

# 강의 1-2

## 함수 1, 객체와 객체 지향 1

Python 입문 과정 3주 프로그램

한날, 2024-03-07, copyright RealWorldPudding all rights reserved.



PuddingCamp

<https://pudding.camp>

# 테스트 자동화



# pytest 패키지 설치

## vscode 터미널 패널에서 진행

- .venv 디렉터리가 있는 경로에서 다음 명령어 실행.
- `VirtualEnv` 활성화(진입)
- `pip install pytest`



# vscode 설정 추가

<https://pudding.camp/~9c43fc8b>

- `.venv` 디렉터리가 있는 경로에 `.vscode` (점 브이에스코드) 디렉터리 생성
- `.vscode` 디렉터리 안에 `settings.json` 파일 생성
- `settings.json` 파일에 다음 내용 입력

```
{  
    "python.defaultInterpreterPath": "./.venv/bin/python",  
    "python.testing.cwd": "./",  
    "python.testing.unittestEnabled": false,  
    "python.testing.pytestEnabled": true,  
    "pythonTestExplorer.testFramework": "pytest",  
    "python.testing.pytestPath": "./.venv/bin/pytest",  
    "python.testing.pytestArgs": [  
        "src"  
    ],  
}
```



# vscode 설정 추가

<https://pudding.camp/~9c43fc8b>

- `.venv` 디렉터리가 있는 경로에 `pytest.ini` 파일 생성
- `pytest.ini` 파일에 다음 내용 입력

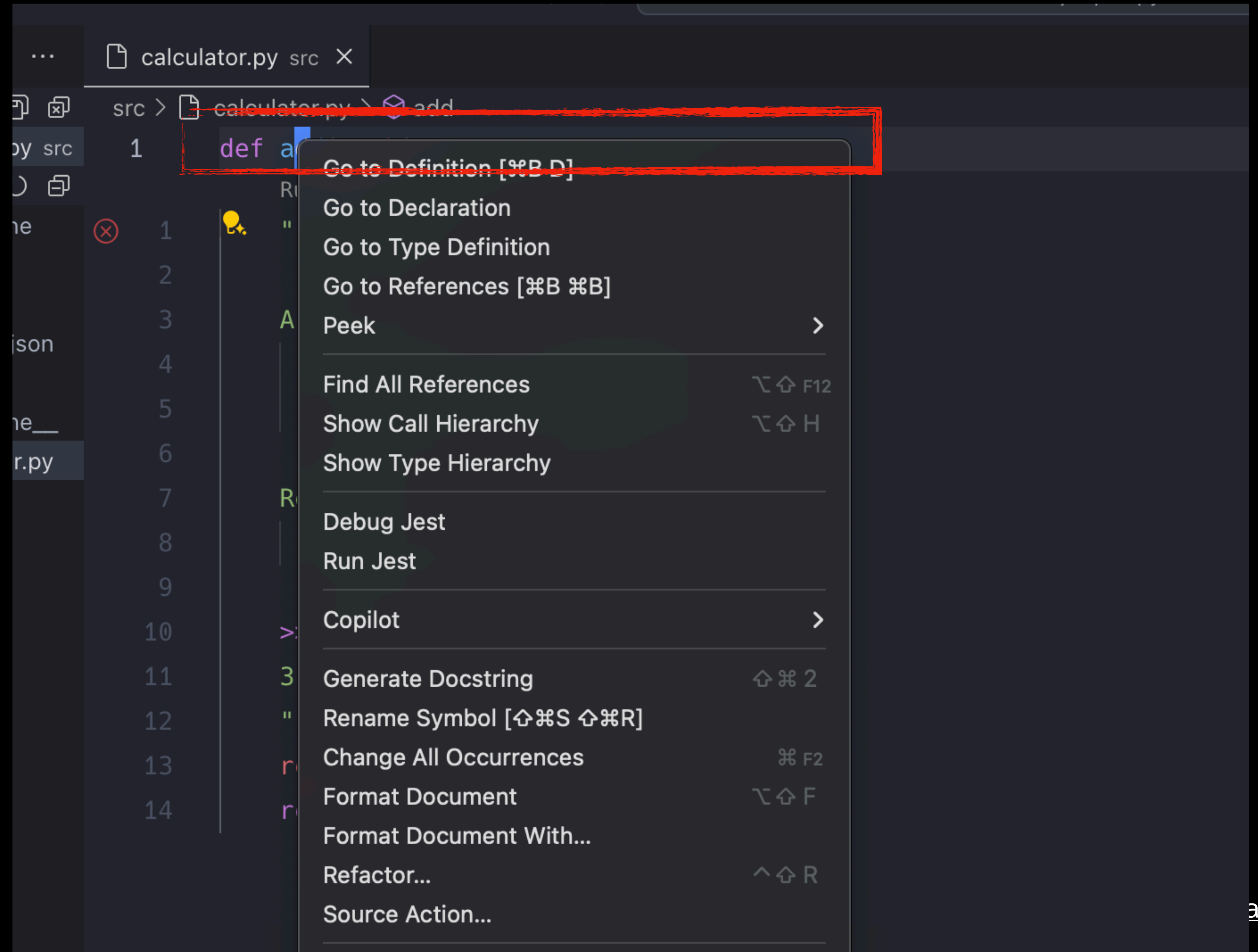
```
[pytest]
addopts = --doctest-modules --doctest-report ndiff
doctest_optionflags = NORMALIZE_WHITESPACE IGNORE_EXCEPTION_DETAIL
```



# vscode에서 테스트하기

## 테스트 실행

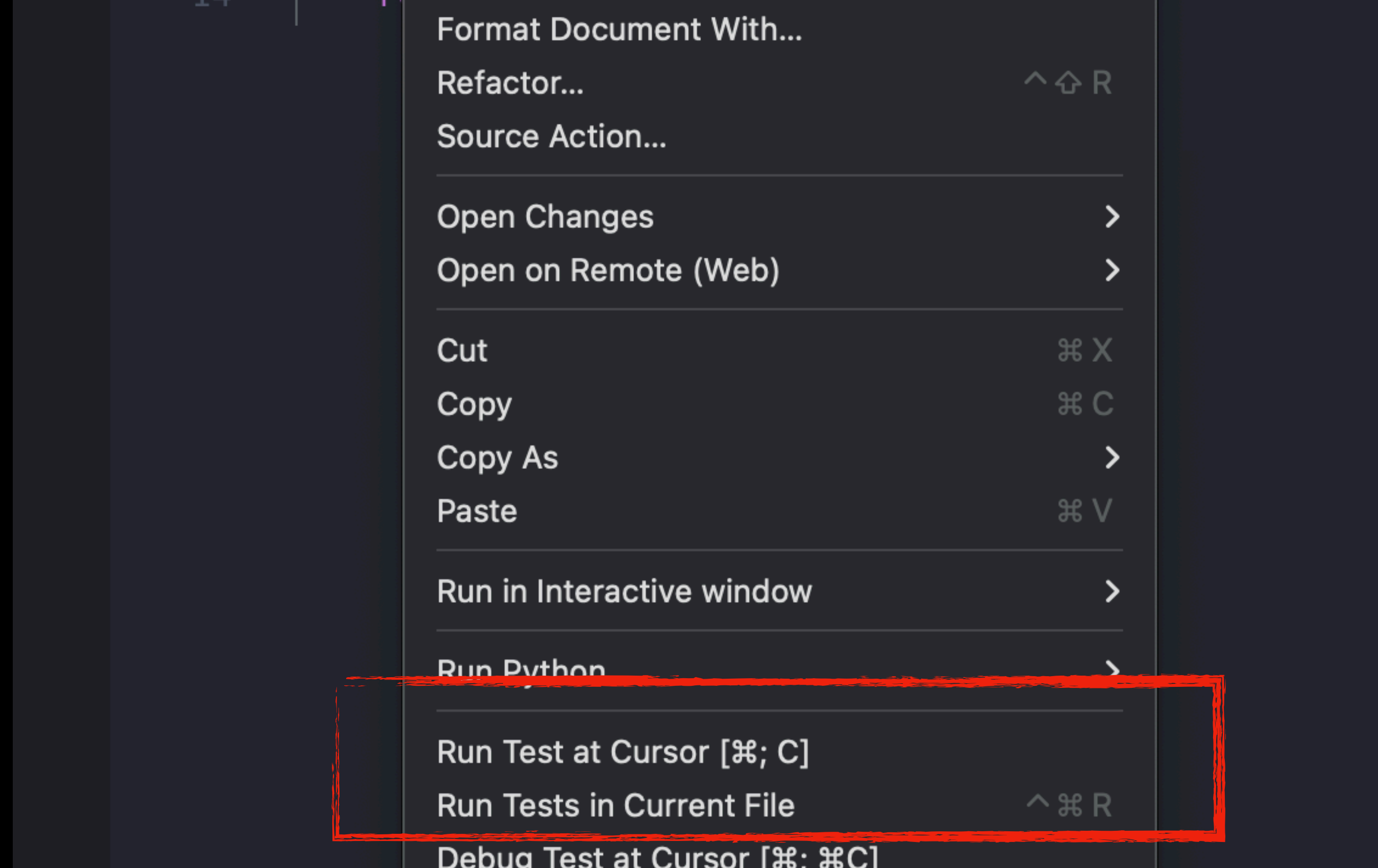
- 함수 몸통 아무데나 클릭
- 그 상태에서 우클릭하여  
컨텍스트 메뉴 열기



# vscode에서 테스트하기

## 테스트 실행

- 컨텍스트 메뉴에서  
“Run Test at Cursor” 클릭.  
(커서 위치에서 테스트 실행)
- vscode 설정에 문제가 있거나  
python 코드에 문제가 있는 경우  
python 코드가 인식되지 않아 Run Test at Cursor 메뉴가 나타나지 않을 수 있습니다.





# vscode에서 테스트하기

## 테스트 결과 보기

- 커맨드 팔레트에서 “test results”로 검색
- Focus on Test Results View 선택
- 테스트 결과 확인

```
CLIENT: Server listening on port 52313
Received JSON data in run script
Running pytest with args: ['-p', 'vscode_pytest', '--rootdir=/Users/hannal/Workspace/realworldpudding/puddingcamp-practices/pre-python-3weeks', '/Users/hannal/Workspace/realworldpudding/puddingcamp-practices/pre-python-3weeks/src/calculator.py::calculator.add']

===== test session starts =====
=====
platform darwin -- Python 3.10.12, pytest-8.0.2, pluggy-1.4.0
rootdir: /Users/hannal/Workspace/realworldpudding/puddingcamp-practices/pre-python-3weeks
configfile: pytest.ini
collected 1 item

src/calculator.py .
[100%]

===== 1 passed in 0.01s =====
=====
Finished running tests!
```

Test Results Sidebar:

- ✓ Test run at 2/29/2024, 7:20:...
  - ✓ calculator.add
- ✗ Test run at 2/29/2024, 7:07:25 PM
  - ✗ calculator.add
- ✗ Test run at 2/29/2024, 7:07:23 PM
  - ✗ calculator.add
- ✓ Test run at 2/29/2024, 7:07:00 PM
  - ✓ calculator.add
  - Test run at 2/29/2024, 7:02:27 PM
  - Test run at 2/29/2024, 7:02:25 PM

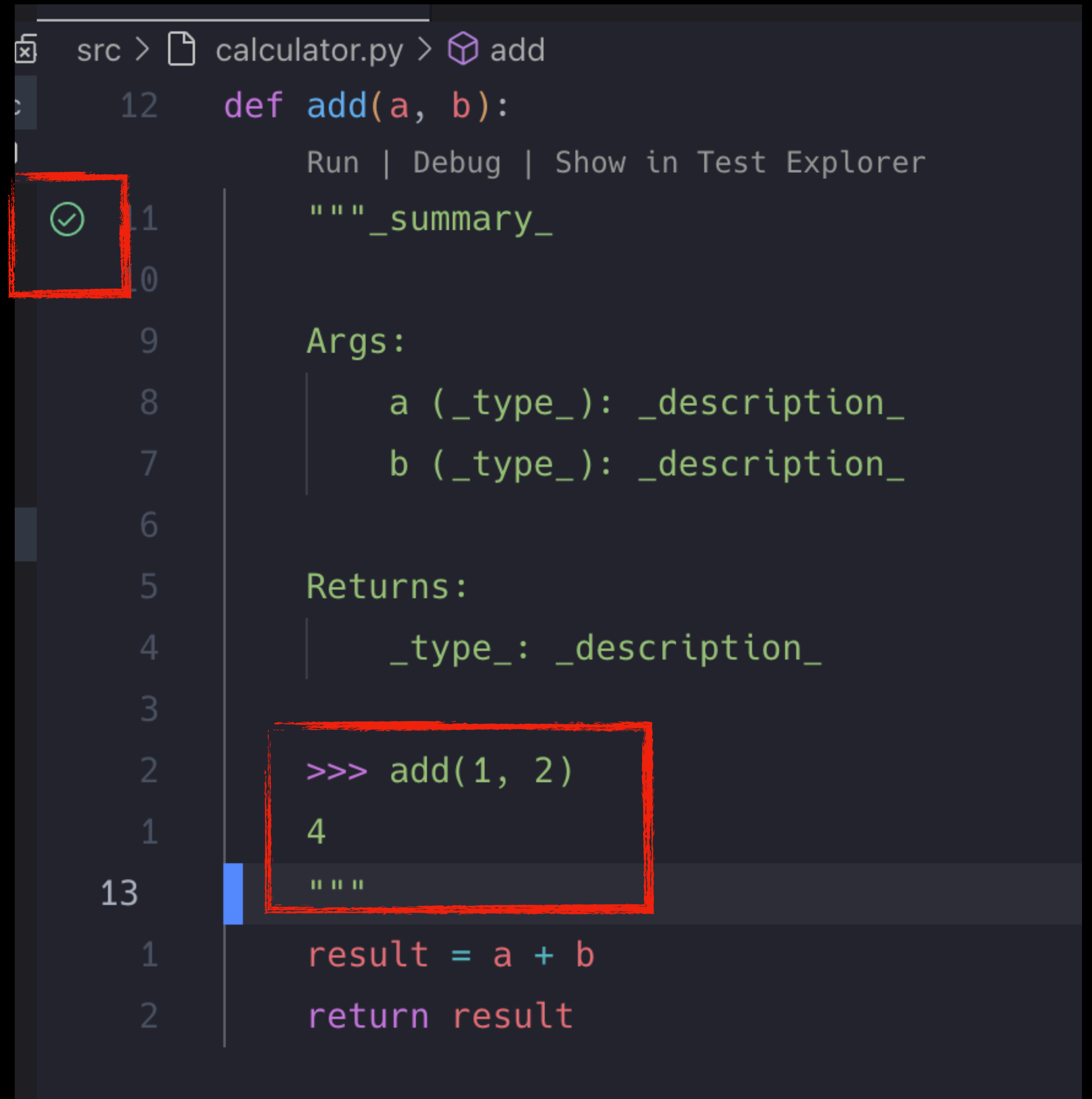




# vscode에서 테스트하기

## 테스트 실패 해보기

- 커맨드 팔레트에서 “test results”로 검색
- Focus on Test Results View 선택
- 테스트 결과 확인



The screenshot shows the VS Code interface with a file named 'calculator.py' open. The file contains a Python function 'def add(a, b):' and its implementation. A green checkmark icon in the left margin is highlighted with a red box. Below the function, a terminal window is open, showing the command '>>> add(1, 2)' and the output '4'. The terminal window is also highlighted with a red box.

```
src > calculator.py > add
12 def add(a, b):
    Run | Debug | Show in Test Explorer
    """_summary_

    Args:
        a (_type_): _description_
        b (_type_): _description_

    Returns:
        _type_: _description_

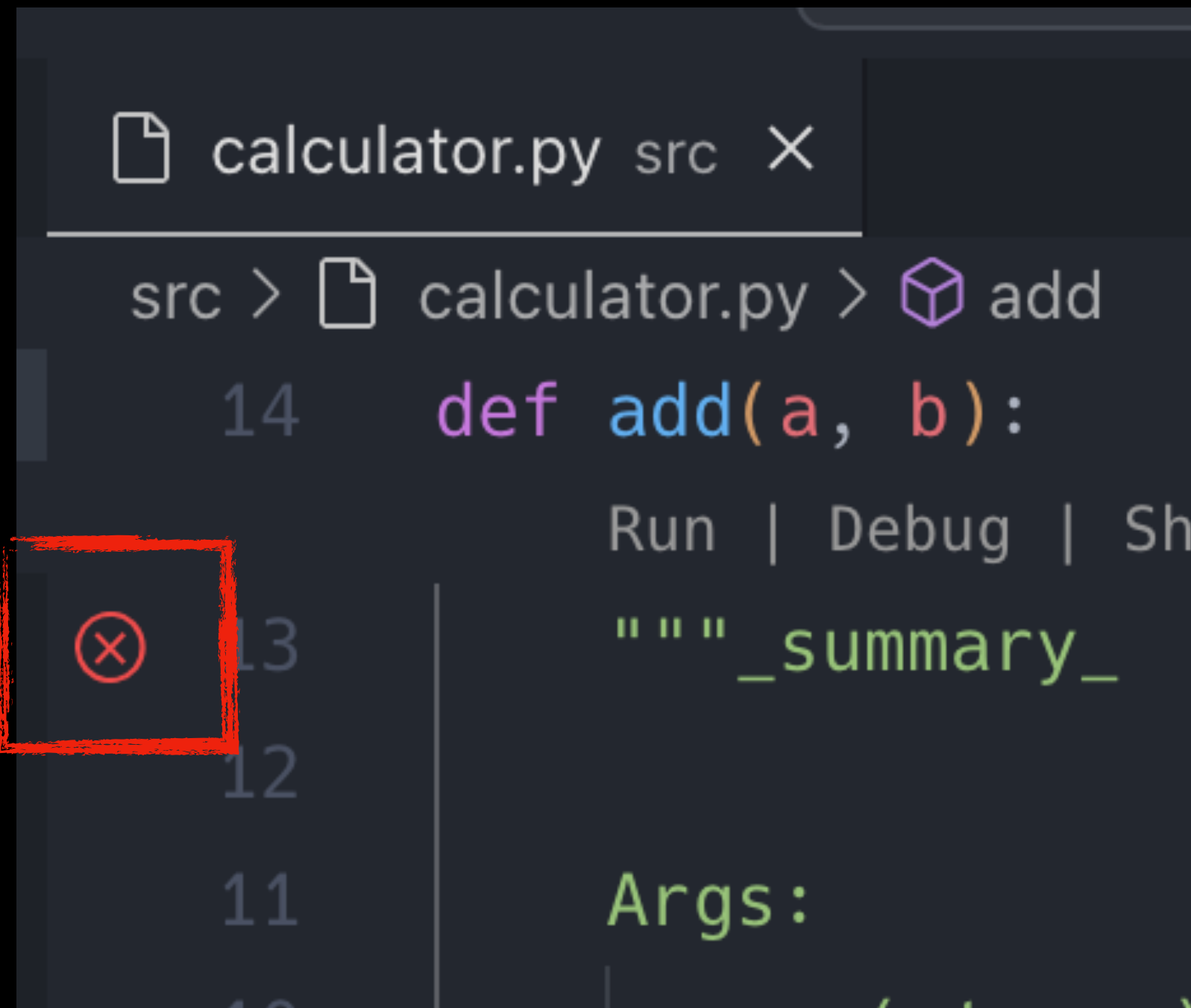
>>> add(1, 2)
4
"""

13
1 result = a + b
2 return result
```

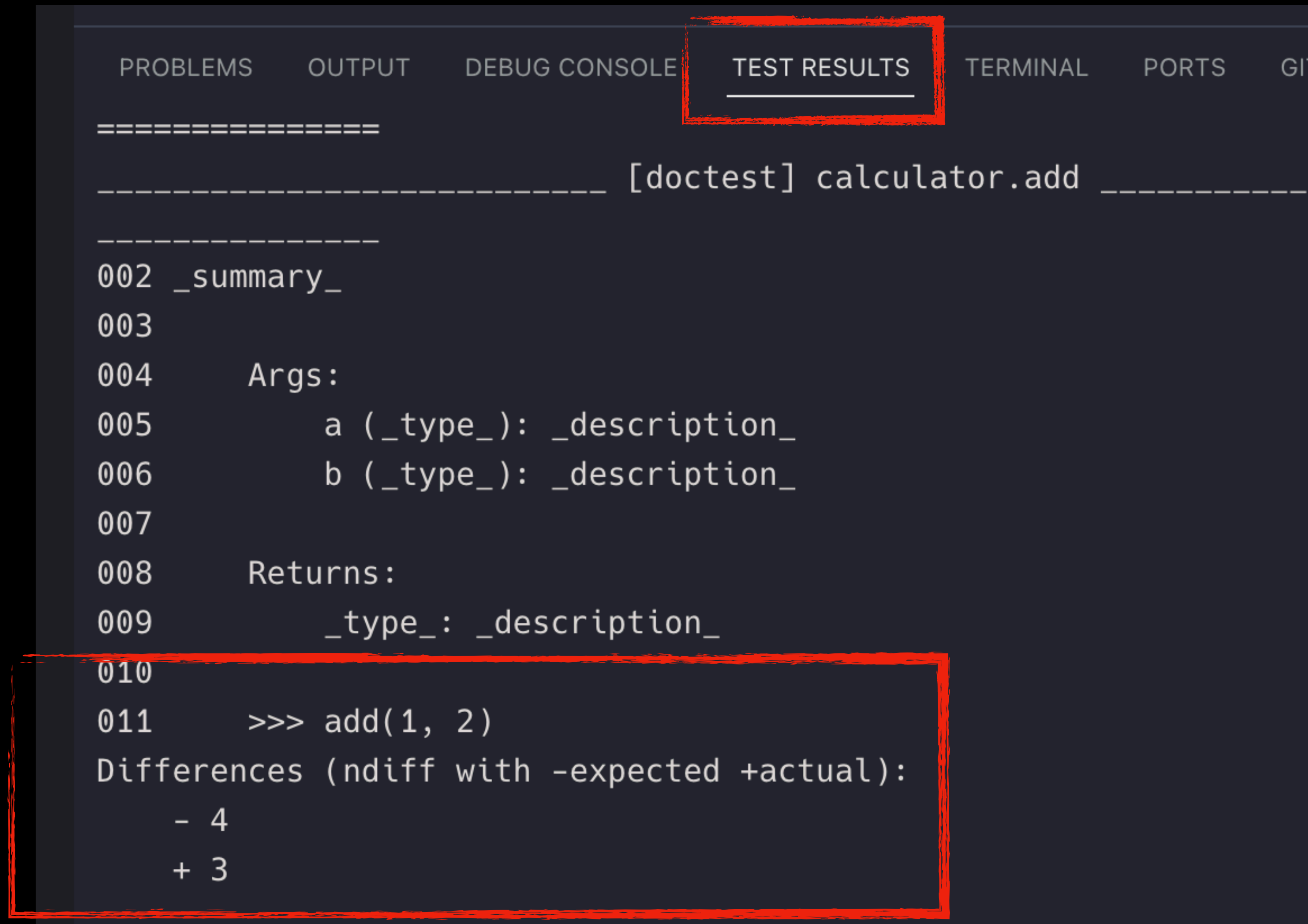


# vscode에서 테스트하기

## 테스트 실패 결과 보기



```
calculator.py src X
src > calculator.py > add
14 def add(a, b):
    Run | Debug | Sh
13 """_summary_
12
11 Args:
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TEST RESULTS TERMINAL PORTS GI
=====
[doctest] calculator.add
-----
002 _summary_
003
004 Args:
005     a (_type_): _description_
006     b (_type_): _description_
007
008 Returns:
009     _type_: _description_
010
011 >>> add(1, 2)
Differences (ndiff with -expected +actual):
- 4
+ 3
```



# 실습

# 실습 20240307-1-2-1

코드, 실행, src 디렉터리 안에 func.py

```
def say(message):  
    print(message)
```

```
say("hello world")
```



# 실습 20240307-1-2-2

코드, 실행, src 디렉터리 안에 func2.py

```
def super_calculator(a: int, b: int) -> int | float:  
    """짱 멋진 계산기
```

Args:

a (int): 첫 번째 숫자  
b (int): 두 번째 숫자

Returns:

int | float : 계산 결과

"""

```
result = a / b  
return result
```



# 실습 20240307-1-2-2

## 실행. Python REPL

```
result = super_calculator(4, 2)  
print(result)
```

```
result = super_calculator(32, 8)  
print(result)
```

```
result = super_calculator(149_597_870_700, 299_792_458)  
print(result)
```

# 이름짓기 규칙





# 객체 이름짓기(Naming) 규칙

- 객체에 붙이는 이름에 사용할 수 있는 기호는 밑줄(\_) 뿐.
- 또한, 숫자로 시작할 수 없다.



# 이름짓기 (Naming) 관례와 스타일

- `class` : PascalCase
    - `protected-like` : `_snake_case`
    - `private-like` : `__snake_case`
  - special attribute
    - `__snake_case__()` : double under, dounder
    - `__init__()`, `__str__()`, `__add__()`, ...
  - `function`, `variable` : `snake_case`
  - `constant` : UPPER\_CASE
- } 관례 (convention)
- } 문법은 아니지만,  
문법에 가까운 관례
- } 관례 (convention)

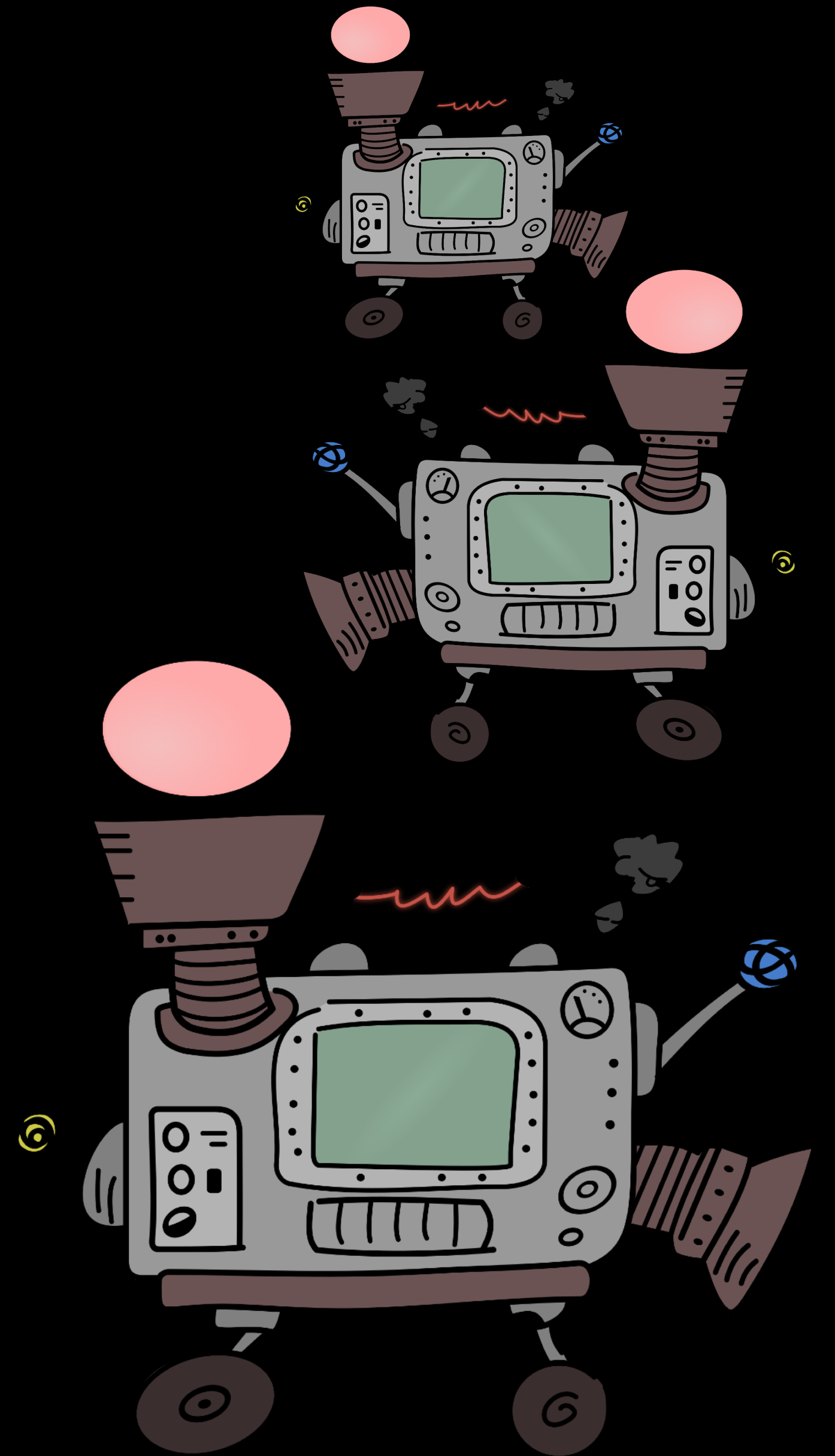


# 함수

# 함수란?

- 정의 : 일을 수행하는 명령들을 묶고, 그 명령 묶음을 필요할 때마다 호출하여 수행하는 단위.
- 동작 : 입력을 받으며 시작해서 출력(결과 반환)하며 끝난다.

- 함수 자본주의\_댄스(돈\_액수)  
만약 돈\_액수 < 1,000,000 이면  
    return False  
  
초 = 10 - (돈\_액수 / 1,000,000)  
엉덩이.흔든다(왕복2회, 초)  
오른팔.파닥인다(6회, 초)  
왼팔.그타원을\_그린다(시계방향, 2회, 초)  
  
return True



# 함수 관련 문법

정의(definition) 예약어

```
def say(message):  
    print(message)
```

```
say("hello world")
```



# 함수 관련 문법

호출(call, invoke)부 정의

```
def say(message):  
    print(message)
```

```
say("hello world")
```



# 함수 관련 문법

함수 본문. Block scope, 들여쓰기

```
def say(message):
```

```
    print(message)
```

```
say("hello world")
```





# 함수 관련 문법

블록 스코프 시작.

```
def say(message):  
    print(message)
```

```
say("hello world")
```



# 함수 관련 문법

호출부로 호출

```
def say(message):  
    print(message)
```

```
say("hello world")
```



# 함수 관련 문법

## docstring

```
def super_calculator(a: int, b: int) -> int | float:
    """짱 멋진 계산기

    Args:
        a (int): 첫 번째 숫자
        b (int): 두 번째 숫자

    Returns:
        int: 계산 결과
    """
    result = a / b
    return result
```



# 함수 관련 문법

## docstring

```
In [6]: print(dir.__doc__)  
dir([object]) -> list of strings
```

If called without an argument, return the names in the current scope.  
Else, return an alphabetized list of names comprising (some of) the attributes of the given object, and of attributes reachable from it.  
If the object supplies a method named `__dir__`, it will be used; otherwise the default `dir()` logic is used and returns:

- for a module object: the module's attributes.
- for a class object: its attributes, and recursively the attributes of its bases.
- for any other object: its attributes, its class's attributes, and recursively the attributes of its class's base classes.

```
In [7]: █
```

```
dir()
```

```
(function) def dir(  
    __o: object = ...,  
    /  
    ) -> list[str]
```

```
In [4]: dir(dir)  
Out[4]:  
['__call__',  
 '__class__',  
 '__delattr__',  
 '__dir__',  
 '__doc__',  
 '__eq__',  
 '__format__',  
 '__ge__',  
 '__getattr__',  
 '__getstate__',  
 '__gt__',  
 '__hash__',  
 '__init__',  
 '__init_subclass__',  
 '__le__',  
 '__lt__',  
 '__module__',  
 '__name__',  
 ...]
```



# 함수 관련 문법

## 인자

```
def super_calculator(a: int, b: int) -> int:
```

- 인자란, 함수 호출자가 함수를 호출하며 함수에 필요한 값을 전달하는데, 이 값을 인자라 함.
- 함수 선언부의 괄호 안에 인자(변수) 이름을 나열하며, 쉼표로 구분.
- Python 객체는 무엇이든지 인자로 전달 가능.
- Python에서 모든 데이터는 객체.



# 함수 관련 문법

## 자료형 각주 (Type annotation)

```
def super_calculator(a: int, b: int) -> int:
```



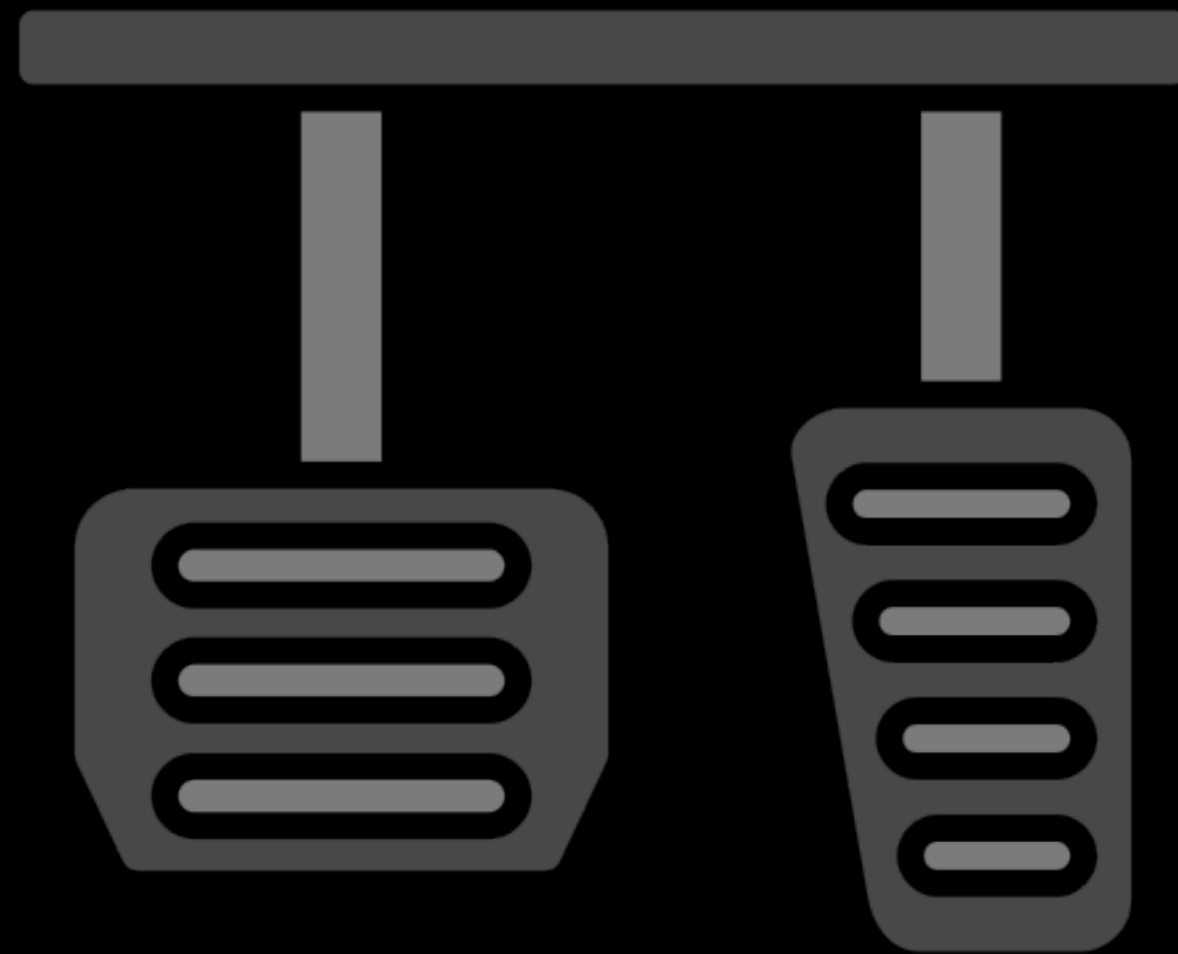
# 인터페이스



# 어?!

## 이거 왜 이래?

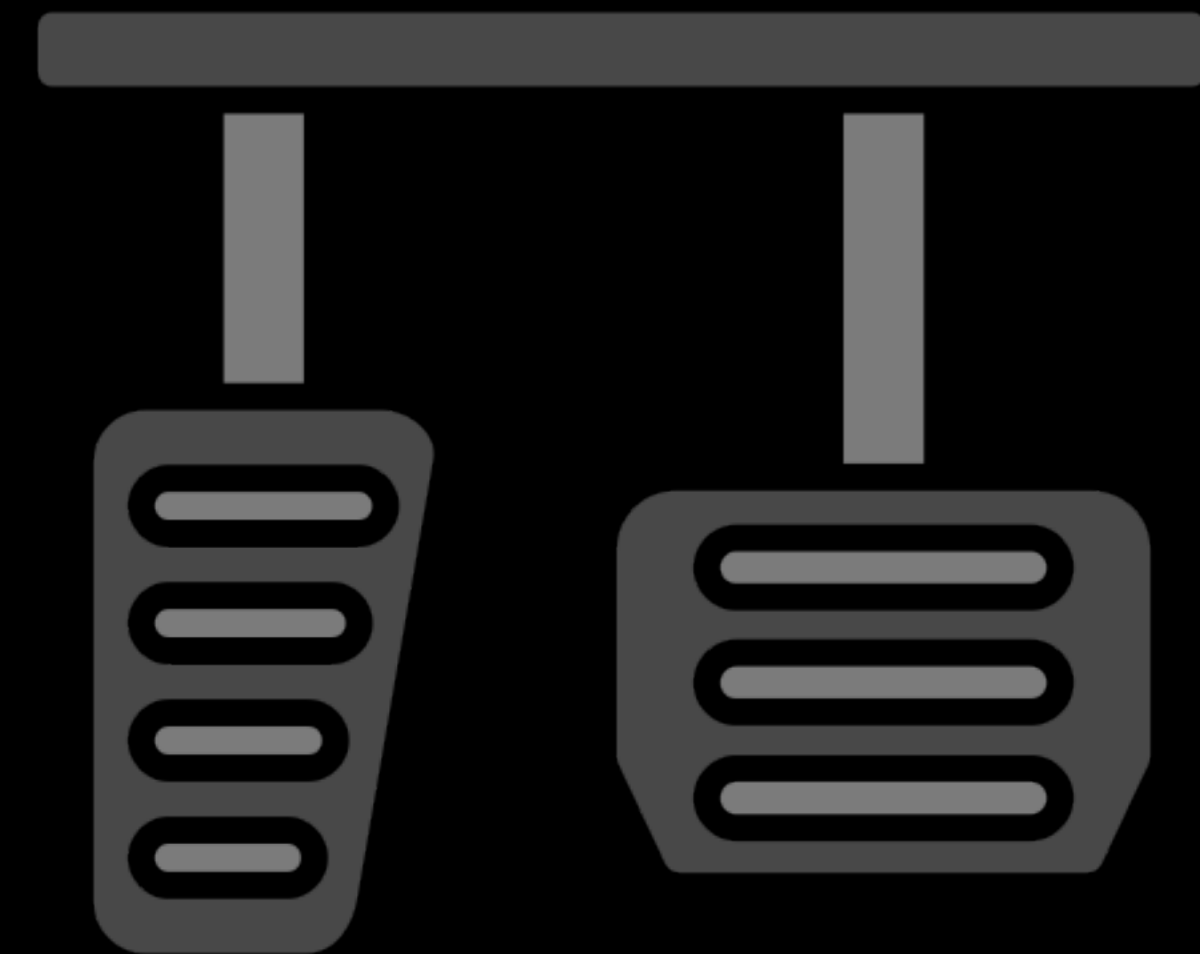
기대 인터페이스



제동

가속

기대를 깨는 인터페이스



가속

조건부 제동



# 실습

# 실습-20240307-1-2-4 : 계산기 만들기

파일명 : **src/calculator.py**

- 더하기 함수, **add** : +
- 빼기 함수, **sub** : -
- 곱하기 함수, **mul** : \*
- 나누기 함수, **div** : /

- docstring 작성
- 계산 결과를 result 변수에 담은 뒤 result 변수를 반환

```
def func(a, b):  
    """func description  
  
    Args:  
        a (int): 인자 설명  
        b (int): 인자 설명  
  
    Returns:  
        int: 반환값 설명  
    """  
    result = ...  
    return result
```



# 실습-20240307-1-2-4 : 계산기 만들기

Visual Studio Code 터미널에서 실행하세요.

- `vscode`

- 커맨드 팔레트

- Terminal : Focus Terminal

- `terminal`

- `python`

- `import calculator as cal`

- `cal.add(1, 2)`

다음 두 개로 짝지어진 숫자들을 네 개 함수에 전달해 실행해보세요.

- 1, 2

- 149597870700, 299792458

- 10, 3

- 5, 2



# 실습-20240307-1-2-4 : 계산기 만들기

## 계산을 정상 처리하는지 확인하기

- python REPL에서 다음과 같이 계산 함수들을 실행해보세요.
- `assert add(1, 2) == 3`
- `assert mul(1, 2) == 13`
- `assert sub(10, 3) == 5`
- `assert div(10, 5) == 2`



# 객체 지향

Object-Oriented

# 객체 지향 프로그래밍 (OOP)

## Object Oriented Programming

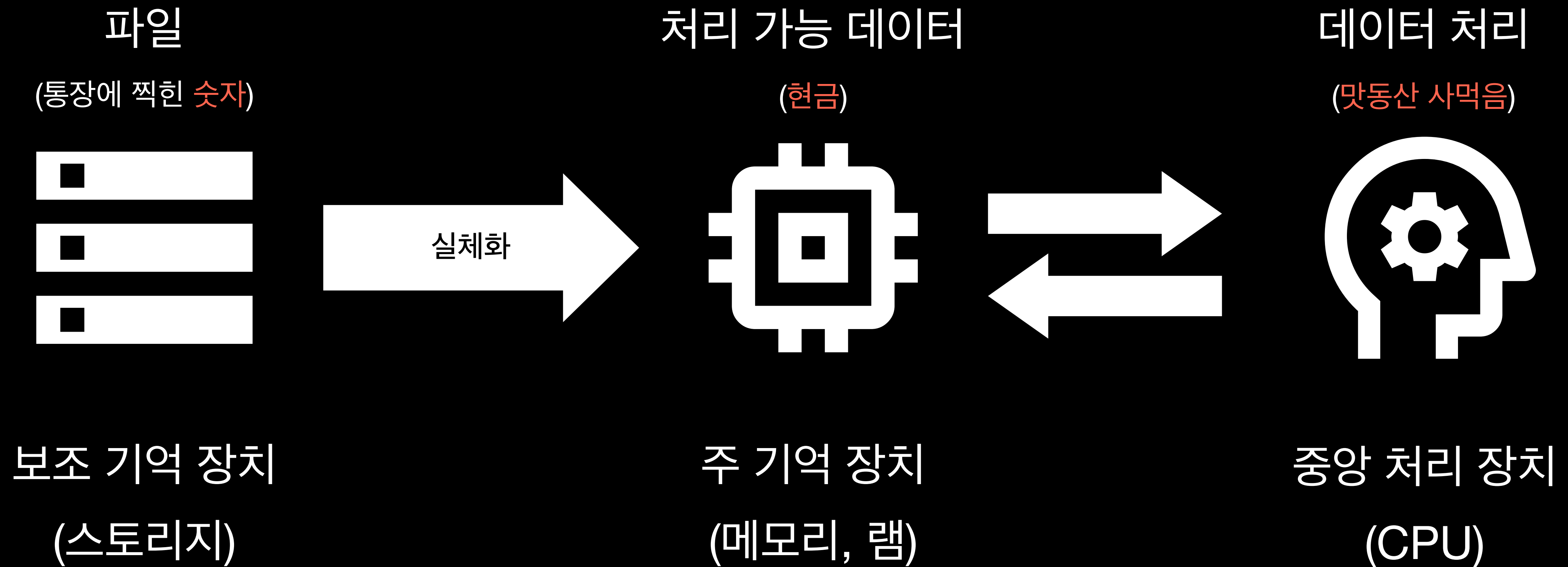
- 객체가 메시지를 주고 받으며 협업하는 패러다임.
- **한날**이 메신저에서 “윤상섭, 과제, 피드백”이라 적은 **메시지**를 **유남주**에게 보낸다.
- **유남주**가 메신저에서 “message failed to send”라고 적은 **메시지**를 **한날**에게 보낸다.





# 객체

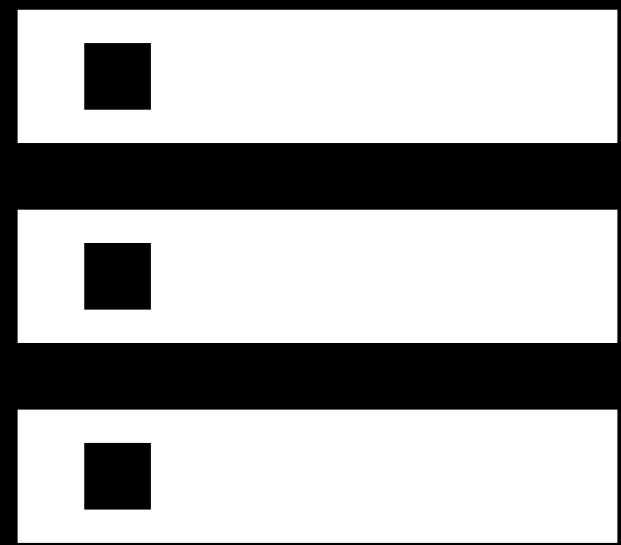
# 프로그램이 실행되어 동작하는 과정



# Python의 모든 데이터는 객체

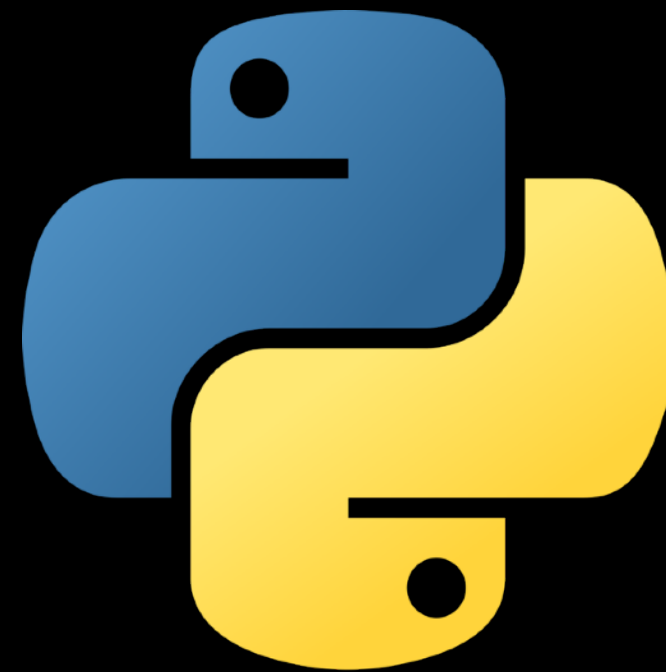
예약어 등 문법 요소를 제외하면 모두 객체

파일  
(puddingcamp.py)



보조 기억 장치  
(스토리지)

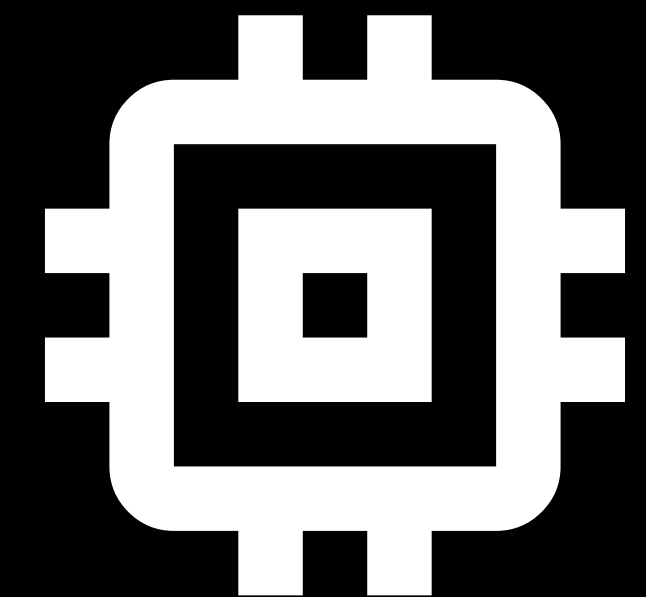
추상화



프로그래밍 언어  
(Python)

실체화(구체화)

처리 가능 데이터  
(python object)



주 기억 장치  
(메모리, 램)



# Python의 모든 데이터는 객체

- 객체 : 물질
- 메시지 : 상호작용
- 언어 문법 : 물리학 법칙
- 물질은 모두 같은 물리학 법칙을 따른다.
  - 물질은 전자기력의 반발로 서로를 통과하지 못한다.
  - 물질의 시간 차원은 한 쪽 방향으로 흐른다.
  - 질량이 있는 물질은 시공간을 휘게 한다.



# 객체의 세 가지 특성

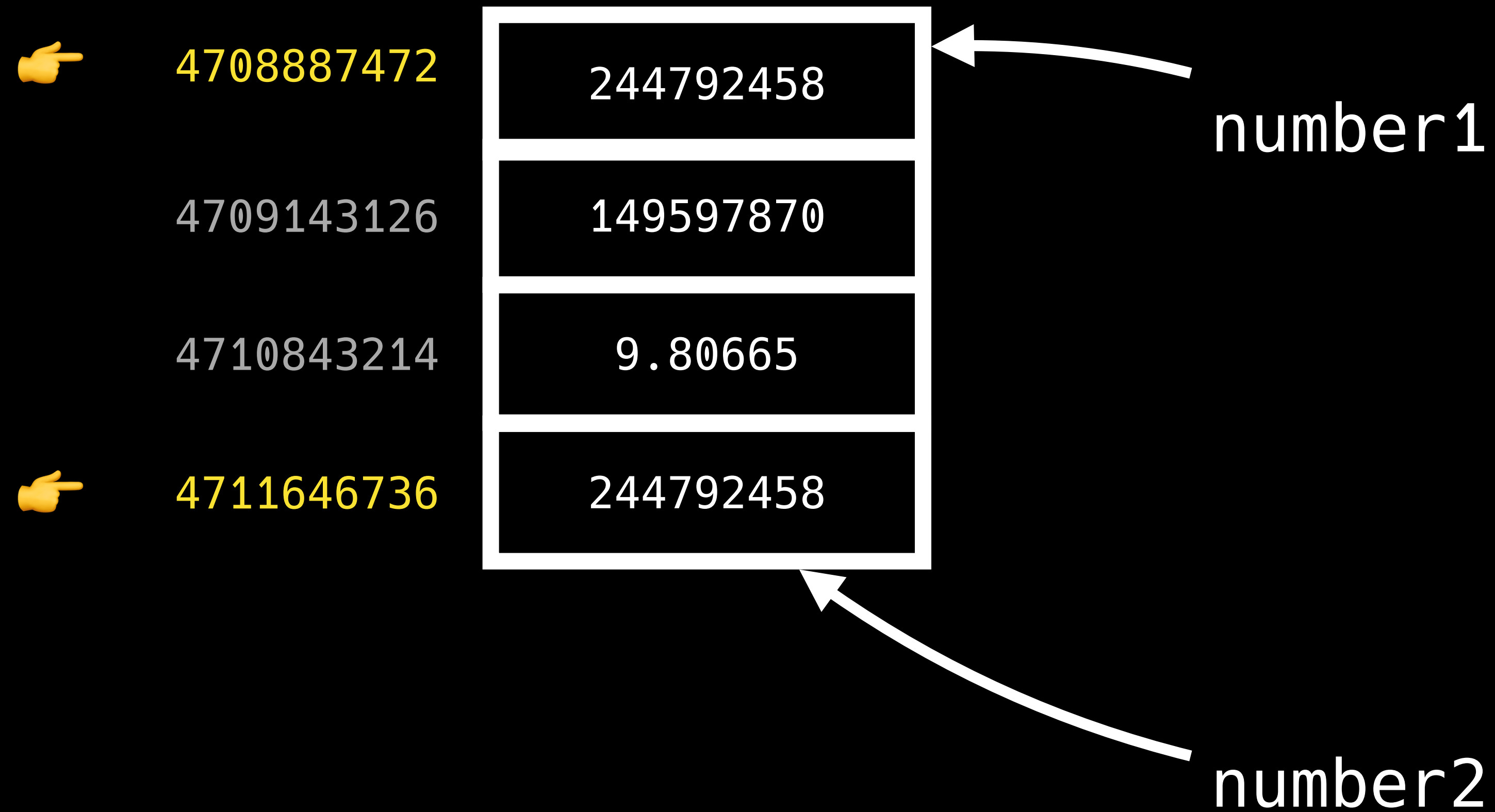
- 유형(type)
- 정체성(identity)
- 값(value)

```
number = 299_792_458
```

- 유형 : `int` class
- 값 : 299792458
  - 코드를 평가(evaluate)하면 그 자체가 객체
  - literal



# 객체의 정체성 (identity)



```
In [1]: number = 299_792_4
```

```
In [2]: id(number)  
Out[2]: 4708887472
```

```
In [3]: number2 = 29979245
```

```
In [4]: id(number2)  
Out[4]: 4711646736
```

```
In [5]: █
```



# 객체 비교 방식 두 가지

## 정체성과 값 비교

- 값 비교 : `number1 == number2`

- 정체성 비교

```
number1 is number2
```

```
id(number1) == id(number2)
```

```
** True, False, None, class **
```

이 셋과 개별 클래스는 Python 프로세스 내에서 유일한 객체이므로 `==` 비교와 `is` 비교 모두 동일하게 평가된다.

하지만, 관례와 (미~~세한) 성능 차이로 `is` 비교를 한다.



# 실습





# 실습 : 20240307-1-2-5

## 객체 정체성 확인 : Python REPL 에서 진행

- `super_calculator()` 함수의 `docstring(__doc__)`의 `id()` 확인
- `number1`과 `number2`에 각각 다음 숫자 넣고, 정체성과 값 비교.
  - 299\_792\_458, 149\_597\_870\_700, 1, 256, 257
- 다음 방식으로 정체성 비교
  - `number1 = number2 = 숫자`



# 객체의 구성 요소

# 속성 (Attribute)

객체를 이루는 요소

- 속성 : 멤버 변수 + 메서드
- 멤버 변수 (member variables, ...) : 객체에 속성으로써 붙어있는 변수
- 메서드 (method) : 객체에 속성으로써 붙어있는 함수



# 변수



# 실습 : 20240307-1-2-3

- Python REPL(Read-Evalute-Print Loop) 실행
- puddigcamp 라고 입력하고 엔터

# 실습 : 20240307-1-2-3

```
>>> pudddigcamp
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pudddigcamp' is not defined
>>> █
```



# Python에서 변수란?

객체에 **이름** 붙인 단위(개체)

- 객체 : 메모리에 할당된 데이터
  - XX도 YY시 ZZZ로 WWW VVVV호에 있는 한날
- 이름 : 객체에 접근하기 위한 수단
- `working_hannal = Hannal(mode="work")`
  - `working_hannal` : 이라는 이름을
  - `=` : 붙인다
  - `Hannal(mode="work")` : 이렇게 실체화된( instantiated ) 객체에.



# 어떤 이름이든 가리키는 대상은 이름이 붙은 객체





# 관측되지 않으면 사라질 수도 있는 미시 세계

참조 (reference)

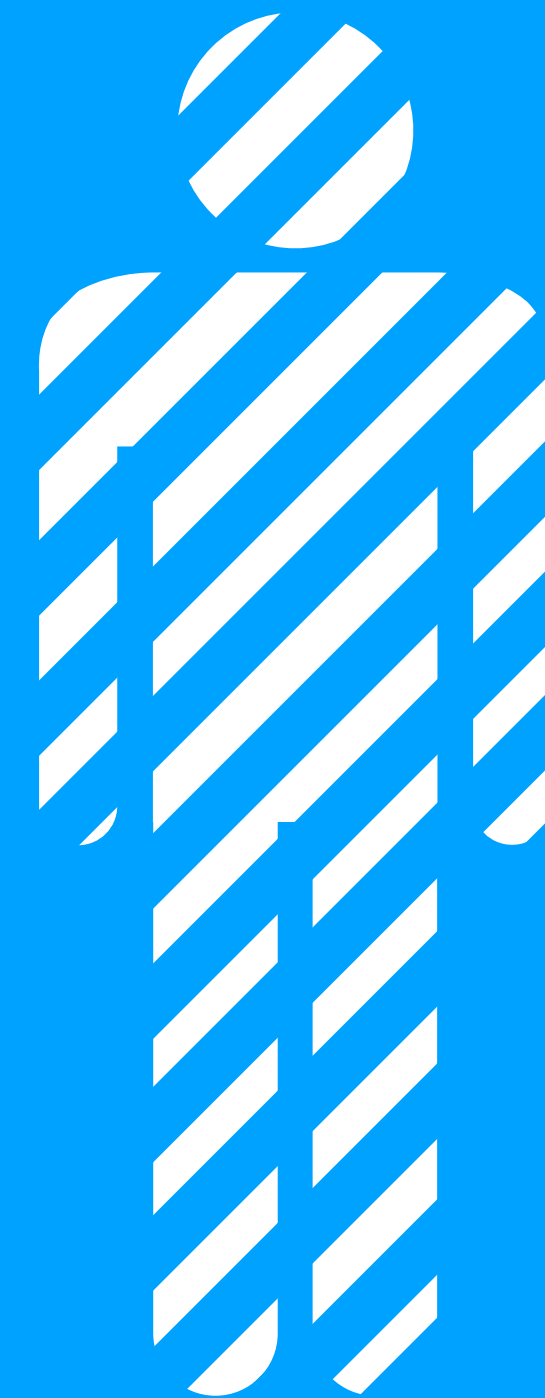
한날

Cha

아빠

차

아들



# 객체 다루기



# 기본적으로 변수와 함수로 구성된 개체

- 멤버 변수는 변수 다루듯이
- 메서드는 함수 다루듯이
- 객체 이름에 마침표(.)를 찍고, 마침표 뒤에 속성 이름을 덧붙이는 차이

```
def work(person_name, hour):  
    ...  
    return True
```

```
name = "Hanna"  
working_hour = 8  
work(name, working_hour)
```

```
kay = Hannal(mode="work")  
kay.name = "Hanna"  
kay.work(8)
```

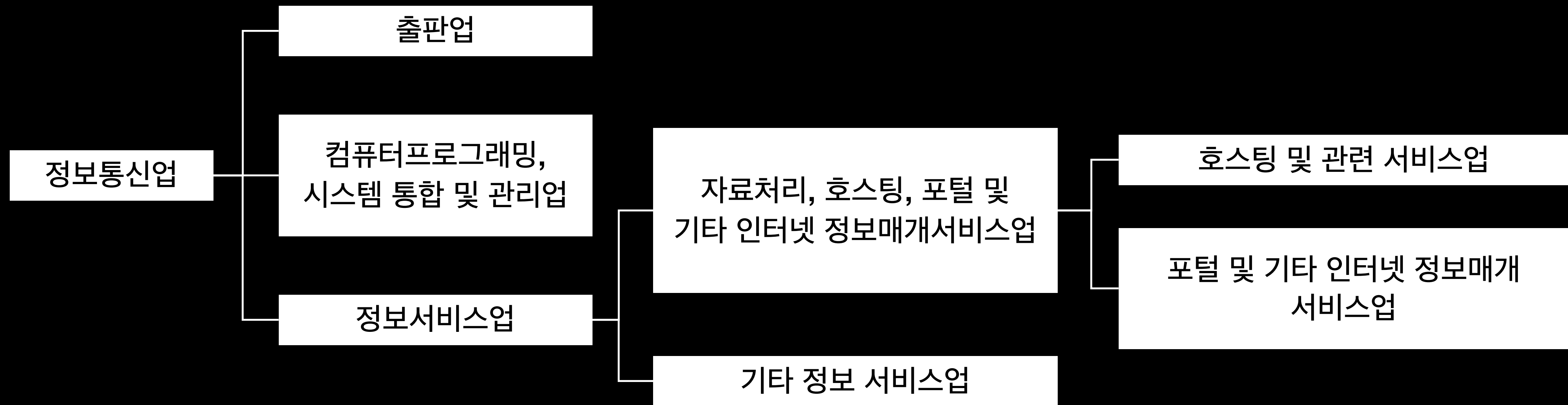


# 클래스



# 분류(classification)

## 한국 표준산업분류(KSIC)



# Python은 class 기반 OOP

## class라는 객체가 하는 일

- Python의 모든 데이터는 객체
- 모든 객체는 분류 체계에 따라 유형(`type`)에 속한다.
- 클래스 : 객체의 유형을 분류하는 객체
- 모든 객체의 최상위 객체(뿌리) : `object`



# 인스턴스 객체 (instance object)

- 클래스를 실체화하는 것 : 인스턴스화(instantiation)

- 클래스 : 사람속(Homo) - 사람종(Homo sapien)

- 인스턴스 객체 : 한날이라는 사람 자체

- 이름(변수) : 한날, 차경묵, ...

```
hannal = Member()
```

- 사람종에 속하는 생물들은 공통된 속성을 갖고 있다.

- 멤버 변수 : 눈, 코, 입 모양과 위치

```
assert hannal.__class__ is Member
```

- 메서드 : 할 줄 아는 것과 하는 건

- 각 사람(인스턴스 객체)마다 다르다.

- 클래스 자체는 인스턴스 객체와 달리 프로세스(process) 당 하나만 존재



# class 선언 문법

## 기본 문법

- 예약어 : class
- 이름 뒤 괄호는 상속받을 클래스 명기. 생략 시 상속받는 클래스 없음.
- 클래스로 인스턴스 객체를 생성하려면 함수처럼 클래스를 호출.
- 함수에 인자를 전달하듯이, 클래스도 호출 시 인자 전달 가능.  
\_\_init\_\_( ) 메서드로 전달 받음.

```
class HomoSapiens(Homo):  
    def __init__(self, height, weight, gender):  
        self.height = height  
        self.weight = weight  
        self.gender = gender  
  
    def speak(self, message):  
        pass  
  
hannal = HomoSapiens(185, 70, 'Male')
```





# class 선언 문법

## 메서드

- 인스턴스 메서드
- 클래스 메서드
- 스태틱 메서드
- 프로퍼티

```
class HomoSapiens(Homo):  
    def __init__(self, height, weight, gender):  
        pass  
  
    def speak(self, message):  
        pass  
  
    @classmethod  
    def build(cls, height, weight, gender):  
        pass  
  
    @staticmethod  
    def puddingcamp(course, number):  
        print("좋아요 ❤️")  
  
    @property  
    def age(self):  
        pass
```



# 객체 유형(분류) 확인

- 인스턴스 객체 : `isinstance()` 함수
- 클래스 : `issubclass()` 함수

```
hanna1 = HomoSapiens(185, 70, 'Male')
```

```
isinstance(hanna1, HomoSapiens)
```

```
isinstance(hanna1, Homo)
```

```
isinstance(hanna1, object)
```

```
isinstance("hanna1", Homo)
```

```
issubclass(HomoSapiens, Homo)
```



# 실습

# 실습 : 20240307-1-2-6

## 계산기 클래스 만들기 : src 디렉터리 안에 calc\_class.py

```
class Calculator:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def add(self):
        return self.a + self.b

    def subtract(self):
        return self.a - self.b

    def multiply(self):
        return self.a * self.b

    def divide(self):
        return self.a / self.b

    def __str__(self):
        return f"a: {self.a}, b: {self.b}"
```



# 인스턴스 메서드와 클래스 메서드의 첫 번째 인자

- 인스턴스 메서드와 클래스 메서드의 첫 번째 인자는 메서드가 속한 객체 자신
  - 다른 프로그래밍 언어에서는 종종 `this` 라는 예약어를 사용
- 암묵적으로 `this`나 `self` 키워드를 사용하는 게 아니라 명시적인 인자로 Python이 자동으로 전달
- 인스턴스 메서드 : 클래스의 인스턴스 객체 자신 (`self`)
- 클래스 메서드 : 클래스라는 객체 자신 (`cls`)
- 다른 이름도 가능하나, 사실상 문법 같은 관례



# 실습

# 실습 : 20240307-1-2-7

```
class Calculator:
    a = 3
    b = 10

    @classmethod
    def add(cls):
        return cls.a + cls.b

    @classmethod
    def subtract(cls):
        return cls.a - cls.b

    @classmethod
    def multiply(cls):
        return cls.a * cls.b

    @classmethod
    def divide(cls):
        return cls.a / cls.b
```

```
class Calculator:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def add(self):
        return self.a + self.b

    def subtract(self):
        return self.a - self.b

    def multiply(self):
        return self.a * self.b

    def divide(self):
        return self.a / self.b

    def __str__(self):
        return f"a: {self.a}, b: {self.b}"
```

파일명 : src 디렉터리 안에 calc\_class2.py



# 과제





# 과제-20240307-1 (필수)

# 실습 코드 전체를 손으로 재작성하세요.

## 매우 중요한 과제!

- 이번 편에 나온 실습 코드 전체를 동일하게 1회 이상 재작성하세요.
  - 실습 코드를 덩어리로 읽고, 그 덩어리를 안 보고 손으로 작성하는 방식이면 더 좋습니다.
  - Python 코드를 작성하는 것이 사실상 처음인 분은 이 과제가 특히 중요해요.
- 암기나 코드를 이해하는 게 아니라 코드 패턴을 손에 익히는 게 더 중요한 과제입니다.
- 코드 패턴이 손에 익어야 학습할 때 드는 의지력이 줄어들어요. 코드 칠 때마다 매번 기억을 뒤져 타이핑하면 금방 지칩니다.
- 무엇보다도 코드 타이핑하는 데 정신이 팔려서 학습 주제를 놓치는 상황을 자주 접하게 됩니다.
- “함수 hello를 만드시고요”라는 말이 나오면 자연스럽게 `def hello():` 를 타이핑할 수 있게 돼야 하며, 눈으로만 볼 땐 당연히 될 거라 생각하지만 막상 그 상황이 되면 막막하고 버벅이게 돼요.
- 다시 강조하지만, 코드를 이해하는 게 아니라 코드 패턴이 손에 익숙해지게 하는 것이 목적입니다!
- 재작성한 파일을 제출. 파일명(`code_on_hand.py`)



# 과제-20240307-2 (필수, 제출)

사칙 연산 함수, `add()`, `sub()`, `mul()`, `div()` 만으로

주어진 문제를 연산하는 함수를 작성하세요.



# BMI 계산기

제출할 파일명 : `calc_index.py`

- BMI == 몸무게(kg) / 신장(m)\*신장(m)
- 몸무게는 g, 신장(키)는 cm 단위.
- 함수명 : `calc_bmi`

```
def calc_bmi(weight, height):  
    return
```

```
calc_bmi(64000, 160) == 25
```



# 가중 평균 자본 비용 (WACC) 계산기

제출할 파일명 : `calc_index.py`

$$WACC = \left(\frac{E}{V}\right) \times Re + \left(\frac{D}{V}\right) \times Rd \times (1 - Tc)$$

- $E$ 는 기업의 시장가치에서의 자기자본 가치
- $V$  : 기업의 전체 시장가치(자기자본 + 부채)
- $Re$  : 자기자본의 비용(보통주 자본비용)
- $D$  : 기업의 시장가치에서의 부채 가치
- $Rd$  : 부채의 비용(이자율)
- $Tc$  : 기업의 법인세율
- 함수명 : `calc_wacc`
- RealWorldPudding 예시
- 자본 : 600,000,000
- 부채 : 400,000,000
- 자기자본 비용 : 8%
- 부채 비용 : 5%
- 법인세율 : 30%
- 계산 결과 : 약 6.2% (0.062)

