파이썬 기초

보건빅데이터통계분석

이새봄
삼육대학교 SW융합교육원

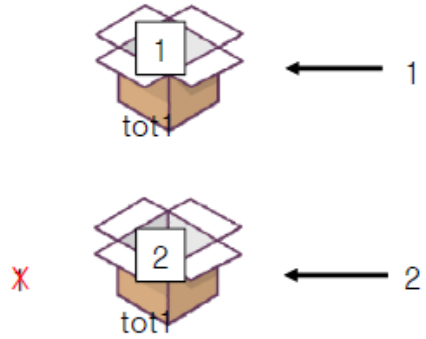
파이썬 기초

프로그래밍의 기본 원칙

- 변수선언
- 데이터형
  - 숫자, 문자, 자료
  - 자료구조
- 연산자
  - 산술연산자, 증가감소연산자, 대입연산자
  - 비교연산자, 논리연산자, 연결연산자
- 제어문
  - If
- 반복문
  - for
  - while
- 함수

변수명

- 변수선언
  - 변수명 = 값(문자 또는 숫자)



- 변수명 규칙
  - # 제외
  - 첫 문자를 영문자로 시작해서 영문 대문자(A~Z)와 소문자(a~z), 숫자(0~9),
  - 밑줄(_)을 사용하여 작성
    - 언더 바를 제외한 특수 기호는 쓰지 못함
  - 첫문자를 숫자로 표기할 수 없음: ex) 1tot
  - 변수의 값이 무엇을 나타내는지 쉽게 표기
  - 특수기호(!, @,# …)는 사용할 수 없음
  - 미리 정해진 예약어(var, if, for…)는 사용할 수 없음
  - 대/소문자를 구분
    - 즉, ‘name’과 ‘nAme’은 다른 변수

데이터형

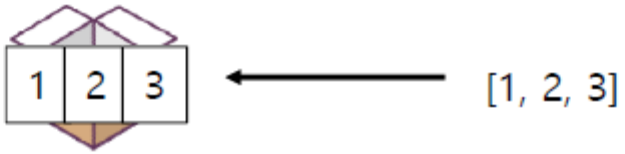
- 기본 자료형
  - 정수형, 실수형, 부울형, 문자형
- 집합형 자료형
  - 리스트형, 튜플형, 사전형
- type() : 자료형 확인

종류	종류	문법 예
정수형	int	42, 0, -41
실수형	float	3.141
불울형	bool	True False
문자형	str	'홍길동' "홍길동"

데이터형

- 데이터 집합

	형식	데이터 수정, 삭제	index 이용
리스트(list)	num = [1, 2, 3]	O	O
튜플(tuple)	num = (1, 2, 3)	X	O
세트(set)	num = {1, 2, 3}	O	X
딕셔너리(dictionary)	num = {a: 1, 2, 3}	O	X



Index	num[0]	num[1]	num[2]	num[3]	num[4]
data	1	2	3	4	5

산술 및 관계 연산자

- 산술연산자

산술연산자	설명	예제	사례
+	더하기 연산	A + B	5 + 10 = 15
-	빼기 연산	A – B	5 - 10 = -5
*	곱하기 연산	A * B	5 * 10 = 50
/	나누기 연산	A / B	10 / 5 = 2
**	승수	A ** B	5 ** 2 = 25
%	나머지	A % B	13 % 5 = 3

산술 및 관계 연산자

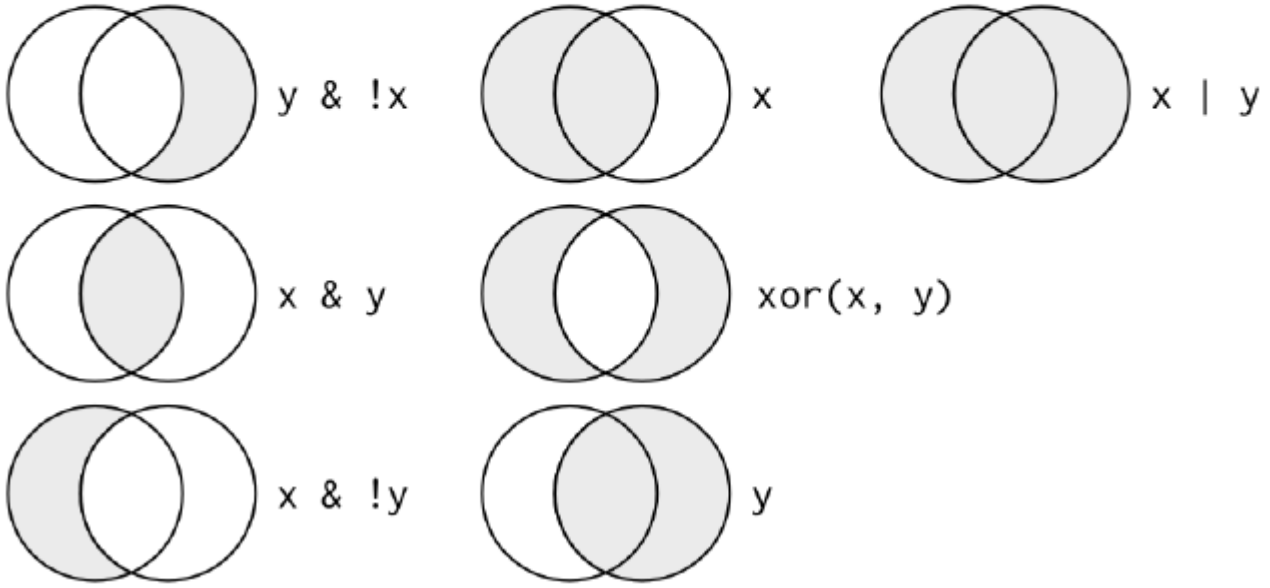
- 관계연산자

관계 연산자	설명	예제	사례
==	A, B가 같다.	변수 == B	Name == "홍길동"
!=	A, B가 같지 않다.	변수 != B	Name != "홍길동"
<	A가 B보다 작다.	변수 < B	Weight < 40
<=	A가 B보다 작거나 같다.	변수 <= B	Weight <= 40
>	A가 B보다 크다.	변수 > B	Weight > 40
>=	A가 B보다 크거나 같다.	변수 >= B	Weight >= 40

산술 및 관계 연산자

- 논리연산자

관계 연산자	설명	예제
&	A와 B가 모두 참이면 참(A가 거짓이여도 B 검사)	A & B
|	A와 B가 중 하나만 참이면 참(A가 참이어도 B 검사)	A | B
!	부정	!A



데이터 포멧팅

- 문자열 포멧코드
  - 문자열을 원하는 형식으로 출력하거나 데이터를 특정한 형식으로 변환할 때 사용

코드	설명
%s	문자열(String)
%d	정수(Integer)
%f	부동소수(floating-point) Ex)0.3f 소수점 3자리까지 표기

```
name = "Alice"
print("이름: %s" % name)
```

```
이름: Alice
```

변수선언 및 데이터 형

01.Python 기초 사용법

1. 변수선언 및 데이터 형

```
[1]: tot1 = 5
     tot2 = 3.14
     name = '홍길동'
```

```
[2]: print(tot1)
     name
```

```
5
[2]: '홍길동'
```

```
[3]: type(tot1)
```

```
[3]: int
```

```
[4]: type(tot2)
```

```
[4]: float
```

```
[5]: type(name)
```

```
[5]: str
```

```
[6]: # 여러변수를 동시에 입력
```

변수선언 및 데이터 형

```
[6]: # 여러변수들 동시에 입력
     start, end, step = 1, 10, 2

[7]: print(start, end, step)

     1 10 2
```

2.데이터구조

2.1 리스트

- 순서를 가지는 객체 집합

```
[8]: num = [1, 2, 3, 4]
     num

[8]: [1, 2, 3, 4]

[9]: # 인덱싱(0부터 시작)
     num[0]

[9]: 1

[10]: # 마지막은 제외
      num[1:3]

[10]: [2, 3]

[11]: # 데이터 추가, 제거
```

데이터 구조

2.데이터구조

2.1 리스트

- 순서를 가지는 객체 집합

[8]:
```
num = [1, 2, 3, 4]
num
```

[8]:
```
[1, 2, 3, 4]
```

[9]:
```
# 인덱싱(0부터 시작)
num[0]
```

[9]:
```
1
```

[10]:
```
# 마지막은 제외
num[1:3]
```

[10]:
```
[2, 3]
```

[11]:
```
# 데이터 추가, 제거
num.append(5)
num
```

[11]:
```
[1, 2, 3, 4, 5]
```

[12]:
```
num.remove(5)
num
```

데이터 구조

```
[11]: [1, 2, 3, 4, 5]

[12]: num.remove(5)
      num

[12]: [1, 2, 3, 4]

[13]: min(num)

[13]: 1

[14]: max(num)

[14]: 4
```

2.2 튜플(tuple)

- 자료 추가 X, 속도 빠름

```
[15]: num = (1, 2, 3, 4)
      num

[15]: (1, 2, 3, 4)

[16]: num[0]

[16]: 1

[17]: min(num)
```

데이터 구조

```
[17]: min(num)

[17]: 1

[18]: # 자료추가 안됨
      num.append(5)

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[18], line 2
      1 # 자료추가 안됨
----> 2 num.append(5)

AttributeError: 'tuple' object has no attribute 'append'
```

2.3 집합(set)

- 중복 X, index X, 자료추가 O)

```
[19]: # 중복 안됨
      num = {1, 2, 2, 3, 4, 4}
      num

[19]: {1, 2, 3, 4}

[20]: num.add(5)
      num

[20]: {1, 2, 3, 4, 5}
```

데이터 구조

```
[20]: {1, 2, 3, 4, 5}
```

```
[21]: num.add(4)
      num
```

```
[21]: {1, 2, 3, 4, 5}
```

```
[22]: num.remove(5)
      num
```

```
[22]: {1, 2, 3, 4}
```

```
[23]: num[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[23], line 1
----> 1 num[0]

TypeError: 'set' object is not subscriptable
```

2.4 사전(dictionary)

- key, value 형태로 저장 - index(X)

```
[24]: student_dc = {1:"김길동", 10:"박길동"}
      student_dc
```

```
[24]: {1: '김길동', 10: '박길동'}
```

데이터 구조

2.4 사전(dictionary)

- key, value 형태로 저장 - index(X)

[24]:
```
student_dc = {1:"김길동", 10:"박길동"}
student_dc
```

[24]:
```
{1: '김길동', 10: '박길동'}
```

[25]:
```
# index 대신 key 형태로 사용
student_dc[0]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[25], line 2
      1 # index 대신 key 형태로 사용
----> 2 student_dc[0]

KeyError: 0
```

[26]:
```
student_dc[10]
```

[26]:
```
'박길동'
```

[27]:
```
10 in student_dc
```

[27]:
```
True
```

[28]:
```
student_dc["a-3"] = "이길동"
```

데이터 구조

```
[27]: 10 in student_dc

[27]: True

[28]: student_dc["a-3"] = "이길동"
      student_dc

[28]: {1: '김길동', 10: '박길동', 'a-3': '이길동'}

[29]: student_dc['a-3']

[29]: '이길동'

[30]: student_dc['a-3'] = "홍길동"

[31]: student_dc

[31]: {1: '김길동', 10: '박길동', 'a-3': '홍길동'}
```

2.5 자료형 변환

```
[32]: num

[32]: {1, 2, 3, 4}

[33]: num = list(num)

[34]: num

[34]: [1, 2, 3, 4]
```

데이터 구조

```
[31]: student_dc
[31]: {1: '김길동', 10: '박길동', 'a-3': '홍길동'}

      2.5 자료형 변환

[32]: num
[32]: {1, 2, 3, 4}
[33]: num = list(num)
[34]: num
[34]: [1, 2, 3, 4]
[35]: num = tuple(num)
[36]: num
[36]: (1, 2, 3, 4)

      3.데이터 포멧팅

[37]: print('이름은 ' + name +"입니다")
      이름은 홍길동입니다
[38]: # 문자형 형태로 변환 필요
```

데이터 포멧팅

3.데이터 포멧팅

[37]:
```
print('이름은 ' + name +"입니다")
```
```
이름은 홍길동입니다
```

[38]:
```
# 문자형 형태로 변환 필요
print('tot1: ' + tot1)
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[38], line 2
      1 # 문자형 형태로 변환 필요
----> 2 print('tot1: ' + tot1)

TypeError: can only concatenate str (not "int") to str
```

[39]:
```
print('tot1의 타입은' + type(tot1))
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[39], line 1
----> 1 print('tot1의 타입은' + type(tot1))

TypeError: can only concatenate str (not "type") to str
```

[40]:
```
print('tot1: ' + str(tot1))
print('tot1의 타입은' + str(type(tot1)))
```

데이터 포멧팅

```
TypeError: can only concatenate str (not "type") to str
```

[40]:
```
print('tot1: ' + str(tot1))
print('tot1의 타입은' + str(type(tot1)))
```

```
tot1: 5
tot1의 타입은<class 'int'>
```

[41]:
```
# 기본 출력
# ,로 연결하면 변환없이 출력 가능
print('tot1:', tot1) # 1칸 띄어쓰기 처리
print('tot1의 타입은', type(tot1), ' 입니다.')
```

```
tot1: 5
tot1의 타입은 <class 'int'>  입니다.
```

[42]:
```
# 문자열 포멧 이용
print('tot1: %d' % tot1)
print('tot1: %d' % tot2)
print('tot1: %0.2f' % tot2)
print('이름: %s' % name)
print('tot1의 값: %0.2f, 이름: %s' % (tot1,name) )
```

```
tot1: 5
tot1: 3
tot1: 3.14
이름: 홍길동
tot1의 값: 5.00, 이름: 홍길동
```

데이터 포멧팅

```
tot1: 5
tot1: 3
tot1: 3.14
이름: 홍길동
tot1의 값: 5.00, 이름: 홍길동
```

[43]:
```
# format 이용
print('tot1의 값: {0}'.format(tot1))
print('tot2의 값: {0:0.2f}, 이름: {1}'.format(tot2, name))

#.format 대신에 f 이용
print('tot2의 값: {0.2f}, 이름: {name}')
print(f'tot2의 값: {tot2:0.2f}, 이름: {name}')
```

```
tot1의 값: 5
tot2의 값: 3.14, 이름: 홍길동
tot2의 값: {0.2f}, 이름: {name}
tot2의 값: 3.14, 이름: 홍길동
```

[ ]:

연습문제1

- 1. 변수와 데이터 포멧팅을 이용해 본인을 소개해 보세요
  - 이름(name)
  - 나이(age)
  - 소속(dep)
  - 직위(pos)
  - 취미(hob)

- 결과화면

```
안녕하세요.
저의 이름은 이새봄 이라고 합니다.
간단하게 제 소개를 하고자 합니다.
----------------------
이름: 이새봄
나이: 38
소속: 삼육대학교 SW융합교육원
직위: 교수
취미: 코딩
```

연습문제 2

- 2. 역명을 리스트로 만들고 2번째 역명을 표시해 보세요

  - 역명

    ```
    ["시청", "서울역", "용산", "노량진"]
    ```

  - 결과

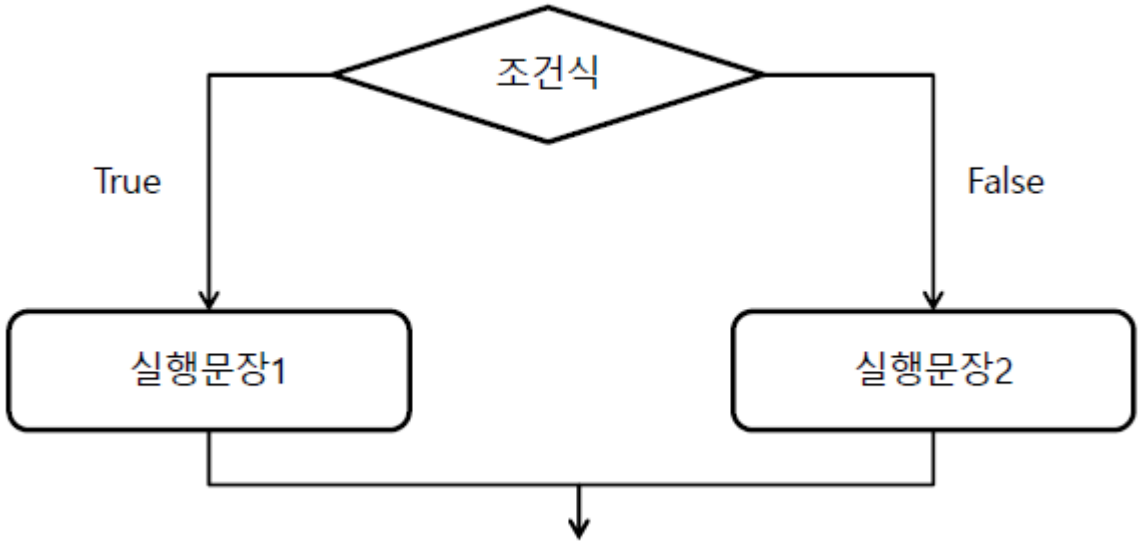
    ```
    역번호: 2, 역명: 서울역
    ```

파이썬 기초 문법

제어문

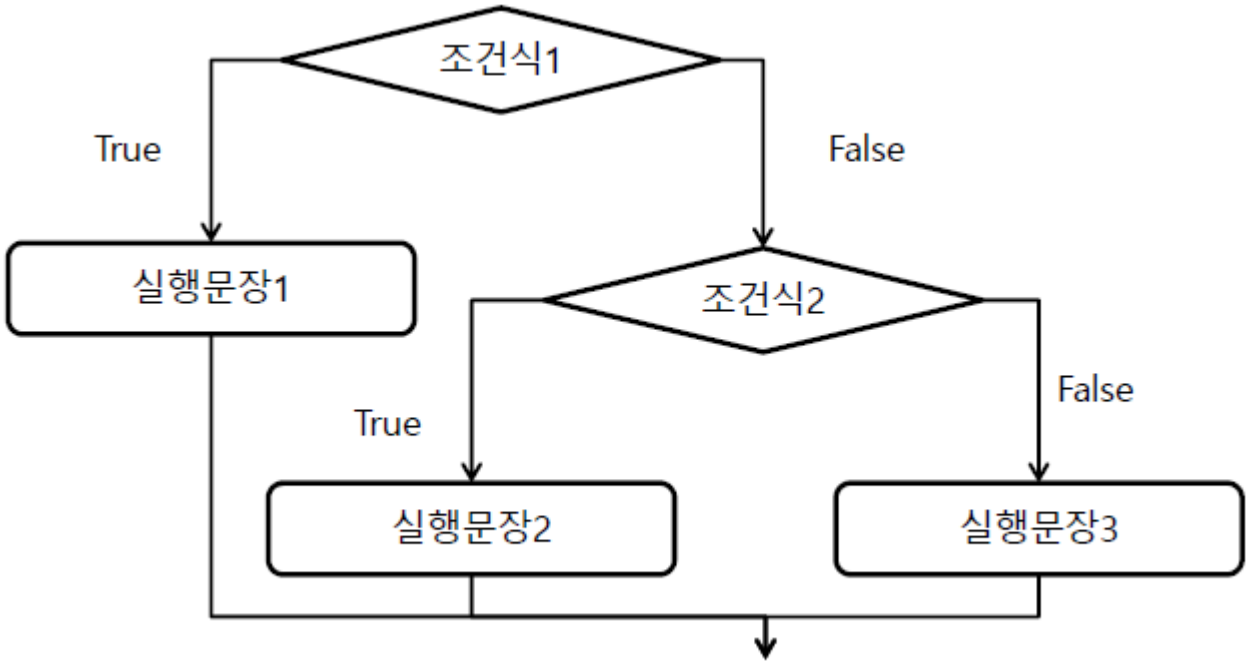
- IF

```
If (조건식):
    실행문장1
else:
    실행문장2
```



다중IF

```
If (조건식):
    실행문장1
elif:
    실행문장2
else:
    실행문장3
```

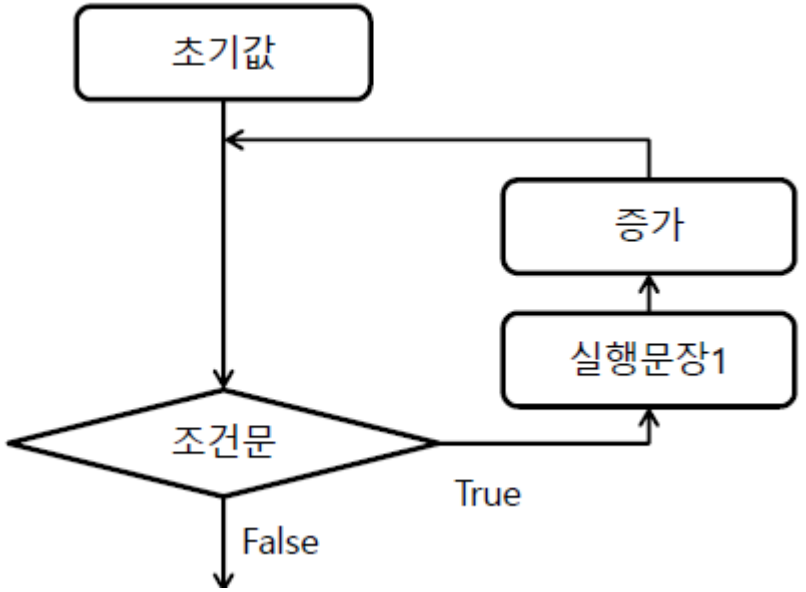


반복문

for

for i in (초기값, 최종값, 증감):

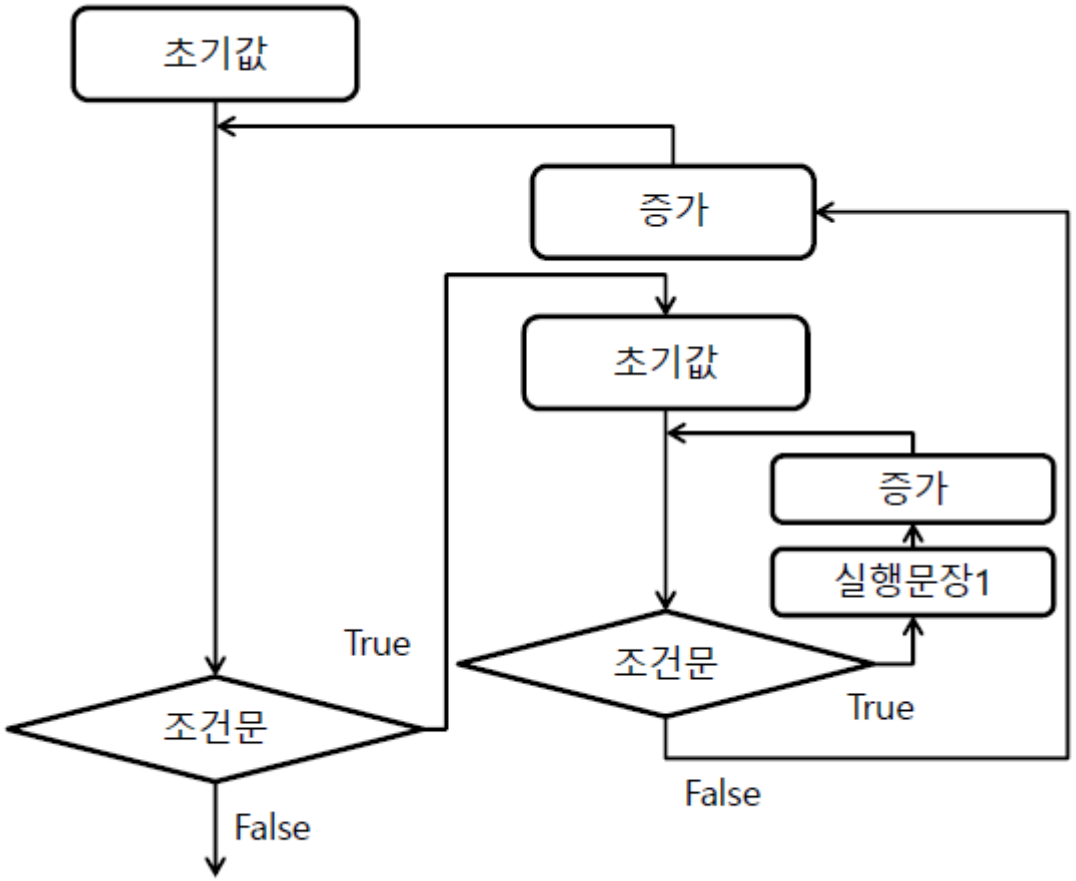
    실행문장1



반복문

다중for

```
for i in (초기값, 최종값, 증감):
    for j in (초기값, 최종값, 증감):
        실행문장1
```



반복문

- 리스트 이용
  - test = [1,2,3,4]
  - for i in (test):
    실행문장1

- enumerate (인덱스 확인이 필요할 때)
  - test = [1,2,3,4]
  - for i in enumerate(test):
    실행문장1

If 제어문

02.Python 기초 문법

- 제어문, 반복문, 함수

1.if 제어문

1.1 일반제어문

[1]:
```
myscore = input('성적은 ')
myscore = int(myscore)

if (myscore >= 80):
    print( "잘했어요")
else:
    print("다음엔 더 열심히")
```

```
성적은  90
잘했어요
```

1.2 다중제어문

[2]:
```
if (myscore >= 90):
    print('A')
elif (myscore >= 80):
    print('B')
else:
```

If 제어문

[1]:
```
myscore = input('성적은 ')
myscore = int(myscore)

if (myscore >= 80):
    print( "잘했어요")
else:
    print("다음엔 더 열심히")
```

```
성적은  90
잘했어요
```

1.2 다중제어문

[2]:
```
if (myscore >= 90):
    print('A')
elif (myscore >= 80):
    print('B')
else:
    print('C')
```

```
A
```

2.for 반복문

2.1 기본 반복문

[3]:
```
cal=0
```

For 반복문

A

2.for 반복문

2.1 기본 반복문

[3]:
```
cal=0

for i in range(101):
    cal += i

print(cal)
```

```
5050
```

2.2 리스트 이용

[4]:
```
test = [1,2,3,4]

for i in test:
    print(i)
```

```
1
2
3
4
```

[5]:
```
name = ['홍길동','이상철']
```

For 반복문

[5]:
```
name = ['홍길동','이상철']

for i in name:
    print(i)
```

```
홍길동
이상철
```

[6]:
```
# 한줄로 for구문 사용할 때
name_len = [len(i) for i in name]
name_len
```

[6]:
```
[3, 3]
```

2.3 enumerate (인덱스 확인이 필요할때)

[7]:
```
for i in enumerate(name):
    print(i)

type(i)
```

```
(0, '홍길동')
(1, '이상철')
```

[7]:
```
tuple
```

3.함수와 클래스

3.1 반환값이 없는 함수

함수와 클래스

[7]: tuple

3.함수와 클래스

3.1 반환값이 없는 함수

[8]:
```
def avg(x, y):
    print((x + y)/2)

avg(4, 5)
```

```
4.5
```

3.2 반환값이 있는 함수

[9]:
```
def avg1(x, y):
    return((x + y)/2)

a = avg1(7,5)
print(a)
```

```
6.0
```

3.3 객체지향 클래스

[10]:
```
class Cal():

    def __init__(self, x1, x2): # init: 초기값 세팅, self=자기객체를 의미
```

함수와 클래스

```
6.0
```

3.3 객체지향 클래스

[10]:
```python
class Cal():

    def __init__(self, x1, x2): # init: 초기값 세팅, self=자기객체를 의미
        self.x1 = x1
        self.x2 = x2

    def add(self):
        result = self.x1 + self.x2
        return result

    def sub(self):
        result = self.x1 - self.x2
        return result
```

[11]:
```python
# 객체 생성
cal = Cal(3,5)

print(cal.add())
print(cal.sub())
```

```
8
-2
```

연습문제1

- Q1. 3의 배수만 더하세요(1 부터 100 까지 중)
  - (예)3+6+9+... 답은 1683입니다
- Q2. 3의 배수이거나 4의 배수인 것을 더하시오(1 부터 100 까지)
  - (예)3+4+6+8+9+12+...답은 2551입니다.
- Q3. 3의 배수이면서 4의 배수인 것을 더하세요(1 부터 100 까지)
  - (예)12+24...: 답은432입니다.
- Q4. 3의 배수이거나 4의 배수인 것 만 빼고 더하시오(1 부터 100까지)
  - (예)1+2+5+7+10+11+...: 단, 12의 배수를 이용하지 마세요. 답은 2499입니다

연습문제2

- Q5. 구구단을 출력해 보세요- 예) 2 * 1 =
  - 2, 2 * 2 = 4...

- Q6. 구구단의 단을 입력받아 그 단만 출력하도록 만들어 보세요.
  - 단, input으로 값을 가져올 때는 변수의 값이 text로 변환되기 때문에 이를 숫자형으로 바꾸어 주어야 합니다.

```
2 단
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
3 단
3 x 1 = 3
3 x 2 = 6
```

```
구구단  4
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
```

연습문제3

- Q7. 역명을 리스트로 만들고 역명을 순서대로 표시해 보세요
  - enumerate를 이용

```
1번역은 시청 역입니다.
2번역은 서울역 역입니다.
3번역은 용산 역입니다.
4번역은 노량진 역입니다.
```

- Q8. 1부터 100까지의 합을 반환값이 있는 함수로 만들어 보세요.

```
1부터 100까지의 합은 5050 입니다.
```

데이터 처리

1. 넘파이

03.데이터 처리(numpy, pandas)

1.넘파이

1.1 넘파이 배열

[1]:
```
num1 = [1, 2, 3, 4]
num2 = [5, 6, 7, 8]
num = [num1, num2]
num
```

[1]:
```
[[1, 2, 3, 4], [5, 6, 7, 8]]
```

[2]:
```
# 계산이 안됨
num1 + num2
```

[2]:
```
[1, 2, 3, 4, 5, 6, 7, 8]
```

[3]:
```
import numpy as np

num1 = np.array(num1)
num2 = np.array(num2)
num1
```

[3]:
```
array([1, 2, 3, 4])
```

1. 넘파이

```
[3]: array([1, 2, 3, 4])

[4]: num1 + num2

[4]: array([ 6,  8, 10, 12])

[5]: num1.sum()

[5]: 10

[6]: num = np.array([num1, num2])
     num

[6]: array([[1, 2, 3, 4],
            [5, 6, 7, 8]])

[7]: # 배열의 행, 열 갯수 확인
     num.shape

[7]: (2, 4)
```

1.2 넘파이 인덱싱

```
[8]: # 배열의 인덱싱
     num[1,3]

[8]: 8

[9]: # 배열 자르기(slicing[행,열])
     num3 = num[:2, :2]
```

1. 넘파이

```
[8]: 8

[9]: # 배열 자르기(slicing[행,열])
     num3 = num[:2, :2]
     num3

[9]: array([[1, 2],
            [5, 6]])

[10]: # 배열 자르기(slicing[행,열])
      num3 = num[0:, 1:]
      num3

[10]: array([[2, 3, 4],
             [6, 7, 8]])
```

1.3 넘파이 함수 ¶

```
[11]: # 열기준으로 합계
      num.sum(axis=0)

[11]: array([ 6,  8, 10, 12])

[12]: # 행기준으로 합계
      num.sum(axis=1)

[12]: array([10, 26])
```

1. 넘파이

```
[10]: array([[2, 3, 4],
             [6, 7, 8]])
```

1.3 넘파이 함수

```
[11]: # 열기준으로 합계
      num.sum(axis=0)
```

```
[11]: array([ 6,  8, 10, 12])
```

```
[12]: # 행기준으로 합계
      num.sum(axis=1)
```

```
[12]: array([10, 26])
```

2.판다스 배열

- https://pandas.pydata.org/docs/user_guide/index.html

2.1 데이터 가져오기(dataframe)

```
[13]: import pandas as pd

      flight_df = pd.read_csv('03.flights.csv', encoding="cp949")
      flight_df.head()
```

[13]:

	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	flight	origin	dest
0	1	9	813.0	-9.0	1014.0	-5.0	EV	4691	EWR	DAY

2. 판다스 배열

2.판다스 배열

- https://pandas.pydata.org/docs/user_guide/index.html

2.1 데이터 가져오기(dataframe)

[13]:
```
import pandas as pd

flight_df = pd.read_csv('03.flights.csv', encoding="cp949")
flight_df.head()
```

[13]:

	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	flight	origin	dest
0	1	9	813.0	-9.0	1014.0	-5.0	EV	4691	EWR	DAY
1	9	4	2031.0	-9.0	2225.0	-34.0	EV	4085	EWR	OMA
2	3	8	2248.0	128.0	134.0	111.0	B6	629	JFK	HOU
3	7	23	743.0	0.0	1106.0	3.0	UA	1668	EWR	SFO
4	3	15	754.0	-5.0	916.0	-20.0	UA	393	EWR	ORD

[14]:
```
# DataFrame: Series 집합
type(flight_df)
```

[14]:
```
pandas.core.frame.DataFrame
```

[15]:
```
# pd.Series: 한개 열로 구성
type(flight_df["month"])
```

2. 판다스 배열

```
[14]: # DataFrame: Series 집합
      type(flight_df)

[14]: pandas.core.frame.DataFrame

[15]: # pd.Series: 한개 열로 구성
      type(flight_df["month"])

[15]: pandas.core.series.Series

[16]: flight_df["month"].head(10)

[16]: 0     1
      1     9
      2     3
      3     7
      4     3
      5     2
      6    10
      7     5
      8     9
      9    11
      Name: month, dtype: int64
```

2.2 data type

```
[17]: flight_df.dtypes
```

2. 판다스 배열

2.2 data type

[17]:
```
flight_df.dtypes
```

[17]:
```
month          int64
day            int64
dep_time     float64
dep_delay    float64
arr_time     float64
arr_delay    float64
carrier       object
flight         int64
origin        object
dest          object
dtype: object
```

[18]:
```
flight_df.shape
```

[18]:
```
(10000, 10)
```

[19]:
```
flight_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   month      10000 non-null  int64
```

2. 판다스 배열

2.3 data 속성

[20]:
```
# arrary형식으로 추출
flight_df.columns
```

[20]:
```
Index(['month', 'day', 'dep_time', 'dep_delay', 'arr_time', 'arr_delay',
       'carrier', 'flight', 'origin', 'dest'],
      dtype='object')
```

[21]:
```
# 변수명만 추출
flight_df.columns.tolist()
```

[21]:
```
['month',
 'day',
 'dep_time',
 'dep_delay',
 'arr_time',
 'arr_delay',
 'carrier',
 'flight',
 'origin',
 'dest']
```

[22]:
```
flight_df.values
```

[22]:
```
array([[1, 9, 813.0, ..., 4691, 'EWR', 'DAY'],
       [9, 4, 2031.0, ..., 4085, 'EWR', 'OMA'],
       [3, 8, 2248.0, ..., 629, 'JFK', 'HOU'],
```

2. 판다스 배열

2.4 column(열), row(행) 추출

[23]:
```
flight_df.head(10)
```

[23]:

	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	flight	origin	dest
0	1	9	813.0	-9.0	1014.0	-5.0	EV	4691	EWR	DAY
1	9	4	2031.0	-9.0	2225.0	-34.0	EV	4085	EWR	OMA
2	3	8	2248.0	128.0	134.0	111.0	B6	629	JFK	HOU
3	7	23	743.0	0.0	1106.0	3.0	UA	1668	EWR	SFO
4	3	15	754.0	-5.0	916.0	-20.0	UA	393	EWR	ORD
5	2	22	1723.0	8.0	1830.0	5.0	EV	4373	EWR	DCA
6	10	23	919.0	-10.0	1234.0	0.0	AA	1223	EWR	DFW
7	5	13	1900.0	40.0	2029.0	28.0	9E	4277	JFK	ORF
8	9	12	1011.0	-4.0	1233.0	21.0	US	2067	JFK	CLT
9	11	5	1947.0	-8.0	2111.0	-35.0	EV	5038	LGA	BHM

[24]:
```
# 1개 열추출
flight_df.month
```

[24]:
```
0     1
1     9
2     3
3     7
4     3
     ..
```

2. 판다스 배열

Name: month, Length: 10000, dtype: int64

[25]:
```
# 1개 열추출
flight_df["month"]
```

[25]:
```
0        1
1        9
2        3
3        7
4        3
       ..
9995     8
9996    11
9997     5
9998     3
9999     4
Name: month, Length: 10000, dtype: int64
```

[26]:
```
# 2개열 추출
# loc(컬럼명으로 추출)
flight_df.loc[:, ['month', 'day']]
```

[26]:

	month	day
0	1	9
1	9	4
2	3	8
3	7	23
4	3	15

2. 판다스 배열

10000 rows × 2 columns

[27]:
```
# 여러개 열을 연속으로 추출
# iloc(index로 추출)
# index= 0으로 시작, 마지막은 +1
flight_df.iloc[:, 0:4]
```

[27]:

	month	day	dep_time	dep_delay
0	1	9	813.0	-9.0
1	9	4	2031.0	-9.0
2	3	8	2248.0	128.0
3	7	23	743.0	0.0
4	3	15	754.0	-5.0
...	...	...	...	...
9995	8	3	1737.0	2.0
9996	11	21	743.0	-2.0
9997	5	9	1359.0	1.0
9998	3	6	NaN	NaN
9999	4	29	1644.0	14.0

10000 rows × 4 columns

[28]:
```
# dataframe에서 행 선택하기
flight_df.iloc[0:5, :]
```

2. 판다스 배열

10000 rows × 4 columns

[28]:
```
# dataframe에서 행 선택하기
flight_df.iloc[0:5, :]
```

[28]:

	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	flight	origin	dest
0	1	9	813.0	-9.0	1014.0	-5.0	EV	4691	EWR	DAY
1	9	4	2031.0	-9.0	2225.0	-34.0	EV	4085	EWR	OMA
2	3	8	2248.0	128.0	134.0	111.0	B6	629	JFK	HOU
3	7	23	743.0	0.0	1106.0	3.0	UA	1668	EWR	SFO
4	3	15	754.0	-5.0	916.0	-20.0	UA	393	EWR	ORD

[29]:
```
# column 삭제
flight_df = flight_df.drop('flight', axis=1)
flight_df.head()
```

[29]:

	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	origin	dest
0	1	9	813.0	-9.0	1014.0	-5.0	EV	EWR	DAY
1	9	4	2031.0	-9.0	2225.0	-34.0	EV	EWR	OMA
2	3	8	2248.0	128.0	134.0	111.0	B6	JFK	HOU
3	7	23	743.0	0.0	1106.0	3.0	UA	EWR	SFO
4	3	15	754.0	-5.0	916.0	-20.0	UA	EWR	ORD

2.5 filtering

2. 판다스 배열

2.5 filtering

[30]:
```
flight_df[(flight_df.month == 1)].head()
```

[30]:

	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	origin	dest
0	1	9	813.0	-9.0	1014.0	-5.0	EV	EWR	DAY
14	1	26	555.0	-6.0	931.0	10.0	UA	JFK	LAX
29	1	25	1723.0	-6.0	2111.0	25.0	DL	EWR	SLC
72	1	4	1128.0	-5.0	1304.0	-5.0	EV	EWR	RDU
73	1	9	2019.0	-11.0	2159.0	-7.0	FL	LGA	CAK

[31]:
```
flight_df[(flight_df.month == 1) & (flight_df.day == 31)].head()
```

[31]:

	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	origin	dest
494	1	31	1124.0	-6.0	1443.0	13.0	DL	JFK	MCO
1070	1	31	804.0	5.0	1026.0	1.0	UA	LGA	DEN
1343	1	31	651.0	17.0	1028.0	53.0	UA	EWR	MIA
1846	1	31	1533.0	-2.0	1843.0	-7.0	AA	LGA	DFW
1959	1	31	1534.0	-6.0	1655.0	-21.0	9E	JFK	ROC

[32]:
```
flight_df[(flight_df.month == 1) | (flight_df.month == 2)].head() # pd에서는 or -> |
```

[32]:

	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	origin	dest
0	1	9	813.0	-9.0	1014.0	-5.0	EV	EWR	DAY
5	2	22	1723.0	8.0	1830.0	5.0	EV	EWR	DCA

2. 판다스 배열

```
[33]: flight_df[flight_df['dep_delay'] < 0].head()
```

[33]:

	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	origin	dest
0	1	9	813.0	-9.0	1014.0	-5.0	EV	EWR	DAY
1	9	4	2031.0	-9.0	2225.0	-34.0	EV	EWR	OMA
4	3	15	754.0	-5.0	916.0	-20.0	UA	EWR	ORD
6	10	23	919.0	-10.0	1234.0	0.0	AA	EWR	DFW
8	9	12	1011.0	-4.0	1233.0	21.0	US	JFK	CLT

```
[34]: flight_df.query('dep_delay < 0').head()
```

[34]:

	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	origin	dest
0	1	9	813.0	-9.0	1014.0	-5.0	EV	EWR	DAY
1	9	4	2031.0	-9.0	2225.0	-34.0	EV	EWR	OMA
4	3	15	754.0	-5.0	916.0	-20.0	UA	EWR	ORD
6	10	23	919.0	-10.0	1234.0	0.0	AA	EWR	DFW
8	9	12	1011.0	-4.0	1233.0	21.0	US	JFK	CLT

```
[35]: flight_df[(flight_df['dep_delay'] < 0) & (flight_df['origin'] == 'EWR')].head()
```

[35]:

	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	origin	dest
0	1	9	813.0	-9.0	1014.0	-5.0	EV	EWR	DAY
1	9	4	2031.0	-9.0	2225.0	-34.0	EV	EWR	OMA
4	3	15	754.0	-5.0	916.0	-20.0	UA	EWR	ORD

2. 판다스 배열

4	3	15	754.0	-5.0	916.0	-20.0	UA	EWR	ORD
6	10	23	919.0	-10.0	1234.0	0.0	AA	EWR	DFW
12	2	16	1857.0	-1.0	2201.0	5.0	UA	EWR	PBI

[36]:
```
filter = (flight_df['dep_delay'] < 0) & (flight_df['origin'] == 'EWR')
flight_df.loc[filter, ['dep_delay', 'origin']]
```

[36]:

	dep_delay	origin
0	-9.0	EWR
1	-9.0	EWR
4	-5.0	EWR
6	-10.0	EWR
12	-1.0	EWR
...	...	...
9968	-2.0	EWR
9979	-2.0	EWR
9982	-4.0	EWR
9989	-7.0	EWR
9996	-2.0	EWR

1730 rows × 2 columns

2.6 sorting

2. 판다스 배열

2.6 sorting

[37]:
```
# sorting
flight_df.sort_values(by="dep_delay", ascending = False)
```

[37]:

	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	origin	dest
9980	8	6	1846.0	411.0	2110.0	410.0	WN	LGA	DEN
5686	5	22	146.0	406.0	255.0	378.0	UA	LGA	ORD
5281	9	12	2125.0	386.0	2258.0	396.0	B6	JFK	BTV
2652	3	8	1746.0	376.0	1938.0	359.0	DL	LGA	MSP
2679	12	5	1548.0	373.0	1712.0	357.0	WN	LGA	MDW
...	...	...	...	...	...	...	...	...	...
9748	3	6	NaN	NaN	NaN	NaN	AA	LGA	ORD
9768	4	10	NaN	NaN	NaN	NaN	US	LGA	BOS
9805	1	30	NaN	NaN	NaN	NaN	EV	EWR	PIT
9971	3	8	NaN	NaN	NaN	NaN	EV	EWR	BTV
9998	3	6	NaN	NaN	NaN	NaN	EV	LGA	CHO

10000 rows × 9 columns

[38]:
```
# sorting
flight_df[(flight_df.month == 12)].sort_values(by="day", ascending = True)
```

[38]:

	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	origin	dest
6400	12	1	1930.0	30.0	2124.0	0.0	9E	LGA	DSM

2. 판다스 배열

819 rows × 9 columns

2.7 새로운 변수 추가

```
[39]: flight_df['gain'] = flight_df['dep_delay'] - flight_df['arr_delay']
```

```
[40]: flight_df
```

[40]:

	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	origin	dest	gain
0	1	9	813.0	-9.0	1014.0	-5.0	EV	EWR	DAY	-4.0
1	9	4	2031.0	-9.0	2225.0	-34.0	EV	EWR	OMA	25.0
2	3	8	2248.0	128.0	134.0	111.0	B6	JFK	HOU	17.0
3	7	23	743.0	0.0	1106.0	3.0	UA	EWR	SFO	-3.0
4	3	15	754.0	-5.0	916.0	-20.0	UA	EWR	ORD	15.0
...	...	...	...	...	...	...	...	...	...	...
9995	8	3	1737.0	2.0	2025.0	-5.0	AA	JFK	IAH	7.0
9996	11	21	743.0	-2.0	1002.0	-10.0	DL	EWR	ATL	8.0
9997	5	9	1359.0	1.0	1631.0	-33.0	B6	JFK	LGB	34.0
9998	3	6	NaN	NaN	NaN	NaN	EV	LGA	CHO	NaN
9999	4	29	1644.0	14.0	1806.0	-5.0	UA	EWR	CLE	19.0

10000 rows × 10 columns

2.8 groupby

2. 판다스 배열

10000 rows × 10 columns

2.8 groupby

```
[41]: flight_df.groupby('origin')['dep_delay'].mean()
```

```
[41]: origin
      EWR    15.034483
      JFK    11.202257
      LGA    10.524015
      Name: dep_delay, dtype: float64
```

Click to add a cell.

연습문제1

- 1.데이터중에서 weather를 가져오기
- 2.origin(출발공항)이 EWR 또는 JFK만 가져오기
- 3.origin에서 temp까지 변수를 모두 가져오기
- 4.화씨 온도를 섭씨온도로 변환하기(F-32)/1.8

$$C = \frac{(온도 - 32)}{1.8}$$

- 5.월별 평균 온도(섭씨)구하기
- 6.평균이 높은 값으로 sorting

```
month
7    27.177465
8    24.454902
6    20.916000
9    20.144186
5    16.675000
Name: temp_c, dtype: float64
```

연습문제2

- Q2. 아래의 순서대로 데이터를 처리하세요
  - 1.데이터중에서 mpg 를 가져오기
  - 2.manufacturer(제조사)가 audi 또는 hyundai 만 가져오기
  - 3.manufacturer, model, displ, cty 변수만 가져오기
  - 4.model로 평균 연비(cty) 구하기
  - 5.평균이 높은 값으로 sorting

```
model
sonata        19.000000
a4            18.857143
tiburon       18.285714
a4 quattro    17.125000
a6 quattro    16.000000
Name: cty, dtype: float64
```

Q&A