Lecture 4 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. (https://florimond.dev/blog/articles/2018/08/python-mutable-defaults-are-the-source-of-all-evil/)

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

Compare it with the following two examples:

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

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

Now it's time to test your understandings. Recall our examples in Lecture 2 and solve it yourself!

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

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

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

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

There are multiple solutions to this (https://www.geeksforgeeks.org/python-cloning-copying-list/), and we will mention one here using the copy *method*.

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

Misc: Some notes about Operator and List Indexing

- Operators you might not be familiar with

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

- Operators might also have unexpected meanings

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

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

- Something special about Division operators in Python 3 (https://www.python.org/dev/peps/pep-0238/) (Things were very different in Python 2, and throughout this course we're going to use Python 3)

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

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

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

- In fact, indexing is also considered as the operator in Python. A very good reference (https://railsware.com/blog/python-for-machine-learning-indexing-and-slicing-for-lists-tuples-strings-and-other-sequential-types/)

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

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

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

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

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

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

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

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

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

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

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

Lists attributes/methods

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

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

It is INCORRECT to write in this way:

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

Some list methods not only return the value, but also modify the list in-place. The pop() method is a very typical example.

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

In [ ]:
```
numbers.sort()
print(numbers)
print(id(numbers))
```

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

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

In [ ]:
```
dir(str)
```

In [ ]:
```
dir(numbers)
dir(list)
```