01-intro

September 16, 2022

1 Python as a calculator

Python can be used as a calculator

[1]:
```
1 + 1
```

[1]:
```
2
```

Being the cast of the values done automatically

[2]:
```
1 + 1.0
```

[2]:
```
2.0
```

[3]:
```
'1' + '1'
```

[3]:
```
'11'
```

[4]:
```
1 + int('1')
```

[4]:
```
2
```

[5]:
```
1 + float('1')
```

[5]:
```
2.0
```

[6]:
```
1 + int(1.3)
```

[6]:
```
2
```

The type of an object can be checked using the `type`methods

[7]:
```
type(1.0)
```

[7]:
```
float
```

[8]:
```
type(1)
```

[8]:
```
int
```

But, sometimes, Python doest't know how it should cast the values (make it int or string!?)

[9]:
```
1 + "1"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [9], line 1
----> 1 1 + "1"

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

2 Long Lines

Long lines can be breaked by a '' - and no following chars

[10]:
```
A = 7 * 3 + \
        5 / 2
```

[11]:
```
A
```

[11]:
```
23.5
```

The next one will rase an error

[12]:
```
D = 1 + 1
*3
```

```
  Cell In [12], line 2
    *3
    ^

SyntaxError: can't use starred expression here
```

Lines can also be braked in other conditions, such as: list, function arguments, etc.

[13]:
```
my_list = ['a', 'b', 'c',       # list (breaked by a ',')
        'd' , 'e']
c = range(1,            # call to a method (breaked by a ',')
        11)
```

3 Everything is an object.

- Everything in Python is an object.
- A first fundamental distinction that Python makes on data is about whether or not the value of an object changes. If the value can change, the object is called **mutable**, while if the value cannot change, the object is called **immutable**.

[14]:
```
age = 42
age = 43
```

The 2nd line hasn't changed the value of age but what it is referencing.

First line: age is a name that is set to point to an int object, whose value is 42. When we type age = 43 another object is created, of the type int and value 43, and the name age is set to point to it.

So, we actually just pointed age to a different location. Let us check it by looking at the `id` of the objects

[15]:
```
age = 42
id(age)
```

[15]: 1533673107024

[16]:
```
age = 43
id(age)
```

[16]: 1533673107056

[17]:
```
age = 42
id(age)
```

[17]: 1533673107024

Interesting, is that, Python is optimized such that the creation of equal objects is avoided

[18]:
```
x = 42
id(x)
```

[18]: 1533673107024

Another example, now with lists

[19]:
```
A = [1, 2]
id(A)
```

[19]: 1533866179200

[20]:
```
 B = A
```

[21]:
```
id(B)
```

[21]: 1533866179200

If the object pointed by A is changed

[22]:
```
A[0] = 10
```

[23]:
```
A
```

[23]: [10, 2]

what would we espect to happen when we look at B?

[24]:
```
B
```

[24]: [10, 2]

to make a hard copy

[25]:
```
C = A[:]
id(C)
```

[25]: 1533866390464

[26]:
```
A[1]=111
C
```

[26]: [10, 2]

The following will change not the object which is “pointed” by A but what A “points” to

[27]:
```
A = [1, 3]
```

[28]:
```
id(A)
```

[28]: 1533863061312

In short,

[29]:
```
A
```

[29]: [1, 3]

[30]:
```
B
```

[30]: [10, 111]

[31]:
```
C
```

[31]: [10, 2]