Computing for Data Science

HW #6

제출 기한: 2021.03.29 오전 10:59

다음 페이지부터 문제가 주어집니다.

주의사항

- 코드를 Jupyter Notebook 에서 작성하였더라도 python 파일(.py)로 제출할 것.
- 함수가 의도한 값을 Return 하는지를 확인. (Print 와 혼동하지 말 것)
- 파일명은 P1.py ~ P4.py 를 유지하고, 해당파일들을 HW6_학번_이름.zip 으로 압축하여 제출할 것. 예를 들면 학번이 2020-12345 이고, 이름이 Keondo Park 이라면 **HW6_2020_12345_KeondoPark.zip** 으로 압축하여 제출.
- 예시로 제시한 입력값 외에도 조교가 랜덤으로 생성한 입력값으로 코드가 잘 작성되었는지 테스트할 것이다.
- 채점은 프로그램에 의해 기계적으로 처리되므로 위 사항을 지키지 않은 경우 누락되거나 불이익을 받을 수 있음.
- **늦은 제출은 받지 않음**.
- **표절 검사를 수행하여 발각될 경우 성적 F 부여**.

모든 문제에 해당하는 사항: 뼈대 코드의 class 이름, method 이름 및 parameter 는 수정하지 말 것.

P1.

In this exercise, you will implement class Country, which represents a country with a name, a population, and an area.

a. Here is a sample interaction from the Python shell:

```
>>> canada = Country('Canada', 34482779, 9984670)
>>> canada.name
'Canada'
>>> canada.population
34482779
>>> canada.area
9984670
```

Implement Country with a constructor (method __init__) that has three parameters: its name, its population, and its area.

b. Consider this code:

```
>>> canada = Country('Canada', 34482779, 9984670)
>>> usa = Country('United States of America', 313914040, 9826675)
>>> canada.is_larger(usa)
True
```

In class Country, define a method named is_larger that takes two Country objects and returns True if and only if the first has a larger area than the second.

c. Consider this code:

```
>>> canada.population_density()
3.4535722262227995
```

In class Country, define a method named population_density that returns the population density of the country (population per area). (10^{-3} 오차허용)

P2.

In this exercise, you will implement a Continent class, which represents a continent with a name and a list of countries. Class Continent will use class Country from the previous exercise.
(P1.py 와 P2.py 가 같은 경로에 있어야함)

a. Here is a sample interaction from the Python shell:

```
>>> canada = Country('Canada', 34482779, 9984670)
>>> usa = Country('United States of America', 313914040, 9826675)
>>> mexico = Country('Mexico', 112336538, 1943950)
>>> countries = [canada, usa, mexico]
>>> north_america = Continent('North America', countries)
>>> north_america.name
'North America'
```

Implement Continent with a constructor (method __init__) that has two parameters: its name, and its list of Country objects.

b. Consider this code:

```
>>> north_america.total_population()
460733357
```

In class Continent, define a method named total_population that returns the sum of the populations of the countries on this continent.

P3.

`Point` 클래스는 다음과 같이 x 좌표와 y 좌표를 입력 받는다.

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

`Shape` 클래스는 다음과 같이 Point 4 개를 입력 받는다.

```
class Shape:
    def __init__(self, p1, p2, p3, p4):
        self.p1 = p1
        self.p2 = p2
        self.p3 = p3
        self.p4 = p4

    def is_square(self): # 점 4 개가 정사각형을 이루는 지 판별

        # return: boolean
```

입력 받은 4 개의 점이 정사각형을 이루면 True 를, 그렇지 않으면 False 를 return 하는 `is_square` method 를 구현하시오.

- 점의 좌표는 모두 정수이다.
- 동일한 점일 수도 있다.
- 클래스 내부 혹은 외부에 다른 함수를 구현하고 이용해도 상관없다.

예시 1.

```
>>> p1 = Point(0,0)
>>> p2 = Point(1,1)
>>> p3 = Point(1,0)
>>> p4 = Point(0,1)
>>> s = Shape(p1, p2, p3, p4)
>>> s.is_square()
True
```

예시 2.

```
>>> p1 = Point(1, 1)
>>> p2 = Point(3, 1)
>>> p3 = Point(3, 3)
>>> p4 = Point(5, 1)
>>> s = Shape(p1, p2, p3, p4)
>>> s.is_square()
False
```

예시 3.

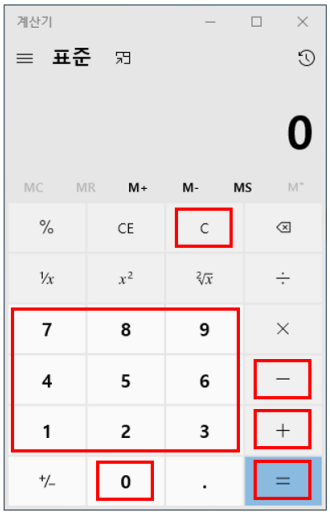
```
>>> p1 = Point(-1, 0)
>>> p2 = Point(1, 0)
>>> p3 = Point(0, 1)
>>> p4 = Point(0, -1)
>>> s = Shape(p1, p2, p3, p4)
>>> s.is_square()
True
```

예시 4.

```
>>> p1 = Point(-1, 0)
>>> p2 = Point(1, 0)
>>> p3 = Point(0, 1)
>>> p4 = Point(1, 0)
>>> s = Shape(p1, p2, p3, p4)
>>> s.is_square()
False
```

P4.

윈도우에 있는 기본 계산기처럼 동작을 하는 `Calculator` 클래스를 구현해야 한다. 단, 기능이 제한적이어서 몇 가지 버튼의 동작만 가능하다. 다음의 버튼을 구현할 것이다.



아래에 나와있는 설명대로 Calculator 클래스를 구현하면 된다. 윈도우 계산기를 떠올리며 설명을 보자. 아래에 나와있지 않은 동작에 대한 것은 구현할 필요도 없고, test case 에도 추가하지 않을 것이다.

```
def __init__(self):
```

- 초기 상태는 '0' 만 입력되어 있는 상태이다.
- 필요한 변수를 정의해서 사용하면 된다.

```
def digit(self, num): #계산기의 숫자 버튼
```

- parameter `num` 을 받는다. `num` 은 0 부터 9 까지의 정수.
- 동작은 숫자 `num` 버튼을 한 번 클릭하는 것이다. 만약 연속으로 클릭하게 된다면 차례대로 숫자가 입력된다. 예를 들어, 3 -> 5 -> 1 을 클릭하면 '351'이라는 숫자가 입력되는 것이다.
- 1 ~ 9 를 입력하기 전에 0 을 여러 번 클릭한 것은 0 을 한 번 클릭한 것과 같다.

```
def plus(self): # 계산기의 + 버튼
```

- 동작은 계산기의 + 버튼을 한 번 클릭하는 것이다.
- 연속으로 + 버튼 혹은 - 버튼을 누른다면, 최종적으로 입력되는 버튼은 가장 마지막에 입력한 연산 버튼이다.

```
def minus(self): # 계산기의 - 버튼
```

- 동작은 계산기의 - 버튼을 한 번 클릭하는 것이다.
- 연속으로 + 버튼 혹은 - 버튼을 누른다면, 최종적으로 입력되는 버튼은 가장 마지막에 입력한 연산 버튼이다.

```
def clear(self): # 계산기의 C 버튼
```

- 동작은 계산기의 C 버튼을 한 번 클릭하는 것이다.
- 클릭하면 초기상태 (0 만 입력된 상태)로 돌아간다.

```
def equal(self): # 계산기의 = 버튼
```

- 계산기의 = 버튼을 한 번 클릭하는 것이다.
- 현재까지의 입력을 바탕으로 연산 결과를 return 해야 한다.
- 숫자만 입력된 상태에서 = 버튼을 클릭하면 해당 숫자를 return 한다.
- 연산 버튼 다음에 바로 = 버튼을 입력하는 경우는 없다.
- 각각의 test case 에서 = 버튼은 딱 한 번 마지막에 클릭을 한다. 그 때의 return 값으로 채점을 한다.

예시 1.

```
c = Calculator()
c.digit(1)
c.digit(2)
c.plus()
c.minus()
c.digit(3)
c.digit(1)
c.digit(2)
c.plus()
c.minus()
c.minus()
c.plus()
c.digit(0)
c.digit(0)
c.digit(1)
c.digit(0)
c.digit(0)
c.equal()

-200
```

예시 2.

```
c = Calculator()
c.clear()
c.digit(9)
c.plus()
c.digit(1)
c.digit(2)
c.digit(5)
c.minus()
c.minus()
c.digit(3)
c.plus()
c.clear()
c.digit(5)
c.digit(2)
c.minus()
c.digit(5)
c.minus()
c.plus()
c.digit(2)
c.equal()

49
```