یادداشت‌های درسی (۴)

اگر انسانی کتاب بهتری بنویسد،
یا خطابه‌ای بهتر از دیگران در پند و موعظت ایراد کند،
یا حتی تله موشی بهتر از همسایگانش بسازد،
چنین کسی، حتی اگر خانه‌اش را در میان جنگل بنا کند،
مردم راهی هموار و استوار تا در خانه او خواهند کشید.

رالف والدو اِمِرسن (۱۸۸۲-۱۸۰۳)

مثال ۱: ارث‌بری چندگانه (Multiple Inheritance)

رابطه ارث‌بری چندگانه در زبان پایتون، وقتی معنا پیدا می‌کند که تعداد کلاس‌های پایه یا تعداد کلاس‌های مشتق، بیشتر از یک باشد. این رابطه به طرق مختلفی قابل تعریف است. یک گونه از رابطه ارث‌بری چندگانه، این است که یک کلاس از چند کلاس ارث ببرد؛ یعنی چند کلاس پایه وجود داشته باشد و یک کلاس مشتق.

```
1.  class Pet:
2.      def __init__(self, breed):
3.          self.breed = breed
4.      def about(self):
5.          print(f"This is {self.breed} breed")
6.  class Insurable:
7.      def __init__(self, amount):
8.          self.amount = amount
9.      def about(self):
10.         print(f"Its insured for an amount of {self.amount}")
11. class Cat(Pet, Insurable):
12.     def __init__(self, weight, breed, amount):
13.         self.weight = weight
14.         Pet.__init__(self, breed)
15.         Insurable.__init__(self, amount)
16.     def get_weight(self):
17.         print(f"{self.breed} Cat weighs around {self.weight} pounds")
18. def main():
19.     cat_obj = Cat(15, "Ragdoll", "$100")
20.     cat_obj.about()
21.     cat_obj.get_weight()
22. if __name__ == "__main__":
23.     main()
```

- از خط ۱ تا خط ۵ ، کلاس پایه Pet (که اشیاء عضو آن حیوانات خانگی هستند) تعریف شده است و از خط ۶ تا خط ۱۰ ، کلاس پایه Insurable (که اشیاء عضو آن حیوانات بیمه شدنی هستند) تعریف شده است.
- از خط ۱۱ تا خط ۱۷ ، کلاس مشتق Cat (که اشیاء عضو آن گربه‌ها هستند) تعریف شده است که از هر دو کلاس پایه Pet و Insurable ارث می‌برد. اشیاء عضو این کلاس، صفت breed را از کلاس پایه Pet و صفت amount را از کلاس Insurable ارث می‌برند. علاوه بر این دو صفت، صفت weight نیز برای اشیاء عضو کلاس مشتق در نظر گرفته شده است.
- برای آنکه در زمان ایجاد یک شیء از کلاس مشتق، بتوان مقادیر هر صفت آن را تعیین کرد، لازم است که در بدنه تابع سازنده کلاس مشتق، توابع سازنده دو کلاس پایه فراخوانی شوند. توابع سازنده کلاس‌های Pet و Insurable در خطوط ۱۴ و ۱۵ فرا خوانی شده‌اند. (اثر فراخوانی این دو تابع، مانند آن است که خطوط ۳ و ۸ در بدنه تابع سازنده کلاس مشتق Cat آورده شده باشند.) برای فراخوانی هر یک از توابع سازنده کلاس‌های پایه در بدنه تابع سازنده کلاس مشتق، ابتدا نام کلاس پایه و سپس عملگر . (نقطه) و سپس امضای تابع سازنده کلاس پایه آورده می‌شود (خطوط ۱۴ و ۱۵).
- در خط ۱۹ ، شیءای از کلاس مشتق Cat ایجاد شده است به نام cat_obj . و از طریق آن، تابع ()about در خط ۲۰ و سپس تابع ()get_weight در خط ۲۱ فرا خوانی شده‌اند.
- تابع ()about در هر دو کلاس پایه Pet و Insurable تعریف شده است. طبق قواعدی (که برای انتخاب تابع در چنین مواردی در نظر گرفته شده است) تابعی که در خط ۲۰ فراخوانی خواهد شد، تابعی است که در کلاس پایه Pet تعریف شده است.

مثال ۲: محاسبه طول کمانی از یک دایره

هدف، نوشتن برنامه‌ای شیءگرا است که طول کمانی از یک دایره را محاسبه کند.

```
1.  import math
2.  class ArcLength:
3.      def __init__(self):
4.          self.radius = 0
5.          self.angle = 0
6.      def calculate_arc_length(self):
7.          result = 2 * math.pi * self.radius * self.angle / 360
8.          print(f"Length of an Arc is {result}")
9.  al = ArcLength()
10. al.radius = 9
11. al.angle = 75
12. print(f"Angle is {al.angle}")
13. print(f"Radius is {al.radius}")
14. al.calculate_arc_length()
```

- در خط ۱ ، ماژول math وارد برنامه شده است.
- از خط ۲ تا خط ۸ ، کلاس ArcLength تعریف شده است. تابع سازنده این کلاس، بدون پارامتر ورودی است. اما در بدنه آن، دو صفت radius و angle برای اشیاء عضو کلاس (کمان‌های دایره) تعریف شده‌اند و مقدار اولیه صفر برای آنها تعیین شده است.

- به تابع سازنده‌ای که امضای آن (self)__init__ باشد، default constructor گفته می‌شود؛ چون هیچ پارامتر ورودی ندارد (self پارامتر پیش‌فرض همه توابع است). به تابع سازنده مثال‌های ۳ و ۴ که علاوه بر پارامتر پیش‌فرض self ، پارامتر ورودی دیگری نیز دارند، parameterized constructor گفته می‌شود.
- از خط ۶ تا خط ۸ ، تابع ()calculate_arc_length تعریف شده است که با استفاده از مقدار دو صفت radius و angle یک شیء (یک کمان دایره) ، طول آن را محاسبه و چاپ می‌کند.
- در خط ۹ ، شیءای از کلاس ArcLength ایجاد شده است به نام al . و در خطوط ۱۰ و ۱۱ ، مقادیر ۹ و ۷۵ برای دو صفت آن، یعنی شعاع و زاویه تعیین شده است.
- دو دستور ()print در خطوط ۱۲ و ۱۳ ، مقدار زاویه و شعاع کمان را چاپ می‌کنند.
- در خط ۱۴ ، تابع ()calculate_arc_length فرا خوانی شده است تا طول کمان al را محاسبه کند.

مثال ۳: تحقیق هم‌خط بودن سه نقطه

هدف، نوشتن برنامه‌ای شیءگرا است که با آن بتوان هم‌خط (روی یک خط) بودن یا نبودن هر سه نقطه‌ای در صفحه را تحقیق کرد.

سه نقطه (x_1, y_1) ، (x_2, y_2) و (x_3, y_3) روی یک خط قرار دارند اگر شیب خطی که از دو نقطه (x_1, y_1) و (x_2, y_2) می‌گذرد، با شیب خطی که از دو نقطه (x_2, y_2) و (x_3, y_3) می‌گذرد، برابر باشد:

$$\frac{y_3 - y_2}{x_3 - x_2} = \frac{y_2 - y_1}{x_2 - x_1}$$

```
1.  class Collinear:
2.      def __init__(self, x, y):
3.          self.x_coord = x
4.          self.y_coord = y
5.      def check_for_collinear(self, point_2_obj, point_3_obj):
6.          if (point_3_obj.y_coord - point_2_obj.y_coord)*(point_2_obj.x_coord - self.x_coord)\
              == (point_2_obj.y_coord - self.y_coord)\
              *(point_3_obj.x_coord - point_2_obj.x_coord):
7.              print("Points are Collinear")
8.          else:
9.              print("Points are not Collinear")
10. def main():
11.     point_1 = Collinear(1, 5)
12.     point_2 = Collinear(2, 5)
13.     point_3 = Collinear(4, 6)
14.     point_1.check_for_collinear(point_2, point_3)
15. if __name__ == "__main__":
16.     main()
```

- از خط ۱ تا خط ۹ ، کلاس Collinear تعریف شده است. در خطوط ۳ و ۴ (بدنه تابع سازنده) ، دو صفت x_coord و y_coord برای اشیاء عضو کلاس (نقاط) تعریف شده است.

- از خط ۵ تا خط ۹ ، تابع check_for_collinear() تعریف شده است. این تابع، سه شیء (نقطه) را به عنوان ورودی می‌گیرد و از روی مقادیر صفات x_coord و y_coord آنها، هم‌خط بودن یا نبودن سه نقطه را مشخص می‌کند. (دستور if در خط ۶ ، به علت طولانی بودن، با استفاده از نماد \ به سه خط کوچک‌تر شکسته شده است.)
- در خطوط ۱۱ و ۱۲ و ۱۳ ، سه شیء از کلاس Collinear ایجاد شده‌اند. و در خط ۱۴ ، از این سه شیء برای فراخوانی تابع check_for_collinear() استفاده شده است. شیء point_1 ، قبل از عملگر . (نقطه) قرار گرفته است و تابع را فراخوانی کرده است و دو شیء point_2 و point_3 به عنوان ورودی به تابع داده شده‌اند. در واقع، شیء point_1 متناظر است با پارامتر self تابع (در عمل، شیء point_1 ، به عنوان ورودی واقعی اول تابع، جایگزین پارامتر اسمی self خواهد شد). و دو شیء point_2 و point_3 ، به عنوان ورودی‌های واقعی دوم و سوم تابع، جایگزین پارامترهای اسمی point_2_obj و point_3_obj خواهند شد.

مثال ۴

این برنامه، چه مسأله‌ای را حل می‌کند؟ خروجی برنامه چیست؟

```
1.  class Time:
2.      def __init__(self, hours, minutes, seconds):
3.          self.hours = hours
4.          self.minutes = minutes
5.          self.seconds = seconds
6.      def add_time(self, duration):
7.          opera_hours = self.hours + duration.hours
8.          opera_minutes = self.minutes + duration.minutes
9.          opera_seconds = self.seconds + duration.seconds
10.         while opera_seconds >= 60:
11.             opera_seconds = opera_seconds - 60
12.             opera_minutes = opera_minutes + 1
13.         while opera_minutes >= 60:
14.             opera_minutes = opera_minutes - 60
15.             opera_hours = opera_hours + 1
16.         print(f"Opera ends at {opera_hours}:{opera_minutes}:{opera_seconds}")
17. def main():
18.     opera_start = Time(10, 30, 30)
19.     opera_duration = Time(2, 45, 50)
20.     opera_start.add_time(opera_duration)
21. if __name__ == "__main__":
22. main()
```