

2024 09 27 수업내용 정리



3. 제어문

스페이스 4개랑 탭을 혼용해서 쓰지 말기

tab

space 4



조건문=result true 1
false 0

if 문

in 쓸 수 있는 곳 -string

'p' in "python"

"p" not in "python"

→for in 이랑 다름 (하나하나씩 꺼냄)

if문 for문 잘쓰면 좋음.그러나 자주 쓰면 안됨(속도때문에, 병목현상때매 원래 있던 대로 돌아가야 되면 주소를 다기억해야 됨.주소를 너무 많이 기억하기 때문에)

assembly - 집합, 아세이(통째)

instructions add move

print 1번지

move 10번지

move ~번지

————→ 계속 반복되는 것을 assembly라고 한다.



언어는 assembly어로 만들고 compile도 assembly로 만들

cpu마다 컴파일러를 다 하나하나씩 만들어야 함.

if

elif

else



while 문 사용할 일 잘 없음(무한반복문에는 좀 씀)

while |:
true" →이 문을 많이 씀



break,continue,pass,return 4개의 차이점

break -기어 (제어문)에서씀

1,2,3 for문의 기어 돌아감→ 3번 박살남 break문 한번만 쓰기

반복문을 부수고 처음위치로 돌아감

continue(제어문)에서씀

반복문을 부수지않고 처음위치로 돌아감

pass 함수에서씀

함수만들어놓고 생각안나면 pass치면 됨

return 함수에서씀



- for 문에서 continue사용할 수 있음.
→어디서사용할까?



맵 이터레이터 란다 →속도느림?

링크드림 깃허브를 통해 자기어필 하기!!



함수 /

입력은 무한정 받을 수 있으나 출력도 무한히 받을수 있음(파이썬)

씨 언어는 하나밖에 안됨

입력값이 무한이라는 거는 입력값이 0이어도 작동한다라는 뜻

리턴값이 없을 수도 있고 입력값이 없을 수 도있다.(리터값이 없다=아무의미 없다)

함수안에서

def ()매개변수 를 항상 연결한다.

지역변수는 (임시변수)



```
(선언)def (a,b,c,d,e)
    print(a+b+c+d+e)
    return 0
```

사용 def (c k d e g)

```
def (c=d,d=e,e=g,a=c,d=k)
def(a=c,b=k,c=d,d=e,e=g)
→이렇게 코드짜는 걸 추천
```

ctrl + /

맥주집 게임

내가 입력한거 외에 다 에러처리해라(그렇지 않으면 터짐)

```
def(*args)
def(a,*args,c)
```

1=a

2,3,4,5,6 (args)

내가 입력한 거외에 에러처리 그래야 오류 안남

*args(왜 * 붙임?)

```
def (*args)
```

```
def (1,2,3,4,5,6)
```

```
args ={1,2,3,4,5,6}
```

*



- args에서 *왜 붙임?

ChatGPT의 말:

ChatGPT

- `args` 에서 `*` 는 Python에서 가변 인자 목록을 처리하는 데 사용되는 문법입니다. 이 문법의 목적과 의미를 자세히 설명하겠습니다.

`args` 의 의미

1. 가변 인자:

- `args` 는 함수가 호출될 때 임의의 수의 인자를 받을 수 있게 해 줍니다. 여기서 `args` 는 **arguments**의 약자로, 인자들을 의미합니다.
- 예를 들어, `args` 를 사용하면 함수에 전달되는 인자의 수에 제한이 없게 됩니다.

2. 별표(`*`)의 역할:

- 별표(`*`)는 Python에서 **언팩(unpack)** 또는 ****패킹(packing)****을 의미합니다. 즉, 함수 호출 시 여러 개의 인자를 하나의 튜플로 묶어서 전달할 수 있게 해줍니다.
- 이 때문에 `args` 를 함수 정의에서 사용하면, 전달된 모든 인자를 하나의 튜플로 묶어 사용할 수 있습니다.

예시

다음은 `*args` 를 사용하는 함수의 간단한 예시입니다:

python

코드 복사

```
def my_function(*args):  
    for arg in args:  
        print(arg)
```

`my_function(1, 2, 3)` # 1, 2, 3이 출력됩니다.

`my_function('a', 'b', 'c', 'd')` # a, b, c, d가 출력됩니다.

- 위의 `my_function` 은 호출 시 몇 개의 인자가 전달되든 상관없이 모두 처리할 수 있습니다. 전달된 인자는 `args` 라는 튜플로 묶여서 함수 안에서 사용됩니다.

비유

- `args` 를 **바구니에 여러 과일을 담는 것**으로 비유할 수 있습니다. 바구니()는 여러 개의 과일(`args`)을 담을 수 있습니다. 각 과일은 인자이며, 바구니에 담긴 과일의 개수는 자유롭게 조절할 수 있습니다.

결론

- `args` 는 함수가 가변적인 수의 인자를 받을 수 있게 해주는 유용한 기능입니다. 별표()는 인자를 묶어서 튜플로 처리할 수 있도록 해주며, 이를 통해 더 유연한 함수 정의가 가능합니다.



`def(* args)` #args 는 계산기 이름 x 값의 주소이기 때문에 *
 for i in args #*은 주소를 뜻하기 때문에 쓰지 않는다. args는 튜플에다가
 {1,2,3,4,5}
 print(i)

→

`def(1,2,3,4,5)`
 →1,2,3,4,5

`def(** kwargs)` **가 2개인이유? 값의 주소랑, 이름주소 즉 2개의 주소를 모두 알아야됨.

print(kwargs)
`def(a=1)`

** 별표 두개는 1. 아파트 동, 2.전체 아파트 이름

`def(name=준기, age=100)`
 ⇒ {name

*args(계산기)

**kwargs(회원가입)할때 씀

주소 1615 계좌,금액

해킹은 레지스터안의 메모리를 건드려서 값을 입력하면 그대로 내가 받아올수 있음(assembly로) 멜트다운?? 스템블리??

c와 C++은 사양..

로스트 추천

C는 가비지 코렉터 없어서 내가 직접 수집해야함

디스코드는 로스트로 바꿈(코렉터가 동적으로 움직임-방이깨끗하면 안돌아감)

** 하나의 주소랑 `def{name=준기,age=100}`

*

드라이버엔지니어 전망좋은 good-연봉1억 컴공 전자과 복전

운영체제의 프로세서 메모리 처서 힘 ~~~공부하기



함수만 알아도 다 짤 수 있음
global(전역변수)는 사용할 일 없음.(메모리를 잡아먹음)



출력부터 짤 어려움
코드는 쉽게 짜야 잘 짜는 거다.



class할 차례(옛날 송춘곤님 강의 한 내용 참고)
파이썬 날개 달기
class란? 객체 생성자
instance 객체
→비유 :class는 빵틀, instance:빵
객체와 instance의 차이는 ??
→ instance즉석으로 빵틀로 찍어내는 것
A
a=A
a는 classA의 instance(0)이다. 객체(object)이다(x)
instance는 관계라는 뜻.. 딱 집은거는 instance
a는 A의 instance이고 a,b,s,는 A,B,C의 객체—→대충 이런개념



객체지향언어는 이처럼 class가 있는 걸 지향한다.(OOP라고 한다.)

OOP-객체지향언어 요즘꺼는 보통 객체지향언어로 만듦

왜 객체지향언어로 만들지?

옛날옛적에

OMR카드 (1970-80)

portran(70년대 미국 주식증권거래가에서 사용하는 언어)/

블룬 버그서버

A → B → C 언어가 있었다. (B는 비주얼 베이직)

이전까지는 객체지향언어(A,B)을 쓰지 않았다.

객체지향언어가 아닌 A,B는 유지보수가 어렵다.

C(OOP)는 유지 보수가 가능, 재사용과 캡슐화가 가능 하다

OOP는 자바 파이썬 오블 이있다.

C++

C++++ = C#



객체지향 언어(Object-Oriented Programming, OOP)로 프로그래밍을 하는 이유는 **코드를 더 효율적이고 구조적으로** 작성할 수 있기 때문입니다. 객체지향 언어는 코드를 ****객체(실세계의 사물 또는 개념)****로 나누어 설계하는 방식으로, 유지보수와 확장이 쉬워지며, 복잡한 프로그램도 체계적으로 관리할 수 있습니다. 객체지향 프로그래밍의 장점을 쉽게 설명하기 위해 비유와 함께 소개할게요.

객체지향 프로그래밍의 주요 장점과 비유

1. 재사용성 (Reusability)

- **비유:** 자동차 부품을 생각해 보세요. 한 번 설계된 엔진, 바퀴, 브레이크 같은 부품은 다른 차에도 재사용할 수 있죠. 마찬가지로, 객체지향에서는 한 번 만든 클래스를 여러 곳에서 재사용할 수 있습니다.
- **설명:** OOP에서는 클래스를 사용해 객체를 만들고, 한 번 작성한 클래스를 다양한 곳에서 재사용할 수 있습니다. 이를 통해 같은 기능을 여러 번 작성하지 않고, 이미 만들어진 코드를 필요할 때마다 쉽게 활용할 수 있습니다.
- **예시:**

```
python
코드 복사
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def start(self):
        return f"{self.make} {self.model}가 시동을 겁니다."

my_car = Car("Hyundai", "Sonata")
print(my_car.start()) # "Hyundai Sonata가 시동을 겁니다."
```

2. 유지보수 용이성 (Maintainability)

- **비유:** 건물 설계를 생각해보세요. 건물의 각 부분이 잘 설계되어 있으면 유지보수가 쉬워지죠. 만약 화장실을 리모델링해야 한다면, 전체 건물을 고칠 필요 없이 해당 부분만 수정하면 됩니다.
- **설명:** OOP에서는 각 객체가 독립적인 역할을 가지고 있기 때문에, 한 부분에서 문제가 생겨도 해당 객체만 수정하면 됩니다. 다른 부분에 영향을 덜 미치기 때문에 유지보수가 용이합니다.
- **예시:** 새로운 차 모델을 만들 때 기존 `Car` 클래스를 수정하지 않고, 필요한 부분만 상속해서 확장할 수 있습니다.

```
python
코드 복사
class ElectricCar(Car):
    def charge(self):
        return f"{self.make} {self.model}가 충전 중입니다."
```

3. 캡슐화 (Encapsulation)

- **비유:** 커피머신을 생각해보세요. 사용자는 커피머신 내부의 복잡한 메커니즘을 알 필요가 없습니다. 버튼을 누르면 커피가 나오는 방식으로 **필요한 기능만 노출**하고 나머지는 숨기는 것이죠.
- **설명:** 캡슐화는 객체 내부의 데이터와 메서드를 외부에서 접근하지 못하도록 숨기고, 필요한 부분만 공개하는 것을 말합니다. 이를 통해 데이터의 무결성을 유지하고, 불필요한 복잡성을 줄일 수 있습니다.
- **예시:** `Car` 클래스에서 엔진의 세부 작동 방식을 외부에 공개하지 않고, `start()` 메서드만 통해서 시동을 걸 수 있게 설계할 수 있습니다.

```
python
코드 복사
class Car:
    def __init__(self, make, model):
        self.__engine_status = "off" # 내부 데이터는 외부에서 접근 불가

    def start(self):
        self.__engine_status = "on"
```

```
return "시동이 걸렸습니다."
```

4. 상속성 (Inheritance)

- **비유:** 부모와 자식을 생각해볼 수 있어요. 부모로부터 자식이 유전적 특성을 물려받듯이, 클래스도 다른 클래스로부터 속성과 메서드를 물려받을 수 있습니다. 이를 통해 **중복 코드**를 줄이고 확장이 가능합니다.
- **설명:** 상속을 통해 기존 클래스의 기능을 그대로 물려받고, 추가 기능을 구현하거나 기존 기능을 변경할 수 있습니다. 이를 통해 더 효율적으로 코드를 확장할 수 있습니다.
- **예시:** 위의 `ElectricCar` 클래스는 `Car` 클래스로부터 `start()` 메서드를 상속받아 새로운 메서드만 추가합니다.

5. 다형성 (Polymorphism)

- **비유:** 동물원에서 "동물"이라는 공통된 개념을 생각해 보세요. 모든 동물이 저마다 다른 소리를 내지만, 결국 **동물**이라는 공통적인 속성을 가지고 있죠. 이처럼 **다양한 형태로 변형**될 수 있지만, 공통적인 인터페이스를 가지고 작동할 수 있습니다.
- **설명:** 다형성은 서로 다른 클래스들이 같은 이름의 메서드를 가지고 있어도, 그 클래스에 맞는 방식으로 동작하게 하는 특성입니다. 이를 통해 **유연한 코드**를 작성할 수 있습니다.
- **예시:** `Car` 와 `ElectricCar` 가 둘 다 `start()` 메서드를 가지고 있지만, 동작은 각각 다를 수 있습니다.

```
python
코드 복사
def start_car(car):
    return car.start()

my_car = Car("Hyundai", "Sonata")
my_electric_car = ElectricCar("Tesla", "Model 3")

print(start_car(my_car))          # "Hyundai Sonata가 시동을 겁니다."
print(start_car(my_electric_car)) # "Tesla Model 3가 시동을 겁니다."
```

odel 3가 충전 중입니다."

객체지향 프로그래밍의 장점 요약

- **재사용성:** 한 번 만든 코드를 여러 곳에서 재사용 가능.
- **유지보수 용이성:** 각 객체가 독립적이므로 수정이 쉬움.
- **캡슐화:** 불필요한 세부 정보를 숨겨 데이터 무결성 유지.
- **상속성:** 중복 코드 없이 기존 클래스를 확장 가능.
- **다형성:** 동일한 인터페이스로 여러 형태의 객체를 처리 가능.

이러한 장점들 때문에 객체지향 프로그래밍은 **큰 프로젝트나 복잡한 시스템**을 설계할 때 매우 유용하며, 코드의 **가독성, 유지보수성, 확장성**을 크게 향상시킵니다.



****CLASS구성인자**

METHOD(함수), 생성자(초기화)→생성자 안에 변수포함되어있음.

CLASS A

def a

def b

def c

def init()→#생성자(내가 초기화할 변수들의 집합)
변수들

def(a(self,a b c)

selfa=3

selfb=4

→

self 라는 것은 (쿠키틀이다)명시적으로 적어줌. 왜냐하면 class의 '틀' 임을 알림.

def 는 믹서기이다. 있는 그대로 →객체이다.

함수는 바로 사용가능하다.

함수는 쿠키를 안만듬.

쿠키들은 객체가 아니다

self는 객체이다.

함수는 있는 그대로도 쓸수있다. (믹서기)

self로 쓰면서 객체 다 로 바뀜

class는

init 생성자→변수들만 들어갈 수 있음

def init()

self.a=0

self.b=0

self.c=0



상속과 오버라이딩의 차이

상속 기능추가

덮어쓰기 지우고 내생각 입힘

—>이름이 같아야 함(메서드)

상속 아무것도 하지 않아도됨

p.s

class는 그냥 못씀

함수는 그냥 쓸 수 있음.

self(너도이제쿠키들이야) class의 틀임을 알린다.



a=ADAS

b

c

d

e

a.c= —>여기서 점(.)은 속성(.)생성자~에 속한다., 접근하다



class 는필요할때만 쓰면 된다. 꼭 쓸 필요 없다.

틀이 정해진 시험에서 많이 쓰인다.

딥러닝에 많이 쓰인다.



그럼 self는 class 자기자신임을 알리는 언어야?

ChatGPT의 말:

ChatGPT

네, 맞습니다! `self` 는 클래스 내부에서 **자기 자신**을 참조하는 특별한 키워드로, **객체 지향 프로그래밍**에서 **클래스의 인스턴스**를 명확하게 지칭하는 역할을 합니다. 이를 통해 클래스 내부의 메서드나 속성에 접근할 수 있도록 해줍니다.

정확한 설명

- `self` 는 객체(인스턴스)의 **고유한 속성**과 **메서드**에 접근할 수 있게 해주며, 현재 메서드가 호출된 인스턴스를 나타냅니다.
- 클래스 안에서 `self` 를 사용함으로써, 해당 메서드가 어떤 인스턴스의 것을 처리하고 있는지를 명확히 알 수 있습니다.

비유를 통해 이해하기

- 예를 들어, 여러 대의 자동차가 있다고 가정해봅시다. 각 자동차는 색상, 모델, 속도 등의 속성을 가집니다. 각 자동차가 자기 자신의 정보를 이야기할 때 `self` 를 사용합니다.
- 자동차가 "나는 빨간색이고, 모델은 소나타야."라고 말할 때, `self.color` 와 `self.model` 을 통해 자신의 속성을 가져오는 것입니다.

실제 예시로 확인해보기

```
python
코드 복사
class Dog:
    def __init__(self, name):
        self.name = name  # 인스턴스의 name 속성에 값 할당

    def bark(self):
        print(f"{self.name}가 멍멍하고 짭니다.")  # self를 사용해 인스턴스의 속성에 접근

# 인스턴스 생성
my_dog = Dog("바둑이")
```



```
my_dog.bark() # "바둑이가 멍멍하고 짖습니다." 출력
```

- 여기서 `my_dog` 는 `Dog` 클래스의 인스턴스입니다. `bark` 메서드에서 `self.name` 을 사용해 현재 객체(`my_dog`)의 이름에 접근하여 **바둑이**의 이름을 출력합니다.

정리

- `self` 는 클래스의 **인스턴스 자신을 참조**하는 키워드로, 클래스 내부에서 자신의 속성과 메서드를 인식하고 관리하는 데 사용됩니다.
- 이는 객체 지향 프로그래밍에서 중요한 개념으로, 각 객체가 자신의 상태를 독립적으로 유지하고, 메서드를 통해 해당 상태를 조작할 수 있게 해줍니다.

결론적으로, `self` 는 클래스의 **자기 자신을 알리는 역할**을 하며, 인스턴스의 속성과 메서드에 접근할 수 있는 중요한 도구입니다!



—>이건 개인적으로 유튜브 영상 좀 참조하려고

과제:: 계산기만들기(주말동안)

문제 1: 기본 계산기 클래스 구현

`Calculator` 클래스를 만드세요. 이 클래스는 기본적인 산술 연산을 제공해야 합니다.

요구사항:

1. 다음 메서드를 구현하세요:

- `add(*args, **kwargs)` : 덧셈
- `subtract(*args, **kwargs)` : 뺄셈
- `multiply(*args, **kwargs)` : 곱셈
- `divide(*args, **kwargs)` : 나눗셈

2. 각 메서드는 위치 인자(`args`)와 키워드 인자(`*kwargs`)를 받아야 합니다.

3. `*kwargs`에는 다음 키를 사용할 수 있어야 합니다:

- `precision` : 결과의 소수점 자릿수 지정 (기본값: None, 즉 반올림하지 않음)
- `return_float` : True일 경우 항상 float 타입 반환, False일 경우 가능하면 int 반환 (기본값: False)

4. 0으로 나누기 등의 에러 상황을 적절히 처리해야 합니다.

예시 사용법:

```
calc = Calculator()
print(calc.add(1, 2, 3, precision=2)) # 출력: 6.00
print(calc.subtract(10, 2, 3, return_float=True)) # 출력:
5.0
print(calc.multiply(2, 3, 4)) # 출력: 24
print(calc.divide(100, 2, precision=3)) # 출력: 50.000
```

문제 2: 공학용 계산기 클래스 구현

`Calculator` 클래스를 상속받아 `EngineeringCalculator` 클래스를 만드세요. 이 클래스는 기본 계산기의 기능을 모두 포함하면서 추가적인 공학 계산 기능을 제공해야 합니다.

요구사항:

1. `Calculator` 클래스의 모든 메서드를 상속받으세요.
2. 다음 새로운 메서드를 추가하세요:

- `square_root(x, **kwargs)` : 제곱근
- `power(x, y, **kwargs)` : 거듭제곱
- `log(x, base=10, **kwargs)` : 로그 (기본값은 상용로그)
- `ln(x, **kwargs)` : 자연로그
- `sin(x, **kwargs)` : 사인
- `cos(x, **kwargs)` : 코사인
- `tan(x, **kwargs)` : 탄젠트

3. 모든 메서드는 `*kwargs` 를 통해 `precision` 과 `return_float` 인자를 받아야 합니다.
4. `divide` 메서드를 오버라이드하여, 0으로 나누려고 할 때 사용자 정의 예외 `DivisionByZeroError` 를 발생시키세요.
5. 각 삼각함수 메서드에 `angle_unit` 키워드 인자를 추가하여 'degree' 또는 'radian' 단위로 입력을 받을 수 있게 하세요. 기본값은 'radian'으로 설정하세요.

예시 사용법:

```
eng_calc = EngineeringCalculator()
print(eng_calc.add(1, 2, 3, precision=2)) # 출력: 6.00
print(eng_calc.square_root(16, precision=3)) # 출력: 4.000
print(eng_calc.log(100, precision=4)) # 출력: 2.0000
print(eng_calc.sin(30, angle_unit='degree', precision=4))
# 출력: 0.5000
try:
    print(eng_calc.divide(5, 0))
except DivisionByZeroError as e:
    print(e) # 출력: "Division by zero is not allowed"
```

추가 과제:

1. 타입 힌팅을 사용하여 모든 메서드와 함수의 입력 및 출력 타입을 명시하세요.
2. 두 계산기 클래스에 대한 간단한 문서화를 작성하세요. (클래스, 메서드, 예외 등)