

ارث بری در کلاس ها

وقتی یک کلاسی از یک کلاس دیگه ارث بری میکنه تمام ویژگی های اون رو از متغیر ها توابع و ... رو به ارث میبره :

```
test2.py > ...
Codiumate: Options | Test this class
1 class A :
2     num = 12
3     Codiumate: Options | Test this method
4     def __init__(self, i_a):
5         self.a = i_a
6         print("Hello from A class")
7
8     def show(self):
9         print(f"its name is {self.a} from A class and it has num of {self.num}")
10
11
12 class B(A):
13     pass
14
15 a = A('erfan')
16 b = B('berfan')
17
18 b.show()
```

```
Hello from A class
Hello from A class
its name is berfan from A class and it has num of 12
```

توی کلاس B من هیچی ننوشتیم ولی تمام ویژگی های A رو داره ارث بری میکنه حتی موقعی که دارم b رو میسازم ازم ارگومان i_a رو به عنوان ورودی میخاد ! . تابع show رو هم روی b اجرا کردم ولی توی B هیچی نیست ! به این میگن ارث بری

به کلاسی که ارث میبره فرزند میگن و به کلاسی که ارث میده میگن والد

حواستون باشه که کلاس فرزند از داشتن ویژگی های منحصر به فرد محروم نمیشه :

```

test2.py > B > good
1 class A :
    Codiumate: Options | Test this method
3     def __init__(self, i_a):
4         self.a = i_a
5
6         print("Hello from A class")
7
8     def show(self):
9         print(f"its name is {self.a} from A class and it has num of {self.num}")
10
11
12     Codiumate: Options | Test this class
13     class B(A):
14         def good(self):
15             print("hello from good fuction of B class")
16
17     a = A('erfan')
18     b = B('berfan')
19
20     b.show()
21     b.good()

```

```

Hello from A class
Hello from A class
its name is berfan from A class and it has num of 12
hello from good fuction of B class

```

اینجا کلاس B که فرزند یک ویژگی تابعی به اسم good هم گرفته که فقط برای خودش

حالا شاید ما بخواهیم فقط بخشی از ویژگی های والد رو بگیریم , یعنی فقط یکسری ویژگی ها به ارث برن :

فرض کنید که یک کلاس ماشین داریم که مقادیر اسم و قیمت رو از ورودی میگیرن , حالا میخاییم علاوه بر گرفتن اون ویژگی های والد (اسم و قیمت) وضعیت بیمه رو هم مشخص کنیم (فقط برای کلاس موتور):

Syntax : super().adjectives

Overwrite :

```
class Car :
    wheel = 4

    Codiumate: Options | Test this method
    def __init__(self , name , price):
        self.name = name
        self.price = price

    def show_car(self):
        print(f"i have a {self.name}")

    Codiumate: Options | Test this class
class Motor(Car):

    Codiumate: Options | Test this method
    def __init__(self , name , price , insurance):
        super().__init__(name , price)
        self.insurance = insurance
```

دقت کنید :

میگه که از باباچه ! برو سراغ داندرا اینیشتش و -> Super().__init__(name , price)
از اون اسم و قیمتش رو به ارث ببر

حواستون باشه که اون name و price باید توی ارگومان تابع فرزند هم قرار بگیره تا
به __init__ خودشم برسه ! به الاوه ویژگی های منحصر به فرد خودش

Insurance یک ویژگی منحصر به فرد برای خود کلاس Motor هست

ما الان اومدیم و متود __init__ تابع فرزند رو بازنویسی (overwrite) کردیم

یکبار دیگه توضیح بدم چون نفهمیدید :

میگم اهای ای کلاس Motor تو برو متود __init__ خودتو بساز که ویژگی name و price و insurance داره , خط بعد super() : اون ویژگی name و price تو از داند اینیت تابع والدت بگیر . self.insurance = insurance : این ویژگی رو هم منحصر به فرد برای خودت اضافه کن

پس وقتی داریم یک ویژگی والد رو overwrite میکنیم , از داخل سوپر صداش میزنیم و اون ویژگی هایی که میخایم رو ازش میاریم . هرچی رو میشه به ارث برد :

```
test2.py > Motor
Codiumate: Options | Test this class
1 class Car :
2     wheel = 4
3
4 Codiumate: Options | Test this method
5 def __init__(self , name , price):
6     self.name = name
7     self.price = price
8
9 def show_car(self):
10    print((f"i have a {self.name}"))
11
12 Codiumate: Options | Test this class
13 class Motor(Car):
14
15 Codiumate: Options | Test this method
16 def __init__(self , name , price , insurance):
17     super().__init__(name , price)
18     self.insurance = insurance
19
20 Codiumate: Options | Test this method
21 def show_motor(self):
22     super().show_car()
23     print("show car of Car is ran and also this is ran from Show motor ")
24
25 honda = Motor('honda' , 1000 , True)
26 honda.show_motor()
27 print(Motor.wheel)
```



```
i have a honda
show car of Car is ran and also this is ran from Show motor
4
```

داخل تابع show_motor اومدم گفتم که : توی کلاس والدت یه تابع show_car هست اول اونو اجرا کن بعد اون بیل بیلک رو پرینت کن . اخرم که پایین کد صداش زدم

!! حواستون باشه : توی تابع Car که موتور ازش ارث بری کرده نوشتیم :

Print(f'i have a {self.name}')

این self.name که هست چون توی کلاس Motor داره اجرا میشه پس هوندا چاپ میکنه نه اون self.name که توی کلاس Car هست مثلاً پراید .

__len__ :

میگه هر وقت از روی ابجکت ها len گرفته شد فلان رفتارو نشون بده :

```
test3.py > ...
Codiumate: Options | Test this class
1 class Class :
2     def __init__(self , name):
3         self.name = name
4
5     def __len__(self):
6         return len(self.name)
7
8
9
10 a = Class('erfan')
11
12 print(len(a))
```




گفتم اگر از روی ابجکت len گرفتن , len اون self.name رو بهش بده

`__add__` :

میگم وقتی ابجکت ما با یه ابجکت دیگه ای جمع شد فلان رفتارو نشون بده :

```
test3.py > First > __add__
Codiumate: Options | Test this class
1 class First :
2     def __init__(self , name):
3         self.name_first = name
4
5     def __add__(self , other_object):
6         return self.name_first + other_object.name_second
Codiumate: Options | Test this class
7 class Second :
8     def __init__(self , name):
9         self.name_second = name
10
11
12
13
14 a = First('erfan')
15 b = Second('gholi')
16
17 print(a + b)
```



erfangholi

با دقت بخونید : گفتیم اگر آبجکتی از کلاس First با آبجکتی از کلاسی دیگه جمع شد فلان رفتار رو نشون بده : توی ارگومان هاش یدونه سلف داریم که هیچی . یدونه `other_object` هم گذاشتیم چرا ؟ وقتی 2 تا ابجکت جم میشن , داندر `add` اولی اجرا و دومی به عنوان ارگومان به داندر `add` اولی ارسال میشه . حالا : گفتیم که `return` کن : `self.name_first` که یعنی اون متغیر رو از همون ابجکت (که داندر صدا زده شده ازش) به اضافه اون یکی ابجکته که یه متغیری داره به اسم `name_second` بکن .

Try except :

بعضی از مواقع ما می‌خواهیم اگر اروری دریافت شد اون رو شخصی سازی کنیم یا اصلن اجازه ندیم که ارور برنامه رو متوقف کنه :

Try :

Statement

Except :

Statement

حالا : میاد اون کد داخل try رو اجرا میکنه و اگر اروری دریافت کرد طبق رفتار کد های داخل except عمل میکنه :

```
a = 5

try :
    print(a.upper())
except:
    print("error - hamchin methodi nadare a")

print('done!')
```



a متدی به اسم upper ندارد پس ارور برمیگردد در حالت عادی ولی من گفتم که اگر ارور برگردوند (except) : بیا چاپ کن اون بیل بیلکه رو . (برنامه متوقف نمیشه)

بعضی از مواقع ما می‌خاییم که ارور هامون برنامه رو متوقف نکنن ! مثل موقعی که طرف ورودی اشتباه داد ولی بعضی از مواقع مثل اینجا من نمی‌خام برنامه متوقف بشه و فقط یه پیغامی چاپ بشه و به راه خودش ادامه بده .

Name mangling:

متغیر ها در کلاس ها به 3 دسته تقسیم میشوند :

1-public : متغیر های عمومی معمولی که تعریف میکنیم

2-protected : متغیر هایی که از لحاظ ظاهری حفاظت شدن که یک اندرسکور دارند

3-privet : متغیر هایی که خارج از کلاس قابل دسترس نیستند که دو اندرسکور دارند

```
test5.py > ...
Codiumate: Options | Test this c
1 class Test:
2     public = 5
3     _protected = 10
4     __privet = 25
5
6     print(Test.public)
7
8
```

X

متغیر پابلیک که قضیه ای نداره

حالا پروتکتد رو چجور باید صدا زد ؟

```
test5.py > ...
Codiumate: Options | Test this class
1 class Test:
2     public = 5
3     _protected = 10
4     __privet = 25
5     def _show(self):
6         print("hello from protected _show")
7
8     a = Test()
9     a._show()
10    print(Test._protected)
11
```


فقط کافیه اسم کامل متود یا متغیر رو (همراه با _ اش) صدا بزنی

****** این فقط جنبه ظاهری داره که وقتی داره کد شمارو میخونه بدوننه که این یه متغیری هست که نباید دست بهش بزنه یا تغییر اساسی توش بده ******

متغیر `privet` : این متغیر یا متود هارو هیچ جوره خارج از کلاس نمیشه دسترسی داشت بهشون :

```
test5.py > ...
Codiumate: Options | Test this class
1 class Test:
2     public = 5
3     _protected = 10
4     __privet = 25
5
6     def show_privet(self):
7         print(f'{self.__privet} is my privet variable which is accessible from inside the funciton')
8
9 a = Test()
10 a.show_privet()
11
```

```
25 is my privet variable which is accessible from inside the funciton
```

همون طور که دارید میبینید از داخل کلاس در دسترسه او ماا :

```
test5.py > ...
Codiumate: Options | Test this class
1 class Test:
2     public = 5
3     _protected = 10
4     __privet = 25
5
6
7 a = Test()
8 print(Test.__privet)
9
```

```
Traceback (most recent call last):
File "c:\Users\hamid\Desktop\New folder (2)\test\test5.py", line 10, in <module>
print(Test.__privet)
^^^^^^^^^^^^^^^^
AttributeError: type object 'Test' has no attribute '__privet'
```

اینجا از خارج از تابع ولی در دسترس نیست ارور میده .

حالا چو کنیم برای دسترسی ؟

Syntax for variable : Class._Class__variable

Syntax for Function: Object._Class__function

مثال :

```
test5.py > ...
Codiumate: Options | Test this class
1 class Test:
2     public = 5
3     _protected = 10
4     __privet_var = 25
5     def __privet(self):
6         print(f'this is called from __privet function with {self.__privet_var} this privet vairable')
7
8 a = Test()
9 print[Test._Test__privet_var]
10 a._Test__privet()
```

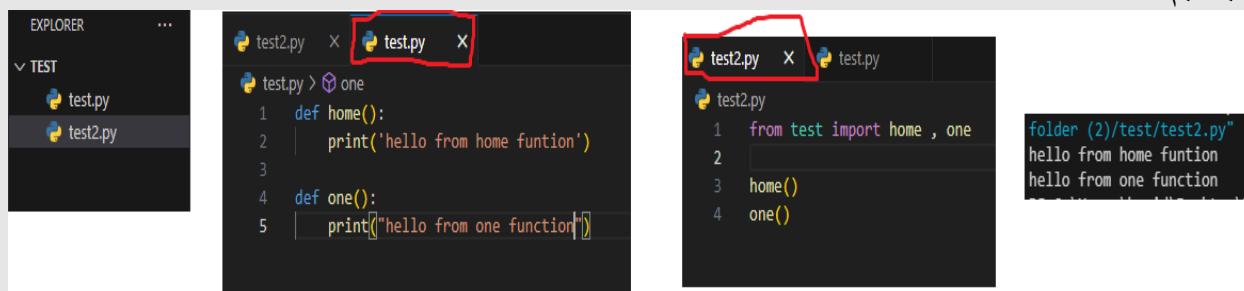
25
this is called from __privet function with 25 this privet vairable

هم اون متغیر رو چاپ کردم هم اون تابع رو صدا زدم

*** تابع از روی ابجکت صدا زده میشه نه خود کلاس ***

File management

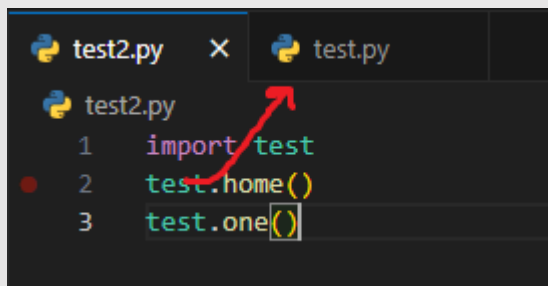
ما همیشه از یک فایل برای پایتون استفاده نمیکنیم و چندین فایل دخیل هستند در پروژه های ما . ما همیشه توابع و کلاس هایمون رو جدا مینویسم و داخل فایل اصلی پروژه وارد میکنیم :



در اینجا من داخل فایل test اومدم توابع رو نوشتم ولی در فایل test2 اون هارو Import کردم و صدا زدم . توی تصویر آخر هم اگر ببینید اون آبییه رو فایل test2 رو اجرا کنم (چون توی فایل 2 صدا زدم) و توابع اجرا میشوند.

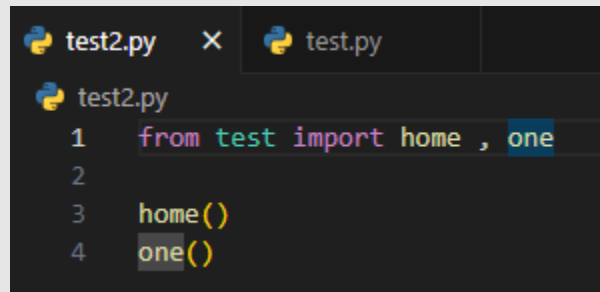
روش ها صدا زدن :

1- کلن هرچی داره وارد کنی و بعد قبل از صدا زدن اسم فایل رو هم بیاری :



وقتی مینویسم test.home() یعنی برو توی فایل home و test رو اجرا کن

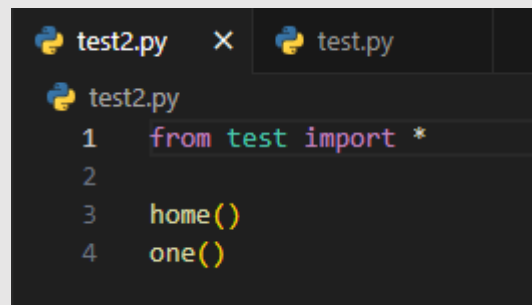
2- ایمپورت دونه دونه :



```
test2.py × test.py
test2.py
1 from test import home , one
2
3 home()
4 one()
```

****** دیگه نیازی به صدا زدن فایل نیست چون خودشو آوردیم ******

مشابه روش 1 کاری میشه کرد که همه رو بیاریم ولی دیگه نخاد اسم فایل رو صدا بزنینم :



```
test2.py × test.py
test2.py
1 from test import *
2
3 home()
4 one()
```

وقتی میگم ستاره یعنی هرچی توشه بردار بیار

!!!! در دنیای واقعی ما هیچوقت همه چی رو برنمیاریم بیاریم چون الان پروژه کوچیکه ولی مثلا کتابخونه بزرگی باشه یا فریم ورک بزرگی باشه اگر هرچی داره برداریم بیاریم پروژه بهینه نمیشه و سرعت خیلیییییییییی میاد پایین :

```
home > views.py > PostUpdateView > dispatch
1  from django.shortcuts import render , redirect , get_object_or_404
2  from django.views import View
3  from .models import Post
4  from django.contrib.auth.mixins import LoginRequiredMixin
5  from django.contrib import messages
6  from .forms import PostUpdateForm
7  from django.utils.text import slugify
8
```

این یک پروژه جنگویی هست. میبینید که هر کدام از ابزار ها دونه دونه وارد میشه

درباره اونی که نوشته model یا forms. که داره به فایل پایتونی اشاره میکنه هم نپرسید که الان وقتش نیست