Abstract Base Class

هدف اصلی کلاس های انتزاعی ارائه یک راه استاندارد برای جلوگیری مشکلات موقع ارث بری مانند اینکه برخی متدها حتما باید override شوند و همچنین با استفاده از کلاس های انتزاعی، یک کلاس می تواند از یک کلاس دیگر بدون ارث بری مستقیم، ارث بری کند.

تعریف یک کلاس انتزاعی

یکی از کتابخانه های پایتون، کتابخانه `abc` است که به ما امکان ساختن یک کلاس انتزاعی را می دهد.

پیش از هرچیزی نیاز است تا با کلاس `ABCMeta` آشنا شوید. این کلاس به ما پایه کلاس های انتزاعی را ارائه می دهد و هر کلاس انتزاعی نیاز است تا از `ABCMeta` به عنوان methaclass استفاده کند.

یکی از متدهایی که در `ABCMeta` وجود دارد، `register` است که به ما این امکان را می دهد تا هر کلاس دلخواهی را به عنوان جد هر کلاس دلخواهی معرفی کنیم. برای مثال در کد زیر قصد داریم تا کلاس `Foo` را به عنوان جد کلاس `list` معرفی کنیم.

Copy Python

```python
import abc


class Foo(metaclass=abc.ABCMeta):
    def bar(self):
        return None


Foo.register(list)

li = []
print(isinstance(li, Foo))
```

▼ خروجی چیه؟

```
True
```

آیا می توانیم یک کلاس را به عنوان جد یک کلاس دیگر تعریف کنیم؟ بله

```python
import abc


class Foo(metaclass=abc.ABCMeta):
    pass


class Bar():
    pass


Foo.register(Bar)

bar = Bar()
print(isinstance(bar, Foo))
```

▼ خروجی چیه؟

```
True
```

آیا فقط همین یک راه وجود دارد؟

```python
import abc


class Foo(metaclass=abc.ABCMeta):
    pass


@Foo.register
class Bar():
    pass

```

```
12
13
14
bar = Bar()
print(isinstance(bar, Foo))
```

▼ خروجی چیه؟

```
True
```

_ _ subclasshook _ _

یکی از مجیک متدهایی که داخل `abc` پیاده سازی شده است، `__subclasshook__` نام دارد که نیاز است تا به صورت کلاس متد پیاده سازی شود و یک ورودی اضافه نیز دارد و همچنین به صورت معمول یکی از مقدارهای `True` یا `False` ریترن می کند.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
import abc


class Foo(metaclass=abc.ABCMeta):
    @classmethod
    def __subclasshook__(cls, other):
        print('subclass hook:', other)
        hook_method = getattr(other, 'hook_method', None)
        return callable(hook_method)


class Bar(object):
    def hook_method(self):
        pass


class Baz(object):
    hook_method = 'hook'


class Qux(object):
```

```
    pass


print(issubclass(Bar, Foo))
print(issubclass(Baz, Foo))
print(issubclass(Qux, Foo))
```

▼ خروجی چیه؟

```
subclass hook: <class '__main__.Bar'>
True
subclass hook: <class '__main__.Baz'>
False
subclass hook: <class '__main__.Qux'>
False
```

@abc.abstractmethod

این دکوراتور باعث می شود تا یک زیرکلاس مجبور شود تا یک متد خاص را حتما override کند.

```
import abc


class Foo(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def baz(self):
        pass


class Bar(AbstractClass):
    pass


bar = Bar()
```

▼ خروجی چیه؟

```
Traceback (most recent call last):
  File "/home/samyar/Desktop/AP/main.py", line 14, in <module>
    bar = Bar()
TypeError: Can't instantiate abstract class Bar without an implementation for
```

```python
import abc


class Foo(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def baz(self):
        pass


class Bar(Foo):
    def baz(self):
        print('valid')


bar = Bar()
bar.baz()
```

▼ خروجی چیه؟

```
valid
```

یک مثال بهتر؟

```python
import abc


class Character(metaclass=abc.ABCMeta):
    def __init__(self, name:str):
        self.name = name


```

```python
    @abc.abstractmethod
    def shoot(self):
        pass


    def greet(self)->None:
        print(f'Hi! my name is {self.name}')


class Wizard(Character):
    def __init__(self, name:str):
        super().__init__(name)
        self.range = 10


    def shoot(self, distance:int)->None:
        if distance <= self.range:
            print('target destroyed')
        else:
            print('out of range')


class Archer(Character):
    def __init__(self, name:str, power:int, ammo:int):
        super().__init__(name)
        self.power = power
        self.ammo = ammo
        self.range = 15


    def shoot(self, distance:int, hp:int)->None:
        if self.ammo > 0:
            if distance <= self.range:
                if hp <= self.power:
                    print('target destroyed')
                else:
                    print('it\'s one shot bro')
            else:
                print('out of range')
            self.ammo -= 1
```

```
        else:
            print('out of ammo')


wiz = Wizard('Merlin')
archer = Archer('Hawk-eye', 40, 30)

wiz.greet()
archer.greet()

wiz.shoot(8)
archer.shoot(12, 20)
archer.shoot(5, 50)
```

▼ خروجی چیه؟

```
Hi! my name is Merlin
Hi! my name is Hawk-eye
target destroyed
target destroyed
it's one shot bro
```

در مثال بالا یک کلاس `Character` داریم که یه ویژگی `name` دارد و همچنین متد `greet` و متد `shoot` که یک متد انتزاعی است و به کلاس هایی که از کلاس `Character` ارث بری می کنند این امکان را می دهد تا مطابق میل و نیاز خود آن را override کنند.

و می بینیم که متد `shoot` در کلاس `Wizard` یک ورودی به جز `self` داشته اما همین متد در کلاس `Archer` با دو ورودی پیاده سازی شده.

مزایا:

استفاده از کلاس‌های انتزاعی مزایای زیادی دارد:

- **انعطاف‌پذیری**: با استفاده از کلاس های انتزاعی می‌توانیم اطمینان حاصل کنیم که کلاس‌های مختلف یک سری متدهای خاص را پیاده‌سازی می‌کنند.

- **کاهش پیچیدگی**: کلاس های انتزاعی کمک می‌کنند تا کدهای ما سازماندهی شده و منطقی باشند. به جای اینکه متدها را در هر کلاس به صورت جداگانه تعریف کنیم، یک قرارداد عمومی داریم که تمام کلاس‌ها از آن پیروی می‌کنند.
- **گسترش‌پذیری**: اگر بخواهیم انواع مختلف جدیدی از کرکترهای دیگر اضافه کنیم، کافی است یک کلاس جدید بسازیم که از کرکترهای موجود ارث ببرد و متدهای مورد نظر را پیاده‌سازی کند.

نکته

استفاده از کلاس‌های انتزاعی در پایتون یک ابزار قدرتمند برای طراحی کدهای منظم، قابل گسترش و قابل نگهداری است. با تعریف کلاس های انتزاعی، می‌توانیم پیاده‌سازی‌های خاص برای انواع مختلف اشیاء را تضمین کنیم و از پیچیدگی‌های غیرضروری جلوگیری کنیم. این روش در پروژه‌های بزرگ و تیمی بسیار مفید است، چرا که به وضوح مشخص می‌کند که هر کلاس باید چه متدهایی را پیاده‌سازی کند و چگونه با دیگر اجزای سیستم ارتباط برقرار کند.