Computing Foundations for Data Science

HW 6

제출기한: 2022/03/29 14:00PM

주의사항

- 코드를 Jupyter Notebook 에서 작성하였더라도 python 파일(.py)로 제출할 것.
- 함수가 의도한 값을 return 하는지를 확인할 것. (print 와 혼동하지 말 것)
- 파일명은 P1.py ~ P4.py 를 유지하고, 해당 파일들을 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.
  - 압축 시 반드시 zip 으로 할 것. (egg, tar, gz, rar, 등은 0 점. 채점 대상 제외)
- 각 파일들은 문제를 해결하기 위한 함수만 있어야 하며 불필요한 출력이 있을 시 불이익을 받을 수 있음
  - 예시) P1.py 에는 def P1() 함수만 있어야함.
  - 테스트를 위한 P1() 함수 호출이 있을 시 불이익 이 있을 수 있음
  - 제출시에 print 문은 지워서 제출할 것
- 예시로 제시한 입력 값 외에도 조교가 임의로 생성한 입력 값으로도 코드가 잘 실행되는지 테스트할 예정.
- 뼈대 코드의 함수 이름 및 매개변수(parameter)는 변경하지 말 것.
- 채점은 프로그램에 의해 기계적으로 처리되므로 위 사항을 지키지 않은 경우 누락되거나 불이익을 받을 수 있음
- 문제의 instruction 이 불명확하거나 clarification 이 필요할 경우 slack 을 활용하여 질문할 것.
- **늦은 제출은 받지 않음**
- 표절 검사를 수행하여 발각될 경우 성적 F 부여

문제 1.

국가명(name), 인구 수(population), 면적(area)을 인자(parameter)로 받는 Country 클래스를 구현해 보아라.

a. __init__메소드를 사용하여 3개의 인자(name, population, area)를 가진 Country 클래스를 정의해 보아라

- 다음은 파이썬 쉘에서 실행하였을 때의 예시이다.

```
>>> from P1 import Country

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

b. Country 클래스의 메소드인 **is_larger**를 정의해 보아라.

- **is_larger** 메소드는 첫 번째 국가가 두 번째 국가에 비해 더 큰 면적을 지녔을 때에만 True를 반환하는 메소드이다. (면적이 같을 경우 False 반환)
- 다음은 파이썬 쉘에서 실행하였을 때의 예시이다.

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

c. Country 클래스의 메소드인 **population_density**를 정의해 보아라.

- **population_density** 메소드는 국가의 인구 밀도 (면적 당 인구 수)를 반환하는 메소드이다. (area가 0인 경우는 없다.)
- 다음은 파이썬 쉘에서 실행하였을 때의 예시이다.

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

문제 2.

대륙명(name), 국가 리스트(countries)를 인자로 받는 Continent 클래스를 구현해 보아라. Continent 클래스는 문제1에서 구현한 Country 클래스를 사용한다.
(P1.py 와 P2.py가 같은 경로에 있어야함)

a. __init__메소드를 사용하여 2개의 인자(name, countries)를 가진 Continent 클래스를 정의해 보아라.

- countries는 Country class의 객체들로 이루어진 list이다.
- countries list가 비어있는 경우는 고려하지 않는다.
- 다음은 파이썬 쉘에서 실행하였을 때의 예시이다.

```
>>> from P1 import Country
>>> from P2 import Continent

>>> 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'
```

b. Continent 클래스의 메소드인 **total_population**을 구현하여라

- **total_poulation** 메소드는 대륙에 속한 국가들의 인구의 합을 반환하는 메소드이다.
- 다음은 파이썬 쉘에서 실행하였을 때의 예시이다.

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

문제 3.

Point 클래스는 다음과 같이 x좌표와 y좌표를 입력 받는다.

```
class Point:
    def __init__(self,x,y):
        """
        (Point, int, int)->NoneType
        >>> p=Point(1,3)
        >>> p.x
        1
        >>> p.y
        3
        """
        self.x=x # x 좌표
        self.y=y # y 좌표
```

LineSegment 클래스는 다음과 같이 2개의 Point를 입력 받는다. 첫 번째 Point는 선분의 시작점이며, 두 번째 Point는 선분의 끝점이다.
*동일한 Point를 입력 받는 경우는 제외하는 것으로 하여 이 경우를 고려하지 않아도 된다.

```
class LineSegment:
    def __init__(self,point1,point2):
        """(LineSegment,Point,Point)->NoneType
        A new LineSegment connecting point1 to point2
        >>> p1=Point(1,3)
        >>> p2=Point(3,2)
        >>> segment=LineSegment(p1,p2)
        >>> segment.startpoint==p1
        True
        >>> segment.endpoint==p2
        True
        """
        self.startpoint=point1
        self.endpoint=point2
```

a. LineSegment 클래스의 메소드인 **slope** 메소드는 선분의 기울기를 반환하는 메소드이다. slope 메소드를 구현해 보아라.

- 선분이 y축과 평행한 경우의 slope는 고려하지 않아도 된다.

b. LineSegment 클래스의 메소드인 **length** 메소드는 선분의 길이를 반환하는 메소드이다. length 메소드를 구현해 보아라.

- 다른 모듈 import가 가능하다.

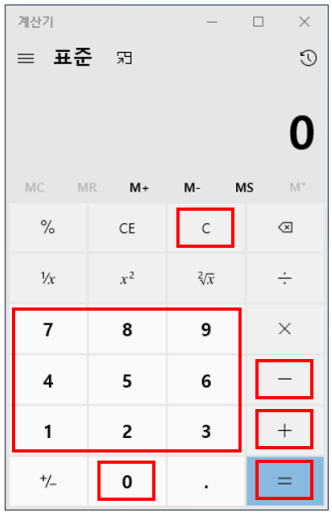
- 다음은 파이썬 쉘에서 실행하였을 때의 예시이다.

```
>>> from P3 import Point,LineSegment
```

```
>>> segment=LineSegment(Point(1,1),Point(3,2))
>>> segment.slope()
0.5
>>> segment.length()
2.23606797749979
```

문제 4.

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 case에도 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.
- 숫자(digit)을 입력하기 전에 -버튼을 누른다면, 숫자가 음수가 된다.

```
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.
- 추가 method 를 선언해 구현하는 것도 가능하다
- 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

예시 3.

```
c = Calculator()
c.equal()
```

0