5. 객체지향 설계

Total points 8/13

객체지향 프로그래밍 원리와 디자인 패턴은 어려운 내용이니 꼼꼼히 이해해봅시다.

✓ 객체지향 프로그래밍을 해야하는 이유로 틀린 것은? * 1/1

- ○ 우리가 기대하는 속성만 가진 객체를 만들어야 한다.
- ○ 데이터를 패키지화하고, 메서드를 제한할 수 있다.
- ○ 객체를 만들 때 유효성 검사를 할 수 있다.
- ◉ 위의 세 이유 모두 객체지향 프로그래밍을 해야 하는 이유이다. ✓
- ○ 모르겠다.

✓ 다음은 클래스, 객체, 인스턴스에 대한 설명이다. 틀린 설명은? * 1/1

- ○ 클래스(Class)는 사전에 정의된 특별한 데이터와 메서드의 집합이다.
- ○ 객체(object)는 클래스에 선언된 모양 그대로 생성된 실체이다.
- ○ 인스턴스(instance)는 객체가 소프트웨어에 실체화될 때(메모리에 할당되어 사용될 때) 이 실체를 의미한다.
- ◉ 인스턴스는 객체를 포함할 수 있으며, 포괄적인 의미를 지닌다. ✓
- ○ 모르겠다.

Feedback

객체는 인스턴스를 포함할 수 있으며, 포괄적인 의미를 지닌다.

https://cerulean85.tistory....

✓ 클래스 인스턴스 생성(Class instantiation)에 대한 설명으로 틀린 것은? * 1/1

- ○ 클래스 인스턴스 생성은 함수 표기법을 사용하여 초기 상태의 객체를 생성하는 일이다.
- ○ 여러 범위의 여러 이름을 같은 객체에 바인딩(binding) 또는 에일리어싱(aliasing)할 수 있다.
- ○ Hello라는 클래스가 있을 때, Hello()를 생성자(Constructor)라고 한다.
- ◉ 생성자를 호출하면 Class.__init__() 메서드가 객체를 초기화한 후, Class.__new__() 메서드가 객체를 할당한다. ✓
- ○ 모르겠다.

✓ 클래스 속성(attribute)에 대한 설명으로 틀린 것은? * 1/1

- ○ 데이터(data)와 메서드(method)로 이루어져 있다.
- ○ 메서드 속성의 첫 번째 인수는 호출된 인스턴스 자신이다.
- ○ "모듈명.함수명"과 같은 표현식에서 모듈명은 모듈 객체이고, 함수명은 객체의 속성 중 하나다.
- ◉ 모든 속성은 del문으로 삭제할 수 있다. ✓
- ○ 모르겠다.

✕ 네임스페이스와 스코프에 대한 설명을 이해하기 위한 문제이다. 옳은 출력값은? * 0/1

```
# namespace_example01.py
def outer_func():
    a = 20

    def inner_func():
        a = 30
        print("Namespace:", __name__)
    inner_func()

a = 10
outer_func()
print("Namespace:", __name__)

# namespace_example02.py
import namespace_example01

a = 10
print("Namespace:", __name__)

# on shell
$ python namespace_example02.py
```

```
Namespace: __main__
```

1

```
Namespace: namespace_example02
```

2

```
Namespace: namespace_example01
Namespace: namespace_example01
Namespace: __main__
```

```
Namespace: namespace_example01
Namespace: namespace_example01
Namespace: namespace_example02
```

3

4

```
Namespace: inner_func
Namespace: namespace_example01
Namespace: namespace_example02
```

```
Namespace: inner_func
Namespace: namespace_example01
Namespace: __main__
```

5

6

모르겠다.

Correct answer

3

Feedback

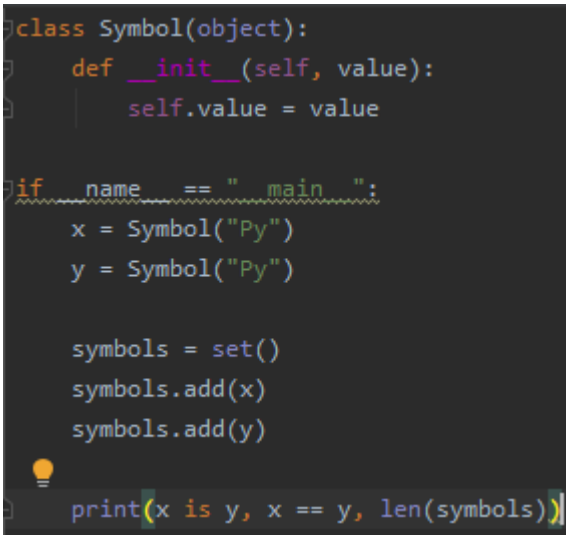
참고 링크

https://hcnoh.github.io/20...

✓ 다음은 객체지향 프로그래밍의 원리에 대한 문제이다. 틀린 설명을 고르시오. * 1/1

- ○ 특수화(specialization)는 슈퍼 클래스(부모 또는 베이스 클래스)의 모든 속성을 상속하여 새 클래스를 만드는 절차다.
- ○ 모든 메서드는 서브 클래스(자식 클래스)에서 재정의(override, 재구현)될 수 있다.
- ◉ 상속(inheritance)는 has-a 관계이다. ✓
- ○ 다형성(polymorphism)은 메서드가 서브 클래스 내에서 재정의될 수 있다는 원리다.
- ○ 합성(composition)은 A와 B가 강한 연관 관계를 맺으며, 의존성이 강하다. 예를 들어 집 클래스는 방 클래스를 갖는다.
- ○ 집합화(aggregation)은 A와 B가 연관 관계가 있지만, 생명주기가 약하며 독립적이다. 예를 들어 학생과 수강 과목 사이의 관계가 있다.
- ○ 파이썬에서 사용자 정의 클래스의 모든 객체는 기본적으로 해시 가능(hashable)하다.
- ○ 모르겠다.

✓ 다음은 객체의 해시에 대한 코드이다. 출력값으로 옳은 것은? * 1/1



```
class Symbol(object):
    def __init__(self, value):
        self.value = value

if __name__ == "__main__":
    x = Symbol("Py")
    y = Symbol("Py")

    symbols = set()
    symbols.add(x)
    symbols.add(y)

    print(x is y, x == y, len(symbols))
```

- (●) False False 2 ✓
- ( ) False True 2
- ( ) False True 1
- ( ) True False 2
- ( ) True False 1
- ( ) True True 1
- ( ) 모르겠다.

✕ 다음은 객체 지향 설계로 원 클래스를 구현한 것이다. 빈 칸을 채우시오. * 0/2

```python
import math

class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def distance_from_origin(self):
        return math.hypot(self.x, self.y)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __repr__(self):
        return "point ({0.x!r}, {0.y!r}".format(self)

    def __str__(self):
        return "({0.x!r}, {0.y!r})".format(self)

class Circle(Point):
    def __init__(self, radius, x=0, y=0):
        super().__init__(x, y)
        self.radius = radius

    def edge_distance_from_origin(self):
        return abs(self.distance_from_origin() - self.radius)

    def area(self):
        return math.pi*(self.radius**2)

    def circumference(self):
        return 2*math.pi*self.radius

    def __eq__(self, other):
        return

    def __repr__(self):
        return "circle (radius: {0.radius!r}, x: {0.x!r}, y: {0.y!r})".format(self)

    def __str__(self):
        return repr(self)

if __name__ == "__main__":
    circle1 = Circle(5, 3, 4)
    circle2 = Circle(5, 3, 4)
    print(circle1 == circle2)
```

실행 결과

```
True
```

self.radius == other.radius

Correct answers

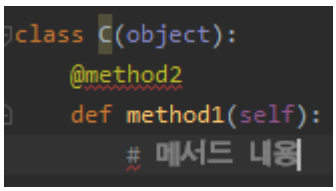
self.radius == other.radius and super().__eq__(other)

return self.radius == other.radius and super(Circle, self).__eq__(other)

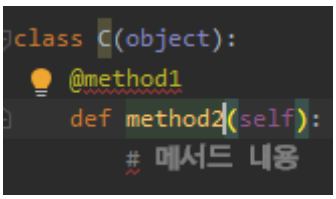
✓ 주어진 코드를 데커레이터 패턴에 맞게 적은 것은? *

1/1

```
class C(object):
    def method1(self):
        # 메서드 내용
    method1 = method2(method1)
```

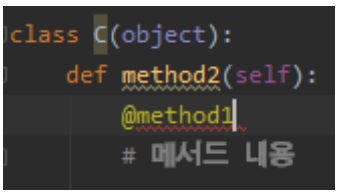


1 ✓



2

3



4

모르겠다.

✓ 다음은 옵서버 패턴에 대한 설명이다. 틀린 것은? * 1/1

- ○ 특정 값을 유지하는 핵심 객체를 갖고, 직렬화된 객체의 복사본을 생성하는 일부 옵서버(관찰자)가 있는 경우 유용하다.
- ○ 객체의 일대다(one-to-many) 의존 관계에서 한 객체의 상태가 변경되면, 그 객체에 종속된 모든 객체에 그 내용을 통지하여 자동으로 상태를 갱신하는 방식이다.
- ○ @property 데커레이터를 사용하여 구현할 수 있다.
- ◉ 모두 다 맞는 이야기이다. ✓
- ○ 모르겠다.

✕ 다음은 싱글턴 패턴을 사용하여 하나의 인스턴스만 생성되도록 제한하는 코드이다. 빈 칸을 채우시오. * 0/2

```
class SinEx:
    _sing = None

    def __new__(self, *args, **kwargs):
        if not self._sing:
            
        return self._sing

if __name__ == "__main__":
    x = SinEx()
    y = SinEx()
    print(x == y)
    print(x)
    print(y)
```

출력 결과

```
True
<__main__.SinEx object at 0x0000027312BD2D08>
<__main__.SinEx object at 0x0000027312BD2D08>
```

'__' ✕

Correct answers

self._sing = super(SinEx, self).__new__(self, *args, **kwargs)

self._sing = super().__new__(self, *args, **kwargs)