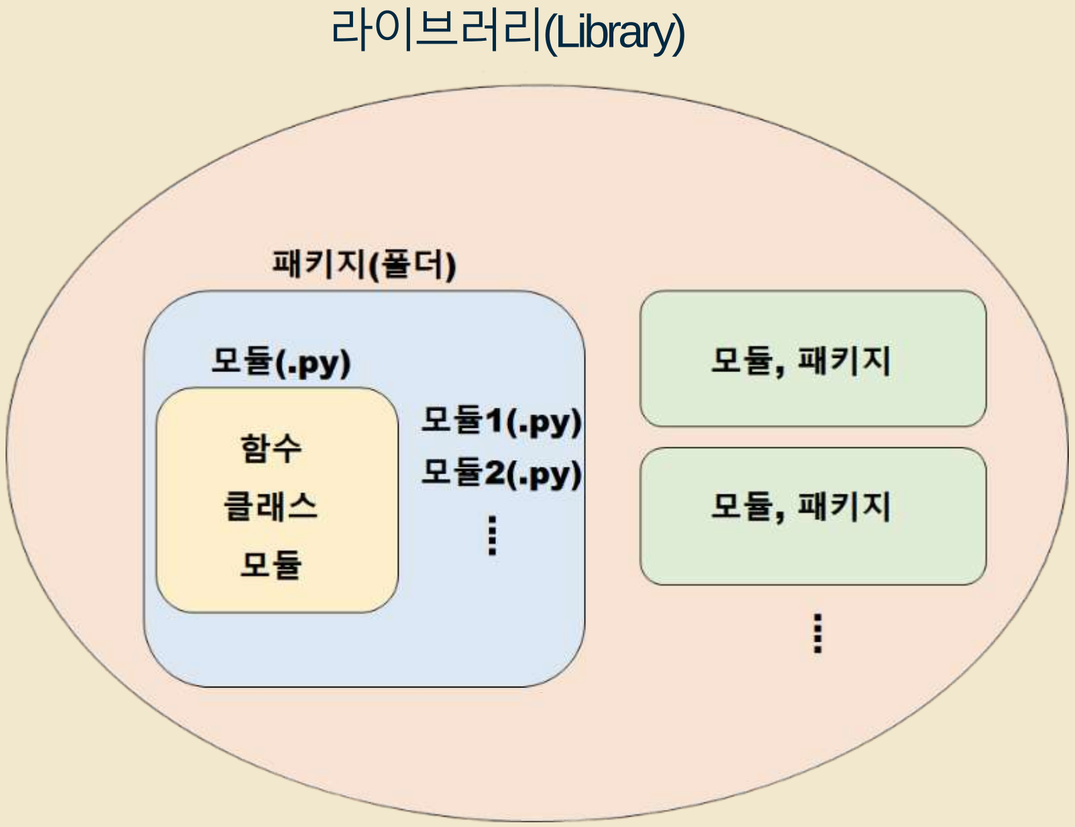
밑바닥부터 시작하는 딥러닝

CHAPTER 01 헬로 파이썬

김지현

용어 정리

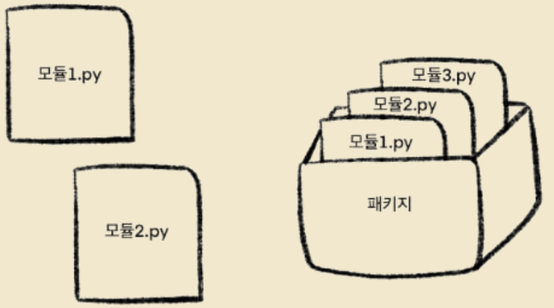


모듈 (Module)

각종 변수, 함수, 클래스를 담고 있는 python 파일(.py)
어떤 기능을 만들고자 할때, 작성을 하게되고, 이를 모듈이라 통칭함

패키지 (Package)

여러 모듈(Module)을 묶은 폴더



라이브러리 (Library)

라이브러리는 재사용 가능한 코드 모음집을 가리키는 포괄적인 용어
패키지와 모듈의 집합체라고 볼 수 있음
- numpy, matplotlib

산술연산자

```
>>> 2+3
5
>>> "hi "+"DL!"
'hi DL!'
```

```
>>> 2*3
6
>>> 2*"hi"
'hihi'
```

```
>>> 2/3
0.6666666666666666
>>> 2//3
0
>>> 2%3
2
```

```
>>> 2-3
-1
>>> "hi " - "DL!"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

```
>>> 2/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

```
>>> 2**3
8
>>> 2**"hi"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for ** or pow(): 'int' and 'str'
```

변수

변수란 '변할 수 있는 것' 어떠한 값을 넣는 **상자**

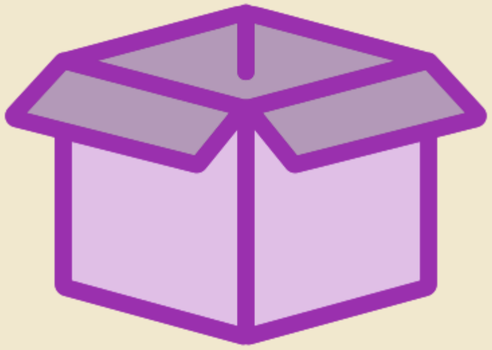
10 > 100

a

2

b

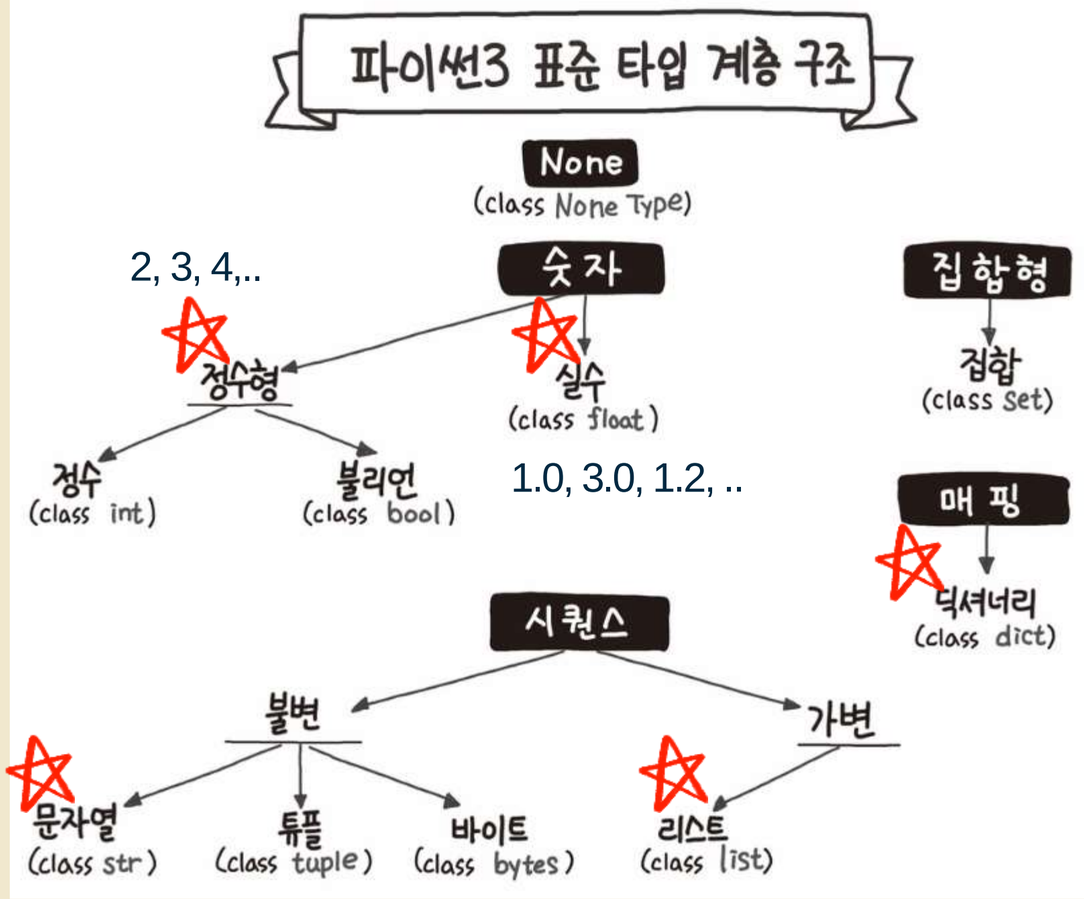
a*b



c

```
>>> a = 10
>>> print(a)
10
>>> a=100
>>> print(a)
100
>>> b=2
>>> print(a+b)
102
>>> c=a*b
>>> print(c)
200
```

자료형



리스트 생성

```python
# 빈 리스트 생성
empty_list = []

# 요소가 포함된 리스트 생성
fruits = ["apple", "banana", "cherry"]
numbers = [1, 2, 3, 4, 5]
```

리스트 요소 추가

```python
fruits.append("orange")  # 리스트 끝에 요소 추가
print(fruits)  # ["apple", "banana", "cherry", "orange"]

fruits.insert(1, "grape")  # 특정 인덱스에 요소 삽입
print(fruits)  # ["apple", "grape", "banana", "cherry", "orange"]
```

딕셔너리 생성

```python
# 빈 딕셔너리 생성
empty_dict = {}

# 요소가 포함된 딕셔너리 생성
person = {"name": "Alice", "age": 25, "city": "New York"}
```

값 접근 및 수정

```python
# 값 접근
print(person["name"])  # "Alice"

# 값 수정
person["age"] = 26
print(person)  # {'name': 'Alice', 'age': 26, 'city': 'New York'}
```

클래스 (Class)

함수

```python
# 함수 정의
def greet():
    print("Hello, World!")

# 함수 호출
greet()  # "Hello, World!" 출력
```

파이썬에서는 def 키워드를 사용하여 함수를 정의

> 함수가 정의된 이후에는 함수 이름을 사용하여 호출

```python
# 매개변수를 사용하는 함수 정의
def greet(name):
    print(f"Hello, {name}!")

# 인자 전달
greet("Alice")  # "Hello, Alice!" 출력
```

함수는 매개변수(parameter)를 사용하여 외부에서 값을 전달받을 수 있음

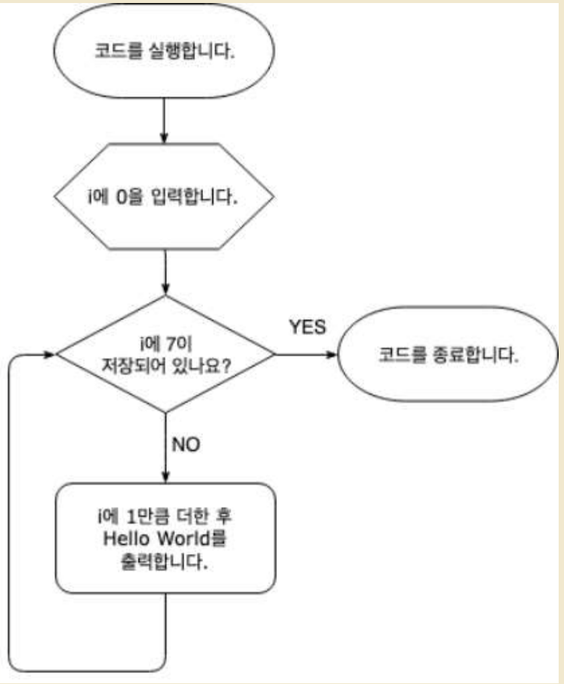
> 함수를 호출할 때 전달하는 값을 인자(argument)라고 함

```python
def add(a, b):
    return a + b

result = add(3, 5)
print(result)  # 8
```

함수는 return 키워드를 사용하여 값을 반환할 수 있음

> 반환값이 없는 함수는 기본적으로 None을 반환합니다.

조건문 / 반복문

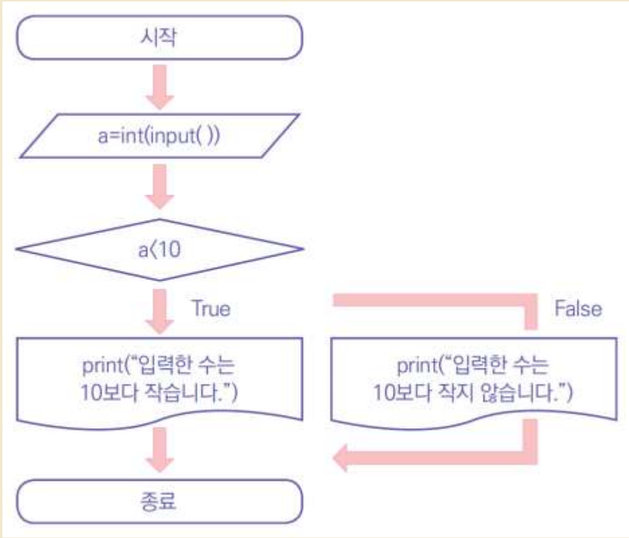


while 문

```python
i = 0  # i에 0을 입력합니다.

while i != 7:  # i에 7이 저장되어 있는지 확인합니다.
    print("Hello World")  # Hello World를 출력합니다.
    i += 1  # i에 1을 더합니다.

print("코드를 종료합니다.")  # i가 7이 되면 코드가 종료됩니다.
```

```python
for i in range(7):  # i가 0부터 6까지 반복합니다.
    print("Hello World")  # Hello World를 출력합니다.

print("코드를 종료합니다.")  # for 문이 끝나면 코드가 종료됩니다.
```

for 문




```python
a = int(input("숫자를 입력하세요: "))  # 사용자로부터 입력을 받습니다.

if a < 10:
    print("입력한 수는 10보다 작습니다.")  # 조건이 참일 때 출력
else:
    print("입력한 수는 10보다 작지 않습니다.")  # 조건이 거짓일 때 출력
```

numpy

Numpy 라이브러리는 하나의 리스트를 입력으로 받아서 Numpy 객체를 만들 수 있도록 해준다.

```python
import numpy as np


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

print(array.size) # 배열의 크기

print(array.dtype) # 배열 원소의 타입

print(array[2]) # 인덱스 2의 원소
```

```python
array2 = np.array([
    [1,2,3],
    [4,5,6],
    [7,8,9]
])
```

1. 표준 파이썬에는 포함되지 않는 '외부'라이브러리를 가져옴

   >> import

2. 파이썬의 리스트를 인수로 받아 넘파이 라이브러리가 제공하는 특수한 형태의 배열(numpy.ndarray)를 반환

   >> np.array()

3. 산술 연산 가능, 원소 수가 다르면 오류 발생 주의
4. 다차원 배열 생성 가능
5. 인덱스, 반복문을 이용해서 원소 접근 가능

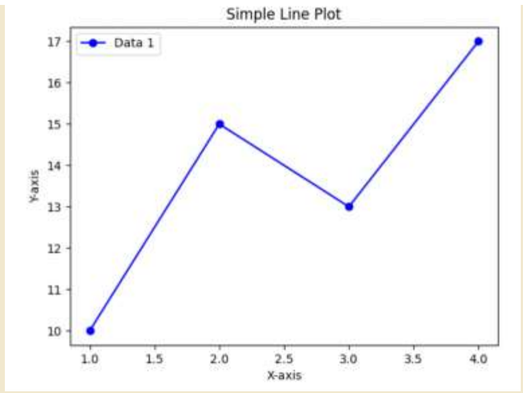
```python
# 행 단위로 출력
for row in array2:
    for element in row:
        print(element, end=' ')
    print()  # 행을 출력한 후 줄바꿈
```

```
1 2 3
4 5 6
7 8 9
```

matplotlib

파이썬에서 데이터 시각화를 위한 가장 인기 있는 라이브러리 중 하나
데이터 과학과 머신러닝 프로젝트에서 데이터를 분석하거나 시각화할 때 자주 사용

```python
import matplotlib.pyplot as plt

# 데이터 설정
x = [1, 2, 3, 4]
y = [10, 15, 13, 17]

# 선 그래프 생성
plt.plot(x, y, label="Data 1", color="blue", marker="o")  # 데이터와 스타일 설정
plt.xlabel("X-axis")  # x축 라벨
plt.ylabel("Y-axis")  # y축 라벨
plt.title("Simple Line Plot")  # 그래프 제목
plt.legend()  # 범례 표시
plt.show()  # 그래프 보여주기
```



1. 축 설정: plt.xlabel()과 plt.ylabel()로 x축과 y축의 라벨을 설정
2. 제목 설정: plt.title()로 그래프의 제목을 설정
3. 범례: 여러 데이터셋이 있는 경우 plt.legend()로 범례를 표시 가능
4. 레이아웃 조정: plt.tight_layout()를 사용하여 그래프 요소 간의 여백을 자동으로 조정 가능
5. 서브플롯: plt.subplot()을 사용하면 하나의 창에 여러 그래프를 배열 가능
6. 종류: 선 그래프, 산점도, 막대 그래프, 히스토그램 등

Thank You