Section 3 Object: Mutable/Immutable, Attributes/Methods

In Python:

- An object's **identity** never changes once it has been created;
- Whether its **value** can change or not really depends:
  - If the value can be changed, the object is called *mutable* -- it is more flexible!
  - If the value cannot be changed, the object is called *immutable* -- it is safer!

For beginners, mutable/immutable objects can easily lead to errors that are very difficult to debug.

- Whether the object is mutable or not? It depends on its **type**:
  - (for built-in types) **List**, Dictionary and Set are mutable;
  - Int, Float, String, Bool, Tuple ... are immutable.
  - Numpy array is also mutable (will talk about it later)

In [1]:
```
a = [1,2,3]
print(id(a))
a[0]=0
print(a)
print(id(a))
```

```
140502230226816
[0, 2, 3]
140502230226816
```

Compare it with the following two examples:

In [2]:
```
a = 1
print(id(a))
a = 0
print(id(a))
```

```
4540867712
4540867680
```

In [3]:
```
a = [1,2,3]
print(id(a))
a = [0,2,3]
print(a)
print(id(a))
```

```
140502230050288
[0, 2, 3]
140502229811600
```

Now it's time to test your understandings. Recall our examples in Section 1 and solve it yourself!

In [4]:
```
a = 1000
b = a
b = 1
print(a)
```

```
1000
```

In [5]:
```
a = [1000,1]
b = a
```

```
b = [1,1]
print(a)
```

```
[1000, 1]
```

In [6]:
```
a = [1000,1]
b = a
b[0] = 1
print(a)
```

```
[1, 1]
```

Indeed, what is the solution if we really want to "copy" a list?

There are multiple solutions to this, and we will mention one here using the copy *method*.

In [35]:
```
a = [1,2,3]
b = a.copy()
a[0] = 0
print(a)
print(b)
```

```
[0, 2, 3]
[1, 2, 3]
```

Misc: Some notes about Operator and List Indexing

- Operators you might not be familiar with

In [8]:
```
print(10%3) # Modulo
print(10**3) # Exponential, it is different with a^b in Matlab
```

```
1
1000
```

- Operators might also have unexpected meanings

In [9]:
```
print('python'+'math')# concatenation of strings
```

```
pythonmath
```

In [10]:
```
[1,2,3]+['python','math'] # concatenation of lists
```

Out[10]:
```
[1, 2, 3, 'python', 'math']
```

- Something special about Division operators in Python 3 (Things were very different in Python 2, and throughout this course we're going to use Python 3)

In [11]:
```
var = 9//4 ## integer division (or floor division)
print(var)
type(var)
```

```
2
```

Out[11]:
```
int
```

In [12]:
```
var = 9.0//4
print(var)
type(var)
```

```
2.0
```

Out[12]:
```
float
```

In [13]:
```
var = 12/4 ## true division (or float division), always return the type of float even for integers!
print(var)
type(var)
```

```
3.0
```

Out[13]:
```
float
```

- In fact, indexing is also considered as the operator in Python. A very good reference

In [14]:
```
mylist = [1,2,3]
print(mylist[0]) # always remember that index starts from 0
print(mylist[1])
print(mylist[2])
```

```
1
2
3
```

In [15]:
```
print(mylist[-1]) # minus index
print(mylist[-2])
print(mylist[-3])
```

```
3
2
1
```

- Slicing: a basic rule is that $[start:stop]$ means $start \leq i < stop$, where i is the index of list, starts from zero.

If there is no step, my strategy is that I will first find the start element, and then count $length = stop - start$ elements.

In [16]:
```
mylist = list(range(1,9)) # range(start,stop) can be understood in the same way.
print(mylist)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

In [17]:
```
print(mylist[2:5])
```

```
[3, 4, 5]
```

- A more complete form of slicing is $[start:stop:step]$, and when parameters are omitted, you just plug in the default value.

In [18]:
```
print(mylist[4:2:-1])
print(mylist[-5::])
print(mylist[:-3:-1])
print(mylist[::2])
```

```
[5, 4]
[4, 5, 6, 7, 8]
[8, 7]
[1, 3, 5, 7]
```

Attributes and Methods of Python Object

Roughly speaking,

- attributes are the variables stored within object;
- methods are the functions stored within object.

String attributes/methods

In [19]:
```
text = "Data Science"
text.__doc__
```

Out[19]:
```
"str(object='') -> str\nstr(bytes_or_buffer[, encoding[, errors]]) -> str\n\nCreate a new string ob
ject from the given object. If encoding or\nerrors is specified, then the object must expose a data
buffer\nthat will be decoded using the given encoding and error handler.\nOtherwise, returns the re
sult of object.__str__() (if defined)\nor repr(object).\nencoding defaults to sys.getdefaultencodin
g().\nerrors defaults to 'strict'."
```

In [20]:
```
text.upper() # return a new string object with upper case
```

Out[20]:
```
'DATA SCIENCE'
```

In [21]:
```
text # See? the original text is not affected
```

Out[21]:
```
'Data Science'
```

In [22]:
```
text.lower() # return a new string object
```

Out[22]:
```
'data science'
```

In [23]:
```
text.capitalize() # return a new string object
```

Out[23]:
```
'Data science'
```

Lists attributes/methods

In [24]:
```
numbers = [1, 4, 0, 2, 9, 9, 10]
numbers.__class__
```

Out[24]:
```
list
```

In [25]:
```
print(numbers)
print(id(numbers))
numbers.reverse() # does NOT return a new LIST object! just modify the original list -- remember th
print(numbers) # [10, 9, 9, 2, 0, 4, 1]
print(id(numbers))
```

```
[1, 4, 0, 2, 9, 9, 10]
140502229813120
[10, 9, 9, 2, 0, 4, 1]
140502229813120
```

It is INCORRECT to write in this way:

In [26]:
```
numbers_reverse = numbers.reverse() # it is the INCORRECT way to reverse a list!!!
print(numbers_reverse)
numbers_reverse = numbers
```

```
None
```

Some list methods not only return the value, but also modify the list in-place (i.e. won't change identity of the list). The pop() method is a very typical example.

In [36]:
```
print(numbers)
print(id(numbers))
element_pop = numbers.pop(4) # the input is index to delete in the list
print(element_pop)
print(numbers)
print(id(numbers))
```

```
[0, 1, 2, 4, 9, 10]
140502229813120
9
[0, 1, 2, 4, 10]
140502229813120
```

In [37]:
```
numbers.sort() # sort the list in ascending order
print(numbers)
print(id(numbers))
```

```
[0, 1, 2, 4, 10]
140502229813120
```

What about descending order? Using the `help` function to check yourself!

In [29]:
```
help(numbers.sort)
```

```
Help on built-in function sort:

sort(*, key=None, reverse=False) method of builtins.list instance
    Stable sort *IN PLACE*.
```

In [30]:
```
help(numbers.pop)
```

```
Help on built-in function pop:

pop(index=-1, /) method of builtins.list instance
    Remove and return item at index (default last).

    Raises IndexError if list is empty or index is out of range.
```

Compared to the built-in list, the Numpy array has more flexible operations such as boolean filters (will talk about it in later lectures).

Using dir() to show all valid attributes.

In [31]:
```
dir(text)
```

Out[31]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__gt__',
```

```
'__hash__',
'__init__',
'__init_subclass__',
'__iter__',
'__le__',
'__len__',
'__lt__',
'__mod__',
'__mul__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__rmod__',
'__rmul__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'capitalize',
'casefold',
'center',
'count',
'encode',
'endswith',
'expandtabs',
'find',
'format',
'format_map',
'index',
'isalnum',
'isalpha',
'isascii',
'isdecimal',
'isdigit',
'isidentifier',
'islower',
'isnumeric',
'isprintable',
'isspace',
'istitle',
'isupper',
'join',
'ljust',
'lower',
'lstrip',
'maketrans',
'partition',
'replace',
'rfind',
'rindex',
'rjust',
'rpartition',
'rsplit',
'rstrip',
'split',
'splitlines',
'startswith',
'strip',
'swapcase',
'title',
'translate',
'upper',
'zfill']
```

Names with dunder (double underscores __) are special attributes/methods.

In [32]:

```
help(text.replace)
```

```
Help on built-in function replace:
```

```
replace(old, new, count=-1, /) method of builtins.str instance
    Return a copy with all occurrences of substring old replaced by new.

      count
        Maximum number of occurrences to replace.
        -1 (the default value) means replace all occurrences.

    If the optional argument count is given, only the first count occurrences are
    replaced.
```

In [33]:

```
dir(str) # str is the built-in string type
```

Out[33]:

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmod__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'capitalize',
 'casefold',
 'center',
 'count',
 'encode',
 'endswith',
 'expandtabs',
 'find',
 'format',
 'format_map',
 'index',
 'isalnum',
 'isalpha',
 'isascii',
 'isdecimal',
 'isdigit',
 'isidentifier',
 'islower',
 'isnumeric',
 'isprintable',
 'isspace',
 'istitle',
 'isupper',
 'join',
```

```
'ljust',
'lower',
'lstrip',
'maketrans',
'partition',
'replace',
'rfind',
'rindex',
'rjust',
'rpartition',
'rsplit',
'rstrip',
'split',
'splitlines',
'startswith',
'strip',
'swapcase',
'title',
'translate',
'upper',
'zfill']
```

In [34]:

```
dir(numbers)
dir(list)
```

Out[34]:

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

In [38]:
```
help(numbers.append)
```

```
Help on built-in function append:

append(object, /) method of builtins.list instance
    Append object to the end of the list.
```

In [39]:
```
numbers
```

Out[39]:
```
[0, 1, 2, 4, 10]
```

In [40]:
```
numbers.append(11)
```

In [41]:
```
numbers
```

Out[41]:
```
[0, 1, 2, 4, 10, 11]
```