

파이썬 기초

13 파이썬 클래스



홍필두 교수
(파이썬기초)



학습내용

- 01 클래스
- 02 메서드
- 03 유틸리티 클래스





학습목표

- 클래스에 대하여 이해하고 파이썬 클래스를 사용할 수 있다.
- 특별한 의미의 메서드를 대하여 이해하고 사용할 수 있다.
- 유틸리티 클래스를 이해하고 몇몇 예제를 사용할 수 있다.



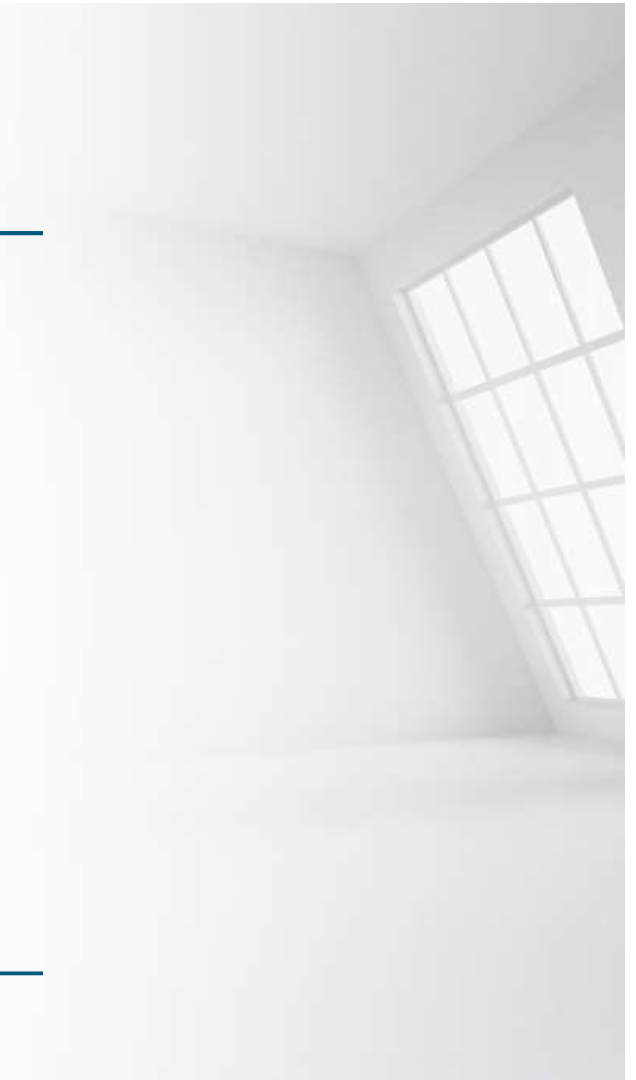
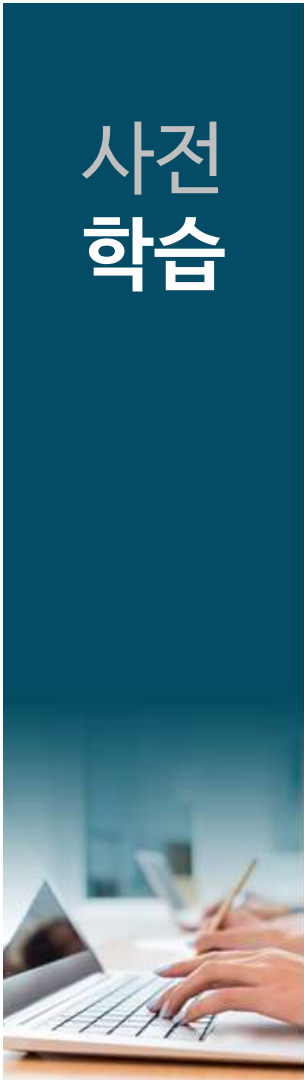
사전 학습

“파이썬 클래스”

클래스는 메소드(Method : 함수의 역할)와 인터페이스 (Interface : 변수-데이터의 역할)로 구성되어 있는 특별한 객체입니다.

파이썬에도 이러한 클래스를 정의할 수 있지만 Java, C++, C#과 같은 객체지향 언어가 아니기 때문에 기능이 상대적으로 부족하고, 개념정의가 어려운 부분이 존재하기 때문에, 파이썬 클래스를 이해하고 사용하는데 조금 노력이 필요합니다.

이번 강의에서는 이러한 파이썬의 클래스를 사용하는 방법에 관한 몇 가지 사항을 배웁니다. 여러분은 먼저 해당 내용에 대하여 검색을 통하여 미리 알아보고 조사해 본 후, 본 강의를 듣도록 합시다.



01

클래스

- 1) 객체지향 프로그래밍
- 2) 구조체 (struct)와 클래스 (class)
- 3) 파이썬 클래스
- 4) self
- 5) 생성자
- 6) 상속
- 7) 정보은닉 (getter, setter)

1) 객체지향 프로그래밍

객체지향 프로그래밍 (OOP : Object Oriented Programming)

Java나 C++과 같은 OO프로그래밍 언어를 사용하여 OOD를
실제화 하는 작업

↳ 객체 (Object)를 프로그램으로 표현 (객체모델화)한다면
단위 Object는 Class로 표현

1) 객체지향 프로그래밍

클래스는 메소드(Method : 함수의 역할)와
인터페이스(Interface : 변수-데이터의 역할)로 구성됨

클래스(Class)

같은 종류 및 특성을 가진
객체들을 모아서 공통의
특성으로 분류하고
템플릿화 하는 것

인스턴스(Instance)

클래스의 실체들로서
템플릿화된 클래스에서
파생된 하나의 실제 객체

예 붕어빵 틀과 붕어빵

2) 구조체 (struct)와 클래스(class)



일반적인 프로그래밍 언어에서 많이 사용하는 구조체

※ 아래는 엘리베이터의 속성을 표현

```
struct elevator
{
    int limit-up-floor;
    int down-floor;
    int floor;
    char help[100];
}
```


2) 구조체 (struct)와 클래스(class)



구조체는 변수들을 묶어서 레코드개념으로 프로그램에서 사용함



단순히 프로그램 내에서 변수들의 묶음으로 사용될 뿐임



파이썬의 리스트 내에 여러 형태의 자료형을 넣을 수 있으나 함수(def)를 포함할 수는 없음

2) 구조체(struct)와 클래스(class)



클래스의 기본은 변수(데이터)와 함수(기능, method)를 가지고 있음

다음 클래스는 엘리베이터라는
실 세계(Real World)의 개념(Object)을 구현한 사례

- 클래스를 설계하는 사람과 전체 프로그램을 구현하는 사람이 다를 수 있음
- 클래스의 세부 내용은 복잡성을 가지기 때문에 내용을 은닉하고 인터페이스만으로 클래스를 사용하도록 단순화 함
- 즉, TV에서 내부에 어떠한 처리과정이 일어나는지(Object 내용)는 사용자에게는 관심 없고, 채널과 볼륨스위치(인터페이스)만 밖에 보여지면 됨
- 인터페이스를 up, down정도만 프로그램에서 제어할 수 있도록 한다면 프로그램은 간단히 구현될 것임

2) 구조체(struct)와 클래스(class)



class elevator를 이용한다면 프로그램은 쉽게 구현됨



일단 프로그램을 개념중심으로 먼저 구현한 후
class의 내부를 구현하는 방법도 적용이 가능함

```
void main()
{
    class *myclass elevator;
    if(올라가라는 전기 신호가 왔다면)
        myclass->up();
    else if(내려가라는 전기 신호가 왔다면)
        myclass->down();
}
```

2) 구조체 (struct)와 클래스(class)



class안에 elevator의 기능을 세세히 정의하지만 사용하는 입장에서는 알 필요는 없음



파이썬에도 이러한 클래스를 정의할 수 있음

BUT

Java, C++, C#과 같은 객체지향 언어가 아니기 때문에 기능이 상대적으로 부족하고, 개념정의를 어려운 부분이 존재함

2) 구조체(struct)와 클래스(class)

```
class elevator
{
    int limit-up-floor=10; //최상위 층
    int down-floor=0; //최하위층
    int floor; //현재층
    char help[100];
    void up() //엘리베이터가 올라감
    {
        if (floor == limit-up-floor)
            help=" last-floor";
        else
            floor++; //최상층이 아닐 때 한 층씩 올라감
    }
    void down() //엘리베이터가 내려감
    {
        if (floor == limit-down-floor)
            help=" last-floor";
        else
            floor++; //최하층이 아닐 때 한 층씩 올라감
    }
}
```

Java프로그램에서 예시

3) 파이썬 클래스



앞의 일반적 클래스 예시를 파이썬으로 구현함



먼저 전체를 따라서 코딩을 해 본 후 하나씩 깊게 살펴보도록 함

3) 파이썬 클래스



아래는 클래스를 이용한 메인 처리 부분임

```
myclass = elevator(1,15,0,"") #(현재1층, 최저0층, 최고15층, 메시지)
                                #값으로 클래스생성

for i in range(20): #20번 up()를 실행
    myclass.up()
for i in range(20): #20번 down()을 실행
    myclass.down()
```

```
>>>
RESTART: ***.py
2층입니다
3층입니다
4층입니다
5층입니다
~~~0층입니다
최하층 입니다
최하층 입니다
최하층 입니다
최하층 입니다
최하층 입니다
>>>
```

3) 파이썬 클래스



클래스 부분은 다음과 같이 구현함

```
class elevator:
    def __init__(self, floor, limit_up_floor, limit_down_floor, msg):
        self.floor=floor
        self.limit_up_floor=limit_up_floor
        self.limit_down_floor=limit_down_floor
        self.msg=msg
    def up(self):
        if self.floor == self.limit_up_floor:
            self.msg="최상층 입니다"
        else:
            self.floor+=1
            self.msg=str(self.floor)+"층입니다"
        print(self.msg)
    def down(self):
        if self.floor == self.limit_down_floor:
            self.msg="최하층 입니다"
        else:
            self.floor-=1
            self.msg=str(self.floor)+"층입니다"
        print(self.msg)
```


4) self



클래스를 호출하면 하나의 복제물 (인스턴스, **instance**)이 생성됨

→ self를 의미함



각 함수에 첫 글자에는 self를 기입함



클래스 내에서 사용되는 변수들에게 self.floor와 같은 방식으로 값을 가지고 있음

4) self

class elevator

- 단순 선언일 뿐 어떤 메모리 공간이나 정의된 함수들이 만들어진 상태는 아님

myclass= elevator()

- 클래스를 호출하면서 사용 가능한 실체인 instance가 생성
- self는 myclass라고 지칭되는 instance를 정의함

yourclass= elevator()

- 다른 곳에서 elevator클래스라는 틀을 가지고 yourclass라는 인스턴스를 생성, 즉 myclass와 yourclass는 전혀 다른 메모리 공간과 메소드를 갖는 객체(object)임

4) self



같은 엘리베이터 라는 설계를 가지고 만들어진 myclass와 yourclass



다음을 실행하면 두 엘리베이터가 각각 다른 값(변수)과 함수(method)로 저장되어 있음을 알 수 있음



각각의 instance는 self라는 정의값으로 분리됨



어렵지만 차분이 생각해 보고, 계속 따라서 사용하다 보면 규칙이 보임

4) self

```
myclass = elevator(1,15,0,"") #(현재1층, 최저0층, 최고15층, 메시지)값으로 클래스생성
yourclass = elevator(5,10,-1,"") #(현재1층, 최저0층, 최고15층, 메시지)값으로 클래스생성

for i in range(20): #20번 up()를 실행
    print("myclass->",end="")
    myclass.up()
    print("yourclass->",end="")
    yourclass.up()

for i in range(20): #20번 down()을 실행
    print("myclass->",end="")
    myclass.down()
    print("yourclass->",end="")
    yourclass.down()
```

5) 생성자



클래스를 처음 생성하여 인스턴스를 만들 때 호출되는 함수 (method)



호출하는 쪽에서는 클래스 명으로 부르고,
클래스에는 `__init__`로 정의함

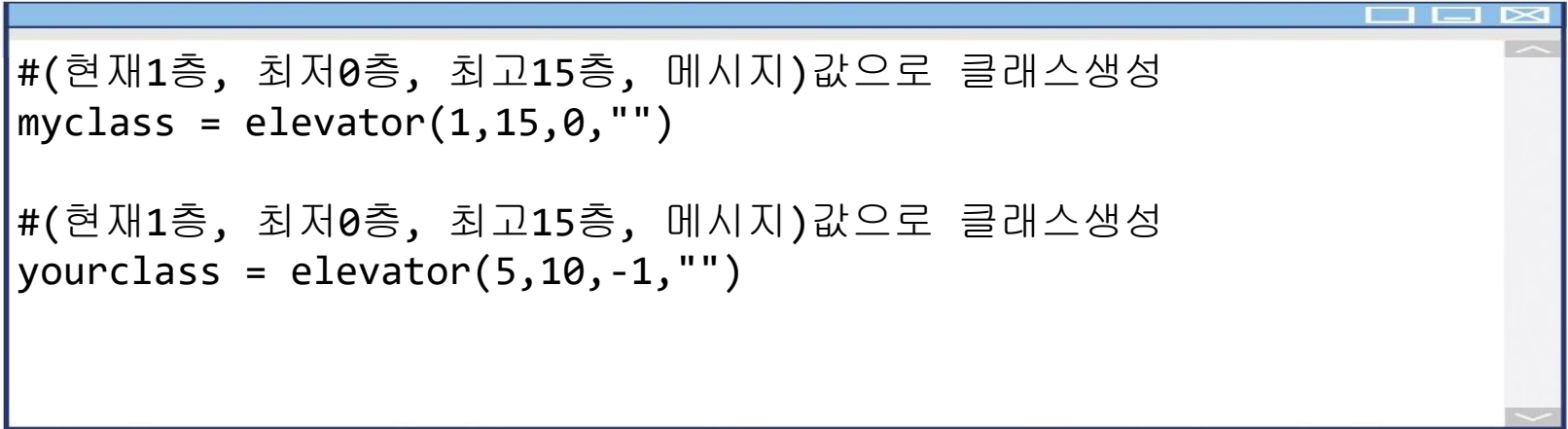
5) 생성자

class 선언 부분

```
class elevator:
    def __init__(self, floor, limit_up_floor, limit_down_floor, msg):
        self.floor=floor
        self.limit_up_floor=limit_up_floor
        self.limit_down_floor=limit_down_floor
        self.msg=msg
```

5) 생성자

class 호출 부분



```
#(현재1층, 최저0층, 최고15층, 메시지)값으로 클래스생성  
myclass = elevator(1,15,0, "")  
  
#(현재1층, 최저0층, 최고15층, 메시지)값으로 클래스생성  
yourclass = elevator(5,10,-1, "")
```

6) 상속

상속

기존 클래스의 정의를 가지고, 즉 성질을 그대로 물려받은 후
변수 및 method의 추가, 수정, 변경, 삭제로
또 다른 클래스를 정의하는 것



다음은 기존 elevator를 가지고
새로운 s_elevator클래스를 생성한 예제임



새로운 클래스는 기존 클래스를 가져다 한 번에
두 개 층을 올라가거나, 내려가는 것으로 정의하였음



기존 부모클래스는 super()로 명명함

6) 상속

상속된 클래스 정의 예

```
class s_elevator(elevator):  
    def __init__(self):  
        super().__init__(1,15,0,"")  
    def up(self):  
        super().up()  
        super().up()  
    def down(self):  
        super().down()  
        super().down()
```

6) 상속

상속된 클래스 호출과 기존 클래스 호출

```
#myclass = elevator(1,15,0,"")
myclass = s_elevator()

for i in range(20): #20번 up()를 실행
    myclass.up()
    myclass.down()
```

7) 정보은닉 (getter, setter)

클래스 내부 변수

해당 클래스를 호출하는 다른 여러 개의 클래스에서 예상치 못한 접근으로 예측할 수 없는 상황을 막기 위하여 입출력을 만들어서 해당 입출력 부분으로만 클래스의 변수에 접근할 수 있도록 함

기본

- [getter를 통한 클래스 값 가지고 오기]
- [setter를 통한 클래스에 값 입력하기]

7) 정보은닉 (getter, setter)

 getter, setter를 통한 접근 예제로 “a**b” 거듭제곱을 구하는 예제

```
class multibox:
    def __init__(self, multi):
        self.multi=multi
    def getnumber(self):
        return self.num**self.multi
    def setnumber(self, num):
        self.num=num
```

```
m=multibox(5)
m.setnumber(5)
print(m.getnumber())
```



```
>>>
RESTART: **.py
3125
```

7) 정보은닉 (getter, setter)



property를 이용하여 getter, setter를 정의할 수 있음

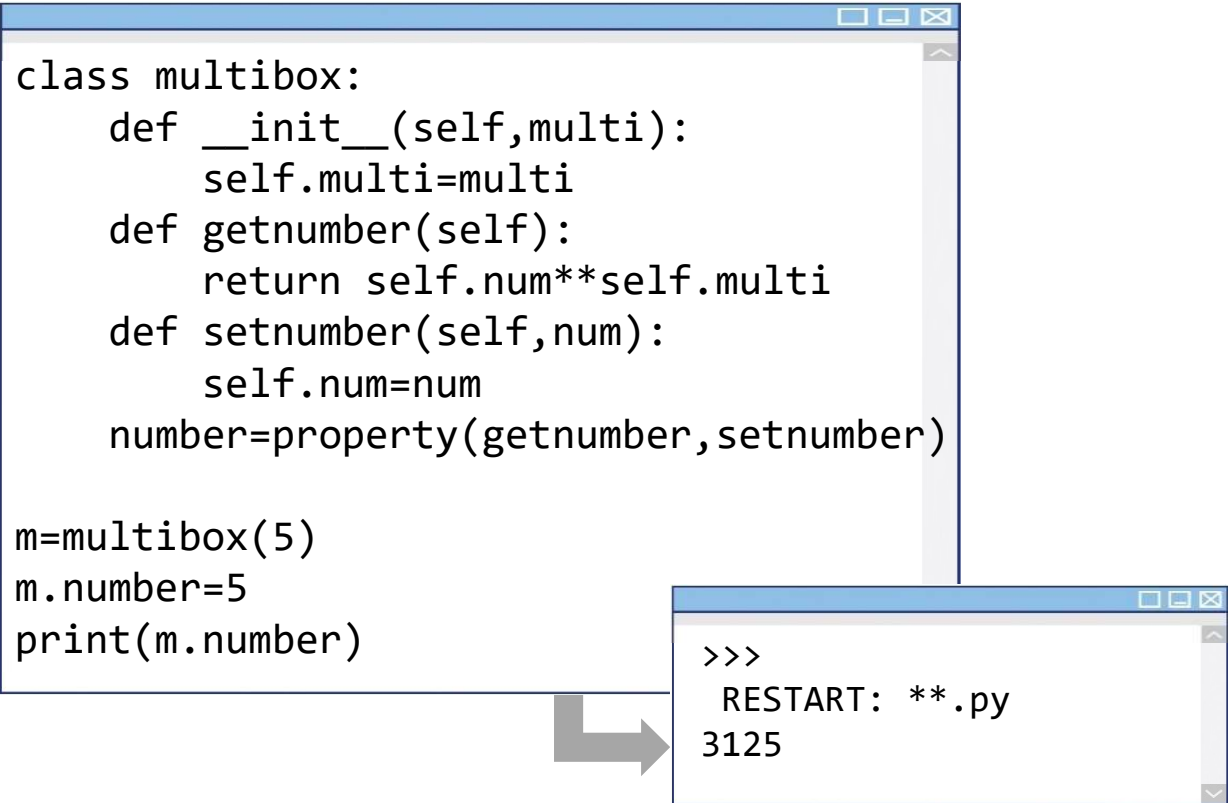


클래스를 사용하는 경우에는 단순히 일반 클래스 변수
같이 사용이 가능함

7) 정보은닉 (getter, setter)

```
class multibox:
    def __init__(self, multi):
        self.multi=multi
    def getnumber(self):
        return self.num**self.multi
    def setnumber(self, num):
        self.num=num
    number=property(getnumber, setnumber)

m=multibox(5)
m.number=5
print(m.number)
```



```
>>>
RESTART: *.py
3125
```

7) 정보은닉 (getter, setter)



데코레이터(@)를 이용하여 getter, setter를 정의할 수 있음




@property, @**.setter로 정의함

7) 정보은닉 (getter, setter)

```
class multibox:
    def __init__(self, multi):
        self.multi=multi
    @property #데코레이터로 선언
    def number(self):
        return self.num**self.multi
    @number.setter
    def number(self, num):
        self.num=num

m=multibox(5)
m.number=5
print(m.number)
```



```
>>>
RESTART: *.py
3125
```




02

메서드

- 1) 클래스 메서드
- 2) 정적 메서드
- 3) 연산자 메서드

1) 클래스 메서드

{ 클래스에서 정의하는 메서드(method, 함수)중에
몇몇 특별한 기능의 메서드를 정의하는 방법을 알아봄 }

클래스 메서드

클래스 정의를 가져다 선언하고 있는 전체의 클래스에서
공유하는 메서드

선언 : @classmethod

cls : 전체 클래스

1) 클래스 메서드

```
class myclass:
    cnt=0 #self가 없는 myclass에서 사용되는 내부 static변수
    def __init__(self):
        myclass.cnt+=1 #myclass의 static변수
    @classmethod #클래스 메서드 선언
    def refcount(cls): #self가 아니라 cls로 정의
        print("클래스 참조횟수는",cls.cnt,"회 입니다")

a=myclass()
b=myclass()
myclass.refcount()
```

>>>
RESTART: **.py
클래스 참조횟수는 2 회 입니다

2) 정적 메서드

정적 메서드

생성되는 클래스와는 별개로 하나의 역할을 수행하는 메서드

선언 : @staticmethod



클래스 생성과 상관없으므로 `__init__`를 수행하지 않아도 됨



`self` 나 `cls`를 인수로 받지도 않음

2) 정적 메서드

```
class myclass:
    cnt=0
    def __init__(self):
        myclass.cnt+=1
    @staticmethod
    def helf():
        print("이 클래스는 아무런 일도 하지 않는다")

myclass.helf()
a=myclass()
myclass.helf()
```

>>>
RESTART: ** .py
이 클래스는 아무런 일도 하지 않는다
이 클래스는 아무런 일도 하지 않는다

3) 연산자 메서드

정적 메서드


클래스와 클래스의 연산관계를 정의하는 특별한 메서드

3) 연산자 메서드

다음 `__add__` 은 클래스간 +연산을 정의한 메서드임

```
class mystr:
    def __init__(self,s):
        self.s=s
    def __add__(self, other): # 클래스간 +연산자를 정의
        return other.s + self.s # 문자간 연산을 반대로 수행

a = mystr("abc")
b = mystr("def")
print(a+b)
```



```
>>>
RESTART: ** .py
defabc
```



03

유틸리티 클래스

- 1) Decimal
- 2) Fraction

1) Decimal

Decimal

유틸리티 클래스는 유용한 기능을 미리 만들어 클래스로 제공하는 것

1) Decimal

유틸리티 클래스를 이해하기 위한 몇몇 클래스를 소개함

- 3분의 1을 십진수로 표현하면 0.33333333... 으로 완벽히 표현할 수 없음
- 계산상 오차가 나타남
- 마찬가지로 2진수로는 0.1 을 완벽히 표현할 수 없음 (컴퓨터 연산오류)

```
sum=0
for i in range(100):
    sum+=0.1

print(sum)
```

오차발생

0.1을 100번 더하면 10이지만 9.999999999999998가 나옴

```
>>>
RESTART: *.py
9.999999999999998
```

1) Decimal



Decimal클래스는 문자형을 숫자형(정수형)으로 바꾸는데 사용함



다른 관점으로 보면 함수처럼 사용

BUT

문자형으로 클래스를 초기화하여 생성하면
컴퓨터 연산오류 없는 계산을 수행가능(십진수 계산)

1) Decimal

```
from decimal import Decimal
```

```
a_sum=0;b_sum=0;c_sum=0
```

```
a=0.1
```

```
b=Decimal(a)
```

```
c=Decimal("0.1")
```

```
for i in range(100):
```

```
    a_sum+=a
```

```
    b_sum+=b
```

```
    c_sum+=c
```

```
print(a,"=>",a_sum)
```

```
print(b,"=>",b_sum)
```

```
print(c,"=>",c_sum)
```

```
>>>
```

```
RESTART: ** .py
```

```
0.1 => 9.999999999999998
```

```
0.10000000000000000055511151231257827021181583
```

```
404541015625 => 10.000000000000000055511151230
```

```
0.1 => 10.0
```

2) Fraction

Fraction 클래스

분수를 나타내 주고 분수계산이 가능한 클래스
(단, 정수 및 실수와 같이 연산하면 실수형으로 바뀜)

```
from fractions import *  
  
a=Fraction(2,3) #3분의 2  
b=Fraction(5,10) #10분의 5로 약분되어 1/2  
  
print(a,"+",b,"=",a+b)  
print(a,"*",b,"=",a*b)  
print(a,"*",1.5,"=",a+1.5)
```

```
>>>  
RESTART: *.py  
2/3 + 1/2 = 7/6  
2/3 * 1/2 = 1/3  
2/3 * 1.5 = 2.1666666666666665
```

04

실습하기 I

- 1) 클래스 - 객체지향 프로그래밍, 구조체와 클래스
- 2) 클래스 - 파이썬 클래스, self
- 3) 클래스 - 생성자, 상속, 정보은닉

실습내용

- 1) 클래스 - 객체지향 프로그래밍, 구조체와 클래스
- 2) 클래스 - 파이썬 클래스, self
- 3) 클래스 - 생성자, 상속, 정보은닉

05

실습하기 II

- 1) 메서드 - 클래스 메서드, 정적 메서드, 연산자 메서드
- 2) 유틸리티 클래스 - Decimal, Fraction
- 3) 끝맺는 말

실습내용

- 1) 메서드 - 클래스 메서드, 스테틱 메서드,
연산자 메서드
- 2) 유틸리티 클래스 - Decimal, Fraction
- 3) 끝맺는 말



학습활동

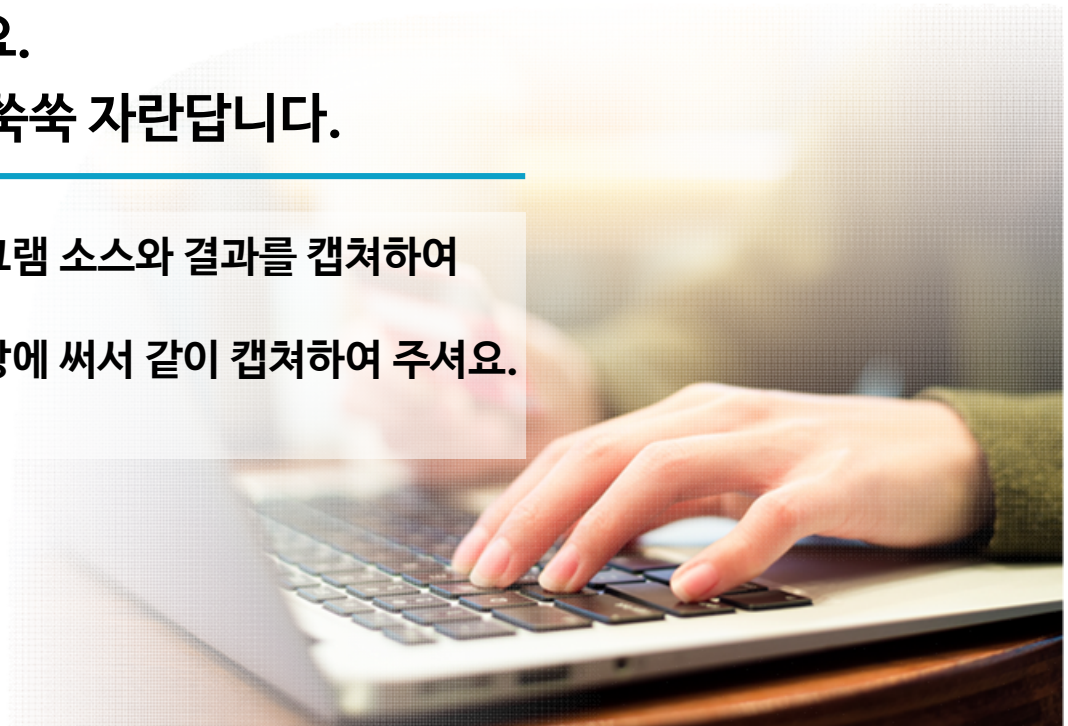
일시정지 버튼을 누른 후, 아래의 학습활동에 참여하세요.

Q

오늘 배운 내용을 스스로 실습하여
자유게시판에 올려 주세요.

이렇게 정리하면 실력이 쑥쑥 자란답니다.

- ① 본인이 실습한 내용을 프로그램 소스와 결과를 캡처하여 올려주세요.
- ② 본인의 학번과 이름을 메모장에 써서 같이 캡처하여 주세요.
- ③ 그리고 설명도 달아 주세요.





학습활동에 대한 교수님 의견

Q

오늘 배운 내용을 스스로 실습하여 자유게시판에 올려 주세요.
이렇게 정리하면 실력이 쑥쑥 자란답니다.

A

[오늘 학습한 내용의 실습 사항]

- ① 클래스 - 객체지향 프로그래밍, 구조체와 클래스
- ② 클래스 - 파이썬 클래스, self
- ③ 클래스 - 생성자, 상속, 정보은닉
- ④ 메서드 - 클래스 메서드, 스테틱 메서드, 연산자 메서드
- ⑤ 유틸리티 클래스 - Decimal, Fraction

학습 평가

Q1

Q1

Q2

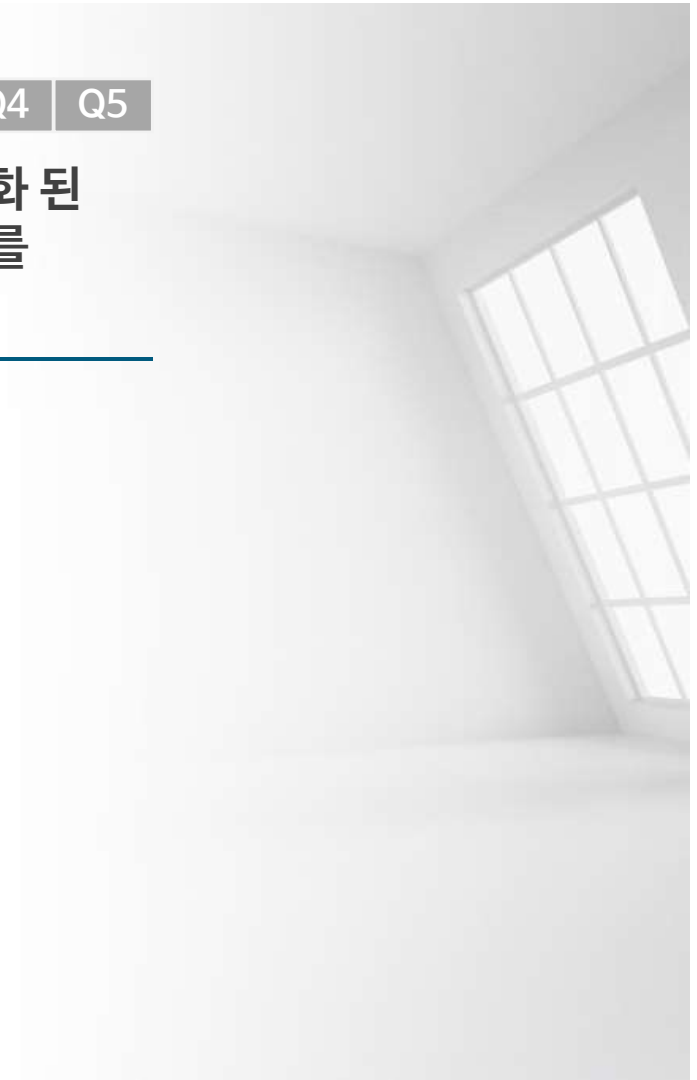
Q3

Q4

Q5

다음 중 클래스의 실체들로서 템플릿화 된 클래스에서 파생된 하나의 실제 객체를 의미하는 용어는?

- 1 메서드
- 2 상속
- 3 인터페이스
- 4 인스턴스



학습 평가

Q1

Q2

Q3

Q4

Q5

Q1

다음 중 클래스의 실체들로서 템플릿화 된 클래스에서 파생된 하나의 실제 객체를 의미하는 용어는?

- 1 메서드
- 2 상속
- 3 인터페이스
- ☒ 4 인스턴스

정답

4번

해설

인스턴스의 정의에 대한 설명입니다.

학습 평가

Q1

Q2

Q3

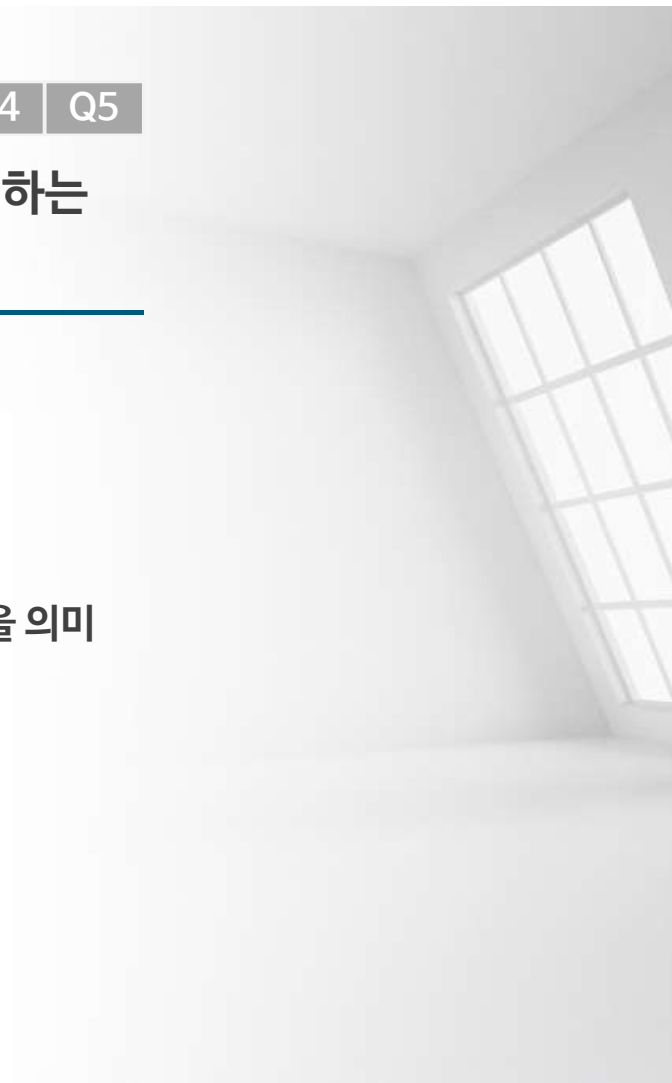
Q4

Q5

Q2

다음 중 class 선언내부에 self가 의미하는 것은?

- 1 무의미 하므로 패스
- 2 스스로 상속하라는 메서드
- 3 클래스 생성시 instance를 의미
- 4 클래스 생성과 상관없이 전체 클래스임을 의미



학습 평가

Q1

Q2

Q3

Q4

Q5

Q2

다음 중 class 선언내부에 self가 의미하는 것은?

- 1 무의미 하므로 패스
- 2 스스로 상속하라는 메서드
- ☒ 3 클래스 생성시 instance를 의미
- 4 클래스 생성과 상관없이 전체 클래스임을 의미

정답

3번

해설

클래스를 호출하면 하나의 복제물 (인스턴스, instance)가 생성되는데, 이 instance가 self를 의미합니다.

학습 평가

Q1

Q2

Q3

Q4

Q5

Q3

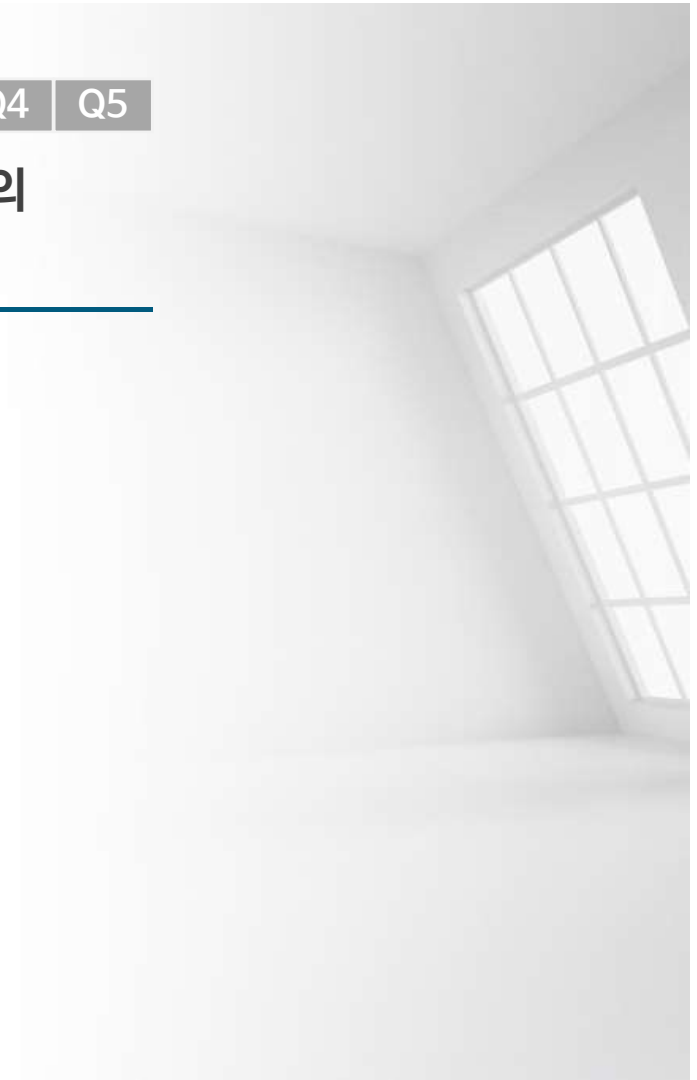
다음 중 class abc:로 정의된 클래스의 생성자를 의미하는 것은?

1 __abc__

2 __class__

3 __init__

4 __start__



학습 평가

Q1

Q2

Q3

Q4

Q5

Q3

다음 중 class abc:로 정의된 클래스의 생성자를 의미하는 것은?

1 __abc__

2 __class__

☒ 3 __init__

4 __start__

정답

3번

해설

생성자는 __init__로 정의합니다.

학습 평가

Q1

Q2

Q3

Q4

Q5

Q4

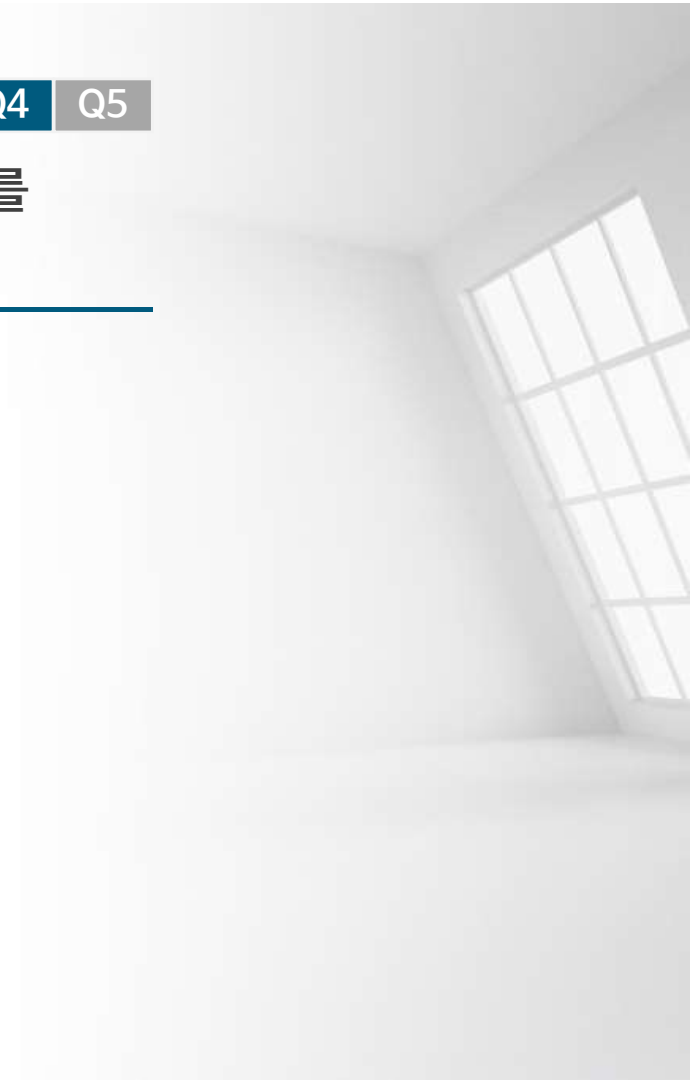
다음 상속된 클래스에서 부모 클래스를 의미하는 것은?

1 parent()

2 upper()

3 super()

4 this()



학습 평가

Q1

Q2

Q3

Q4

Q5

Q4

다음 상속된 클래스에서 부모 클래스를 의미하는 것은?

1 parent()

2 upper()

☒ 3 super()

4 this()

정답

3번

해설

부모클래스는 super()로 명명합니다.

학습 평가

Q1

Q2

Q3

Q4

Q5

Q5

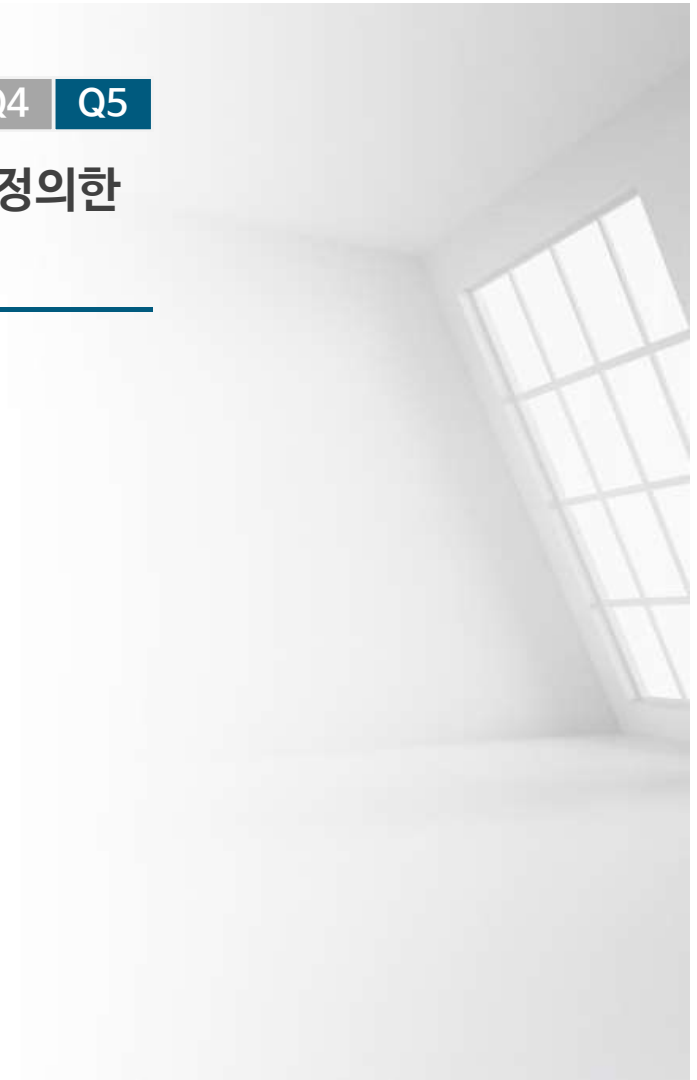
다음 중 클래스 간의 +연산자를 다시 정의한 메서드를 의미하는 것은?

1 __add__

2 @add

3 #add

4 @+



학습 평가

Q1 Q2 Q3 Q4 Q5

Q5

다음 중 클래스 간의 +연산자를 다시 정의한 메서드를 의미하는 것은?

☒ 1 __add__

2 @add

3 #add

4 @+

정답

1번

해설

__add__ 은 클래스간 +연산을 정의한 메서드입니다.



정리하기

클래스

- ✓ 클래스는 메소드(Method : 함수의 역할)와 인터페이스(Interface : 변수-데이터의 역할)로 구성됨
- ✓ 파이썬에도 이러한 클래스를 정의할 수 있지만 Java, C++, C#과 같은 객체지향 언어가 아니기 때문에 기능이 상대적으로 부족하고, 개념정의가 어려운 부분이 존재함
- ✓ 클래스를 호출하면 하나의 복제물(인스턴스, instance)가 생성하는데, 이 instance를 self로 명명함
- ✓ 생성자는 클래스를 처음 생성하여 인스턴스를 만들 때 호출되는 함수(method)로 호출하는 쪽에서는 클래스 명으로 부르고, 클래스에는 __init__로 정의함





정리하기

클래스

✓ 상속

- 기존 클래스의 정의를 가지고, 즉 성질을 그대로 물려받은 후 변수 및 method의 추가, 수정, 변경, 삭제로 또 다른 클래스를 정의하는 것

✓ 정보은닉

- 클래스 내부 변수는 해당 클래스를 호출하는 다른 여러 개의 클래스에서 예상치 못한 접근으로 예측할 수 없는 상황을 막기 위하여 입출력을 만들어서 해당 입출력 부분으로만 클래스의 변수에 접근할 수 있도록 함





정리하기

메서드

- ✓ 클래스에서 정의하는 메서드 중 특별한 기능을 가진 메서드를 정의할 수 있음

유틸리티 클래스

- ✓ 유용한 기능을 미리 만들어 클래스로 제공하는 것

