

1.1 파이썬의 설치와 설정

- 파이썬의 종류
 - CPython : Python 그 자체
 - Cython : C언어 비슷한 문법으로 함수를 작성하고, CPython Library를 만들어주는 패키지
 - Jython : Python을 구현할 때 "JVM"을 사용한 Python
 - IronPython : Python을 구현할 때 ".NET"을 사용한 Python
 - pypy : Python Compiler를 Python으로 작성한 Python 구현체
- 파이썬의 설치
 - 아나콘다 배포판
 - 윈도우 설치 path: C:/Users/사용자계정이름/anaconda3
 - 주피터 노트북
 - 웹 브라우저를 사용하여 문서와 코드를 동시에 지원하는 개발 도구

1.2 파이썬의 사용

- 코드 실행 방식에 따른 분류

1. 주피터 노트북,REPL(Read-Eval-Print Loop)
2. 스크립트(script) 방식

- 파이썬 패키지 설치

패키지(package), 미리 만들어진 프로그램의 집합. 라이브러리(library)와 동일

- 아나콘다 패키지 서버(<https://anaconda.org/>)

Conda 패키지관리자

Conda list

Conda install 패키지이름

```
$ conda list
# packages in environment at /home/dockeruser/anaconda3:
#
# Name                               Version           Build    Channel
_ipyw_jlab_nb_ext_conf              0.1.0             py37_0
_py-xgboost-mutex                    2.0               cpu_0    conda-forg
absl-py                              0.7.1             pypi_0    pypi
alabaster                            0.7.12            py37_0
anaconda                             custom            py37_0
... (이하 생략)
```

- 파이썬 패키지는 PyPI 서버(<https://pypi.org/>)

pip 패키지관리자

pip list

Pip install 패키지이름

```
$ pip list
Package                               Version
-----
absl-py                               0.7.1
alabaster                             0.7.12
anaconda-clean                        1.0
anaconda-client                      1.7.2
anaconda-navigator                   1.9.7
anaconda-project                     0.8.2
... (이하 생략)
```

1.3 파이썬 패키지 소개

- 데이터 분석용 패키지

- **NumPy** (넘파이)

- 파이썬에서 수치 해석, 특히 선형 대수(linear algebra) 계산 기능을 제공
 - 수치해석 라이브러리

- **Pandas** (판다스)

- 테이블 형태의 데이터를 다루는 데이터프레임(Data Frame) 자료형을 제공
 - 자료의 탐색이나 정리에 아주 유용한 데이터 분석 필수 라이브러리.

- **Matplotlib** (맷플롯리브)

- Matplotlib패키지는 파이썬에서 각종 그래프나 차트 등을 그리는 시각화 기능을 제공.
 - 시각화 라이브러리, MATLAB 플롯 기능 구현

1.4 파이썬의 기초문법

- 연산기호(기본수식)

| | | |
|----|--------|-------|
| + | 3 + 2 | 덧셈 |
| - | 3 - 2 | 뺄셈 |
| * | 3 * 2 | 곱셈 |
| // | 3 // 2 | 실수반환 |
| / | 3 / 2 | 정수반환 |
| % | 3 % 2 | 나머지연산 |
| ** | 3 ** 2 | 거듭제곱 |

- 연산기호(부등식)

> : 키보드 

< : 키보드 

= : 키보드 

≠ : 키보드 

≥ : 키보드 

≤ : 키보드 

- 변수의 사용

- 변수의 이름은 알파벳으로 시작하며 뒤에는 숫자가 올 수 있다.
- 파이썬에서는 변수 이름의 **대문자와 소문자를 구분**

1.5 문자열 연산

- 문자열의 덧셈과 곱셈연산
 - `print("내 이름은 " + "홍길동" + "입니다.")`
 - `print("*"*10)`
- 숫자를 문자열로 바꾸기
 - `str` 명령
- 한 줄 띄우기
 - `\n`
- 이어서 출력하기
 - `end=""`
- 문자열 변수
- 여러 줄의 문자열 출력
 - `"""` 혹은 `'''`
- 문자열 메소드
 - `replace`: 문자열 치환
 - `upper, lower`: 대소문자 변환
 - `count` : 문자열의 갯수
 - `Find, index`: 특정 문자열 위치
 - `Split`: 공백으로 분리, list 형태로 반환
- 문자열 슬라이싱
 - `slice(first, last, step)` :
first 인덱스 부터(포함), last 인덱스 전까지(미포함).
step 만큼 인덱스 증가
 - `slice(count)` :
0번 인덱스 부터 count 개수만큼 슬라이싱
 - 단축 표기법 : `[:]` 또는 `:::` 사용.

1.6 문자열 형식화

- %기호 사용

| 형식지정 문자열 | 의미 |
|----------|----------|
| %s | 문자열 |
| %d | 정수 |
| %f | 부동소수점 실수 |

| 고급 형식지정 문자열 | 의미 |
|-------------|-------------------------------|
| %20s | 전체 20칸을 차지하는 문자열(공백을 앞에 붙인다.) |
| %-10d | 전체 10칸을 차지하는 숫자(공백을 뒤에 붙인다.) |
| %.5f | 부동소수점의 소수점 아래 5자리까지 표시 |

- Format 메서드 사용

| 고급 형식지정 문자열 | 의미 |
|-------------|--|
| {:>10} | 전체 10칸을 차지하며 공백을 앞에 붙임 (문자열을 오른쪽에 붙여서 출력) |
| {:<10} | 전체 10칸을 차지하며 공백을 뒤에 붙임 (문자열을 왼쪽에 붙여서 출력) |
| {:^10} | 전체 10칸을 차지하며 공백을 앞뒤에 붙임 (문자열을 중앙에 붙여서 출력) |
| {:.5f} | 부동소수점의 소수점 아래 5자리까지 표시 |
| {:,} | 천단위 쉼표 표시 |

1.7 조건문과 반복문 기초

- 조건문

- If~else

```
a = 1
if a % 2 == 0:
    print("짝수")
else:
    print("홀수")
```

- If~elif~else

```
c = 6

if c >= 8:
    print("A")
elif c >= 5:
    print("B")
else:
    print("C")
```

- 반복문

- for 카운터 변수 in range(start):
- for 카운터 변수 in range(start,end):
- for 카운터 변수 in range(start,end,step):

- 중첩 for 반복문

```
for j in range(10):
    sum = 0
    for i in range(j + 1):
        sum = sum + (i + 1)
    print(sum)
```

1.8 여러 개의 자료를 한 변수에 담기

- 리스트(list) 자료형

리스트변수 = [자료1, 자료2, 자료3]

Ex) $x = [88, 90, 100]$

인덱싱

리스트변수[인덱스]

Ex) $x[0] \Rightarrow 88, x[1] \Rightarrow 90, (x[0]+x[1]) / 2 \Rightarrow 89.0$

- 딕셔너리 자료형

타입 이름 'dict' 사용

Ex) $b = \text{dict}(\text{math}=88, \text{english}=90, \text{history}=100)$

딕셔너리변수 = {자료이름1: 자료값1, 자료이름2: 자료값2,.....}

Ex) $b = \{\text{"math"}: 88, \text{"english"}: 90, \text{"history"}: 100\}$

$b[\text{"math"}] \Rightarrow 88, b[\text{"english"}] \Rightarrow 90$

- 리스트와 딕셔너리 혼합 자료형

➤ 리스트변수 내 정수+실수+문자열 혼합

➤ 리스트변수 내 리스트 변수 혼합

➤ 딕셔너리 변수내 리스트, 딕셔너리 혼합

1.9 리스트 자료형 다루기

- 리스트 생성

- `b=list(range(10)) => [0,1,2,3,4,5,6,7,8,9]`
- `c=list(range(1,10))=> [1,2,3,4,5,6,7,8,9]`
- `d = list(range(1, 10, 3)) => [1,4,7]`

- 리스트 주요 메소드

- `len()` : 리스트 길이 리턴
- `append` : 리스트의 맨 뒤에 요소 추가
- `insert` : 리스트의 특정 인덱스에 요소
- `del` : 리스트 요소삭제
- `remove`: 첫번째 특정 값을 삭제
- `pop`:리스트의 맨 마지막 요소 리턴 및 삭제
- `reverse`: 리스트 요소의 순서 변경
- `sort`:리스트의 정렬
- ※ `sorted(리스트변수)`

- 슬라이싱

- `List[start:end]`
- `List[start:]`
- `List[:]`

※ start인덱스 포함, end인덱스 불포함

- 역 인덱싱

- 음수 인덱스
- -1은 가장 끝의 원소

1.10. 딕셔너리 자료형 다루기

- 딕셔너리 요소에 접근
 - []연산자: 변수[key]
 - get메소드: get(key)
- 새로운 key/value 생성
 - 변수[key]=value
- Key/value 삭제
 - Del 변수[key]
- 딕셔너리 자료형의 반복
 - : 딕셔너리 자료형은 내부적으로 자료의 순서를 보장하지 않는다
 - 키만 반복하는 경우

```
for k in x:  
    print(k)
```
 - 값만 반복하는 경우

```
for v in x.values():  
    print(v)
```
 - 키와 값 쌍을 반복하는 경우

```
for k, v in x.items():  
    print("key [%s] => value [%d]"  
          % (k, v))
```

1.11 파이썬 패키지 사용하기

- 패키지 импорт

- import 패키지이름
- import 패키지이름 as 패키지별명
Ex) import sklearn as 나

- 패키지 내용 살펴보기

- dir(패키지이름 또는 패키지별명)

- 패키지에 포함된 함수 사용하기

- Ex) import numpy as np
np.arange(10)

- 선택적 импорт

- from 패키지이름 import 명령어
Ex) from numpy import arange
arange(10)

2.1 NumPy 배열 프로그래밍

- NumPy 배열과 리스트의 차이점.
- NumPy 배열의 생성 및 사용.
- NumPy 배열의 연산
- 난수를 발생을 통한 그 결과의 분석하는 방법

❖ NumPy 배열

- NumPy 패키지 импорт
 - `import numpy as np`
 - 자료형: `numpy.ndarray`
- 벡터화 연산(vectorized operation)
 - 배열이나 행렬과 같은 다차원 데이터 구조에 대해 요소 별 연산을 한번에 처리하는 방법
 - 간결하고 효율적인 코드의 작성이 가능

❖ List와 NumPy 배열

리스트의 덧셈, 뺄셈, 곱셈, 나눗셈의 경우

```
a=[1,2,3]
```

```
b=[10,20,30]
```

```
a+b
```

```
[1, 2, 3, 10, 20, 30]
```

```
a*3
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
a-b
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-36-5ae0619f8fe1> in <module>
----> 1 a-b
```

TypeError: unsupported operand type(s) for -: 'list' and 'list'

```
a/b
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-37-aae42d317509> in <module>
----> 1 a/b
```

TypeError: unsupported operand type(s) for /: 'list' and 'list'

NumPy 덧셈, 뺄셈, 곱셈, 나눗셈의 경우

```
1 import numpy as np
2 c=np.array([1,2,3])
3 d=np.array([10,20,30])
```

```
1 c+d
```

```
array([11, 22, 33])
```

```
1 c*3
```

```
array([3, 6, 9])
```

```
1 d/2
```

```
array([ 5., 10., 15.])
```

❖ N차원 배열

- 2차원 배열(행x열)

- `np.array([[0, 1, 2], [3, 4, 5]])` # 2 x 3 array

- 3차원 배열(깊이 x 행x열)

- `np.array([[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]],
[[11, 12, 13, 14], [15, 16, 17, 18], [19, 20, 21, 22]]
])` # 2 x 3 x 4 array

- 배열의 차원과 크기

- `ndim`

- `shape`

❖ NumPy배열 인덱싱

- 인덱싱
 - 일차원 배열 : 리스트와 동일, 변수[인덱스]
 - 다차원 배열 : 콤마(comma ,)를 사용, 변수[행,열]
 - ※ **a[1, 3] 혹은 a[1][3]** 처럼 값을 가져올 수 있습니다. 그러나, 실제 사용에서는 **a[1, 3]**과 같은 인덱싱 방법이 권장됩니다.
- 배열 인덱싱(fancy indexing)
 - 불리언(Boolean) 배열방식
 - 정수 배열방식

❖ NumPy배열 슬라이싱

- 다차원배열의 복수 개에 접근
- 파이썬의 문자열 슬라이싱과 동일
 - `a[0, :]` # 첫번째 행 전체
 - `a[:, 1]` # 두번째 열 전체
 - `a[1, 1:]` # 두번째 행의 두번째 열부터 끝열까지
 - `a[:2, :2]` # 두번째 행 두번째 열까지

#연습문제

```
m = np.array([[ 0,  1,  2,  3,  4], [ 5,  6,  7,  8,  9], [10, 11, 12, 13, 14]])
```

1. 이 행렬에서 값 7 을 인덱싱한다.
2. 이 행렬에서 값 14 을 인덱싱한다.
3. 이 행렬에서 배열 [6, 7] 을 슬라이싱한다.
4. 이 행렬에서 배열 [7, 12] 을 슬라이싱한다.
5. 이 행렬에서 배열 [[3, 4], [8, 9]] 을 슬라이싱한다.

2.2 NumPy배열의 생성과 변형

- NumPy의 자료형

| dtype 접두사 | 설명 | 사용 예 |
|----------------|----------|-------------------------------|
| <code>b</code> | 불리언 | <code>b</code> (참 혹은 거짓) |
| <code>i</code> | 정수 | <code>i8</code> (64비트) |
| <code>u</code> | 부호 없는 정수 | <code>u8</code> (64비트) |
| <code>f</code> | 부동소수점 | <code>f8</code> (64비트) |
| <code>c</code> | 복소 부동소수점 | <code>c16</code> (128비트) |
| <code>O</code> | 객체 | <code>0</code> (객체에 대한 포인터) |
| <code>S</code> | 바이트 문자열 | <code>S24</code> (24 글자) |
| <code>U</code> | 유니코드 문자열 | <code>U24</code> (24 유니코드 글자) |

- Inf와 NaN

- 무한대 `np.inf(infinity)`
- 정의할 수 없는 숫자 `np.nan(not a number)`

- 배열의 생성

- zeros, ones
- zeros_like, ones_like
- empty
- arange(list의 range와 동일)

- 전치연산

- 2차원배열의 행과 열을 바꿈
- 배열의 T속성

- 배열의 크기 변형

- reshape
 - 1차원=>다차원
- flatten, ravel
 - 다차원=>1차원

- 배열의 연결

- hstack
 - 행의 수가 같은 배열, 좌우 연결
- vstack
 - 열의 수가 같은 배열, 위아래 연결
- dstack
 - 깊이방향으로 배열 연결
 - 가장 안쪽의 원소의 차원이 증가
- stack
 - dstack의 기능확장
 - 사용자 지정차원의 배열로 연결, axis인수
 - Axis 디폴트 인수값 0, 가장 앞쪽 차원증가
- tile
 - 동일한 배열을 반복 연결

#연습문제

위의 명령어를 사용하여 다음과 같은 배열을 만들어라.

```
array([[ 0.,  0.,  0.,  1.,  1.],
       [ 0.,  0.,  0.,  1.,  1.],
       [ 0.,  0.,  0.,  1.,  1.],
       [10., 20., 30., 40., 50.],
       [60., 70., 80., 90., 100.],
       [110., 120., 130., 140., 150.],
       [ 0.,  0.,  0.,  1.,  1.],
       [ 0.,  0.,  0.,  1.,  1.],
       [ 0.,  0.,  0.,  1.,  1.],
       [10., 20., 30., 40., 50.],
       [60., 70., 80., 90., 100.],
       [110., 120., 130., 140., 150.]])
```

2.3 NumPy 배열의 연산

- 벡터화 연산

- 두 벡터의 합

```
x = np.arange(1, 10001)
y = np.arange(10001, 20001)
z = x + y
```

- 논리연산

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

- 브로드캐스팅 기능

- 서로 다른 크기를 가진 두 배열의 사칙연산을 지원, 작은 크기의 배열을 자동으로 확장해서 크기가 큰 배열에 맞추는 방법

$$\begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + 1 = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + [0 \ 1 \ 2] = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix}$$

- 차원 축소 연산

- 최대/최소: min, max, argmin(위치), argmax(위치)
- 통계: sum, mean, median(중간값), std(분산), var
- 불리언: all(true), any(true)

- 연산의 대상이 2차원인 경우 axis인수 사용

- Axis=0 :열 연산
- Axis=1: 행 연산

Ex) x = np.array([[1, 1], [2, 2]])
x.sum() =>6

x.sum(axis=0) # 열 합계
=>array([3, 3])
x.sum(axis=1) # 행 합계
=>array([2, 4])

- 정렬

- Sort 대상이 2차원인 경우 axis인수 사용

- Axis=0 :열 기준 정렬
- Axis=1: 행 기준 정렬

- np.sort()[::-1] 내림차순

- np.sort()[::1] 오름차순

- argsort:정렬 순서,인덱스

2.4 난수 발생과 카운팅

- 시드 설정과 난수 생성
 - `np.random.seed(0)`
 - `np.random.rand(5)`
- 데이터 샘플링
 - `choice`
 - **`numpy.random.choice(배열, size=None, replace=True, p=None)`**
 - 배열: 원래의 데이터, 정수이면 `arrange(a)` 명령으로 데이터 생성
 - size: 정수, 샘플숫자
 - replace: 불리언, True이면 한번 선택한 데이터를 다시 선택가능
 - p: 배열, 각 데이터가 선택될 수 있는 확률
- 난수생성
 - `rand`: 0부터 1사이의 균일 분포
 - `randn`: 표준 정규 분포
 - `randint`: 균일 분포의 정수 난수
 - `numpy.random.randint(low, high=None, size=None)`
- 정수 데이터 카운팅
 - `Unique`
 - 중복되지 않는 값의 리스트
 - 중복되지 않는 데이터 개수
 - ※데이터가 없는 경우 0으로 카운트하지 않음
 - `Bincount`
 - 설정 범위내 숫자에 대해 카운트