고급통계프로그래밍 Solution 7

컴퓨터과학부 2017920054 이호준

과제 index는 번역본 기준입니다.

Ex 15.1

Solution

```
import math

class Point():
    def __init__(self, x, y):
        self.x = x
        self.y = y

def distance_between_points(point_a, point_b):
    return math.sqrt((point_a.x - point_b.x)**2 + (point_a.y - point_b.y)**2)
```

- Point Class를 선언하고, 생성자 (__init__)을 이용해서 x와 y를 객체 선언과 동시에 입력받을 수 있게 한다.
- 함수 distance_between_points를 선언한다.
  - 두 점을 나타내는 point_a, point_b를 인자로 받는다.
  - 두 점 사이의 거리 구하는 공식 - $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ - 를 제곱연산자 `**`와 제곱근 함수 `math.sqrt()`를 이용해서 구현하고, 이를 반환한다.

Result

```
# Test 1
a = Point(1, 1)
b = Point(2, 2)

print(distance_between_points(a, b))

# in Stdout 1
1.4142135623730951

# Test 2
a = Point(3, 4)
b = Point(6, 8)
```

```
# in Stdout 2
5.0
```

Ex 15.2

Solution

```
class Rectangle():
    def set_width_height(self, w, h):
        self.width = w
        self.height = h

    def set_ld_point(self, x, y):
        self.ld_corner = Point(x, y)

def move_rectangle(rect, dx, dy):
    rect.ld_corner.x += dx
    rect.ld_corner.y += dy
```

- Rectangle Class를 선언하고, `set_width_height()` 함수와 `` `set_ld_point()` 함수를 이용해서 각각 width, height / ld_corner를 설정해준다.
- `move_rectangle()` 함수를 선언하였다.
  - 인자로 rect(Rectangle형 객체), dx, dy(number)를 받고, rect 객체 내부의 ld_corner 객체의 x와 y를 shift 해준다.

Result

```
# Test - (3, 4) 를 (+17, +16) 평행이동
rect = Rectangle()
rect.set_width_height(100, 100)
rect.set_ld_point(3, 4)
move_rectangle(rect, 17, 16)

print(rect.ld_corner.x, rect.ld_corner.y)

# in Stdout:
20 20
```

Ex 15.3

Solution

```
import copy

...

def move_rectangle(rect, dx, dy):
    new_rect = copy.deepcopy(rect)
    new_rect.ld_corner.x += dx
    new_rect.ld_corner.y += dy
    return new_rect
```

- 기본적인 구조는 #Ex 15.2와 유사하다.
- new_rect 객체를 하나 만들고, rect의 깊은 복사본을 만든다.
    - 얕은 복사와 다른 점은, 깊은 복사의 경우 객체 안에있는 객체 ld_corner 또한 복사된다. (얕은 복사는 이를 복사하지 않고 aliasing이 일어남)
- 이 new_rect에 기존에 했던 로직을 적용하고, 이 객체를 반환하는 형태로 구현하였다.

Result

```
# Test
rect = Rectangle()
rect.set_width_height(100, 100)
rect.set_ld_point(3, 4)

shifted_rect = move_rectangle(rect, 17, 16)

print(rect.ld_corner.x, rect.ld_corner.y)
print(shifted_rect.ld_corner.x, shifted_rect.ld_corner.y)

# in Stdout:
3 4
20 20
```

- 기존 rect의 ld_corner의 x, y는 (3, 4)로 그대로이다.
- shift_rect의 ld_corner의 x, y는 (20, 20)으로 변경되었다. 이를 통해깊은 복사가 일어났음을 확인할 수 있다.

Ex 16.6

Solution

```
class Time():
    def __init__(self, h, m, s):
        self.hour = h
        self.minute = m
        self.second = s
```

- `Time` Class를 만든다.
    - field에는 hour, minute, second가 있으며 이는 생성자에 의해 초기화된다.

```
def mul_time(time, num):
    mul = Time(time.hour, time.minute, time.second)
    mul.hour *= num
    mul.minute *= num
    mul.second *= num

    if mul.second >= 60:
        mul.minute += (mul.second // 60)
        mul.second -= (mul.second // 60) * 60

    if mul.minute >= 60:
        mul.hour += (mul.minute // 60)
        mul.minute -= (mul.minute // 60) * 60

    if mul.hour > 24:
        mul.hour %= 24

    return mul
```

- `mul_time()` 함수를 생성한다.
    - Time 객체 mul을 선언한다.
    - muld의 세 attribute에 각각 num을 곱한다.
    - mul.second가 60 이상인 경우, 초과분만큼 minute에 반영하고 mul.second를 그만큼 뺀다.
    - mul.minute가 60 이상인 경우, 초과분만큼 hour에 반영하고 mul.minute를 그만큼 뺀다.
    - mul.hour가 24를 넘어가는 경우 이를 modulo 연산으로 바꿔서 치환한다.
      ex) 25시 : 25%24 == 1시
    - 최종적으로 연 mul 객체를 반환한다.

Result

```
# Test 1
now_time = Time(4, 40, 0)
next_time = mul_time(now_time, 2)

print(next_time.hour, next_time.minute, next_time.second)

# in Stdout 1:
9 20 0
```

```
# Test 2
now_time = Time(1, 10, 0)
next_time = mul_time(now_time, 100)

print(next_time.hour, next_time.minute, next_time.second)

# in Stdout 2:
20 40 0
```

Ex 16.7

Solution 1

```
from datetime import date, datetime

num_to_weekday = {0:"Monday", 1:"Tuesday", 2:"Wednesday", 3:"Thursday",
                  4:"Friday", 5:"Saturday", 6:"Sunday"}

# Sol (1)
print(num_to_weekday[datetime.now().weekday()])
```

- `.now()` : 현재 시간대(UTC +9)의 현재 시간을 돌려준다.
- `.weekday()` : 해당 시간의 요일을 0~6으로 encode하여 돌려준다. 월요일은 0, 일요일은 6이다.
- 위 2 method를 활용해서 현재 시각의 요일에 해당하는 숫자를 구하고, 0~6 → 해당하는 요일 string의 관계를 갖는 `num_to_weekday` 사전형을 선언하여 결과적으로는 요일 문자열을 출력하도록 하였다.

Result

```
# Test
print(datetime.now())
print(datetime.now().weekday())
```

```
print(num_to_weekday[datetime.now().weekday()])

# in Stdout:
2020-11-04 16:48:15.650737
2
Wednesday
```

Solution 2

```
# Sol (2)
birthday = [int(i) for i in input("생일을 입력해주세요. ex) 1988-02-10\n").split("-")]

next_birthday = datetime(datetime.now().year, birthday[1], birthday[2])

if (next_birthday - datetime.now()).total_seconds() < 0:
    next_birthday = datetime(datetime.now().year + 1, birthday[1], birthday[2])

print(next_birthday - datetime.now())
```

- 변수 `birthday`에 "1988-02-10" 형태로 입력을 받고, '-'를 기준으로 parsing하여 int형으로 각 원소를 저장한다.
  - 인덱스 0에는 년, 1에는 월, 2에는 일이 담기게 된다.
- 변수 `next_birthday` 에 올해의 년도와 생일의 월, 일을 담는다.
  - 만약 올해의 생일과 현재 시간의 차이가 음수라면: (즉, 올해의 생일이 이미 지났다면)
    - next_birthday를 내년을 기준으로 재설정
- 위와같이 next_birthday를 보정한 후, next_birthday와 현재 시간의 차이를 출력.

Result

```
# Test 1
생일을 입력해주세요. ex) 1988-02-10
1998-12-3

# in Stdout 1:
28 days, 6:31:43.139069

# Test 2
생일을 입력해주세요. ex) 1988-02-10
1998-11-3

# in Stdout 2:
363 days, 6:31:38.973379
```

- (datetime.now()는 11월 4일입니다.)
- 첫 번째 테스트에서는 아직 올해에 나타나지 않은 생일이므로 올해의 생일과 현재 날짜의 차이를 반환, 출력합니다.
- 두 번째 테스트에서는 올해에 이미 지난 생일이므로 내년의 생일과 현재 날짜의 차이를 반환, 출력합니다.