AI Programming

Python

03. Python Basic 3

first coding

클래스와 객체지향

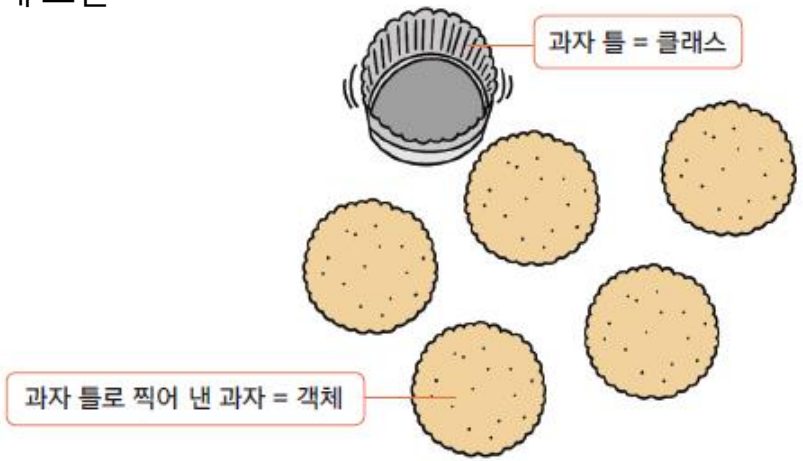
클래스

클래스

- **클래스와 객체**

  - **클래스**(class)
    - 똑같은 무언가를 계속 만들어 낼 수 있는 설계 도면

  - **객체**(object)
    - 클래스로 만든 결과물



클래스

- **클래스와 객체**

  - 클래스로 만든 객체의 특징
    - 객체마다 고유한 성격을 가짐
    - 동일한 클래스로 만든 객체들은 서로 전혀 영향을 주지 않음

  - 파이썬 클래스의 가장 간단한 예
    - Cookie 클래스의 객체를 만드는 방법

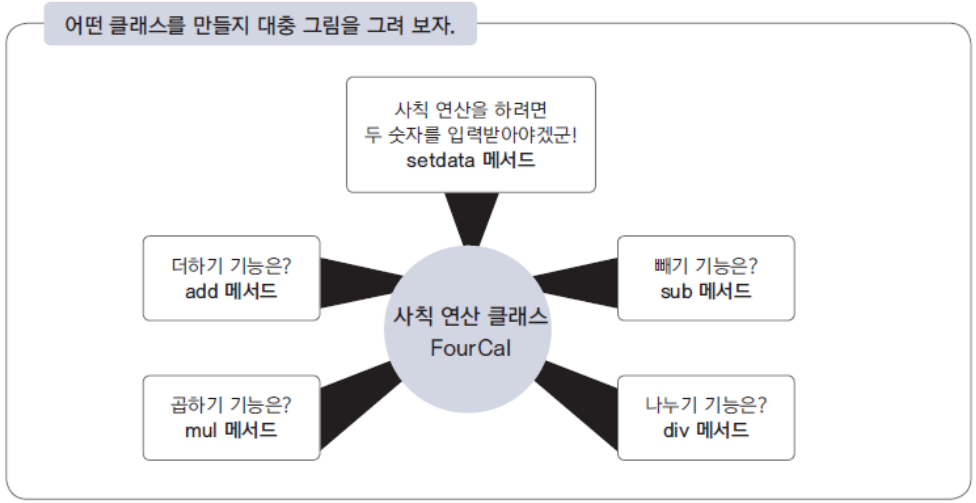
```
>>> class Cookie:
...     pass
```

```
>>> a = Cookie()
>>> b = Cookie()
```

클래스

- 사칙연산 클래스 만들기

  - 클래스를 어떻게 만들지 먼저 구상하기
    - 사칙연산을 가능하게 하는 FourCal 클래스 만들기



클래스

- 사칙연산 클래스 만들기

  - 클래스 구조 만들기
    - Pass라는 문장만을 포함한 FourCal 클래스 만들기
    - FourCal 클래스는 아무 변수나 함수도 포함하지 않지만 객체를 만들 수 있는 기능이 있음

```
>>> class FourCal:
...     pass
```

```
>>> a = FourCal()
>>> type(a)
<class '__main__.FourCal'> <— 객체 a의 타입은 FourCal 클래스이다.
```

클래스

- **사칙연산 클래스 만들기**

  - 객체에 연산할 숫자 지정하기
    - 더하기 · 나누기 · 곱하기 · 빼기 등의 기능을 하는 객체 만들기
    - 우선 객체에 사칙 연산을 할 때 사용할 2개의 숫자를 알려 주어야 함
    - pass 문장을 삭제하고 setdata 함수 정의

```
>>> class FourCal:
...     def setdata(self, first, second):
...         self.first = first
...         self.second = second
```

```
>>> a = FourCal()
```

```
>>> a.setdata(4, 2)
```

- **사칙연산 클래스 만들기**

  - 객체에 연산할 숫자 지정하기
    - **메서드**(method) : 클래스 안에 구현된 함수
    - 일반적인 함수
    - 메서드도 클래스에 포함되어 있다는 점만 제외하면 일반 함수와 같음

```
def 함수_이름(매개변수):
    수행할_문장
    ...
```

```
def setdata(self, first, second):   ← ① 메서드의 매개변수
    self.first = first
    self.second = second            ② 메서드의 수행문
```

- 사칙연산 클래스 만들기

  - 객체에 연산할 숫자 지정하기

```
def setdata(self, first, second): ← ① 메서드의 매개변수
```

① setdata 메서드의 매개변수

  - self, first, second 3개의 입력 값
  - 일반 함수와는 달리 메서드의 첫 번째 매개변수 self는 특별한 의미를 가짐
  - a 객체를 만들고 a 객체를 통해 setdata 메서드 호출하기

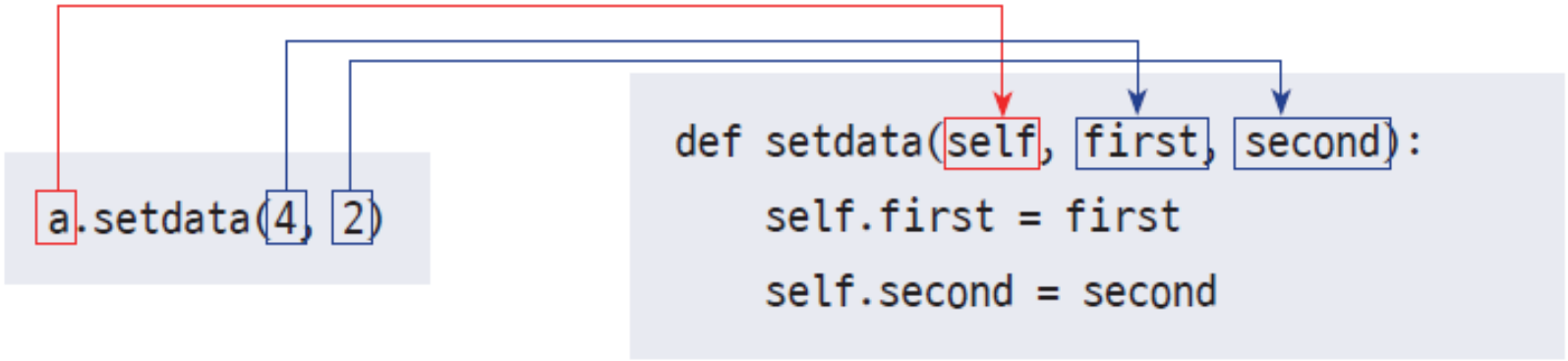
```
>>> a = FourCal()
>>> a.setdata(4, 2)
```

- 사칙연산 클래스 만들기

  - 객체에 연산할 숫자 지정하기

    ① setdata 메서드의 매개변수

      - setdata 메서드의 첫 번째 매개변수 self에는 setdata 메서드를 호출한 객체 a가 자동으로 전달



- 사칙연산 클래스 만들기

② setdata 메서드의 수행문

– a 객체에 객체변수 first와 second가 생성되고 지정된 값이 저장됨

```
    self.first = first
    self.second = second
```

② 메서드의 수행문

```
>>> a = FourCal()
>>> a.setdata(4, 2)
>>> a.first   <-- a 객체의 first 변숫값 출력
4
>>> a.second  <-- a 객체의 second 변숫값 출력
2
```

클래스

- 사칙연산 클래스 만들기

  - 객체에 연산할 숫자 지정하기
    - 객체의 객체변수 특징 살펴보기
      - a, b 객체 생성
      - b 객체의 객체변수 first 생성
      - a 객체의 객체변수 first 생성

```
>>> a = FourCal()
>>> b = FourCal()
```

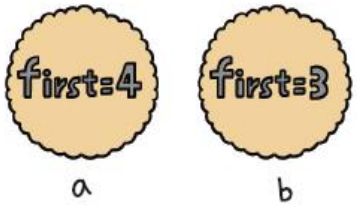
```
>>> a.setdata(4, 2) <— a 객체에 객체변수 first와 second가 생성되고 값 4와 2 대입
>>> a.first <— a 객체의 first 값 출력
4
```

```
>>> b.setdata(3, 7) <— b 객체에 객체변수 first와 second가 생성되고 값 3과 7 대입
>>> b.first <— b 객체의 first 값 출력
3
```

- **사칙연산 클래스 만들기**

  - 객체에 연산할 숫자 지정하기
    - 객체의 객체변수 특징 살펴보기
      - b 객체의 객체변수 first에 3이 저장됐을 때, a 객체의 first는 3으로 변할까? 아니면 4를 유지할까?

```
>>> a.first
4
```



        » a 객체의 first 값은 b 객체의 first 값에 영향 받지 않고 원래 값을 유지!

      - **클래스로 만든 객체의 객체변수는 다른 객체의 객체변수와 상관없이 독립적인 값을 유지!**

클래스

- 사칙연산 클래스 만들기

  - 더하기 기능 만들기
    - 클래스에 2개의 숫자를 더하는 add 메서드 추가
    - 클래스를 사용해 add 메서드 호출

```
>>> class FourCal:
...     def setdata(self, first, second):
...         self.first = first
...         self.second = second
...     def add(self):
...         result = self.first + self.second
...         return result
```

```
>>> a = FourCal()
>>> a.setdata(4, 2)
>>> a.add()
6
```

- 사칙연산 클래스 만들기

  - 곱하기, 빼기, 나누기 기능 만들기
    - add 메서드와 동일한 방법으로 mul, sub, div 메서드 생성

```
...     def add(self):
...         result = self.first + self.second
...         return result
...     def mul(self):
...         result = self.first * self.second
...         return result
...     def sub(self):
...         result = self.first - self.second
...         return result
...     def div(self):
...         result = self.first / self.second
...         return result
```

```
>>> a = FourCal()
>>> b = FourCal()
>>> a.setdata(4, 2)
>>> b.setdata(3, 8)
>>> a.add()
6
>>> a.mul()
8
>>> a.sub()
2
```

```
>>> a.div()
2.0
>>> b.add()
11
>>> b.mul()
24
>>> b.sub()
-5
>>> b.div()
0.375
```

- 생성자

  - AttributeError 오류
    - FourCal 클래스의 객체 a에 setdata 메서드를 수행하지 않고 add 메서드를 수행하면, 'AttributeError: 'FourCal' object has no attribute 'first'' 오류 발생
    - setdata 메서드를 수행해야 객체 a의 객체변수 first와 second가 생성되기 때문

```
>>> a = FourCal()
>>> a.add()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 6, in add
AttributeError: 'FourCal' object has no attribute 'first'
```

클래스

- 생성자

  - 생성자(constructor)
    - 객체가 생성될 때 자동으로 호출되는 메서드
    - 메서드명으로 __init__ 사용
    - 객체에 초기 값을 설정해야 할 필요가 있을 때 생성자를 구현하는 것이 안전한 방법
    - 객체 생성 시 생성자의 매개변수에 해당하는 값을 전달해야 함
    - __init__ 메서드의 매개변수에 전달되는 값

```
>>> class FourCal:
...     def __init__(self, first, second):
...         self.first = first
...         self.second = second
```

```
>>> a = FourCal(4, 2)
```

```
>>> a = FourCal(4, 2)
>>> a.first
4
>>> a.second
2
```

- 클래스의 상속

  - 상속(Inheritance)
    - '물려받다'라는 뜻
    - 어떤 클래스를 만들 때 다른 클래스의 기능을 물려받을 수 있게 만드는 것

```
class 클래스_이름(상속할_클래스_이름)
```

    - FourCal 클래스를 상속하는 MoreFourCal 클래스

```
>>> class MoreFourCal(FourCal):
...     pass
```

클래스

- 클래스의 상속

  - MoreFourCal 클래스

    - FourCal 클래스를 상속했으므로 FourCal 클래스의 모든 기능을 사용할 수 있어야 함

```
>>> a = MoreFourCal(4, 2)
>>> a.add()
6
>>> a.mul()
8
>>> a.sub()
2
>>> a.div()
2.0
```

클래스

- 클래스의 상속
  - a^b를 계산하는 MoreFourCal 클래스
    - MoreFourCal 클래스로 만든 a 객체에 값 4와 2를 지정한 후 pow 메서드 호출

```
>>> class MoreFourCal(FourCal):
...     def pow(self):
...         result = self.first ** self.second
...         return result
```

```
>>> a = MoreFourCal(4, 2)
>>> a.pow()
16
>>> a.add()
6
```

- 메서드 오버라이딩(method overriding)

  - 부모 클래스(상속한 클래스)에 있는 메서드를 동일한 이름으로 다시 만드는 것
  - FourCal 클래스를 상속하는 SafeFourCal 클래스
    - FourCal 클래스 대신 SafeFourCal 클래스를 사용

```
>>> class SafeFourCal(FourCal):
...     def div(self):
...         if self.second == 0:  <— 나누는 값이 0인 경우, 값 0을 리턴하도록 수정
...             return 0
...         else:
...             return self.first / self.second
```

```
>>> a = SafeFourCal(4, 0)
>>> a.div()
0
```

first coding

모듈과 패키지

모듈

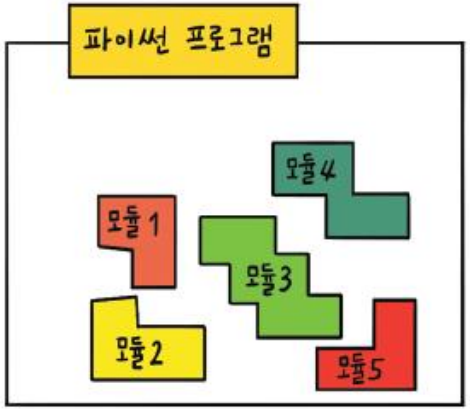
패키지

모듈

- **모듈 만들기**

  - 모듈
    - 함수나 변수 또는 클래스를 모아 놓은 **파일**
    - 다른 파이썬 프로그램에서 불러와 사용할 수 있도록 만든 **파이썬 파일**

  - add와 sub 함수만 있는 파일(모듈) mod1.py




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

def sub(a, b):
    return a - b
```

- **모듈 불러오기**

  - 대화형 인터프리터에서 mod1.py 불러오기
    - 'import mod1'이라고 입력하여, mod1.py 불러오기
    - mod1.py 파일에 있는 add 함수를 사용
      → 모듈 이름(mod1) 뒤에 도트 연산자(.)를 붙이고 함수 이름을 입력

```
>>> import mod1
>>> print(mod1.add(3, 4))
7
>>> print(mod1.sub(4, 2))
2
```

모듈

- **모듈 불러오기**

  - import
    - 이미 만들어 놓은 파이썬 모듈을 사용할 수 있게 해 주는 명령어
    - mod1.py에서 확장자 .py를 제거한 mod1만이 모듈 이름

```
import 모듈_이름
```

    - mod1.add, mod1.sub처럼 쓰지 않고 모듈 이름 없이 함수 이름만 쓰고 싶은 경우

```
from 모듈_이름 import 모듈_함수
```

```
>>> from mod1 import add
>>> add(3, 4)
7
```

- **모듈 불러오기**

  - import
    - 모듈의 함수를 여러 개 불러오고 싶은 경우

      1) 쉼표(,)로 구분하여 필요한 함수 불러오기

```
from mod1 import add, sub
```

      2) * 문자 사용하기

```
from mod1 import *
```

» * 문자는 '모든 것'이라는 뜻으로, 모듈의 모든 함수를 불러와 사용하겠다는 의미

모듈

- if __name__ == "__main__":의 의미
  - mod1.py 파일을 다음과 같이 수정
    - mod1 모듈을 import할 때 mod1.py 파일이 실행되어 **결과 값을 출력하는 문제**

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

def sub(a, b):
    return a - b

print(add(1, 4))
print(sub(4, 2))
```

```
>>> import mod1
5
2
```

모듈

- if __name__ == "__main__":의 의미

  - mod1.py 파일을
    if __name__ == "__main__": 을 사용하여 수정
    - mod1.py 을 실행 하면
      - __name__ 변수에 __main__ 값이 저장
      - __name__ == "__main__" 문은 실행
    - 다른 파일에서 이 모듈을 불러와 사용할 경우
      - __name__ 변수에 모듈 이름 mod1이 저장
      - __name__ == "__main__" 문은 실행 되지 않음

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

def sub(a, b):
    return a - b

if __name__ == "__main__":
    print(add(1, 4))
    print(sub(4, 2))
```

```
>>> import mod1
>>>
```

- **클래스나 변수 등을 포함한 모듈**

  - mod2.py 파일 만들기
    - 원의 넓이를 계산하는 Math 클래스
    - 두 값을 더하는 add 함수
    - 원주율 값 PI

```
PI = 3.141592

class Math:
    def solv(self, r):
        return PI * (r ** 2)

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

```
>>> import mod2
>>> print(mod2.PI) <— PI 변수 사용
3.141592
```

```
>>> a = mod2.Math() <— Math 클래스 사용
>>> print(a.solv(2))
12.566368
```

```
>>> print(mod2.add(mod2.PI, 4.4)) <— add 함수 사용
7.541592
```

first coding

표준라이브러리

표준 라이브러리

- **파이썬 표준 라이브러리**

  - 라이브러리(library) = 도서관(원하는 정보를 찾아보는 곳)
  - 파이썬 라이브러리는 전 세계의 파이썬 고수들이 만든 유용한 프로그램을 모아 놓은 것
  - 파이썬을 설치할 때 자동으로 컴퓨터에 설치됨
  - 자주 사용되고 꼭 알아 두면 좋은 라이브러리 소개

표준 라이브러리

- datetime.date

  – 연, 월, 일로 날짜를 표현할 때 사용하는 함수

A 군과 B 양이 2021년 12월 14일부터 만나기 시작했다면 2023년 4월 5일은 둘이 사귄 지 며칠째 되는 날일까?

```
>>> import datetime
>>> day1 = datetime.date(2021, 12, 14)
>>> day2 = datetime.date(2023, 4, 5)
```

```
>>> diff = day2 - day1
>>> diff.days
477  <-- 둘이 만난 지 477일째
```

표준 라이브러리

- time

1) time.time()
2) time.localtime()
3) time.asctime()
4) time.ctime()
5) time.strftime()
6) time.sleep()

```
>>> import time
>>> time.time()
1684983953.5221913
```

```
>>> time.localtime(time.time())
time.struct_time(tm_year=2023, tm_mon=5, tm_mday=21, tm_hour=16,
    tm_min=48, tm_sec=42, tm_wday=1, tm_yday=141, tm_isdst=0)
```

```
>>> time.asctime(time.localtime(time.time()))
'Fri Apr 28 20:50:20 2023'
```

```
>>> time.ctime()
'Fri Apr 28 20:56:31 2023'
```

```
>>> import time
>>> time.strftime('%x', time.localtime(time.time()))
'05/25/23'  <-- 현재 설정된 지역의 날짜 출력
>>> time.strftime('%c', time.localtime(time.time()))
'Thu May 25 10:13:52 2023'  <-- 날짜와 시간 출력
```

표준 라이브러리

- random
  - 난수(규칙이 없는 임의의 수)를 발생시키는 모듈

1) random.random
2) random.sample
3) random.choice
4) random.sample

```
>>> import random
>>> random.random()
0.53840103305098674
```

```
>>> random.randint(1, 10)
6
```

```
def random_pop(data):
    number = random.choice(data)
    data.remove(number)
    return number
```

```
>>> import random
>>> data = [1, 2, 3, 4, 5]
>>> random.sample(data, len(data))
>>> data
[5, 1, 3, 4, 2]
```

표준 라이브러리

- OS

  – 환경 변수나 디렉터리, 파일 등의
  OS 자원을 제어할 수 있게 해 주는 모듈

1) os.environ
2) os.chdir
3) os.getcwd
4) os.system
5) os.popen
6) os.mkdir / os.rmdir
7) os.remove
8) os.rename

```
>>> import os
>>> os.environ
environ({'PROGRAMFILES': 'C:\\Program Files', 'APPDATA': ...생략...})
```

```
>>> os.environ['PATH']
'C:\\ProgramData\\Oracle\\Java\\javapath;(...생략...)'
```

```
>>> os.chdir("C:\WINDOWS")
```

```
>>> os.getcwd()
'C:\WINDOWS'
```

```
>>> os.system("dir")
```

```
>>> f = os.popen("dir")
```

표준 라이브러리

- json

  – JSON 데이터를 쉽게 처리하고자 사용하는 모듈

예) 개인정보를 JSON 형태의 데이터로 만든 myinfo.json 파일을 읽어 딕셔너리로 변환

```
{
    "name": "홍길동",
    "birth": "0525",
    "age": 30
}
```

```
>>> import json
>>> with open('myinfo.json') as f:
...     data = json.load(f)
...
>>> type(data)
<class 'dict'>
>>> data
{'name': '홍길동', 'birth': '0525', 'age': 30}
```

표준 라이브러리

- json
  - 딕셔너리 자료형을 JSON 형태로 생성
  - 파이썬 자료형을 JSON 문자열로 만드는 방법
  - JSON 문자열을 딕셔너리로 변환

```
>>> import json
>>> data = {'name': '홍길동', 'birth': '0525', 'age': 30}
>>> with open('myinfo.json', 'w') as f:
...     json.dump(data, f)
```

```
>>> import json
>>> d = {"name":"홍길동", "birth":"0525", "age": 30}
>>> json_data = json.dumps(d)
>>> json_data
'{"name": "\\ud64d\\uae38\\ub3d9", "birth": "0525", "age": 30}'
```

```
>>> json.loads(json_data)
{'name': '홍길동', 'birth': '0525', 'age': 30}
```

표준 라이브러리

- json
  - 한글 문자열이 아스키 형태의 문자열로 변경되는 것을 방지하는 방법
  - 딕셔너리 외에 리스트나 튜플처럼 다른 자료형도 JSON 문자열로 변경 가능
  - 출력되는 JSON 문자열을 보기 좋게 정렬하는 방법

```
>>> d = {"name":"홍길동", "birth":"0525", "age": 30}
>>> json_data = json.dumps(d, ensure_ascii=False)
>>> json_data
'{"name": "홍길동", "birth": "0525", "age": 30}'
>>> json.loads(json_data)
{'name': '홍길동', 'birth': '0525', 'age': 30}
```

```
>>> d = {"name":"홍길동", "birth":"0525", "age": 30}
>>> print(json.dumps(d, indent=2, ensure_ascii=False))
{
  "name": "홍길동",
  "birth": "0525",
  "age": 30
}
```

```
>>> json.dumps([1, 2, 3])
'[1, 2, 3]'
>>> json.dumps((4, 5, 6))
'[4, 5, 6]'
```

first coding

넘파이 (Numpy)

넘파이 (Numpy) 라이브러리

넘파이 (Numpy) 라이브러리의 주요기능 과 문법

인공지능 모델 개발에 필요한 기능

넘파이 (Numpy) 라이브러리

- **넘파이**
  - 넘파이는 수치 계산에 특화된 라이브러리
  - 다차원 배열 객체인 ndarray를 중심으로 작동
    → 고성능 다차원 배열 객체를 제공하며, 요소의 데이터 타입을 통일
  - 다양한 함수 지원
    → 배열 연산, 선형 대수, 푸리에 변환, 난수 생성 등 **다양한 수학적 기능을 제공**

넘파이 (Numpy) 라이브러리

- **주요기능 과 문법**

  - **배열 생성**
    - np.array(), np.zeros(), np.ones(), np.arange(), np.linspace() 등
  - **배열 연산**
    - +, -, *, /와 같은 기본 연산과 np.dot()(행렬 곱), np.sum(), np.mean() 등 통계 연산
  - **배열 변형**
    - reshape(), flatten(), transpose()
  - **브로드캐스팅**
    - 배열 크기가 달라도 호환 가능하면 연산 수행

넘파이 (Numpy) 라이브러리

- **넘파이 설치**

pip install numpy

예제

```
import numpy as np

# 배열 생성
arr = np.array([1, 2, 3, 4, 5])
print(arr + 10)  # [11, 12, 13, 14, 15]

# 2D 배열 생성 및 연산
matrix = np.array([[1, 2], [3, 4]])
print(matrix.T)  # 전치행렬 [[1, 3], [2, 4]]
```

넘파이 (Numpy) 라이브러리

- **기본 문법**

  - 넘파이(Numpy)는 배열(ndarray)을 중심으로 이루어져 있음
    - 배열 생성, 조작, 계산 등을 이해하면 넘파이를 쉽게 다룰 수 있음

1. 배열 생성: np.array(), np.zeros(), np.ones(), np.arange()
2. 배열 조작: 인덱싱, 슬라이싱, reshape(), flatten()
3. 수학 연산: 요소별 연산, 브로드캐스팅
4. 통계 함수: sum(), mean(), max(), min()
5. 난수 생성: np.random.rand(), np.random.randint()

넘파이 (Numpy) 라이브러리

- **기본 문법**

1. 넘파이 배열 생성
   - 기본 배열 생성
     - np.array()
       : 넘파이 배열 생성
     - Python 리스트를 배열로 변환

```
import numpy as np

# 1D 배열 생성
arr1 = np.array([1, 2, 3, 4])
print(arr1)
```

```
[1 2 3 4]
```

```
# 2D 배열 생성
arr2 = np.array([[1, 2], [3, 4]])
print(arr2)
```

```
[[1 2]
 [3 4]]
```

넘파이 (Numpy) 라이브러리

- **기본 문법**

1. 넘파이 배열 생성
   - 특별한 배열 생성
     - 배열 생성에 유용한 함수를 제공

```
# 0으로 채워진 배열
zeros = np.zeros((2, 3))
print(zeros)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

```
# 1로 채워진 배열
ones = np.ones((3, 2))
print(ones)
```

```
[[1. 1.]
 [1. 1.]
 [1. 1.]]
```

```
# 연속적인 숫자로 배열 생성
# 0부터 10까지 2 간격으로 생성
sequence = np.arange(0, 10, 2)
print(sequence)
```

```
[0 2 4 6 8]
```

```
# 일정 범위에서 동일 간격으로 숫자 생성
# 0부터 1까지 5개로 나눔
linspace = np.linspace(0, 1, 5)
print(linspace)
```

```
[0.   0.25 0.5  0.75 1.  ]
```

넘파이 (Numpy) 라이브러리

- **기본 문법**

2. 배열의 속성 확인
   - 배열의 크기, 차원, 데이터 타입 확인

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

print(arr.shape)  # (2, 3): 배열의 크기 (행, 열)
print(arr.ndim)   # 2: 배열의 차원 (2D 배열)
print(arr.size)   # 6: 배열의 총 원소 개수
print(arr.dtype)  # int64: 배열 원소의 데이터 타입
```

```
(2, 3)  : 배열의 크기 (행, 열)
2       : 배열의 차원 (2D 배열)
6       : 배열의 총 원소 개수
Int64   : 배열 원소의 데이터 타입
```

넘파이 (Numpy) 라이브러리

- **기본 문법**

3. 배열의 인덱싱과 슬라이싱
   - 인덱싱
     - Python 리스트처럼
       배열의 특정 위치에 접근
     - 2D 배열에서는 행과 열을 지정하여 접근

```
arr = np.array([10, 20, 30, 40])

print(arr[0])  # 10
print(arr[-1])  # 40


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

print(arr[0, 1])  # 2 (1행 2열)
print(arr[1, -1])  # 6 (2행 마지막 열)
```

```
10
40
2
6
```

- **기본 문법**

3. 배열의 인덱싱과 슬라이싱
   - 슬라이싱
     - 배열의 일부분을 추출
     - 2D 배열에서도 슬라이싱

```
arr = np.array([10, 20, 30, 40, 50])

print(arr[1:4]) # 1번 인덱스부터 4번 인덱스 전까지
print(arr[:3])  # 처음부터 3번 인덱스 전까지
print(arr[::2]) # 2칸씩 건너뛰며 선택
```

```
[20 30 40] (1번 인덱스부터 4번 인덱스 전까지)
[10 20 30] (처음부터 3번 인덱스 전까지)
[10 30 50] (2칸씩 건너뛰며 선택)
```

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

print(arr[:2, 1:]) # 첫 두 행과 2번째 열부터 끝까지
print(arr[1:, :2]) # 두 번째 행부터, 첫 두 열
```

```
[[2 3]
 [5 6]]

[[4 5]
 [7 8]]
```

- **기본 문법**

4. 배열의 연산
   - 기본 연산
     - 배열 간 연산은 요소별로 수행
   - 브로드캐스팅
     - 크기가 다른 배열 간의 연산

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

print(arr1 + arr2)
print(arr1 * arr2)
print(arr1 ** 2)
```

```
[5 7 9]
[4 10 18]
[1 4 9]
```

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

# 배열에 스칼라 값 더하기
print(arr + 10)
```

```
[[11 12 13]
 [14 15 16]]
```

넘파이 (Numpy) 라이브러리

- **기본 문법**

5. 배열의 형태 변경
   - 배열 형태 변경 (reshape)
     - 배열의 크기를 변경
   - 배열 펼치기 (flatten)
     - 다차원 배열을 1차원 배열로 변환

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

reshaped = arr.reshape(2, 3)  # 2행 3열로 변경
print(reshaped)
```

```
[[1 2 3]
 [4 5 6]]
```

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

flattened = arr.flatten()
print(flattened)
```

```
[1 2 3 4 5 6]
```

넘파이 (Numpy) 라이브러리

- **기본 문법**

6. 유용한 함수들
   - 통계 함수

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

reshaped = arr.reshape(2, 3)  # 2행 3열로 변경
print(reshaped)
```

```
[[1 2 3]
 [4 5 6]]
```

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

flattened = arr.flatten()
print(flattened)
```

```
[1 2 3 4 5 6]
```

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

filtered = arr[arr > 3]
print(filtered)
```

```
[4 5]
```

넘파이 (Numpy) 라이브러리

- **기본 문법**

6. 랜덤 데이터 생성
   - 난수 생성
     - AI 모델의 초기 가중치나 무작위 데이터 생성에 사용

```
# 0에서 1 사이의 랜덤 값
random_values = np.random.rand(3)
print(random_values)
```

```
[0.1234 0.5678 0.9101]
```

```
# 정수 난수 생성
random_ints = np.random.randint(1, 10, size=(2, 3))
print(random_ints)
```

```
[[3 7 2]
 [1 9 8]]
```

넘파이 (Numpy) 라이브러리

- **넘파이 주요 기능과 인공지능에서의 활용**

1. 배열 생성
   - AI 모델은 데이터를 벡터나 행렬 형태로 처리
     → 배열을 쉽게 생성하고 컨트롤

```
import numpy as np

# 입력 데이터: 2개의 특징(feature)을 가진 샘플 5개
data = np.array([[1.2, 3.4],
                 [2.1, 1.3],
                 [3.1, 4.2],
                 [0.5, 0.7],
                 [1.0, 2.5]])

--

# 출력 데이터: 각 샘플의 라벨
labels = np.array([0, 1, 1, 0, 1])

print("데이터:\n", data)
print("라벨:\n", labels)
```

넘파이 (Numpy) 라이브러리

- **넘파이 주요 기능과 인공지능에서의 활용**

2. 벡터화 연산
   - AI에서 입력 데이터와 가중치를 곱하기
   - 손실을 계산
   - 벡터화 연산이 자주 사용

```
# 가중치 벡터
weights = np.array([0.5, -0.2])


# 가중치와 데이터의 내적
outputs = np.dot(data, weights)
print("가중치와 데이터 내적 결과:\n", outputs)
```

```
--
```

넘파이 (Numpy) 라이브러리

- **넘파이 주요 기능과 인공지능에서의 활용**

3. 활성화 함수
   - 신경망에서 활성화 함수는 데이터를 비선형적으로 변환
     → 시그모이드 함수 구현

```
# 시그모이드 함수
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# 내적 결과에 시그모이드 적용
sigmoid_outputs = sigmoid(outputs)
print("시그모이드 함수 결과:\n", sigmoid_outputs)
```

```
--
```

넘파이 (Numpy) 라이브러리

- **넘파이 주요 기능과 인공지능에서의 활용**

4. 난수 생성
   - AI 학습에서 초기 가중치를 무작위로 설정
   - 데이터셋을 무작위로 섞을 때 난수를 사용

   → 초기 가중치와 데이터 섞기

```
# 초기 가중치 생성 (난수)
initial_weights = np.random.rand(2)
print("초기 가중치:\n", initial_weights)

# 데이터 섞기
shuffled_indices = np.random.permutation(len(data))
shuffled_data = data[shuffled_indices]
shuffled_labels = labels[shuffled_indices]

print("섞인 데이터:\n", shuffled_data)
print("섞인 라벨:\n", shuffled_labels)
```

```
--
```

넘파이 (Numpy) 라이브러리

- **넘파이 주요 기능과 인공지능에서의 활용**

5. 행렬 연산
   - 딥러닝에서는 입력 데이터를 여러 층의 가중치와 곱해 결과를 계산

```
# 입력 데이터
input_data = np.array([[1.0, 2.0],
                       [0.5, 1.5]])

# 첫 번째 층 가중치와 바이어스
weights_layer1 = np.array([[0.2, 0.4],
                           [-0.5, 0.3]])
bias_layer1 = np.array([0.1, -0.2])

# 두 번째 층 가중치와 바이어스
weights_layer2 = np.array([[0.6],
                           [0.1]])
bias_layer2 = np.array([0.3])

# 첫 번째 층 출력
layer1_output = sigmoid(np.dot(input_data,
weights_layer1) + bias_layer1)

# 두 번째 층 출력
layer2_output = sigmoid(np.dot(layer1_output,
weights_layer2) + bias_layer2)

print("신경망 최종 출력:\n", layer2_output)
```

--

넘파이 (Numpy) 라이브러리

- **넘파이 주요 기능과 인공지능에서의 활용**

6. 손실 함수 계산

   - AI 모델은 학습 중
     예측 값과 실제 값의 차이를
     손실 함수로 계산
     → 평균 제곱 오차(MSE)

```
# 예측값
predictions = np.array([0.9, 0.4, 0.3])

# 실제값
true_values = np.array([1, 0, 0])

# MSE 계산
mse = np.mean((predictions - true_values) ** 2)
print("평균 제곱 오차:", mse)
```

```
--
```

- **넘파이 주요 기능과 인공지능에서의 활용**

7. 배열 변환
   - AI 프로그래밍에서는
     데이터를 재구성하거나 형식을 변환

```
# 1D 배열 -> 2D 배열로 변환
reshaped_data = data.reshape(-1, 2)
print("리쉐이프된 데이터:\n", reshaped_data)
```

```
[0.1234 0.5678 0.9101]
```

first coding

판다스 (Pandas)

판다스 (Pandas) 라이브러리

판다스 (Pandas) 라이브러리의 주요기능 과 문법

인공지능 모델 개발에 필요한 기능

판다스 (Pandas) 라이브러리

- **판다스 (Pandas)**

  - 판다스는 데이터 분석과 조작에 특화된 라이브러리
  - **구조화된 데이터를 처리하는 데 유용**
    - Series와 DataFrame이라는 두 가지 데이터 구조를 중심으로 작동
  - **구조화된 데이터 처리**
    - 행(row)과 열(column) 기반의 데이터 프레임 구조를 사용
  - **다양한 데이터 소스 지원**
    - CSV, Excel, SQL, JSON 등 다양한 형식의 데이터를 읽고 쓰기
  - **편리한 데이터 조작**
    - 필터링, 그룹화, 집계, 결측 값 처리 등을 간단히 수행

판다스 (Pandas) 라이브러리

- 판다스 설치

pip install pandas

예제

```
import pandas as pd

# 데이터프레임 생성
data = {'Name': ['Alice', 'Bob', 'Charlie’],
        'Age': [25, 30, 35]}
df = pd.DataFrame(data)

# 데이터 조회
print(df.head())  # 처음 5개 행 출력

# 필터링
filtered = df[df['Age'] > 25]
print(filtered)

# 통계 정보
print(df['Age'].mean())  # 평균값 계산
```

판다스 (Pandas) 라이브러리

- **판다스의 핵심 기본적인 문법**

1. 데이터 구조
   - Series
     - 1차원 데이터 구조
     - 값(value)과 인덱스(index)

```
import pandas as pd

# Series 생성
s = pd.Series([10, 20, 30], index=['A', 'B', 'C'])
print(s)
```

```
A    10
B    20
C    30
dtype: int64
```

```
# 값과 인덱스 접근
print(s['A'])
print(s.index)
print(s.values)
```

```
10
Index(['A', 'B', 'C'], dtype='object’)
[10 20 30]
```

판다스 (Pandas) 라이브러리

- **판다스의 핵심 기본적인 문법**

1. 데이터 구조
   - DataFrame
     - 2차원 데이터 구조
       행(row)과 열(column)로 구성된
       테이블 형식의 데이터
     - 다양한 형태의 데이터를 저장할 수 있음

```
# DataFrame 생성
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'Score': [90, 85, 95]
}
df = pd.DataFrame(data)
print(df)
```

```
      Name  Age  Score
0    Alice   25     90
1      Bob   30     85
2  Charlie   35     95
```

- **판다스의 핵심 기본적인 문법**

2. 데이터 확인
    - 데이터의 기본 정보를 빠르게 확인

```
# 상위 몇 개의 행 확인
print(df.head())  # 기본적으로 상위 5개 행 출력
print(df.tail(2))  # 하위 2개 행 출력

# 데이터의 크기 확인 (행, 열 개수)
print(df.shape)  # (3, 3)

# 데이터 타입 및 결측 값 확인
print(df.info())

# 수치형 데이터의 요약 통계
print(df.describe())
```

```
====
```

판다스 (Pandas) 라이브러리

- **판다스의 핵심 기본적인 문법**

3. 열 선택
    - 특정 열(컬럼)을 선택

```
# 단일 열 선택
print(df['Name'])

# 여러 열 선택
print(df[['Name', 'Score']])
```

```
====
```

- **판다스의 핵심 기본적인 문법**

3. 열 선택
   - 특정 행을 선택할 때는 loc와 iloc를 사용
     - loc: 라벨(이름) 기반 인덱싱
     - iloc: 정수 기반 인덱싱

```
# loc: 이름 기반
print(df.loc[0])  # 첫 번째 행 출력
print(df.loc[:, 'Name'])  # 모든 행에서 Name 열만 출력

# iloc: 정수 기반
print(df.iloc[1])  # 두 번째 행 출력
print(df.iloc[:, 1])  # 모든 행에서 두 번째 열 출력
```

```
====
```

- **판다스의 핵심 기본적인 문법**

3. 열 선택
   - 조건 필터링
     - 특정 조건을 만족하는 행만 선택

```
# 조건을 만족하는 데이터 선택 (Score가 90 이상)
high_scores = df[df['Score'] >= 90]
print(high_scores)
```

```
      Name  Age  Score
0    Alice   25     90
2  Charlie   35     95
```

- **판다스의 핵심 기본적인 문법**

4. 데이터 조작
    - 열 추가
        - 새로운 열을 추가

```
# Passed 열 추가
df['Passed'] = df['Score'] >= 90
print(df)
```

```
      Name  Age  Score  Passed
0    Alice   25     90    True
1      Bob   30     85   False
2  Charlie   35     95    True
```

- **판다스의 핵심 기본적인 문법**

4. 데이터 조작
   - 행 추가
     - 새로운 행을 추가하려면 append를 사용

```
new_row = {'Name': 'David’,
           'Age': 28,
           'Score': 88,
           'Passed': False}
df = df.append(new_row, ignore_index=True)
print(df)
```

```
--
```

- **판다스의 핵심 기본적인 문법**

4. 데이터 조작
    - 데이터 수정
        - 특정 값이나 열을 수정

```
# 특정 값 수정
df.loc[0, 'Score'] = 95

# 열 전체 수정
df['Passed'] = df['Score'] >= 90
print(df)
```

```
--
```

- **판다스의 핵심 기본적인 문법**

5. 결측 값 처리
   - 결측 값(NaN)을 처리하는 방법

```
# 결측 값 생성
df.loc[1, 'Score'] = None

# 결측 값 확인
print(df.isnull())  # True는 결측 값을 의미

# 결측 값 채우기
df['Score'] = df['Score'].fillna(0)

# 결측 값 제거
df = df.dropna()
```

--

- **판다스의 핵심 기본적인 문법**

6. 데이터 정렬
    - 데이터를 특정 열을 기준으로 정렬

```
# 오름차순 정렬
sorted_df = df.sort_values(by='Score')
print(sorted_df)

# 내림차순 정렬
sorted_df = df.sort_values(by='Score',
ascending=False)
print(sorted_df)
```

```
--
```

- **판다스의 핵심 기본적인 문법**

7. 데이터 그룹화
    - 데이터를 그룹화하고, 집계 작업

```
# 데이터 생성
data = {
    'Team': ['A', 'A', 'B', 'B'],
    'Score': [85, 90, 78, 88]
}
df = pd.DataFrame(data)

# 팀별 평균 점수 계산
grouped = df.groupby('Team')['Score'].mean()
print(grouped)
```

```
Team
A    87.5
B    83.0
```

- **판다스의 핵심 기본적인 문법**

8. 데이터 입출력
    - 데이터 읽기
        - CSV 파일 등 다양한 파일 읽기

```
# 데이터 생성
data = {
    'Team': ['A', 'A', 'B', 'B'],
    'Score': [85, 90, 78, 88]
}
df = pd.DataFrame(data)

# 팀별 평균 점수 계산
grouped = df.groupby('Team')['Score'].mean()
print(grouped)
```

```
Team
A    87.5
B    83.0
```

- **판다스의 핵심 기본적인 문법**

8. 데이터 입출력
    - 데이터 저장
        - 가공한 데이터를 파일로 저장

```
# CSV 파일로 저장
df.to_csv('output.csv', index=False)
```

```
--
```

판다스 (Pandas) 라이브러리

- **인공지능 프로그래밍에 판다스 (Pandas) 활용**

  - 핵심 개념
    - 데이터 구조
      - Series: 1차원 데이터 구조 (인덱스 + 값)
      - DataFrame: 2차원 데이터 구조 (행 + 열)
    - 데이터 조작 기능
      - 데이터 필터링, 그룹화, 결합, 결측 값 처리, 정렬 등
    - 데이터 입출력
      - CSV, Excel, SQL, JSON 등 다양한 파일 형식을 지원

판다스 (Pandas) 라이브러리

- **인공지능 프로그래밍에 판다스 (Pandas) 활용**

  - 인공지능에서 판다스의 활용
    - 데이터 탐색
      - 데이터를 요약하고 확인
    - 데이터 전 처리
      - 결측 값 처리, 데이터 필터링, 변환
    - 특징 엔지니어링
      - 새로운 열 추가, 기존 데이터를 변형
    - 훈련 데이터 준비
      - 데이터를 AI 모델에 맞는 형태로 가공