

شعی گرایی : مبحث شعی گرایی باید خوب مطالعه بشه تا به خوبی درک بشه

---

یک مثال برای تشخیص شعی گرایی هست که همیشه من استفاده میکنم :

شما یک ماشین رو در نظر بگیرید که قراره ساخته بشه . ماشین ما یک کلاس هست. این کلاس یک ویژگی های ثابتی قراره داشته باشه که تمام ماشین های دنیا قراره داشته باشن : تمام ماشین ها قراره بدنه لاستیک استارت و پدال گاز داشته باشن .

حال ما میایم و از روی ماشین مون نمونه میسازیم : مثلا پراید . پراید از تمام ویژگی های ماشین ( کلاس ) ارث بری میکنه یعنی لاستیک استارت و پدال گاز داره . همچنین یکسری ویژگی هایی میگیرد که فقط مخصوص خودش است و به بقیه مدل ها داده نمیشود مثلا پراید ها قابلیت تیک آف دارند که فقط مخصوص پراید است .

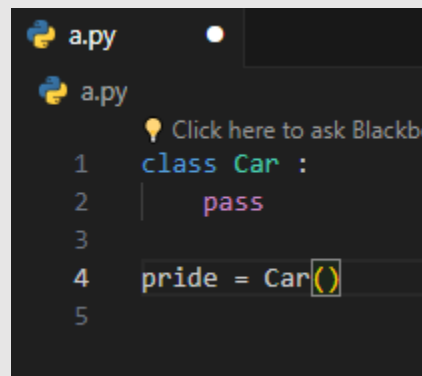
یک ماشین دیگه هم ساخته میشه به اسم بنز . این بنز هم از ویژگی های کلاس ماشین ارث بری میکنه و یکسری ویژگی منحصر به فرد میگیره مثل قابلیت ناز کردن راننده و شاگرد ولی دیگه ویژگی های پراید مثل تیک اف رو نداره چون فقط ویژگی تیک اف به خود پراید اختصاص داده شده.

حالا با پایتون چطوری کلاس بسازیم :

Class name :

Expression

P1 = name()



```
a.py
a.py
Click here to ask Blackboard
1 class Car :
2     pass
3
4 pride = Car()
5
```

حالا پراید از تمام ویژگی های Car ارث بری میکند و میتواند ویژگی های جدید نیز بگیرد

به نمونه های ساخته شده instance میگویند

---

حالا میتوانیم داخل کلاس ها یکسری توابع بنویسیم که به آن ها متود گفته میشوند . برای دیکلریفیکشن کردن براتون این مثال رو ببینید :

```
a.py > ...  
Click here to ask Blackbox to help  
1 my_list = []  
2  
3 print(type(my_list)) <class 'list'>  
4  
5 my_list.clear()  
6 new = my_list.copy()
```

وقتی نوع یک لیست یا هرچیزی رو میگیرفتیم نوع آنرا یک کلاس برمیگردونه یعنی چی :  
لیست یک کلاس هست که یکسری ویژگی مشترک داره که تمام لیست ها دارن مثل قابل  
تغییر بودن یا ایندکس داشتن یا قابلیت دایلیکیت شدن داشتن . یکسری ویژگی ها هم  
مخصوص بعضی از لیست ها هست مثل خالی بودن یا عدد داشتن و ...

اینجا میخام براتون کلاس لیست رو باز سازی کنم :

```
a.py > list > extend  
Codiumate: Options | Test this  
1 class list :  
2     def clear():  
3         pass  
4     def copy():  
5         pass  
6     def append():  
7         pass  
8     def extend():  
9         pass
```

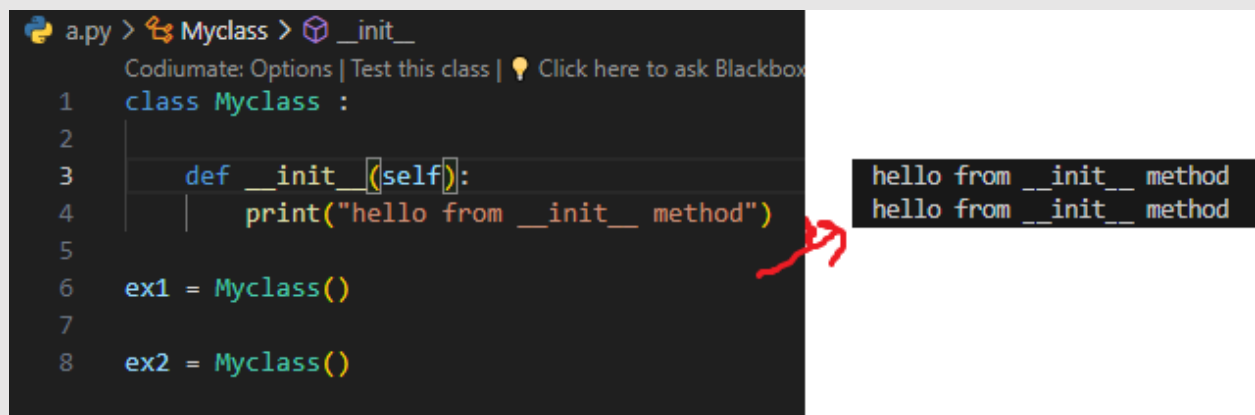
این یک مثال بازسازی شده از لیست ها هست . نیگا کنید : تمام توابعی که روی لیست ها استفاده می‌کردیم داخل کلاس ها به عنوان متود تعریف شده اند

پس به توابع داخل کلاس ها متد میگویند چه میخاد توابع داخل کلاس های ما باشند یا توابع داخلی خود کلاس های پایتون مثل لیست توپل استرینگ و ...

---

حالا تمام کلاس ها بایستی یک تابعی داشته باشند به اسم `__init__` ( داندر اینیت ) که عمده کلاس ها این رو دارند .

این تابع موقعی که نمونه گیری از کلاس میشه اجرا میشوند :



```
a.py > Myclass > __init__
Codiumate: Options | Test this class | Click here to ask Blackbox
1 class Myclass :
2
3     def __init__(self):
4         print("hello from __init__ method")
5
6 ex1 = Myclass()
7
8 ex2 = Myclass()
```

hello from \_\_init\_\_ method  
hello from \_\_init\_\_ method

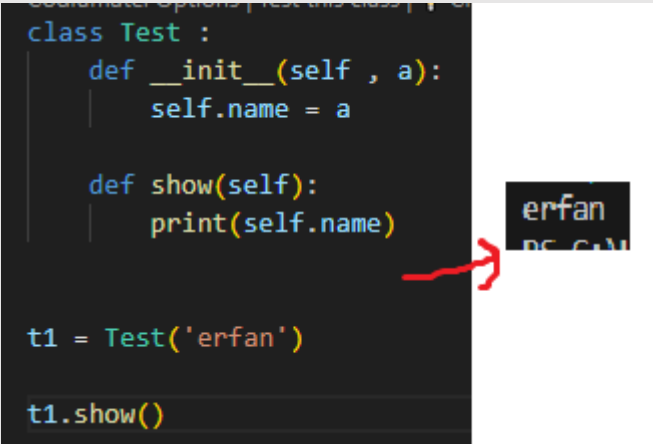
من الان بدون اینکه تابعی رو صدا بزنم اون متود `__init__` اجرا شده . چرا ؟ گفتیم که تابع `__init__` موقع نمونه گیری اجرا میشوند و ما اینجا 2 تا نمونه گرفتیم به اسم `ex1` و `ex2` .

سلف (self) : این سلف کوفتی چیه که همه جا هست و خیلیا نمیفهمن ؟

ببینید این سلف نماینده تمام متغیر های داخلی کلاس (class variable) هست . یعنی اگر متغیری در کلاس تعریف بشه ( فقط داخل کلاس و جای دیگه نشه استفاده کرد ) از داخل این سلف قابل دسترسی

مثلا :

```
class Test :  
    def __init__(self , a):  
        self.name = a  
  
    def show(self):  
        print(self.name)  
  
t1 = Test('erfan')  
  
t1.show()
```

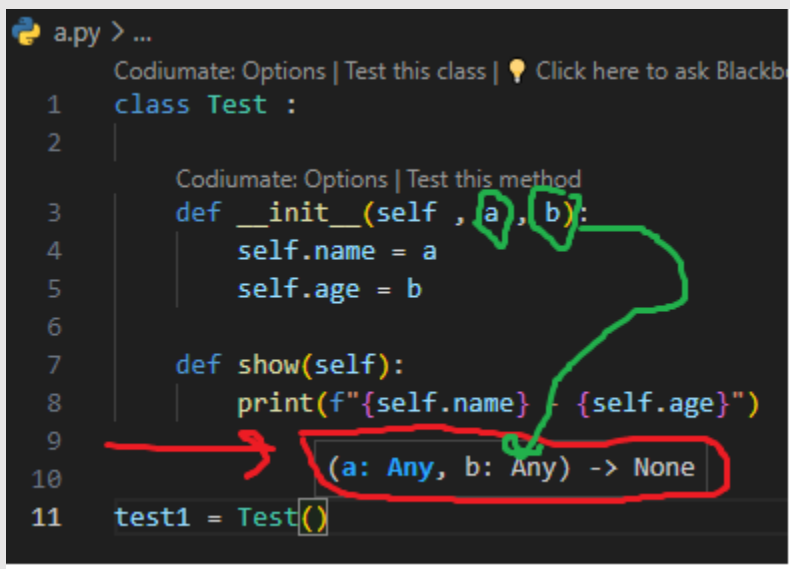


اینجا میبینید که موقع نمونه سازی اول تابع \_\_init\_\_ صدا زده میشود و بعد چون مقدار a را میخواهد باید موقع نمونه سازی به آن بدهیم . حالا داخل تابع \_\_init\_\_ اومدیم گفتیم که من میخام یه متغیر داخل کلاسی بسازم به اسم name و چون باید همه اینا در self ذخیره شوند نوشتیم self.name و به آن مقدار a را دادیم . این کار را با تابع show اثبات کردیم . به تابع show مقدار self را به عنوان ارگومان دادیم تا self که نماینده تمام متغیر های داخلی هست رو بگیره و بعد داخل تابع اومدیم و از توی self مقدار name رو صدا زدیم تا مقدار a چاپ شود

یک جای مناسب برای وارد کردن متغیر ها از خارج به داخل کلاس ها داخل تابع `__init__` هست :

گفتم که تابع `__init__` درجا بعد از نمونه گیری اجرا میشه و اگر ارگومانی داشته باشه موقع نمونه گیری باید داده بشه :

همونطور که میبینید موقع صدا زدن کلاس از مون مقدار `a` و `b` رو میخاد :



```
a.py > ...
Codiumate: Options | Test this class | Click here to ask Blackbox
1 class Test :
2
3     Codiumate: Options | Test this method
4     def __init__(self, a, b):
5         self.name = a
6         self.age = b
7
8     def show(self):
9         print(f"{self.name} - {self.age}")
10
11 test1 = Test()
```

```
a.py > ...
Codiumate: Options | Test this class | Click here to ask Blackbox to help you code faster
1 class Test :
2
3     Codiumate: Options | Test this method
4     def __init__(self , a , b):
5         self.name = a
6         self.age = b
7
8     def show(self):
9         print(f"{self.name} - {self.age}")
10
11 test1 = Test("erfna" , 12)
12 test1.show()
```

erfna - 12

اون مقادیری که به موقع صدا زدن کلاس ارسال میشوند به داخل تابع `__init__` میروند و به `a` و `b` میروند و اونجا ریخته میشوند به ترتیب و بعد آن `a` و `b` هم داخل توابع داخلی `name` و `age` ریخته میشوند. `( self.name )` نوشتم چون میخام داخل `self` یعنی نمایندش ذخیره بشه تا در کل کلاس قابل دسترسی باشه.

حالا یک مثال کاربردی :

```
a.py > ...
Codiumate: Options | Test this class | Click here to ask Blackbox to help you code faster
1 class Car :
2
3     Codiumate: Options | Test this method
4     def __init__(self , a , b):
5         self.name = a
6         self.model = b
7
8     def start(self):
9         print(f"{self.name} with {self.model} is starting")
10
11 pride = Car("pride" , 1400)
12 pride.start()
```

pride with 1400 is starting