기초문법

class

최필주

◉ 구조

```
class class_name:
    class_statements
    class func0(self, 매개변수01, 매개변수02, ...):
        func0_statements
    ...
    class funcN(self, 매개변수N1, 매개변수N2, ...):
        funcN_statements
```

◉ 예시

```
In [ ] : class miner:
             def genMiner(self):
                 print('광부가 생성되었습니다')
In [ ] : SCV = miner()
```

◉ 정의와 호출 예시

In [ ] :
```
class miner:
    def genMiner(self):
        print('광부가 생성되었습니다')
```

In [ ] :
```
SCV = miner()
SCV.genMiner()
```

In [ ] :
```
SCV = miner()
minr.genMiner(SCV)
```

◉ 생성자와 소멸자 - 예시

In [ ] :
```
class miner:
    def __init__(self, mineral):
        self.mineral = mineral
        print('광부가 생성되었습니다')
    def __del__(self):
        print('광부가 죽었습니다')
```

In [ ] :
```
SCV = miner(100)
```

In [ ] :
```
del SCV
```

- Magic method
  - __call__ 예시

In [ ] :
```
class miner:
    def __init__(self, mineral):
        self.mineral = mineral
    def __call__(self):
        print('현재 광물량은:', mineral)
```

In [ ] :
```
SCV = miner(100)
```

In [ ] :
```
SCV()
```

◉ 호출 예제

In [ ] :
```
class miner:
    def __init__(self, mineral):
        self.mineral = mineral
    def miningMineral(self): self.mineral += 10
SCV = miner(100)
```

In [ ] :
```
SCV.miningMineral()
```

In [ ] :
```
SCV.mineral
```

In [ ] :
```
miner.miningMineral(SCV)
```

In [ ] :
```
SCV.mineral
```

instance 변수의 접근

- 예시

In [ ] :
```
class miner:
    def __init__(self, mineral):
        self.mineral = mineral
SCV = miner(100)
```

In [ ] :
```
SCV.mineral += 10
```

In [ ] :
```
SCV.mineral
```

- private
  - 예시

```
In [ ] : class miner:
             def __init__(self, mineral):
                 self.__mineral = mineral
         SCV = miner(100)
```

```
In [ ] : SCV.__mineral
```

```
In [ ] : SCV.__mineral += 10
```

private의 접근

- 접근자(getter)/설정자(setter) - 예시

In [ ] :
```
class miner:
    def __init__(self, mineral):
        self.__mineral = mineral
    def getMineral(self): return self.__mineral
    def setMineral(self, mineral): self.__mineral = mineral
SCV = miner(100)
```

In [ ] :
```
SCV.getMineral()
```

In [ ] :
```
SCV.setMineral(200)
```

In [ ] :
```
SCV.getMineral()
```

private의 접근 … @property 사용하기

- 예시

In [ ] :
```
class miner:
    def __init__(self, mineral):
        self.__mineral = mineral
    @property
    def mineral(self): return self.__mineral
    @mineral.setter
    def mineral(self, m): self.__mineral = m
SCV = miner(100)
```

In [ ] :
```
SCV.mineral
```

In [ ] :
```
SCV.mineral = 200
```

In [ ] :
```
SCV.mineral
```

In [ ] :
```
class miner:
    def __init__(self, mineral):
        self.__mineral = mineral
    @property
    def mineral(self): return self.__mineral
    @mineral.setter
    def mineral(self, m): self.__mineral = m
    def miningMineral(self, m): self.__mineral += m
```

- 추가
  - instance 변수로 gas를 추가하세요
    - private이 되도록 이름 설정하세요
    - @property를 이용하여 접근자와 설정자 method를 추가하세요
    - miningGas method를 추가하여 호출 시 gas값이 5씩 증가하도록 하세요.
  - instance이름으로 함수를 호출하면 현재 mineral과 gas가 출력되도록 하세요

- 수정
  - instance를 생성할 때 gas도 같이 입력할 수 있도록 하세요.

class 변수의 접근

- 예시

In [ ] :
```
class miner:
    totalMineral = 0                    # class 정의 내 - method 정의 외
    def __init__(self, mineral):
        self.mineral = mineral
        miner.totalMineral += mineral  # instance method 정의 내
    def miningMineral(self):
        self.mineral += 10
        miner.totalMineral += 10
```

In [ ] :
```
SCV = miner(50)
probe = miner(100)
```

In [ ] :
```
miner.totalMineral          # class 정의 외 - class이름으로 접근
```

In [ ] :
```
SCV.miningMineral()
```

In [ ] :
```
SCV.totalMineral            # class 정의 외 - instance이름으로 접근
```

- class 변수와 instance 이름이 같은 경우에는?

class 변수의 접근

- 예시 - class 변수와 instance 이름이 같은 경우

In [ ] :
```
class miner:
    mineral = 0                 # class 정의 내 - method 정의 외
    def __init__(self, mineral):
        self.mineral = mineral
        miner.mineral += mineral  # instance method 정의 내
    def miningMineral(self):
        self.mineral += 10
        miner.mineral += 10
```

In [ ] :
```
SCV = miner(50)
probe = miner(100)
```

In [ ] :
```
miner.mineral          # class 정의 외 - class이름으로 접근
```

In [ ] :
```
SCV.miningMineral()
```

In [ ] :
```
SCV.mineral
```

class method의 정의

- 예시

In [ ] :

```
class miner:
    totalMineral = 0
    def __init__(self, mineral):
        self.mineral = mineral
        miner.totalMineral += mineral
    def miningMineral(self):
        self.mineral += 10
        miner.totalMineral += 10
    @classmethod
    def printMineral(cls):
        print(cls.totalMineral)
```

class method의 호출

- 예시

In [ ] :
```
class miner:
    totalMineral = 0
    def __init__(self, mineral):
        self.mineral = mineral
        miner.totalMineral += mineral
    def miningMineral(self):
        self.mineral += 10
        miner.totalMineral += 10
    @classmethod
    def printMineral(cls):
        print(cls.totalMineral)
```

In [ ] :
```
SCV = miner(50)
miner.printMineral()
```

In [ ] :
```
SCV.miningMineral()
SCV.printMineral()
```

- instance 함수와 class 함수의 이름이 같은 경우에는?

class method의 호출

- 예시 – instance 함수와 class 함수의 이름이 같은 경우

In [ ] :
```
class miner:
    totalMineral = 0
    def __init__(self, mineral):
        self.mineral = mineral
        miner.totalMineral += mineral
    def printMineral(self):
        print(self.totalMineral)
    @classmethod
    def printMineral(cls):
        print(cls.totalMineral)
```

In [ ] :
```
SCV = miner(50)
probe = miner(100)
miner.printMineral()
```

In [ ] :
```
SCV.miningMineral()
SCV.printMineral()    # 결과로 500이 나올까? 1500이 나올까?
```

In [ ] :
```
class miner:
    totalMineral = 0
    def __init__(self, mineral):
        self.mineral = mineral
        miner.totalMineral += mineral
    def miningMineral(self):
        self.mineral += 10
        miner.totalMineral += 10
```

◉ 추가 및 수정

- instance 변수 gas와 class 변수 totalGas를 추가하세요
  - instance 생성 시 gas도 같이 입력되고 totalGas도 반영되게 하세요.
  - miningGas method를 추가하고 호출 시 gas와 totalGas가 5씩 증가하도록 하세요.

- class 변수 minerNum을 추가하세요
  - instance가 생성될 때마다 1씩 증가하도록 하세요.
  - instance가 삭제될 때마다 1씩 감소하도록 하세요.
  - class method 추가
    - printMinerNum: 호출 시 현재 minerNum가 출력되도록 하세요

- instance끼리 연산을 수행하는 것을 어떻게 표현할까?
  - Mineral양을 더해 새로운 miner 만들기

In [ ] :
```
class miner:
    def __init__(self, mineral):
        self.mineral = mineral
    def add(self, other):
        return miner(self.mineral + other.mineral)
```

In [ ] :
```
SCV = miner(50)
probe = miner(100)
```

In [ ] :
```
drone = SCV.add(probe)
drone.mineral
```

    - SCV.add(probe)를 SCV + probe 처럼 연산자를 사용하여 나타낼 수 없을까?

- instance끼리 연산을 수행하는 것을 어떻게 표현할까?
  - Mineral양을 더해 새로운 miner 만들기: + 연산자로 표현하기

In [ ] :
```
class miner:
    def __init__(self, mineral):
        self.mineral = mineral
    def __add__(self, other):
        return miner(self.mineral + other.mineral)
```

In [ ] :
```
SCV = miner(50)
probe = miner(100)
```

In [ ] :
```
drone = SCV + probe  # SCV.__add__(probe)와 동일
drone.mineral
```

기본코드

In [ ] :
```
class miner:
    def __init__(self, mineral, gas):
        self.mineral = mineral
        self.gas = gas
```

추가

- + 연산: 두 instance의 mineral과 gas가 더해 반환
- - 연산: 두 instance의 mineral과 gas가 서로 바뀜(반환 X)
- * 연산: 두 instance의 mineral과 gas의 값 중 큰 값들만 반환

◉ 슈퍼 클래스로 사용할 코드

In [ ] :
```
class miner:
    def __init__(self, mineral):
        self.mineral = mineral
    def miningMineral(self):
        self.mineral += 10
```

◉ 자식 클래스

In [ ] :
```
class goldMiner(miner):
    def __init__(self, mineral, gold):
        super().__init__(mineral)
        self.gold = gold
    def miningGold(self):
        self.gold += 10
```

예시1

In [ ] :
```
class goldMiner(miner):
    def __init__(self, mineral, gold):
        super().__init__(mineral)
        self.gold = gold
```

예시2

In [ ] :
```
class goldMiner(miner):
    def __init__(self, mineral, gold):
        miner.__init__(self, mineral)
        self.gold = gold
```

예시

- miningMineral method 호출 시 mineral 증가량 변경

In [ ] :
```
class miner:
    def __init__(self, mineral):
        self.mineral = mineral
    def miningMineral(self):
        self.mineral += 10
```

In [ ] :
```
class goldMiner(miner):
    def __init__(self, mineral, gold):
        super().__init__(mineral)
        self.gold = gold
    def miningMineral(self):
        self.mineral += 10 + self.gold * 0.01
```

◉ 예시1 – 슈퍼 클래스의 이름 사용

In [ ] :

```
class grandparent:
    def __init__(self):
        print("grandparent")
class parent1(grandparent):
    def __init__(self):
        grandparent.__init__(self)
        print("parent1")
class parent2(grandparent):
    def __init__(self):
        grandparent.__init__(self)
        print("parent2")
class child(parent1, parent2):
    def __init__(self):
        parent1.__init__(self)
        parent2.__init__(self)
        print("child")
person = child()
```

◉ 예시2 – super() 활용

In [ ] :

```
class grandparent:
    def __init__(self):
        print("grandparent")
class parent1(grandparent):
    def __init__(self):
        super().__init__()
        print("parent1")
class parent2(grandparent):
    def __init__(self):
        super().__init__()
        print("parent2")
class child(parent1, parent2):
    def __init__(self):
        super().__init__()
        print("child")
person = child()
```

방법

	가져오기	사용하기	예시
기본방법	import *모듈1, 모듈2, ...*	*모듈1.함수1*()	import random random.randint(1, 10)
alias 지정	import *모듈* as *alias*	*alias.함수1*()	import random as rn rn.randint(1, 10)
일부만 가져오기	from *모듈* import *함수1, 함수2, ...*	*함수1*()	from random import * randint(1, 10)
일부만 가져오면서 alias 지정	from *모듈* import *함수1 as a1, 함수2 as a2, ...*		

- *: 모든 것을 의미
- 함수 말고 모듈의 변수, class도 사용 가능

패키지 가져오기: *모듈* → *패키지.모듈*로 표현

예시

- cal.py

```
def add(x, y): return x + y
def sub(x, y): return x – y

print(add(3, 4))
print(sub(7, 2))
print("cal.py의 이름:", __name__)
```

```
def add(x, y): return x + y
def sub(x, y): return x – y

if __name__ == "__main__":
    print(add(3, 4))
    print(sub(7, 2))
```

- 왼쪽의 경우 import cal을 하면 print문들이 실행됨
- __name__의 값 확인
  - cal.py를 import하였을 때 마지막 print의 결과(__name__의 값) 확인하기
  - cal.py를 직접 실행시켰을 때의 값과 비교하기