

A stylized blue brain composed of glowing circuit lines and binary code (0s and 1s) on a black background, symbolizing artificial intelligence or digital cognition.

빅데이터 분석 실습 1: Python basic

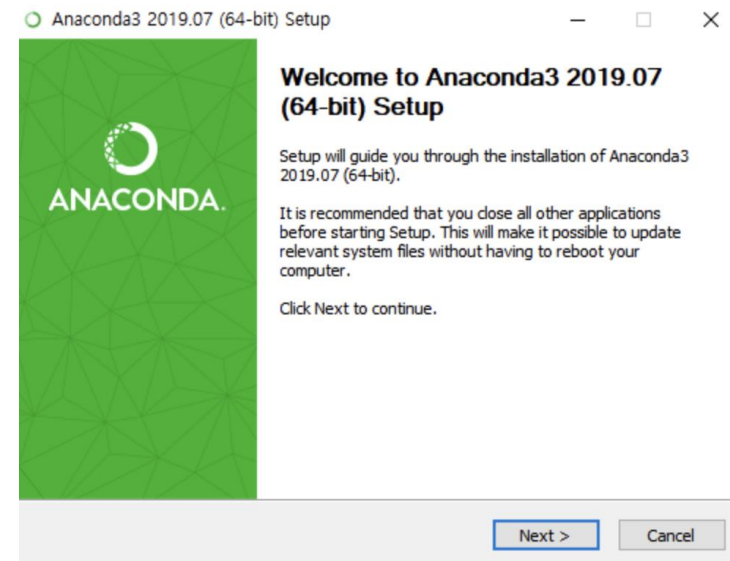
Overview

0. 개발환경 설치 (아나콘다, 코랩)
1. 파이썬 시작, 입출력
2. 자료형의 종류
3. 제어문
4. 함수
5. 클래스
6. 기타 파이썬 관련 정보

0. 개발환경 설치

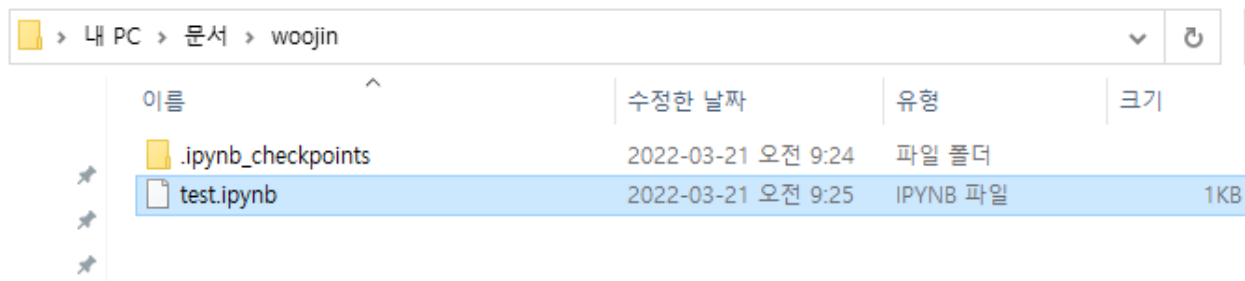
개발환경 설치

- 아나콘다 = 파이썬 + 오픈 라이브러리 + IDE(주피터 노트북)
- 아나콘다 설치: <https://www.anaconda.com/products/individual>
- OS 따라 파일 다운
- 설치 시 옵션은 recommended 체크
- 버전 확인: `conda --version`
- 업데이트: `conda update -n base conda`
- 주피터 노트북
 - 아나콘다 프롬프트에서 jupyter notebook 입력
 - 주피터 노트북은 프롬프트 창이 열려 있어야 실행됨



개발환경 설치

- 작업 폴더 만들기
 - 폴더 만들기: mkdir [폴더 이름]
 - 커맨드(터미널) 창에서 특정 디렉토리로 이동: cd Documents (예시)
 - 상위 디렉토리로 이동: cd ../
- 주피터 노트북에서 해당 폴더 안에 .ipynb 파일 만들어보기

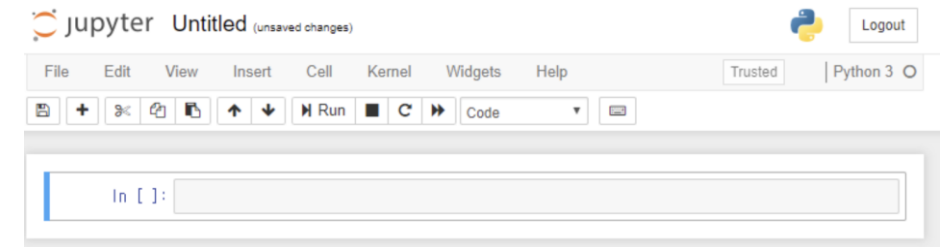


```
Anaconda Prompt (anaconda3)

(base) C:\Users\User>cd documents
(base) C:\Users\User\Documents>mkdir woojin
(base) C:\Users\User\Documents>cd ../
(base) C:\Users\User>cd woojin
지정된 경로를 찾을 수 없습니다.
(base) C:\Users\User>cd documents/woojin
(base) C:\Users\User\Documents\woojin>
```

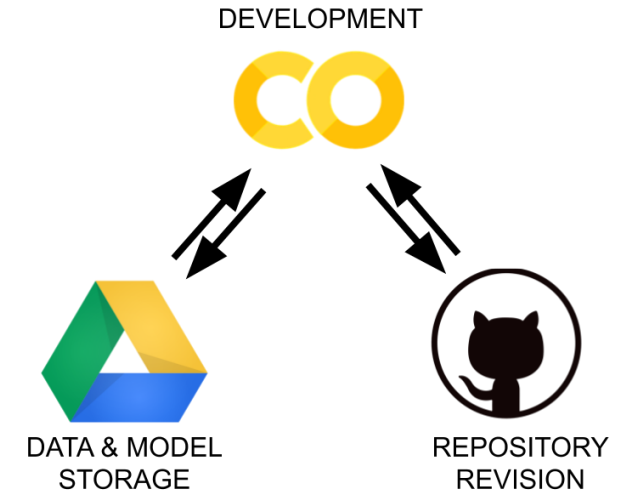
주피터 노트북 작성, 실행방법

- 아나콘다 설치가 정상적으로 이루어졌다면, 주피터 노트북이 실행됨
커맨드(터미널) 창에 jupyter notebook 입력 > <http://localhost:8888/tree>에 접속됨
- 새로운 노트 실행
우측 상단의 New > Python3 클릭하면 새로운 노트 파일 열림
- 셀에 코드 작성 후 실행해보기
ex) `print(3+4)` > 셀 클릭 후 **Run** 버튼 클릭하거나 **Shift+Enter** (**Command+Enter**)로 셀 실행
- 주피터 노트북 팁
 - 여러 셀 동시 선택: Shift 누른 상태로 셀 왼쪽 연속 클릭
 - 마크다운: 셀 클릭 후 Code 대신 Markdown 항목 클릭 (# Head, * 글머리, ## small head, ****Bold**, ...)



Colab

- 딥러닝 실습 환경을 구축하기 위해서는 로컬 PC의 한계가 있음
- 구글에서는 인터넷만 된다면 바로 딥러닝 모델을 구현할 수 있는 Colab 개발 환경을 지원 (<https://colab.research.google.com/>)
- 구글 계정 로그인 후 좌측 상단의 파일 > 새 노트
 - Github에서 코드 가져올때: 파일 > 노트 업로드 > GitHub url 입력
- GPU 사용하는 방법: 런타임 > 런타임 유형 변경 > 하드웨어 가속기 > GPU 선택 후 저장
- 코랩에 데이터 업로드: 로컬에 있는 csv 파일을 업로드 or 링크 연결



```
#iris = pd.read_csv('./iris.csv')
from google.colab import files
file_uploaded = files.upload()
iris = pd.read_csv('iris.csv')
iris.head()
```

... 파일 선택 선택된 파일 없음 Cancel upload

1. 파이썬 시작, 입출력

파이썬 기초문법

- 주석: # 뒤 문장은 코드로 인식되지 않음!
- 여러 줄 한 줄로 잇기: 백슬래시(\) 사용
- 치환(=)
 - = 기준으로 왼쪽에는 변수, 오른쪽에는 값을 할당 (a=3)
 - 변수의 자료형은 = 오른쪽의 자료형에 의해 결정됨
 - $X \text{ (연산자)} = y \Leftrightarrow X = X \text{ (연산자)} y$
ex) $x += 1 \Leftrightarrow x = x + 1$

입력과 출력

- input() 함수
 - 값을 입력받을 때 사용
 - 여러개를 한번에 입력받을 때 (split)
 - 정수/실수/... 값을 입력받은 경우에만 실행

```
name = input("이름: ") #문자열 입력
age = int(input("나이: ")) #정수 입력
```

이름: 우진
나이: 25

```
#여러개 한번에 입력받기
str_list = input('문자열을 입력해주세요: ').split()
print(str_list)
```

문자열을 입력해주세요: hello good world
['hello', 'good', 'world']

입력과 출력

■ print() 함수

■ end: 마지막 출력 문자를 변경

■ sep: 항목 간의 출력 문자 설정

■ 서식 출력

■ **format()** 함수: format(값, 형식 지정)

■ %d, %s, %f, ...

■ 자료형 배울 때 자세히!

```
print('오늘은 %s요일. %d월 %d일입니다.' % ('월', 3, 22))
print('오늘은 {0}요일. {1}월 {2}일입니다.'.format('월', 3, 22))
print('오늘은 {day}요일. {month}월 {date}일입니다.'.format(day='월', month=3, date=22))
print('삼삼칠 박수 : {0}!!! {0}!!! {1}!!!'.format('짝'*3, '짝'*7))
print('-' * 40)
```

```
오늘은 월요일. 3월 22일입니다.
오늘은 월요일. 3월 22일입니다.
오늘은 월요일. 3월 22일입니다.
삼삼칠 박수 : 짹짹!!! 짹짹!!! 짹짹짹짹짹!!!
-----
```

2. 자료형의 종류

자료형의 종류

■ 자료형: 컴퓨터로 표현할 수 있는 데이터 종류

class	설명	구분
list	mutable 한 순서가 있는 객체 집합	mutable
set	mutable 한 순서가 없는 고유한 객체 집합	mutable
dict	key와 value가 <u>맵핑된</u> 객체, 순서 없음	mutable
bool	참(True),거짓(False)	immutable
int	정수	immutable
float	실수	immutable
tuple	immutable 한 순서가 있는 객체 집합	immutable
str	문자열	immutable
frozenset	immutable한 set	immutable

수치 자료형

- type 함수: 변수의 자료형을 알려주는 함수
- 숫자: int, float, long, complex
- 수치 자료형을 폭넓게 다루기 위해서는 “math” 모듈 유용
- 기타 math 모듈에 대한 정보
 - <https://docs.python.org/3/library/math.html> 참고

math module

```
from math import exp, log, sqrt
print("{0:.4f}".format(exp(3)))
print("{0:.2f}".format(log(4)))
print("{0:.1f}".format(sqrt(81)))
```

```
20.0855
1.39
9.0
```

참고 - 모듈(module)

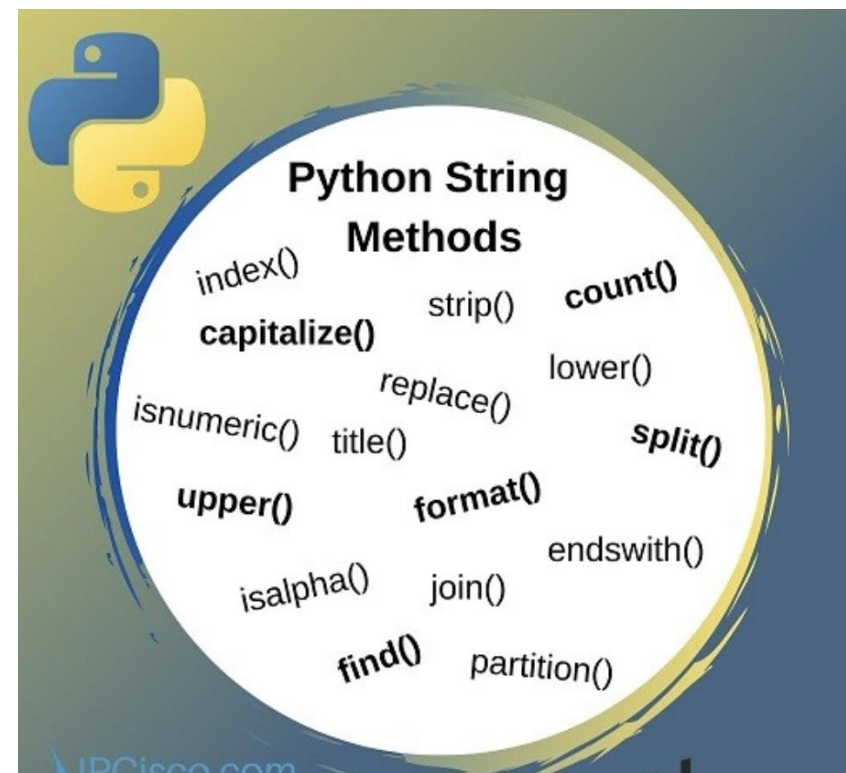
- 모듈 설치: `pip install [모듈명]`
- 주피터 노트북에서 `import`로 설치한 모듈을 불러옴
- 모듈 이름이 길거나 복잡하면 “as” 사용하여 간단히 표현
 - `import matplotlib.pyplot as plt`
- 모듈 전체를 불러오지 않고 필요한 기능만 부르기 위해 “from” 사용
 - `from collections import defaultdict, Counter`

문자열(string)

- 일련의 문자(알파벳, 숫자, 특수기호 등)들이 나열된 것
- " " 또는 ' ' 로 표현
- 긴 문자열의 가독성을 높이기 위해 백슬래시(\)를 사용하여 라인 분리
(라인의 마지막이 \)

문자열의 대표적인 함수들

- **split**: 하나의 문자열을 나누어 리스트로 반환
 - `str.split(분할될 자리, 분할 횟수)`
 - 아무것도 입력되지 않으면 공백 있는 자리에서 분할됨
- **join**: 분할된 문자열 리스트를 하나의 리스트로 출력
 - "연결부위 문자".`join(string)`
- 문자열들을 연결할 때에는 +로 연결
- **replace**: 문자열 내의 문자를 다른 문자로 바꿈
 - `str.replace('변경하고 싶은 문자', '변경 후 문자', 횟수)`
- **lower, upper, capitalize**: 대문자를 소문자로, 소문자를 대문자로 바꿈



리스트 (list)

- 리스트는 대괄호([]) 안에 원소들이 나열돼 있고 쉼표(,)로 구분

- len(): 리스트의 길이 (원소 갯수)

- max(), min(): 리스트 안 최대/최소 원소

- count(): 리스트 내 특정 값의 갯수

- 인덱스(index): 리스트 내 특정 원소 번호

- [0]: 첫번째 원소, [-1]: 마지막 원소, [-2]: 마지막에서 두번째 원소, ...

- 리스트 분할: [x:y]는 인덱스 x ~ 인덱스 y-1까지 원소들을 출력

- 리스트 병합: +, 리스트 반복: *

```
s = 'show how to index into sequences'.split()
print(s)
print(s[0], s[-1], s[-2])
print(s[1:3])
print(s[:-2])
print(s[2:])
```

```
['show', 'how', 'to', 'index', 'into', 'sequences']
show sequences into
['how', 'to']
['show', 'how', 'to', 'index']
['to', 'index', 'into', 'sequences']
```

```
a_list = [1,2,2,3,3,3,4,4,4,4]
a_list + [5]*5
```

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

리스트 관련 함수들

- **in, not in**: 특정 원소의 리스트 내 포함 여부를 반환 (True, False)
- **append, insert, remove, pop**: 원소 추가/제거 관련 함수
- **reverse**: 리스트 순서 뒤집기
- **sort**: 정렬하는 함수
 - 디폴트는 오름차순 정렬, reverse=True 설정하면 내림차순
 - Key 옵션으로 정렬 기준을 정할 수 있음

```
m = '나는 파이썬을 잘하고 싶다'
m = m.split()
print(m)
m.sort(key=len)
print(m)
```

```
['나는', '파이썬을', '잘하고', '싶다']
['나는', '싶다', '잘하고', '파이썬을']
```

```
a = [1, 2, 3]
a.append(4)
print(a)
a.insert(3,5) #index 3에 5 삽입
print(a)
a.remove(2)
print(a)
a.pop() #마지막 원소 제거
print(a)
```

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

```
b = [1,4,2,3]
b.reverse()
print(b)
sorted(b)
b.sort()
print(b)
```

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

튜플(tuple)

- 튜플은 리스트와 유사하지만, **변경하지 못한다**는 차이가 있음 (immutable)
- 대체로 (, ,)로 표현하기는 하지만 필수 조건은 아님
- 리스트와 마찬가지로 튜플의 원소는 자료형 제한 없음
- 원소가 1개일 때에는 뒤에 쉼표(,)를 붙여야 함
- len, min, max, index, in, ... 등 함수 (변경 못하기 때문에 append, remove 등 함수x)
- 튜플은 한번에 여러 변수에 값 할당 가능 (tuple unpacking) → 변수간 값 교환할 때 유용
- 다른 객체를 튜플로 만들기

딕셔너리(dictionary)

- 딕셔너리란?

사전. 사전에는 단어와 그 단어에 대응되는 여러 뜻들이 나열되어 있음
→ key(단어), value(대응되는 값)를 한 쌍으로 갖는 자료형이 딕셔너리!

key	value
name	pey
phone	01199993323
birth	1118

- 예) dic = {'name': 'pey', 'phone': '0119999323', 'birth': '1118'}

- value에는 정수, 문자열, 리스트 등이 올 수 있지만, key에는 **immutable 객체**만 올 수 있다. (Int, tuple, ..)

- Key를 사용해 value 얻기: **get**

- 딕셔너리에서 in 함수는 "key"에만 적용됨

```
grade = {'pey': 10, 'julliet': 99}
print(grade['pey'])
print(grade['julliet'])
print(grade.get('pey'))
print(grade.get('tom'))
```

```
10
99
10
None
```

딕셔너리(dictionary)

```
a = {'alice': [1, 2, 3], 'bob': 20, 'tony': 15, 'suzy': 30}
a['alice'] = 5
print(a)
a.update({'bob':99, 'tony':99, 'kim': 30}) #여러 개 한번에 수정
print(a)
del a['alice']
print(a)
```

```
{'alice': 5, 'bob': 20, 'tony': 15, 'suzy': 30}
{'alice': 5, 'bob': 99, 'tony': 99, 'suzy': 30, 'kim': 30}
{'bob': 99, 'tony': 99, 'suzy': 30, 'kim': 30}
```

```
print(a.values())
print(list(a.keys())) #list로 변환
```

```
dict_values([5, 99, 99, 30, 30])
['alice', 'bob', 'tony', 'suzy', 'kim']
```

```
a.items() #key, value 쌍을 튜플로 묶음
```

```
dict_items([('alice', 5), ('bob', 99), ('tony', 99), ('suzy', 30), ('kim', 30)])
```

```
'alice' in a
```

```
True
```

Quiz break..

- replace 함수로 문자열 "a:b:c:d"를 "a%b%c:d"로 바꾸어 출력해보기
- 핸드폰 번호의 뒷 네자리만 출력해보기

phone = "010-1234-5678"

3. 제어문 (if, for, while)

제어문의 종류

- 파이썬의 제어문에는 if, while, for문이 있다.
- **if-else**: ~이면(if) ~을 실행하고, 그렇지 않으면(else) ~을 실행
- **for**: 인덱스를 활용하여 반복문 실행
- **while**: 특정 조건이 만족하는 한 실행되는 반복문

관계연산자와 if문

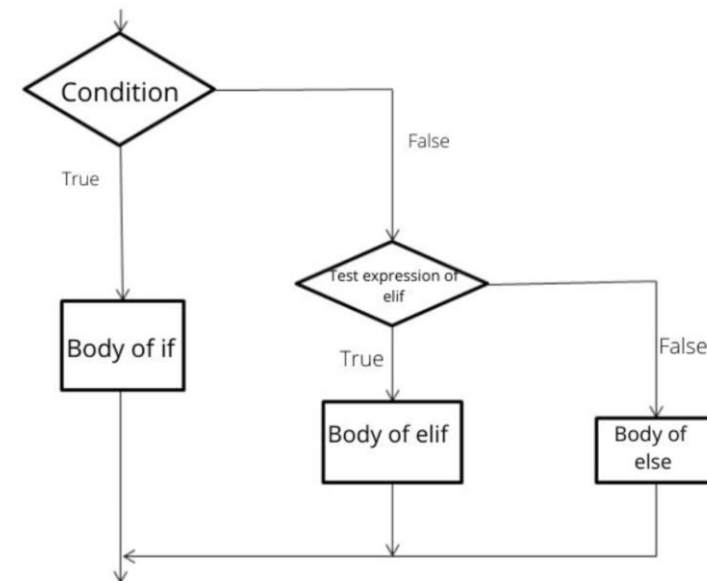
- 관계연산자: ==, !=, <, >, <=, >= 등 → True/False 반환
- if문의 기본 구조, *elif = else if
- 들여쓰기 조심
- If문을 한 줄로 표현하기

```
score=70  
message="success" if score>=60 else "failure"  
print(message)
```

success

```
h=40  
if h>50:  
    print("50보다 크면 실행")  
elif h<20:  
    print("20보다 작으면 실행")  
else:  
    print("20이상 50이하면 실행")
```

20이상 50이하면 실행



for문

■ 기본 구조

리스트나 튜플, 문자열의 첫번째 원소부터 순차적으로 변수(i)에 대입되어 반복적으로 실행되는 구조

■ for문과 함께 자주 사용되는 **range** 함수

range(a, b): a ~ b-1, range(a): 0 ~ a-1

range(a, b, c): a ~ b-1까지 c씩 증가*

■ for문의 중첩사용

end=" " : j가 변하는 동안 한 라인에 출력되도록 해줌

print(' '): i가 변할 때마다 enter 처리

```
add = 0
for i in range(1, 11):
    add = add + i
print(add)
```

55

```
for i in range(2, 10):           # ①번 for문
    for j in range(1, 10):       # ②번 for문
        print(i*j, end=" ")
    print('')
```

```
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

참고: enumerate

- 해당 원소가 몇번째 반복문을 돌고 있는지 알고 싶을 때 유용
- Tuple 형태로 반환: (*index*, *element*)
- Example

```
t = [1, 5, 7, 33, 39, 52]
for i, v in enumerate(t):
    print("index : {}, value: {}".format(i,v))
```

```
index : 0, value: 1
index : 1, value: 5
index : 2, value: 7
index : 3, value: 33
index : 4, value: 39
index : 5, value: 52
```

While문

■ 기본구조

while <조건문> : <수행할 문장>

■ 무한루프와 break

while True: 무한루프

강제로 빠져나오기 위해 **break** 사용 (중단)

■ Continue

반복문을 빠져나오긴 하지만 중단시키지
않고, 다음 반복으로 넘어감

```
while True:
    response = input('숫자를입력하세요:')
    if int(response) % 10 == 0 :
        print('10으로 나눈 나머지가 0입니다.')
        break
```

숫자를입력하세요:18
숫자를입력하세요:29
숫자를입력하세요:30
10으로 나눈 나머지가 0입니다.

```
while True:
    response = input('숫자를 입력하세요:')
    result = int(response) % 10
    if result == 0 :
        continue
    print("10으로 나눈 나머지는 {}입니다.".format(result)) #끝나지 않음
```

숫자를 입력하세요:27
10으로 나눈 나머지는 7입니다.
숫자를 입력하세요:30
숫자를 입력하세요:50
숫자를 입력하세요:29
10으로 나눈 나머지는 9입니다.

숫자를 입력하세요:

Quiz break...

- While문을 사용하여 1000이하 3의 배수의 합 구하기

- for문을 사용하여 A 학급의 점수 평균을 구해보기

A = [70, 60, 55, 75, 95, 90, 80, 80, 85, 100]

- 어떤 주의 요일별 최고기온

요일	월	화	수	목	금	토	일
최고 기온	25.5	28.3	33.2	32.1	17.3	35.3	33.3

1) 이 자료를 딕셔너리 자료형으로 저장하고,

2) 최고기온이 30도 이상인 **요일**을 출력하는 코드 짜보기

Quiz break...

문제

첫째 줄에는 별 1개, 둘째 줄에는 별 3개, ..., N번째 줄에는 별 $2 \times N - 1$ 개를 찍는 문제

별은 가운데를 기준으로 대칭이어야 한다.

입력

첫째 줄에 $N(1 \leq N \leq 100)$ 이 주어진다.

출력

첫째 줄부터 N번째 줄까지 차례대로 별을 출력한다.

예제 입력 1 복사

5

예제 출력 1 복사

```
  *
 ***
*****
*****
*****
```

4. 함수

사용자 정의 함수

- 파이썬의 기본 함수를 이용하거나 다른 사람이 만든 모듈 설치하는 것보다 내가 직접 함수 만드는게 편할 수도 있다!
- 함수를 정의하는 방법
`def 함수명(매개변수):`
.....
`return 결과값`
- 입력값 없는 함수
리턴값 없는 함수
둘다 없는 함수

```
#입력값 없는 함수: 결과값 받을 변수 = 함수이름()  
def say():  
    return 'Hi'  
  
a = say()  
print(a)
```

Hi

```
#결과값 없는 함수 = 특정 동작을 실행  
def add(a,b):  
    print("%d, %d의 합은 %d입니다." % (a,b,a+b))  
  
add(4,5)
```

4, 5의 합은 9입니다.

```
print(add(4,5)) #결과값이 none
```

4, 5의 합은 9입니다.
None

```
#입력, 결과값 둘다 없는 경우  
def say():  
    print('Hi')  
say()
```

Hi

사용자 정의 함수

- 입력값이 몇개인지 모를 때에는 어떻게 할까?

***매개변수 (*args)로 표현**

- 실습해보기

입력받은 숫자들을 모두 더하거나 곱하는 함수 정의하기

- 함수의 리턴 값은 언제나 한 개!

여러 개를 리턴하더라도 하나의 튜플 형태로 반환

리턴을 여러 번 쓴 경우는 첫번째 리턴만 실행됨

```
def add_many(*args):  
    result=0  
    for i in args:  
        result+=i  
    return result  
  
print(add_many(1,2,3))  
print(add_many(1,2,3,4,5,6,7,8))
```

6
36

***실습: 출력 예시**

```
print(add_mul('add', 1,2,3,4,5,6))  
print(add_mul('mul', 1,2,3))  
print(add_mul('div', 4,2))
```

21
6
input error

사용자 정의 함수

- 매개변수에 초깃값 설정하기
- 함수 안에서 선언한 변수의 효력 범위
- **lambda**

함수 정의를 간단하게 하고 싶을 때 (한 줄로) 사용

```
def double(x):  
    """  
    보통은 이곳에 함수에 대한 설명을 주석으로 표현  
    this function multiplies its input by 2.  
    """  
    return x*2  
  
print(double(8))
```

16

```
g = lambda x: x*2  
print(g(8))
```

16

```
def fullname(first="what", last="something"):  
    print(first+" "+last)
```

```
fullname("woojin", "Jeong")  
fullname("woojin")  
fullname("Jeong")  
fullname(last="Jeong")  
fullname()
```

```
woojin Jeong  
woojin something  
Jeong something  
what Jeong  
what something
```

Quiz break..

- N을 인자로 가지고, N번째 피보나치 수를 반환하는 함수 정의하기

피보나치 수열: 0, 1, 1, 2, 3, 5, 8,

예) $f(10) \rightarrow 55$

- N을 인자로 가지고, 1~N까지의 제곱수를 원소로 가지는 리스트를 반환하는 함수 정의하기

예) $sq(23) \rightarrow [1,4,9,16]$

- (날씨 예제) 딕셔너리를 인자로 가지고, 기온이 30도 이상인 **요일 리스트** 반환하는 함수 정의하기

5. 클래스(class)

클래스(class)

- 클래스란?

복잡한 문제를 다루기 쉽게 만들기 위해 사용

객체의 구조와 행동을 정의하며, 초기화를 통해 제어

- 메소드(method): 클래스 내의 함수

파이썬에서 클래스 메소드의 첫번째 파라미터는 관행적으로 **self**를 사용

- 클래스와 객체

Flight: 클래스, f: 객체

```
class Student:
    def __init__(self, name, age): #생성자
        self.name = name
        self.age = age
    def aboutMe(self):
        print("제 이름은 "+self.name + "이고, 제 나이는" +str(self.age)+"살 입니다.")
```

```
woojin = Student("정우진", 25) #student 클래스의 객체 생성
woojin.aboutMe()
```

제 이름은 정우진이고, 제 나이는25살 입니다.

생성자

- 생성자: 객체를 생성할 때 호출되는 함수 (`__init__`)
클래스가 호출되면 객체의 인자들을 초기화시킴
init 앞뒤에 `_` 붙는 의미: python에서 특별히 정해둔 메소드

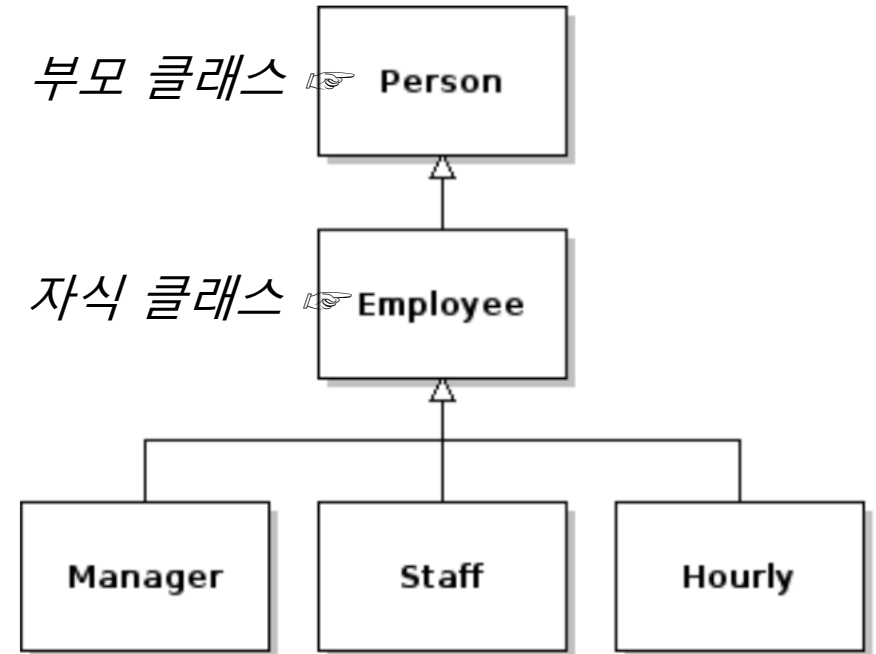
```
class Student:
    def __init__(self, name, age): #생성자
        self.name = name
        self.age = age
    def aboutMe(self):
        print("제 이름은 "+self.name + "이고, 제 나이는" +str(self.age)+"살 입니다.")
```

```
woojin = Student("정우진", 25) #student 클래스의 객체 생성
woojin.aboutMe()
```

제 이름은 정우진이고, 제 나이는25살 입니다.

상속(inheritance)

- 부모 클래스의 내용을 자식 클래스가 물려받는 것
- 부모 클래스에서 정의된 속성을 자식 클래스에서 사용 가능
- **Class 자식클래스 (부모클래스): ...**
- 다중 상속도 가능 (부모 클래스가 여러개)



클래스 상속 예시

```
class Person:
    def __init__(self, name, age, gender):
        self.Name = name
        self.Age = age
        self.Gender = gender
    def aboutMe(self):
        print("저의 이름은 " + self.Name + "이구요, 제 나이는 " + self.Age + "살 입니다.")

class Employee(Person): #직원 클래스는 사람 클래스를 상속받음
    def __init__(self, name, age, gender, salary, hiredate):
        super().__init__(name, age, gender) #부모 클래스의 생성자 호출
        self.Salary = salary
        self.Hiredate = hiredate
    def doWork(self):
        print("열심히 일을 합니다.")
    def aboutMe(self): #부모 클래스의 메소드와 이름은 같지만 다른 함수 구현 가능
        Person.aboutMe(self) #self를 인자로 넣어주어야함
        print("제 급여는 " + self.Salary + "원 이구요, 제 입사일은 " + self.Hiredate + " 입니다.")
```

```
woojin = Employee("정우진", "25", "female", "10억", "3월 1일")
woojin.aboutMe()
```

저의 이름은 정우진이구요, 제 나이는 25살 입니다.
제 급여는 10억원 이구요, 제 입사일은 3월 1일 입니다.

6. 기타 파이썬 관련 정보

pip (python package index)

- pip는 파이썬의 각종 라이브러리를 설치 및 관리해주는 패키지 매니저
- pip 업그레이드: `pip install --upgrade pip`
- pip 패키지 검색: `pip search [검색 패키지명]`
- pip 설치 리스트 확인: **`pip list`**
- 패키지 설치/삭제: **`pip install [패키지명]`** / **`pip uninstall [패키지명]`**
특정 버전 이상의 패키지 설치: **`pip install [패키지명]>=버전넘버(예: 2.3.0)`**
- 파이썬 프로젝트에서는 일반적으로 필요한 라이브러리 리스트를 requirements.txt 파일로 만들어둠.
`pip install -r requirements.txt`

numpy & pandas

- 데이터 분석 시 사용하는 대표적인 라이브러리
 - **Numpy**: 수치 데이터(벡터, 행렬 등) 다룰 때 유용
 - **Pandas**: "dataframe", "series" 등의 데이터 구조 지원
dataframe = index(행), column(열)로 이루어진 2차원 데이터
 - **Matplotlib**: 차트, 플롯으로 시각화하는 패키지
- 넘파이, 판다스, matplotlib 설치 확인하기 (아나콘다 설치하면 포함돼 있음)
pip list

with문, open

- txt 파일 읽고 쓸 때 자주 사용됨
- 파일은 open() 함수로 열고, close() 함수로 닫아야 한다.
`f = open('test.txt', mode = 'rt', encoding='utf-8')`
- with 블록 사용하면 close()를 호출하지 않아도 파일을 닫는 효과 있음
`with open('test.txt', mode='wt', encoding='utf-8') as f:`
 `f.write('파이썬으로 파일을 작성하고 있습니다.')`
 `f.write('...')`
`dosomething()...`

가상환경 (venv, conda)

■ 가상환경이란?

파이썬에서는 라이브러리 당 **하나의 버전**만 설치 가능

여러 프로젝트를 진행하는 경우에는 작업을 바꿀 때마다 다른 버전의 라이브러리를 설치해야하기 때문에 **충돌 위험** 있음 → 독립적인 작업환경을 위해 가상환경 필요

■ 가상환경의 대표적인 모듈

■ venv: 파이썬 3.3버전 이후로 기본 모듈에 포함됨

■ conda: 아나콘다 설치했을 때 사용

가상환경 생성: `conda create -n [가상환경 이름]`

활성화/비활성화: `source activate [가상환경 이름]` / `source deactivate`

가상환경 제거: `conda env remove -n [가상환경 이름]`

```
(base) C:\Users\User>conda activate bigdata
(bigdata) C:\Users\User>
```

Reference

- Data Science from Scratch
- Foundations for Analytics with Python, Clinton Brownley (파이썬 데이터 분석 입문, 한빛 미디어)
- 점프 투 파이썬