

# 파이썬

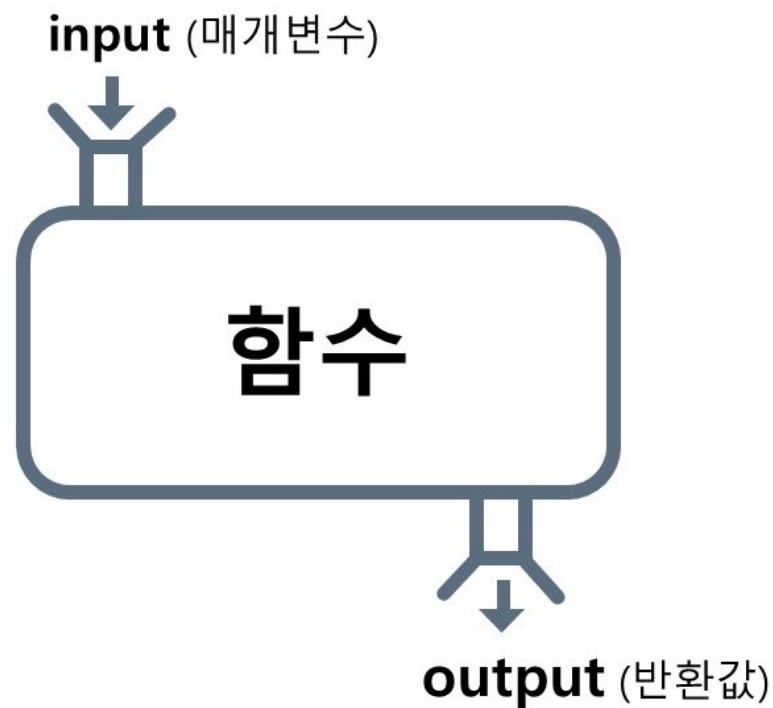
(주)인피닉스 - 강호용 연구원

Infinyx



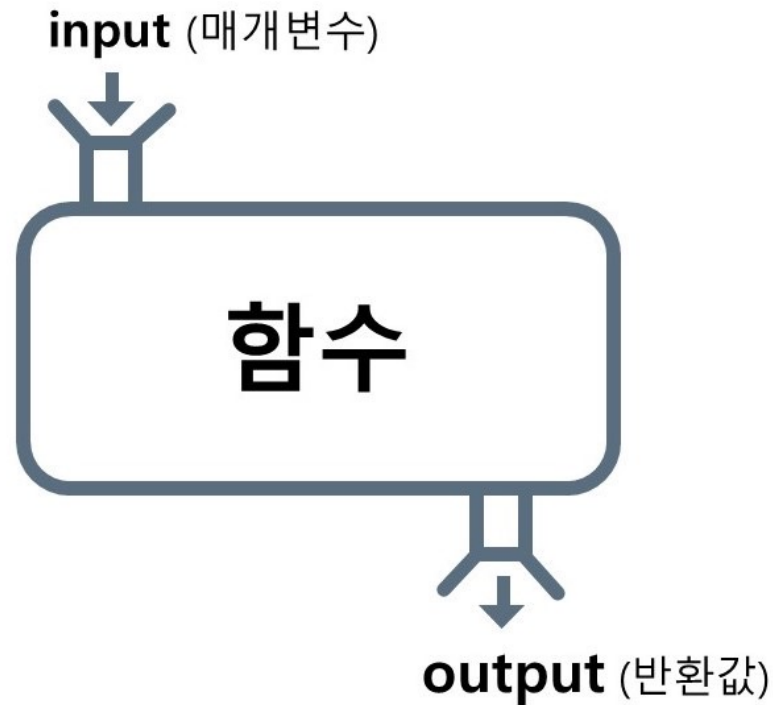
# 함수, 클래스

## # 함수 소개



- 반복되는 코드를 묶어서 하나의 기능을 수행한다.  
→ 반복 사용이 가능하고 코드의 가독성이 높아진다.
- 반드시 인풋과 아웃풋이 있다. input은 parameter라 하고, output은 return 값이다.
- return 값이 없으면 None을 반환한다.

## # 함수 소개



## 그러면 왜 사용하는가?

코드의 재사용성을 높이고 중복을 최소화 한다.  
절차적 분해 - 절차에 따라 혹은 기능에 따라 분해하여 테스트 하기 편함

함수는 정의와 호출로 구성 되어있음

- def 문으로 정의하고, 함수명()으로 호출한다.
- def 문을 만나면 해당 함수명을 네임스페이스에 저장  
-> 'id(함수명)'으로 숫자주소를 볼 수 있다.  
ex) 변수도 마찬가지이다.

## # 함수 기본적인 구조 형태

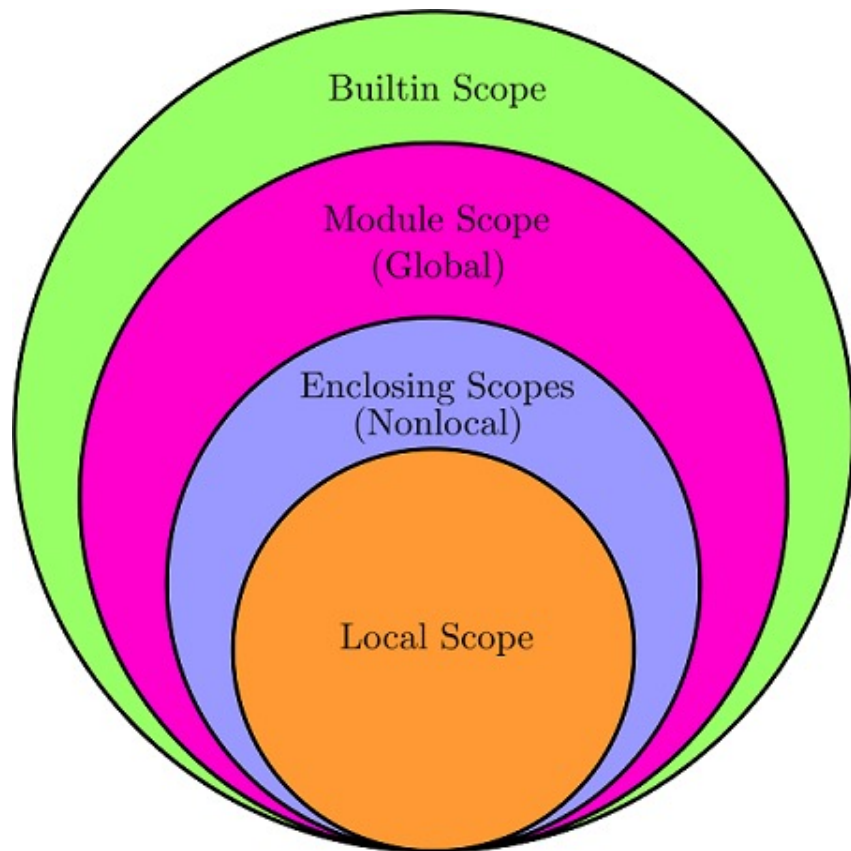
함수 생성  
키워드      함수의 이름

↓                      ↓  
`def` say\_hello() :  
    prinit("Hello, world")

·  
·  
·

} 함수가 수행할 코드

## # Scope



Scope는 '범위'라는 뜻이다.  
프로그래밍언어에서 Scope는 어떠한 객체(변수, 함수 등)가 필요한  
범위를 이야기한다.

범위를 벗어나면 해당 객체는 사용될 수 없다.  
파이썬에서 Scope는 항상 객체가 선언된 지점에서 위로는 상위 객  
체 까지, 아래로는 모든 하위 객체들과 그 안에까지 범위를 갖는다.

## # Scope

- Local Scope : 가지고 있는 변수나 함수 혹은 객체는 이름 그대로 특정 범위에서만 유효하다.  
주로 함수 안에서 선언된 변수나 함수가 local scope를 가지고 있다. 이런 변수들은 해당 함수 안에서만 유효하다

변수 a의 local scope.  
즉 변수가 a가 유효한  
범위

{

```
def func1():  
    c = 1  
    print(c)
```

함수안에서 선언 되었으므로  
해당 함수안에서만 유효한 변수임  
즉 local scope를 가지고 있는 변수

print(c) →

Traceback (most recent call last): in <module>  
---> print(c)  
NameError: name 'c' is not defined

## # Scope

- Enclosing Scope : 중첩함수가 있을때 적용되는 Scope
  - 부모 함수에서 선언된 변수는 중첩함수 안에서도 유용한 범위를 가짐

변수 a가 유용한  
범위

1 usage

```
def fun1():
```

```
    c = 1
```

```
    print(c)
```

```
    def inner():
```

```
        b = 7
```

```
        print(c * b)
```

```
    inner()
```

```
    print(b)
```

```
fun1()
```

여기서는 b변수는 inner 함수에서만 적용되는  
Local scope

여기서는 a 변수는 부모함수는 fun1함수에 선언됐지만  
Enclosing scope 임 즉 inner 함수에서도 사용가능

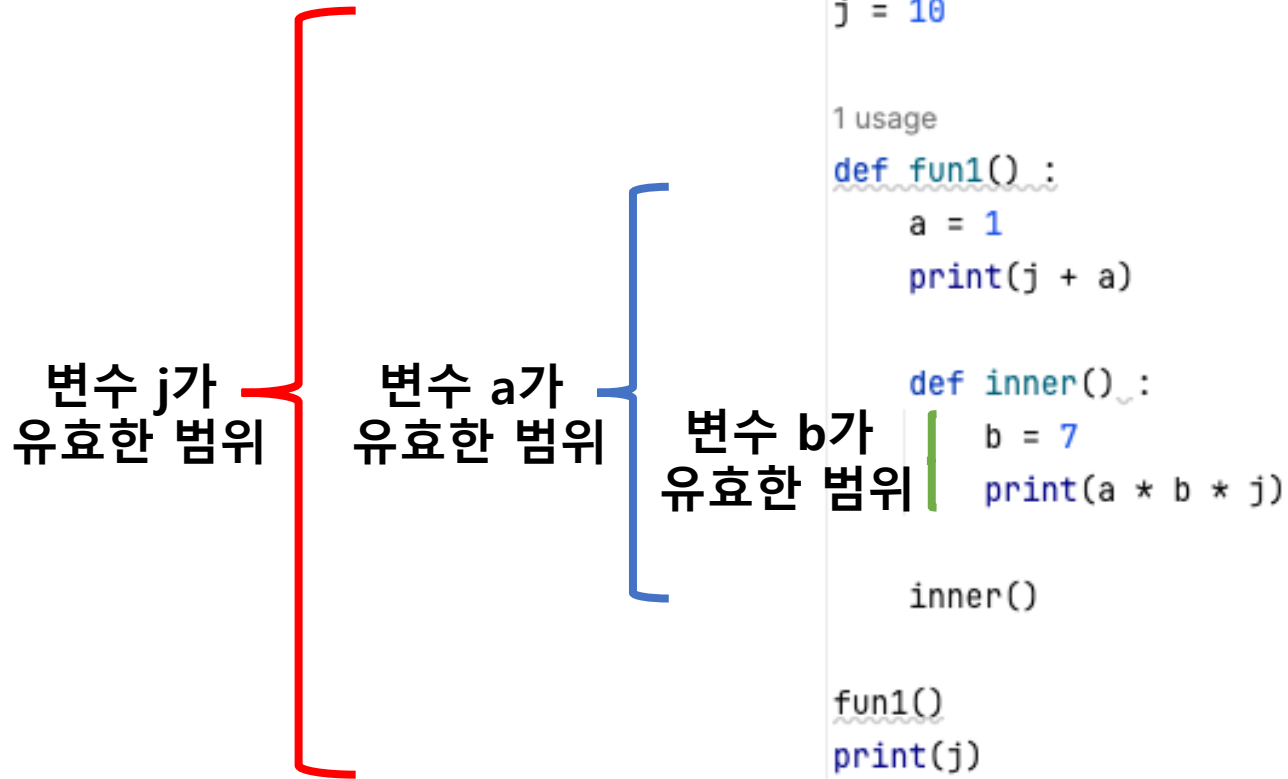
B는 innert 함수에서만 적용되는 local scope를 가지고 있음으로  
inner 함수를 벗어나서는 사용불가 그래서 에러가 발생

NameError: name 'b' is not defined



## # Scope

- Global Scope



Global Scope : 함수 안에서 선언된 것이 아닌, 함수 밖에서 선언된 변수나 함수를 말함

변수나 함수는 선언된 지점과 동일한 level의 지역, 그리고 더 안쪽의 지역들까지 범위가 유효하다.

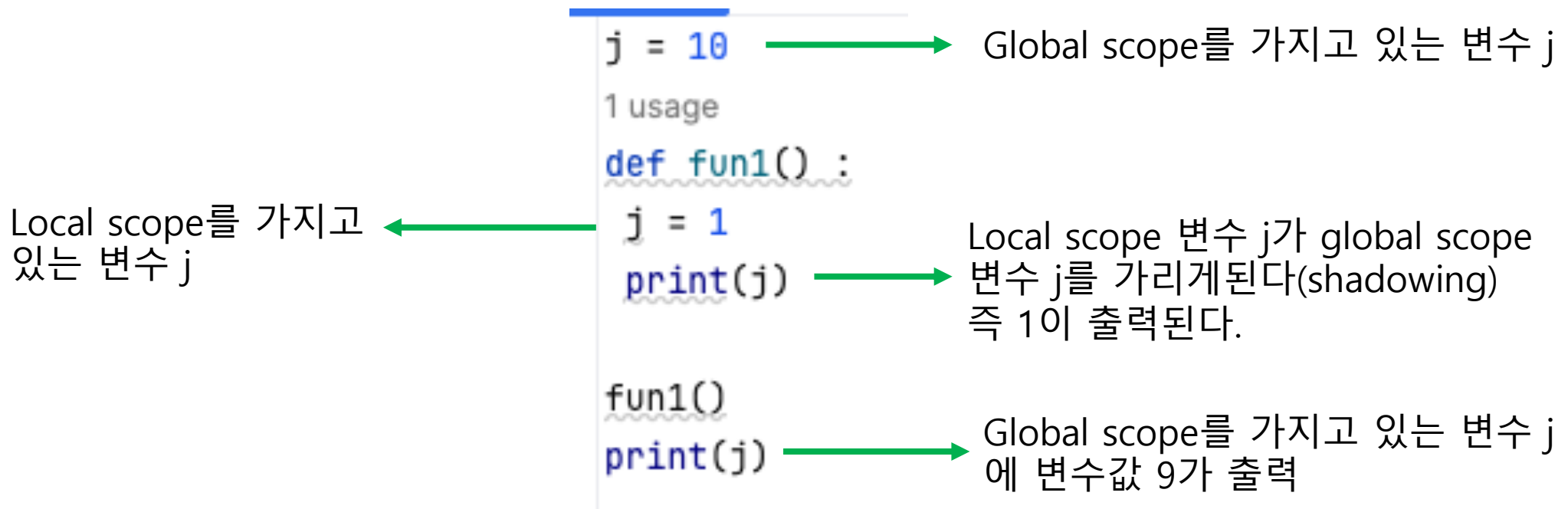
그리고 global scope를 가지고 있는 변수와 함수들은 선언된 지점이 해당 파일에서 가장 바깥쪽에서 선언되므로 해당 파일에서 선언된 지점 아래로는 다 유효한 범위를 가짐

## # Scope

- Built-in Scope : **가장 광범위한 scope이다.**  
파이썬안에 내장되어 있는, 파이썬이 제공하는 함수 또는 속성들이 built-in scope를 가지고 있다.  
그리고 **built-in scope는 따로 선언이 없이도 모든 파이썬 파일에서 유용한 범위를 가지고 있다.**  
예를 들어, list등과 같은 자료구조의 element 총 개수를 리턴하는 len() 함수가 바로 built-in scope의 예이다.

## # Shadowing

- 파이썬은 변수나 함수의 정의를 찾을때 다음순서의 scope들 안에서 찾는다.
  - Local -> Enclosing -> Global -> Built-in 순서로 서칭
- 즉 가장 좁은 유효 범위 부터 시작해서 가장 넓은 범위로 나아가며, 사용되는 변수나 함수의 정의를 찾는다. 그러므로 동일한 이름의 변수들이 서로 다른 scope에서 선언되면, 더 좁은 범위에 있는 변수(혹은 함수)가 더 넓은 범위에 있는 변수를 가리는 Shadowing 효과가 나타난다.



## # Parameters 소개

함수는 input parameter를 받아서 return 값을 output으로 리턴한다.

### Positional arguments (위치 인수)

- 입력된 인자의 순서는 parameter의 순서대로 전달된다.
- 입력 인자의 순서가 바뀌어도 파악이 어려운 단점이 있다.

### Keyword Arguments

- parameter 순서에 맞추지 않고 값을 전달할 수 있다.
- **함수를 호출할때** parameter 이름과 함께 값을 전달하는 방식이다.

```
func(parameter2 = value2, parameter1 = value1)
```

어떤 parameter에 어떤 argument가 입력되는지 명확히 알 수 있다.

## # Parameters 소개

### ➤ Parameter Default Value

```
def func(a=1,b,c):  
    print(a)  
    print(b)  
    print(c)  
  
func(2,3)  
  
# SyntaxError: non-default argument follows default argument
```

→ 첫번째 인수 2가 a 파라미터로 들어가서 초기값을 바꿔야 하는지, b에 들어가야 하는지 알 수 없기 때문이다.

- default value를 정해두면 **입력값이 없어도** 함수 호출이 가능하다.
- 함수를 정의할때 parameter에 값을 지정한다.  
def func(param1, param2 = default\_value):
- parameter2 가 **다른 값으로 입력되어도** 작동된다.
- **default 값이 있으면 non-default argument보다 뒤에 와야한다.**
- 그렇지 않으면, SyntaxError: non-default argument follows default argument 가 발생한다.

## # Parameters 소개

### ➤ Variable length arguments

- 위치 인수의 개수가 정해지지 않았을 때 사용한다.
- `*args` 파라미터로 표현한다.
- 튜플로 변환되어 함수로 전달된다.
- 인수가 없을 수도 있다.

```
def func_param_with_var_args(name, *args, age):  
    print("name=",end=""), print(name)  
    print("args=",end=""), print(args)  
    print("age=",end=""), print(age)  
  
func_param_with_var_args("정우성", "01012341234", "seoul", 20)  
  
#`TypeError: func_param_with_var_args() missing1 required keyword-only argument: 'age'`
```

→ `*args` 파라미터는 여러개의 인수를 받을 수 있기 때문에 `age` 파라미터의 값을 무엇으로 할지 알 수 없다.

→ 해결 : `age=20` 처럼 키워드를 이용해 `age` 파라미터의 값을 명시해서 입력해야한다. 아니면 **`*args` 파라미터를 가장 끝으로 보내고 20 을 두번째 인수로 입력한다.**

## # Parameters 소개

### ➤ Variable length keyword arguments

- 키워드 인수의 개수가 정해지지 않았을 때 사용한다. (딕셔너리)
- `**kwargs` 파라미터로 표현한다.
- 딕셔너리로 변환되어 함수로 전달된다.
- 인수가 없을 수도 있다.

# 에러발생

```
def mixed_params(name="아이유", *args, age, **kwargs, address):  
    print("name=",end=""), print(name)  
    print("args=",end=""), print(args)  
    print("age=",end=""), print(age)  
    print("kwargs=",end=""), print(kwargs)  
    print("address=",end=""), print(address)
```

```
mixed_params(20, "정우성", "01012341234", "male",  
             mobile="01012341234", address="seoul")
```

예시는 `SyntaxError: invalid syntax` 에러 발생

age 와 address 를 앞으로 옮기면? address가 여러 값을 받는다.  
`TypeError: mixed_params() got multiple values for argument 'address'`

address 파라미터는 키워드로 인수를 입력하는 keyword-only argument이다.

## # Parameters 소개

### ➤ 함수 파라미터의 순서

```
func(param1, param2, param3=default_value, *args, key=value, **kwargs)
```

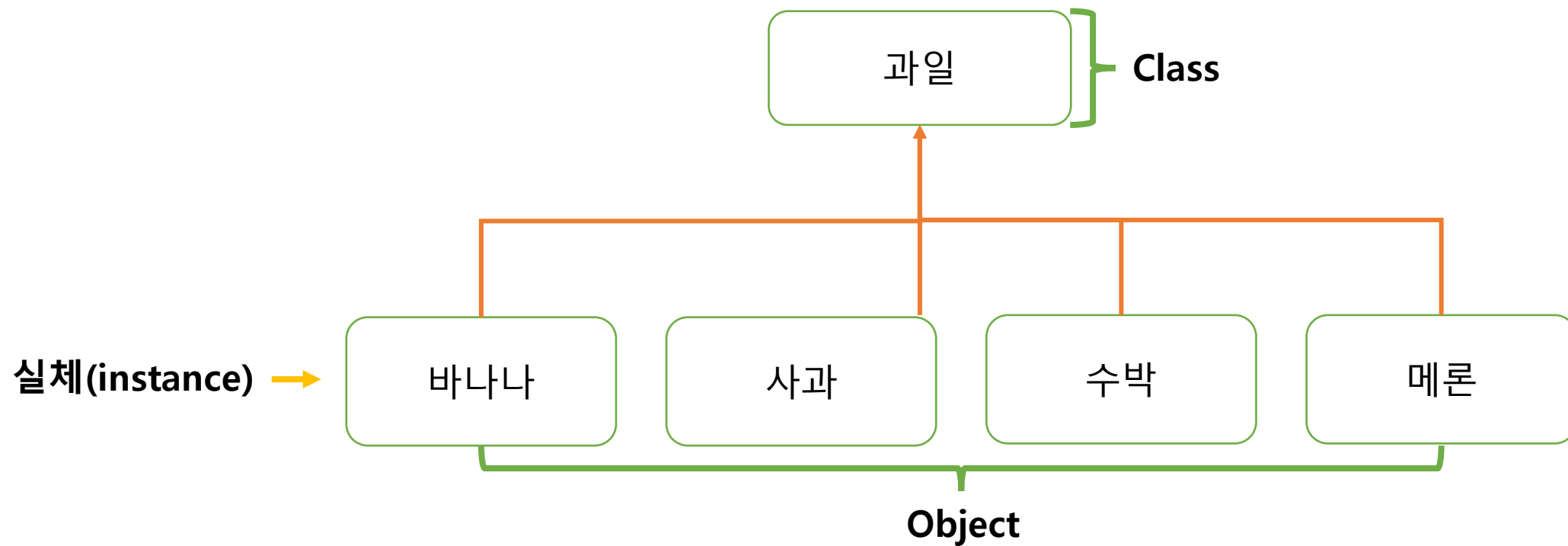
위치 인수 - 디폴트 인수 - 가변 위치 인수 - 키워드 인수 - 가변 키워드 인수

1. Regular Positional args. : 인수 순서를 지키는 기본 형태
2. Default args. : 파라미터에 초기값이 설정되어 있는 경우
3. Variable length positional args. : 여러개의 위치 인수를 입력하는 경우 \*args
4. keyword-only args. : 인수에 키워드를 지정해서 입력하는 경우
5. Variable length keyword args. : 파라미터에 없는 키워드로 값을 입력하는 경우 \*\*kwargs



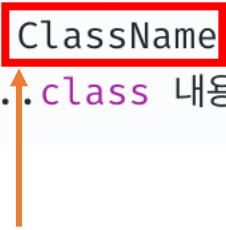
## # 클래스

- Class 사전적의미 중 '부류'라는 의미에 가깝다. 즉 공통점이 많은 사람들을 가리켜 '부류'라고 하는 것이다.



## # 클래스 구문


```
## Class Syntax  
class ClassName:  
    ...class 내용 코드들
```



Class 이름은 각 단어의 앞 글자를 대문자로 사용 ex) car -> Car

Class가 정의되면 실체화(instantiate) 할 수 있다. 클래스를 실체화 하는 방법은 함수를 호출하듯 클래스를 호출 하면 됨

```
temp_class = ClassName()
```



ClassName를 실체(instance)화 한것이 'temp\_class'라는 객체(Object)이다.

## # 클래스 attribute(속성) 정의하기

Class 에 정의되는 공통 요소들을 전문용어로 class의 attribute(성질 혹은 속성) 이라고 한다.

과일 속성을 예를 들면


- 과일 이름
- 과일 색상
- 과일 크기

위와 같은 속성(attribute)을 갖기 위해서는 `__init__`이라는 함수를 통해 정의가 필요 (Class 내 함수는 Method라 한다.)

## # 클래스 attribute(속성) 정의하기

```
class Fruit:

    def __init__(self, name, color, size):
        self.name = name
        self.color = color
        self.size = size
```

 \_\_init\_\_ 이라는 메소드는 class가 실체화 될때 사용되는 함수

 \_\_init\_\_ 메소드 위 속성3개의 parameter와 self라는 parameter도 갖는다

- 이 self parameter는 어떠한 실체(instance)를 가르키는 것으로 위 예시에서 말하자면 과일 가 된다

# 클래스 attribute(속성) 정의하기

**fruit\_temp = Fruit( ' 바나나 ', ' 노란색 ', '10' )**

self

name color size

**\_init\_ 메소드**

## # 클래스 Method

- Class내에는 `__init__` 말고도 원하는 메소드들을 원하는 대로 추가할 수 있다. attribute는 해당 객체(instance)의 이름 등의 정해진 성질이고,

1 usage

```
class Fruit:
```

```
    def __init__(self, name, color, size):
```

```
        self.name = name
```

```
        self.color = color
```

```
        self.size = size
```

1 usage

```
def upload_dataset(self):
```

```
    return "업로드 완료"
```

Method는 upload\_dataset 등 객체가 행할 수 있는 어떤 action이다.

```
fruit_temp = Fruit(name="바나나", color="노랑", size="10")
```

```
fruit_temp.upload_dataset()
```

객체에서 메소드를 사용할 때는 dot(.)을 사용하여 객체를 호출하면 된다.



# 예외처리(Exception)

## # Exceptions

**Exceptions**이란 '예외'라는 뜻이다.

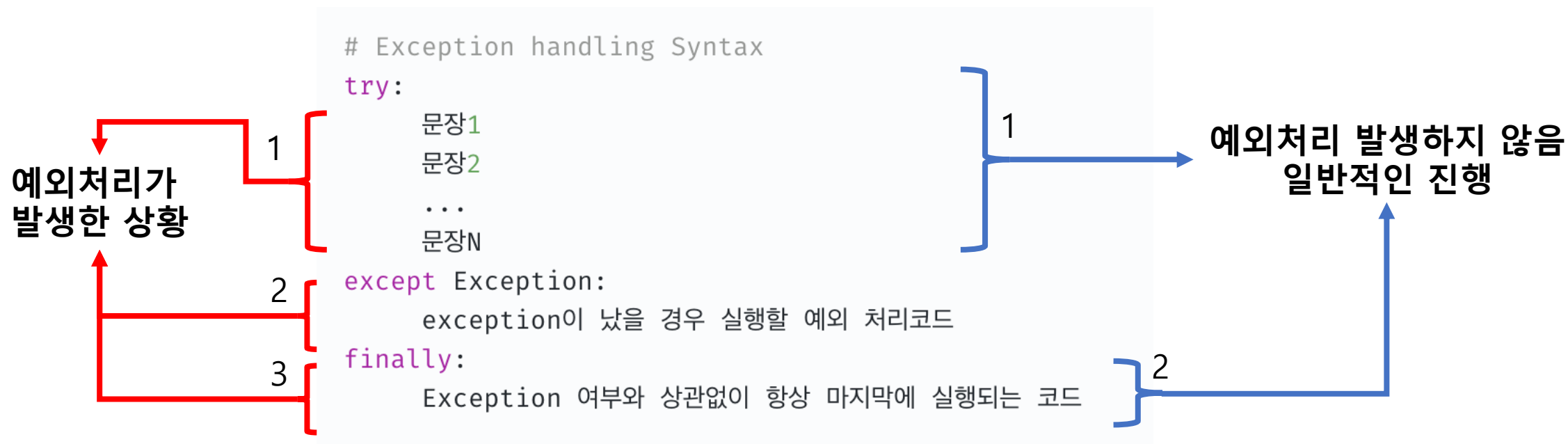
그렇다면 파이썬에서 예외상황은 언제를 이야기 하는 걸까?

**당연히 의도하지 않은 Error가 났을 경우 상황**이 일어나는 경우를 일반적으로 Exceptions이 발생

파이썬에서 보통 Exception이 발생하면, 발생한 코드 위치에서 다음 코드들이 실행되지 않고 곧바로 프로그램이 종료를 하게 된다.



# Exception handling은 try except 구문을 사용해서 실행한다

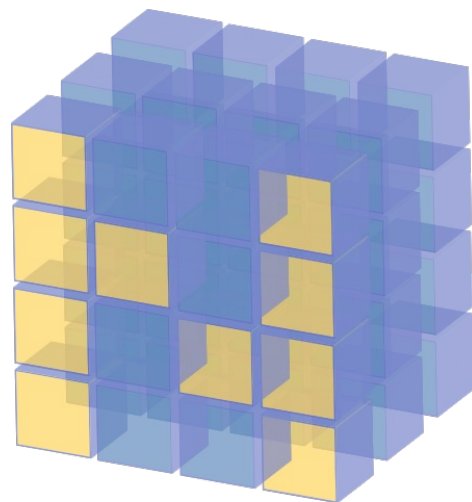


- exception 은 여러 exception을 catch 할 수 있다.



# NumPy

---



# NumPy

**NumPy 무엇인가?**

## # Numpy 소개

- 데이터 과학과 머신 러닝 분야에서 핵심적인 역할을 수행하는 파이썬 라이브러리입니다.
- 고성능의 다차원 배열과 행렬 연산을 제공하여 데이터 처리와 분석 작업을 효율적으로 수행할 수 있도록 도와줍니다.

## # Numpy의 주요 기능

1. **다차원 배열 생성:** NumPy는 다차원 배열을 생성하고 조작하는 다양한 함수를 제공합니다. 배열은 동일한 데이터 타입의 원소들로 구성되며, 이를 통해 벡터화 연산을 수행할 수 있습니다.
2. **표준 수학 함수:** NumPy는 다양한 수학 함수를 제공하여 배열의 원소별 연산을 지원합니다.
3. **인덱싱 및 슬라이싱:** NumPy 배열은 다차원이므로 원하는 부분을 색인화하거나 슬라이스하여 추출하는 작업을 수행할 수 있습니다.
4. **선형 대수 연산:** NumPy는 선형 대수 연산을 위한 기능도 제공합니다. 행렬 곱셈, 역행렬, 특잇값 분해 등의 연산이 가능합니다.
5. **통계 함수:** 데이터 분석을 위한 다양한 통계 함수를 제공하여 평균, 분산, 상관관계 등을 계산할 수 있습니다.

## # Numpy 다차원 배열과 차원 축

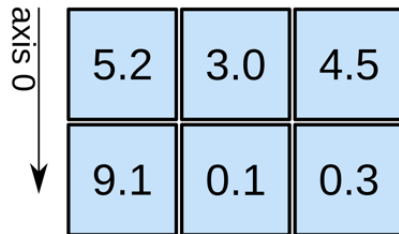
1D array



axis 0 →

shape: (4,)

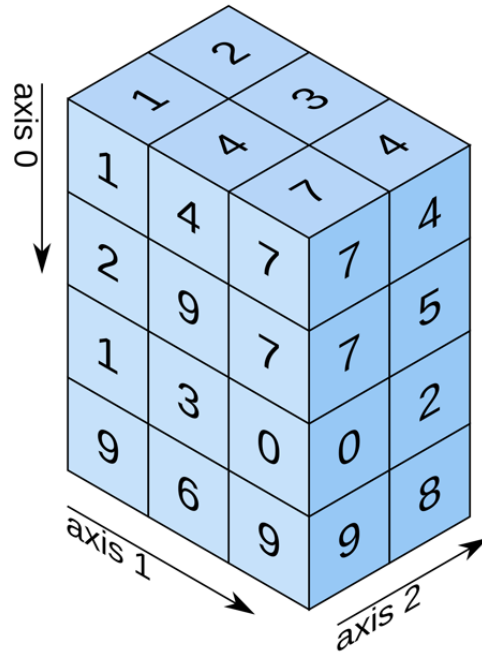
2D array



axis 1 →

shape: (2, 3)

3D array



shape: (4, 3, 2)

- 차원의 수에 따라 n-D array 로 분류
- 각 차원의 크기는 shape (n, m, ...)로 표시
- 각 요소는 모두 동일한 데이터 타입
- 2-D array 이상에서의 축 방향에 주의

## # NumPy 배열 생성

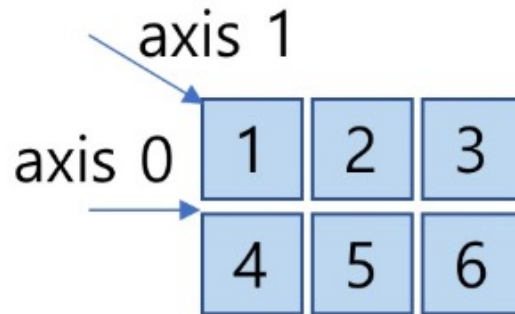
- NumPy에서 가장 기본적인 데이터 구조는 배열입니다.



1D array



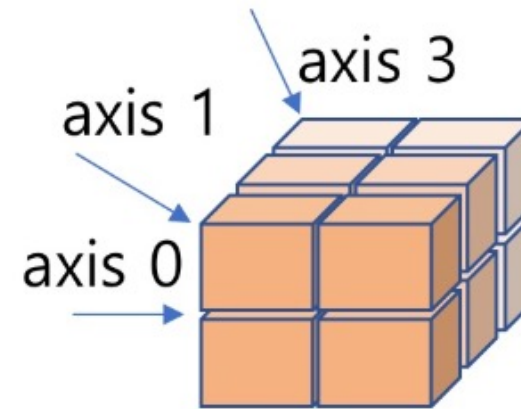
```
a = np.array([1, 2, 3])
```



2D array



```
b = np.array([[1, 2, 3], [4, 5, 6]])
```



3D array




```
c = np.array([[[1, 2], [3, 4]],  
              [[5, 6], [7, 8]]])
```

## # Numpy 배열의 속성

```
import numpy as np
```

```
b = np.random.randint(10, size=(3,4)) # 2차원 배열
```



```
[[5 5 2 8]
 [2 6 0 4]
 [9 1 9 2]]
```

```
print("ndim", b.ndim)
```

랭크 차원의 수 : 2

```
print("shape", b.shape)
```

(형성, 각 차원의 크기) : (3, 4)

```
print("size", b.size)
```

```
print("dtype", b.dtype)
```

전체 배열의 크기 : 12



배열 데이터 타입 (int32)



## # Numpy 배열 연산

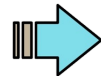
```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

# 원소별 덧셈
c = a + b # [5, 7, 9]

# 원소별 곱셈
d = a * b # [4, 10, 18]

# 스칼라와의 연산
e = a + 1 # [2, 3, 4]
```



- NumPy 배열은 다른 배열 또는 스칼라와의 연산을 지원합니다.
- NumPy 배열의 연산은 배열의 원소별(element-wise)로 이루어집니다.

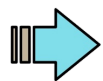
# Numpy 배열 연산에는 다양한 함수와 메소드 존재 가장 기본적인 연산은 sum, mean, min, max

```
import numpy as np
```

```
a = np.array([1, 2, 3])
```

# 합계

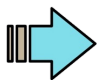
```
b = np.sum(a) # 6
```



$$1 + 2 + 3 = 6$$

# 평균

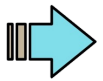
```
c = np.mean(a) # 2.0
```



$$6 / 3 = 2$$

# 최소값

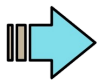
```
d = np.min(a) # 1
```



1

# 최대값

```
e = np.max(a) # 3
```



3

## # NumPy 배열 인덱싱 과 슬라이싱 소개

- Python 리스트의 인덱싱과 슬라이싱과 매우 유사합니다. NumPy 배열의 인덱싱과 슬라이싱을 사용하여 배열의 일부를 선택할 수 있습니다.

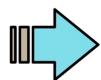
```
import numpy as np
```

```
a = np.array([1, 2, 3, 4, 5])
```

```
# 인덱싱
```

```
b = a[0] # 1
```

```
c = a[2] # 3
```



1차원 배열에 꺾쇠괄호로 인덱스를 지정

```
# 슬라이싱
```

```
d = a[1:4] # [2, 3, 4]
```

```
e = a[:3] # [1, 2, 3]
```

```
f = a[3:] # [4, 5]
```

## # NumPy 배열 인덱싱 과 슬라이싱 소개

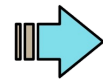
- 다차원 NumPy 배열에서는 각 차원의 인덱스를 콤마로 구분하여 인덱싱

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])

# 인덱싱
b = a[0, 0] # 1
c = a[1, 2] # 6

# 슬라이싱
d = a[0, 1:3] # [2, 3]
e = a[:, 1]   # [2, 5]
f = a[:, :2]  # [[1, 2], [4, 5]]
```

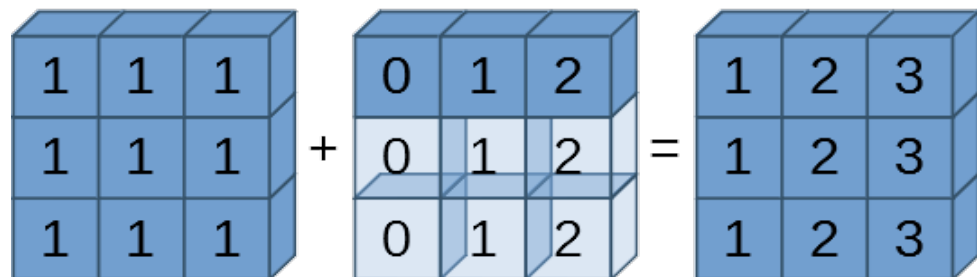
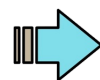
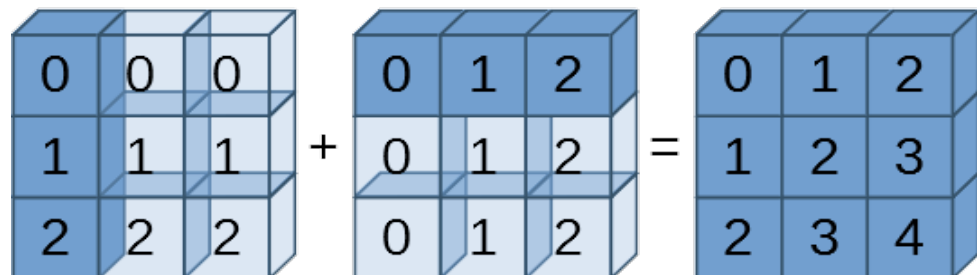


다차원 배열에서는 콤마로 구분된 인덱스를 사용.



주의 : 파이썬 리스트와 달리 NumPy 배열을 고정 타입을 가지므로 다른 타입의 데이터 할당시 형변환에 주의

## # NumPy : 브로드 캐스팅

`np.arange(3) + 5``np.ones((3, 3)) + np.arange(3)``np.ones((3, 1)) + np.arange(3)`


- 규칙 1: 차원 수가 다르면 작은 쪽 배열의 왼쪽 차원을 1로 채워서 맞춘다
- 규칙 2: 각 차원의 크기(형상)가 다르면 차원의 크기가 1인 차원의 크기를 일치하도록 늘린다
- 규칙 3: 임의의 차원에서 크기가 일치하지 않고 1도 아니라면 오류를 발생한다

## # NumPy 배열 병합과 분리

### ➤ NumPy 배열을 병합하는 방법

```
import numpy as np
```

```
a = np.array([1, 2, 3])  
b = np.array([4, 5, 6])
```




1차원 배열 기준

# 배열 병합

```
c = np.concatenate((a, b)) # [1, 2, 3, 4, 5, 6]
```

```
a = np.array([[1, 2], [3, 4]])  
b = np.array([[5, 6]])
```




다차원 배열 기준 : concatenate() 함수를 사용합니다.

# 배열 병합

```
c = np.concatenate( arrays: (a, b), axis=0) # [[1, 2], [3, 4], [5, 6]]
```

axis : 인자를 사용하여 병합할 방법을 지정할 수 있습니다.  
두개 이상의 배열 병합 할 경우는 튜플 형태로 전달 필요



## # NumPy 배열 병합과 분리

- 1차원 NumPy 배열을 분리 하는 방법

```
a = np.array([1, 2, 3, 4, 5, 6])  
  
# 배열 분리  
b, c = np.split(a, indices_or_sections: [3]) # [1, 2, 3], [4, 5, 6]
```

- 다차원 NumPy 배열을 분리하는 방법

```
a = np.array([[1, 2, 3], [4, 5, 6]])  
  
# 배열 분리  
b, c = np.split(a, indices_or_sections: [1], axis=0) # [[1, 2, 3]], [[4, 5, 6]]
```

Axis=0 첫 번째 차원을 따라 배열을 분리  
따라서 a 배열의 첫번째 행을 기준으로  
배열을 분리

↓  
split() 함수는 분리된 배열을 튜플 형태로 반환합니다.

## # Numpy 주요 함수 : np.zeros(), np.ones()

- 0 으로 채운 길이 10의 정수 배열 생성

```
import numpy as np
```

```
a = np.zeros(shape: 10, dtype=int) ➡ [0 0 0 0 0 0 0 0 0 0]
```

- 1 으로 채운 3x5 정수 배열 생성

```
import numpy as np
```

```
b = np.ones(shape: (3, 5), dtype=int) ➡   
[[1 1 1 1 1]  
 [1 1 1 1 1]  
 [1 1 1 1 1]]
```



## # NumPy 주요 함수 : arange()

- 범위 내의 일정 간격을 가진 배열을 생성합니다. 함수의 인수로는 생성할 배열의 범위와 간격을 지정합니다.

```
import numpy as np
```

```
arr = np.arange(1, 10, 2) ➡ [1 3 5 7 9]
```

```
print(arr)
```



출력 결과에서 보듯이 np.arange() 함수는 범위 내의 일정 간격을 가진 배열을 생성합니다.

## # NumPy 주요 함수 : linspace()

- 범위 내에서 균등 간격으로 원하는 개수의 배열을 생성합니다..

```
import numpy as np
```

```
arr = np.linspace(start: 0, stop: 1, num: 5) ➡ [0.  0.25 0.5  0.75 1. ]  
print(arr)
```



출력 결과에서 보듯이 np.linspace() 함수는 범위 내에서 균등 간격으로 원하는 개수의 배열을 생성합니다.


### # NumPy 주요 함수 : np.random.random()

- 0과 1사이의 균등 분포에서 난수를 생성하여 배열을 만듭니다.
- 함수의 인수로는 생성할 배열의 크기를 지정할 수 있습니다.

```
import numpy as np
```

```
arr = np.random.random((3, 3))
```

```
print(arr)
```



```
[[0.43611932 0.53309413 0.85062663]  
 [0.97878835 0.10396215 0.97385621]  
 [0.39815549 0.96824433 0.96906603]]
```



지정된 크기의 배열을 생성하며, 배열의 각 원소는 0과 1 사이의 난수로 채워집니다.

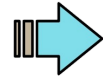
## # NumPy 주요 함수 : np.random.randn()

- 평균이 0이고 표준편차가 1인 정규 분포를 따르는 난수를 생성하여 배열을 만듭니다.

```
import numpy as np
```

```
arr = np.random.randn(2, 4)
```

```
print(arr)
```



```
[[ 0.81178951 -0.35850229 -0.25348391  1.01845018]  
 [-1.17900514  1.68795195  2.03777159  0.57035806]]
```



출력 결과에서 보듯이 np.random.randn() 함수는 지정된 크기의 배열을 생성하며, 배열의 각 원소는 평균이 0이고 표준편차가 1인 정규 분포를 따르는 난수로 채워집니다.

## # NumPy 주요 함수 : 원소 정렬

```
import numpy as np

array = np.array([5, 10, 20, 15, 30, 2])
array.sort()
print(array) # 오름차순 정렬 ➡ [ 2  5 10 15 20 30]

# 내림 차순 정렬
print(array[::-1]) ➡ [30 20 15 10  5  2]
```

## # NumPy 주요 함수 : 원소 정렬 - 각 열 행 을 기준으로 정렬 하는 방법

- 열을 기준으로 정렬

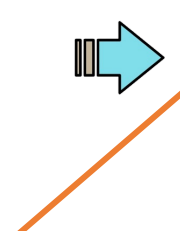
```
import numpy as np

array = np.array([[5, 10, 20, 15, 30, 2], [7, 3, 5, 62, 4, 8]])
org_array = array.copy()

# 각 열을 기준으로 정렬
print("각 열을 기준으로 정렬 전 \n", array)

array.sort(axis=0)
print("각 열을 기준으로 정렬 후 \n", array)
```

각 열을 기준으로 정렬 전



5	10	20	15	30	2
7	3	5	62	4	8

열 끼리 정렬 진행

각 열을 기준으로 정렬 후

5	3	5	15	4	2
7	10	20	62	30	8

## # NumPy 주요 함수 : 원소 정렬 - 각 열 행 을 기준으로 정렬 하는 방법

- 행을 기준으로 정렬

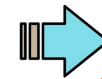
```
import numpy as np

array = np.array([[5, 10, 20, 15, 30, 2], [7, 3, 5, 62, 4, 8]])
org_array = array.copy()

# 각 열을 기준으로 정렬
print("각 행을 기준으로 정렬 전 \n", array)

array.sort(axis=1)
print("각 행을 기준으로 정렬 후 \n", array)
```

행 끼리 정렬 진행



각 행을 기준으로 정렬 전

```
[[ 5 10 20 15 30 2]
 [ 7  3  5 62  4  8]]
```

각 행을 기준으로 정렬 후

```
[[ 2  5 10 15 20 30]
 [ 3  4  5  7  8 62]]
```

## # NumPy 주요 함수 : 배열 객체 복사

```
import numpy as np

array1 = np.arange(0, 10)
array2 = array1

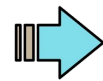
array2[0] = 99
print(array1)
print(array2)
```



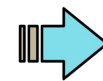
```
[99  1  2  3  4  5  6  7  8  9]
[99  1  2  3  4  5  6  7  8  9]
```



문제 이렇게 복사를 하면 메모리 주소를 공유하므로 나는 array2 0번지 99만 변경되길 원하는데 array1번도 변경된 것을 볼 수 있습니다.



그러면 어떻게 하면 되는가?



copy() 함수를 사용해야함



## # NumPy 주요 함수 : 배열 객체 복사 → copy() 사용

```
import numpy as np

array1 = np.arange(0, 10)
array2 = array1.copy()

array2[0] = 99
print(array1)
print(array2)
```



[0 1 2 3 4 5 6 7 8 9]



[99 1 2 3 4 5 6 7 8 9]

copy() 사용하면 새로운 메모리 공간이 할당 되어서  
더이상 앞에 예제와 같은 결과는 발생하지 않습니다.

## # NumPy 기타 함수 소개

참고 자료 : 수학함수

- `sum()`, `mean()` : 배열 전체 합, 평균
- `cumsum()`, `cumprod()` : 배열 누적 합, 누적 곱
- `std()`, `var()` : 표준편차, 분산
- `min()`, `max()` : 최소값, 최대값
- `argmin()`, `argmax()`: 최소 원소의 색인 값, 최대 원소의 색인 값

## # NumPy 기타 함수 소개

참고 자료 : 난수 함수 - I

- `seed()` : 난수 발생기의 seed를 지정한다.
- `permutation()` : 임의의 순열을 반환한다.
- `shuffle()` : 리스트나 배열의 순서를 뒤섞는다.
- `rand()` : 균등분포에서 표본을 추출한다.
- `randint()` : 주어진 최소/최대 범위 안에서 임의의 난수를 추출한다.
- `randn()` : 표준편차가 1이고 평균값이 0인 정규분포에서 표본을 추출한다.

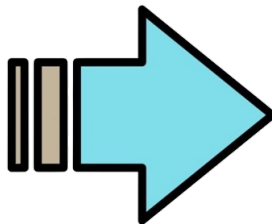
## # NumPy 기타 함수 소개

참고 자료 : 난수 함수 - II

- `binomial()` : 이항분포에서 표본을 추출한다.
- `normal()` : 정규분포(가우시안)에서 표본을 추출한다.
- `beta()` : 베타분포에서 표본을 추출한다.
- `chisquare()` : 카이제곱분포에서 표본을 추출한다.
- `gamma()` : 감마분포에서 표본을 추출한다.
- `uniform()` : 균등(0,1)에서 표본을 추출한다.

## # NumPy 저장과 불러오기

	[0]	[1]	[2]	[3]	[4]
[0]					
[1]					
[2]					
[3]					
[4]					



aaa.npy

## # NumPy 저장하기

### - 단일 객체 저장 및 불러오기

```
import numpy as np
```

```
# 단일 객체 생성
```

```
array1 = np.arange(0, 10) ← 단일 NumPy 객체 생성
```

```
print(array1)
```

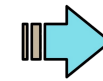
```
# numpy 형태로 저장
```

```
np.save(file="save.npy", array1) ← NumPy 저장 (파일명, NumPy 객체)
```

```
# numpy data load
```

```
result = np.load("save.npy") ← NumPy 파일 불러오기 (NumPy 파일명)
```

```
print(result)
```



저장 값 : [0 1 2 3 4 5 6 7 8 9]

저장 값 호출 결과 : [0 1 2 3 4 5 6 7 8 9]



동일한 값이 나오는 걸  
확인가능

## # 복수 객체 저장 및 불러오기

```
import numpy as np
```

```
# 복수 객체 생성
```

```
array1 = np.arange(0, 10)
```

```
array2 = np.arange(10, 20)
```

```
print(array1, array2)
```

복수 객체 생성

```
# numpy save
```

```
np.savez(file: "save.npz", array1=array1, array2=array2)
```

기존 save 가 아닌 → savez() 함수를 사용  
저장 형태 타입이 다름 .npy → .npz

```
data = np.load("save.npz")
```

복수 객체가 저장된 .npz 읽기

```
# 객체 호출
```

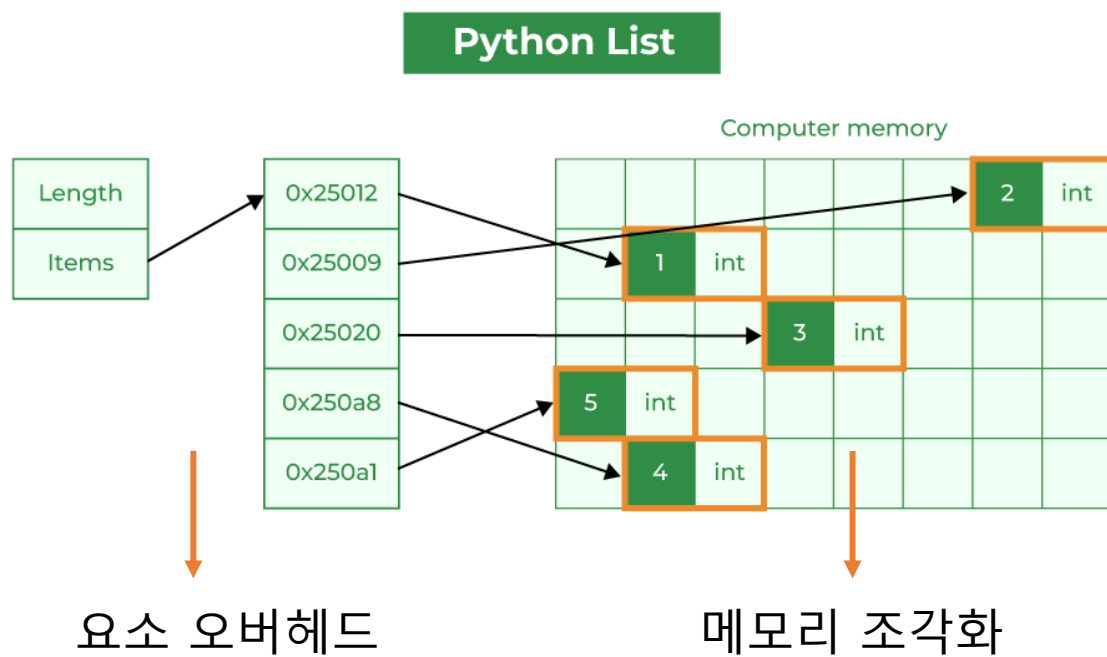
```
result1 = data['array1']
```

```
result2 = data['array2']
```

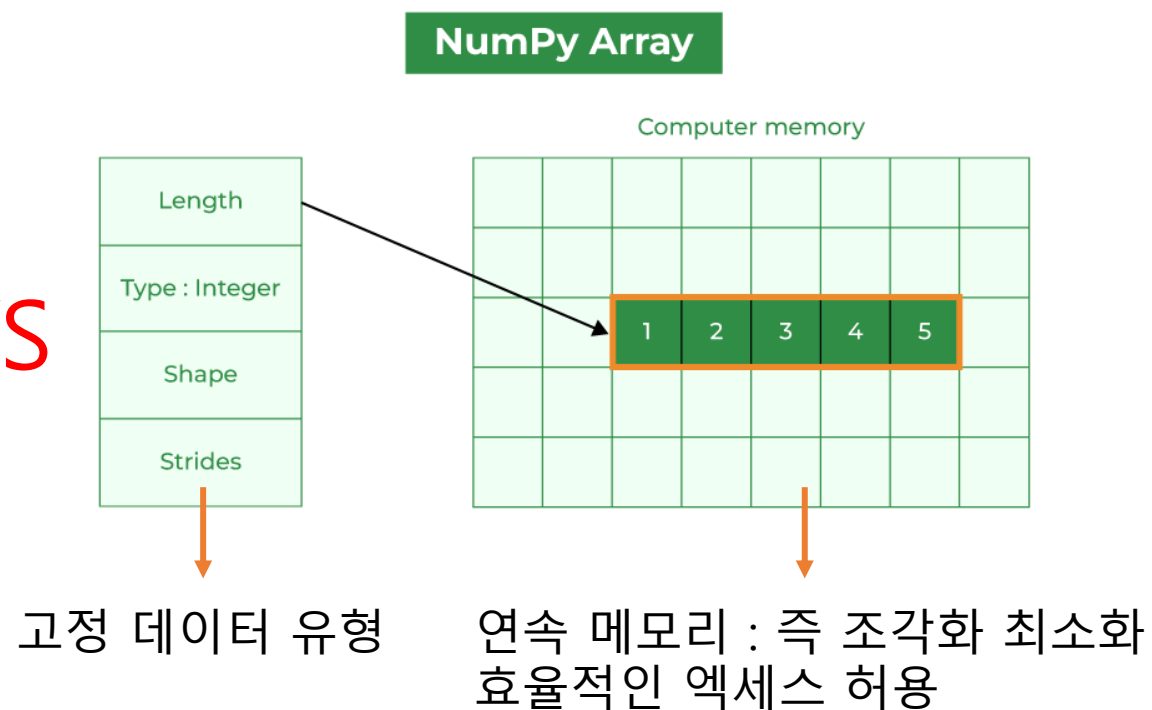
```
print(result1, result2)
```

객체 호출 방식은 위에서 저장에서 지정한 Key값으로 호출

## # 파이썬 리스트와 NumPy 랑 차이점



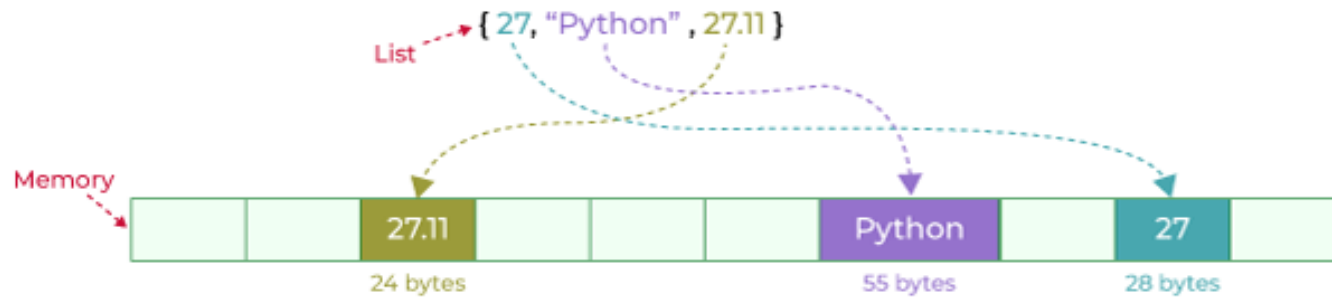
VS



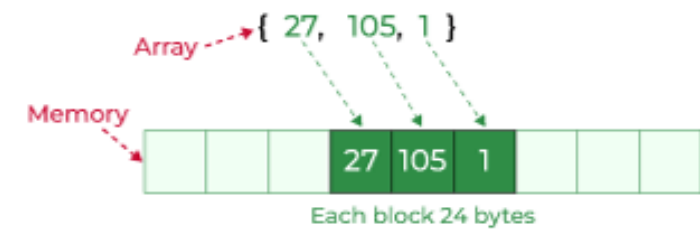


## # 파이썬 리스트와 NumPy 랑 차이점

- 파이썬 리스트와 NumPy 메모리 할당 차이



**Python List**



**NumPy Array**

- Memory size in bytes are in Mac OS X.

An isometric illustration on a blue background showing a futuristic AI environment. Several people and robots are interacting with floating, light-blue rectangular blocks. One robot is pushing a stroller, another is sitting on a block, and others are standing or sitting near various icons like a house, a gear, and a music note. The scene is set against a dark blue background with floating hexagonal shapes.

# Pickle 소개

---



`pickle`은 파이썬에서 사용하는 **딕셔너리, 리스트, 클래스** 등의 자료형을 변환 없이 그대로 파일로 저장하고 이를 불러올 때 사용하는 모듈

## # pickle 저장 하는 방법

```
import pickle

data = {
    'a': [1, 2.0, 3, 4+6j],
    'b': ("character string", b"byte string"),
    'c': {None, True, False}
}

# save
with open('data.pickle', 'wb') as f:
    pickle.dump(data, f, pickle.HIGHEST_PROTOCOL)
```

} 딕셔너리 데이터

} pickle 저장

↑  
저장할 데이터↑  
메모리 부족 오류 해결을 위한 인수 (대용량 피클 파일 저장하는 경우)

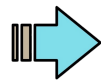
## # pickle 데이터 읽기

```
import pickle

# load
with open('data.pickle', 'rb') as f:
    data = pickle.load(f)
print(data)
```

저장된 피클 경로

저장된 데이터 확인



```
{'a': [1, 2.0, 3, (4+6j)], 'b': ('character string', b'byte string'),
'c': {False, True, None}}
```

기존 원본 저장된 데이터와 동일한 것을 확인 가능

## # pickle gzip을 이용하여 pickle 저장된 데이터를 압축하고 해제하는 예제

```
import pickle
import gzip

data = {
    'a': [1, 2.0, 3, 4+6j],
    'b': ("character string", b"byte string"),
    'c': {None, True, False}
}

# save and compress.
with gzip.open( filename: 'testPickleFile.pickle', mode: 'wb') as f:
    pickle.dump(data, f)
```

} 데이터

} 압축 → pickle

## # pickle gzip을 이용하여 pickle 저장된 데이터를 압축하고 해제하는 예제

```
import pickle
import gzip
```

```
# load and uncompress.
```

```
with gzip.open(filename: 'testPickleFile.pickle', mode: 'rb') as f:
    data = pickle.load(f)
```

} 해제 하고 → pickle 파일 읽기

```
print(data) ← 저장 데이터 확인
```



{'a': [1, 2.0, 3, (4+6j)], 'b': ('character string', b'byte string'), 'c': {False, True, None}}

감사합니다.

