برنامه سازی پیشرفته
(برنامه نویسی شیءگرا: وراثت، ماژول ها و پکیج ها)

صادق اسکندری - دانشکده علوم ریاضی، گروه علوم کامپیوتر

eskandari@guilan.ac.ir

یادآوری ...

پایتون برای هر یک از عملگرهای درون ساخت، متدهای خاصی تحت عنوان متدهای جادویی (Magic Methods) را تعریف کرده است. این متدها دارای نام مشخص بوده و در ابتدا و انتهای نام آنها از ___ استفاده شده است.

عملگر	نام متد معادل
+, -, %, ** *, /, //	__add__, __sub__, __mod__, __pow__ __mul__, __truediv__, __floordiv__
==, >, <=, >=, <, !=	__eq__, __gt__, __le__, __ge__, __lt__, __ne__
طول	__len__
|	__or__
+=, -=, %=, **= *=, /=, //=	__iadd__, __isub__, __imod__, __ipow__ __imul__, __idiv__, __ifloordiv__

یادآوری ...

کپسوله سازی: مخفی کردن پیچیدگی های غیرضروری

کپسوله سازی در نوع داده لب تاپ: کاربر نیازی به دیدن و دسترسی به حافظه، پردازنده و ... ندارد بنابراین، در محصول نهایی، این قطعات نباید به شکل فیزیکی دیده شوند. حتی بخش هایی که مشتری نیاز به دسترسی به آنها دارد نیز از طریق واسط ها انجام می شوند. به عنوان مثال، کاربر نیاز دارد تا ورودی را بر روی باغرهای ورودی قرار دهد. ولی کاربر دسترسی مستقیم به باغرها ندارد و این عمل از طریق واسطی به نام کیبورد انجام می شود.

کپسوله سازی در نوع داده ماشین: کاربر نیازی به دیدن و دسترسی به گیربکس، تسمه تایم، انژکتور و ... ندارد بنابراین، در محصول نهایی، این قطعات نباید به شکل فیزیکی دیده شوند. اگرچه کاربر نیاز به تغییر جهت حرکت چرخ ها دارد، ولی این دسترسی از طریق واسطی به نام فرمان انجام می شود.

یادآوری ...

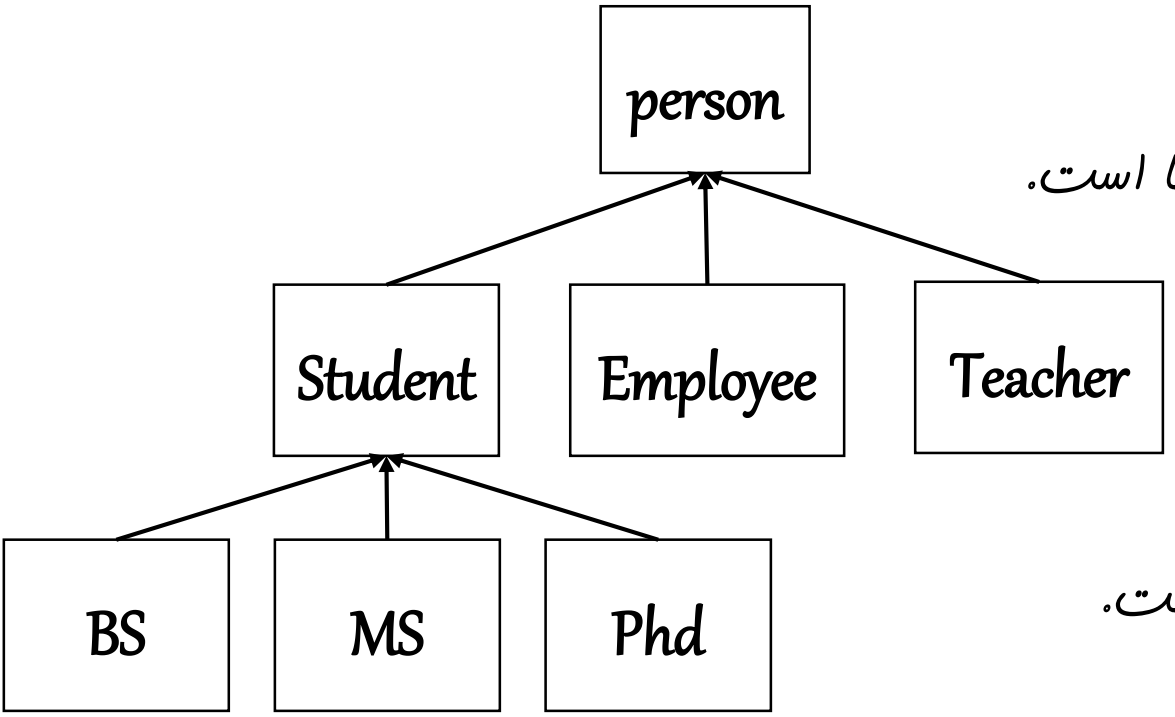
کپسوله سازی در طراحی کلاس ها:

۱- متدهایی که خارج از کلاس استفاده نمی شوند (مانند متد `seconds_to_time()`) را با قرار دادن __ در ابتدای نام آنها، مخفی کن

۲- صفاتی که دسترسی مستقیم به آنها از طریق اشیاء می تواند مشکل ساز باشد (مانند دقیقه ها و ثانیه ها در ساعت) را با قرار دادن __ در ابتدای نام آنها مخفی کرده و سپس یک متد setter و یک متد getter جهت دسترسی و دستکاری آنها ایجاد کن.

وراثت

وراثت (Inheritance) روشی است جهت دسته بندی سلسله مراتبی انواع داده، از کلاس های کلی تا کلاس های خاص



کلاس Person، یک کلاس مافوق (Superclass) برای سایر کلاس ها است.

کلاس Student، یک کلاس مافوق برای کلاس های BS، MS و Phd است.

کلاس Student، یک کلاس مشتق (Subclass) از کلاس Person است.

هر کلاس مشتق، نوع خاصی از کلاس مافوق است. بنابراین، تمامی ویژگی ها و متدهای کلاس مافوق را به ارث برده و خود تعدادی ویژگی و متد به آنها اضافه می کند. این کار به روال طراحی کمک کرده و موجب استفاده مجدد از کدها می شود.

وراثت

مثال: اشکال دوبعدی

```
class TwoDShape:
    def __init__(self, w=1, h=1):
        self.width = w
        self.height = h

    def __str__(self):
        return "TwoDShape: (Width: %.2f , Height: %.2f)"%(self.width, self.height)
```

```
t = TwoDShape(10,20)
print(t)
```

```
TwoDShape: (Width: 10.00 , Height: 20.00)
```

وراثت

مثال: اشکال دوبعدی

کلاس Rectangle یک کلاس مشتق از TwoDShape

```
class Rectangle(TwoDShape):
    def __init__(self, w=1, h=1):
        super(Rectangle,self).__init__(w,h)

    def area(self):
        return self.width*self.height

    def __str__(self):
        return "Rectangle --> (%s) --> (area: %.2f)"%(super(Rectangle,self).__str__(),self.area())
```

فراخوانی متد کلاس مافوق توسط کلاس مشتق با استفاده از کلمه کلیدی super

متد خاص کلاس مشتق (کلاس مافوق فاقد این متد است)

در صورت نیاز، کلاس مشتق می تواند تعریف متدهای کلاس مافوق را عوض کند (Overload)

وراثت

مثال: اشکال دوبعدی

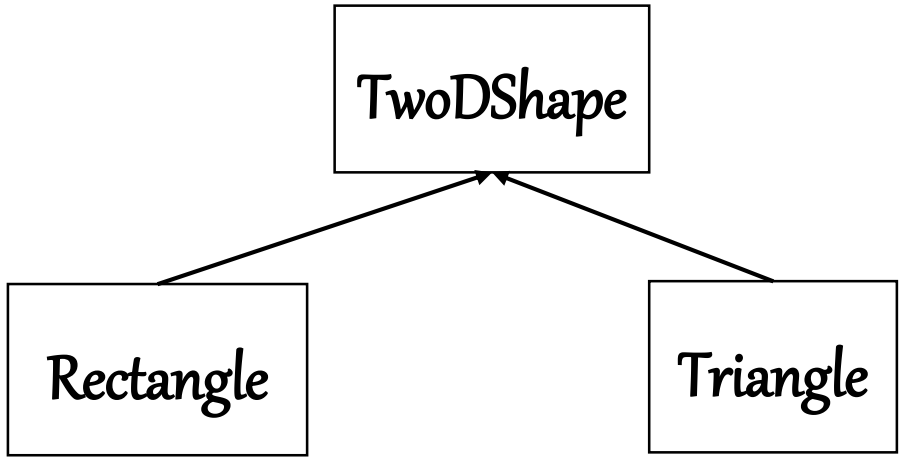
```
class Rectangle(TwoDShape):
    def __init__(self, w=1, h=1):
        super(Rectangle,self).__init__(w,h)

    def area(self):
        return self.width*self.height

    def __str__(self):
        return "Rectangle --> (%s) --> (area: %.2f)"%(super(Rectangle,self).__str__(),self.area())
```

```
t = TwoDShape(10,20)
print(t)

r = Rectangle(10,20)
print(r)
```

```
TwoDShape: (Width: 10.00 , Height: 20.00)
Rectangle --> (TwoDShape: (Width: 10.00 , Height: 20.00)) --> (area: 200.00)
```

وراثت

مثال: اشکال دوبعدی



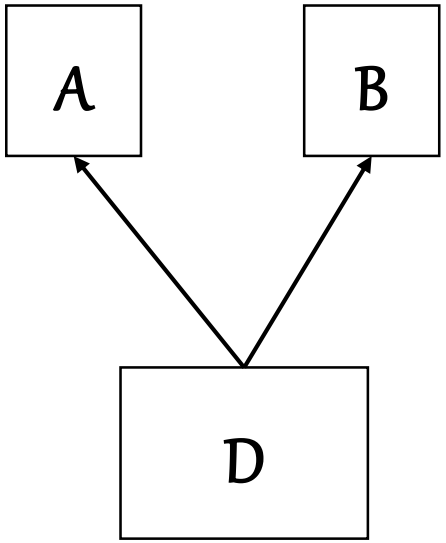

```
class Triangle(TwoDShape):
    def __init__(self, w=1, h=1):
        super(Triangle,self).__init__(w,h)

    def area(self):
        return self.width*self.height/2

    def __str__(self):
        return "Triangle --> (%s) --> (area: %.2f)"%(super(Triangle,self).__str__(),self.area())
```

```
tr = Triangle(10,20)
print(tr)
```

```
Triangle --> (TwoDShape: (Width: 10.00 , Height: 20.00)) --> (area: 100.00)
```

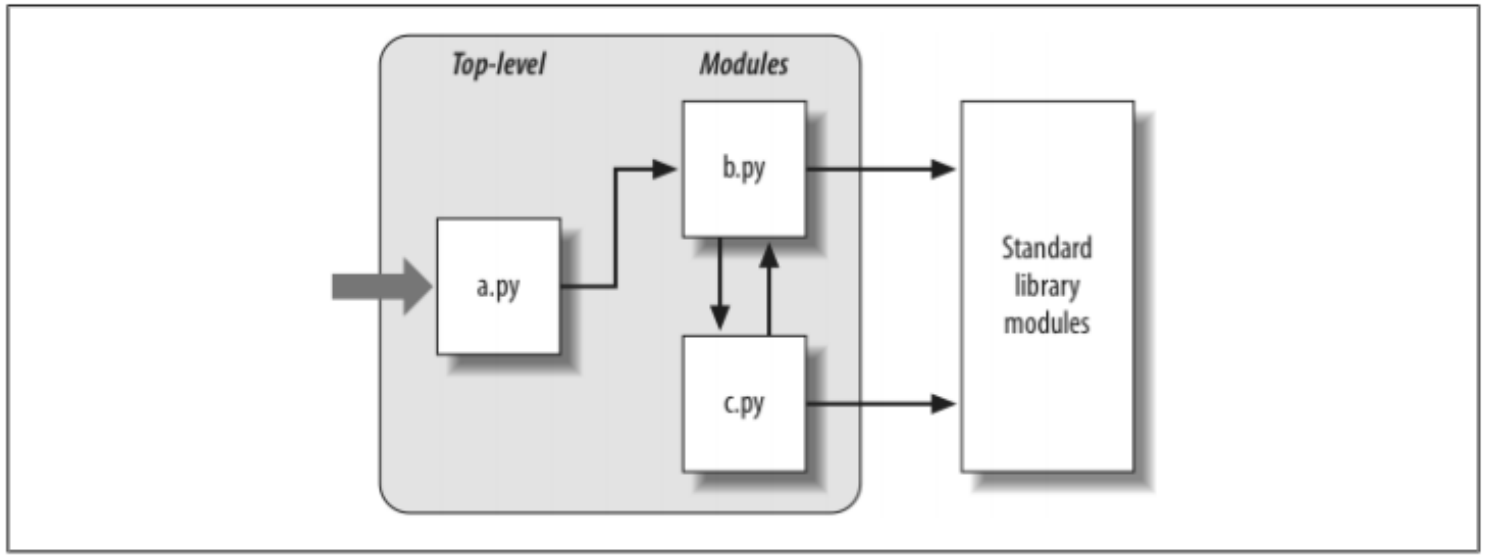
وراثت

نکته: در پایتون، امکان وراثت چندگانه (مشتق شدن یک کلاس از چندین کلاس) وجود دارد. ولی با توجه به پیچیدگی هایی که ایجاد می کند، استفاده از آن توصیه نمی شود.



ماژول ها

یک ماژول (Module) فایلی شامل تعاریف و دستورالعمل های پایتون جهت استفاده در سایر برنامه های پایتون، است.



اغلب یک برنامه به زبان پایتون، سیستمی از ماژول ها است. این برنامه شامل یک فایل اسکریپت سطح بالا (**a.py**) جهت اجرای برنامه است. سایر ماژول ها ممکن است توسط خود برنامه نویس نوشته شده باشند (**b.py** و **c.py**)، یا بخشی از کتابخانه استاندارد نصب شده (**math**، **Turtle**، **pygame**) باشند.

ماژول ها

مثال:

محتویات فایل (ماژول) IntOps.py

```python
import math
def isPrime(n):
    for i in range(2,int(math.sqrt(n)+1)):
        if n%i == 0:
            return False
    return True

def num_of_digits(n):
    if(n<10):
        return 1
    return 1 + num_of_digits(n//10)

def sum_of_digits(n):
    if(n<10):
        return n
    return (n%10) + sum_of_digits(n//10)
```

ماژول ها

محتویات فایل (ماژول) اصلی برنامه

مثال:

```
import IntOps

print(IntOps.isPrime(23))
print(IntOps.sum_of_digits(23))
```

```
True
5
```

☹☹

```
import IntOps as myops

print(myops.isPrime(23))
print(myops.sum_of_digits(23))
```

```
True
5
```

☹

```
from IntOps import isPrime, sum_of_digits

print(isPrime(23))
print(sum_of_digits(23))
```

```
True
5
```

☺

```
from IntOps import isPrime as prime_check, sum_of_digits as sod

print(prime_check(23))
print(sod(23))
```

```
True
5
```

☺☺

پکیج ها

یک پکیج (Package) شامل مجموعه ای از ماژول ها است.

از نظر فنی، یک پکیج، پوشه ای شامل تعدادی ماژول به همراه یک فایل به نام __init__.py است.

```
sound
|-- effects
|   |-- echo.py
|   |-- __init__.py
|   |-- reverse.py
|   `-- surround.py
|-- filters
|   |-- equalizer.py
|   |-- __init__.py
|   |-- karaoke.py
|   `-- vocoder.py
|-- formats
|   |-- aiffread.py
|   |-- aiffwrite.py
|   |-- auread.py
|   |-- auwrite.py
|   |-- __init__.py
|   |-- wavread.py
|   `-- wavwrite.py
`-- __init__.py
```

فایل __init__.py می تواند یک فایل خالی باشد.

یک پکیج می تواند شامل تعدادی زیرپکیج (زیرپوشه) باشد که هر یک دارای ماژول ها و فایل های __init__.py خود هستند.