Python Programming

Inheritance with Slots

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)

Slots

- Sometimes we need to create thousands of instances from a specific class
  - Think Geometry Point, Article, Employee, etc
- We know the attributes of a class stored in __dict__
- Dict is strong, but slow and memory consuming on object level!
  - This means set/get the attributes will be: slow
- Slots is another mechanism for handling the attributes
  - Faster in access and consumes less memory
  - But a bit not flexible in extending the object with new attributes
  - But there are some workarounds

The normal way

```
3   class Employee:
4       employees_cnt = 0
5
6       def __init__(self, name, salary):
7           self.name = name
8           self.salary = salary
9
10  emp = Employee('most', 12)
11  print(dir(emp)) # 'employees_cnt', 'name', 'salary'
12  print(emp.name) # most
13
14  print(Employee.__dict__)    # {'employees_cnt': 0, '__doc__': None, etc}
15  print(emp.__dict__)  # {'name': 'mostafa'}
16  print(vars(emp))     # {'name': 'mostafa'}
17
18
```

With slots

- Almost same usage, but observe the inflexibility

```
class Employee:
    employees_cnt = 0
    __slots__ = "name", "salary"    # tuple/iterable

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        #self.age = 1   # u can't

print(Employee.__dict__)    # {'employees_cnt': 0, '__doc__': None, etc}
emp = Employee('most', 12)
print(dir(emp)) # 'employees_cnt': 0, '__slots__': ['name', 'salary']
                # 'name': <member 'name' of 'Employee' objects>, 'salary':
print(emp.name) # most
#print(emp.__dict__)  # AttributeError no attribute '__dict__'
#print(vars(emp))      # TypeError: vars() argument must have __dict__ attribute
del emp.name
```

With inheritance: Way #1

- In child class, you can normally add attributes in the normal way

```
2  class Person:
3      __slots__ = ['name', 'email']
4      def __init__(self, name, email):
5          self.name = name
6          self.email = email
7
8  #Person('mostafa', 'm@g').__dict__ # Error
9
10 class Student(Person):
11     def __init__(self, name, email, gpa):
12         Person.__init__(self, name, email)
13         print(self.__dict__)   # {}
14         self.gpa = gpa
15         # Will use the parents slots + dict by default
16
17 st = Student('mostafa', 'm@g', 3.7)
18 print(st.__dict__)   # {'gpa': 3.7}
```

With inheritance: Way #2

- We can also extend the child class with its attributes using slot
- But u again is restricted

```
class Person:
    __slots__ = ['name', 'email']
    def __init__(self, name, email):
        self.name = name
        self.email = email

class Student(Person):
    __slots__ = ['gpa'] # EXTEND with new attributes
    def __init__(self, name, email, gpa):
        Person.__init__(self, name, email)
        self.gpa = gpa

st = Student('mostafa', 'm@g', 3.7)
#print(st.__dict__)   # Now error!

# Note: Although we can respecify __slots__ as
# __slots__ = ['name', 'email', 'gpa']
# but this hides parent ones! Overall, highly discouraged
# Note: Probably this will be prevented in the future
```

The best of the 2 worlds

- By adding __dict__, you allow it as an attribute, and hence allows for more flexible entries in it

```
class Person:
    __slots__ = ['name', 'email']
    def __init__(self, name, email):
        self.name = name
        self.email = email

class Student(Person):
    __slots__ = ['gpa', '__dict__']
    def __init__(self, name, email, gpa):
        Person.__init__(self, name, email)
        self.gpa = gpa

st = Student('mostafa', 'm@g', 3.7)
st.temp = '111'
print(st.__dict__)  # {'temp': '111'}

# By adding __dict__ as slot
# we can have both slot and dynamic attributes!
```

Finally

- Python 3.3 Key-Sharing Dictionaries
  - New impl for the standard dict
  - shares the **keys** between **multiple dictionaries** and improve memory use
  - Now with thousands objects of same class: we have shared keys
  - Overall: Time & Memory faster
  - Some guys claim that this feature reduce/cancel the need for slots
  - In future if you needed in a critical situation: do timing with/without slots and decide
    - Be careful from implementation changes
- Future readings: link link link

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."