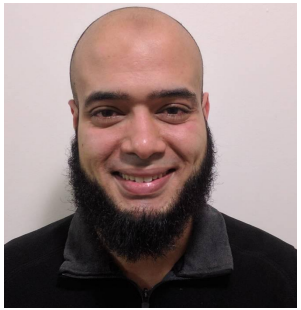
Python Programming

Class vs Instance namespace

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Last session: Confusion is coming!

- Static variables are nice as long as you used them carefully
- As long as you use the Class to access/modify the static var ⇒ Perfect
- Once you use the object to modify the static var issues may occur
  - We need to understand instance namespace vs class namespace
  - We need to take into consideration: mutable vs immutable objects
- Similar issue if you have an attribute with same name as static var!
- Before next session
  - Practice what we learned
  - Take 5-10 min to guess the behaviour of the next 2 slides
    - No need to play with code or Google

Namespace

- A namespace is a mapping from names to objects
  - No relation between names in different namespaces
  - Typically implemented using dictionary
- When we define a class blueprint ⇒ we have a class namespace for it
- When we create object1 ⇒ we have an instance (1) namespace
- When we create object2 ⇒ we have an instance (2) namespace
- 3 namespaces with maybe common names, but no relations

Class vs Instance namespace

```python
class Employee:
    """Class Employee is TODO"""
    total_employees = 0
    def __init__(self, name):

        self.name = name
        Employee.total_employees += 1

    def print(self):
        pass

    @classmethod
    def our_f(cls):
        pass
```

```python
if __name__ == '__main__':
    obj = Employee('Mostafa')
    print(obj.__dict__)
    # {'name': 'Mostafa'}

    print(Employee.__dict__)
    # '__doc__': 'Class Employee is TODO',
    # 'total_employees': 1,
    # '__init__': <function Employee.__init__>,
    # 'print': <function Employee.print>,
    # 'our_f': <classmethod object>
```

Class vs Instance namespace

```
obj1 = Employee('obj1')
obj2 = Employee('obj2')

print(Employee.total_employees) # 2
print(obj1.total_employees)    # 2
print(obj2.total_employees)    # 2
```

Employee Namespace
total_employees ⇒ 2 (0x222)

Obj1 Namespace
name ⇒ ‘Obj1’ (0x888)

Obj2 Namespace
name ⇒ ‘Obj2’ (0x999)

- So far, just accessing with no modification
- obj1.total_employees
  - Is it in the instance namespace?
  - No, is it in class namespace? Yes, use it

Class vs Instance namespace

```
obj1 = Employee('obj1')
obj2 = Employee('obj2')

print(Employee.total_employees) # 2
print(obj1.total_employees)    # 2
print(obj2.total_employees)    # 2

obj1.total_employees = 7
print(Employee.total_employees)  # 2
print(obj1.total_employees)      # 7
print(obj2.total_employees)      # 2
```

Employee Namespace
total_employees ⇒ 2 (0x222)

Obj1 Namespace
name ⇒ ‘Obj1’ (0x888)
total_employees ⇒ 7 (0x777)

Obj2 Namespace
name ⇒ ‘Obj2’ (0x999)

- With assigning, obj1.total_employees is bounded to its own attribute
- obj1.total_employees
  - Is it in the instance namespace? Yes, use it

Class vs Instance namespace

```
obj1 = Employee('obj1')
obj2 = Employee('obj2')

print(Employee.total_employees) # 2
print(obj1.total_employees)    # 2
print(obj2.total_employees)    # 2

obj1.total_employees = 7
print(Employee.total_employees)  # 2
print(obj1.total_employees)      # 7
print(obj2.total_employees)      # 2

obj2.total_employees += 3
print(Employee.total_employees)  # 2
print(obj1.total_employees)      # 7
print(obj2.total_employees)      # 5
```

Employee Namespace
total_employees ⇒ 2 (0x222)

Obj1 Namespace
name ⇒ ‘Obj1’ (0x888)
total_employees ⇒ 7 (0x777)

Obj2 Namespace
name ⇒ ‘Obj2’ (0x999)
total_employees ⇒ 5 (0x555)

- Similarly, with +=, obj2 has its own attribute

Class vs Instance namespace

```
obj1 = Employee('obj1')
obj2 = Employee('obj2')

print(Employee.total_employees) # 2
print(obj1.total_employees)    # 2
print(obj2.total_employees)    # 2

obj1.total_employees = 7
print(Employee.total_employees)  # 2
print(obj1.total_employees)      # 7
print(obj2.total_employees)      # 2

obj2.total_employees += 3
print(Employee.total_employees)  # 2
print(obj1.total_employees)      # 7
print(obj2.total_employees)      # 5

del obj1.total_employees
print(Employee.total_employees)  # 2
print(obj1.total_employees)      # 2
print(obj2.total_employees)      # 5
```

Employee Namespace
total_employees ⇒ 2 (0x222)

Obj1 Namespace
name ⇒ ‘Obj1’ (0x888)

Obj2 Namespace
name ⇒ ‘Obj2’ (0x999)
total_employees ⇒ 5 (0x555)

- With deletion, the attribute is removed from obj1 namespace
- So again, the search use the class namespace

Deleting attributes and vars

```
8  if __name__ == '__main__':
9      emp1 = Employee('Mostafa')
10     emp2 = Employee('Belal')
11
12     emp1.total_employees = 10          # Re-bind
13     print(emp1.total_employees)        # 10: refers to its attribute
14     del emp1.total_employees
15     print(emp1.total_employees)        # 3 now: I see shared static
16
17     # del emp1.total_employees         # AttributeError
18     del Employee.total_employees
19
20     # print(emp1.total_employees)      # AttributeError
21     # print(emp2.total_employees)      # AttributeError
22     # print(Employee.total_employees)  # AttributeError
```

Mutable static var

```
class Employee:
    lst = [2, 5]    # mutable
    def __init__(self, name):
        self.name = name

if __name__ == '__main__':
    obj1 = Employee('obj1')
    obj2 = Employee('obj2')

    print(Employee.lst) # [2, 5]
    print(obj1.lst)     # [2, 5]
    print(obj2.lst)     # [2, 5]

    obj1.lst = [10, 20]
    print(Employee.lst)  # [2, 5]
    print(obj1.lst)      # [10, 20]
    print(obj2.lst)      # [2, 5]

    obj2.lst += [3]
    print(Employee.lst)  # [2, 5, 3]
    print(obj1.lst)      # 10, 20]
    print(obj2.lst)      # [2, 5, 3]
```

When to use class attributes (static vars)?

- Constants
- Defining default values
- Tracking (e.g. you want to keep list of all employees names)
- Statistics (total object creations, total number of function calls, etc)
- Tips
  - Access/modify the class attributes using the Class name
  - Don’t use instance attributes same as class attributes
  - Avoid mutable data for class attributes, or be so careful
- About __dict__ is a dictionary: key/value
  - You may add/remove attributes to it, and this will affect the actual object

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."