.

1.3 Variable Types

The values assigned to the identifier names can be of different types. Below are the description of the most common:

1.3.1 Integers

Integers are objects - instances of the `int` class representing integer values.

```
[2]: print(type(100))
```

```
<class 'int'>
```

For example, the number 100 is an integer:

```
[3]: type(100)
```

```
[3]: int
```

```
[4]: x = 100
     type(x)
```

```
[4]: int
```

Integer Constructor To construct or transform a variable to `int`, we can use the constructor function `int()`.

```
[5]: int(10)      ## converts to integer
```

```
[5]: 10
```

```
[6]: int(10.9)    ## applies the floor function
```

```
[6]: 10
```

```
[7]: int("10")
```

```
[7]: 10
```

```
[8]: int("101", base=2)   ## base 2
```

```
[8]: 5
```

1.3.2 Floats

The float class can be used to represent real numbers.

```
[9]: float(10)
```

```
[9]: 10.0
```

```
[10]: float(3.14)
```

```
[10]: 3.14
```

[11]:

```python
from fractions import Fraction
float(Fraction('22/7'))
```

[11]:

```
3.142857142857143
```

[12]:

```python
format(0.125, '.25f')  ## you can format the float to show different digits
```

[12]:

```
'0.1250000000000000000000000'
```

equality Because not all real numbers have an exact float representation, equality testing can be tricky.

[13]:

```python
x = 0.1 + 0.1 + 0.1
y = 0.3
x == y
```

[13]:

```
False
```

[14]:

```python
print('0.1 --> {0:.25f}'.format(0.1))
print('x --> {0:.25f}'.format(x))
print('y --> {0:.25f}'.format(y))
```

```
0.1 --> 0.1000000000000000055511151
x --> 0.3000000000000000444089210
y --> 0.2999999999999999888977698
```

1.3.3 Booleans

All objects in Python have an associated truth value. The `bool()` function gives back the associated "truthyness" value of an object

[15]:

```python
bool(1), bool(-1), bool(100)
```

[15]:

```
(True, True, True)
```

[16]:

```python
bool(0)
```

[16]:

```
False
```

[17]:

```python
bool(100) ## this is the same as (100).__bool__()
```

[17]:

```
True
```

[18]:

```python
## Any zero value is false
from fractions import Fraction
from decimal import Decimal
bool(0), bool(0.0), bool(Fraction(0,1)), bool(Decimal('0')), bool(0j)
```

```
[18]: (False, False, False, False, False)
```

```
[19]: ## Any empty object is false
      bool([]), bool(()), bool('')
```

```
[19]: (False, False, False)
```

```
[20]: ## The None object is false
      bool(None)
```

```
[20]: False
```

Booleans within if statements

```
[21]: a = ''
      if a:
          print(a[0])
      else:
          print('a is None, or a is empty')
```

```
a is None, or a is empty
```

```
[22]: b = None
      if b:
        print("This is printed")    ## nothing gets printed because the if statement␣
       ↪evaluates to False
```

Comparison Operators The == and != operators are value comparison operators.

```
[23]: bool(2) == True
```

```
[23]: True
```

```
[24]: a = 10
      b = 10
      a == b
```

```
[24]: True
```

```
[25]: 6 != False
```

```
[25]: True
```

Ordering Comparisons

```
[26]: 3 < 4
```

```
[26]: True
```

```
[27]: 5 <= 5
```

```
[27]: True
```

```
[28]: 3 >= 2
```

```
[28]: True
```

```
[29]: 1 < 2 < 3 ## means 1 < 2 and 2 < 3
```

```
[29]: True
```

The `in` and `not in` operators are used with iterables and test membership

```
[30]: 1 in [1, 2, 3]
```

```
[30]: True
```

```
[31]: [1, 2] in [1, 2, 3]
```

```
[31]: False
```

The `is` and `is not` operators are used to check memory addresses, so avoid using if you are comparing variable values.

```
[32]: a = 10000
      b = 10000
      a is b
```

```
[32]: False
```

```
[33]: id(a) ## shows memory address
```

```
[33]: 139746823320912
```

```
[34]: id(b)
```

```
[34]: 139746823320592
```