این مار خوش خط و خال

قسمت چهارم

آشنایی و کار با زبان برنامه‌نویسی پایتون

» احمد شریف پور

انسان ناخودآگاه و به شیوه‌ای نوستالژیک سعی دارد پدیده‌های تازه اطرافش را به موارد شناخته شده ربط داده یا تبدیل کند و آن‌ها را بر اساس قوانین شناخته شده توضیح دهد. به عنوان مثال، اتومبیل‌ها را با صفاتی مانند خشمگین، هیولا و.... بخواند، برنامه‌های مخرب کامپیوتری را ویروس بنامد و حتی در برنامه‌نویسی سعی کند همه‌چیز یک برنامه را به شیء تبدیل کند. گویی در این دنیای صفر و یکی هم مانند زندگی روزمره با خرده‌ریزهای فیزیکی سروکار دارد. بیشتر شما به یقین اصطلاح برنامه نویسی شیءگرا یا OOP (سرنام Object Oriented Programming) را شنیده‌اید. این اصطلاح به‌طور معمول در برابر اصطلاح برنامه‌نویسی تابع‌گرا یا Functional مطرح می‌شود و به نوعی از برنامه‌نویسی اشاره دارد که به‌صورت خلاصه سعی دارد مجموعه داده‌های دارای خواص و عملکرد یکسان را به عنوان یک نوع داده یا DataType طبقه‌بندی کرده و از این راه مدیریت کد را ساده‌تر کند. به‌واسطه قولی که در قسمت پیشین داده بودیم، این شماره را به بررسی برنامه‌نویسی شیءگرا و به‌ویژه شیءگرایی در پایتون اختصاص داده‌ایم.

میانه راه

نکته جالب توجه این که شما استفاده از اشیا، کلاس‌ها، متدها و خلاصه تمام جنبه‌های برنامه‌نویسی شیءگرا را زمانی که نخستین خط کد به زبان پایتون را نوشته‌اید، شروع کرده‌اید. بنابراین، شما اکنون در میانه راه هستید! تمام اجزای پایتون براساس سیستم شیءگرا نوشته شده‌اند. در پایتون هرچیزی، حتی یک مقدار عددی نظیر ۱/۶۷ نیز یک شیء است. اما تمام این هیاهو بر سر چیست؟ کلاس و شیء چیستند؟

در تعاریف برنامه‌نویسی «شیء» مجموعه‌ای از داده‌ها است که دارای تعدادی خصوصیت یا Attribute و تعدادی عملکرد یا Method است. خواص، نگه‌دارنده مشخصات و وضعیت شیء هستند و متدها، قابلیت‌ها و عملیات ممکن روی یک شیء را نشان می‌دهند. کلاس نیز درواقع دستورالعمل ساخت یک شیء جدید است. اگر یک اتومبیل را به عنوان یک شیء در نظر بگیریم، دارای خصوصیاتی نظیر نام، مدل، حداکثر سرعت، تاریخ ساخت و... خواهد بود. عملیات استارت زدن، حرکت به جلو، ترمز گرفتن و... نیز متدهای این شیء به شمار می‌آیند. در این حالت مستندات و طرح‌های موجود در کارخانه، که اتومبیل از روی آن‌ها ساخته می‌شود، کلاس این شیء خواهد بود.

به خط فرمان پایتون بروید و متغیر s را برابر "ABCD" (به بزرگ بودن تمام حروف دقت کنید) تعریف کنید. می‌دانید که این متغیر از نوع رشته خواهد بود. به‌صورت دقیق‌تر s یک شیء (Object) از نوع (Class) رشته

```
Python 2.7 (r27:82525, Jul  4 2010, 09:01:59) [MSC v.1500
32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more infor
mation.
>>> s="ABCD"
>>> dir(s)
['__add__', '__class__', '__contains__', '__delattr__', '
__doc__', '__eq__', '__format__', '__ge__', '__getattribu
te__', '__getitem__', '__getnewargs__', '__getslice__', '
__gt__', '__hash__', '__init__', '__le__', '__len__', '__
lt__', '__mod__', '__mul__', '__ne__', '__new__', '__redu
ce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__
', '__setattr__', '__sizeof__', '__str__', '__subclasshoo
k__', '_formatter_field_name_split', '_formatter_parser',
'capitalize', 'center', 'count', 'decode', 'encode', 'end
swith', 'expandtabs', 'find', 'format', 'index', 'isalnum
', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle',
'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition
', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', '
rsplit', 'rstrip', 'split', 'splitlines', 'startswith', '
strip', 'swapcase', 'title', 'translate', 'upper', 'zfill
']
>>>
```

شکل۱ فهرست خواص و متدهای شیء

```
1  class Point:
2          x=0
3          y=0
4
5  p1= Point()
6  p2= Point()
7
8  p1.x=7
9  p2.x=9
10 p1.name = “Start Point”
11
12 print p1.x , p2.x
13 print p1.name
14 print p1
```

فهرست۱ تعریف نوع داده‌ای جدید برای شیء نقطه

است. دستور dir(s) را تایپ کنید. همان‌گونه که در شکل ۱ می‌بینید، فهرست تمام خواص و عملکردهای شیء s برای شما نمایش داده خواهد شد. برای استفاده از خواص و متدها، از نماد نقطه‌گذاری معمول استفاده می‌کنیم. تنها توجه داشته باشید که برای فراخوانی متدها به علامت () نیاز خواهیم داشت.. به عنوان نمونه دستور s.__doc__ را تایپ کنید. (علامت __ از دو زیر خط پیوسته تشکیل شده است). همان‌طور که می‌بینید این خاصیت توضیحات مربوط به شیء رشته را چاپ خواهد کرد. اکنون از دستور s.isupper() استفاده کنید. این یکی از متدهای شیء رشته است که اگر تمام حروف رشته بزرگ باشند، مقدار True وگرنه False را بازخواهد گرداند. اگر این متد را بدون پرانتز فراخوانی کنید، پایتون تنها نوع و آدرس آن را برای شما نمایش خواهد داد. از dir() می‌توان برای استخراج خواص و متدهای تمام اشیا، داده‌ها، توابع، ماجول‌ها و... استفاده کرد.

کاربرد

استفاده از سیستم شیءگرا در برنامه‌ها مزیت‌های فراوانی را به همراه خواهد داشت. ابتدا این‌که اجزای برنامه به سیستم ادراکی روزمره ما نزدیک‌تر شده و به همین دلیل درک منطق برنامه‌نویسی در بسیاری موارد ساده‌تر از حالت تابعی (روشی که در برنامه‌های قسمت‌های قبل

```
>>>
7 9
Start Point
<__main__.Point instance at 0x01F3BBE8>
>>>
```

شکل۲ خروجی برنامه فهرست ۱

به‌کار می‌بردیم) خواهد بود. همچنین در این روش به دلیل پیاده‌سازی متدها و داده‌ها در خود شیء، نیاز به انتقال اطلاعات بین توابع مختلف و ارجاع‌های متعدد از میان خواهد رفت و دیگر این‌که با فراهم شدن امکان ارث‌بری خصوصیات میان اشیا، پیاده سازی اشیا جدید و کلاس‌های پیچیده راحت‌تر می‌شود.

فرض کنید با برنامه‌ای سروکار داریم که قرار است مشخصات و روابط هندسی میان تعدادی نقطه را پردازش و محاسبه کند. به مفهوم ریاضی نقطه فکر کنید که عبارت است از دو عدد (مختصات) که در مجموع به عنوان یک شیء واحد در نظر گرفته می‌شوند. در پیاده‌سازی برنامه و به عنوان یک راه‌حل ساده می‌توانید مختصات نقاط را در یک توپل یا یک لیست قرار دهید. اگرچه این کار می‌تواند تا حدودی نیاز برنامه را برآورده کند، اما برای کار با داده‌های زیاد، اصلاً مناسب نخواهد بود. بهترین کار تعریف یک نوع داده (Data Type) اختصاصی جدید برای کار با نقطه است. فهرست ۱ را در ویرایشگر دلخواه خود وارد کرده، با نام Points01.py ذخیره کرده و آن را اجرا کنید. خروجی باید همانند شکل ۲ باشد.

در خط اول با کلمه کلیدی class و سپس نام نوع داده جدید، آن را تعریف کرده‌ایم. در خطوط بعدی، مختصات طول و عرض در داخل خود این کلاس تعریف شده و برابر صفر مقدار داده شده‌اند. تعاریف کلاس به‌طور معمول در ابتدای برنامه و بعد از دستورات import آورده می‌شوند و رسم بر این است که نام کلاس‌های جدید همواره با حروف بزرگ آغاز شوند. در خطوط 5, 6 دو متغیر جدید از نوع نقطه تعریف شده و سپس مقدار x و y آن‌ها دست‌کاری شده است. تعریف متغیر از یک نوع جدید را به اصطلاح وله‌سازی، نمونه‌سازی یا Instantiation می‌نامند. در خطوط 12 تا 14 مقادیر برخی خصوصیات این اشیا چاپ خواهد شد. همان‌گونه که در خط 10 مشاهده می‌کنید، به کمک نماد نقطه‌گذاری می‌توان داده جدیدی (نظیر نام) را نیز به یک نمونه از یک کلاس اضافه کرد. اگر تلاش کنید تا خود شیء p1 را چاپ کنید، پایتون نوع یا کلاس سازنده آن را باز خواهد گرداند.

به سادگی می‌توان نمونه‌های یک شیء را همانند یک مقدار یا یک متغیر به یک تابع پاس کرد. فایل قبلی را همانند فهرست ۲ ویرایش کرده و با نام Points02.py ذخیره کنید. پس از اجرای این کد خروجی برنامه همانند شکل ۳ خواهد بود.

```
>>>
Point at 5,9
Point at -9,4

P1 after moving . . .
Point at 2,9
P2 after moving . . .
Point at 1,4
>>>
```

شکل۳ خروجی برنامه فهرست ۲

```
1  class Point:
2          x=0
3          y=0
4  #### Functions
5  def PrintPoint(p):
6          text=»Point at %d,%d» %(p.x,p.y)
7          print text
8  def MoveLeft(p,how_much):
9          p.x = p.x - how_much
10 def MoveRight(p,how_much):
11         p.x = p.x + how_much
12 #### Instantiation
13 p1= Point()
14 p1.x = 5
15 p1.y = 9
16
17 p2= Point()
18 p2.x = -9
19 p2.y = 4
20
21 PrintPoint(p1)
22 PrintPoint(p2)
23 #### Moving points
24 MoveLeft(p1,3)
25 MoveRight(p2,10)
26 #### Printing results
27 print
28 print “P1 after moving . . . “
29 PrintPoint(p1)
30 print “P2 after moving . . . “
31 PrintPoint(p2)
```

فهرست ۲ تعریف توابعی برای کار با انواع داده جدید

در خط 5 تابعی تعریف کرده‌ایم که یک نقطه را به عنوان ورودی گرفته و با اســتخراج مقادیر x و y، آن را در قالب بهتری نســبت به دستور print چاپ می‌کند. توابع Move Left و / Move Right هم نقطه را در جهت Xچپ و راست می‌کنند. توجه داشته باشید که تابع تعریف شده هیچ اطلاعی از نوع یــا مقداری که در آینده به آن ارجاع خواهد شــد ندارد و فعلاً برنامه‌نویس باید کنترل کند که آیا مقدار پاس شده با کارکرد تابع همخوانی دارد یا خیر. می‌توانیــد برای امتحــان یک مقدار عددی را به تابع پــاس کنید و نتیجه را ببینید. همچنین در این قطعه کد توابعی تعریف شده‌اند که نقطه را به اندازه مشخصی به چپ یا راست می‌برند و نتیجه عملیات آن‌ها توسط خطوط .... چاپ خواهد شد.

کمی پیشرفته‌تر

اما همان‌گونه که از ابتدا گفتیم، یکی از مهم‌ترین قابلیت‌های شیءگرایی، انتقال توابع و محاســبات و اطلاعات به داخل خود شــیء است. در واقع ما می‌توانیم توابع تعریف شــده در فهرســت ۲ را به داخل خود کلاس Point منتقــل کنیم تا نیازی به پاس کردن نقطه‌ها به توابع مختلف وجود نداشــته باشــد. فهرســت ۳ را وارد کرده و با نــام Points03.py ذخیره کنید. نتیجه اجرای این کد در شکل ۴ آورده شده است.

```
1  class Point:
2          def __init__(self,input_x,input_y):
3                  self.x = input_x
4                  self.y = input_y
5          def PrintPoint(self):
6                  text=”Point at %d,%d” %(self.x,self.y)
7                  print text
8          def MoveLeft(self,how_much):
9                  self.x = self.x - how_much
10         def MoveRight(self,how_much):
11                 self.x = self.x + how_much
12 #### Instantiation
13 p1= Point(6,13)
14 p2= Point(9,11)
15 ####
16 p1.PrintPoint()
17 p2.PrintPoint()
18 #### Moving points
19 p1.MoveLeft(3)
20 p2.MoveRight(10)
21 #### Printing results
22 print
23 print “P1 after moving . . . “
24 p1.PrintPoint()
25 print “P2 after moving . . . “
26 p2.PrintPoint()
```

فهرست ۳ تعریف متدها در یک کلاس

در این تعریف جدید از کلاس Point توابع مورد نیاز برای دســتکاری یــک نقطــه به داخــل خــود کلاس منتقل شــده‌اند. در خط 2 شــما یکی از اساســی‌ترین توابع یک کلاس یعنی __init__ را مشــاهده می‌کنید. هربار که نمونه جدیدی از یک کلاس ســاخته می‌شــود، این تابع اجرا خواهد شد. اگر نمونه‌های شــما در همان هنگام ســاخت احتیاج بــه مقداردهی اولیه یا تنظیم برخی ویژگی‌ها داشــته باشــند، محل انجام تمام ایــن عملیات بدنه تابع __init__ اســت. به خاطر داشــته باشــید که معرفــی آرگومان‌های تابــع __init__ (بــه جز self) در هنــگام ایجاد یک نمونــه جدید از کلاس الزامی است. (خطوط 13 و 14 را ببینید)

با این سیستم انتقال توابع به داخل کلاس، خطوط 13 تا 19 از فهرست ۲ به سادگی به خطوطی شبیه 13 و 14 از فهرست ۳ تبدیل می‌شود. نکته‌ای که به یقین توجه شــما را جلب کرده، آرگومانی به نام self است. این آرگومان همواره به خود شیء اشاره می‌کند. اگرچه این آرگومان باید در تمام توابع موجود در کلاس آورده شود، اما هنگام فراخوانی این توابع، به وارد کردن آن نیازی نیســت و پایتون به‌صورت خودکار، خود شیء را به تابع ارسال می‌کنــد. از کلمه self در تمام قســمت‌های تعریف یک کلاس برای ارجاع به خود شیء استفاده می‌شود.

```
>>>
Point at 6,13
Point at 9,11

P1 after moving . . .
Point at 3,13
P2 after moving . . .
Point at 19,11
>>>
```

شکل ۴ خروجی برنامه فهرست ۳

```
>>>
r1 is:
Rectangle corner is Point at 2,3
Width = 10       Height = 12
r2 is:
Rectangle corner is Point at 2,3
Width = 10       Height = 12

r2 changed to
Rectangle corner is Point at 2,3
Width = 40       Height = 12

r1 changed too !!
Rectangle corner is Point at 2,3
Width = 40       Height = 12
>>>
```

شکل ۵ خروجی برنامه فهرست ۵

تابع خاص دیگری که در این کد وجود دارد __str__ است که تنها آرگومان آن هم خود شیء، یعنی self است. همان‌گونه که در ابتدای بحث اشاره شد، پایتون زبانی کاملاً شیء گراست و تمام اجزای آن در حقیقت شیء هستند. هنگامی که شما شیء، متغیر یا عبارتی را در برابر کلمه کلیدی print قرار می‌دهید، پایتون در واقع تنها متد __str__ آن شیء را فراخوانی می‌کند. بنابراین، به جای تعریف یک تابع مستقل PrintPoint با تعریف __str__ می‌توانیم رفتار دستور print را برای شیء موردنظرمان یعنی نقطه، به دلخواه تغییر دهیم.

به همین ترتیب، ما در انتهای کد به سادگی از کلمه کلیدی print استفاده کرده‌ایم که درواقع مقدار برگشتی تابع __str__ را نمایش خواهد داد. سایر توابع با همان فرمت قبل و فقط با افزوده شدن یک آرگومان self مورد استفاده قرار گرفته‌اند. به خاطر داشته باشید که پس از این کار، این توابع متعلق به شیء نقطه هستند و برای فراخوانی آن‌ها باید از روش نشانه‌گذاری نقطه‌ای استفاده کرد.

بار اضافی

برای انجام عملیات ریاضی نظیر جمع و ضرب هم می‌توان مانند مثال قبلی توابعی جداگانه ایجاد کرد یا برای ساده‌تر شدن کار با اشیا می‌توان عملگرهای معمول ریاضی را به گونه‌ای تغییر داد که با اشیا نیز کار کنند. به عنوان مثال، اگر بخواهید عمل جمع بین دو شیء نقطه را برای پایتون تعریف کنید، کافی است تابعی با نام __add__ را مانند فهرست ۴ در تعریف کلاس آن بگنجانید.

در این‌صورت هنگامی که پایتون با عبارتی نظیر p1+p2 روبه‌رو شود که در آن p1 و p2 هردو از جنس نقطه باشند، تنها متد p1.__add__(p2) را اجرا خواهد کرد. این کار را باردهی اضافی عملگر یا Operator Overloading می‌نامند. برای باردهی اضافی تفریق، ضرب و تقسیم باید به ترتیب توابعی با نام‌های __mul__، __sub__ و __div__ را در کلاس موردنظر پیاده کنیم. برای بررسی تساوی یا بزرگ‌تر و کوچک‌تر بودن هم باید تابعی با نام __cmp__ را تعریف و تنظیم کرد.

```
1  def __add__(self,Second_Point):
2          result_x = self.x + Second_Point.x
3          result_y = self.y + Second_Point.y
4          return Point(result_x,result_y)
```

فهرست ۴ بار دهی اضافی عملگر جمع

اشیای مرکب

اشیا می‌توانند در ترکیب با یکدیگر به ساخت اشیا جدید و پیچیده‌تر کمک کنند. به عنوان مثال، اگر مدل‌سازی یک مستطیل مدنظر باشد، برای تعیین مشخصات آن می‌توانیم از نقطه گوشه و طول اضلاع آن استفاده کنیم. در این‌صورت کافی است کلاس Rectangle را ایجاد کنیم و برای نگه‌داری اطلاعات مربوط به گوشه مستطیل از یک شیء نقطه استفاده کنیم. کد فهرست ۵ را تایپ، با نام Rectangle.py ذخیره و اجرا کنید. نتیجه خروجی باید همانند شکل ۵ باشد. همان‌گونه که در خط 11 مشاهده می‌کنید، برای نگه‌داری نقطه گوشه مستطیل از یک شیء نقطه استفاده شده است. همین‌طور به خط 15 نیز توجه کنید که برای ساخت خروجی __str__ مستطیل تابع __str__ مربوط به نقطه گوشه هم فراخوانده شده است.

```
1  class Point:
2          def __init__(self,input_x,input_y):
3                  self.x = input_x
4                  self.y = input_y
5          def __str__(self):
6                  text=»Point at %d,%d» %(self.x,self.y)
7                  return text
8
9  class Rectangle:
10         def __init__(self,x,y,w,h):
11                 self.corner = Point(x,y)
12                 self.width = w
13                 self.height = h
14         def __str__(self):
15                 text = “Rectangle corner is %s” %self.
16 corner.__str__()
17                 text += “\nWidth = %d \t Height = %d” %(self.
18 width , self.height)
19                 return text
20 r1= Rectangle(2,3,10,12)
21 print “r1 is:\n” , r1
22 r2=r1
23 print “r2 is: \n” , r2
24 r2.width = 40
25 print “\nr2 changed to \n” , r2
26 print “\nr1 changed too !! \n” , r1
```

فهرست ۵ تعریف شیء مستطیل (توجه کنید که خطوط ۱۵، ۱۶ و ۱۷، ۱۸ باید در یک سطر تایپ شود)

نکته‌ای که همیشه باید درباره اشیا به یاد داشته باشید این است که استفاده از علامت مساوی برای انتساب یک شیء به شیء دیگر، درواقع به محل آن شیء در حافظه ارجاع می‌دهد.

به عبارت دیگر، در خط شماره 22 r2 مستطیلی جداگانه نیست، بلکه دقیقاً به همان محل نگه‌داری r1 در حافظه اشاره می‌کند و بنابراین، هر تغییری در r2 یا r1 (همانند خط شماره 24) به هردوی آن‌ها اعمال می‌شود. برای تهیه کپی‌های مجزا از اشیا، باید از ماجولی به نام copy استفاده کنیم.

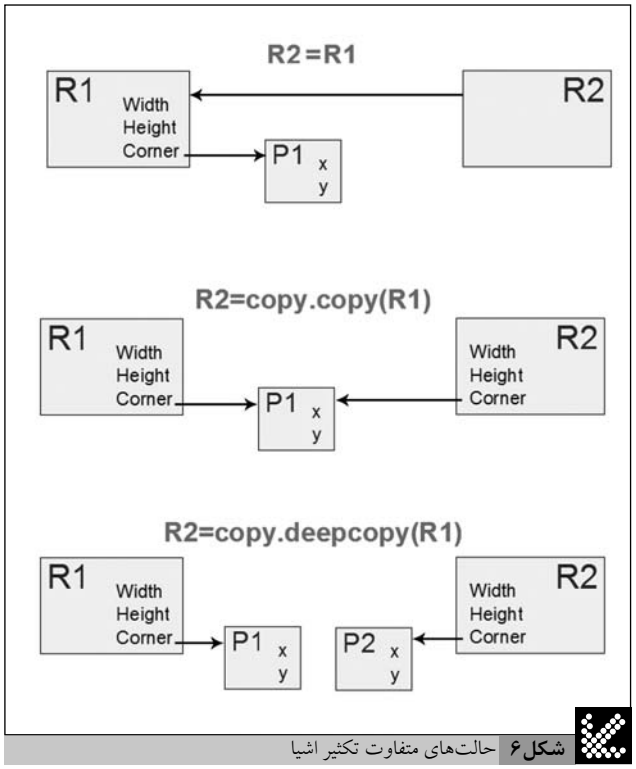
این ماجول دو متد بسیار پرکاربرد دارد؛ یکی متد copy و دیگری متد deepcopy. تفاوت این دو متد در هنگام برخورد با اشیا جاسازی شده در یک شیء مرکب مشخص می‌شود. به شکل ۶ توجه کنید.

این دیاگرام وضعیت r1 و r2 را در حالت‌های مختلف نشان می‌دهد. همان‌گونه که مشاهده می‌کنید با استفاده از عبارت r2=r1 به طور عملی شیء جدیدی ایجاد نخواهد شد، بلکه شیء r2 تنها به محل r1 در حافظه اشاره خواهد کرد. در صورت استفاده از متد copy.copy() اگرچه r2 یک شیء جدید خواهد شد، اما همانند r1 برای نگه‌داری نقطه گوشه همچنان به محل شیء p1 در حافظه اشاره خواهد کرد. یعنی اشیا داخل شیء اصلی کپی نمی‌شوند. بنابراین، تغییر گوشه مستطیل r2 باعث تغییر r1 هم خواهد شد و برعکس.

اما استفاده از متد copy.deepcopy() نه تنها خود شیء که تمام ارجاعات و اشیا موجود در آن را نیز به‌صورت مستقل کپی خواهد کرد.

```
1  class Point:
2          def __init__(self,input_x,input_y):
3                  self.x = input_x
4                  self.y = input_y
5          def __str__(self):
6                  text="Point at %d,%d" %(self.x,self.y)
7                  return text
8          def MoveLeft(self,how_much):
9                  self.x = self.x - how_much
10         def MoveRight(self,how_much):
11                 self.x = self.x + how_much
12 class NewPoint(Point):
13         def MoveUp(self,how_much):
14                 self.y = self.y + how_much
15
16 p=NewPoint(5,6)
17 print p ,"\n"
18
19 p.MoveLeft(2)
20 print p ,"<-- Moved left\n"
21
22 p.MoveUp(5)
23 print p ,''<-- Moved Up\n"
```

فهرست ۶ ارث بری یک کلاس جدید از کلاس موجود



شکل ۶ حالت‌های متفاوت تکثیر اشیا

```
>>>
Point at 5,6

Point at 3,6 <-- Moved left

Point at 3,11 <-- Moved Up

>>>
```

شکل ۷ خروجی فهرست ۶

ارث‌بری خصوصیات

هر شیء در پایتون می‌تواند والد شیء دیگری باشد و تمام یا برخی خصوصیات خود را برای این شیء جدید به ارث بگذارد. برای درک این موضوع کد فهرست ۶ را تایپ کرده و با نام Points05.py ذخیره و اجرا کنید. خروجی این قطعه کد باید همانند شکل ۷ باشد.

توجه کنید که همه اشیا، حتی آن‌هایی که خود شما به‌صورت مستقل ایجاد می‌کنید، در نهایت فرزند یک شیء مادر به نام object هستند و خصوصیاتی نظیر __doc__ و __module__ را از آن به ارث می‌برند. در واقع object والد تمام اشیا پایتون است. همان‌گونه که در خط 12 مشاهده می‌کنید، برای مشخص کردن والد یک کلاس جدید کافی است نام کلاس والد را پس از نام کلاس جدید در یک پرانتز ذکر کنید. در این مثال کلاس NewPoint تمام خصوصیات و متدهای کلاس Point (همانند MoveLeft) را به ارث برده است. این کلاس متد جدیدی (MoveUp) دارد که والد آن، یعنی Point فاقد آن است. اگر قصد تغییر یکی از متدهای کلاس والد را در این کلاس جدید دارید، کافی است آن متد را بازنویسی کنید.

در قسمت بعدی به کمک سیستم شیء‌گرا و نرم‌افزار Boa Constructor به سراغ ساخت برنامه‌های گرافیکی خواهیم رفت.