



HANDBOOK

실패 없이 완주하는 파이썬 데이터 분석 입문

Selena 강사님

목차

03	STEP 1. 파이썬 시작
06	STEP 2. 변수와 자료형
21	STEP 3. 입력과 출력
25	STEP 4. 조건문
28	STEP 5. 반복문
33	STEP 6. 함수
37	STEP 7. 클래스
43	STEP 8. 모듈
50	STEP 9. 예외 처리
55	STEP 10. 파이썬 라이브러리 개념
58	STEP 11. 수치계산
73	STEP 12. 데이터 처리
88	STEP 13-1. 데이터 시각화
98	STEP 13-2. 데이터 시각화
106	STEP 14. 웹 데이터 수집

STEP 1. 파이썬 시작

1. Python Programming 의미

- Python
 - 컴퓨터 시스템을 구동시키는 소프트웨어를 작성하기 위한 형식 언어
 - 인간이 컴퓨터에 명령을 내릴 때 필요한 프로그래밍 언어
- Programming
 - 인간이 생각하는 것을 컴퓨터에 명령하는 것
 - 프로그램을 만드는 모든 작업이며 개발이라고 칭하기도 함
- Python Programming
 - Python 언어를 이용하여 컴퓨터에 명령을 내리는 행위

“Selena's Comment!”

한국에서는 한국어로 대화를 나누고 미국에 나가면 영어로 의사소통을 해요.

그렇다면 컴퓨터와 이야기를 나눠야 한다면 어떻게 해야 할까요?

그때 사용하는 것이 프로그래밍 언어예요!

이번 강의에서는 파이썬이라는 프로그래밍 언어에 대해 배워볼 거예요.

그러면 파이썬 프로그래밍은 무슨 뜻일까요?

파이썬이라는 프로그래밍 언어를 이용하여 컴퓨터에 명령을 내리는 행위로 알아주시면 됩니다! 쉽죠~?

2. Python 소개

- 가장 활용도가 높고 쉬운 프로그래밍 언어
- MATLAB, R과 같은 도메인 특화 언어와 Java, C 같은 범용 언어의 장점을 고루 갖춤
- 통계, 머신러닝, 자연어, 이미지, 시각화 등을 포함한 풍부한 라이브러리를 지님
- 인공지능 개발, 웹과 앱 그리고 게임 제작, 핀테크 및 블록체인 구현으로 활용
- 브라우저 기반의 인터랙티브 프로그래밍 환경인 Jupyter Notebook으로 쉽게 구현

3. Python 장점

1. 간결하고 직관적인 문법
 - 인간의 사고와 비슷하여 다른 언어에 비해 빠르게 배워 활용 가능
2. 같은 결과를 요구할 때 Java로 10줄, Python으로 3줄!
 - 높은 확장성과 이식성
 - 다른 언어나 라이브러리에 쉽게 연동 가능
3. 다양한 라이브러리 존재
 - 파이썬을 배우고 필요한 라이브러리만 익힌다면 불가능한 일은 없다!

4. Python 활용 분야

- 인공지능 제작
 - 기계 학습에 도움이 되는 다양한 라이브러리로 쉽게 인공지능 개발 가능
- 웹과 애플리케이션 제작
 - Python으로 제작된 구글, 인스타그램, 넷플릭스, 드롭박스처럼 개발 가능
- 게임 제작
 - 2D 게임부터 3D 게임까지 개발 가능
- 핀테크 및 블록체인 구현
 - 핀테크의 다양한 분야에서 응용되며 블록체인 구현 가능

5. Python Editor 종류

- Editor 란?
 - 소스 코드가 들어 있는 파일을 편집할 수 있는 프로그래밍 툴
 - Python은 편의성을 위해 별도의 에디터 프로그램을 설치하여 사용
- Editor 종류
 - Pycharm – 개발자들이 가장 많이 사용하는 에디터
 - Jupyter – 웹 브라우저에서 파이썬을 작성하고 실행하는 에디터
 - Visual Studio Code – 마이크로 소프트가 개발한 에디터
 - IDLE – 파이썬 설치시 내장되어 있는 기본 파이썬 에디터이며 심화적인 기능은 부족한 편

6. Python Jupyter Notebook

- 오픈 소스 기반의 웹 애플리케이션
- 파이썬으로 작성한 여러 개의 코드와 실행 결과를 하나의 문서처럼 관리 가능
- 즉, 프로그램 코드 + 결과 + 문서를 위한 대화식 개발 환경
- 기존의 파이썬 IDLE을 사용하는 것과 비교했을 때, 일부 코드만 실행하여 결과 확인 가능

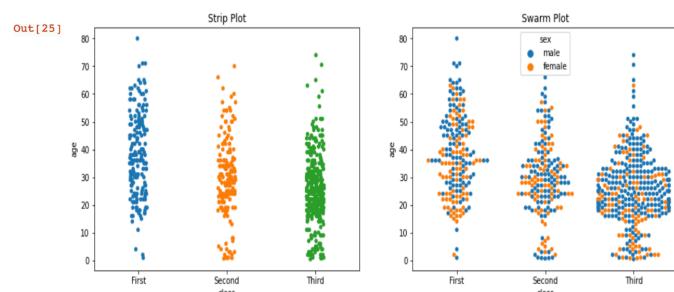
```
In [25]:# 그래프 객체 생성 (figure에 2개의 서브 플롯을 생성)
fig = plt.figure(figsize=(15, 5))
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)

# 이산형 변수의 분포 – 데이터 분산 미고려
# x축 변수, y축 변수, 데이터 셋, axe 객체(1번째 그래프)
sns.stripplot(x='class', y='age', data=titanic, ax=ax1)

# 이산형 변수의 분포 – 데이터 분산 고려 (중복 X)
# x축 변수, y축 변수, 데이터 셋, axe 객체(2번째 그래프), 성별로 색상 구분
sns.swarmplot(x='class', y='age', data=titanic, ax=ax2, hue='sex')

# 차트 제목 표시
ax1.set_title('Strip Plot')
ax2.set_title('Swarm Plot')

plt.show()
```



7. Google Colaboratory (Colab)

- Googel Colaboratory = Google Drive + Jupyter Notebook
- 구글 코랩은 주피터 노트북 기반의 오픈소스 프로젝트
- 구글 코랩의 장점
 - 대부분의 파이썬 패키지들이 설치 되어 있음
 - 구글 아이디와 인터넷만 있다면 하드웨어와 장소에 구애받지 않고 코딩 가능
 - 구글 드라이브와 연동하여 파일 불러오기
 - 무료로 GPU도 사용할 수 있어서 고성능 딥러닝 프로젝트 가능
 - 주피터 노트북 문서를 여러 사람이 동시에 열어서 함께 편집 가능

STEP 2. 변수와 자료형

01. 변수

1. 변수(variable)

✓ 변수는 어떠한 값(value)을 담는 저장 공간(storage location)과 그에 상응하는 이름(symbolic name)

1-1. 변수 이름

- 값(value)의 의미를 추측할 수 있는 이름 사용
- 변수 선언할 때 몇 가지 규칙 존재
 - 영문자 혹은 _(underscore)로 시작
 - 대소문자 구분
 - 숫자로 시작할 수 없음
 - 특수 문자(+, -, %, &)는 사용할 수 없음
 - 변수 이름에 공백이 있으면 안 됨(보통 _로 이어줌)
 - 파이썬의 예약어(if, while, for 등)는 사용할 수 없음

“Selena’s Comment!”

Q. Selena 강사님, 변수의 이름이 중요할까요?

A. 정말 중요합니다! 혼자 코딩할 때는 A1, A2, ... 라는 이름을 사용해도 괜찮아요. 하지만 협업에서 코딩을 하거나 여러 명과 프로젝트를 할 때는 함께 알아볼 수 있는 변수 이름을 사용해야 해요. 그래서 수강생 여러분들은 기초를 배울 때부터 변수 이름을 신경 쓰는 연습을 합니다!

1-2. 변수의 의미

- 변수란, '변할 수 있는 것'으로 어떠한 값을 담는 '상자'라고 생각하면 됨
 - 예를 들어, A = 1이라고 하는 것은 1을 A라는 상자(변수)에 담아줘! 라는 의미
 - 그리고 이것을 '변수를 선언한다'라고 표현
 - 그동안 사용했던 같다라는 의미의 '='(등호)는 프로그래밍에서 '=='로 표시
- 변수에 줄 수 있는 값은 숫자, 문자, 다른 데이터타입의 형들이 모두 가능
- 변수를 만들고 나면 변수를 저장하는 메모리를 자동으로 생성하고 주소가 생김

1-2. 변수의 의미

- 변수란, '변할 수 있는 것'으로 어떠한 값을 담는 '상자'라고 생각하면 됨
 - 예를 들어, $A = 1$ 이라고 하는 것은 1을 A라는 상자(변수)에 담아줘! 라는 의미
 - 그리고 이것을 '변수를 선언한다'라고 표현
 - 그동안 사용했던 같다라는 의미의 '='(등호)'는 프로그래밍에서 '=='로 표시
- 변수에 줄 수 있는 값은 숫자, 문자, 다른 데이터타입의 형들이 모두 가능
- 변수를 만들고 나면 변수를 저장하는 메모리를 자동으로 생성하고 주소가 생김

1-3. 변수 선언

- 하나의 변수에 하나의 값을 대입
- 두 개의 변수에 같은 값을 대입
- 변수에 변수를 대입

```
# 7을 a라는 변수에 담기
# 1을 A라는 변수에 담기
>>> a = 7
>>> A = 1
```

1-4. 변수 선언

- 변수에 값이 잘 들어갔는지 확인하고 싶다면, print() 함수를 이용

```
# a 변수의 값 출력
# A 변수의 값 출력
>>> print(a)
>>> print(A)
7
1
```

02. 자료형

1. 자료형(data type)

- 프로그램에서 표현하고 저장하는 데이터 유형
- 순서 자료형
 - 숫자(number) : 정수, log형 정수, 실수, 복소수 등
 - 문자열(strings) : 문자들의 모임
 - 리스트(Lists) : 순서를 가지는 python 객체의 집합
 - 튜플(tuples) : 순서를 가지는 python 객체의 집합

- 비순서 자료형
 - 딕셔너리(dictionary) : 순서를 가지지 않는 객체의 쌍 (key & value)
 - 집합(set) : 중복이 없고 순서를 가지지 않는 객체의 집합 (집합 관련 데이터 처리)
 - 부울(bool) : 참과 거짓을 나타내는 자료형

2. 숫자형(number)

2-1. 종류

- 숫자형은 숫자 형태로 이루어진 자료형
 - 정수(integer) : 소수점이 없는 숫자
 - 실수, 부동 소수점(floating point) : 소수점이 있는 숫자

2-2. 데이터 타입 출력

- `type()` : 괄호 안에 입력한 데이터 타입을 알려주는 함수
 - `int` : 정수
 - `float` : 실수

```
# type()은 괄호 안에 입력한 자료의 유형을 알려주는 함수
# 정수 3과 실수 -3.2 대한 자료 유형 출력
>>> print(type(3))
>>> print(type(-3.2))

<class 'int'>
<class 'float'>
```

2-3. 연산자

- 사칙 연산자 : 덧셈(+), 뺏셈(-), 곱셈(*), 나눗셈(/)
- 정수 나누기 연산자 : 나머지(%)
- 나머지 연산자 : 몫(//)
- 제곱 연산자 : 제곱(**)

```
# a, b, c 변수 선언
a = 4
b = 3.2
c = 2

# a, c 변수를 통한 나눗셈, 나머지, 몫
>>> print(a/c)
>>> print(a%c)
>>> print(a//c)

2.0
0
2
```

2-4. 관련 함수(자주 사용하는 파이썬 내장 함수)

- int() : 숫자나 문자열을 정수형(integer)으로 변환
- float() : 숫자나 문자열을 실수형(float)으로 변환
- abs() : 절대값 출력
- pow() : 제곱 출력

```
# 절대값 출력
```

```
>>> print(abs(-6.2))
```

```
6.2
```

```
# 제곱 출력
```

```
>>> print(pow(4,2))
```

```
16
```

3. 문자열 자료형(string)

3-1. 문자열 만들기

- 큰 따옴표로 문자열 만들기
- 작은 따옴표로 문자열 만들기
- 문자열 내부에 따옴표 넣기
- 이스케이프 문자를 사용해 문자열 만들기
- 여러 줄 문자열 만들기
- 줄바꿈 없이 문자열 만들기

```
# 문자열 내부에 큰 따옴표 안에 작은 따옴표 넣기
```

```
>>> print("I'm smart")
```

```
I'm smart
```

```
# 이스케이프 문자를 사용해 문자열 만들기
```

```
# 이스케이프 문자는 역슬래시(\) 기호와 함께 조합해서 사용하는 특수문자를 의미
```

```
# \와 함께 사용하면 '문자열을 만드는 기호'가 아니라 '단순한 따옴표'로 인식
```

```
>>> print('Selena는 \'파이썬 프로그래밍\' 강의를 한다. !')
```

```
Selena는 '파이썬 프로그래밍' 강의를 한다.
```

3-2. 이스케이프 문자

- 출력물을 보기 좋게 만드는 용도
 - \n : 줄바꿈
 - \t : 탭
 - \\ : 역슬래시
 - \': 작은 따옴표
 - \" : 큰 따옴표

```
# 줄바꿈, 탭, 작은 따옴표, 큰 따옴표 출력
>>> sentence1 = "1\n2"
>>> sentence2 = "3\t4"
>>> sentence3 = '\\'작은 따옴표\' 출력'
>>> sentence4 = '\"큰 따옴표\" 출력'

>>> print(sentence1)
>>> print(sentence2)
>>> print(sentence3)
>>> print(sentence4)

1
2
3 4
'작은 따옴표' 출력
"큰 따옴표" 출력
```

3-3. 연산자

- 문자열도 숫자형처럼 연산 가능
 - 문자열 연결 연산자 (+)
 - 문자열 반복 연산자 (*)

```
# [+] 문자열 연결 연산자
>>> a = "파이썬! "
>>> b = "프로그래밍!"
>>> print(a+b)

파이썬! 프로그래밍!
```

```
# [*] 문자열 반복 연산자
>>> equal = "=" * 30
>>> sentence = equal + "\n파이썬을 제대로 공부해보자!\n" + equal
>>> print(sentence)

=====
파이썬을 제대로 공부해보자!
=====
```

“Selena’s Comment!”

문자열 자료형에서 연산자인 +, *를 기억하시면 현업에서 유용하답니다!

3-4. 인덱싱과 슬라이싱

- 인덱싱
 - 문자열 내부의 문자 하나를 선택하는 연산자
 - 문자 선택 연산자 ([])
 - 대괄호 [] 안에는 선택할 문자의 위치를 지정하며, 이 숫자를 인덱스(index)라고 부름
 - 순방향 인덱스 : 0부터 시작
 - 역방향 인덱스 : -1부터 시작
 - 문자열[인덱스]로 호출

```
# 문자열 인덱싱 : 문자 하나 선택
#           012 3 4567 8 910111213
#                           -2-1
>>> sentence = '파이썬 하루만에 끝내보자!'
```

```
# 문자열[인덱스]로 호출
# 순방향 인덱스 : 0 부터 시작
# 역방향 인덱스 : -1 부터 시작
>>> print(sentence)
>>> print(sentence[0])
>>> print(sentence[-1])
```

파이썬 하루만에 끝내보자!
파
!

- 슬라이싱

- 문자열의 특정 범위를 선택할 때 사용하는 연산자
- 문자열 범위 선택 연산자 ([:])
- 문자열[인덱스1 : 인덱스2]로 호출
- 파이썬은 '마지막 숫자를 포함하지 않음'으로 적용
- 인덱스1 이상부터 인덱스 2 미만으로 적용

```
# 문자열 슬라이싱 : 특정 범위 선택
#           012 3 4567 8 910111213
#                           -2-1
>>> sentence = '파이썬 하루만에 끝내보자!'
```

```
# 문자열[인덱스1 : 인덱스2]로 호출
# 4:8 = 4, 5, 6, 7
# 4 이상부터 8 미만까지의 범위 선택하여 출력
# 마지막 숫자는 포함하지 않음
>>> print(sentence)
>>> print(sentence[4:8])
```

파이썬 하루만에 끝내보자!
하루만에

3-5. 포맷팅(formatting)

- 문자열 안에 어떤 값을 삽입하는 방법
- % 연산자 포맷팅
 - 문자열 포맷 코드에 맞춰 바로 숫자와 문자를 대입할 때 사용
 - 숫자(정수) 대입 : %d
 - 숫자(실수) 대입 : %f
 - 문자 1개 대입 : %c
 - 문자열 대입 : %s

```
# 한 개의 % 연산자 포맷팅
# 숫자 대입은 %d
>>> sentence = "나는 %d호선을 타고 다녀." % 6
>>> print(sentence)
```

나는 6호선을 타고 다녀.

```
# 두 개 이상의 % 연산자 포맷팅
# 숫자 나타내는 변수와 문자를 나타내는 변수 대입
# 숫자 대입은 %d, 문자 대입은 %s
>>> a = 4
>>> b = "학교"
>>> sentence = "나는 %d호선을 타고 %s를 가." % (a, b)
>>> print(sentence)
```

나는 4호선을 타고 학교를 가.

- format() 함수 포맷팅
 - 중괄호{}를 포함한 문자열 뒤에 도트(.)를 찍고 format() 함수 사용
 - 중괄호 개수와 format() 함수 괄호 안의 매개변수 개수는 동일
 - 함수를 사용하면 문자열의 {} 기호가 format() 함수 괄호 안에 있는 매개변수로 차례로 대치

```
# 한 개의 format() 함수 포맷팅
# {} 안에 숫자 대입
# format 함수 안에 넣을 숫자 삽입
>>> print("나는 {}호선을 타고 다녀.".format(6))
>>> print("나는 {0}호선을 타고 다녀.".format(6))
```

나는 6호선을 타고 다녀.
나는 6호선을 타고 다녀.

```
# 한 개의 format() 함수 포맷팅
# {} 안에 숫자 대입
# format 함수 안에 넣을 숫자 삽입
>>> print("나는 {}호선을 타고 다녀.".format(6))
>>> print("나는 {0}호선을 타고 다녀.".format(6))
```

나는 2호선과 4호선, 2호선을 주로 타고 다녀.
나는 4호선과 2호선을 타고 회사를 가.

3-6. 관련 함수

- len(x) : 문자열 길이 출력
- split() : 문자열 나누기
- count(x) : 문자 개수 세기
- replace('a', 'b') : 문자열 바꾸기
- find('a') : 위치 알려주기
- upper() : 소문자를 대문자로 바꾸기
- lower() : 대문자를 소문자로 바꾸기
- join(x) : 문자열 삽입

```
# len(x) : 문자열 길이 출력
>>> sentence1 = "Selena는 샤브샤브를 좋아해"
>>> print(len(sentence1))
```

17

```
# split() : 문자열 나누기
>>> print(sentence1.split())
['Selena는', '샤브샤브를', '좋아해']

# replace('a', 'b') : 문자열 바꾸기
>>> sentence2 = sentence1.replace('샤브샤브', "육회")
>>> print(sentence1)
>>> print(sentence2)

Selena는 샤브샤브를 좋아해
Selena는 육회를 좋아해

# join(x) : 문자열 삽입
>>> sentence = '.'.join('abcde')
>>> print(sentence)

a.b.c.d.e
```

4. 리스트 자료형(list)

- ✓ 지금까지 활용한 숫자나 문자가 '개별적인 자료'였다면 리스트는 '여러 자료들을 모아서 사용할 수 있는 형태의 자료'
- ✓ 여러 자료형을 담을 수 있고, 여러 요소를 하나의 변수로 사용하고 싶을 때 사용

4-1. 리스트 선언

- 리스트를 생성하는 방법은 대괄호[]에 자료를 쉼표로 구분하여 입력
- 대괄호[] 내부에 넣는 자료를 요소라고 부르며 영어로는 element라고 명칭

4-2. 인덱싱과 슬라이싱

- 리스트 기호인 대괄호[] 안에 들어가는 숫자를 인덱스(index)라고 부름
- 리스트 안에 있는 요소를 각각 사용하려면 리스트 이름 바로 뒤에 대괄호[]를 입력하고
- 자료의 위치를 나타내는 숫자를 입력
- 파이썬은 0부터 시작
- 인덱싱
 - 문자열 내부의 문자 하나를 선택하는 연산자
- 슬라이싱
 - 문자열의 특정 범위를 선택할 때 사용하는 연산자

```
# 대괄호[] 내부에 여러 종류의 자료를 넣어 선언
# 선언한 리스트를 출력하면 내부의 자료를 모두 출력
>>> myList = [1, 1.25, 1, 'word', [1, 2, [3,4]]]
>>> print(myList)

[1, 1.25, 1, 'word', [1, 2, [3, 4]]]
```

```
# [인덱싱] 0 인덱스 값을 가진 요소 값 출력
>>> print(myList[0])
[3, 4]
```

“Selena's Comment!”

앞서 배웠던 숫자나 문자열은 하나의 타입으로 이루어져 있었어요. 하지만 리스트는 특이하게 여러 가지 자료들을 모아서 만들 수 있어요. 굉장히 유용하겠죠?!
현업에서 리스트 잘 쓰면 능력자가 될 수 있습니다!! 명심!

4-3. 연산자

- '+' : 덧셈
- '*' : 반복
- len() : 길이 구하기

```
# list1 2번 반복
# list1과 list2 덧셈
# 연산 결과의 길이 구하기
>>> list1 = [1, 2, 3]
>>> list2 = [4, 5, 6]
>>> print(list1*2 + list2)
>>> print(len(list1*2 + list2))

[1, 2, 3, 1, 2, 3, 4, 5, 6]
9
```

4-4. 변경과 삭제

- 리스트는 값을 변경하거나 삭제할 수 있음

4-5. 관련 함수

- sort() : 리스트 정렬
- reverse() : 리스트 뒤집기
- append(x) : 리스트에 요소 추가
- extend(x) : 리스트 확장
- insert(a, b) : 리스트 요소 삽입, 리스트 a번째 위치에 b를 삽입
- remove(x) : 리스트 요소 제거
- pop() : 리스트 요소 끄집어내기
- count(x) : 리스트에 포함된 요소 x의 개수 세기

```
# myList 선언
>>> myList = [1, 5.2, 2, 6, 2]
>>> print(myList)

[1, 5.2, 2, 6, 2]
```

```
# sort() : 리스트의 요소를 순서대로 정렬하는 함수
>>> myList.sort()
>>> print(myList)

>>> [1, 2, 2, 5.2, 6]
```

```
# reverse() : 리스트 요소 뒤집는 함수
>>> myList.reverse()
>>> print(myList)

[6, 5.2, 2, 2, 1]
```

```
# append(x) : 리스트 맨 마지막에 추가하는 함수
# 한 개의 숫자를 리스트에 추가한 경우
>>> myList.append1.
>>> print(myList)

[6, 5.2, 2, 2, 1, 1]
```

5. 튜플 자료형(tuple)

- ✓ 리스트는 []로 생성하지만 튜플은 ()로 생성
- ✓ 리스트와 다른 점은 리스트는 그 값의 생성, 삭제, 수정이 가능하지만
- ✓ 튜플은 값을 추가, 삭제, 수정 할 수 없음
- ✓ 수정해서는 안 되는 값을 저장할 때 사용

5-1. 튜플 생성

- 괄호 있는 튜플
 - (데이터, 데이터, 데이터, ...) 형태로 생성
 - 한 개의 값을 가질 때는 마지막에 ",(쉼표)"를 써줘야 함

```
# 괄호 있는 튜플
# tuple1 : 한 개의 값을 가질 때는 마지막에 ", (쉼표)"를 써줘야 함
# float1 : 한 개의 요소지만 ", (쉼표)" 안 써줘서 튜플로 생성이 안 됨
>>> tuple1 = (2.3,)
>>> float1 = (2.3)
>>> print(tuple1)
>>> print(float1)
>>> print(type(tuple1))
>>> print(type(float1))

(2.3,)
2.3
<class 'tuple'>
<class 'float'>
```

- 괄호 없는 튜플
 - 괄호를 생략해도 튜플로 인식할 수 있음

```
# 괄호 없는 튜플
# tuple2 : 여러 개 요소인 튜플 생성
>>> tuple2 = 1,2,3,4,5
>>> print(tuple2)
>>> print(type(tuple2))

(1, 2, 3, 4, 5)
<class 'tuple'>
```

“Selena's Comment!”

리스트 자료형과 달리 튜플 자료형은 값을 추가, 삭제, 수정할 수 없기 때문에 제약이 많은 자료형이에요.
현업에서는 튜플을 수정해서는 안 될 값을 저장할 때 사용해요!

5-2. 인덱싱과 슬라이싱

- 리스트와 완전 동일
- 인덱싱
 - 문자열 내부의 문자 하나를 선택하는 연산자
- 슬라이싱
 - 문자열의 특정 범위를 선택할 때 사용하는 연산자

```
# 인덱싱 : 문자열 내부의 문자 하나를 선택하는 연산자
>>> tuple1 = (9,8,7,6)
>>> tuple1[0]
```

9

```
# 슬라이싱 : 문자열의 특정 범위를 선택할 때 사용하는 연산자
# [1:3] : 1 인덱스부터 3 인덱스 미만까지 범위
>>> tuple2 = (5,4,3,2)
>>> tuple2[1:3]

(4, 3)
```

5-3. 연산자

- '+': 덧셈
- '*': 반복
- len(): 길이 구하기

```
# tuple1 2번 반복
# tuple1과 tuple2 덧셈
# 연산 결과의 길이 구하기
>>> tuple1 = (1, 2, 3)
>>> tuple2 = (4, 5, 6)
>>> print(tuple1*2 + tuple2)
>>> print(len(tuple1*2 + tuple2))

(1, 2, 3, 1, 2, 3, 4, 5, 6)
9
```

5-4. 변경과 삭제

- 튜플을 값을 변경할 수 없고 삭제할 수 없음
- 오류 발생!

6. 딕셔너리 자료형(dictionary)

- 사람은 누구든지 '이름' = '셀레나', '취미' = '풋살' 등으로 구별할 수 있음
- 파이썬은 이러한 대응 관계를 나타내기 위해 딕셔너리 사용
- 딕셔너리 뜻 그대로 사전을 의미하며 key와 value를 한 쌍으로 갖는 자료형
- 'book'이라는 단어에 '책'이라는 뜻이 부합되듯이 key는 'book'이고 value는 '책'이다.
- 리스트가 '인덱스(index)' 기반으로 값을 저장하는 것이라면,
- 딕셔너리는 '키(key)' 기반으로 값을 저장하는 것

“Selena’s Comment!”

자료형들을 개별적으로만 생각하지 말고 연관 지어보세요.

예를 들면, 이전에 배웠던 리스트의 특징과 지금 배우는 딕셔너리는 어떤 차이점이 있는지 생각해 보는 거죠.

리스트는 인덱스 기반으로 값을 저장했었는데 딕셔너리는 키를 기반으로 값을 저장하구나! 하면서요.

복습할 겸 계속 반복학습을 해주세요~!

6-1. 딕셔너리 선언

- 괄호 있는 튜플
- 딕셔너리는 중괄호 {}로 선언하며, '키 : 값' 형태를 쉼표(,)로 연결하여 생성
- 키는 문자열, 숫자, 부울 등으로 선언
- 일반적으로 키는 문자열을 사용
- { key1 : value1, key2 : value2, ... key_n : value_n }
- 딕셔너리 요소에 접근하려면 대괄호[] 내부에 키를 입력

```
# 딕셔너리 선언
>>> myDictionary = {
    "name" : "Selena",
    "job" : "data scientist"
}
```

```
# 딕셔너리 값 출력
>>> print(myDictionary['name'])
>>> print(myDictionary['job'])
```

```
Selena
data scientist
```

6-2. 변경과 삭제

- 딕셔너리는 값을 변경하거나 삭제할 수 있음

```
# 딕셔너리 값 추가
>>> myDictionary['grade'] = [98, 100, 88]
>>> print(myDictionary)

{'name': 'Selena', 'job': 'data scientist', 'grade': [98, 100, 88]}
```

```
# 딕셔너리 값 수정
>>> myDictionary['grade'] = [100, 100, 90]
>>> print(myDictionary)

{'name': 'Selena', 'job': 'data scientist', 'grade': [100, 100, 90]}
```

```
# 딕셔너리 값 삭제
>>> del myDictionary['grade']
>>> print(myDictionary)

{'name': 'Selena', 'job': 'data scientist'}
```

6-3. 관련 함수

- '+' : 덧셈
- keys() : key 리스트 만들기
- values() : value 리스트 만들기
- items() : key, value 쌍 얻기
- get() : key로 value 얻기
- in() : 해당 key가 딕셔너리 안에 있는지 조사
- clear() : key:value 쌍 모두 지우기

```
# 딕셔너리 생성
>>> myDictionary = {
    "name" : "Selena",
    "job" : "data scientist",
    "grade" : [100, 90, 80]
}
>>> print(myDictionary)

{'name': 'Selena', 'job': 'data scientist', 'grade': [100, 90, 80]}
```

```
# keys() : key 리스트 만들기
>>> print(myDictionary.keys())

dict_keys(['name', 'job', 'grade'])
```

```
# values() : value 리스트 만들기
>>> print(myDictionary.values())

dict_values(['Selena', 'data scientist', [100, 90, 80]])
```

```
# items() : key, value 쌍 얻기
>>> print(myDictionary.items())

dict_items([('name', 'Selena'), ('job', 'data scientist'), ('grade', [100, 90, 80])])
```

```
# get() : key로 value 얻기
>>> print(myDictionary.get('grade'))
>>> print(myDictionary.get('hobby'))

[100, 90, 80]
None
```

```
# in() : 해당 key가 딕셔너리 안에 있는지 조사
>>> print('name' in myDictionary)
>>> print('hobby' in myDictionary)

True
False
```

```
# clear() : key:value 쌍 모두 지우기
>>> myDictionary.clear()
>>> print(myDictionary)

{}
```

7. 집합 자료형(set)

7-1. 집합 생성

- set 키워드를 통해 생성
 - set 괄호 안에 리스트 입력
 - set 괄호 안에 문자열 입력
- 중복을 허용하지 않음
- 비순서 자료형
- 순서를 사용하고 싶다면 리스트나
- 튜플로 변환하여 사용

```
# 리스트 입력하여 집합 생성
>>> set1 = set([10, 11, 12])
>>> print(set1)
>>> print(type(set1))

{10, 11, 12}
<class 'set'>
```

```
# 문자열 입력하여 집합 생성
# 중복을 허용하지 않기 때문에 e가 하나만 출력
# 비순서 자료형 : 인덱싱 지원 안 함
>>> set2 = set("selena")
>>> print(set2)
>>> print(type(set2))

{'s', 'a', 'e', 'l', 'n'}
<class 'set'>
```

7-2. 교집합, 합집합, 차집합

- 합집합 : |
- 교집합 : &
- 차집합 : -

```
# 집합 생성
>>> set1 = {1, 2, 3, 4, 5}
>>> set2 = {1, 3, 5, 7, 9}
```

```
# 합집합
>>> print(set1 | set2)

{1, 2, 3, 4, 5, 7, 9}
```

7-3. 관련 함수

- add() : 한 개의 값 추가
- update() : 여러 개의 값 추가
- remove() : 특정 값 제거

```
# 집합 생성
>>> set1 = {1, 2, 3, 4, 5}
```

```
# add() : 한 개의 값 추가
>>> set1.add(6)
>>> print(set1)

{1, 2, 3, 4, 5, 6}
```

8. 부울 자료형(bool)

- ✓ 참과 거짓을 나타내는 자료형
- ✓ 주로 조건문의 반환 값으로 사용

```
# 조건문의 반환 값
>>> "selena" == "selena"
```

```
True
```

```
# 조건문의 반환 값
>>> 10 < 11
```

```
True
```

8-1. 변하지 않는 참과 거짓

- 문자열, 리스트, 튜플, 딕셔너리 등의 값이 비어 있다면 False(거짓)
- 문자열, 리스트, 튜플, 딕셔너리 등의 값이 비어 있지 않다면 True(참)
- if문(조건문)으로 참, 거짓 판

STEP 3. 입력과 출력

“Selena's Comment!”

앞으로 우리는 사용자 입력과 출력에 대해 배울 거예요.

예를 들면, 블로그에 글을 적고 게시 버튼을 눌러야만(입력) 작성한 글이 블로그에 올라가게 돼요(출력).

예시와 같이 파이썬에서는 입력한 값을 어떤 변수에 대입하고 출력하고 싶을 때 어떻게 해야 하는지 알아볼게요.

01. 입력

1. 사용자 입력(input)

- ✓ 사용자로부터 데이터를 입력 받을 때
- ✓ input() 함수 사용

1-1. 사용자 입력 받기

- input() 함수 괄호 안에 입력한 내용을 '프롬프트 문자열'이라고 부름
- 사용자로부터 입력을 요구하는 내용 적기

```
# input() 함수 안에 사용자에게 요구하는 내용 적기
>>> input("이름을 적어주세요. : ")
이름을 적어주세요. : selena
'selena'
```

1-2. 변수 대입

- input() 함수의 결과 값을 변수에 대입
- input() 함수는 사용자가 무엇을 입력해도 결과는 무조건 문자열 자료형

```
# input() 함수의 결과 값을 name 이라는 변수에 대입
# name은 문자열로 입력을 받아서 데이터 타입은 문자열
>>> name = input("이름을 적어주세요. : ")
>>> print(type(name))

이름을 적어주세요. : selena
<class 'str'>
```

2. 숫자로 변환

- `input()` 함수의 결과는 항상 문자열 자료형
- 그렇기 때문에 입력받은 문자열을 숫자로 변환해야 숫자 연산에 사용 가능
- `int()` 함수를 이용하여 숫자(정수)로 변환
- `float()` 함수를 이용하여 숫자(실수)로 변환

```
# input() 함수의 결과는 항상 문자열 자료형
# int() 함수를 이용하여 숫자로 변환
# string1, string2 변수의 타입은 문자열 자료형
# int1, int2 변수의 타입은 숫자 자료형
>>> string1 = input("1 입력해주세요. : ")
>>> int1 = int(string1)

>>> string2 = input("2 입력해주세요. : ")
>>> int2 = int(string2)

# 문자열 덧셈이라서 12(숫자가 아닌 문자열) 출력
>>> print(string1 + string2)

# 숫자 덧셈이라서 1+2인 3(숫자) 출력
>>> print(int1 + int2)

1 입력해주세요. : 1
2 입력해주세요. : 2
12
3
```

3. 문자열로 변환

- `str()` 함수를 이용하여 문자열로 변환

“Selena’s Comment!”

현업에 있다 보면 파이썬 자료형 타입 때문에 오류가 나는 경우가 많아요.
그래서 `input()`로 출력된 결과는 항상 문자열이라는 걸 잊지 마세요.
이런 작은 지식들이 쌓여 실력이 됩니다!

02. 출력

1. 출력 (print)

1-1. 큰 따옴표와 더하기

- print() 함수는 자료형을 출력하는 함수
- 큰 따옴표(")로 둘러싸인 문자열은 더하기(+) 연산과 동일

```
# 큰 따옴표("")로 둘러싸인 문자열은 + 연산과 동일
# 큰 따옴표("")로 둘러싸인 문자열
# + 연산으로 사용한 문자열
>>> print("파이썬 " "프로그래밍 " "수업")
>>> print("파이썬 "+"프로그래밍 "+"수업")

파이썬 프로그래밍 수업
파이썬 프로그래밍 수업
```

1-2. 콤마

- 문자열 띄어쓰기는 콤마 사용

```
# 문자열 띄어쓰기는 콤마 사용
>>> print("파이썬", "프로그래밍", "수업")

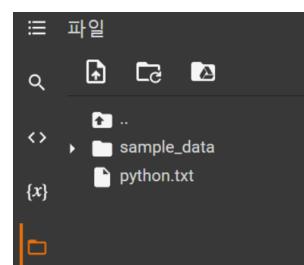
파이썬 프로그래밍 수업
```

03. 파일 입출력

1. 파일 열기(open)

- 파일을 열 때는 open() 함수 사용
- 파일 객체 = open(파일 경로, 파일 열기 모드)
- open() 함수의 첫 번째 매개변수에는 파일 경로(path)를 입력
- open() 함수의 두 번째 매개변수에는 파일 열기 모드(mode)를 지정
- 파일 열기 모드(mode) 종류
 - w : write 모드(쓰기 모드, 파일에 내용을 쓸 때 사용)
 - a : append 모드(추가 모드, 파일의 마지막에 새로운 내용을 추가할 때 사용)
 - r : read 모드(읽기 모드, 파일을 읽기만 할 때 사용)

```
# 파일 열기
# 왼쪽 파일 메뉴에 "python.txt" 생성된 걸 확인
# w : write 모드(쓰기 모드, 파일에 내용을 쓸 때 사용)
>>> myFile = open("python.txt", "w")
```



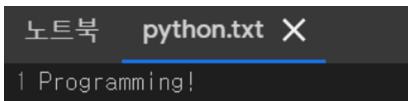
2. 파일 쓰기(write)

- 파일을 쓸 때는 write() 함수 사용
- 파일객체.write("문자열")
- write() 함수 안에 작성하고 싶은 문자열 입력

3. 파일 닫기(close)

- 파일을 닫을 때는 close() 함수 사용

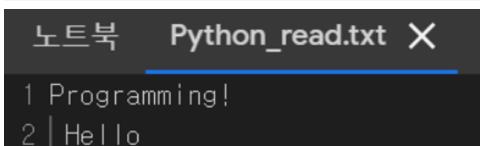
```
# 파일 열고 쓰고 닫기
>>> myFile = open("python.txt", "w")
>>> myFile.write("Programming!")
>>> myFile.close()
```



4. 파일 읽기(read)

- 파일을 읽기모드(r)로 열기
- readline() : 텍스트 한 줄씩 읽어서 반환
 - 한 줄씩 읽어 출력할 때 줄 끝에 \n 문자가 있다면 빈 줄도 같이 출력됨
- readlines() : 파일의 모든 줄을 읽어서 각각의 줄을 리스트로 반환
- strip() : 줄 바꿈(\n) 문자 제거

```
# 2줄의 내용을 가진 파일 열고 쓰고 닫기
>>> myFile = open("Python_read.txt", "w")
>>> myFile.write("Programming! \n Hello")
>>> myFile.close()
```



```
# readlines() : 파일의 모든 줄을 읽어서 각각의 줄을 리스트로 반환
>>> all_text = read_file.readlines()
>>> print(all_text)

['Programming! \n', 'Hello']
```

STEP 4. 조건문

“Selena’s Comment!”

수강생 여러분! 드디어 조건문이네요. 이제 찐 코딩의 세계가 시작됩니다!
현업에서 가장 많이 쓰는 걸 고르라고 한다면 조건문 아닐까요?!
Selena 강사와 함께 조건문에 대해 배우러 가봅시다~

01. if 조건문

1. if 문

- 조건이 참일 경우 실행
- 조건이 거짓일 경우 패스
- 들여쓰기 주의
- 콜론(:) 주의
- "Selena 수업을 들으면 파이썬을 정복할 수 있다."
- 위의 문장을 파이썬 조건문으로 표현

```
if 조건문 :
    조건이 참일 때 수행할 문장 1
```

```
# "Selena 수업을 들으면 파이썬을 정복할 수 있다."
# 위의 문장을 파이썬 조건문으로 표현
>>> selena_lecture = True

>>> if selena_lecture:
        print("파이썬을 정복할 수 있다.")

파이썬을 정복할 수 있다.
```

1-1. 들여쓰기

- if 문에 속한 모든 문장은 들여쓰기 필수

```
if 조건문 :
    조건이 참일 때 수행할 문장 1
    조건이 참일 때 수행할 문장 2
    조건이 참일 때 수행할 문장 3
```

1-2. 콜론(:)

- if 조건문 뒤에는 반드시 콜론(:)을 삽입
- 파이썬 문법 구조

2. else 문

- else 문은 if 문 뒤에 사용하며, if문 조건이 거짓일 때 실행되는 부분
- "Selena 수업을 들으면 파이썬을 정복할 수 있고
Selena 수업을 듣지 않으면 파이썬을 정복할 수 없다."
- 위의 문장을 파이썬 조건문으로 표현

```
if 조건문 :
    조건이 참일 때 수행할 문장 1
else :
    조건이 거짓일 때 수행할 문장 2
```

```
# "Selena 수업을 들으면 파이썬을 정복할 수 있고 Selena 수업을 듣지 않으면 파이썬을 정복할 수 없다."
# 위의 문장을 파이썬 조건문으로 표현
>>> selena_lecture = True

>>> if selena_lecture:
        print("파이썬을 정복할 수 있다.")
    else :
        print("파이썬을 정복할 수 없다.")
```

파이썬을 정복할 수 있다.

3. elif 문

- elif 문은 세 개 이상의 조건을 연결할 때 사용

```
if 조건문A :
    조건문A가 참일 때 수행할 문장 1
elif 조건문B :
    조건문B가 참일 때 수행할 문장 2
elif 조건문C :
    조건문C가 참일 때 수행할 문장 3
else :
    모든 조건이 거짓일 때 수행할 문장 4
```

```
# class를 공지하는 식
>>> score = "C"

>>> if score == "A" :
        print("A class")
    elif score == "B" :
        print("B class")
    elif score == "C" :
        print("C class")
    else :
        print("D class")

C class
```

4. in / not in 연산자

- 데이터 안에 찾고자 하는 것이 있는지 없는지 확인하는 연산자

```
# python 문자열에 k가 포함되어 있지 않다면 True, 포함 되어 있다면 False 출력
>>> if 'k' not in 'python':
        print(True)
    else:
        print(False)

True
```

5. pass 키워드

- 조건은 만족하지만 아무 일도 수행하고 싶지 않을 때 사용
- 프로그래밍의 전체 골격을 잡아놓고, 내부에 처리할 내용은 추후에 만들고자 할 때 사용

```
# 바구니에 딸기가 없다면 pass
>>> basket = ['사과', '바나나']

>>> if '딸기' not in basket:
    pass
```

“Selena's Comment!”

현업에서 저는 정확하게 조건을 주고 코딩을 하는 편이라서 최대한 조건을 elif로 걸어주고 나서 else를 해줘요. in/ not in 연산자와 pass 키워드는 정말 유용하니까 기억해 주세요!

6. 관계 연산자

- 두 개의 값을 비교하여 그 값을 참 또는 거짓으로 판별하는 연산자
 - ==, !=, <, >, <=, >=

7. 논리 연산자

- 논리 값을 판단해주는 연산자
 - not(논리 부정)
: 한 개의 값이 False면 결과 값으로 True 반환, True이면 결과 값으로 False 반환
 - and(논리 곱)
: 두 개의 값이 모두 True인 경우에만 결과 값으로 True 반환, 나머지는 모두 False 반환
 - or(논리 합)
: 두 개의 값이 모두 False인 경우에만 결과 값으로 False 반환, 나머지는 모두 True 반환

STEP 5. 반복문

“Selena's Comment!”

조건문과 양대 산맥을 이루는 것은 바로 반복문이죠!

반복 작업을 편리하게 하기 위해서 사용하는 것으로 현업에서 하루에 수십 번 사용하는 중요한 명령어예요

01. for 반복문

1. 기본 구조

- 특정 코드로 반복 작업을 하고 싶을 때 사용
- 리스트와 튜플, 그리고 문자열의 첫 번째 요소부터 마지막 요소까지 차례로 변수에 대입되어 '반복적으로 수행할 문장1' 수행

```
for 변수 in 리스트( 또는 튜플, 문자열 ) :
    반복적으로 수행할 문장1
```

```
for 반복자 in 반복할 수 있는 것 :
    반복적으로 수행할 문장1
```

```
# print("Selena 파이썬 프로그래밍") 코드를 5번 반복하고 싶을 때
# range() 함수를 사용하여 0부터 5미만까지 5번 반복
>>> for i in range5.:
    print("Selena 파이썬 프로그래밍")
```

```
Selena 파이썬 프로그래밍
```

2. 문자열 반복

- 문자열은 문자의 나열이며, 큰따옴표 혹은 작은따옴표로 입력
- 문자열을 이용한 for문은 문자 하나하나가 변수에 들어가며, 차례차례 반복

```
# myList 변수에 국가명인 'Korea' 문자열을 넣고 반복문으로 출력
>>> myList = 'Korea'
>>> for a in myList:
    print(a)
```

```
K
o
r
e
a
```

3. 리스트 반복

- 리스트는 여러 가지 자료를 저장할 수 있는 자료형
- 요소(element)는 리스트 안에 있는 각각의 내용을 의미
- 인덱스(index)는 리스트 안에 있는 값의 위치를 의미
- 리스트를 이용한 for문은 요소 하나하나가 변수에 들어가며, 차례차례 반복

```
# myList 변수에 국가명이 담긴 리스트를 넣고 반복문으로 출력
>>> myList = ['Korea', 'USA', 'Japan']

>>> for a in myList:
    print(a)

Korea
USA
Japan
```

4. 딕셔너리 반복

- 딕셔너리는 키를 기반으로 여러 자료를 저장하는 자료형
- 키(key)는 딕셔너리 안에서 값을 접근할 때 사용하는 것
- 값(value)은 딕셔너리 안에서 각각의 내용을 의미
- 딕셔너리를 이용한 for문은 딕셔너리 안에 있는 키(key)가 변수에 들어감

```
for 키 변수 in 딕셔너리 :
    반복을 수행할 문장1

# 딕셔너리 생성
>>> myDictionary = {
    "name" : "Selena", "job" :
    "data scientist"
}

# 딕셔너리 안에 있는 키(key)가 변수로 들어감
# key 값과 myDictionary[key] 내용을 반복문으로 출력
>>> for key in myDictionary:
    print(key, ":", myDictionary[key])

name : Selena
job : data scientist
```

5. 범위 반복

5-1. range() 함수

- range() 함수
 - range(A) : 매개변수로 숫자 한 개 입력
 - 0부터 A-1까지의 정수로 범위 생성
 - range(A, B) : 매개변수로 숫자 두 개 입력
 - A부터 B-1까지의 정수로 범위 생성
 - range(A, B, C) : 매개변수로 숫자 세 개 입력
 - A부터 B-1까지의 정수로 범위 생성하는데 앞 뒤 숫자가 C만큼의 차이를 가짐

- list() 함수
 - 범위를 리스트로 변경하면 범위 안에 어떤 값이 있는지 확인 가능

```
# range() 함수로 바로 출력하면 range(0, 5)로 출력됨
>>> print(range5.)
```

```
range(0, 5)
```

```
# list() 함수를 이용해 범위를 리스트로 변경하면 범위 내부에 어떤 값
이 있는지 확인 가능
```

```
# range() 함수의 매개변수 개수에 따른 결과 값 출력
>>> print(list(range5.))
>>> print(list(range(5, 10)))
>>> print(list(range(5, 10, 2)))
```

```
[0, 1, 2, 3, 4]
[5, 6, 7, 8, 9]
[5, 7, 9]
```

5-2. range() 반복

```
# range() 함수를 사용하여 5번 반복
# 0부터 5미만까지 반복
>>> for i in range5.:
print(i)

0
1
2
3
4
```

```
# 3부터 시작해서, 20 미만의 수를 4간격으로
>>> for i in range(3, 20, 4):
print(i)
```

```
3
7
11
15
19
```

02. while 반복문

1. 기본 구조

- 조건을 만족하는 한, 반복적으로 문장을 실행
- 부울(bool) 표현식이 참인 동안 문장을 계속 반복
- 부울 표현식이 거짓이 되면 반복문 종료
- 무한 반복이 되지 않도록 주의!

```
while 조건 :
    반복을 수행할 문장1
    반복을 수행할 문장2
    반복을 수행할 문장3
    ...
    ...
```

```
# print("Selena 파이썬 프로그래밍") 코드를 5번 반복하고 싶을 때
# range() 함수를 사용하여 0부터 5미만까지 5번 반복
>>> for i in range5.:
print("Selena 파이썬 프로그래밍")

Selena 파이썬 프로그래밍
```

2. break 키워드

- 반복문을 벗어날 때 사용하는 키워드
- 일반적으로 무한 반복문을 만들고, 내부의 반복을 벗어날 때 사용

```
# 반복문 조건은 'a가 3보다 클 때 반복을 시켜줘'
# a를 10으로 초기 값을 부여하여 반복문의 조건은 항상 참이 나옴
# 그래서 무한 반복문이 됨
# break 키워드를 통해 'count가 20보다 클 때 반복을 멈춰' 실행하여 무한 반복문을 빠져나옴
>>> a = 10
>>> count = 0

>>> while a > 3:
    count = count + 1
    a = a + 1
    if count > 20:
        break
    print("a = ", a, "\t count =", count)

>>> print("반복문 종료")

a = 11 count = 1
a = 12 count = 2
a = 13 count = 3
a = 14 count = 4
a = 15 count = 5
a = 16 count = 6
a = 17 count = 7
a = 18 count = 8
a = 19 count = 9
a = 20 count = 10
a = 21 count = 11
a = 22 count = 12
a = 23 count = 13
a = 24 count = 14
a = 25 count = 15
a = 26 count = 16
a = 27 count = 17
a = 28 count = 18
a = 29 count = 19
a = 30 count = 20
반복문 종료
```

3. continue 키워드

- 반복문의 현재 반복을 생략할 때 사용하는 키워드
- 현재 반복을 생략하고, 다음 반복으로 넘어갈 때 사용

```
# 반복문 조건은 'num이 10보다 작을 때 반복을 시켜줘'  
# num이 5 이하일 때 continue 키워드를 사용하여  
# 현재 반복을 생략하고 다음 반복으로 넘어가서 print가 안 됨  
# num이 5 초과일 때 print 됨  
>>> num = 0  
  
>>> while num < 10:  
    num = num + 1  
    if num <= 5:  
        continue  
    print(num)  
  
6  
7  
8  
9  
10
```

“Selena's Comment!”

반복문을 배워보았어요!

for 반복문은 조건식이 들어가 있기 때문에 구하고자 하는 값의 조건이 무엇인지 정확할 경우 사용하는 거고
while 반복문은 구하고자 하는 값의 정확한 조건을 모를 때 사용해요. 그 대신 무한 루프가 생기지 않도록
break를 걸어줘야겠죠?

아직 어렵다면 실습 코드를 변경해가면서 복습하며 자신의 것으로 만들어 주세요~!

복습만이 유일한 해결점 :)

STEP 6. 함수

“Selena’s Comment!”

찐 코딩 세계의 1단계인 조건문과 반복문을 배워보았어요.

2단계는 함수입니다.

잘 짠 함수 하나, 열 명령어 부럽지 않다!

함수는 입력 값에 따라 결과는 달라질 수 있지만 그 안에 든 로직은 계속 재사용할 수 있어요.

예를 들면, 강아지를 목욕 시키는 로봇이 있어요. 그렇다면 더러운 푸들(입력 1)을 로봇에 넣으면 목욕을 시켜서 깨끗한 푸들(출력 1)로 돌려주고 더러운 비글(입력 2)을 넣으면 깨끗한 비글(출력 2)로 돌려주죠.

이렇듯 목욕시킨다(함수의 로직)는 로봇의 능력은 어떤 강아지가 와도 동일하게 작동해요.

그래서 함수를 잘 짠다면 업무를 효율적으로 할 수 있어요. 그러면 함수를 배우러 가봅시다~!

01. 함수(function)

1. 기본 구조

- [입력 >> 함수 >> 출력] 구조
- 반복적으로 사용되는 기능을 함수로 사용
- 입력(들어가는 값)만 바뀌고 같은 코드가 반복되는 경우 특정 코드들을 모아두는 용도로 사용
- 함수는 def 키워드 사용하여 생성

```
def 함수이름(매개변수) :
    수행할 문장1
    수행할 문장2
    ...
    return 결과
```

- 함수 호출은 함수를 실행하는 행위를 의미

```
함수이름(인수1, 인수2)
```

```
# add 함수 정의
# 매개변수 x와 y를 입력으로 받기
# x + y 코드를 계산하여 출력
>>> def add(x, y):
    return x + y
```

```
# add 함수를 호출하여 인수로 3과 6 입력
# z값에 함수의 결과를 출력하여 저장
>>> z = add(3, 6)
>>> print(z)
```

9

2. 매개변수(parameter)

- ✓ 함수에 입력으로 전달된 값을 받는 변수

```
def 함수이름(매개변수1, 매개변수2) :
    수행할 문장1
    ...
    return 결과
```

2-1. 디폴트 매개변수

- 매개변수의 값이 입력되지 않으면 디폴트 값으로 자동 입력
- 매개변수가 여러 개인 경우 뒤의 매개변수부터 디폴트 값을 가질 수 있음

```
def 함수이름(매개변수 = 디폴트 값) :
    수행할 문장1
    return 결과
```

```
# add 함수 정의
# 매개변수 x와 y를 입력으로 받기
# y는 디폴트 매개변수로 입력이 없다면 디폴트 값으로 자동 입력
# x + y 코드를 계산하여 출력
>>> def add(x, y=3):
    return x + y
```

```
# add 함수를 호출하여 인수로 3과 6 입력
# z값에 함수의 결과를 출력하여 저장
# y 매개변수에 값을 입력하면 그대로 입력
>>> z = add(3, 6)
>>> print(z)

9
```

```
# add 함수를 호출하여 인수로 3만 입력
# z값에 함수의 결과를 출력하여 저장
# y 매개변수에 입력이 없다면 디폴트 값으로 자동 입력
>>> z = add3.
>>> print(z)

6
```

2-2. 가변 매개변수

```
def 함수이름(*매개변수):
    수행할 문장1
    return 결과
```

- 원하는 만큼의 인자를 받을 수 있는 함수를 가변 매개변수를 사용한다고 표현
- *매개변수 : 가변 매개변수는 매개변수 앞에 *를 붙임
- 생성된 가변 매개변수는 리스트처럼 사용
- 가변 매개변수의 제약
 - 가변 매개변수 뒤에는 일반 매개변수가 올 수 없음
 - 가변 매개변수는 하나만 사용할 수 있음

```
# values라는 가변 매개변수
# menu 함수를 통해 카페 메뉴 출력
# 가변 매개변수는 리스트처럼 사용
>>> def menu(*values):
    print("Selena 카페의 메뉴는? ")

    for value in values:
        print(value)

# 정의한 menu 함수 호출
# 인수로 4가지 메뉴 입력
>>> menu('코코아', '캐모마일', '얼그레이', '에이드')

Selena 카페의 메뉴는?
코코아
캐모마일
얼그레이
에이드
```

2-3. 키워드 매개변수

```
def 함수(매개변수A, 매개변수B, 매개변수C):
    수행할 문장1
    return 결과
```

- 매개변수의 이름을 지정해서 입력하여 함수 호출
- 키워드 매개변수를 넘겨줄 때 매개변수의 순서와 상관없이 나열 가능

```
함수(매개변수B = 2, 매개변수C = 1, 매개변수A = 3)
```

```
# add 함수 정의
# 매개변수 x와 y, 그리고 z는 디폴트 매개변수
# x + y + z 코드를 계산하여 출력
>>> def add(x=1, y=2, z=3):
    return x + y + z
```

```
# 키워드 매개변수를 사용하여 add 함수 호출
# 키워드 매개변수를 넘겨줄 때 매개변수의 순서와 상관없이 나열 가능
# 키워드 매개변수는 x, y, z
>>> add(y=5, z=5, x=1)
```

11

3. 인수(arguments)

- 함수를 호출할 때 전달하는 입력 값

```
함수(인수1, 인수2)
```

```
# add 함수 정의
# 매개변수 x와 y를 입력으로 받기
# x + y 코드를 계산하여 출력
>>> def add(x, y):
    return x + y
```

```
# add 함수를 호출하여 인수로 3과 6 입력
# 인수는 함수를 호출할 때 전달하는 입력 값
>>> add(3, 6)

9
```

4. 리턴(return)

- 함수를 호출한 곳으로 결과 값 반환

```
def 함수이름(매개변수)
    실행문
    return 되돌려 주는 결과 값

함수(인수1, 인수2)
```

4-1. 한 개의 리턴 값

```
# 한 개의 리턴 값
# returnTest 함수 정의하고 함수를 호출하여 return 값 확인
>>> def returnTest() :
    return "되돌려 주는 결과 값"

>>> value = returnTest()
>>> print(value)

되돌려 주는 결과 값
```

4-2. 두 개의 리턴 값

```
# 두 개의 리턴 값
# returnTest 함수 정의하고 함수를 호출하여 return 값 확인
# 반환되는 두 개의 값을 차례대로 value1, value2에 저장하여 출력
>>> def returnTest() :
    a = 3
    b = 4
    return a, b

>>> value1, value2 = returnTest()
>>> print(value1)
>>> print(value2)

3
4
```

STEP 7. 클래스

“Selena's Comment!”

클래스 내용은 어려워서 꼭 여러 번 직접 실습을 해보시면서 익숙해지세요.

강의에서는 곰돌이 젤리를 예로 들어서 설명하고 있어요. 강의를 들어가기 앞서, 두 가지를 기억해 주세요.

첫 번째는 ‘클래스’인 곰돌이 젤리 틀과 두 번째는 ‘객체’인 곰돌이 젤리 틀에 의해서 만들어진 곰돌이 젤리예요.

젤리 공장에서 곰돌이 젤리 틀로 노란색 곰돌이 젤리와 빨간색 곰돌이 젤리를 생산해냈어요.

생산된 두 가지 젤리는 각각 색도 다르고 서로 영향을 주지 않아요.

서로 영향을 주지 않는다는 건 제가 노란색 곰돌이 젤리의 다리 부분을 먹었다고 해서 빨간색 곰돌이 젤리의 다리 부분이 사라지진 않죠! 조금 잔인하지만 이해되시죠?!

간단하게 파악해보았고 이제 제대로 배우러 가보겠습니다!

01. 클래스

1. 객체 지향 프로그래밍

- 객체 지향 프로그래밍(object oriented programming)
 - 프로그램 설계 방법론
 - 프로그램을 여러 개의 독립적인 단위인 '객체'라는 기본 단위로 나누고
 - '객체'들의 상호작용을 통해 프로그램을 설계하고 개발하는 방식
- OOP의 장점
 - 코드 재사용 용이
 - 유지보수 용이
 - 대형 프로젝트 적합
- OOP의 단점
 - 실행 속도가 상대적으로 느림
 - 객체가 많으면 프로그램 용량 커짐
 - 설계시 많은 시간 소요

2. 클래스와 객체

- 클래스(class)
 - 동일한 무언가를 계속 만들어 낼 수 있는 설계 도면을 의미
 - 곰돌이 젤리를 만드는 틀

- 객체(object)
 - 클래스로 만들어진 모든 것을 의미
 - 곰돌이 젤리 틀에 의해서 만들어진 젤리
- 클래스로 만든 객체의 특징
 - 객체마다 고유한 성질을 가짐
 - 동일한 클래스로 만든 객체들은 서로 전혀 영향을 주지 않음
 - 곰돌이 젤리 틀로 만든 젤리들은 서로 다른 맛을 첨가하여 빨간색과 노란색, 그리고 초록색 젤리가 존재
 - 젤리들은 각자의 고유한 맛을 가짐
 - Selena가 한 개의 빨간색 젤리를 먹었다고 해서 다른 젤리들에겐 전혀 영향을 주지 않음

3. 클래스 선언

- class 키워드를 사용하여 클래스 선언

```
class 클래스 이름 :
    클래스 내용
```

```
# 곰돌이 젤리 클래스 생성
>>> class TeddyBearJelly:
        pass
```

4. 객체와 인스턴스

- 객체(object)
 - 'red_jelly = TeddyBearJelly()'로 만든 red_jelly는 객체
 - red_jelly는 객체라는 표현이 적절
- 인스턴스(instance)
 - red_jelly는 TeddyBearJelly의 인스턴스
 - 인스턴스는 특정 객체(red_jelly)가 어떤 클래스(TeddyBearJelly)의 객체인지를 관계 위주로 설명할 때 사용
 - red_jelly는 TeddyBearJelly의 인스턴스라는 표현이 적절
- 인스턴스 생성
 - 클래스 기반으로 만들어진 객체를 인스턴스라고 부름
 - 클래스 인스턴스를 생성하려면 클래스 이름 뒤에 ()괄호를 적으면 됨

```
인스턴트 이름 = 클래스 이름( )
```

```
# 빨간색, 노란색, 초록색 젤리 객체 생성
# 각 jelly들은 TeddyBearJelly의 인스턴스
>>> red_jelly = TeddyBearJelly()
>>> yellow_jelly = TeddyBearJelly()
>>> green_jelly = TeddyBearJelly()
```

“Selena’s Comment!”

객체와 인스턴스는 달라요.

헷갈리시는 분들이 많으세요. 꼭 확실하게 이해하고 넘어가 주세요~!

5. 메소드

- 메소드(method)
 - 클래스가 가지고 있는 함수를 의미

```
class 클래스 이름 :
    def 메소드 이름(self, 추가적인 매개변수) :
        pass
```

- 클래스 내부의 함수의 첫 번째 매개변수는 반드시 self 입력
- 객체를 호출할 때 호출한 객체 자신이 전달되기 때문에 self 사용

“Selena’s Comment!”

앞 STEP에서 함수를 배웠죠?

메소드는 클래스 안에서 사용되는 함수를 가리키는 말이에요.
기억해주세요~!

```
# 곰돌이 젤리 클래스 선언
# 'set_info' 함수 선언 : 젤리 생성시 필요한 속성 입력
# 'print_info' 함수 선언 : 젤리 속성 출력
>>> class TeddyBearJelly :
    def set_info(self, color, taste):
        self.color = color
        self.taste = taste

    def print_info(self):
        print("====")
        print("Color : ", self.color)
        print("Taste : ", self.taste)
        print("====")
```

```
# 젤리 객체 선언
>>> yellow_jelly = TeddyBearJelly()
```

```
# 젤리 객체의 필요한 속성 입력
# 메소드 사용
>>> yellow_jelly.set_info("yellow", "lemon")
```

```
# yellow_jelly의 속성에 접근하는 방법
>>> print(yellow_jelly.color)
>>> print(yellow_jelly.taste)

yellow
lemon
```

```
# 젤리 객체 내용 출력
# 메소드 사용
>>> yellow_jelly.print_info()

=====
Color : yellow
Taste : lemon
=====
```

6. 생성자

- 메소드(method)
 - 클래스가 가지고 있는 함수를 의미
 - 클래스 내부의 함수의 첫 번째 매개변수는 반드시 self 입력
 - 객체를 호출할 때 호출한 객체 자신이 전달되기 때문에 self 사용
- 생성자(constructor)
 - 클래스로부터 인스턴스가 생성될 때 자동으로 실행되는 함수
 - 클래스를 실행하면 가장 먼저 생성자인 '__init__' 함수가 호출
 - '__init__' 함수는 인스턴스의 생성과 동시에 필요한 정보를 입력 받도록 구현하는 역할

```
class 클래스 이름 :
    def __init__(self, 추가적인 매개변수) :
        pass
```

- 클래스 내부의 함수의 첫 번째 매개변수는 반드시 self 입력
- 객체를 호출할 때 호출한 객체 자신이 전달되기 때문에 self 사용

```
# 메소드는 클래스 내부의 함수를 의미
# '5. 메소드'에서 구현했던 'TeddyBearJelly' 클래스의
# 'set_info()' 함수가 데이터를 입력받는 역할 수행함
# 'set_info' 메소드를 '__init__' 메소드로 이름 변경

# 곰돌이 젤리 클래스 선언
# 생성자 선언
>>> class TeddyBearJelly :
    def __init__(self, color, taste):
        self.color = color
        self.taste = taste
```

```
# 젤리 리스트 선언
>>> jelly = [
    TeddyBearJelly("red", "strawberry"),
    TeddyBearJelly("yellow", "lemon"),
    TeddyBearJelly("green", "apple")
]
```

```
# jelly 인스턴스의 속성에 접근하는 방법
>>> print(jelly[0].color)
>>> print(jelly[0].taste)

red
strawberry
```

7. 상속

- 상속(inheritance)
 - '물려 받다'라는 뜻
 - 새로운 클래스를 만들 때 기존에 있던 클래스의 기능을 물려 받을 수 있음
 - 상위 클래스를 부모 클래스(parent class, super class) 표현
 - 하위 클래스를 자식 클래스(child class, sub class) 표현
- 상속을 하는 이유
 - 기존 클래스를 변경하지 않고 기능을 추가하거나 기존 기능을 변경할 때 사용
 - 기존 클래스가 라이브러리 형태로 제공되거나 수정이 허용되지 않는 상황에 사용

7-1. 상속 선언

- 자식 클래스를 선언할 때 소괄호로 부모 클래스를 포함
- 그러면 자식 클래스에서는 부모 클래스의 속성과 메소드는 따로 기재하지 않아도 자동 포함

7-2. 메소드 오버라이딩

- 부모 클래스의 메소드를 자식 클래스에서 재정의 하는 것

1) 일반적인 메소드 오버라이딩

- 자식 클래스에서 생성된 객체의 메소드를 부르면 부모 클래스의 메소드는 무시

2) 부모 메소드 호출

- 부모 클래스의 메소드도 수행하고 자식 클래스의 메소드도 함께 수행하고 싶을 때 사용
- super() 키워드 사용하여 자식 클래스 내에서 부모 클래스 호출 가능

```
# 부모 클래스 선언
>>> class Jelly :
    def __init__(self, color, taste):
        self.color = color
        self.taste = taste

    def print_info(self):
        print("부모 클래스입니다.")
        print("젤리")

# 자식 클래스 선언
# 자식 클래스를 선언할 때 소괄호로 부모 클래스를 포함
# 일반적인 메소드 오버라이딩
# 자식 클래스에서 생성된 메소드로 부모 클래스의 메소드 무시
>>> class TeddyBearJelly(Jelly):
    def print_info(self):
        print("자식 클래스입니다.")
        print("곰돌이 젤리")
        print("Taste : ", self.taste)
        print("Color : ", self.color)
```

```
# 자식 클래스 선언
# 자식 클래스를 선언할 때 소괄호로 부모클래스를 포함
# 부모 메소드 호출
# super() 키워드를 사용하여 자식 클래스에서 부모 클래스 호출 가능
>>> class HeartJelly(Jelly):
    def print_info(self):
        super().print_info()
        print("자식 클래스입니다.")
        print("하트 젤리")
        print("Taste : ", self.taste)
        print("Color : ", self.color)
```

```
# 부모 클래스로 객체 생성
# 부모 클래스의 print_info() 함수
>>> red_jelly = Jelly("red", "strawberry")
>>> red_jelly.print_info()
```

부모 클래스입니다.
젤리

```
# 자식 클래스로 객체 생성
# 자식 클래스의 print_info() 함수
# 자식 클래스에서 생성된 메소드로 부모 클래스의 메소드 무시
>>> yellow_jelly = TeddyBearJelly("yellow", "lemon")
>>> yellow_jelly.print_info()
```

자식 클래스입니다.
곰돌이 젤리
Taste : lemon
Color : yellow

```
# 자식 클래스로 객체 생성
# 자식 클래스의 print_info() 함수
# super() 키워드를 사용하여 자식 클래스에서 부모 클래스 호출 가능
>>> green_jelly = HeartJelly("green", "apple")
>>> green_jelly.print_info()
```

부모 클래스입니다.
젤리
자식 클래스입니다.
하트 젤리
Taste : apple
Color : green

7-3. 다중 상속

- 자식 클래스를 선언할 때 소괄호로 원하는 부모클래스를 모두 포함
- 다중 상속의 개수 제한 없음

```
class 부모 클래스1 :
    pass
class 부모 클래스2 :
    pass
class 자식 클래스 (부모 클래스1, 부모 클래스2) :
    pass
```

STEP 8. 모듈

01. 내부 모듈

1. 내부 모듈

- 모듈은 여러 변수와 함수를 가지고 있는 집합
- 내부 모듈은 파이썬에 기본적으로 내장되어 있는 모듈
- 다양한 내부 모듈 존재 : [파이썬 공식 홈페이지](#) 참고
 - 숫자와 수학 모듈
 - numbers — 숫자 추상 베이스 클래스
 - math — 수학 함수
 - cmath — 복소수를 위한 수학 함수
 - decimal — 십진 고정 소수점 및 부동 소수점 산술
 - fractions — 유리수
 - random — 의사 난수 생성
 - statistics — 수학 통계 함수
 - 데이터형
 - datetime — 기본 날짜와 시간 형
 - 파이썬 실행시간 서비스
 - sys — 시스템 특정 파라미터와 함수
 - sysconfig — 파이썬의 구성 정보에 접근하기
 - 일반 운영 체제 서비스
 - os — 기타 운영 체제 인터페이스
 - io — 스트림 작업을 위한 핵심 도구
 - time — 시간 액세스와 변환
 - 데이터 지속성
 - pickle — 파이썬 객체 직렬화
 - copyreg — pickle 지원 함수 등록

2. import 구문

- import는 모듈 전체를 가져올 때 사용
- 모듈을 사용할 때는 모듈 뒤에 도트(.) 입력하고 원하는 변수나 함수를 입력

```
import 모듈 이름
```

```
# 파이썬 math 모듈은 수학 함수가 담겨있음
```

```
>>> import math
```

```
>>> print(math.floor(1.2))
>>> print(math.ceil(1.2))
```

```
1
2
```

3. from 구문

- from은 모듈 내에서 필요한 것만 골라서 가져올 때 사용
- 모듈에는 많은 변수와 함수가 있기 때문에 활용하고 싶은 것만 선택하여 사용
- 여러 개의 가져오고 싶은 변수 또는 함수 입력 가능

```
from 모듈 이름 import 가져오고 싶은 변수 또는 함수
```

- 모듈 모두를 가져오고 싶을 때는 * 기호 사용

```
from 모듈 이름 import *
```

- 모듈을 사용할 때는 원하는 변수나 함수만 입력

```
# floor(내림), ceil(올림) 함수만 선택
# 모듈을 사용할 때는 원하는 변수나 함수만 입력
>>> from math import floor, ceil

>>> print(floor(1.2))
>>> print(ceil(1.2))

1
2
```

“Selena’s Comment!”

현업에서 import로 시작해서 모듈을 불러오기도 하고 from을 사용해서 불러오기도 해요.
헷갈릴 수 있는 내용이니 정확하게 차이점을 파악해 주세요~!

4. as 구문

- as는 모듈의 별칭을 붙일 때 사용
- 모듈을 가져올 때 이름 충돌하는 경우 발생할 때 사용
- 모듈의 이름이 너무 길어서 줄이고 싶을 때 사용

```
from 모듈 as 모듈의 별칭
```

```
# 모듈을 사용할 때는 모듈 별칭 뒤에 도트(.) 입력하고 원하는 변수나 함수를 입력
>>> import math as m

>>> print(m.floor(1.2))
>>> print(m.ceil(1.2))
```

02. 외부 모듈

1. 외부 모듈

- 외부 모듈은 파이썬에서 기본적으로 제공해 주는 것이 아닌 외부 사람들이 만들어서 배포한 모듈
- 외부 모듈 종류 (Selena 파이썬 라이브러리 활용 강의에서 배우는 모듈)
 - NumPy : 수치 계산
 - Pandas : 데이터 처리
 - Matplotlib : 데이터 시각화
 - Seaborn : 데이터 시각화
 - BeautifulSoup : 웹 데이터 수집
 - Scikit-Learn : 머신러닝

“Selena’s Comment!”

외부 모듈에서는 시각화 라이브러리인 seaborn을 간단하게 다뤄볼 거예요.
제가 현업에서 자주 사용하는 모듈을 간단하게 준비해보았어요!

2. seaborn 모듈 실습

- seaborn : 데이터 시각화 라이브러리로 고급 인터페이스 제공
- seaborn 모듈 실습 순서
 1. 모듈 설치하기
 2. 모듈 불러오기
 3. 데이터 불러오기
 4. 데이터 파악
 5. 데이터 시각화

2-1. 모듈 설치하기

- 패키지가 없는 경우, 설치 명령어 : pip install seaborn

2-2. 모듈 불러오기

- import 구문을 통해 모듈 불러오기
- as 구문으로 별칭 지정하기

```
# seaborn 불러와서 별칭 sns로 사용
>>> import seaborn as sns
```

2-3. 데이터 불러오기

- seaborn 라이브러리에서 제공하는 titanic 데이터 불러오기
- seaborn의 load_dataset() 함수를 이용

```
# seaborn 불러와서 별칭 sns로 사용
>>> import seaborn as sns
```

2-4. 데이터 파악

- head() 함수 : titanic 데이터 상단 5개의 행 확인

```
# titanic 데이터 내용 확인
# .head() : 데이터의 상단 5개 행 출력
>>> titanic.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

- info() 함수 : 행과 열의 크기, 컬럼명, 결측치, 데이터 타입 확인

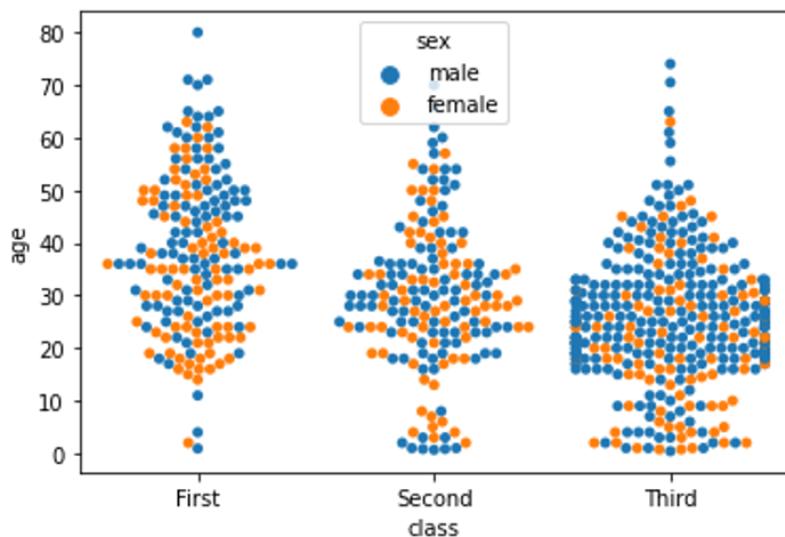
```
# .info() :데이터에 대한 전반적인 정보 제공
# 행과 열의 크기, 컬럼명, 컬럼별 결측치, 컬럼별 데이터 타입
>>> print(titanic.info())
```

<class 'pandas.core.frame.DataFrame'>	6 fare 891 non-null float64
RангIndex: 891 entries, 0 to 890	7 embarked 889 non-null object
Data columns (total 15 columns):	8 class 891 non-null category
# Column Non-Null Count Dtype	9 who 891 non-null object
--- -----	10 adult_male 891 non-null bool
0 survived 891 non-null int64	11 deck 203 non-null category
1 pclass 891 non-null int64	12 embark_town 889 non-null object
2 sex 891 non-null object	13 alive 891 non-null object
3 age 714 non-null float64	14 alone 891 non-null bool
4 sibsp 891 non-null int64	
5 parch 891 non-null int64	

2-5. 데이터 시각화

- swarmplot() 함수: 데이터의 분산까지 고려하여 데이터 포인트가 서로 중복되지 않도록 시각화. 즉, 데이터가 퍼져 있는 정도를 입체적으로 파악 가능
- swarmplot() 함수의 매개변수
 - x축 변수
 - y축 변수
 - 데이터 셋
 - hue: 특정 열 데이터로 색상을 구분하여 출력

```
# 이산형 변수의 분포 - 데이터 분산 고려 (중복 X)
# x축 변수, y축 변수, 데이터 셋, 성별로 색상 구분
>>> sns.swarmplot(x='class', y='age', data=titanic, hue='sex')
```



“Selena's Comment!”

데이터 분석과 시각화 과정에서 자주 들으실 말이 있어요!

저는 코딩을 할 때 효율을 중요시해서 결과를 내더라도 한 가지가 아닌 여러 가지를 나타내는 걸 좋아한다는 말이요.

방금 배워본 swarmplot 함수가 그런 효율적인 함수 중에 하나인데요.

클래스 정보와 나이 정보를 나타내는 동시에 성별 분포와 나이별 분포까지 파악할 수 있어요.

현업에서 사용하면 아주 좋겠죠?!

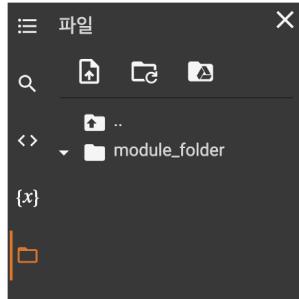
파이썬 기초를 다 배우고 나면 현업 데이터 사이언티스트인 Selena의 노하우를 데이터 분석, 시각화 STEP에서 배우실 수 있으십니다! 그때까지 기본기를 더 탄탄하게 다져봅시다! 조금만 더 화이팅해요~!

03. 모듈과 패키지

1. 모듈 생성

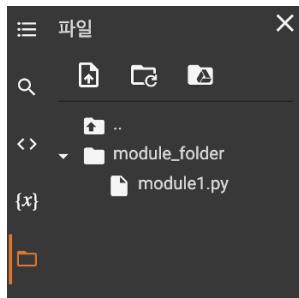
1-1. 폴더 만들기

- 구글 코랩의 왼쪽에서 파일 이모티콘 눌러서 새폴더 만들기
- 'module_folder' 이름의 새폴더 생성



1-2. 모듈.py 파일 만들기

- module_folder 폴더 안에 새파일 만들기
- 'module1.py' 이름의 새파일 생성

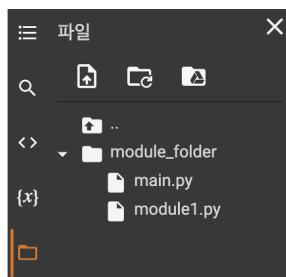


- 'module1.py' 파일 안에 코드 추가
 - add() 함수 구현

```
def add(x, y, z):
    return x + y + z
```

1-3. 메인.py 파일 만들기

- module_folder 폴더 안에 새파일 만들기
- 'main.py' 이름의 새파일 생성



- 'main.py' 파일 안에 코드 추가
 - 'module1' 모듈 불러오기
 - result 변수에 module1에서 구현한 add 함수 불러오기
 - result 출력

```
import module1
result = module1.add(1, 2, 3)
print(result)
```

1-4. 구글 코랩 노트북에서 메인.py 파일 실행

- ! python 키워드 사용
- .py 파일이 있는 경로 입력

```
# main.py 파일 실행
# 1 + 2 + 3인 6이 출력
! python /content/module_folder/main.py
```

2. 패키지

- 패키지는 모듈이 모여서 구조를 이룬 것
- 도트(.)를 사용하여 파이썬 모듈을 디렉터리 구조로 관리할 수 있게 해줌
 - 모듈 이름이 A.b인 경우, A는 패키지이고 b는 A패키지의 b모듈 의미

STEP 9. 예외 처리

“Selena’s Comment!”

프로그래밍을 하다 보면 수많은 오류를 만나게 돼요.
 오류를 통해 프로그램을 수정할 수 있지만 가끔은 이런 오류를 무시하고 싶을 때가 있어요.
 이를 위해 파이썬에서는 try, except 사용하여 예외 처리를 해줘요.

01. 오류

1. 구문 오류

- 프로그램 실행 전에 발생하는 오류
- 조건문과 반복문의 들여쓰기, 괄호의 개수 문제 등으로 발생
- 해당 오류는 프로그램이 실행조차 되지 않기 때문에 예외 처리 방법으로 해결을 할 수 없으며
- 문법적으로 문제가 발생한 부분을 수정 해줘야 함

```
# 문자열 닫힘 따옴표 문제로 오류 발생
# SyntaxError는 구문에 문제가 있어서 프로그램이 실행조차 안 된 경우
>>> print("Selena 파이썬 프로그래밍")

File "<ipython-input-1-b6535199254d>", line 3
    print("Selena 파이썬 프로그래밍")
                  ^
SyntaxError: EOL while scanning string literal
```

```
# 문자열 닫힘 따옴표 문제를 해결하여 오류 해결
# 해당 오류는 해결을 해줘야만 프로그램을 실행할 수 있음
>>> print("Selena 파이썬 프로그래밍")

Selena 파이썬 프로그래밍
```

2. 예외(런타임 오류)

- 프로그램 실행 중에 발생하는 오류
- 예외 처리를 통해 문제를 해결할 수 있음

```
# myList 변수에 리스트 선언 -> 프로그램 실행됨
# myList[7]은 myList에서 얻을 수 없는 값으로 오류 발생
>>> myList = [1, 2, 3, 4]
>>> myList[7]

-----
IndexError                                                 Traceback (most recent call last)
<ipython-input-3-06f92ceab1ed> in <module>()
      3 myList = [1, 2, 3, 4]
      4
----> 5 myList[7]

IndexError: list index out of range
```

```
# myList 변수에 리스트 선언 -> 프로그램 실행됨
# myList[3]은 myList에서 얻을 수 있는 값으로 수정하여 오류 해결
# 또는 예외 처리를 통해 문제를 해결할 수 있음!
>>> myList = [1, 2, 3, 4]
>>> myList[3]

4
```

02. 예외 처리 기법

1. try - except 구문

- try 문 안에는 예외가 발생할 가능성이 있는 모든 코드 삽입
- except 문 안에는 예외가 발생했을 때 실행할 모든 코드 삽입
- except 문 안에 pass 키워드를 넣으면 아무것도 출력하지 않고 강제 종료도 막을 수 있음

```
try :
    예외가 발생할 가능성이 있는 코드
except :
    예외가 발생했을 때 실행할 코드
```

```
# try 문 안에는 예외가 발생할 가능성이 있는 모든 코드 삽입
# except 문 안에는 예외가 발생했을 때 실행할 모든 코드 삽입
# 프로그램이 강제로 종료되는 일 없이 예외 처리를 하고 정상적으로 종료
>>> try :
    myList = [1, 2, 3, 4]
    myList[7]
>>> except :
    print("예외가 발생하였습니다.")

예외가 발생하였습니다.
```

```
# try 문 안에는 예외가 발생할 가능성이 있는 모든 코드 삽입
# except 문 안에는 예외가 발생했을 때 실행할 모든 코드 삽입
# except 문 안에 pass 키워드를 넣으면 아무것도 출력하지 않고 강제 종료도 막을 수 있음
>>> try :
    myList = [1, 2, 3, 4]
    myList[7]
>>> except :
    pass
```

2. try - except - else 구문

- try 문 안에는 예외가 발생할 가능성이 있는 모든 코드 삽입
- except 문 안에는 예외가 발생했을 때 실행할 모든 코드 삽입
- else 문 안에는 예외가 발생하지 않았을 때 실행할 코드 삽입

```
try :
    예외가 발생할 가능성이 있는 코드
except :
    예외가 발생했을 때 실행할 코드
else :
    예외가 발생하지 않았을 때 실행할 코드
```

```
# try 문 안에는 예외가 발생할 가능성이 있는 모든 코드 삽입
# except 문 안에는 예외가 발생했을 때 실행할 모든 코드 삽입
# else 문 안에는 예외가 발생하지 않았을 때 실행할 코드 삽입
# 문자 입력 시 예외 발생
>>> try :
    Number = int(input("숫자 입력 : "))
>>> except :
    print("예외가 발생하였습니다.")
>>> else :
    print("입력한 숫자는 ", Number, "입니다.")
```

숫자 입력 : selena
예외가 발생하였습니다.

```
# try 문 안에는 예외가 발생할 가능성이 있는 모든 코드 삽입
# except 문 안에는 예외가 발생했을 때 실행할 모든 코드 삽입
# else 문 안에는 예외가 발생하지 않았을 때 실행할 코드 삽입
# 숫자 입력시 프로그램 정상 작동
>>> try :
    Number = int(input("숫자 입력 : "))
>>> except :
    print("예외가 발생하였습니다.")
>>> else :
    print("입력한 숫자는 ", Number, "입니다.")
```

숫자 입력 : 7
입력한 숫자는 7 입니다.

3. try - except - finally 구문

- try 문 안에는 예외가 발생할 가능성이 있는 모든 코드 삽입
- except 문 안에는 예외가 발생했을 때 실행할 모든 코드 삽입

```
try :
    예외가 발생할 가능성이 있는 코드
except :
    예외가 발생했을 때 실행할 코드
finally :
```

```
# try 문 안에는 예외가 발생할 가능성이 있는 모든 코드 삽입
# except 문 안에는 예외가 발생했을 때 실행할 모든 코드 삽입
# finally 문 안에는 예외 발생 여부와 상관없이 무조건 실행할 코드 삽입
# 문자 입력 시 예외 발생
>>> try :
    Number = int(input("숫자 입력 : "))
>>> except :
    print("예외가 발생하였습니다.")
>>> finally :
    print("finally 문은 무조건 실행하는 코드")
```

숫자 입력 : selena
예외가 발생하였습니다.
finally 문은 무조건 실행하는 코드

- finally 문 안에는 예외 발생 여부와 상관없이 무조건 실행할 코드 삽입

```
# try 문 안에는 예외가 발생할 가능성이 있는 모든 코드 삽입
# except 문 안에는 예외가 발생했을 때 실행할 모든 코드 삽입
# finally 문 안에는 예외 발생 여부와 상관없이 무조건 실행할 코드 삽입
# 숫자 입력시 프로그램 정상 작동
>>> try :
    Number = int(input("숫자 입력 : "))
>>> except :
    print("예외가 발생하였습니다.")
>>> finally :
    print("finally 문은 무조건 실행하는 코드")

숫자 입력 : 7
finally 문은 무조건 실행하는 코드
```

4. try - except - else - finally 구문

- try 문 안에는 예외가 발생할 가능성이 있는 모든 코드 삽입
- except 문 안에는 예외가 발생했을 때 실행할 모든 코드 삽입
- else 문 안에는 예외가 발생하지 않았을 때 실행할 코드 삽입
- finally 문 안에는 예외 발생 여부와 상관없이 무조건 실행할 코드 삽입

```
try :
    예외가 발생할 가능성이 있는 코드
except :
    예외가 발생했을 때 실행할 코드
else :
    예외가 발생하지 않았을 때 실행할 코드
finally :
    예외 발생 여부와 상관없이 무조건 실행할 코드
```

```
# try 문 안에는 예외가 발생할 가능성이 있는 모든 코드 삽입
# except 문 안에는 예외가 발생했을 때 실행할 모든 코드 삽입
# else 문 안에는 예외가 발생하지 않았을 때 실행할 코드 삽입
# finally 문 안에는 예외 발생 여부와 상관없이 무조건 실행할 코드 삽입
# 문자 입력 시 예외 발생
>>> try :
    Number = int(input("숫자 입력 : "))
>>> except :
    print("예외가 발생하였습니다.")
>>> else :
    print("입력한 숫자는 ", Number, "입니다.")
>>> finally :
    print("finally 문은 무조건 실행하는 코드")

숫자 입력 : selena
예외가 발생하였습니다.
finally 문은 무조건 실행하는 코드
```

```
# try 문 안에는 예외가 발생할 가능성이 있는 모든 코드 삽입
# except 문 안에는 예외가 발생했을 때 실행할 모든 코드 삽입
# else 문 안에는 예외가 발생하지 않았을 때 실행할 코드 삽입
# finally 문 안에는 예외 발생 여부와 상관없이 무조건 실행할 코드 삽입
# 숫자 입력시 프로그램 정상 작동
>>> try :
    Number = int(input("숫자 입력 : "))
>>> except :
    print("예외가 발생하였습니다.")
>>> else :
    print("입력한 숫자는 ", Number, "입니다.")
>>> finally :
    print("finally 문은 무조건 실행하는 코드")
```

```
숫자 입력 : 7
입력한 숫자는 7 입니다.
finally 문은 무조건 실행하는 코드
```

“Selena's Comment!”

수강생 여러분!

Selena 강사와 함께 파이썬 기초에 대해 정복하셨나요?!

제가 현업에 있으면서 중요하다고 생각한 내용들을 정리해서 가르쳐드렸어요.

파이썬 프로그래밍이 쉬우면서 또 헷갈리는 부분이 많을 수 있는데요. 정말 고생 많으셨어요!

지금까지 배웠던 내용들이 기반이 되어서 생각하는 모든 걸 만들어낼 수 있어요.

그게 바로 프로그래밍의 매력이거든요. IT세계에 오신 걸 환영합니다!!! 闾 闾 闾)闾)

앞으로는 파이썬을 활용한 데이터 전처리, 분석, 시각화, 그리고 크롤링에 대해 배워볼게요!!

다음 단계로 가기 전에! 파이썬 프로그래밍에 대해 복습하면서 Selena 강사에게 강의 관련해서 궁금한 내용을 질문해 보세요~! 감사합니다. :)

STEP 10. 파이썬 라이브러리 개념

01. Python과 Machine Learning

- 파이썬은 뛰어난 확장성과 연계 호환성을 가짐
 - 분석 영역을 넘어 ML 기반의 다양한 Application 개발 용이
 - 기존 Application과 연계가 쉬움
- 딥 러닝 프레임워크들이 파이썬 기반으로 작성
 - 대부분의 딥 러닝 관련 튜토리얼, 설명 자료들이 파이썬으로 작성되어 제공
 - 현 시점에서 딥 러닝을 학습하기 위한 최적의 언어는 파이썬!

02. 수치 계산 라이브러리(NumPy)

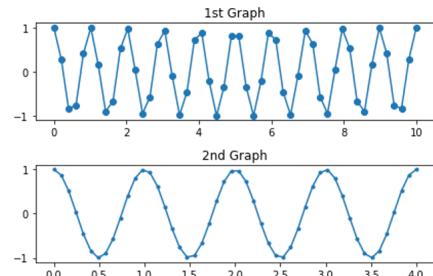
- Numerical Python의 약자로 대표적인 파이썬 기반 수치 해석 라이브러리
- 선형대수 연산에 필요한 다차원 배열과 배열 연산을 수행하는 다양한 함수 제공
- 설치 방법
 - pip install NumPy
- 사용 방법
 - import NumPy as np
- <http://www.NumPy.org>에서 NumPy에 대한 다양한 설명 참고

03. 데이터 처리 라이브러리(Pandas)

- 파이썬에서 사용하는 데이터 분석 라이브러리
- 행과 열로 이루어진 2차원 데이터를 효율적으로 가공할 수 있는 다양한 기능 제공
- 설치 방법
 - pip install pandas
- 사용 방법
 - import pandas as pd
- <https://pandas.pydata.org/>에서 pandas에 대한 다양한 설명 참고

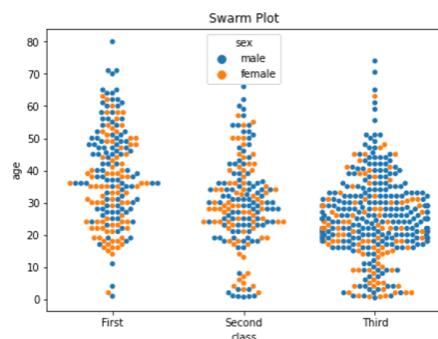
04. 데이터 시각화 라이브러리(Matplotlib)

- matplotlib은 파이썬에서 데이터를 차트나 플롯으로 시각화하는 라이브러리
- matplotlib.pyplot 모듈의 함수를 이용하여 간편하게 그래프를 만들고 변화를 줄 수 있음
- 설치 방법
 - pip install matplotlib
- 사용 방법
 - import matplotlib.pyplot as plt
- <https://matplotlib.org/>에서 matplotlib에 대한 다양한 설명 참고



05. 데이터 시각화 라이브러리(Seaborn)

- seaborn은 matplotlib 기반의 시각화 라이브러리
- 유익한 통계 기반 그래픽을 그리기 위한 고급 인터페이스를 제공
- 설치 방법
 - pip install seaborn
- 사용 방법
 - import seaborn as sns
- <https://seaborn.pydata.org/>에서 seaborn에 대한 다양한 설명 참고



06. 웹 데이터 수집 라이브러리(BeautifulSoup)

- HTML 및 XML에서 데이터를 쉽게 처리하는 파이썬 라이브러리
- HTML은 태그로 이루어져 있고, 수많은 공백과 변화하는 소스들 때문에 오류가 있을 가능성이 높지 만 BeautifulSoup을 이용하면 이러한 오류를 잡아서 고친 후 데이터를 전달해줌
- 설치 방법
 - pip install beautifulsoup4
- 사용 방법
 - from bs4 import BeautifulSoup
- <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>에서 BeautifulSoup에 대한 다양한 설명 참고

종가	견일비	시가	고가	저가	거래량
70600.0	100.0	70700.0	71000.0	70300.0	4735065.0
70700.0	600.0	71300.0	71600.0	70600.0	11027606.0
71300.0	100.0	71500.0	72000.0	71300.0	10919239.0
71400.0	800.0	71700.0	71900.0	70900.0	12420710.0
70600.0	700.0	70200.0	70900.0	69900.0	10087450.0
...
84600.0	200.0	84800.0	85400.0	83400.0	22112205.0
84400.0	1400.0	84100.0	84600.0	83700.0	26302077.0
83000.0	1000.0	81700.0	83400.0	81000.0	28046832.0
82000.0	1700.0	84500.0	85000.0	82000.0	39615978.0
83700.0	1900.0	83200.0	85600.0	83200.0	31859808.0

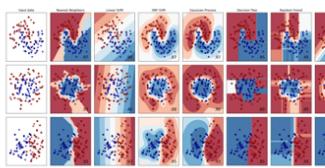
[그림. 삼성전자 일별 시세 정보 웹 크롤링 결과]

07. 머신러닝 라이브러리(Scikit Learn)

- 파이썬 기반의 머신러닝을 위한 가장 쉽고, 효율적인 개발 라이브러리
- 머신러닝은 곧 Scikit-Learn을 의미할 정도로 오랜 기간 파이썬 영역에서 인정 받아 옴
- R에 비견될 수 있는 Python 세계의 분석 라이브러리
- 머신러닝을 위한 다양한 알고리즘과 개발을 위한 프레임워크와 API를 제공
- 회귀 모델, 분류 모델, 데이터 전처리, 성능 측정등 머신러닝에 필요한 도구를 두루 갖춤
- 설치 방법
 - pip install scikit-learn
- 사용 방법
 - import scikit-learn
- <http://scikit-learn.org/stable/documentation>에서 scikit-learn에 대한 다양한 설명 참고

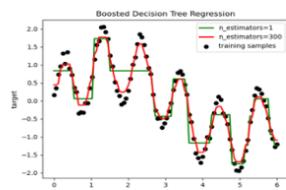
Classification
Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...



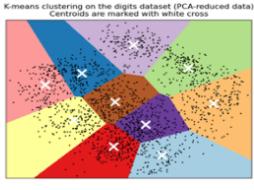
Regression
Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...



Clustering
Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...



Examples Examples Examples

STEP 11. 수치계산

“Selena’s Comment!”

Selena와 함께 배우는 첫 라이브러리예요.
NumPy는 수치 계산 라이브러리로 협업에서 항상 사용하고 있어요.
그만큼 중요한 라이브러리예요. 화이팅해봅시다!!

01. NumPy 기본 개념

- Numerical Python의 약자로 대표적인 파이썬 기반 수치 해석 라이브러리
- 선형대수 연산에 필요한 다차원 배열과 배열 연산을 수행하는 다양한 함수 제공
- 설치 방법
 - pip install NumPy
- 사용 방법
 - import NumPy as np
- <http://www.NumPy.org>에서 넘파이에 대한 다양한 설명 참고

02. NumPy 배열

- NumPy에서 배열은 ndarray 또는 array 부름
- NumPy.array와 python.array는 다름
 - NumPy array는 NumPy array끼리 연산이 가능하지만 python list는 덧셈만 가능
 - ✓ NumPy array : $[10, 5, 3, 7, 1, 5] + [10, 5, 3, 7, 1, 5] = [20, 10, 6, 14, 2, 10]$
 - ✓ python list : $[10, 5, 3, 7, 1, 5] + [10, 5, 3, 7, 1, 5] = [10, 5, 3, 7, 1, 5, 10, 5, 3, 7, 1, 5]$
 - NumPy array는 array 전체에 연산이 가능하지만 Python list는 곱셈(list 요소 반복)만 가능
 - ✓ NumPy array : $[10, 5, 3, 7, 1, 5] + 5 = [15, 10, 8, 12, 6, 10]$
 - ✓ python list : $[10, 5, 3, 7, 1, 5] * 2 = [10, 5, 3, 7, 1, 5, 10, 5, 3, 7, 1, 5]$
- NumPy array가 python list에 비해 문법이 간단하고 성능이 뛰어나다.

“Selena’s Comment!”

NumPy.array와 python.array는 달라요. 헷갈려 하시는 분들이 많으셔서 준비했어요.

문법적으로 차이점이 있다는 걸 이해해 주시고 성능 측면에서 차이가 커요.

NumPy.array가 문법도 간단하고 훨씬 짧은 시간으로 연산을 해서 성능이 뛰어나요.

Python list는 값을 추가하고 제거하는 업무에서 쓰고 NumPy array는 수치 계산이 많고 복잡할 때나 행렬 같은 다차원 배열을 쓰는 업무에서 사용하면 됩니다!

- NumPy는 모든 배열의 값이 기본적으로 같은 타입
- NumPy에서는 각 차원(Dimension)을 축(axis)이라고 표현

1D array shape: (4,)

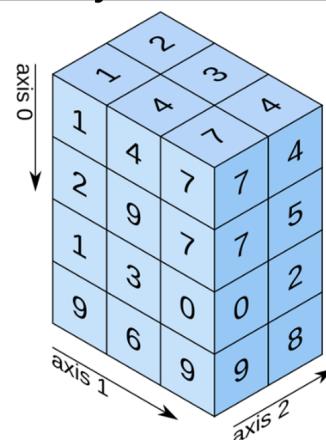
1	0	3	5
---	---	---	---

axis 0 →

2D array shape: (2, 4)

3.3	2.7	0.1	2.1
8.1	9.9	5.4	6.2

axis 0 ↓ axis 1 →

3D array shape: (4, 3, 2)**“Selena’s Comment!”**

제가 강의할 때 정말 중요하다고 하는 부분이에요.

NumPy 배열 1차원, 2차원, 3차원을 정확하게 이해하고 넘어가셔야 해요.

그래야 나중에 숫자로 출력되었을 때 이해할 수 있어요.

위의 그림을 보고 axis0, axis1, axis2에 각각 어떤 값이 들어가는지 정확하게 이해하고 넘어갑니다! 명심!

2-1. 배열 예제

```
# 아래와 같은 데이터는 2개의 축을 가지며,
# 첫 번째 축은 길이가 2이고 두 번째 축은 길이가 3이다.
>>> [[ 1,  0,  0],
       [ 0,  1,  2]]
```

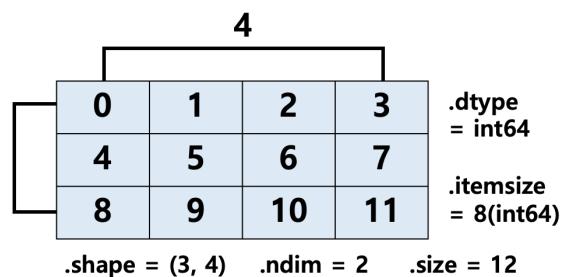
```
[[1, 0, 0], [0, 1, 2]]
```

```
# 아래와 같은 데이터는 1개의 축을 가지며,
# 축은 3가지 요소(element)를 가지고 있다고 하고 길이(length)는 3이다.
>>> [1, 2, 1]
```

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

2-2. 배열 대표적인 속성값

- ndarray.shape : 배열의 각 축(axis)의 크기
- ndarray.ndim : 축의 개수(Dimension)
- ndarray.dtype : 각 요소(Element)의 타입
- ndarray.itemsize : 각 요소(Element)의 타입의 bytes 크기
- ndarray.size : 전체 요소(Element)의 개수



```
# a라는 변수에 (3, 4) 크기의 2D 배열을 생성해보자.
>>> a = np.arange(12).reshape(3, 4)
>>> print(a)

[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
# ndarray.shape : 배열의 각 축(axis)의 크기
>>> print(a.shape)

(3, 4)
```

```
# ndarray.dtype : 각 요소(Element)의 타입
>>> print(a.dtype)

int64
```

```
# ndarray.itemsize : 각 요소(Element)의 타입의 bytes 크기
>>> print(a.itemsize)

8
```

```
# ndarray.size : 전체 요소(Element)의 개수
>>> print(a.size)

12
```

03. NumPy 배열 생성

- np.array를 이용하여 Python에서 사용하는 Tuple(튜플)이나 List(리스트)를 입력으로 NumPy.ndarray를 생성
- np.zeros, np.ones, np.empty를 이용하여 다양한 차원의 데이터를 쉽게 생성 가능
 - np.zeros(shape) : 0으로 구성된 N차원 배열 생성
 - np.ones(shape) : 1로 구성된 N차원 배열 생성
 - np.empty(shape) : 초기화되지 않은 N차원 배열 생성

“Selena's Comment!”

ones는 주로 알고리즘을 짤 때 값을 초기화시키는 동시에 배열을 생성할 때 사용해요.
이때, 주의할 점은 np.empty는 메모리가 초기화되지 않고 할당되기 때문에 쓰레기 값이 있을 수 있습니다. 주의해서 사용해 주세요.

3-1. NumPy 배열 생성

```
# a 배열 생성 & 타입 확인
>>> a = np.array([2,3,4])
>>> print(a)
>>> print(a.dtype)

[2 3 4]
int64

# b 배열 생성 & 타입 확인
>>> b = np.array([1.2, 3.5, 5.1])
>>> print(b)
>>> print(b.dtype)

[1.2 3.5 5.1]
float64
```

3-2. 다양한 차원의 배열 생성

```
# (3,4) 크기의 배열을 생성하여 0으로 채움
>>> print(np.zeros((3,4)))

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

# (2,3,4) 크기의 배열을 생성하여 1로 채움
>>> print(np.ones((2,3,4), dtype=np.int64))

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

04. NumPy arange와 linspace 데이터 생성

4-1. 연속적인 데이터 생성

- np.arange와 np.linspace를 이용하여 연속적인 데이터를 쉽게 생성 가능
 - np.arange : N 만큼 차이 나는 숫자 생성
 - np.linspace : N 등분한 숫자 생성

```
# 10이상 30미만 까지 5씩 차이나게 생성
>>> print(np.arange(10, 30, 5))

[10 15 20 25]
```

```
# 0~99까지 100등분
>>> x = np.linspace(0, 99, 100)
>>> print(x)

[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.
 18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.
 36.  37.  38.  39.  40.  41.  42.  43.  44.  45.  46.  47.  48.  49.  50.  51.  52.  53.
 54.  55.  56.  57.  58.  59.  60.  61.  62.  63.  64.  65.  66.  67.  68.  69.  70.  71.
 72.  73.  74.  75.  76.  77.  78.  79.  80.  81.  82.  83.  84.  85.  86.  87.  88.  89.
 90.  91.  92.  93.  94.  95.  96.  97.  98.  99.]
```

4-2. arange와 linspace의 차이점

- np.arange와 np.linspace의 차이점
 - np.arange([start], stop, [step].)
 - [] 생략 가능, 끝 값 포함 안 함
 - 장점 : step, 범위를 구간, 간격 강조할 때 사용하면 코드 가독성 UP!
 - np.linspace(start, stop, num=50.)
 - 처음 값과 끝 값 포함, 몇 개로 만들지 매개변수로 줌
 - 장점 : 개수 강조할 때 사용하면 코드 가독성 UP!

```
# arange 함수는 끝 값을 포함하지 않기 때문에
# 가독성을 위해서 1.25를 1+0.25로 표현!
# 0부터 1.25 미만까지(끝 값 포함 안하니까 1까지 출력) 0.25씩 차이나게 생성
>>> print(np.arange(0, 1+0.25, 0.25))

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

```
# 위와 동일한 결과를 linspace 함수를 이용하여 코딩
# 0부터 1까지(끝 값 포함) 5등분
>>> print(np.linspace(0,1,5))

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

“Selena’s Comment!”

제가 많은 수강생분들을 가르치면서 많은 분들이 겪은 일인데요. :)
 arrange가 아니라 arange 함수라는 거예요. 오탏가 많이 나요. 주의!
 np.arange와 np.linspace 두 함수 모두 현업에서 많이 사용해요.
 잘 사용하면 정말 유용해요. 제대로 연습해서 자신의 것으로 만듭시다!
 실습으로 차근차근 정확히 이해하고 넘어가주세요.

05. NumPy 배열 출력

- 1D와 2D 배열은 설명하지 않아도 어떻게 출력되는지 확인하실 수 있으나,
- 3D 배열은 2차원이 N개 출력되는 형식으로 나타남

```
# 3차원 배열 출력하는 코드
```

```
c = np.arange(24).reshape(2,3,4)
print(c)
```

```
# (3, 4)크기의 2차원 배열이 2개 출력되는 형식
```

```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9  10 11]]
 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

“Selena’s Comment!”

1차원, 2차원 배열 출력은 쉽게 이해가 되지만

3차원 배열 출력은 어려워하세요.

헷갈린다면 오늘 앞 시간에 배웠던 배열 그림 부분을 다시 보고 이해해 주세요!

할 수 있어요~!

06. NumPy 기본 연산

6-1. element wise 연산

- NumPy에서 수치 연산은 기본적으로 element wise 연산
 - 차원(축)을 기준으로 행렬 내에서 같은 위치에 있는 원소끼리 연산을 하는 방식

```
# a와 b 배열 생성하여 출력
>>> a = np.array([20,30,40,50])
>>> b = np.arange4.
>>> print(a)
>>> print(b)

[20 30 40 50]
[0 1 2 3]
```

```
# a에서 b에 각각의 원소를 - 연산
>>> c = a-b
>>> print(c)

[20 29 38 47]
```

```
# b 각각의 원소에 제곱 연산
>>> print(b**2)

[0 1 4 9]
```

```
# a 각각의 원소에 *10 연산
>>> print(10*a)

[200 300 400 500]
```

```
# a 각각의 원소가 35보다 작은지 Boolean 결과
>>> print(a < 35)

[ True True False False]
```

6-2. 여러가지 곱셈

- * : 각각의 원소끼리 곱셈 (Elementwise product, Hadamard product)
- @ : 행렬 곱셈 (Matrix product)

```
# A와 B 배열 생성하여 출력
>>> A = np.array( [[1,1],
                   [0,1]] )
>>> B = np.array( [[2,0],
                   [3,4]] )
>>> print(A)
>>> print(B)

[[1 1]
 [0 1]]
[[2 0]
 [3 4]]
```

```
# A * B
# 각각의 원소끼리 곱셈
>>> print(A * B)

[[2 0]
 [0 4]]
```

```
# A @ B
# 행렬 곱셈 사용
>>> print(A @ B)

[[5 4]
 [3 4]]
```

“Selena’s Comment!”

@은 행렬 곱셈이에요.
흔히 사용하는 곱셈 기호는 *라는 점 기억해 주세요~!

6-3. 자동 형 변환

- 수치 연산을 진행할 때 각각의 .dtype이 다르면 타입이 큰 쪽(int < float < complex)으로 자동으로 변경

```
# a와 b 배열 생성 & 타입 확인
>>> a = np.ones(3, dtype=np.int32)
>>> b = np.linspace(0, np.pi, 3)
```

```
>>> print(a)
>>> print(b)
>>> print(a.dtype)
>>> print(b.dtype)
```

```
[1 1 1]
[0.           1.57079633  3.14159265]
int32
float64
```

```
# a(int), b(float) 연산 시 float로 upcasting
>>> c = a + b
>>> print(c)
>>> print(c.dtype)
```

```
[1.           2.57079633  4.14159265]
float64
```

```
# 마찬가지로 복소수 연산 시 complex(복소수)로 upcasting
# exp 함수는 지수 함수
>>> d = np.exp(c*1j)
>>> print(d)
>>> print(d.dtype)
```

```
[ 0.54030231+0.84147098j -0.84147098+0.54030231j -0.54030231-0.84147098j]
complex128
```

6-4. 집계함수

- .sum : 모든 요소의 합
- .min : 모든 요소 중 최소값
- .max : 모든 요소 중 최대값
- .argmax : 모든 요소 중 최대값의 인덱스
- .cumsum : 모든 요소의 누적합

.min = 0

0	1	4	9
16	25	36	49

.max = 49

.argmax = 7

.sum = 0 + 1 + ... + 36 + 49 = 140

.cumsum = [0, 1, 5, ..., 140]

```
# a 배열 생성 & 출력
# 0부터 8미만까지 출력하고 (2,4) 크기로 재가공하고 제곱하여 출력
>>> a = np.arange8..reshape(2, 4)**2
>>> print(a)
```

```
[[ 0  1  4  9]
 [16 25 36 49]]
```

```
# 모든 요소의 합
>>> print(a.sum())
```

140

```
# 모든 요소 중 최대값의 인덱스
```

```
>>> print(a.argmax())
```

7

```
# 모든 요소의 누적합
```

```
# 14 = 0 + 1 + 4 + 9
```

```
>>> print(a.cumsum())
```

[0 1 5 14 30 55 91 140]

6-5. 집계함수 axis 값을 매개변수로 입력

- `sum()`, `.min()`, `.max()`, `.cumsum()`과 같은 연산에 `axis` 값을 입력하면, 축을 기준으로도 연산 가능
 - `axis=0` (열 기준)
 - `axis=1` (행 기준)

0	1	2	3
4	5	6	7
8	9	10	11

`.sum(axis=1)`
`= [6, 22, 38]`
`.sum(axis=0) = [12, 15, 18, 21]`

```
# b 배열 생성 & 출력
```

```
>>> b = np.arange(12).reshape(3, 4)
```

```
>>> print(b)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
# axis = 0은 열 기준으로 연산
```

```
>>> print(b.sum(axis=0))
```

[12 15 18 21]

```
# axis = 1은 행 기준으로 연산
```

```
>>> print(b.sum(axis=1))
```

[6 22 38]

07. NumPy 범용 함수

- 필요한 범용 함수는 사이트 확인
 - <https://NumPy.org/doc/1.18/reference/ufuncs.html#available-ufuncs>

add(x1, x2, /[, out, where, casting, order, …])	fabs(x, /[, out, where, casting, order, …])
subtract(x1, x2, /[, out, where, casting, …])	rint(x, /[, out, where, casting, order, …])
multiply(x1, x2, /[, out, where, casting, …])	sign(x, /[, out, where, casting, order, …])
divide(x1, x2, /[, out, where, casting, …])	heaviside(x1, x2, /[, out, where, casting, …])
logaddexp(x1, x2, /[, out, where, casting, …])	conj(x, /[, out, where, casting, order, …])
logaddexp2(x1, x2, /[, out, where, casting, …])	conjugate(x, /[, out, where, casting, …])
true_divide(x1, x2, /[, out, where, …])	exp(x, /[, out, where, casting, order, …])
floor_divide(x1, x2, /[, out, where, …])	exp2(x, /[, out, where, casting, order, …])
negative(x, /[, out, where, casting, order, …])	log(x, /[, out, where, casting, order, …])
positive(x, /[, out, where, casting, order, …])	log2(x, /[, out, where, casting, order, …])
power(x1, x2, /[, out, where, casting, …])	log10(x, /[, out, where, casting, order, …])
remainder(x1, x2, /[, out, where, casting, …])	expm1(x, /[, out, where, casting, order, …])
mod(x1, x2, /[, out, where, casting, order, …])	log1p(x, /[, out, where, casting, order, …])
fmod(x1, x2, /[, out, where, casting, …])	sqrt(x, /[, out, where, casting, order, …])
divmod(x1, x2[, out1, out2], / [[, out, …]])	square(x, /[, out, where, casting, order, …])
absolute(x, /[, out, where, casting, order, …])	cbrt(x, /[, out, where, casting, order, …])
	reciprocal(x, /[, out, where, casting, …])
	gcd(x1, x2, /[, out, where, casting, order, …])

```
# B 배열 생성 & 출력
>>> B = np.array([1, 4, 9])
>>> print(B)

[1 4 9]
```

```
# y = sqrt(x)
# sqrt는 제곱근 계산
>>> print(np.sqrt(B))

[1. 2. 3.]
```

08. NumPy 인덱싱과 슬라이싱

- NumPy 인덱싱(Indexing, 가리킴)과 슬라이싱(Slicing, 잘라냄)은 각각 문자열에서 한 개 또는 여러 개를 가리켜서 그 값을 가져오거나 뽑아내는 방법

8-1. 인덱싱과 슬라이싱

- NumPy에서 인덱싱과 슬라이싱에 대한 개념은 Python과 기본적으로 동일

```
# a 배열 생성 & 출력
>>> a = np.arange(10)**2
>>> print(a)

[ 0  1  4  9 16 25 36 49 64 81]
```

```
# a 배열의 2번째 인덱스 출력
>>> print(a[2])
4

# a 배열의 2~4번 인덱스 출력
>>> print(a[2:5])
[ 4 9 16]

# reverse : 배열의 요소 거꾸로 출력
>>> print(a[ : :-1])
[81 64 49 36 25 16 9 4 1 0]

# 0~5번에서 2Step 인덱스 출력
# a[0:6:2] = a[:6:2]
# 인덱스 0, 2, 4 해당하는 값에 1000 삽입
>>> a[0:6:2] = 1000
>>> print(a)
[1000 1 1000 9 1000 25 36 49 64 81]
```

8-2. 인덱스 배열로 인덱싱

- 인덱스를 가진 배열로 인덱싱 진행

`a = np.arange(8)**2`

0	1	4	9	16	25	36	49
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

`i = np.array([1, 1, 3, 5])`

<code>a[i] =</code>	1	1	9	25
---------------------	----------	----------	----------	-----------

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

<code>a[j] =</code>	9	16
	4	25

```
# a 배열 생성 & 출력
>>> a = np.arange(8)**2
>>> print(a)

[ 0 1 4 9 16 25 36 49]
```

```
# i 1차원 배열 생성 & 출력
# a 배열의 index로 i를 삽입하여 출력
>>> i = np.array([1, 1, 3, 5])
>>> print(a[i])

[ 1 1 9 25]
```

```
# j 2차원 배열 생성
# a 배열의 index로 j를 삽입하여 출력
>>> j = np.array([[3,4], [2,5]])
>>> print(a[j])

[[ 9 16]
 [ 4 25]]
```

8-3. boolean 인덱스

- Boolean 타입을 가진 값들로 인덱싱 진행

`a = np.arange(12).reshape(3,4)`

0	1	2	3
4	5	6	7
8	9	10	11

`b = a > 4`

False	False	False	False
False	True	True	True
True	True	True	True

`a[b]`

5	6	7	8	9	10	11
---	---	---	---	---	----	----



`a[b] = 0`

0	1	2	3
4	0	0	0
0	0	0	0

```
# a 배열 생성 & 출력
>>> a = np.arange(12).reshape(3,4)
>>> print(a)

[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
# b는 a > 4 조건이 적용된 Boolean 값이 든 배열
>>> b = a > 4
>>> print(b)
```

```
[[False False False False]
 [False  True  True  True]
 [ True  True  True  True]]
```

```
# Boolean 값이 든 b 배열을 a 배열의 index로 삽입
# True인 값들만 출력
>>> print(a[b])
>>> a[b].shape
```

```
[ 5  6  7  8  9 10 11]
(7,)
```

```
# a[b]에 해당하는 애들만 0 삽입하여 a 출력
>>> a[b] = 0
>>> print(a)
```

```
[[0 1 2 3]
 [4 0 0 0]
 [0 0 0 0]]
```

“Selena's Comment!”

NumPy 인덱싱과 슬라이싱은 실무에서 많이 사용하는 중요한 개념이에요.

Selena 강사는 데이터 전처리 단계에서 NumPy boolean 인덱싱으로 원하는 데이터만 추출해서 사용해요!

정말 많이 사용하는 방법 중 하나예요.

09. NumPy 크기 변경

- np.ndarray의 shape를 다양한 방법으로 변경 가능
 - .ravel : 1차원으로 변경
 - reshape : 지정한 차원으로 변경
 - T : 전치(Transpose) 변환

.ravel()
== **.reshape(-1)**

0	1	2	3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	---	---	---	----	----

a = np.arange(12).reshape(3,4)

0	1	2	3
4	5	6	7
8	9	10	11



.reshape(2,6)

0	1	2	3	4	5
6	7	8	9	10	11

.T

0	4	8
1	5	9
2	6	10
3	7	11

```
# a 배열 생성 & shape 출력
>>> a = np.arange(12).reshape(3,4)
>>> print(a)
>>> print(a.shape)

[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
(3, 4)
```

```
# .ravel : 모든 원소를 1차원으로 변경
>>> print(a.ravel())
>>> print(a.reshape(-1))

[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

```
# a[b]에 해당하는 애들만 0 삽입하여 a 출력
>>> a[b] = 0
>>> print(a)
# .reshape : 지정한 차원으로 변경
# [3,4] => [2,6]로 변경
>>> print(a.reshape(2,6))

[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
[[0 1 2 3]
 [4 0 0 0]
 [0 0 0 0]]
```

```
# .T : [3,4]의 전치(transpose)변환으로 [4,3] 출력
>>> print(a.T)
>>> print(a.T.shape)
```

```
[[ 0  4  8]
 [ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]]
(4, 3)
```

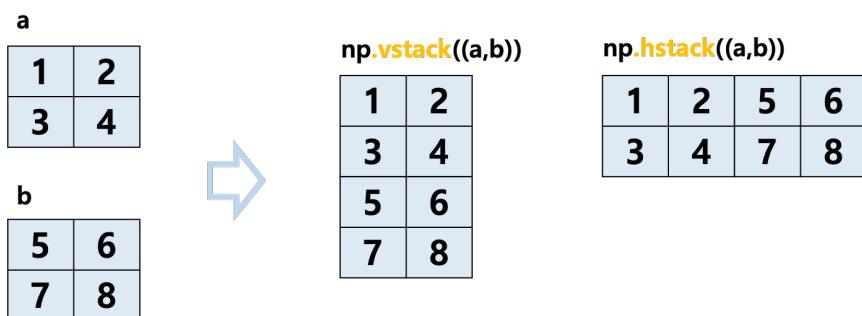
만약 전치 형태로 a 배열에 저장하고 싶다면!

```
>>> a = a.T
>>> print(a)
```

```
[[ 0  4  8]
 [ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]]
```

10. NumPy 데이터 합치기

- np.vstack와 np.hstack을 통해 데이터를 합치기
 - np.vstack : axis=0(열)기준으로 쌓음
 - np.hstack : axis=1(행)기준으로 쌓음



```
# a 배열 생성 & 출력
>>> a = np.array([1, 2, 3, 4]).reshape(2, 2)
>>> print(a)

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

```
# b 배열 생성 & 출력
>>> b = np.array([5, 6, 7, 8]).reshape(2, 2)
>>> print(b)

[[5 6]
 [7 8]]
```

```
# [2,2] => [4,2]
# np.vstack(): axis=0(열) 기준으로 쌓음
>>> print(np.vstack((a,b)))
```

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

11. NumPy 데이터 쪼개기


```
a = np.arange(12).reshape(2,6)
```

0	1	2	3	4	5
6	7	8	9	10	11



`np.hsplit(a, 3) : 3개로 등분`

0	1	2	3	4	5
6	7	8	9	10	11

`np.hsplit(a, (3,4))` : 3번 째 열부터
~ 4번 째 열 미만을 기준으로 분할

0	1	2	3	4	5
6	7	8	9	10	11

```
# a 배열 생성 & 출력  
>>> a = np.arange(12).reshape(2, 6)  
>>> print(a)
```



```
[[ 0  1  2  3  4  5]  
 [ 6  7  8  9 10 11]]
```

```
# [2,6] => [2,2] 데이터 3개로 등분  
>>> print(np.hsplit(a, 3))
```

```
[array([[0, 1],  
       [6, 7]]), array([[2, 3],  
       [8, 9]]), array([[ 4,  5],  
       [10, 11]])]
```

```
# [2,6] => [:, :3], [:, 3:4], [:, 4:]로 분할
# a를 3번째 열 ~ 4번째 열 미만 기준으로 분할하여 3개의 array를 반환
>>> print(a)
>>> print(np.hsplit(a, (3,4)))
```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
[array([[0, 1, 2],
       [6, 7, 8]]), array([[3],
       [9]]), array([[ 4,  5],
       [10, 11]])]
```

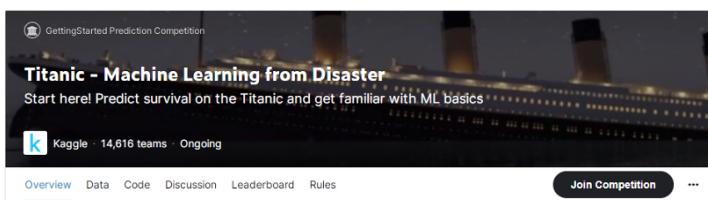
STEP 12. 데이터 처리

01. Pandas 기본 개념

- 파이썬에서 사용하는 데이터 분석 라이브러리
- 행과 열로 이루어진 2차원 데이터를 효율적으로 가공할 수 있는 다양한 기능 제공
- 설치 방법
 - pip install pandas
- 사용 방법
 - import pandas as pd
- <https://pandas.pydata.org/> 에서 판다스에 대한 다양한 설명 참고

02. Kaggle, Titanic 데이터 설명

- Kaggle(캐글)은 데이터 분석 경진 대회를 주최하는 플랫폼
 - 경진대회는 회사의 과제, 연구, 주요 서비스를 위해 분석이 필요한 데이터를 제공해서 주최
 - 실제 기업 데이터를 다룰 수 있음
 - 대기업 경력직을 채용할 때 면접 문제로 사용됨
- Titanic 데이터는 Kaggle(캐글)의 대표적인 데이터 분석 입문용 데이터셋
 - 문제의 목표 : 타이타닉에서 살아남을 수 있는 승객을 예측하기
 - <https://www.kaggle.com/c/titanic>에서 데이터 다운로드 가능



“Selena’s Comment!”

캐글은 데이터 분석 입문자들이 쉽게 실무 데이터셋을 얻을 수 있는 사이트예요.
그리고 데이터 사이언티스트 지인이 경력으로 다른 회사를 지원할 때 캐글 문제를 과제로 줘서 면접을 진행하는 경우도 보았어요. 유용한 사이트니까 잘 활용해 보세요!

보통 ‘타이타닉’ 데이터로 처음 데이터 분석을 시작해요.
더 추천해 드리자면!

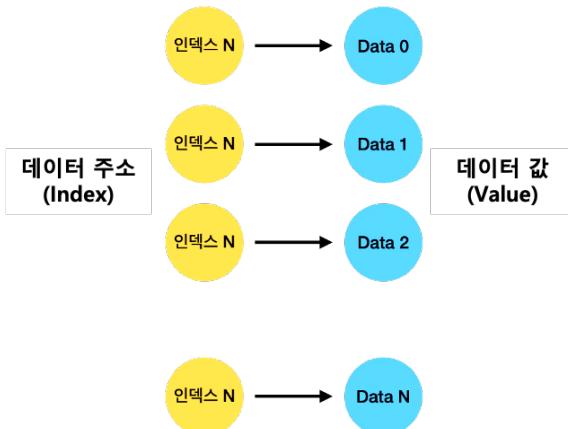
‘House Prices-Advanced Regression Techniques’ 데이터로 연습하면 좋아요. 변수가 많아서 번거롭지만 그만큼 전처리를 배울 수 있고 모든 모델에 적용해서 예측 문제를 수행할 수 있어서 연습하기 좋아요.

03. Pandas 데이터 형식

- pandas는 시리즈(Series)와 데이터프레임(DataFrame)이라는 구조화된 데이터 형식을 제공

3-1. 시리즈(Series)

- 시리즈는 데이터가 순차적으로 나열된 1차원 배열의 형태
- index와 value가 일대일 대응 관계



```
# 딕셔너리로 Series 생성
# 딕셔너리의 키는 시리지의 인덱스와 대응하고,
# 딕셔너리의 각 키에 매칭되는 값은 시리즈의 데이터 값으로 변환됨.
>>> dict_data = {'a':1,'b':2,'c':3}
>>> series_data = pd.Series(dict_data)
```

```
# 데이터 타입, 내용 확인
>>> print(type(series_data))
>>> print(series_data)

<class 'pandas.core.series.Series'>
a    1
b    2
c    3
dtype: int64
```

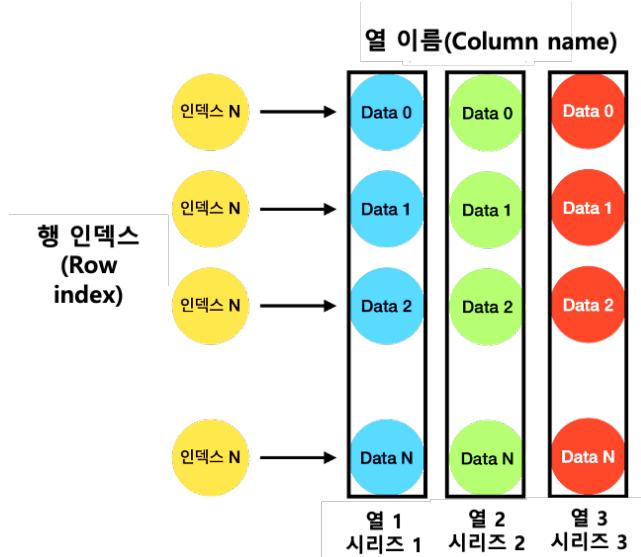
```
# 리스트로 Series 생성
# 리스트를 시리즈로 변환할 때는 딕셔너리의 키처럼 인덱스로 변환될 값이 없음.
# 인덱스를 별도로 지정하지 않으면 디폴트로 정수형 위치 인덱스가 지정됨.
>>> list_data = ['2022-10-11',3.14,'ABC',100,True]
>>> series_data = pd.Series(list_data)
```

```
# 데이터 타입, 내용 확인
>>> print(type(series_data))
>>> print(series_data)

<class 'pandas.core.series.Series'>
0    2022-10-11
1        3.14
2        ABC
3        100
4      True
dtype: object
```

3-2. 데이터프레임(DataFrame)

- 데이터프레임은 행과 열로 만들어지는 2차원 배열의 형태
- 데이터프레임의 열은 각각의 시리즈 객체



```
# 딕셔너리로 DataFrame 생성
>>> dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}
>>> df = pd.DataFrame(dict_data)
```

```
# 데이터 탑, 내용 확인
# 1개의 열들이 1개의 시리즈라고 했으므로, 키값이 열 이름이 됨.
>>> print(type(df))
>>> print(df)

<class 'pandas.core.frame.DataFrame'>
   c0   c1   c2   c3   c4
0    1    4    7   10   13
1    2    5    8   11   14
2    3    6    9   12   15
```

04. Pandas 파일 불러오기

- pandas는 다양한 형태의 외부 파일을 읽어와서 데이터프레임으로 변환하는 함수를 제공
 - 데이터 입출력 도구

File Format	Reader	Writer
CSV	read_csv	to_csv
Excel	read_excel	to_excel
JSON	read_json	to_json
SQL	read_sql	to_sql
HTML	read_html	to_html

```
# colab 업로드 메뉴를 통해 드라이브 업로드
# csv 파일 읽어오기
titanic = pd.read_csv("/content/titanic.csv")
```

05. Pandas 데이터 내용 확인

- **.columns**: 컬럼명 확인
- **.head()**: 데이터의 상단 5개 행 출력
- **.tail()**: 데이터의 하단 5개 행 출력
- 팔호 () 안에 숫자를 넣으면 그 숫자만큼 행을 출력
- **.shape**: (행, 열) 크기 확인
- **.info()**: 데이터에 대한 전반적인 정보 제공
 - 행과 열의 크기
 - 컬럼명
 - 컬럼별 결측치
 - 컬럼별 데이터 타입
- **.type()**: 데이터 타입 확인

```
# 데이터의 컬럼명 확인
>>> print(titanic.columns)
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

```
# .shape :(행, 열) 크기 확인
>>> titanic.shape
(891, 12)
```

```
# .head() : 데이터의 상단 5개 행 출력
>>> titanic.head()
```

```
# .info() : 데이터에 대한 전반적인 정보 제공
>>> titanic.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          --          --      
 0   PassengerId 891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare          891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
# .type() : 데이터 타입 확인
>>> type(titanic)
pandas.core.frame.DataFrame
```

“Selena’s Comment!”

현업에서 Pandas 데이터 내용을 확인하는 함수들은 정말 중요해요!
저도 데이터를 처음 받으면 head 함수로 데이터 어떻게 들어가 있는지 확인하고
info 함수로 데이터의 전반적인 정보를 파악해요!

06. Pandas 특정 열 선택

6-1. 특정 열 선택 (시리즈 반환)

- 열 1개 선택 = 시리즈(Series) 객체 반환
- 데이터프레임의 열 데이터를 1개만 선택할 때는 2가지 방식
 - 1) 대괄호([]) 안에 열 이름을 따옴표(“ ”)와 함께 입력**
 - 2) 도트(.) 다음에 열 이름을 입력**

```
# 열 1개 선택 = 시리즈(Series) 객체 반환
# 대괄호([ ]) 안에 열 이름을 따옴표(“ ”)와 함께 입력
# 데이터 내용 확인
>>> names = titanic['Name']
>>> names.head()

0           Braund, Mr. Owen Harris
1    Cumings, Mrs. John Bradley (Florence Briggs Th...
2           Heikkinen, Miss. Laina
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)
4            Allen, Mr. William Henry
Name: Name, dtype: object
```

6-2. 특정 열 선택 (데이터프레임 반환)

- 열 n개 선택 = 데이터프레임(DataFrame) 반환
- 데이터프레임의 열 데이터를 n개 선택할 때는 객체 1가지 방식
 - 1) 2중 대괄호([[]]) 안에 열 이름을 따옴표(“ ”)와 함께 입력**
- 만약 열 1개를 데이터프레임 객체로 추출하려면 2중 대괄호를 사용

```
# 열 n개 선택 = 데이터프레임(DataFrame) 객체 반환
# 2중 대괄호([[ ]]) 안에 열 이름을 따옴표(“ ”)와 함께 입력
# "sex", "age" column만 추출하여 새 구성
# 데이터 내용 확인
>>> passenger = titanic[["Sex", "Age"]]
>>> passenger.head()
```

	Sex	Age
0	male	22.0
1	female	38.0
2	female	26.0
3	female	35.0
4	male	35.0

07. Pandas 데이터 필터링

- **불리언 인덱싱 + .isin()**
 - 데이터의 특정 범위만 추출
- **.isna()**
 - 결측 값은 True 반환, 그 외에는 False 반환
- **notna()**
 - 결측 값은 False 반환, 그 외에는 True 반환

7-1. 불리언(Boolean) 인덱싱

- True 값을 가진 행만 추출

```
# 조건 : 35살 초과인 데이터 추출 (True, False 반환)
>>> print(passenger["Age"] > 35)

0      False
1      True
2      False
3      False
4      False
...
886     False
887     False
888     False
889     False
890     False
Name: Age, Length: 891, dtype: bool
```

```
# 조건 : 35살 초과인 데이터 추출
# 불리언(Boolean) 인덱싱 : 조건의 결과가 True 값인 행만 추출
>>> above35 = passenger[passenger["Age"] > 35]
>>> above35.head()
```

	Sex	Age
1	female	38.0
6	male	54.0
11	female	58.0
13	male	39.0
15	female	55.0

“Selena's Comment!”

현업에서 데이터 전처리를 진행할 때, 원하는 데이터만 추출하기 위해서
 주로 불리언 인덱싱을 사용하는 편이에요.
 조건의 결과가 True인 것만 추출해 줘서 되게 간편해요!
 실습 예제를 바꿔보면서 연습해 보세요.

7-2. 불리언 인덱싱 + .isin()

- .isin(): 각각의 요소가 데이터프레임 또는 시리즈에 존재하는지 파악하여 True/False 값 반환

```
# .isin 함수는 각각의 요소가 데이터프레임 또는 시리즈에 존재하는지 파악하여 True/False 값 반환
# Pclass 변수의 값이 1일 경우, True/False 값 반환
>>> titanic["Pclass"].isin([1])

0      False
1      True
2     False
3      True
4     False
...
886    False
887    True
888    False
889    True
890    False
Name: Pclass, Length: 891, dtype: bool
```

- 불리언 인덱싱 + .isin(): 데이터의 특정 범위만 추출

```
# 불리언 인덱싱 + .isin(): 데이터의 특정 범위만 추출
# Pclass 변수의 값이 1일 경우만 추출하여 class1 저장
>>> class1 = titanic[titanic["Pclass"].isin([1])]
>>> class1.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
23	24	1	1	Sloper, Mr. William Thompson	male	28.0	0	0	113788	35.5000	A6	S

```
# 불리언 인덱싱 + .isin(): 데이터의 특정 범위만 추출
# Age 변수의 값이 20살부터 40살까지의 범위만 추출하여 age2040 저장
>>> age2040 = passenger[passenger["Age"].isin(np.arange(20, 41))]
>>> age2040.head()
```

	Sex	Age
0	male	22.0
1	female	38.0
2	female	26.0
3	female	35.0
4	male	35.0

“ Selena's Comment! ”

불리언 인덱싱과 isin() 함수를 함께 사용해서 데이터의 특정 범위만 추출할 수 있어서 굉장히 유용해요.
많이 연습해서 꼭 기억해두세요~!

7-3. .isna()와 .notna()

- .isna(): 결측 값은 True 반환, 그 외에는 False 반환
- .notna(): 결측 값은 False 반환, 그 외에는 True 반환

```
# .isna( ) : 결측 값은 True 반환, 그 외에는 False 반환
# .isna 함수는 데이터프레임 내에 결측 값을 확인하기 위해 사용
# 5번째 행 True 출력
>>> passenger[ "Age" ].isna()[ 0:7 ]

0    False
1    False
2    False
3    False
4    False
5    True
6    False
Name: Age, dtype: bool
```

```
# 결측 값 파악을 위한 데이터 확인
>>> passenger.head7.
```

	Sex	Age
0	male	22.0
1	female	38.0
2	female	26.0
3	female	35.0
4	male	35.0
5	male	NaN
6	male	54.0

```
# .notna( ) : 결측 값은 False 반환, 그 외에는 True 반환
# .notna 함수는 누락되지 않은 값을 찾기 위해 사용
# 5번째 행 False 출력
>>> passenger[ "Age" ].notna()[ 0:7 ]

0    True
1    True
2    True
3    True
4    True
5    False
6    True
Name: Age, dtype: bool
```

```
# 결측 값을 제거한 누락되지 않은 값을 확인
# 5번째 행 제거
>>> ages = passenger[ passenger[ "Age" ].notna() ]
>>> ages.head7.
```

	Sex	Age
0	male	22.0
1	female	38.0
2	female	26.0
3	female	35.0
4	male	35.0
5	male	NaN
6	male	54.0

08. Pandas 결측치 제거

- `.dropna(axis=0) == .dropna()` : 결측 값이 들어있는 행 전체 삭제
- `.dropna(axis=1)` : 결측 값이 들어있는 열 전체 삭제
- `.notna()` : 결측 값은 False 반환, 그 외에는 True 반환

“Selena’s Comment!”

데이터 분석에서는 결측치 처리가 중요한 부분이에요.
 보통 데이터의 양이 많다면 결측치를 제거해 주지만
 현실에서는 데이터의 양이 많지 않은 경우가 대부분이라서 결측치를 대체해 주는 편이긴 해요!
 가끔 열과 행에 많은 결측치가 존재할 때가 있어요.
 그때 dropna 함수로 결측치가 존재하는 열과 행을 제거해 줘요.

```
# 결측 값 파악을 위한 데이터 확인
>>> titanic.head3.
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

```
# .dropna(axis=0) : 결측 값이 들어있는 행 전체 삭제
>>> titanic.dropna(axis=0).head3.
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	C85	C
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C85	S
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	7.9250	NaN	S

```
# .dropna(axis=1) : 결측 값이 들어있는 열 전체 삭제
>>> titanic.dropna(axis=1).head3.
```

	PassengerId	Survived	Pclass	Name	Sex	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Th...	female	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	0	0	STON/O2. 3101282	7.9250

09. Pandas 이름과 인덱스로 특정 행과 열 선택

- **.loc[]**: 행 이름과 열 이름을 사용
- **.iloc[]**: 행 번호와 열 번호를 사용

구분	코드
.loc[]	DataFrame 객체.loc[행 이름, 열 이름]
.iloc[]	DataFrame 객체.iloc[행 번호, 열 번호]

“Selena's Comment!”

loc 함수는 locatio의 약자로, 데이터 프레임의 행이나 열의 label로 인덱싱하는 방법이에요.
 또한, iloc 함수는 integer location의 약자로, 데이터 프레임의 행이나 열의 순서를 나타내는 정수로 특정 값을 추출해오는 방법이에요.
 loc 함수와 iloc 함수를 잘 쓰면 유용해요. 많이 연습해보세요!

```
# .loc[] : 행 이름과 열 이름을 사용
# 나이가 35세 초과인 사람의 이름과 나이 출력
>>> names35 = titanic.loc[titanic["Age"] > 35, ["Name", "Age"]]
>>> names35.head()
```

	Name	Age
1	Cumings, Mrs. John Bradley (Florence Briggs Th... Th...	38.0
6	McCarthy, Mr. Timothy J	54.0
11	Bonnell, Miss. Elizabeth	58.0
13	Andersson, Mr. Anders Johan	39.0
15	Hewlett, Mrs. (Mary D Kingcome)	55.0

```
# .iloc[] : 행 번호와 열 번호를 사용
# 1번째행부터 3번째행까지의 0번째 열의 값을 "No name" 변경
>>> names35.iloc[[1,2,3], 0] = "No name"
>>> names35.head()
```

	Name	Age
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0
6	No name	54.0
11	No name	58.0
13	No name	39.0
15	Hewlett, Mrs. (Mary D Kingcome)	55.0

10. Pandas 데이터 통계

- **.mean()** : 평균값
- **.median()** : 중앙값
- **.describe()** : 다양한 통계량 요약
 - mean, std, min, 25%, 50%, 75%, max
- **.agg()** : 여러 개의 열에 다양한 함수를 적용
 - 모든 열에 여러 함수를 매팅 : group 객체.agg([함수1, 함수2, 함수3, …])
 - 각 열마다 다른 함수를 매팅 : group 객체.agg({'열1': 함수1, '열2': 함수2, …})
- **.groupby()** : 그룹별 집계
- **.value_counts()** : 값의 개수

“Selena’s Comment!”

Pandas 라이브러리의 꽃이라고 생각하는 groupby 함수가 나왔네요.

익숙한 groupby 함수 하나, 열 함수 안 부럽다!

groupby 함수는 같은 값을 하나로 묶어 통계 결과를 얻는 방법이에요.

현업에서 명령어 하나로 많은 인사이트를 얻을 수 있어서 자주 사용해요.

꼭 여러 번 반복하셔야 하는 함수입니다! 명심!

```
# Age 변수의 평균 구하기
```

```
>>> print(titanic["Age"].mean())
```

```
29.69911764705882
```

```
# Age 변수의 중앙값 구하기
```

```
>>> print(titanic["Age"].median())
```

```
28.0
```

```
# describe() 함수를 통해 다양한 통계량 요약
# 특정 변수 "Age", "Fare" 추출
>>> print(titanic[["Age", "Fare"]].describe())
```

	Age	Fare
count	714.000000	891.000000
mean	29.699118	32.204208
std	14.526497	49.693429
min	0.420000	0.000000
25%	20.125000	7.910400
50%	28.000000	14.454200
75%	38.000000	31.000000
max	80.000000	512.329200

```
# agg()를 통해 여러 개의 열에 다양한 함수를 적용
# 모든 열에 여러 함수를 매핑 : group 객체.agg([함수1, 함수2, 함수3,...])
# 각 열마다 다른 함수를 매핑 : group 객체.agg({'열1': 함수1, '열2': 함수2, ...})
>>> print(titanic.agg({"Age" : ["min", "max", "median", "std"],
                      "Fare" : ["min", "max", "mean", "median"]}))

Age           Fare
max      80.000000  512.329200
mean        NaN       32.204208
median     28.000000  14.454200
min       0.420000  0.000000
std      14.526497       NaN
```

```
# groupby() 함수를 이용하여 그룹별 집계 <예제 1>
# groupby로 성별과 클래스로 묶어주고 나이와 요금의 평균 구하기
>>> titanic.groupby(["Sex", "Pclass"])[["Age", "Fare"]].mean()
```

		Age	Fare
Sex	Pclass		
female	1	34.611765	106.125798
	2	28.722973	21.970121
	3	21.750000	16.118810
male	1	41.281386	67.226127
	2	30.740707	19.741782
	3	26.507589	12.661633

```
# groupby() 함수를 이용하여 그룹별 집계 <예제 2>
# groupby를 통해 성별을 묶은 다음, 생존율의 평균 구하기
>>> survive = titanic.groupby("Sex")['Survived'].mean()
>>> (survive * 100).head()
```

```
Sex
female    74.203822
male      18.890815
Name: Survived, dtype: float64
```

```
# .value_counts( ) 함수를 이용하여 개수 구하기
>>> titanic["Pclass"].value_counts()

3      491
1      216
2      184
Name: Pclass, dtype: int64
```

11. Pandas 행과 열 추가와 삭제

• 행 추가

- DataFrame.loc['새로운 행 이름'] = 데이터 값

• 열 추가

- DataFrame 객체['추가하려는 열 이름'] = 데이터 값

• 행 삭제

- DataFrame.drop(index, axis = 0)

• 열 삭제

- DataFrame.drop(변수명, axis = 1)

	이름	국어	영어
0	HM	90	88
1	SY	92	96
2	JH	86	97

“Selena's Comment!”

데이터 분석을 하면 파생 변수를 만드는 경우가 불가피해요.

이때 데이터 프레임에 열을 추가하여 파생 변수를 만들어요!

(파생 변수란 기존의 변수를 조합하여 새로운 변수를 만들어 내는 것을 의미해요.)

11-1. 행 추가

- 행 추가 : DataFrame.loc['새로운 행 이름'] = 데이터 값

```
# newRow 변수에 titanic 데이터의 0번째 행, 모든 열을 삽입(titanic.iloc[0,:])
# .loc 함수를 사용하여 891 이름을 가진 행 자리에 newRow 데이터 삽입
>>> newRow = titanic.iloc[0,:]
>>> titanic.loc[891] = newRow
>>> titanic
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... ...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q
891	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S

```
# 행이 891에서 892로 증가
>>> print(titanic.shape)

(892, 12)
```

11-2. 열 추가

- 열 추가 : DataFrame 객체['추가하려는 열 이름'] = 데이터 값

```
# Pclass * 3 값을 새로운 '3Pclass' 열을 추가
>>> titanic['3Pclass'] = titanic["Pclass"] * 3
```

```
# Pclass * 3 값을 새로운 '3Pclass' 열을 추가
>>> titanic['3Pclass'] = titanic["Pclass"] * 3
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	3Pclass
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	9
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	C85	C	3
2	3	1	3	Heikkinen, Miss. Laina Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	9
3	4	1	1	Allen, Mr. William Henry	female	35.0	1	0	113803	53.1000	C123	S	3
4	5	0	3		male	35.0	0	0	373450	8.0500	NaN	S	9

```
# 열이 12에서 13로 증가
>>> print(titanic.shape)

(892, 13)
```

11-3. 행 삭제

- 행 삭제 : DataFrame.drop(index, axis = 0)

```
# drop 함수를 사용하여 880번째 행부터 889번째 행까지 삭제
>>> titanic = titanic.drop(np.arange(880,890), axis = 0)
```

11-4. 열 삭제

- 열 삭제 : DataFrame.drop(변수명, axis = 1)

```
# 열 삭제
# drop 함수를 사용하여 '3Pclass' 열 삭제
>>> titanic = titanic.drop('3Pclass', axis = 1)
>>> titanic
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...
877	878	0	3	Petroff, Mr. Nedelio	male	19.0	0	0	349212	7.8958	NaN	S
878	879	0	3	Laleff, Mr. Kristo	male	NaN	0	0	349217	7.8958	NaN	S
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11767	83.1583	C50	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q
891	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S

“Selena’s Comment!”

현업 데이터 분석에서 가장 많이 사용하는 Pandas 라이브러리를 배워보았어요!

학습방법 동영상에서 자신만의 강의 노트를 제작하라고 했었죠?!

Pandas 강의노트에 캐글에서 다운 받은 새로운 데이터를 올려보세요.

그리고 pandas 함수들을 사용하여 데이터 분석을 자유자재로 해보세요~!

STEP 13-1. 데이터 시각화

“Selena's Comment!”

데이터 시각화는 데이터 사이언티스트에서 정말 중요한 부분이에요.

최근 채용들을 살펴보면 데이터 분석을 하고 그 결과를 바탕으로 시각화를 진행하여 인사이트를 얻는 역할이 많아졌어요.

데이터 분석과 시각화는 1+1처럼 떼려야 뗄 수 없는 관계예요.

그러면 시각화의 기본기를 다지기 위해 Matplotlib을 배워볼까요?

01. Matplotlib 기본 개념

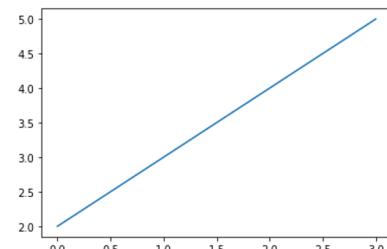
- matplotlib은 파이썬에서 데이터를 차트나 플롯으로 시각화하는 라이브러리
- matplotlib.pyplot 모듈의 함수를 이용하여 간편하게 그래프를 만들고 변화를 줄 수 있음
- 설치 방법
 - pip install matplotlib
- 사용 방법
 - import matplotlib.pyplot as plt
- <https://matplotlib.org/>에서 matplotlib에 대한 다양한 설명 참고

02. Matplotlib 숫자 입력

2-1. 한 개의 리스트 입력

- 한 개의 숫자 리스트 형태로 값을 입력하면 y값으로 인식
- x값은 기본적으로 [0, 1, 2, 3]으로 설정됨
- 파이썬 튜플, 넘파이 배열 형태도 가능
- plt.show() 함수는 그래프를 화면에 나타나도록 함

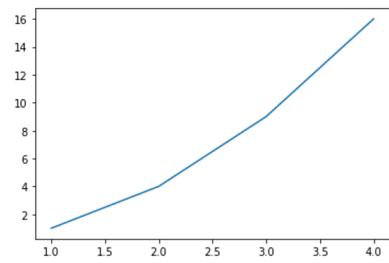
```
# 하나의 숫자 리스트 입력
# 리스트의 값들이 y 값들이라고 가정하고 x값 [0, 1, 2, 3]을 자동으로 만들어냄
# plt.show( ) 함수는 그래프를 화면에 나타나도록 함
>>> plt.plot([2, 3, 4, 5])
>>> plt.show()
```



2-2. 두 개의 리스트 입력

- 두 개의 숫자 리스트 형태로 값을 입력하면 순서대로 x, y 값으로 인식
- 순서쌍(x, y)으로 매칭된 값을 좌표평면 위에 그래프 시각화

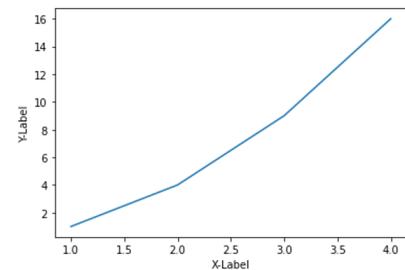
```
# 두 개의 숫자 리스트 입력
# 첫 번째 리스트의 값은 x값, 두 번째 리스트의 값은 y로 적용됨
# 순서쌍(x, y)으로 매칭된 값을 좌표평면 위에 그래프 시각화
>>> plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
>>> plt.show()
```



03. Matplotlib 축 레이블 설정

- xlabel() 함수를 사용하여 그래프의 x축에 대한 레이블 표시
- ylabel() 함수를 사용하여 그래프의 y축에 대한 레이블 표시

```
# xlabel() 함수에 'X-Label' 입력하여 x축에 대한 레이블 표시
# ylabel() 함수에 'Y-Label' 입력하여 y축에 대한 레이블 표시
>>> plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
>>> plt.xlabel('X-Label')
>>> plt.ylabel('Y-Label')
>>> plt.show()
```

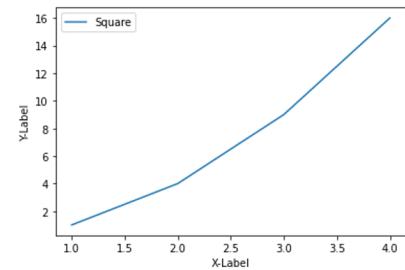


04. Matplotlib 범례(Legend) 설정

- 범례(Legend)는 그래프에 데이터의 종류를 표시하기 위한 텍스트
- legend() 함수를 사용해서 그래프에 범례 표시
- plot() 함수에 label 파라미터 값으로 삽입

```
# plot() 함수의 label 매개변수에 'Square(제곱)' 문자열 입력
# legend() 함수를 사용해서 그래프에 범례 표시
>>> plt.plot([1, 2, 3, 4], [1, 4, 9, 16], label = 'Square')
>>> plt.xlabel('X-Label')
>>> plt.ylabel('Y-Label')
>>> plt.legend()

>>> plt.show()
```

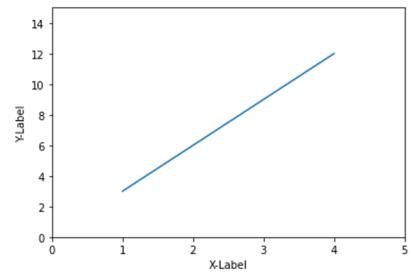


05. Matplotlib 축 범위 설정

- xlim() : X축이 표시되는 범위 지정 [xmin, xmax]
- ylim() : Y축이 표시되는 범위 지정 [ymin, ymax]
- axis() : X, Y축이 표시되는 범위 지정 [xmin, xmax, ymin, ymax]
- 입력 값이 없으면 데이터에 맞게 자동으로 범위 지정

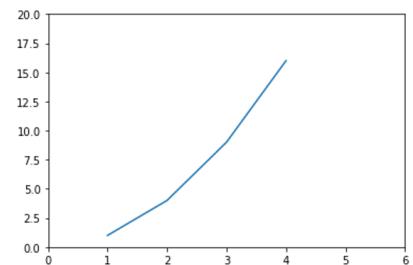
```
# X축의 범위: [xmin, xmax] = [0, 5]
# Y축의 범위: [ymin, ymax] = [0, 20]
>>> plt.plot([1, 2, 3, 4], [3, 6, 9, 12])
>>> plt.xlabel('X-Label')
>>> plt.ylabel('Y-Label')
>>> plt.xlim([0, 5])
>>> plt.ylim([0, 15])

>>> plt.show()
```



```
# axis() : X, Y축이 표시되는 범위 지정 [xmin, xmax, ymin, ymax]
>>> plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
>>> plt.axis([0, 6, 0, 20])

>>> plt.show()
```



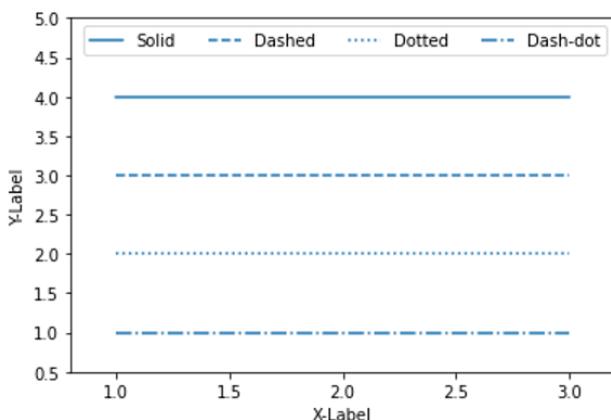
06. Matplotlib 선 종류 설정

- plot() 함수의 포맷 문자열 사용
 - '-' (Solid), '--' (Dashed), ':' (Dotted), '-.' (Dash-dot)
- plot() 함수의 linestyle 파라미터 값으로 삽입
 - '-' (solid), '--' (dashed), ':' (dotted), '-.' (dashdot)
- 튜플을 사용하여 선의 종류 커스터마이즈
 - (0, (1, 1)) [dotted], (0, (5, 5)) [dashed], (0, (3, 5, 1, 5)) [dashdotted]

```
# plot() 함수의 포맷 문자열 사용
# plot() 함수의 linestyle 값으로 삽입
# 축 이름, 범위, 범례 설정
>>> plt.plot([1, 2, 3], [4, 4, 4], '-.', color='C0', label='Solid')
>>> plt.plot([1, 2, 3], [3, 3, 3], '--', color='C0', label='Dashed')
>>> plt.plot([1, 2, 3], [2, 2, 2], linestyle='dotted', color='C0', label='Dotted')
>>> plt.plot([1, 2, 3], [1, 1, 1], linestyle='dashdot', color='C0', label='Dash-dot')

>>> plt.xlabel('X-Label')
>>> plt.ylabel('Y-Label')
>>> plt.axis([0.8, 3.2, 0.5, 5.0])
>>> plt.legend(loc='upper right', ncol=4)

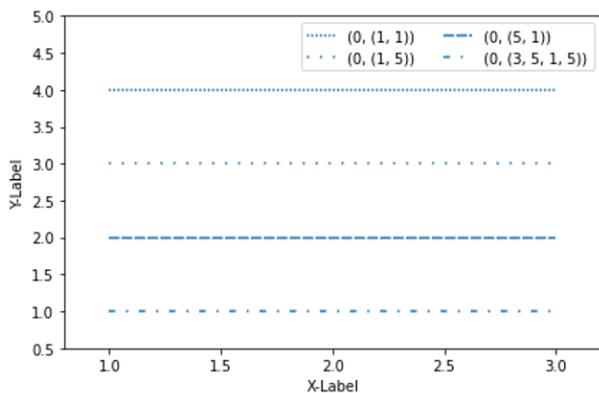
>>> plt.show()
```



```
# 튜플을 사용하여 선의 종류 커스터마이즈
# 축 이름, 범위, 범례 설정
# tuple(offset, (on_off_seq))
# offset : 플롯의 간격 띄우기를 조정
>>> plt.plot([1, 2, 3], [4, 4, 4], linestyle=(0, (1, 1)), color='C0', label='(0, (1, 1))')
>>> plt.plot([1, 2, 3], [3, 3, 3], linestyle=(0, (1, 5)), color='C0', label='(0, (1, 5))')
>>> plt.plot([1, 2, 3], [2, 2, 2], linestyle=(0, (5, 1)), color='C0', label='(0, (5, 1))')
>>>
plt.plot([1, 2, 3], [1, 1, 1], linestyle=(0, (3, 5, 1, 5)), color='C0', label='(0, (3, 5, 1, 5))')
')

>>> plt.xlabel('X-Label')
>>> plt.ylabel('Y-Label')
>>> plt.axis([0.8, 3.2, 0.5, 5.0])
>>> plt.legend(loc='upper right', ncol=2)
>>> plt.tight_layout()

>>> plt.show()
```



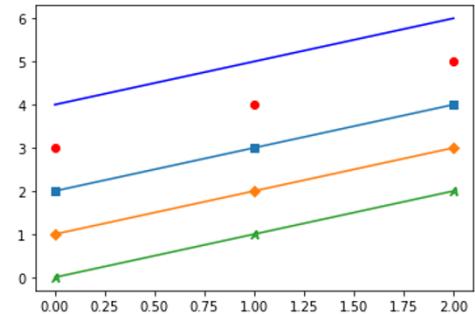
07. Matplotlib 마커 설정

- 기본적으로는 실선 마커
- plot() 함수의 포맷 문자열 (Format string)을 사용해서 마커 지정
 - 'ro'는 빨간색 ('red')의 원형 ('circle') 마커를 의미
 - 'k^'는 검정색 ('black')의 삼각형 ('triangle') 마커를 의미
- plot() 함수의 marker 파라미터 값으로 삽입
 - 's'(square), 'D'(diamond)

marker	symbol	description	marker	symbol	description	marker	symbol	description	marker	symbol	description
"."	●	point	"1"	▼	tri_down	"P"	✚	plus (filled)	"D"	◆	diamond
"_"	.	pixel	"2"	▲	tri_up	"*"	★	star	"d"	◆	thin_diamond
"o"	●	circle	"3"	◀	tri_left	"h"	●	hexagon1	"l"		vline
"v"	▼	triangle_down	"4"	▶	tri_right	"H"	●	hexagon2	"_"	—	hline
"^"	▲	triangle_up	"8"	●	octagon	"+"	+	plus			
"<"	◀	triangle_left	"s"	■	square	"x"	✗	x			
">"	▶	triangle_right	"p"	◆	pentagon	"x"	❖	x (filled)			

출처 링크: https://matplotlib.org/stable/api/markers_api.html#module-matplotlib.markers

```
# 'b' blue, 'ro' red+circle
# 's' square, 'D' diamond
# '$문자$' 문자 마커
>>> plt.plot([4, 5, 6], "b")
>>> plt.plot([3, 4, 5], "ro")
>>> plt.plot([2, 3, 4], marker="s")
>>> plt.plot([1, 2, 3], marker="D")
>>> plt.plot([0, 1, 2], marker='$A$')
>>> plt.show()
```



08. Matplotlib 색상 설정

- plot() 함수의 포맷 문자열 (Format string)을 사용해서 색상 지정
- plot() 함수의 color 파라미터 값으로 삽입
- 다양한 색상 링크 (https://matplotlib.org/stable/gallery/color/named_colors.html?highlight=css%20color) 참고

Colors	
character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

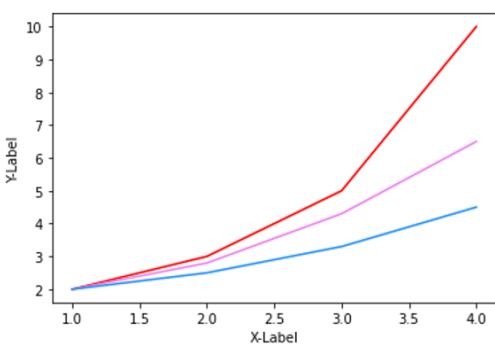


- plot() 함수의 포맷 문자열 (Format string)을 사용해서 색상 지정
- plot() 함수의 color 파라미터 값으로 삽입

```
# 'r' red, 'violet', 'dodgerblue'
>>> plt.plot([1, 2, 3, 4], [2.0, 3.0, 5.0, 10.0], 'r')
>>> plt.plot([1, 2, 3, 4], [2.0, 2.8, 4.3, 6.5], color = 'violet')
>>> plt.plot([1, 2, 3, 4], [2.0, 2.5, 3.3, 4.5], color = 'dodgerblue')

>>> plt.xlabel('X-Label')
>>> plt.ylabel('Y-Label')

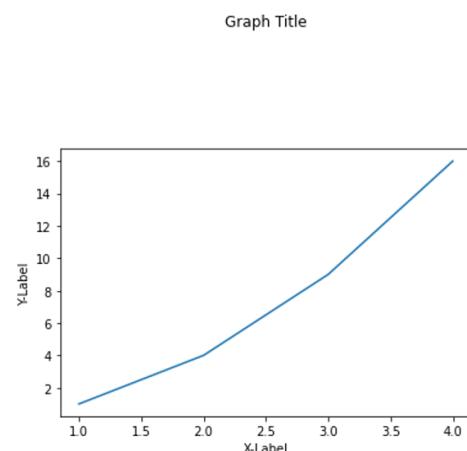
>>> plt.show()
```



09. Matplotlib 타이틀 설정

- title() 함수를 이용하여 타이틀 설정
- title() 함수의 loc 파라미터 값으로 위치 설정
 - loc 파라미터 : {'left', 'center', 'right'}
- title() 함수의 pad 파라미터 값으로 타이틀과 그래프와의 간격(포인트 단위) 설정

```
>>> plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
>>> plt.xlabel('X-Label')
>>> plt.ylabel('Y-Label')
>>> plt.title('Graph Title', loc='center', pad=100)
```



“Selena's Comment!”

데이터 시각화에서 타이틀은 절대 빼먹으시면 안 됩니다!
사람에게 이름이 없는 것과 같아요. :)

10. Matplotlib 눈금 표시

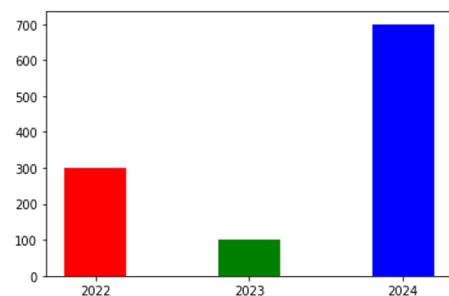
- xticks(), yticks() 함수는 각각 X축, Y축에 눈금 설정
- xticks(), yticks() 함수의 label 파라미터 값으로 눈금 레이블 설정

11. Matplotlib 막대 그래프

- bar() 함수 이용하여 막대 그래프 시각화
- bar() 함수의 color 파라미터 값으로 색상 설정
- bar() 함수의 width 파라미터 값으로 막대 폭 설정

```
# years는 X축에 표시될 연도이고, values는 막대 그래프의 y 값
# xticks(x, years) : x축의 눈금 레이블
#   '2022', '2023', '2024' 순서대로 설정
# color와 width로 막대 그래프 파라미터 설정
>>> x = [1, 2, 3]
>>> years = ['2022', '2023', '2024']
>>> values = [300, 100, 700]

>>> plt.bar(x, values, color=['r', 'g', 'b'], width=0.4)
>>> plt.xticks(x, years)
>>> plt.show()
```



12. Matplotlib 산점도

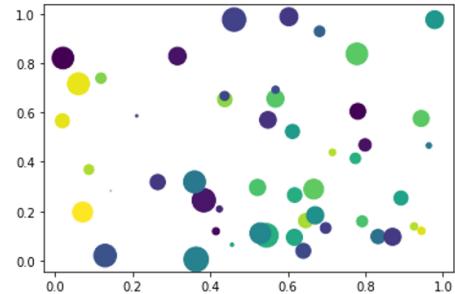
- scatter() 함수 이용하여 산점도 시각화
- scatter() 함수의 color 파라미터 값으로 마커의 색상 설정
- scatter() 함수의 size 파라미터 값으로 마커의 크기 설정

```
# NumPy의 random 모듈의 rand 함수를 통해 숫자 랜덤하게 생성
# color와 size로 산점도 파라미터 설정
>>> import NumPy as np

>>> np.random.seed(0)

>>> n = 50
>>> x = np.random.rand(n)
>>> y = np.random.rand(n)
>>> size = (np.random.rand(n) * 20)**2
>>> colors = np.random.rand(n)

>>> plt.scatter(x, y, s=size, c=colors)
>>> plt.show()
```

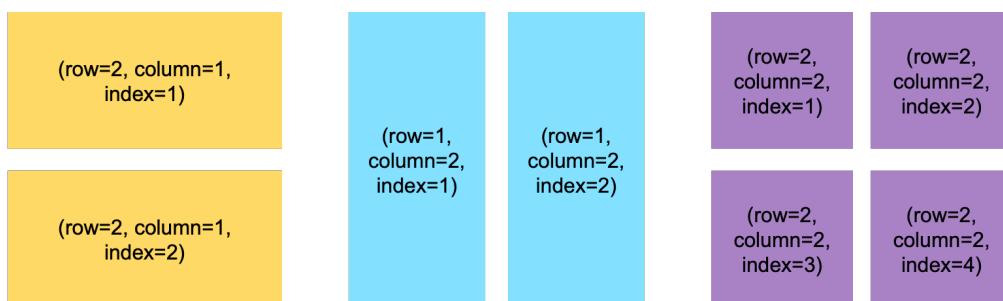


13. Matplotlib 다양한 그래프 종류

- matplotlib.pyplot.bar() : 막대 그래프
- matplotlib.pyplot.barch() : 수평 막대 그래프
- matplotlib.pyplot.scatter() : 산점도
- matplotlib.pyplot.hist() : 히스토그램
- matplotlib.pyplot.errorbar() : 에러바
- matplotlib.pyplot.pie() : 파이 차트
- matplotlib.pyplot.matshow() : 히트맵

14. Matplotlib subplot 이용한 여러 개 그래프 시각화

- subplot() 함수는 영역을 나눠 여러 개의 그래프 시각화
- plt.subplot(row, column, index)



- tight_layout() 함수는 모서리와 서브플롯의 모서리 사이의 여백(padding)을 설정

“Selena's Comment!”

데이터 시각화에서 효율적으로 시각화를 시킨다면 여러 개의 그래프를 한번에 보여주는 거겠죠?
이런 스킬들에 익숙해진다면 회사에서 능력자가 되실 수 있습니다!

```
# linspace : 몇등분할지 생각하면 디폴트는 50
# np.linspace(0, 10) : 0부터 10까지 50등분한 결과를 배열로 반환
>>> x1 = np.linspace(0, 10)
>>> x1

array([ 0.        ,  0.20408163,  0.40816327,  0.6122449 ,  0.81632653,
       1.02040816,  1.2244898 ,  1.42857143,  1.63265306,  1.83673469,
       2.04081633,  2.24489796,  2.44897959,  2.65306122,  2.85714286,
       3.06122449,  3.26530612,  3.46938776,  3.67346939,  3.87755102,
       4.08163265,  4.28571429,  4.48979592,  4.69387755,  4.89795918,
       5.10204082,  5.30612245,  5.51020408,  5.71428571,  5.91836735,
       6.12244898,  6.32653061,  6.53061224,  6.73469388,  6.93877551,
       7.14285714,  7.34693878,  7.55102041,  7.75510204,  7.95918367,
       8.16326531,  8.36734694,  8.57142857,  8.7755102 ,  8.97959184,
       9.18367347,  9.3877551 ,  9.59183673,  9.79591837, 10.        ])
```

```
# linspace : 몇등분할지 생각하면 디폴트는 50
# np.linspace(0, 4) : 0부터 4까지 50등분한 결과를 배열로 반환
>>> x2 = np.linspace(0, 4)
>>> x2

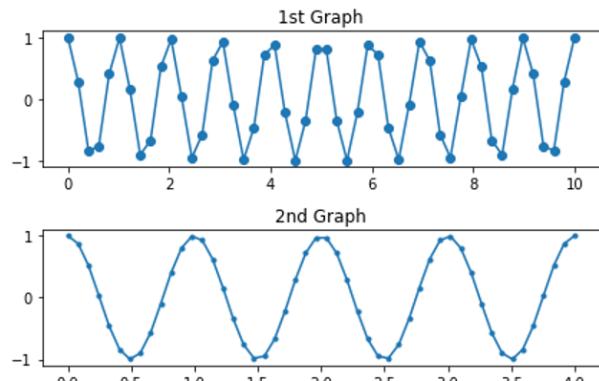
array([0.        ,  0.08163265,  0.16326531,  0.24489796,  0.32653061,
       0.40816327,  0.48979592,  0.57142857,  0.65306122,  0.73469388,
       0.81632653,  0.89795918,  0.97959184,  1.06122449,  1.14285714,
       1.2244898 ,  1.30612245,  1.3877551 ,  1.46938776,  1.55102041,
       1.63265306,  1.71428571,  1.79591837,  1.87755102,  1.95918367,
       2.04081633,  2.12244898,  2.20408163,  2.28571429,  2.36734694,
       2.44897959,  2.53061224,  2.6122449 ,  2.69387755,  2.7755102 ,
       2.85714286,  2.93877551,  3.02040816,  3.10204082,  3.18367347,
       3.26530612,  3.34693878,  3.42857143,  3.51020408,  3.59183673,
       3.67346939,  3.75510204,  3.83673469,  3.91836735, 4.        ])
```

```
# y값은 역동적인 그래프를 위한 np.cosine 함수 이용
# np.pi 함수로 원주율(파이) 값 사용
>>> y1 = np.cos(2 * np.pi * x1)
>>> y2 = np.cos(2 * np.pi * x2)

# subplot
# nrows=2, ncols=1, index=1
>>> plt.subplot(2, 1, 1)
>>> plt.plot(x1, y1, 'o-')
>>> plt.title('1st Graph')

# subplot
# nrows=2, ncols=1, index=2
>>> plt.subplot(2, 1, 2)
>>> plt.plot(x2, y2, '.-')
>>> plt.title('2nd Graph')

# tight_layout() 함수는 모서리와 서브플롯의 모서리 사이
# 의 여백(padding)을 설정
>>> plt.tight_layout()
>>> plt.show()
```

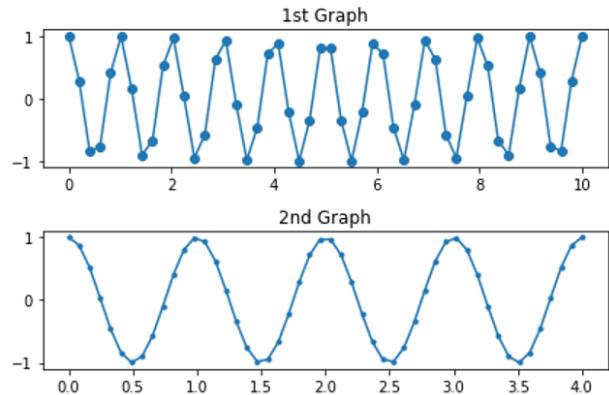


```
# tight_layout() 함수 지운 결과 비교
# tight_layout() 함수는 모서리와 서브플롯의 모서리 사이
# 의 여백(padding)을 설정
>>> y1 = np.cos(2 * np.pi * x1)
>>> y2 = np.cos(2 * np.pi * x2)

>>> plt.subplot(2, 1, 1)
>>> plt.plot(x1, y1, 'o-')
>>> plt.title('1st Graph')

>>> plt.subplot(2, 1, 2)
>>> plt.plot(x2, y2, '.-')
>>> plt.title('2nd Graph')

>>> plt.show()
```



“Selena’s Comment!”

`tight_layout` 함수가 있을 때와 없을 때의 차이를 보여주고 있어요.

여백이 사라져서 그래프가 안 예쁜 걸 확인할 수 있어요.

15. Matplotlib 한 좌표 평면 위에 다른 종류의 그래프 시각화

- `matplotlib.pyplot.bar()` : 막대 그래프
- `plt.subplots()` 함수는 한 좌표 평면 위에 여러 개 그래프 시각화 가능
- `plt.subplots()` 함수의 디폴트 파라미터는 1이며 즉 `plt.subplots(nrows=1, ncols=1)` 의미
- `plt.subplots()` 함수는 `figure`와 `axes` 값을 반환
- `figure`
 - 전체 subplot 의미
 - 서브플롯 안에 몇 개의 그래프가 있던지 상관없이 그걸 담는 전체 사이즈를 의미
- `axe`
 - 전체 중 낱낱개 의미
 - ex) 서브플롯 안에 2개(a1,a2)의 그래프가 있다면 a1, a2 를 일컬음
- `twinx()` 함수는 `ax1`과 축을 공유하는 새로운 Axes 객체 생성

```

>>> import matplotlib.pyplot as plt
>>> import NumPy as np

# x는 X축에 표시될 연도이고, y1, y2는 y 값
>>> x = ['2022', '2023', '2024']
>>> y1 = np.array([1, 7, 14])
>>> y2 = np.array([1, 3, 9])

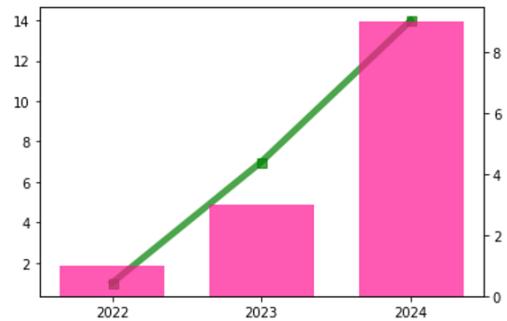
# plt.subplots() 함수는 여러 개 그래프를 한 번에 가능, 객체 생성
# plt.subplots(nrows=1, ncols=1) = plt.subplots()
>>> fig, ax1 = plt.subplots()

# -s(solid line style + square marker), alpha(투명도)
>>> ax1.plot(x, y1, '-s',
             color='green', markersize=7, linewidth=5, alpha=0.7)

# .twinx() 함수는 ax1과 축을 공유하는 새로운 Axes 객체 생성
>>> ax2 = ax1.twinx()
>>>
ax2.bar(x, y2, color='deeppink', alpha=0.7, width=0.7)

>>> plt.show()

```



“Selena’s Comment!”

한 좌표 평면 위에 여러 개 그래프를 나타내는 건 제가 정말 좋아하는 시각화 방법 중에 하나예요. 한 화면에 여러 정보를 보여주는 건 한눈에 보기에도 좋고 효율적이잖아요~!

저는 항상 효율을 중요시하면서 일을 합니다! :)

한 좌표 평면 위에 y축이 2개입니다. 그래서 결과를 비교하기도 좋아요.
잊지 말고 꼭 사용해 보세요!

STEP 13-2. 데이터 시각화

“Selena’s Comment!”

Matplotlib으로 시각화 기본기를 다렸다면 이제 응용을 해보도록 할게요.

Seaborn 라이브러리를 통해 고급 인터페이스를 구현해 볼 거예요.

01. Seaborn 기본 개념

- seaborn은 matplotlib 기반의 시각화 라이브러리
- 유익한 통계 기반 그래픽을 그리기 위한 고급 인터페이스를 제공
- Seaborn 패키지 추가
- 패키지가 없는 경우, 설치 명령어 : pip install seaborn

```
# seaborn 불러와서 sns로 사용
import seaborn as sns
```

02. Seaborn 데이터 불러오기

- seaborn 라이브러리에서 제공하는 titanic 데이터 불러오기
- seaborn의 load_dataset() 함수를 이용
- seaborn 내장 데이터 사이트(<https://github.com/mwaskom/seaborn-data>) 참고

anscombe.csv	Add anscombe dataset
attention.csv	Add attention dataset
brain_networks.csv	Add brain networks dataset
car_crashes.csv	Add 538 car crash dataset
diamonds.csv	Add diamonds dataset
dots.csv	Add dots dataset
exercise.csv	Add exercise dataset
flights.csv	Add flights dataset

geyser.csv	Add geyser dataset
iris.csv	Add iris dataset
mpg.csv	Add mpg dataset
penguins.csv	Change culmen to bill in penguins dataset
planets.csv	Add planets dataset
taxis.csv	Add green taxis to the taxis dataset
tips.csv	Add tips dataset
titanic.csv	Update titanic dataset to remove index variable

```
# titanic 데이터 불러오기
>>> titanic = sns.load_dataset('titanic')
```

```
# titanic 데이터 내용 확인
# .head() : 데이터의 상단 5개 행 출력
>>> titanic.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	TRUE	NaN	Southampton	no	FALSE
1	1	1	female	38.0	1	0	71.2833	C	First	woman	FALSE	C	Cherbourg	yes	FALSE
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	FALSE	NaN	Southampton	yes	TRUE
3	1	1	female	35.0	1	0	53.1000	S	First	woman	FALSE	C	Southampton	yes	FALSE
4	0	3	male	35.0	0	0	8.0500	S	Third	man	TRUE	NaN	Southampton	no	TRUE

```
# .info() :데이터에 대한 전반적인 정보 제공
# 행과 열의 크기, 컬럼명, 컬럼별 결측치, 컬럼별 데이터 타입
>>> print(titanic.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   survived    891 non-null    int64  
 1   pclass      891 non-null    int64  
 2   sex         891 non-null    object  
 3   age         714 non-null    float64 
 4   sibsp       891 non-null    int64  
 5   parch       891 non-null    int64  
 6   fare         891 non-null    float64 
 7   embarked    889 non-null    object  
 8   class        891 non-null    category
 9   who          891 non-null    object  
 10  adult_male  891 non-null    bool   
 11  deck         203 non-null    category
 12  embark_town 889 non-null    object  
 13  alive        891 non-null    object  
 14  alone        891 non-null    bool  
dtypes: bool2., category2., float642., int644., object5.
memory usage: 80.6+ KB
None
```

03. Seaborn 선형 회귀선 있는 산점도

- regplot() 함수 : 선형 회귀선이 있는 산점도 시각화
- regplot() 함수의 파라미터
 - x축 변수
 - y축 변수
 - 데이터셋
 - axe 객체
 - fit_reg : 선형 회귀선 표시 여부
- 선형 회귀선
 - 간단한 선형 데이터 집합에 사용되는 가장 적합한 직선(= 추세선)
 - 데이터를 시간 축으로 봤을 때, 데이터의 값이 장기적으로 어떻게 변하는지 직선으로 표현한 것

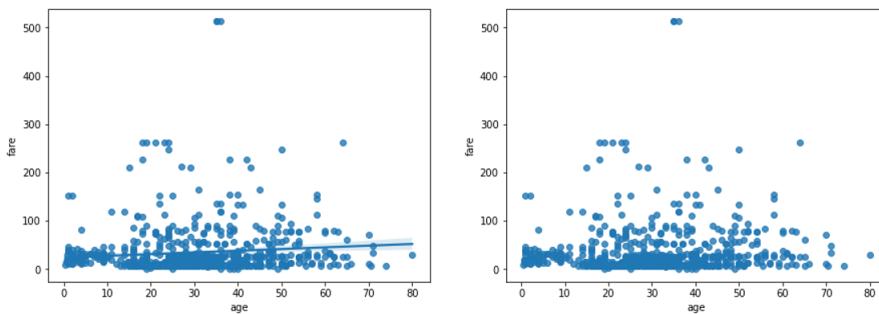
```
>>> import matplotlib.pyplot as plt

# 그래프 객체 생성 (figure에 2개의 서브 플롯을 생성)
>>> fig = plt.figure(figsize=(15,5))
>>> ax1 = fig.add_subplot(1, 2, 1)
>>> ax2 = fig.add_subplot(1, 2, 2)

# 산점도에 선형회귀선 표시(fit_reg=True)
# x축 변수, y축 변수, 데이터 셋, axe 객체(1번째 그래프)
>>> sns.regplot(x='age', y='fare', data=titanic, ax=ax1)

# 산점도에 선형회귀선 미표시(fit_reg=False)
# x축 변수, y축 변수, 데이터 셋, axe 객체(2번째 그래프)
>>> sns.regplot(x='age', y='fare', data=titanic, ax=ax2, fit_reg=False)

>>> plt.show()
```



04. Seaborn 히스토그램과 커널 밀도 그래프

- distplot() 함수 : 히스토그램과 커널 밀도 그래프 시각화
- distplot() 함수의 파라미터
 - 분포를 그릴 데이터 변수
 - hist : True는 히스토그램 표시, False는 히스토그램 표시 안 함
 - kde : True는 커널 밀도 그래프 표시, False는 커널 밀도 그래프 표시 안 함
 - axe 객체
- histplot() 함수 : 히스토그램(하나의 변수 데이터의 분포를 확인할 때 사용하는 함수)
- kdeplot() 함수 : 커널 밀도 그래프(그래프와 x축 사이의 면적이 1이 되도록 그리는 밀도 함수)

```
# 그래프 객체 생성 (figure에 3개의 서브 플롯을 생성)
>>> fig = plt.figure(figsize=(15, 5))
>>> ax1 = fig.add_subplot(1, 3, 1)
>>> ax2 = fig.add_subplot(1, 3, 2)
>>> ax3 = fig.add_subplot(1, 3, 3)

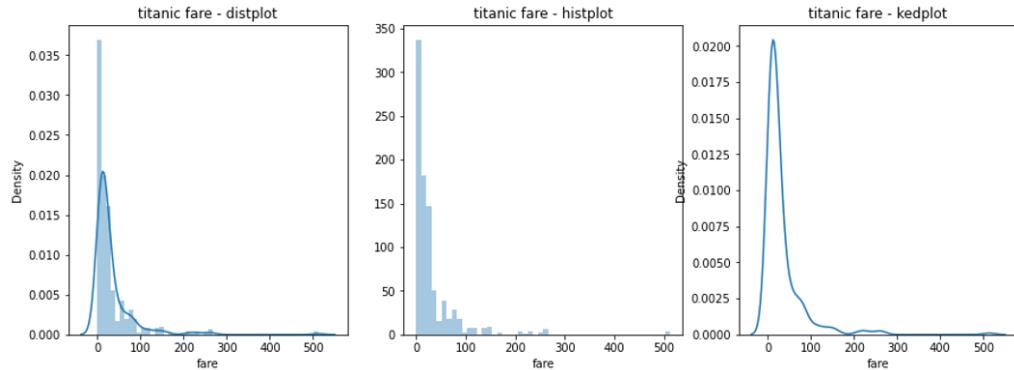
# distplot
# 히스토그램과 커널 밀도 그래프 표시
# 분포를 그릴 데이터 변수, axe 객체(1번째 그래프)
>>> sns.distplot(titanic['fare'], ax=ax1)

# histplot
# 분포를 그릴 데이터 변수, 커널 밀도 그래프 표시 안 함, axe 객체(2번째 그래프)
#sns.histplot(x='fare', data=titanic, ax=ax2)
>>> sns.distplot(titanic['fare'], kde=False, ax=ax2)
```

```
# kdeplot
# 분포를 그릴 데이터 변수, 히스토그램 표시 안 함, axe 객체(2번째 그래프)
#sns.kdeplot(x='fare', data=titanic, ax=ax3)
>>> sns.distplot(titanic['fare'], hist=False, ax=ax3)

# 차트 제목 표시
>>> ax1.set_title('titanic fare - distplot')
>>> ax2.set_title('titanic fare - histplot')
>>> ax3.set_title('titanic fare - kedplot')

>>> plt.show()
```



05. Seaborn 범주형 데이터의 산점도

- stripplot() 함수 : 데이터 포인트가 중복되어 범주별 분포를 시각화
- stripplot() 함수의 파라미터
 - x축 변수
 - y축 변수
 - 데이터 셋
 - axe 객체
 - hue : 특정 열 데이터로 색상을 구분하여 출력
- swarmplot() 함수 : 데이터의 분산까지 고려하여 데이터 포인트가 서로 중복되지 않도록 시각화
- swarmplot() 함수의 파라미터
 - x축 변수
 - y축 변수
 - 데이터 셋
 - axe 객체
 - hue : 특정 열 데이터로 색상을 구분하여 출력

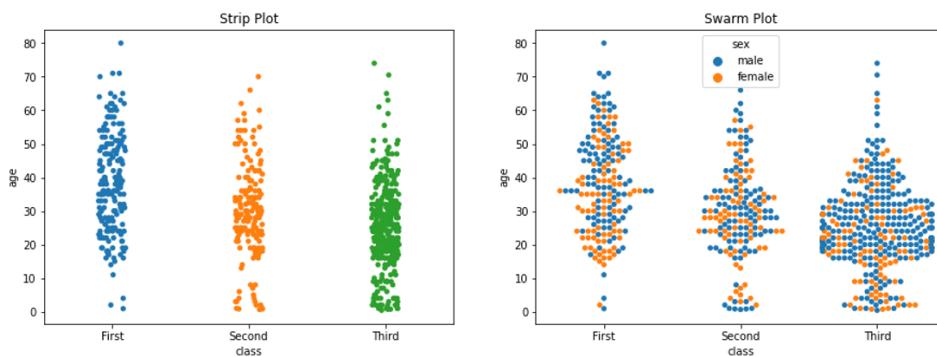
```
# 그래프 객체 생성 (figure에 2개의 서브 플롯을 생성)
>>> fig = plt.figure(figsize=(15, 5))
>>> ax1 = fig.add_subplot(1, 2, 1)
>>> ax2 = fig.add_subplot(1, 2, 2)

# 이산형 변수의 분포 - 데이터 분산 미고려
# x축 변수, y축 변수, 데이터 셋, axe 객체(1번째 그래프)
>>> sns.stripplot(x='class', y='age', data=titanic, ax=ax1)

# 이산형 변수의 분포 - 데이터 분산 고려 (중복 X)
# x축 변수, y축 변수, 데이터 셋, axe 객체(2번째 그래프), 성별로 색상 구분
>>> sns.swarmplot(x='class', y='age', data=titanic, ax=ax2, hue='sex')

# 차트 제목 표시
>>> ax1.set_title('Strip Plot')
>>> ax2.set_title('Swarm Plot')

>>> plt.show()
```



“Selena’s Comment!”

Selena 강사가 정말 좋아하는 그래프가 나왔어요!

그림만 봐도 웅장한 차이가 느껴지지 않으시나요?

strip plot은 단순히 클래스와 나이에 대한 정보만 얻고 있다면

swarm plot은 클래스와 나이에 대한 정보에서 더 나아가 성별과 나이의 분포 정보도 얻을 수 있어요.

데이터 시각화는 아는 만큼 표현할 수 있다고 생각해요.

수강생 여러분들도 배우실 때 제대로 연습해 보세요~!

06. Seaborn 빈도 그래프

- countplot() 함수 : 각 범주에 속하는 데이터의 개수를 막대 그래프 시각화
- countplot() 함수의 파라미터
 - x축 변수
 - palette
 - 데이터 셋
 - axe 객체
 - hue : 특정 열 데이터로 색상을 구분하여 출력

```

# 그래프 객체 생성 (figure에 3개의 서브 플롯을 생성)
>>> fig = plt.figure(figsize=(15, 5))
>>> ax1 = fig.add_subplot(1, 3, 1)
>>> ax2 = fig.add_subplot(1, 3, 2)
>>> ax3 = fig.add_subplot(1, 3, 3)

# class별 인원 파악
# x축 변수, 데이터 셋, axe 객체(1번째 그래프)
>>> sns.countplot(x='class', palette='Set1', data=titanic, ax=ax1)

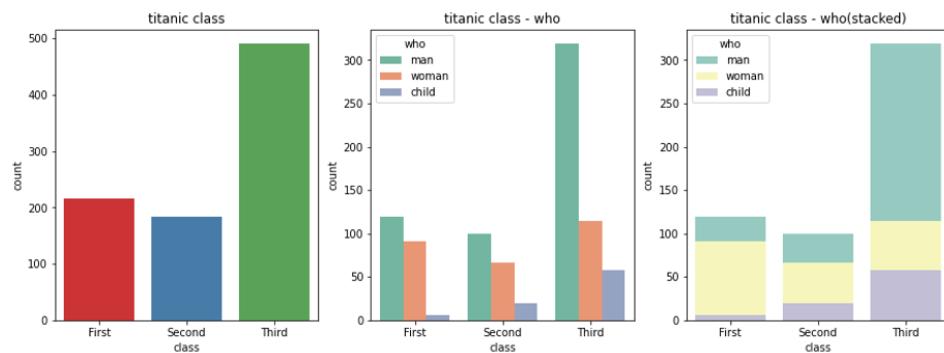
# hue 옵션에 'who' 추가
# x축 변수, 데이터 셋, axe 객체(2번째 그래프)
>>> sns.countplot(x='class', hue='who', palette='Set2', data=titanic, ax=ax2)

# dodge=False 옵션 추가 (축 방향으로 분리하지 않고 누적 그래프 출력)
# x축 변수, hue, 데이터 셋, axe 객체(3번째 그래프)
>>> sns.countplot(x='class', hue='who', palette='Set3', dodge=False, data=titanic, ax=ax3)

# 차트 제목 표시
>>> ax1.set_title('titanic class')
>>> ax2.set_title('titanic class - who')
>>> ax3.set_title('titanic class - who(stacked)')

>>> plt.show()

```



07. Seaborn 조인트 그래프

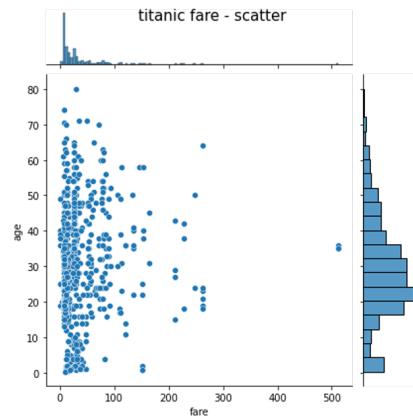
- jointplot() 함수 : 산점도를 기본으로 표시, x-y축에 각 변수에 대한 히스토그램을 동시에 시각화
- jointplot() 함수의 파라미터
 - x축 변수
 - y축 변수
 - 데이터 셋
 - kind = 'reg' : 선형 회귀선 추가
 - kind = 'hex' : 육각 산점도 추가
 - kind = 'kde' : 커널 밀집 그래프 추가

```

# 조인트 그래프 - 산점도(기본값)
# x축 변수, y축 변수, 데이터 셋
>>> j1 = sns.jointplot(x='fare', y='age', data = titanic)
>>> j1.fig.suptitle('titanic fare - scatter', size=15)

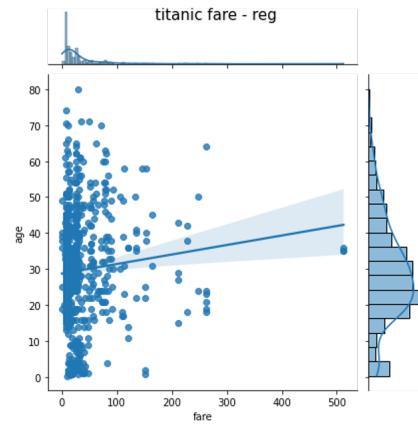
>>> plt.show()

```



```
# 조인트 그래프 - 회귀선(kind = 'reg')
# x축 변수, y축 변수, 데이터셋
>>>
j2 = sns.jointplot(x='fare', y='age', kind='reg', data=titanic)
>>> j2.fig.suptitle('titanic fare - reg', size=15)

>>> plt.show()
```

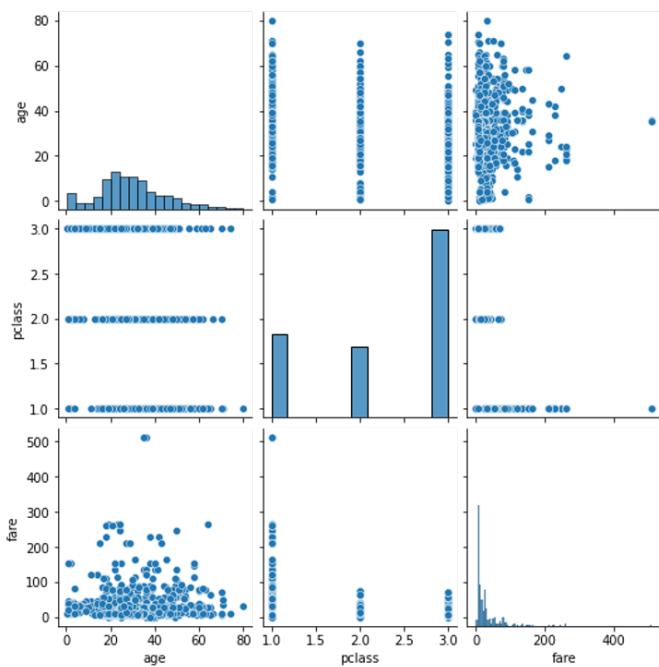


08. Seaborn 관계 그래프

- pairplot() 함수
 - 인자로 전달되는 데이터프레임의 열(변수)을 두 개씩 짹 지을 수 있는 모든 조합에 대해서 표현
 - 열은 정수/실수형이어야 함
 - 3개의 열이라면 3행 x 3열의 크기로 모두 9개의 그리드 생성
 - 각 그리드의 두 변수 간의 관계를 나타내는 그래프를 하나씩 그림
 - 같은 변수끼리 짹을 이루는 대각선 방향으로는 히스토그램 시각화
 - 서로 다른 변수 간에는 산점도 시각화

```
# titanic 데이터셋 중에서 분석 데이터 선택하기
>>> titanic_pair = titanic[['age', 'pclass', 'fare']]

# 3개의 열이라면 3행 x 3열의 크기로 모두 9개의 그리드 생성
# 각 그리드의 두 변수 간의 관계를 나타내는 그래프를 하나씩 그림
# 같은 변수끼리 짹을 이루는 대각선 방향으로는 히스토그램 시각화
# 서로 다른 변수 간에는 산점도 시각화
>>> sns.pairplot(titanic_pair)
```



“Selena's Comment!”

Seaborn 라이브러리를 통해 고급 시각화를 배워보았어요!

Selena 강사가 실무에서 자주 쓰는 그래프들로 구성해 보았어요.

아는 만큼 구현할 수 있는 게 시각화입니다!

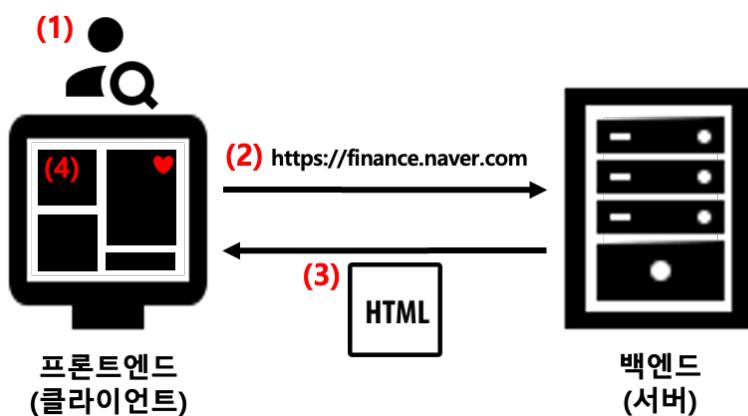
배우시는 과정에서 최대한 많은 시각화를 구현해 보세요.

STEP 14. 웹 데이터 수집

01. 웹 크롤링 개념 다지기

- 웹 크롤링은 인터넷 상에 존재하는 데이터를 자동으로 수집하는 행위를 의미
- 데이터 분석가에게 데이터를 탐색하고 원하는 조건에 맞는 데이터를 직접 수집하고 저장까지 하기 위한 목적으로 사용

1. 웹 페이지 정보 가져오기
 - 파이썬 Requests 라이브러리 사용
2. HTML 소스를 파싱(분석)하여 원하는 정보 얻기
 - 파이썬 BeautifulSoup 라이브러리 사용
1. 사용자는 브라우저로 접속하고 싶은 주소(url) 입력
2. 브라우저가 해당 주소의 서버에게 “페이지 구성 정보를 줘”라고 요청(request)
3. 웹 서버는 구성에 필요한 정보를 코드(html) 형태로 전달(response)
4. 브라우저는 서버가 전달 준 정보(html)를 해석해서 사용자 화면에 보여줌



- HTML 및 XML에서 데이터를 쉽게 처리하는 파이썬 라이브러리
- HTML은 태그로 이루어져 있고, 수많은 공백과 변화하는 소스들 때문에 오류가 있을 가능성이 높지만 BeautifulSoup을 이용하면 이러한 오류를 잡아서 고친 후 데이터를 전달해줌
- 설치 방법
 - `pip install beautifulsoup4`
- 사용 방법
 - `from bs4 import BeautifulSoup`
- <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>에서 BeautifulSoup에 대한 다양한 설명 참고

02. BeautifulSoup 기본 개념

- BeautifulSoup 패키지 추가
- 패키지가 없는 경우, 설치 명령어 : pip install beautifulsoup4

```
# BeautifulSoup 불러오기
from bs4 import BeautifulSoup
```

03. BeautifulSoup HTML 코드 작성

- html이라는 변수에 간단한 html 코드 저장
- html 코드는 """ 사이에 입력

```
# HTML 코드 작성
>>> html = """
<html>
  <body>
    <h1 id = 'title'>Selena 파이썬 라이브러리 활용!</h1>
    <p id = 'body'>오늘의 주제는 웹 데이터 수집</p>
    <p class = 'scraping'>삼성전자 일별 시세 불러오기</p>
    <p class = 'scraping'>이해 쑥쑥 Selena 수업!</p>
  </body>
</html>
"""
```

04. BeautifulSoup HTML 파싱

- soup = BeautifulSoup(html, 'html.parser')
 - html을 파이썬에서 읽을 수 있게 파싱(파이썬 객체로 변환)
 - html이라는 변수에 저장한 html 소스코드를 .parser를 붙여 변환
 - parser는 파이썬의 내장 메소드

```
# BeautifulSoup 함수를 이용하여 soup 객체 생성
# html이라는 변수에 저장한 html 소스코드를 .parser를 붙여 변환
>>> soup = BeautifulSoup(html, 'html.parser')
```

05. BeautifulSoup 데이터를 텍스트로 반환

- for text in soup:


```
print(text)
```
- soup : soup의 데이터를 모두 가져와서 텍스트로 반환
- soup.contents : soup의 데이터를 모두 가져와서 리스트로 반환
- soup.stripped_strings : 공백도 함께 제거하여 텍스트로 반환

5-1. soup

- soup : soup의 데이터를 모두 가져와서 텍스트로 반환
- soup.contents : soup의 데이터를 모두 가져와서 리스트로 반환

```
# soup의 데이터를 모두 가져와서 텍스트로 반환
>>> for text in soup:
    print(text)

<html>
<body>
<h1 id="title">Selena 파이썬 라이브러리 활용!</h1>
<p id="body">오늘의 주제는 웹 데이터 수집</p>
<p class="scraping">삼성전자 일별 시세 불러오기</p>
<p class="scraping">이해 쑥쑥 Selena 수업!</p>
</body>
<html>
</html></html>
```

5-2. soup.stripped_strings

- soup.stripped_strings : 공백도 함께 제거하여 텍스트로 반환

```
# soup의 데이터를 모두 가져와서
# 공백도 함께 제거하여 텍스트로 반환
>>> for stripped_text in soup.stripped_strings:
    print(stripped_text)

Selena 파이썬 라이브러리 활용!
오늘의 주제는 웹 데이터 수집
삼성전자 일별 시세 불러오기
이해 쑥쑥 Selena 수업!
```

06. BeautifulSoup Find 함수

- scraping = soup.find(class_='scraping')
scraping.string

6-1. find()

- find 함수는 id, class, element 등을 검색 가능
 - find : 조건에 해당하는 첫 번째 정보만 검색
 - 클래스 이름을 알 경우, class_ 형태로 사용

```
# find 함수
# id 값이 'title'인 조건에 해당하는 첫 번째 정보만 검색
>>> title = soup.find(id='title')
>>> print(title)

<h1 id="title">Selena 파이썬 라이브러리 활용!</h1>
```

```
# find 함수
# class 값이 'scraping'인 조건에 해당하는 첫 번째 정보만 검색
# 클래스 이름을 알 경우, class_ 형태로 사용
>>> scraping = soup.find(class_='scraping')
>>> print(scraping)

<p class="scraping">삼성전자 일별 시세 불러오기</p>
```

6-2. find_all()

- **find_all**: 조건에 해당하는 모든 정보 검색
- **•string**: 태그 내부의 텍스트만 출력

```
# find_all 함수
# class 값이 'scraping'인 조건에 해당하는 모든 정보 검색
>>> scraping_all = soup.find_all(class_='scraping')
>>> print(scraping_all)

[<p class="scraping">삼성전자 일별 시세 불러오기</p>, <p class="scraping">이해 쑥쑥 Selena 수업!</p>]
```

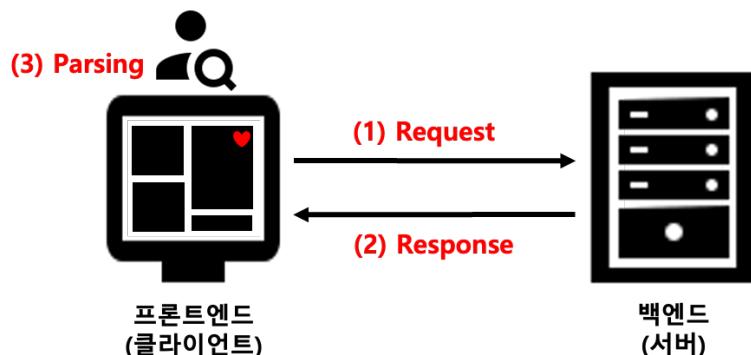
6-3. string

- **string**: 태그 내부의 텍스트만 출력

```
>>> scraping.string
삼성전자 일별 시세 불러오기
```

07. BeautifulSoup 웹 크롤링 3단계 과정

- Request**: 웹 페이지의 URL 이용해서 HTML 문서를 요청
- Response**: 요청한 HTML 문서를 회신
- Parsing**: 태그 기반으로 파싱(일련의 문자열을 의미 있는 단위로 분해)



“Selena’s Comment!”

크롤링에서는 3가지 단계인 Request, Response, Parsing을 제대로 이해하면 돼요!
주식 데이터를 직접 크롤링하면서 3단계를 이해해 볼게요.

08. BeautifulSoup F12(개발자 도구) URL 찾기

- 네이버 금융 홈페이지 접속 : <https://finance.naver.com/>
- 삼성전자(code : 005930) 검색
- 시세 메뉴 클릭 후 URL 확인 : <https://finance.naver.com/item/sise.naver?code=005930>

날짜	종가	전일비	시가	고가	저가	거래량
2022.06.24	58,400	▲ 1,000	57,900	59,100	57,700	23,133,782
2022.06.23	57,400	▼ 200	57,700	58,000	56,800	28,338,608
2022.06.22	57,600	▼ 900	59,000	59,100	57,600	23,334,687
2022.06.21	58,500	▼ 200	58,700	59,200	58,200	25,148,109
2022.06.20	58,700	▼ 1,100	59,800	59,900	58,100	34,111,306
2022.06.17	59,800	▼ 1,100	59,400	59,900	59,400	29,053,450
2022.06.16	60,900	▲ 200	61,300	61,800	60,500	23,394,895
2022.06.15	60,700	▼ 1,200	61,300	61,500	60,200	26,811,224
2022.06.14	61,900	▼ 200	61,200	62,200	61,100	24,606,419
2022.06.13	62,100	▼ 1,700	62,400	62,800	62,100	22,157,816

- 키보드 F12(개발자 도구) 클릭 > 메뉴 Elements 클릭 > 키보드 Ctrl과 F(검색 단축기) 클릭
> '일별 시세' 검색 > scr 값 복사

```

<div class="section inner_sub">
  <div style="float:left; width:381px; margin-right:9px;">...</div>
  <div style="float:left; width:290px;">...</div>
  <iframe name="day" src="/item/sise_time.naver?code=005930&thistime=20220624161133" width="100%" height="360" marginheight="0" bottommargin="0" scrolling="no" frameborder="0" title="주요 시세" style="margin-top:20px; clear:both;">...</iframe>
  <iframe name="day" src="/item/sise_day.naver?code=005930" width="100%" height="360" marginheight="0" bottommargin="0" topmargin="0" scrolling="no" frameborder="0" title="일별 시세" style="margin-top:20px; clear:both;">...</iframe>
  <!-- // 일별 시세 -->
  <div id="aside">...</div>
  </div>
  <div id="footer">...</div>
  <script type="text/javascript">...</script>
  <script language="javascript" type="text/javascript"> jindo.$Fn(doUpdateInformation).attach(document, 'domready'); </script>
</div>
</body>
</html>

```

일별 시세 검색! 1 of 2 Cancel

09. BeautifulSoup 첫 번째 단계. Request

9-1. URL 저장

- stock_url이라는 변수에 네이버 금융 사이트의 삼성전자 시세 정보가 담긴 URL 저장

```
# stock_url이라는 변수에 URL 저장
stock_url = 'https://finance.naver.com/item/sise_day.naver?code=005930'
```

일별 시세						
날짜	종가	전일비	시가	고가	저가	거래량
2022.06.24	58,400	▲ 1,000	57,900	59,100	57,700	23,133,782
2022.06.23	57,400	▼ 200	57,700	58,000	56,800	28,338,608
2022.06.22	57,600	▼ 900	59,000	59,100	57,600	23,334,687
2022.06.21	58,500	▼ 200	58,700	59,200	58,200	25,148,109
2022.06.20	58,700	▼ 1,100	59,800	59,900	58,100	34,111,306
2022.06.17	59,800	▼ 1,100	59,400	59,900	59,400	29,053,450
2022.06.16	60,900	▲ 200	61,300	61,800	60,500	23,394,895
2022.06.15	60,700	▼ 1,200	61,300	61,500	60,200	26,811,224
2022.06.14	61,900	▼ 200	61,200	62,200	61,100	24,606,419
2022.06.13	62,100	▼ 1,700	62,400	62,800	62,100	22,157,816

1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 다음 | 맨뒤 | >

9-2. User-agent 설정

- headers에 user-agent 값 저장
- user-agent 확인 사이트 : <http://www.useragentstring.com/>

User Agent String.Com

User Agent String explained :

```
Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.69
Safari/537.36
```

Copy/paste any user agent string in this field and click 'Analyze'

Analyze

- user-agent란, 웹 크롤링을 진행하면 대부분 서버에서 봇을 차단하기 때문에, 종종 페이지에서 아무것도 받아오지 못하는 경우 발생!
- 그래서 user-agent를 headers에 저장하면 오류를 해결할 수 있습니다.

```
# header에 user-agent 값 저장
headers = {'User-Agent':'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36'}
```

“Selena's Comment!”

user-agent 까먹으면 안 돼요.
 웹 크롤링을 하다 보면, 페이지에서 아무런 정보도 못 받아오는 경우가 발생하는데요!
 이때 서버가 크롤링 하는 저희를 봇으로 생각하고 차단해서 그래요.
 차단을 푸는 방법은 user-agent를 headers에 저장해 줘서 나는 봇이 아니야라고 사이트에 알려 주면 됩니다.
 간단하죠~?!

9-3. requests.get()

- 웹 페이지의 URL 이용해서 HTML 문서를 요청
- requests.get(stock_url, headers = headers)**
 - URL 값을 파라미터 값으로 입력
 - 해당 사이트는 반드시 헤더 정보를 요구하기 때문에 파라미터 값으로 헤더 입력

```
# stock_url이라는 변수에 URL 저장
>>> stock_url = 'https://finance.naver.com/item/sise_day.naver?code=005930'
```

```
# header에 user-agent 값 저장
>>> headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36'}
```

```
# request.get 함수를 통해 URL을 이용하여 HTML 문서를 요청
>>> requests.get(stock_url, headers = headers)
```

```
<Response [200]>
```

10. BeautifulSoup 두 번째 단계. Response

- 요청한 HTML 문서를 회신
- response = requests.get(stock_url, headers = headers)**
 - 서버에서 요청을 받아 처리한 후, 요청자에게 응답 줌
 - HTML 코드 형태

```
# response 변수에 요청한 HTML 문서를 회신하여 저장
>>> response = requests.get(stock_url, headers = headers)
```

11. BeautifulSoup 세 번째 단계. Parsing

- 태그 기반으로 파싱(일련의 문자열을 의미 있는 단위로 분해)
- soup = BeautifulSoup(response.text, 'html.parser')**
 - html을 파이썬에서 읽을 수 있게 파싱(파이썬 객체로 변환)
 - response.text에 저장한 html 소스코드를 .parser를 붙여 변환
 - parser는 파이썬의 내장 메소드

```
# soup 변수에 BeautifulSoup의 객체 생성
# HTML 코드를 파이썬에서 읽을 수 있도록 파싱
>>> soup = BeautifulSoup(response.text, 'html.parser')
```

“Selena’s Comment!”

네이버 금융 사이트에서 삼성전자의 일별 시세 테이블을 가져와보는 실습을 진행할 거예요!

12. BeautifulSoup 반복문으로 일별 종가 구현

- 200일 동안의 일별 종가 정보 가져오는 반복문 구현
- 반복문 코드 설명
 - 1) 200일 일별 종가 정보는 1 Page 당 10일의 일별 종가 정보 담겨있어서 **20 Page** 필요
 - 2) 일별 종가 담긴 URL과 Header 정보로 **requests.get 함수** 구현
 - 3) 요청한 HTML 문서를 회신하여 **response 변수**에 저장
 - 4) BeautifulSoup함수로 HTML을 읽을 수 있도록 파싱하여 **soup 변수**에 저장
 - 5) Page 개수만큼 20번 반복
 - "**tr**" 태그 조건에 해당하는 모든 정보를 검색하여 **parsing_list 변수**에 저장
 - 6) 1 Page 당 10일의 일별 종가 정보 담겨있어서 10번 반복
 - "**td**" 태그의 **align**가 "center"인 값들 중 0번째 조건에 해당하는 정보 검색하여 출력
 - "**td**" 태그의 **class**가 "num"인 값들 중 0번째 조건에 해당하는 정보 검색하여 출력
- ✓ 태그 정보는 F12(개발자 도구) 클릭하여 찾기

```
<tr onmouseover="mouseOver(this)" onmouseout="mouseOut(this)" style="background-color: #55;">
  <td align="center">
    <span class="tah p10 gray03">2022.06.24</span>
  </td>
  <td class="num">
    <span class="tah p11">58,400</span>
  </td>
```

```
# 200일 동안의 일별 종가 정보 가져오는 반복문(1페이지 당 10일 정보 담겨있음)
>>> for page in range(1, 21):
    print (str(page))

    # url + page 번호 합치기
    stock_url = 'http://finance.naver.com/item/sise_day.nhn?code=005930' + '&page=' + str(page)

    # header 정보
    headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36'}

    # request : 웹 페이지의 URL, header 이용해서 HTML 문서 요청
    # response : 요청한 HTML 문서 회신
    response = requests.get(stock_url, headers = headers)

    # parsing : HTML을 읽을 수 있도록 파싱
    # soup 변수에 BeautifulSoup의 객체 생성
    soup = BeautifulSoup(response.text, 'html.parser')

    # "tr" 태그 조건에 해당하는 모든 정보 검색
    parsing_list = soup.find_all("tr")

    # None 값은 걸러주기 위한 변수 생성
    isCheckNone = None
```

```
# 페이지당 일별 종가 출력하기 위한 반복문 <들여쓰기 주의>
for i in range(1, len(parsing_list)):
    # None 값은 걸러주기 위한 조건문 <들여쓰기 주의>
    # .span()는 매치된 문자열의 (시작, 끝)에 해당하는 튜플을 돌려주는 함수
    if(parsing_list[i].span != isCheckNone):
        # parsing_list[i] : i번째 parsing_list, i 번째 "tr" 태그 값
        # .find_all("td", align="center")[0].text : "td" 태그의 align가 "center"인 값들 중 0번째 값
        # .find_all("td", class_="num")[0].text : "td" 태그의 class가 "num"인 값들 중 0번째 값
        print(parsing_list[i].find_all("td", align="center")[0].text,
              parsing_list[i].find_all("td", class_="num")[0].text)
```

1
2022.06.24 58,400
2022.06.23 57,400
2022.06.22 57,600
2022.06.21 58,500
2022.06.20 58,700
2022.06.17 59,800
2022.06.16 60,900
2022.06.15 60,700
2022.06.14 61,900
2022.06.13 62,100
2
2022.06.10 63,800
2022.06.09 65,200
2022.06.08 65,300
2022.06.07 65,500
2022.06.03 66,800
2022.06.02 66,700
2022.05.31 67,400
2022.05.30 67,700
2022.05.27 66,500
2022.05.26 65,900

...

20
2021.09.13 76,300
2021.09.10 75,300
2021.09.09 75,300
2021.09.08 76,300
2021.09.07 76,100
2021.09.06 77,300
2021.09.03 76,600
2021.09.02 76,000
2021.09.01 76,800
2021.08.31 76,700

“Selena’s Comment!”

보너스로 Pandas를 이용하여 네이버 금융 사이트에서 삼성전자의 일별 시세 테이블을 가져와보는 실습을 진행할 거예요!

13. Pandas 일별 시세 테이블 구현

- 1) Pandas 라이브러리와 Requests 라이브러리 이용
- 2) 200일 일별 종가 정보는 1 Page 당 10일의 일별 종가 정보 담겨있어서 20 Page 필요
- 3) 일별 종가 담긴 URL과 Header 정보로 requests.get 함수 구현
- 4) **pandas.read_html** 함수를 통해 HTML 불러와서 파싱
- 5) **append** 함수를 이용하여 dataframe 끝에 추가하고 싶은 요소를 추가하여 dataframe 리턴
- 6) **dropna** 함수를 통해 결측 값 제거

```
# 네이버 금융 일별 시세 테이블 불러오기
# pandas 라이브러리와 requests 라이브러리 이용
# code = 회사 코드, page = 일별 시세 테이블의 페이지 수 (200 행의 데이터 불러오려면 20 페이지 입력)
>>> import pandas as pd

>>> df = pd.DataFrame()

>>> for page in range(1,21):
    stock_url = 'http://finance.naver.com/item/sise_day.nhn?code=005930' + '&page=' + str(page)

    # header 정보
    headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36'}

    # request : 웹 페이지의 URL, header 이용해서 HTML 문서 요청
    # response : 요청한 HTML 문서 회신
    response = requests.get(stock_url, headers = headers)

    # response.text로 응답을 주면 HTML 코드이기 때문에 read_html로 불러오기
    # append() : dataframe 끝에 추가하고 싶은 요소를 추가하여 새로운 dataframe을 리턴
    df = df.append(pd.read_html(response.text, header=0)[0], ignore_index=True)

# 결측 값 있는 행 제거
>> df = df.dropna()
>> df
```

	날짜	종가	전일비	시가	고가	저가	거래량
1	2022.06.24	58400.0	1000.0	57900.0	59100.0	57700.0	23133782.0
2	2022.06.23	57400.0	200.0	57700.0	58000.0	56800.0	28338608.0
3	2022.06.22	57600.0	900.0	59000.0	59100.0	57600.0	23334687.0
4	2022.06.21	58500.0	200.0	58700.0	59200.0	58200.0	25148109.0
5	2022.06.20	58700.0	1100.0	59800.0	59900.0	58100.0	34111306.0
...
294	2021.09.06	77300.0	700.0	76800.0	77600.0	76600.0	12861180.0
295	2021.09.03	76600.0	600.0	76400.0	76700.0	76000.0	12096419.0
296	2021.09.02	76000.0	800.0	76800.0	76800.0	75700.0	15347486.0
297	2021.09.01	76800.0	100.0	76700.0	77100.0	75900.0	16114775.0
298	2021.08.31	76700.0	2100.0	74900.0	76700.0	74300.0	24630370.0

“Selena’s Comment!”

우!와! 수강생 여러분!!!
드디어 완강입니다!!! 정말 고생 많으셨어요.
진정한 승자는 바로 완강한 여러분입니다!

Selena 강의를 믿고 끝까지 들어주셔서 감사합니다.

최종 복습하면서 그동안 했던 내용들 살펴보세요~!
복습하시면서 헷갈리는 내용은 꼭 질문해 주세요.

그동안 감사했습니다!!! 다른 강의에서 또 만나요♥