برنامه سازی پیشرفته
(برنامه نویسی شیءگرا: مفاهیم بیشتر)

صادق اسکندری - دانشکده علوم ریاضی، گروه علوم کامپیوتر

eskandari@guilan.ac.ir

یاد آوری ....

```python
class MyTime:
    def __init__(self, h = 0, m = 0, s = 0):
        self.hours = h
        self.minutes = m
        self.seconds = s
        self.hours, self.minutes, self.seconds = self.seconds_to_time(self.to_seconds())

    def to_seconds(self):
        return self.hours * 3600 + self.minutes * 60 + self.seconds

    def seconds_to_time(self,s):
        h = s // 3600
        m = (s % 3600) // 60
        s = (s % 3600) % 60
        return h,m,s

    def __str__(self):
        return '%d : %d : %d'%(self.hours, self.minutes, self.seconds)

    def after(self, time2):
        """ Return True if I am strictly greater than time2 """
        return self.to_seconds() > time2.to_seconds()

    def increment(self,time2):
        self.hours, self.minutes, self.seconds = self.seconds_to_time(self.to_seconds()+time2.to_seconds())
```

Operator Overloading

باز تعریف عملگرها و متدهای درون-ساخت پایتون برای انواع داده جدید

تابحال با یک نمونه بازتعریف آشنا شدیم:

برای استفاده از تابع درون-ساخت print، از متد __str__ استفاده کردیم.

Operator Overloading

سوال: خروجی کد زیر چیست؟

```
t1 = MyTime(7,61,61)
t2 = MyTime(8,0,86)

t3 = t1 + t2
print(t3)
```

```
Traceback (most recent call last):
  File "h:\Tadris\Advanced Programming\Python\Codes\test\MyTime\MyTimeV2.py", line 34, in <module>
    t3 = t1+t2
TypeError: unsupported operand type(s) for +: 'instance' and 'instance'
```

سوال: چگونه می توان از عملگر + برای افزودن دو زمان به یکدیگر استفاده کرد؟

باید عملگر + را overload کنیم. یعنی معنی جدیدی برای آن تعریف کنیم.

Operator Overloading

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__

Operator Overloading

بنابراین، برای افزودن قابلیت جمع دو ساعت، باید متد جادویی __add__ را در کلاس MyTime بازتعریف (Overload) کنیم.

```
    def __add__(self, time2):
        return MyTime(0,0,self.to_seconds() + time2.to_seconds())
```

```
t1 = MyTime(7,61,61)
t2 = MyTime(8,0,86)

t3 = t1 + t2
print(t3)
```

```
16 : 3 : 27
```

Operator Overloading

چند مثال دیگر:

```
    def __le__(self,time2):
        return self.to_seconds() <= time2.to_seconds()

    def __eq__(self, time2):
        return self.to_seconds == time2.to_seconds()

    def __sub__(self,time2):
        if self <= time2:
            return MyTime(0,0,0)
        else:
            return MyTime(0,0,self.to_seconds() - time2.to_seconds())
```

```
t1 = MyTime(7,61,61)
t2 = MyTime(8,0,86)

print(t1 - t2)
print(t2 - t1)
```

```
0 : 0 : 35
0 : 0 : 0
```

Operator Overloading

یک مثال دیگر:

```
    def __del__(self):
        print("%s  is dying :( "%(self.__str__()))
```

```
for i in range(10000):
    t = MyTime(0,0,i)
```

```
0 : 0 : 0  is dying :(
0 : 0 : 1  is dying :(
0 : 0 : 2  is dying :(
0 : 0 : 3  is dying :(
0 : 0 : 4  is dying :(
0 : 0 : 5  is dying :(
0 : 0 : 6  is dying :(
0 : 0 : 7  is dying :(
0 : 0 : 8  is dying :(
0 : 0 : 9  is dying :(
0 : 0 : 10  is dying :(
0 : 0 : 11  is dying :(
0 : 0 : 12  is dying :(
```

تمرین

کلاس Point را به گونه ای بازنویسی کنید که امکان مقایسه نقاط (از نظر فاصله تا مرکز)، جمع و تفریق دو نقطه و ضرب یک نقطه در یک عدد ثابت وجود داشته باشد.

کلاس مستطیل را به گونه ای بازنویسی کنید که امکان مقایسه دو مستطیل از نظر اندازه وجود داشته باشد.

متد جادویی _del_ در کلاس Ball را به گونه ای بازنویسی کنید که توپ ها در زمان برخورد به یکدیگر منفجر شده و حذف گردند.

با استفاده از متد _add_، کلاس Ball را به گونه ای بازنویسی کنید که در زمان برخورد توپ ها با یکدیگر، توپ بزرگتر، توپ کوچکتر را بلعیده و اندازه آن افزایش پیدا کند.

کپسوله سازی (Encapsulation)

```python
class MyTime:
    def __init__(self, h = 0, m = 0, s = 0):
        self.hours = h
        self.minutes = m
        self.seconds = s
        self.hours, self.minutes, self.seconds = self.seconds_to_time(self.to_seconds())

    def to_seconds(self):
        return self.hours * 3600 + self.minutes * 60 + self.seconds

    def seconds_to_time(self,s):
        h = s // 3600
        m = (s % 3600) // 60
        s = (s % 3600) % 60
        return h,m,s

        ⋮
```

این متد تنها درون کلاس مورد استفاده قرار می گیرد (متد کمکی برای سایر متدها). چگونه می توان آن را مخفی کرد؟

```python
t1 = MyTime(10,10,10)
t1.minutes=100
print(t1)
```

```
10 : 100 : 10
```

☹

چگونه می توان مقداردهی به ویژگی های کلاس را کنترل کرد؟

کپسوله سازی (Encapsulation)

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.

کپسوله سازی (Encapsulation)

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.

کپسوله سازی (Encapsulation)

کپسوله سازی در کلاس MyTime

۱- متد seconds_to_time یک متد کمکی است و بنابراین، بهتر است آن را به صورت زیر تعریف کنیم:

```
def __seconds_to_time(self,s):
    h = s // 3600
    m = (s % 3600) // 60
    s = (s % 3600) % 60
    return h,m,s
```

کپسوله سازی (Encapsulation)

کپسوله سازی در کلاس MyTime

۲- ویژگی hours نمی تواند مقداری منفی دریافت کند. بنابراین، بهتر است این ویژگی را مخفی کرده (نام آن را __hours قرار دهیم) و برای آن یک setter و یک getter تعریف کنیم:

```
@property
def Hours(self):
    return self.__hours

@Hours.setter
def Hours(self, value):
    if value < 0:
        print("hours should be positive")
    else:
        self.__hours = value
```

```
t1 = MyTime(10,20,30)
print(t1)
t1.Hours=40
print(t1)
t1.Hours = -20
print(t1.Hours)
#print(t1.__hours)      ===> Error
```

```
10 : 20 : 30
40 : 20 : 30
hours should be positive
40
```

کپسوله سازی (Encapsulation)

```
@property
def Minutes(self):
    return self.__minutes

@Minutes.setter
def Minutes(self,value):
    if value >= 60 or value < 0:
        print("minutes should be in range [0,60)")
    else:
        self.__minutes = value

@property
def Seconds(self):
    return self.__seconds

@Seconds.setter
def Seconds(self,value):
    if value >= 60 or value < 0:
        print("seconds should be in range [0,60)")
    else:
        self.__seconds = value
```

کپسوله سازی در کلاس MyTime

۳- ویژگی های minutes و seconds نمی توانند مقداری خارج از [0,60) دریافت کنند. بنابراین، بهتر است این ویژگی ها را نیز مخفی کرده و برای آنها setter و getter تعریف کنیم: