



데이터 분석 & 빅데이터

곽경일 강사

Phase	Step	Define	Outputs
Identify Problem (문제 규명)	1. 과제선정	경영 상의 이슈를 도출하여, 프로젝트로 선정	프로젝트 실행 계획서
	2. 과제의 선정배경	선정된 주제가 왜 중요한가를 사실 중심으로 기술	선정 배경 기술
	3. 목표 및 기대효과	프로젝트의 목표와 범위를 설정, 기대효과 산정	목표 및 기대효과 기술
	4. 분석모델 개념도	과제 선정단계에서 1차적인 분석모델을 구체화	분석 모델 개념도 (1차)
	5. 세부 활동계획	일정, 팀 구성, 단계 별 보고 등 팀 활동계획 수립	프로젝트 일정 및 팀 구성도
Collect Data (데이터 수집)	6. 데이터 탐색	과제와 연관된 내부 및 외부, 혹은 기존 및 신규 데이터 탐색	탐색계획 및 탐색결과 요약
	7. 데이터 선정 및 정의	탐색된 데이터를 평가 및 선정하고, 데이터의 특성을 정의	변수 유형별 정의서
	8. 분석모델 구체화	선정된 데이터를 바탕으로 분석모델을 수정 보완 함 (2차 모델)	분석 모델 개념도
	9. 데이터 수집	선정된 데이터의 수집계획을 마련하고, 데이터를 수집함	데이터 수집 계획서, RAW 데이터
	10. 데이터 분석 Set 준비	분석 모델에 따른 데이터 분석을 위한 데이터 Set을 준비	분석 데이터 Set
Analyze Data (데이터 분석)	11. 데이터 신뢰성 확인	데이터의 오류, Missing Data, 등을 점검하고 수정함	데이터 신뢰성 점검 결과요약
	12. 개별 변수 분석	개별 변수의 Trend, 분포, 중심, 분산, 이상치, 신뢰구간을 파악	Trend, 분포, 이상치 조치 결과
	13. 변수 간 관계 분석	결과변수(Y)와 원인변수(X), 층별변수(s) 간의 관계 파악	변수 간 관계분석 결과
	14. 모델링 분석	$Y=f(X's)$ 의 분석모델을 통해, 최적화 및 예측 실시	모델링 분석 결과
	15. 분석 결론 및 개선방향	분석결과를 정리하고, 추가적인 분석계획 및 개선안을 마련	분석결과 별 추가분석 및 개선방향
Improve & Systematize (개선 및 시스템화)	16. 개선안 선정 및 기대효과	개선안을 도출, 평가, 선정하고, 구체적인 실행계획을 마련	개선안 선정 및 실행계획표
	17. 개선안 표준화	개선안을 표준화 하고 관리계획을 수립함	개선안 표준화 계획 및 결과물
	18. 분석절차 문제점 개선	주기적인 분석이 필요할 경우, 분석 상의 문제점 및 개선안 마련	분석절차 상의 문제점 도출 및 선정
	19. 분석절차 시스템화	IT 시스템으로 데이터 수집, 저장, 변환, 분석 표현, 활용을 구축	IT 시스템화 계획표
	20. 프로젝트 성과확인	프로젝트의 성과를 파악하고 프로젝트의 결과를 문서화	완료 보고서



Python Programming

- 1991년 Gudi Van Rossum 발표
- 플랫폼 독립적 / 뛰어난 확장성 (DataBase / Web / Application ...)
- 인터프리터 언어 / 객체 지향
- Ai 구현 및 빅데이터 분석을 위한 라이브러리 제공
- 다른 프로그래밍 언어들에 비해 쉬운 코딩 / 가독성이 높은 언어



* Python 언어와 C 언어 비교

▼ C언어의 "Hello!" 문구 출력

Code :

```
#include<stdio.h>
int main()
{
    printf("Hello!\n");
    return 0 ;
}
```

Output : Hello!

▼ Python언어의 "Hello!" 문구 출력

Code :

```
print("Hello!")
```

Output : Hello!

Anaconda Environment

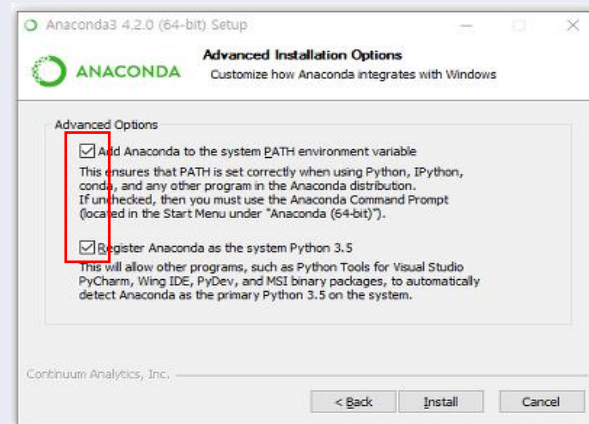
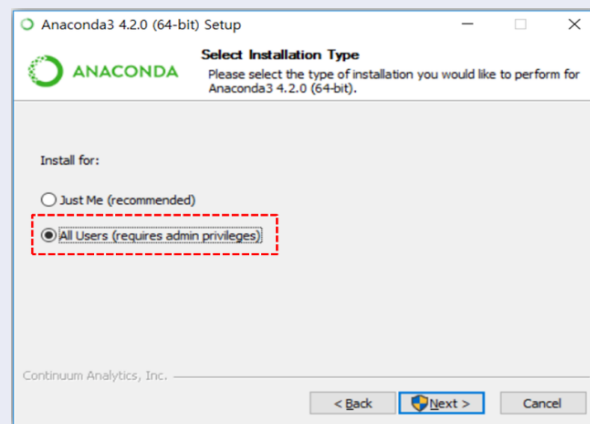
- Python과 R언어를 사용할 수 있는 통합 Platform
- Python Package들이 자동으로 설치됨
- Scikit-Learn / Tensorflow / Theano 라이브러리를 이용해
기계학습 및 딥러닝 모델 개발 환경 제공
- Numpy / Pandas / Numba 를 사용하여 데이터 분석 및 다른 Platform과 연동



* **Download :** <https://www.anaconda.com/distribution/>

* **설치 시 주의 사항 :**

1. 설치 시 All User에 설치 (Anaconda 관리자 권한부여)
2. 환경 변수에 Path 추가



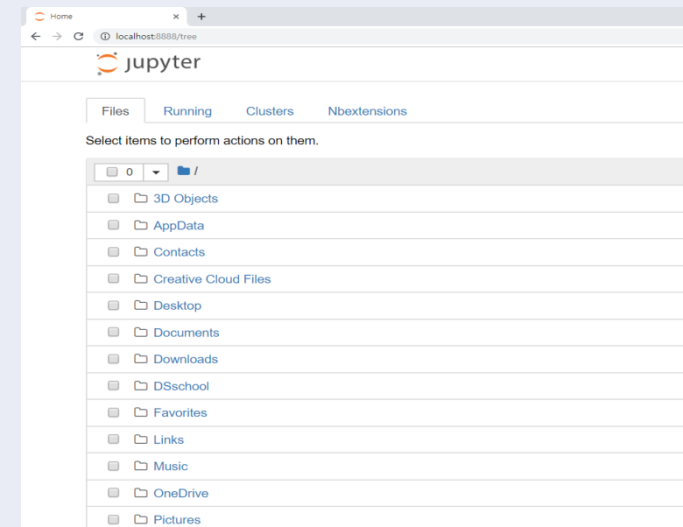
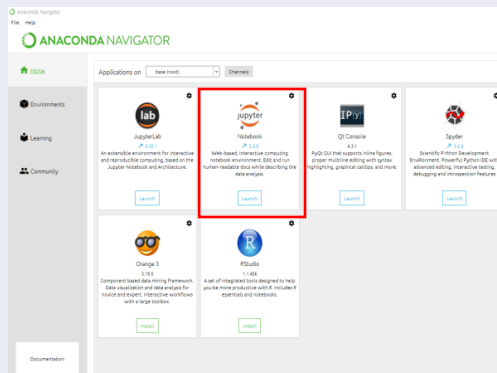
Jupyter Notebook

- 데이터 분석을 위한 Python 프로그래밍 전용 프로그램
- 웹 기반 프로세스, 분석결과를 리포트 형태로 출력
- 데이터분석의 결과를 즉각적으로 확인 가능
- 실무에서 가장 많이 사용하는 Python 데이터 분석 Tool



* Jupyter Notebook 실행

1. [시작] - [Anaconda3] 폴더 클릭 - [Jupyter Notebook] 실행
2. [Anaconda Navigator] 실행 - [Jupyter Notebook] 클릭



▲ Jupyter Notebook 실행 시 나오는 첫 화면

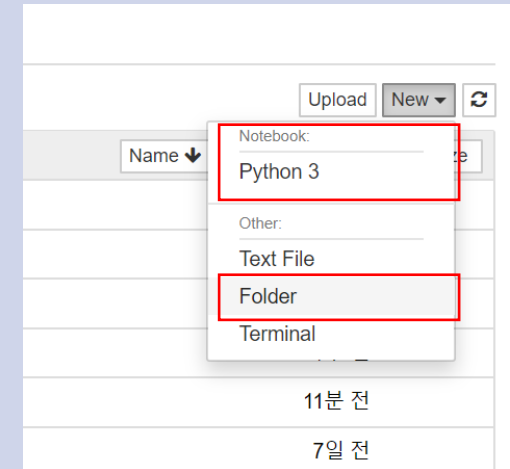
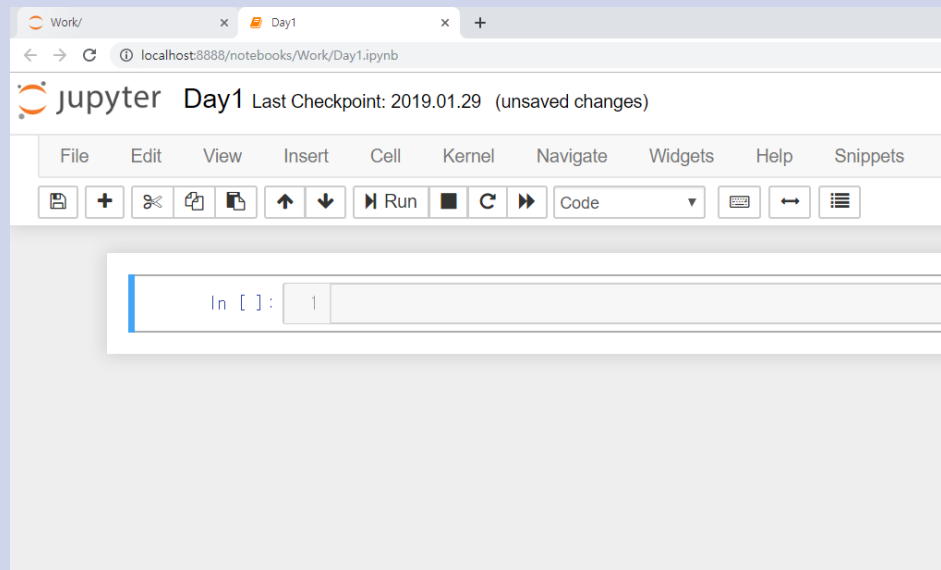
Jupyter Notebook

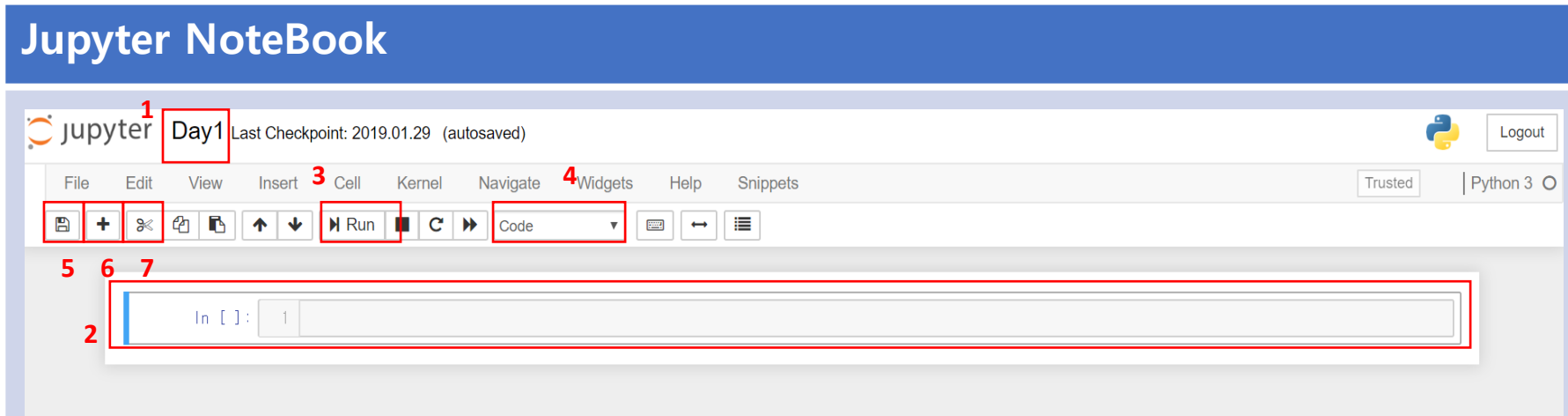
* 자신이 사용할 **Working Directory** 를 설정 또는 생성

> **폴더 생성** : 우측 상단 [New] 버튼 클릭 – [Folder] 버튼 클릭

> **Notebook 파일 생성** : 우측 상단 [New] 버튼 클릭 – [Python 3] 버튼 클릭

* 아래와 같은 새 창이 생성





1) 파일 이름 설정

2) 코드 입력 창

- 코드 입력 모드 : 코드 입력 창이 초록색, 코드를 직접 타이핑 할 수 있는 상태
- 코드 커맨드 모드 : 코드 입력 창이 파란색, 새로운 코드입력, 주석, 입력 창 삭제 및 이동

3) 코드 실행 : 단축키 [Shift] + [Enter] Key

4) 코드 또는 주석(Markdown) 설정 :

- 코드설정 ; 코드 입력 창이 파란색 일 때, [Y] Key
- 주석설정 ; 코드 입력 창이 파란색 일 때, [M] Key

5) 파일 저장

6) 코드 입력 창 추가

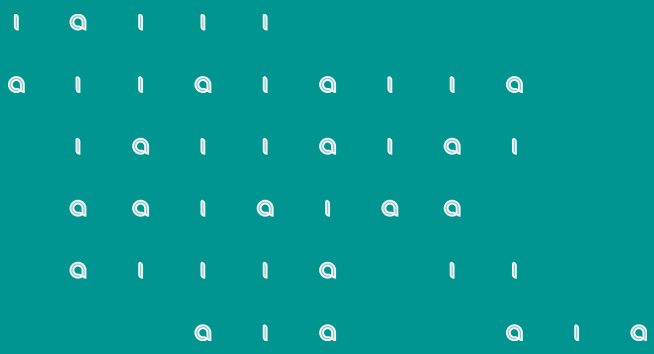
7) 코드 입력 창 삭제

참고 자료 : <https://dojang.io/mod/page/view.php?id=2457>

Jupyter Notebook Working Directory 변경

1. Command 실행 (윈도우 검색창에 'cmd' 라고 입력)
2. jupyter notebook --generate-config 입력
3. 사용자 폴더에 .jupyter 폴더 진입 (cd .jupyter)
4. jupyter_notebook_config.py 파일 실행
5. 약 200번 째 줄에 있는 `***#c.NotebookApp.notebook_dir = "****` 주석을 변경
 - * 주석 제거 후, 자신이 원하는 디렉토리로 변경 `c.NotebookApp.notebook_dir = 'C://시작폴더'`
 - * 여기서 역슬래시를 두번 사용해 주어야 한다. //
6. 저장 후, 바탕화면에 주피터노트북 바로 가기 아이콘 생성.
7. 주피터 노트북의 아이콘의 속성을 열어 보면,
 - 대상(target)이 아래와 같이 맨 끝에 `%USERPROFILE%` 문구를 제거
8. Jupyter Notebook 재 실행

* 참고자료 <http://blog.daum.net/allmani/6>



Python Programming

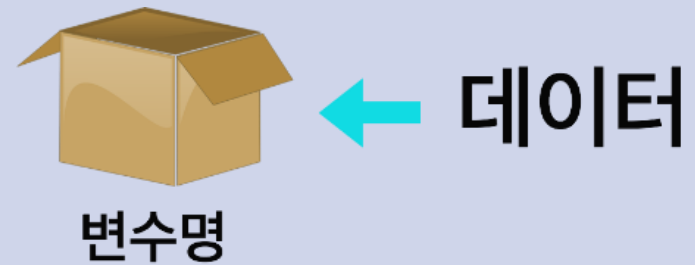


1. 변수와 자료형

변수와 자료형

■ 변수

- 데이터를 특정 이름을 붙여 저장함
- 연산을 쉽게 처리하거나, 큰 데이터를 한번에 불러올 수 있음
- 여러 형태의 데이터를 하나 이상의 변수로 선언 가능
- Python에서 변수명을 한글로 선언 가능



■ 자료형

- 데이터 자료형을 규명하는 것은 프로그래밍에 있어서 가장 기본적이고 중요함
- 파이썬에서는 데이터의 자료형을 변수 선언 시, 자동으로 지정
- 자료형에 따라 분석의 기법이 달라질 수 있음
- 기본적으로 3가지 자료형이 가장 많이 사용됨
 - 정수형 int : 양의 정수, 음의 정수, 0
 - 실수형 float : 정수를 포함한 실수, 0.n 형태로 출력
 - 문자형 str : 영어나 기호, 또는 숫자로 구성된 문자

1. 변수와 자료형

```
1 a = 10
2 print(a)
3 print(type(a))
```

10
<class 'int'>

◀ 정수형 10 데이터가 a 에 선언

```
1 a = "B"
2 print(a)
3 print(type(a))
```

B
<class 'str'>

◀ 문자형 B 데이터가 a 에 선언

```
1 num = 23.1
2 print(num)
3 print(type(num))
```

23.1
<class 'float'>

◀ 실수형 23.1 데이터가 num 에 선언

자료형

- print() : 데이터나 변수를 출력
- type() : 데이터의 타입을 확인

- Jupyter Notebook의 경우, print함수를 사용하지 않고, 변수명만 입력해도 데이터가 출력된다.
- 파이썬에서 등호 (=) 기호는 '같다'를 의미하는 것이 아닌, '선언한다'라는 의미가 있다.

1. 변수와 자료형

▼ 자료형이 일치하지 않으면 오류가 발생

```
1 a = 10
2 b = "B"
3
4 a+b
```

```
-----
TypeError                                Traceback (most recent
call last)
<ipython-input-7-9fd1f8fd3d20> in <module>()
      2 b = "B"
      3
----> 4 a+b

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

▼ 임의로 데이터의 타입을 바꾸고 싶은 경우

```
1 a = 10
2 type(a)
```

int

```
1 a = float(10)
2 type(a)
```

float

자료형

- 자료형이 일치하지 않으면, 오류가 발생한다.
- 자료형을 임의로 바꾸려면, 데이터 앞에 타입을 입력하여 변수를 선언한다.

2. 문자열 자료형(String)

문자열 자료형(String)

- 프로그래밍 뿐만 아니라, 데이터 분석에 있어서 문자를 처리하는 것이 중요

- 문자열 : 문자, 단어 등으로 구성된 문자들의 집합

"단어" / "A" / "Python Programming" / '123'

- 문자열에는 모두 큰따옴표(") 또는 작은따옴표(')를 이용해 선언

-> 문자열 내 큰따옴표나 작은따옴표가 쓰일 경우를 대비해, 두 가지 형태로 선언 가능

- 여러 줄의 문자열을 대입할 경우 큰따옴표 또는 작은따옴표 세 개를 사용

""" 안녕하세요

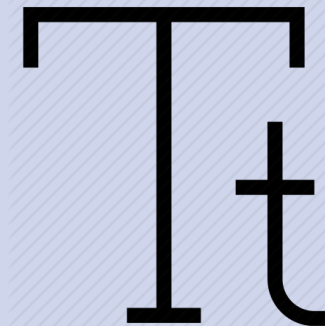
즐거운 하루 보내세요 """

- 문자열을 여러 형태로 연산이 가능함

- 인덱싱(indexing)과 슬라이싱(Slicing)기능이 많이 사용됨

Indexing : 특정 위치의 문자를 추출

Slicing : 특정 위치의 문자를 잘라냄

A large, black, serif-style capital letter 'T' and a lowercase letter 't' are displayed side-by-side. They are set against a light blue background with a subtle diagonal line pattern.

2. 문자열 자료형(String)

▼ 문자를 그대로 작성하면 오류가 발생

```
1 안녕하세요

NameError                                Traceback (most recent call last)
<ipython-input-10-041a565a82b3> in <module>()
--> 1 안녕하세요
```

NameError: name '안녕하세요' is not defined

```
1 '안녕하세요'
```

'안녕하세요'

```
1 "안녕하세요"
```

'안녕하세요'

```
1 '''안녕하세요'''
```

'안녕하세요'

```
1 """안녕하세요"""
```

'안녕하세요'

▼ 여러 문장을 사용할 경우

```
1 """
2 여러문장의 경우
3 따옴표를 세 개를 사용해 주면 됩니다.
4 감사합니다.
5 """
```

'\n여러문장의 경우\n따옴표를 세 개를 사용해 주면 됩니다.\n감사합니다.\n'

```
1 a = '안녕하세요'
2 b = """저는 문자형 입니다.
3 반갑습니다"""
4
5 print(a)
6 print(b)
7 type(a), type(b)
```

안녕하세요
저는 문자형 입니다.
반갑습니다

(str, str)

문자형

- 따옴표를 활용하여 문자형을 문자열 자료형으로 사용할 수 있다.
- 여러 개의 문장의 경우, 세 개의 따옴표를 이용한다.
- 문자열도 변수로 저장해 줄 수 있다.

2. 문자열 자료형(String)

▼ 연산자를 이용해 문자열 연산이 가능

```
1 str1 = "Python 실무 데이터 분석"
2 str2 = "에 오신 여러분을 환영합니다."
3
4 str1 + str2
```

'Python 실무 데이터 분석에 오신 여러분을 환영합니다.'

```
1 str3 = '감사합니다'
2
3 str3 * 3
```

'감사합니다감사합니다감사합니다'

▼ len을 이용해 문자열의 길이를 계산

```
1 str4 = "이 문자열의 길이는 len함수를 이용해 구할 수 있습니다."
2
3 len(str4)
```

32

문자형

- 연산자를 활용해, 문자열 연산이 가능하다. (덧셈 / 곱셈)
- len() : 해당 문자열의 길이를 구하는 함수 (데이터의 길이나 크기를 구할 때도 사용)

2. String Indexing

▼ Indexing 과 연산의 활용

```
1 str5 = "너의 삶은 짧지만, 계속 구하면 가치있는 것을 찾으리라"  
2  
3 str5[0]
```

'너'

```
1 str5[14]
```

'구'

```
1 str5[-2]
```

'리'

```
1 str5[0] + str5[14] + str5[-2]
```

'너구리'

▼ Indexing 결과를 새로운 변수에 선언

```
1 str6 = str5[0] + str5[14] + str5[-2]  
2 print(type(str6))  
3 str6
```

<class 'str'>

'너구리'

문자형

- Indexing : 특정 위치의 글자를 추출해 올 수 있다.
- 대괄호를 이용해 추출하며, 0부터 순서가 매겨진다.
- 뒷부분부터 셀 땐, 음수를 붙여 세어줄 수 있다.
- 실무에서 복잡한 문자데이터를 다룰 때, (설문, 제품명, 항목명 등) 주로 사용된다.

2. String Slicing

▼ Slicing 과 연산의 활용

```
1 str5 = "너의 삶은 짧지만, 계속 구하면 가치있는 것을 찾으리라"  
2  
3 str5[0:2]
```

'너의'

```
1 str5[:11]
```

'너의 삶은 짧지만, '

```
1 str5[18:]
```

'가치있는 것을 찾으리라'

```
1 str5[:11] + str5[18:]
```

'너의 삶은 짧지만, 가치있는 것을 찾으리라'

▼ Slicing 사용

```
1 str7 = "2019Python실무"  
2 day = str7[:4]  
3 lesson = str7[4:]  
4  
5 print(day)  
6 print(lesson)
```

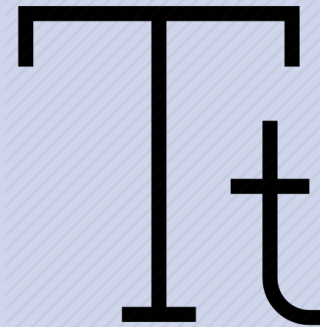
2019
Python실무

문자형

- Slicing : 특정 글자를 잘라서 추출해 올 수 있다.
- str[n:m] : str 변수의 문자열에서, n번째 부터, m번째 글자 전까지 추출해 온다.
- str[:m] : 맨 처음부터 m 번째 글자 전 까지 추출해 온다.
- str[n:] : n 번째 글자부터 맨 끝 글자 까지 추출해 온다.
- 처음번호와 끝 번호를 생략하면 모든 문자열이 출력된다.

문자열 함수

- 파이썬에서 여러 가지 문자열 함수를 지원
- **count()** : 특정 문자의 개수를 출력
- **find() / index()** : 특정 문자가 어디에 위치해 있는지 출력
(index 함수의 경우, 해당 문자가 없으면 오류 발생)
- **join()** : 문자열에 특수기호나 특정 문자를 삽입
- **upper() / lower()** : 영어 소문자를 대문자로 변경 (upper) / 영어 대문자를 소문자로 변경 (lower)
- **lstrip() / rstrip() / strip()** : 문자열 좌,우 또는 전체 공백을 제거
- **replace()** : 특정 문자를 다른 문자로 변환
- **split()** : 문자를 공백 기준으로 리스트의 형태로 나누어 출력
(쌍반점(:) 기호 사용 시, 전체 문자열이 리스트에 하나의 객체로 출력)

A large, black, serif-style capital letter 'T' and a lowercase letter 't' are displayed side-by-side. The letters are set against a light blue background with a subtle diagonal line pattern.

2. String Function

▼ 여러 가지 String Function

- count

```
1 str8 = "Python Business Data Analyze"
2 str8.count('a')
```

3

- find / index

```
1 str8 = "Python Business Data Analyze"
2 str8.find('D')
```

16

```
1 str8 = "Python Business Data Analyze"
2 str8.index('D')
```

16

- Join

```
1 ", ".join('가나다라마')
```

'가,나,다,라,마'

- upper/ lower

```
1 str8 = "Python Business Data Analyze"
2 str8.upper()
```

'PYTHON BUSINESS DATA ANALYZE'

```
1 str8.lower()
```

'python business data analyze'

- lstrip / rstrip / strip

```
1 str9 = "    Python    "
```

```
1 str9.lstrip()
```

'Python '

```
1 str9.rstrip()
```

' Python'

```
1 str9.strip()
```

'Python'

- replace

```
1 str10 = "python Big 데이터"
2 str10.replace("데이터", "data")
```

'python Big data'

- split

```
1 str10 = "python Big 데이터"
2 str10.split()
```

['python', 'Big', '데이터']

```
1 str10.split(':')
```

['python Big 데이터']

3. List

리스트 (List)

- 리스트 (list) : 여러 개의 데이터 다룰 때, 하나의 변수에 많은 값을 집어 넣을 수 있음
- 대괄호를 이용하여, 데이터를 묶어 줌

`a = [1,2,3,4,5]` #5개의 데이터가 a라는 변수에 모두 담겨있음

- C 언어의 배열(array)과 유사한 List 를 이용
- 서로 다른 데이터 타입의 값들도 하나의 List에 넣을 수 있음
- 리스트 내 데이터 간 순서가 존재
- 리스트 내 데이터를 삭제하거나 추가, 수정이 가능함

`append()` : 추가

`insert()` : 삽입

`remove()` : 삭제

- Size가 정해져 있지 않고 유동성이 있다.



3. List

▼ 리스트 선언과 형태

```
1 list1 = [100, 200, 300, 400, 500]
2 list1
```

[100, 200, 300, 400, 500]

```
1 type(list1)
```

list

```
1 list2 = ['김종찬', 70, 170, '서울 강남']
2 list2
```

['김종찬', 70, 170, '서울 강남']

```
1 type(list2)
```

list

▼ 리스트의 특정 위치에 값을 추출할 때

```
1 list1 = [100, 200, 300, 400, 500]
2 list1[0]
```

100

```
1 list1[1], list1[2], list1[3]
```

(200, 300, 400)

리스트

- 문자열도 리스트로 선언이 가능하다.
- 리스트에서 특정 값을 추출 (Indexing)
 - => 리스트 이름을 입력 후, 대괄호를 이용해 위치를 호출
 - => 순서는 0부터 시작

3. List

▼ 리스트의 여러 가지 함수

```
1 sub = ['국어', '영어', '수학', '과학']
2 sub
```

['국어', '영어', '수학', '과학']

```
1 sub.append('사회')
2 sub
```

['국어', '영어', '수학', '과학', '사회']

```
1 sub.insert(0, '한국사')
2 sub
```

['한국사', '국어', '영어', '수학', '과학', '사회']

```
1 sub.remove('수학')
2 sub
```

['한국사', '국어', '영어', '과학', '사회']

▼ 리스트의 연산

```
1 score = [10, 20, 30, 40]
2 score
```

[10, 20, 30, 40]

```
1 score * 3
```

[10, 20, 30, 40, 10, 20, 30, 40, 10, 20, 30, 40]

```
1 score + sub
```

[10, 20, 30, 40, '한국사', '국어', '영어', '과학', '사회']

리스트

- append() : 리스트에 해당 값을 추가한다
- insert() : 리스트 특정위치에 해당 값을 추가한다.
- remove () : 특정 값을 리스트에서 삭제한다.
- 연산기호를 활용하여, 리스트 간 연산이 가능하다.

참고 자료 : https://wikidocs.net/14#_6

3. Tuple

튜플 (Tuple)

- 튜플 (tuple) : 리스트와 같이 여러 개의 데이터를 집어넣을 수 있는 공간
- 소괄호를 이용하여, 데이터를 묶어 줌

`c = (1,2,3,4,5)` #5개의 데이터가 c라는 변수에 모두 담겨있음

- 튜플 내 데이터 간 순서가 존재
- 한번 선언된 튜플은 변경이 불가능 함
- Packing과 Unpacking을 활용하여, 데이터를 추출하거나 튜플을 생성할 수 있음
 - Packing : 여러 개의 데이터를 쉼표(,) 구분자를 이용해, 하나의 변수로 생성
 - Unpacking : 하나의 튜플을 여러 개의 변수로 선언하여, 변수에 각 데이터를 선언
- 함수와 반복문 같이 중요한 하이퍼파라미터(Hyper Parameter)들을 보호 할 때 사용

하이퍼파라미터 : 수식 내 값이 변하지 않는 인자나 상수



3. Tuple

▼ 튜플 선언과 형태

```
1 t1 = (1,2,3,4,5)
2 print(type(t1))
3 t1
```

```
<class 'tuple'>
```

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

```
1 t1.append('A')
```

```
AttributeError                                Traceback (most recent call last)
<ipython-input-67-24fe340e95c6> in <module>()
--> 1 t1.append('A')
```

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

```
1 t1.remove(3)
```

```
AttributeError                                Traceback (most recent call last)
<ipython-input-68-8e4f1a7376f5> in <module>()
--> 1 t1.remove(3)
```

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

▼ 튜플 내 데이터 간 순서가 존재

```
1 t2 = (1,2,'삼',4,5)
2 t2
```

```
(1, 2, '삼', 4, 5)
```

```
1 t2[2]
```

```
'삼'
```

```
1 t2[-1]
```

```
5
```

튜플

- 소괄호를 이용해, 튜플을 생성할 수 있다.
- 튜플은 한번 선언되면, 변경이 불가능하다.
- 그러나 각 데이터를 하나의 튜플로부터 추출해 낼 수 있다.

3. Tuple

▼ Packing 과 Unpacking

Packing

```
1 pack1 = 1,2,3,4,5,6
2 type(pack1)
```

tuple

```
1 pack1
```

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

Unpacking

```
1 a,b,c,d,e,f = pack1
2 type(a)
```

int

```
1 a
```

1

```
1 a+b+c+d+e+f
```

21

▼ 튜플 함수 및 연산

```
1 a = (10, 20, 30, 40, 50)
2 a[0]
```

10

```
1 a[-1]
```

50

```
1 a[2:]
```

(30, 40, 50)

```
1 b = (1, 2, 3, 4)
2 c = (5, 6, 7, 8)
3 c+b
```

(5, 6, 7, 8, 1, 2, 3, 4)

```
1 c * 2
```

(5, 6, 7, 8, 5, 6, 7, 8)

튜플

- Packing : 각각의 데이터를 하나의 변수로 선언할 때, 해당 변수들을 하나의 튜플로 선언할 수 있다.
- Unpacking : 하나의 튜플 내 데이터를 각각의 변수로 선언할 수 있다.
- Indexing과 Slicing은 가능하다. 또한 기본 연산자를 이용한 연산이 가능하다.

3. Set

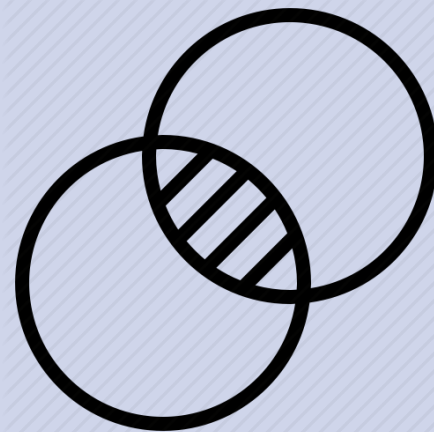
세트 (Set)

- 세트 (Set) : 리스트와 같이 여러 개의 데이터를 집합의 형태로 집어넣을 수 있는 공간
- 중괄호를 이용하여, 데이터를 묶어 줌

`c = {1,2,3,4,5}` #5개의 데이터가 c라는 변수에 모두 담겨있음

- 집합 내 데이터 간 순서 없음, 중복을 허용하지 않음
- 집합 내 데이터 변경이 가능
- 집합연산이 가능 (Van Diagram의 개념)

집합 연산	
A&B	A와 B의 교집합 연산
A B	A와 B의 합집합 연산
A-B	A집합에서 B집합의 원소를 제외한 나머지
A^B	A집합과 B집합의 교집합을 제외한 나머지



3. Set

▼ Set의 선언과 형태

```
1 A = {10, 20, 30, 40, 10}
2 print(A)
3 print(type)
```

```
{40, 10, 20, 30}
<class 'type'>
```

```
1 B = {10, 10, 10, 10, 10, 10}
2 print(B)
```

```
{10}
```

```
1 S = set([1, 2, 3, 4, 5, 6])
2 print(S)
3 print(type(S))
```

```
{1, 2, 3, 4, 5, 6}
<class 'set'>
```

▼ Set의 집합연산

```
1 C = {10, 20, 30, 40, 50}
2 D = {40, 50, 60, 70}
```

```
3
4 # 교집합 연산
```

```
5 print(C&D)
```

```
6 # 합집합 연산
```

```
7 print(C|D)
```

```
8 # 차집합 연산 : C 집합에서 D 집합의 원소를 뺀 나머지
```

```
9 print(C-D)
```

```
10 # 대칭 차집합 연산 : 교집합을 제외한 나머지 원소 출력
```

```
11 print(C^D)
```

```
{40, 50}
{50, 20, 70, 40, 10, 60, 30}
{10, 20, 30}
{70, 10, 20, 60, 30}
```

셋

- set() : 괄호 내부에 있는 list를 set형태로 선언한다.
- 중복된 값은 스스로 제외하고 출력한다.
- 집합 연산과 동일하게 작동한다.

3. Set

▼ Set에서 사용하는 여러 함수

```
1 print(C.intersection(D))  
2 print(C.union(D))  
3 print(C.difference(D))
```

```
{40, 50}  
{50, 20, 70, 40, 10, 60, 30}  
{10, 20, 30}
```

```
1 C.add(100)  
2 C
```

```
{10, 20, 30, 40, 50, 100}
```

```
1 C.update([1,2,3,4,5])  
2 C
```

```
{1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 100}
```

```
1 C.remove(30)  
2 C
```

```
{1, 2, 3, 4, 5, 10, 20, 40, 50, 100}
```

셋

- A.intersection(B) : A와 B의 교집합을 연산 (A&B와 같은 결과)
- A.union(B) : A와 B의 합집합을 연산 (A|B와 같은 결과)
- A.difference(B) : A와 B의 차집합을 연산 (A-B와 같은 결과)
- A.add(n) : A의 Set에 n이라는 값을 추가
- A.update([n1,n2,n3]) : A의 Set에 [n1,n2,n3]의 여러 값을 추가

3. Dictionary

딕셔너리 (Dictionary)

- 딕셔너리 (Dictionary) : 데이터를 Key와 Value의 Pair 형태로 하나의 변수에 선언
- 중괄호를 이용하여, key-value Pair 형태로 묶어줌

```
d = {"name" : "Kim", "value" : 100 }
```

#Kim이라는 데이터가 name이라는 키값과 쌍을 이룸.

#100이라는 데이터가 value라는 키값과 쌍을 이룸.

- 매우 많이 사용되는 데이터 자료형 중 하나
- 데이터를 구조적으로 다룰 수 있음
- Key 값은 중복 되지 않음
- 이후, Pandas 라이브러리의 Series 와 비슷한 개념



3. Dictionary

▼ Dictionary 의 선언과 형태

```
1 D = {'A':100, 'B':200, 'C':300}
2 print(D)
3 print(type(D))
```

```
{'A': 100, 'B': 200, 'C': 300}
<class 'dict'>
```

▼ Dictionary Indexing

```
1 dict1 = {'이름': '안기모', '소속': '빅데이터 그룹', '월 수익': 2000}
2 dict1
```

```
{'이름': '안기모', '소속': '빅데이터 그룹', '월 수익': 2000}
```

```
1 dict1['이름']
```

```
'안기모'
```

```
1 dict1.get('소속')
```

```
'빅데이터 그룹'
```

```
1 dict1.keys()
```

```
dict_keys(['이름', '소속', '월 수익'])
```

```
1 dict1.values()
```

```
dict_values(['안기모', '빅데이터 그룹', 2000])
```

딕셔너리

- 딕셔너리는 Key - Value Pair 형태의 데이터 타입을 갖는다.
- D['Key'] : D라는 딕셔너리의 Key값에 해당하는 Value값을 출력한다.
- D.get('Key') : D라는 딕셔너리의 Key값에 해당하는 Value값을 출력한다.
- D.keys() : D라는 딕셔너리의 Key값을 모두 가져온다.
- D.value() : D라는 딕셔너리의 Value 값을 모두 가져온다.

3. Dictionary

▼ Dictionary 와 List의 사용

```

1 dict1['이름'] = ['나천재', '안기모']
2 dict1['소속'] = ['빅데이터 그룹', '경영혁신 그룹']
3 dict1['월 수익'] = [1500, 2000]
4
5 dict1

```

```
{'이름': ['나천재', '안기모'], '소속': ['빅데이터 그룹', '경영혁신 그룹'], '월 수익': [1500, 2000]}
```

```

1 name = ['안기모']
2 group = ['빅데이터 그룹']
3 income = [2000]
4
5 dict3 = {'이름': name, '소속': group, '월 수익': income}
6 dict3

```

```
{'이름': ['안기모'], '소속': ['빅데이터 그룹'], '월 수익': [2000]}
```

```

1 name.append('나천재')
2 group.append('경영혁신 그룹')
3 income.append(1500)

```

```

1 dict3 = {'이름': name, '소속': group, '월 수익': income}
2 dict3

```

```
{'이름': ['안기모', '나천재'], '소속': ['빅데이터 그룹', '경영혁신 그룹'], '월 수익': [2000, 1500]}
```

딕셔너리

- 딕셔너리의 Value값도 List 형태로 넣을 수 있다.
⇒ 여러 형태의 데이터를 Key - Value 형태로 선언하고, 관리할 수 있다.
- 딕셔너리와 리스트를 이용하여, 해당 Key 값에 Value를 계속 추가 해 줄 수 있다.

4. 산술 관계 연산자

기본 연산자

- Python에서는 기본적인 수학연산을 지원

- 산술 연산자

- 연산의 우선순위 존재 : 거듭제곱 -> 곱셈 및 나눗셈 -> 나머지 및 몫 -> 덧셈 뺄셈 연산
(연산의 순위가 같을 경우, 왼쪽에서 오른쪽으로 연산 진행)

$+$ $-$
 \div \times

- 비교 연산자

- 두 개의 연속형 데이터를 비교
- 명제가 맞는 경우 True,
틀린 경우 False의 Bool 형태 결과를 출력
(Boolean : 참 또는 거짓을 나타내는 자료형)
- Boolean의 경우 조건문에서 많이 사용된다.

산술 연산자		비교 연산자	
+ / -	덧셈 / 뺄셈	a > b	a가 b보다 크다
*	곱셈	a >= b	a가 b와 같거나 크다
/	나눗셈	a == b	a와 b가 같다
//	몫	a != b	a와 b가 다르다
%	나머지	True	위의 문장이 맞을 때
**	거듭제곱	False	위의 문장이 틀릴 때

- 논리 연산자

- And / Or / Not : ~이고, 그리고~ / ~이거나, 또는 / ~이 아니다

4. 산술 관계 연산자

▼ 산술 연산자 연습

```
1 a = 100  
2 b = 35
```

```
1 a+b, a-b
```

```
(135, 65)
```

```
1 a*b, a/b
```

```
(3500, 2.857142857142857)
```

```
1 a//b, a%b
```

```
(2, 30)
```

```
1 a//b
```

```
2
```

▼ 비교 연산자 연습

```
1 a = 50  
2 b = 67  
3  
4 a>b, a<b
```

```
(False, True)
```

```
1 a>=b, a<=b
```

```
(False, True)
```

```
1 a==b
```

```
False
```

```
1 a!=b
```

```
True
```

```
1 type(a!=b)
```

```
bool
```

기본 연산

- 쉼표 (,) 를 이용해, 여러 개의 결과를 동시에 출력할 수 있다.
- 결과에 대한 자료형을 type() 기능을 이용해 파악 할 수 있다.
- 연산이 길어질 땐, 소괄호를 이용하여 연산의 우선순위를 규정해주는 것이 좋다.
⇒ 컴퓨터는 사람처럼 직관적으로 생각하지 않으므로,
연산이 길어질 경우, 순차적 계산에 의한 에러가 발생할 수 있음.

1. 조건문 (If)

조건문

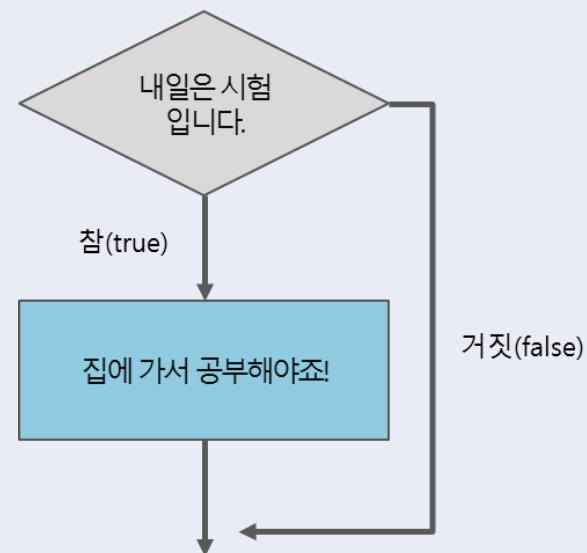
- 조건문 : 특정 상황에 대한 판단 또는 해당 조건에 대한 실행여부 결정
- If문을 사용하여 구현함
- 들여쓰기를 이용해, 해당 조건에 대한 수행문의 종속을 표현
- 주로 비교 연산자와 함께 사용됨
- 특히 데이터 전처리를 할 때, 매우 많이 사용됨

• If 조건문 기본 구조

▼ If 문의 기본 구조

```
if (조건1) :  
    수행할 문장 1  
else :  
    수행할 문장 2
```

⇒ “조건1”을 만족하면, “수행할 문장 1”이 실행됨.
⇒ “조건1”을 만족하지 않으면, “수행할 문장 2”가 실행됨



1. 조건문 (If)

▼ If 조건문을 이용한 연속형 변수 비교

```
1 A = 100
2 B = 200
3
4 if A > B :
5     print("Hi")
6 else:
7     print("안녕하세요")
```

안녕하세요

* 들여쓰기를 이용해 문장의 종속관계를 표현함

▼ 입력 함수를 이용하여 숫자를 비교하는 프로그램

```
1 C = int(input("첫번째 정수를 입력하세요 : "))
2 D = int(input("두번째 정수를 입력하세요 : "))
3
4 if C>D:
5     print("첫번째 정수 ",C,"가 더 큼니다.")
6 else:
7     print("두번째 정수 ",D,"가 더 큼니다.")
8
9 if C==D:
10     print("첫번째 정수와 두번째 정수가 같습니다.")
11 else:
12     print("첫번째 정수와 두번째 정수가 다릅니다.")
```

첫번째 정수를 입력하세요 : 100
두번째 정수를 입력하세요 : 200
두번째 정수 200 가 더 큼니다.
첫번째 정수와 두번째 정수가 다릅니다.

if 조건문

- 일반적인 프로그래밍에 가장 많이 쓰이는 구문
- 기본적으로 if와 else를 이용해, 간단한 비교 구문 제작
- 데이터 전처리를 할 때, apply 함수와 함께 많이 사용됨

1. 조건문 (If)

▼ elif문을 이용한 여러 조건의 비교

```
if (조건1) :  
    수행할 문장 1  
elif (조건2) :  
    수행할 문장 2  
elif (조건3) :  
    수행할 문장 3  
else :  
    수행할 문장 4
```

⇒ “조건1”을 만족하면, “수행할 문장 1” 실행됨
⇒ “조건2”을 만족하면, “수행할 문장 2” 실행됨
⇒ “조건3”을 만족하면, “수행할 문장 3” 실행됨
⇒ “조건1,2,3”을 만족하지 않으면, “수행할 문장 4” 실행됨

▼ if 문을 이용한 성적처리 프로그램

```
1 score = int(input("성적을 입력하세요! : "))  
2  
3 if score >= 95 :  
4     print("학생의 성적은 A+ 입니다.")  
5 elif score >= 90 :  
6     print("학생의 성적은 A 입니다.")  
7 elif score >= 80 :  
8     print("학생의 성적은 B 입니다.")  
9 elif score >= 70 :  
10    print("학생의 성적은 C 입니다.")  
11 else :  
12    print("학생의 성적은 F 입니다.")
```

성적을 입력하세요! : 100
학생의 성적은 A+ 입니다.

▼ Pass 구문의 사용

```
1 A, B = 100, 20  
2  
3 if A > B :  
4     pass  
5 else :  
6     print("Error")
```

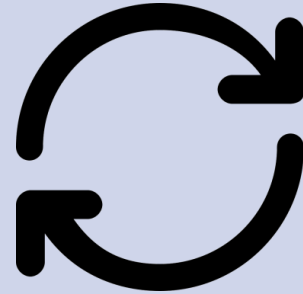
if 조건문

- 여러 가지 조건을 동시에 판별할 땐, elif 구문을 이용한다.
- elif문은 개수의 제한 없이 사용이 가능하다.
- 특정 조건에 수행할 문장을 사용하지 않을 땐, Pass 구문을 이용한다.

2. 반복문 (While, For)

반복문

- 반복문 : 특정 수식이나 기능을 반복 수행할 때 사용
- 특정 조건이 맞을 때, 계속 반복되는 구조
- 들여쓰기를 이용해, 반복 조건에 대한 수행문의 종속을 표현
- 주로 수학연산에서 사용
- 데이터 전처리를 할 때, 문자열 처리에서 많이 사용됨



• While 반복문과 For반복문 기본 구조

▼ While 문의 기본 구조

```
While (조건1):  
    수행할 문장 1  
    수행할 문장 2  
    ...  
    break
```

⇒ “조건1”을 만족하면, 종속된 문장이 반복 실행됨.
⇒ “break” 구문을 만나면, 반복이 멈추게 됨

▼ For 문의 기본 구조

```
For n in range(0,i):  
    수행할 문장 1  
    수행할 문장 2  
    ...
```

⇒ 0에서 부터, i번째 직전까지, n값이 수행문장 변수에 대입되어, 반복 실행 됨.

2. 반복문 (While)

▼ While 반복문

```
1 i = 0
2
3 while i < 5 :
4     print("Hi Python~!")
5     i = i + 1
6
7 print("반복종료")
```

Hi Python~!
Hi Python~!
Hi Python~!
Hi Python~!
Hi Python~!
반복종료

▼ Break문을 이용해 While문을 중단

```
1 i = 1
2 while i > 0:
3     print("Hello")
4     i = i + 1
5
6     if i > 3 :
7         print("강제 탈출 ")
8         break
9
10 print("반복종료 ")
```

Hello
Hello
Hello
강제 탈출
반복종료

While 반복문

- While문은 일반적으로 특정 조건만 만족되면 수행문이 반복되기 때문에, 중간에 수행문을 멈출 조건을 걸어줘야 한다.
- Break문을 이용하면, 바로 While문을 나올 수 있다.

2. 반복문 (For)

▼ For 반복문

```
1 i = 1
2 for i in range(1,10):
3     print(i)
```

1
2
3
4
5
6
7
8
9

▼ 반복의 범위를 변수를 이용해 잡아 준 경우

```
1 n = 10
2 for i in range(0,n+1):
3     print(i)
```

0
1
2
3
4
5
6
7
8
9
10

For 반복문

- For문은 기본적으로 range() 함수와 같이 사용이 된다.
- range(0, n+1) : 0부터 n까지의 범위를 부여한다.
- 반복횟수가 종료되면, 자동으로 수행문이 종료된다.

2. 반복문 (For)

▼ 여러 가지 자료형을 이용한 For 문

```
1 list1 = ['a','b','c','d']
2 for i in list1:
3     print(i)
```

a
b
c
d

```
1 a = [(1,2),(3,4),(5,6)]
2 for (n1, n2) in a :
3     print(n1+n2)
```

3
7
11

▼ For문과 if문을 같이 사용한 경우

```
1 score = [100,23,30,76,34,50]
2 num = 0
3
4 for i in score:
5     num = num + 1
6     if i > 60 :
7         print("합격입니다.")
8     else:
9         print("불합격입니다.")
```

합격입니다.
불합격입니다.
불합격입니다.
합격입니다.
불합격입니다.
불합격입니다.

For 반복문

- 리스트나 튜플 같은 자료형을 이용해 For 반복문을 실행 시킬 수 있다.
- if 문과 같이 쓰여, 자동업무 구현에 많이 사용 된다.

2. 반복문 (For)

▼ 1부터 n까지 더하는 프로그램

```

1 sum = 0
2
3 for i in range(1,101):
4     sum = sum + i
5
6 print(sum)

```

5050

```

1 sum = 0
2
3 for i in range(1,100000001):
4     sum = sum + i
5
6 print(sum)

```

50000000500000000

▼ 가우스 수열을 이용해 더 빠른 연산이 가능

```

1 n = 100000000
2
3 sum = n*(n+1)/2
4
5 print(sum)

```

50000000500000000.0

$$\begin{aligned}
 S &= a_1 + a_2 + a_3 + \cdots + a_{n-1} + a_n \\
 + \quad S &= a_n + a_{n-1} + \cdots + a_3 + a_2 + a_1 \\
 \hline
 2S &= (a_1 + a_n) \times n \\
 \therefore S &= \frac{n(a_1 + a_n)}{2}
 \end{aligned}$$

For 반복문

- 계속 같은 수행문이 반복되기 때문에, 때에 따라서는 처리속도가 매우 느려질 수 있다.
- 최대한 For문을 피하여, 간단하게 코딩을 하는 것이 좋다.

3. 함수

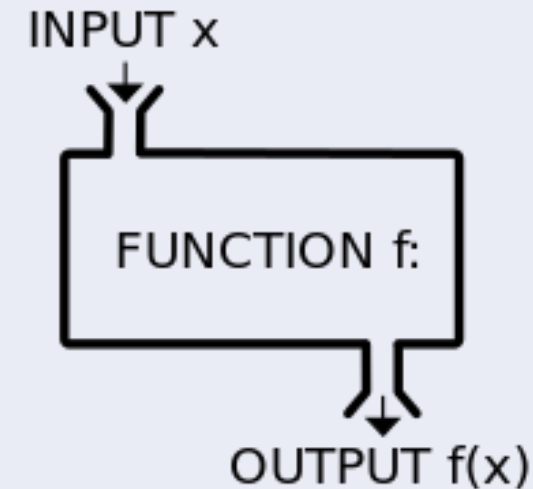
함수

- 함수 : 입력 x 값에 대한 결과 Y 값을 반환(Return)하는 기능을 하나의 묶음으로 선언
- Input x 와 Output Y
- 반복적으로 사용되는 특별한 기능을 함수로 선언
- 여러 함수들의 집합을 '모듈(module)'이라고 부름
- 복잡한 데이터를 전처리할 때, 특정 기능을 반복적으로 구현하고 싶을 때 사용

• 함수의 기본 구조

```
def function1(X) :  
    수행할 문장 1  
    수행할 문장 2  
    return(Y)
```

⇒ 코드에서 function1의 함수를 input값 x 를 넣어 실행하면,
함수 내 수행문이 실행된 후 결과 값 y 가 반환된다.



3. 함수

▼ 입력 받은 수의 합을 구하는 함수

```
1 def sum(num):
2     for i in range(1,num+1):
3         sum = num*(num+1)/2
4     return print(sum)
5
6 num = int(input("정수를 입력하시오 : "))
7 sum(num)
```

정수를 입력하시오 : 100
5050.0

1
2

함수의 이름이 나올 때, 비로소 함수가 작동

코드가 처음으로 시작 되는 부분

▼ 한번 선언된 함수는 계속 사용 가능

```
1 def circle_area(r):
2     return (r**2)*(3.14)
```

```
1 data = 1000
2 circle_area(data)
```

3140000.0

* 한번 선언된 함수는, 다른 코드 입력 창에서 입력을 해도 사용이 가능하다.
* 입력 받는 변수가 바뀌어도, 함수 입력 값은 동일하게 들어간다.

Function

- 함수는 코드가 진행될 때, 함수 이름이 언급되지 않으면, 실행 되지 않는다.
 - ⇒ def 라는 구문은 무시하고 코드가 진행
 - ⇒ 코드 진행 중, 함수이름을 발견했을 때, 함수가 실행
- 함수를 실행하는 것을 '**호출**'(Call) 이라고 부른다.

3. 함수

▼ 두 개의 입력 값을 받아 하나의 결과 값을 출력

```
1 def cylinder_volume(r,h):
2     return (r**2)*(3.14)*(h)
```

```
1 num1 = 1000
2 num2 = 2000
3
4 cylinder_volume(num1,num2)
```

62800000000.0

▼ 날짜 데이터를 함수를 이용해 처리

```
1 def datetime(date):
2     date = str(date)
3     result1 = date[0:4]+'-'+date[4:6]+'-'+date[6:]
4     return result1
5
6 datetime(20190724)
```

'2019-07-24'

▼ 하나의 값을 입력해 여러 개의 결과 값을 출력

```
1 def timetable(data):
2     list1 = [data * 1, data * 2, data * 3]
3     return list1[0], list1[1], list1[2]
4
5 timetable(5)
```

(5, 10, 15)

▼ 입력 값과 결과 값이 없는 경우

```
1 def function1( ):
2     print("hi")
3
4 function1()
```

hi

Function

- 입력 값에는 여러 가지 데이터를 넣을 수 있다.
- 출력되는 데이터의 형태도 여러 형태로 출력할 수 있다.
- 입력 값 또는 결과 값이 없는 형태로도 출력할 수 있다.
- 데이터 전처리에서 매우 많이 사용된다.

빅 데이터

1. 새로운 인사이트와 지식의 원천
2. 의사 결정 개선
3. 경쟁 우위 확보
4. 혁신 촉진
5. 개인화 및 맞춤형 서비스 제공



빅 데이터의 특징

1. Volume (용량)
2. Velocity (속도)
3. Variety (다양성)
4. Veracity (정확성)
5. Value (가치)





데이터 산업

1. 데이터 수집
2. 데이터 저장 및 관리
3. 데이터 처리 및 분석
4. 데이터 시각화 및 보고
5. 데이터 기반 의사결정 지원

데이터 과학자(Data Scientist)

데이터 과학자는 수학, 통계학, 컴퓨터 과학 등의 지식을 바탕으로 복잡한 데이터 분석을 수행하고 기계학습, 데이터 마이닝, 통계적 모델링 등의 기술을 사용하여 데이터로부터 인사이트를 추출하고 예측 모델을 만드는 역할



데이터 엔지니어(Data Engineer)

데이터 엔지니어는 데이터 파이프라인, 데이터 저장소, ETL(Extract, Transform, Load) 프로세스 등 데이터 인프라 구축과 관리를 담당하고 데이터 과학자가 분석할 수 있도록 데이터를 수집, 정제, 저장하는 시스템을 구축



데이터 분석가(Data Analyst)

데이터 분석가는 데이터를 분석하여 비즈니스 인사이트를 제공하고, SQL, Excel, 데이터 시각화 도구 등을 활용하여 데이터를 분석하고, 분석 결과를 기반으로 의사결정을 지원



비즈니스 인텔리전스(BI) 전문가

BI 전문가는 조직 내부의 데이터를 분석하여 비즈니스 성과를 개선하기 위한 인사이트를 제공하고 데이터 시각화 도구를 사용하여 복잡한 데이터를 이해하기 쉬운 형태로 변환하고, 비즈니스 전략 수립을 지원



빅데이터 아키텍트(Big Data Architect)

빅데이터 아키텍트는 전체 데이터 환경의 설계를 담당, 데이터 처리 시스템의 아키텍처 설계, 데이터 모델링, 시스템 통합 등을 통해 데이터의 효율적인 관리와 활용을 도움



빅데이터 플랫폼

1. 데이터 수집(Data Collection)

데이터 소스로부터 데이터를 수집하는 기능

Ex) 로그 파일, 소셜 미디어, IoT(사물인터넷) 장치, 온라인 트랜잭션

2. 데이터 저장(Data Storage)

대용량의 데이터를 효율적으로 저장하기 위한 분산 파일 시스템과 데이터베이스를 제공

Ex) HDFS(Hadoop Distributed File System), NoSQL 데이터베이스(Cassandra, MongoDB), 데이터 웨어하우스

3. 데이터처리 (Data Processing)

수집된 데이터를 처리하고 분석할 수 있는 분산 처리 프레임 워크

MapRedcue, Apache Spark, Apache Flink



빅데이터 플랫폼

4. 데이터 분석 (Data Analysis)

데이터를 분석하여 인사이트를 도출하고, 머신 러닝 모델을 개발할 수 있는 도구와 서비스를 제공

Ex) Python, R 프로그래밍 언어, Jupyter Notebook과 같은 분석 도구

5. 데이터 시각화 (Data Visualization)

분석 결과를 시각적으로 표현하여 이해하기 쉽게 만드는 도구를 제공

Ex) Tableau, Power BI, Apache Superset



인공지능(Artificial Intelligence, AI)

기계가 인간과 유사한 지능적인 행동을 수행할 수 있게 하는 기술 분야

1. 기계 학습(Machine Learning, ML)
2. 딥러닝(Deep Learning)
3. 자연어 처리(Natural Language Processing, NLP)



주요 법률 및 제도

기술의 발전과 더불어 다양한 법률과 제도가 도입되고 있음

1. 일반 데이터 보호 규정(GDPR)
2. 캘리포니아 소비자 프라이버시 법안(CCPA)
3. 한국의 개인정보 보호법(PIPA)



주요 법률 및 제도

AI와 데이터분석에서의 법률과 규제

1. 데이터 최소화 원칙
2. 익명화 및 가명 처리
3. 투명성과 정보 주체의 권리
4. 보안 조치



데이터 최소화 원칙

AI와 데이터분석에서의 법률과 규제



익명화

AI와 데이터분석에서의 법률과 규제

1. 직접 식별자 제거
2. 간접 식별자 처리
3. 데이터 세트의 재구성
4. 데이터 축소
5. 리스크 평가
6. 지속적인 모니터링



데이터 마스킹

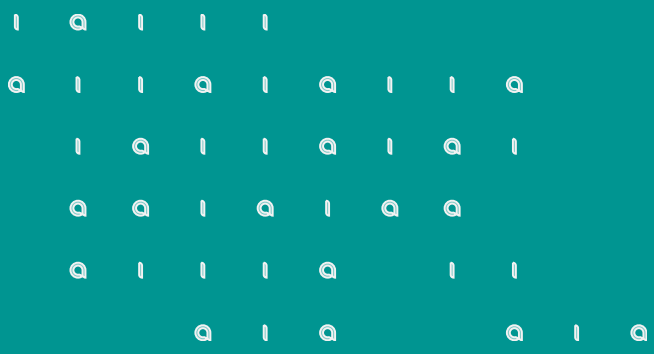
AI와 데이터분석에서의 법률과 규제

Un-Masked Data

name: Jane Smith
ssn: 555-55-0123
credit card: 0012-3456-7891-2345
address: 12529 Oak Rd. AZ
date of birth: 03-29-1964
e-mail: j.smith@mail.com

Masked Data

name: Joan Stevens
ssn: 777-89-1234
credit card: 3456-0123-3456-9876
address: 25352 Willow Dr. CA
date of birth: 04-30-1964
e-mail: j.stevens@mail1.com



DataFrame Handling



1. DataFrame

DataFrame

- **DataFrame** : 엑셀과 같이, 인덱스(Index), 변수(Column), 값(Value)로 이루어진 데이터 구조
- Pandas DataFrame의 장점

-> 대용량 데이터를 빠르고 쉽게 다룰 수 있다.

한계용량 : 엑셀 약 100MB / Pandas 1GB ~ 100GB

-> 복잡한 기능을 구현하기가 쉽고, 데이터 전처리를 쉽게 할 수 있다.

-> 다른 시스템과 연동이 쉽다.

Flask 라이브러리 : 웹 개발 / SQLAlchemy : 데이터베이스 / Sklearn : 머신러닝

- Numpy 라이브러리에서 지원하는 수학 및 통계 연산을 그대로 이용 할 수 있다.
- 가장 구조적인 데이터 형태로서, 직관적이고 설명하기 쉬우며, csv나 excel 형태로 저장 할 수 있다.

1. DataFrame

▼ 리스트를 이용한 DataFrame 생성

```
1 import pandas as pd
```

```
1 num3 = [  
2     [1,2,3],  
3     [4,5,6],  
4     [7,8,9]  
5 ]  
6 num3
```

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

```
1 pd.DataFrame(num3)
```

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

DataFrame

- `pd.DataFrame()` : 데이터 프레임을 생성하는 함수
- 리스트 뿐만 아니라, 다른 여러 방식으로 데이터프레임을 생성할 수 있다.
- csv 파일이나 Excel 형태 SQL 등 여러 형식의 파일도 데이터프레임 형태로 불러 올 수 있다.

1. DataFrame

▼ 이중 리스트를 이용한 데이터프레임 생성

```

1 table1 = [
2     ['2019-01-01', 1000, 'False', 'gum'],
3     ['2019-01-04', 3000, 'True', 'snack'],
4     ['2019-01-06', 2000, 'True', 'beverage'],
5     ['2019-01-06', 1000, 'True', 'gum']
6 ]
7 pd.DataFrame(table1)

```

	0	1	2	3
0	2019-01-01	1000	False	gum
1	2019-01-04	3000	True	snack
2	2019-01-06	2000	True	beverage
3	2019-01-06	1000	True	gum

▼ Column 이름을 리스트 형태로 붙인 모습

```

1 pd.DataFrame(table1, columns=['일자', '가격', '구매여부', '제품'])

```

	일자	가격	구매여부	제품
0	2019-01-01	1000	False	gum
1	2019-01-04	3000	True	snack
2	2019-01-06	2000	True	beverage
3	2019-01-06	1000	True	gum

▼ 딕셔너리 형태로 데이터프레임 생성

```

1 table2 = {
2     '일자': ['2019-01-01', '2019-01-04', '2019-01-06', '2019-01-06'],
3     '가격': [1000, 3000, 2000, 1000],
4     '구매여부': ['False', 'True', 'True', 'True'],
5     '제품': ['gum', 'snack', 'beverage', 'gum']
6 }
7 pd.DataFrame(table2)

```

	일자	가격	구매여부	제품
0	2019-01-01	1000	False	gum
1	2019-01-04	3000	True	snack
2	2019-01-06	2000	True	beverage
3	2019-01-06	1000	True	gum

▼ 변수로 선언된 리스트를 이용해, 이름을 붙인 모습

```

1 name = ["A", "B", "C", "D"]
2
3 pd.DataFrame(table2, index = name )

```

	일자	가격	구매여부	제품
A	2019-01-01	1000	False	gum
B	2019-01-04	3000	True	snack
C	2019-01-06	2000	True	beverage
D	2019-01-06	1000	True	gum

1. DataFrame - Table Information

DataFrame

• DataFrame의 3가지 구성요소 : 인덱스(Index), 변수(Column), 값(Value)

- Index : 데이터의 순서를 나타내는 지표
- Column : 데이터의 항목을 나타내는 지표
- Value : 데이터 값 (Numpy의 배열 형태)

- 데이터 분석 전, 데이터를 파악할 때,
이 3가지 요소에 대해 가장 먼저 확인해 봐야 함

- Index와 Column은 Pandas Core 형태로 구성, Value는 Numpy의 ndarray로 구성 됨

	column			
	일자	가격	구매여부	제품
A	2019-01-01	1000	False	gum
B	2019-01-04	3000	True	snack
C	2019-01-06	2000	True	beverage
D	2019-01-06	1000	True	gum

value

```
1 type(df1.columns)
pandas.core.indexes.base.Index

1 type(df1.index)
pandas.core.indexes.base.Index

1 type(df1.values)
numpy.ndarray
```

```
1 df1.columns
Index(['일자', '가격', '구매여부', '제품'], dtype='object')

1 df1.index
Index(['A', 'B', 'C', 'D'], dtype='object')

1 df1.values
array([[ '2019-01-01', 1000, 'False', 'gum'],
       [ '2019-01-04', 3000, 'True', 'snack'],
       [ '2019-01-06', 2000, 'True', 'beverage'],
       [ '2019-01-06', 1000, 'True', 'gum']], dtype=object)
```

1. DataFrame - Table Information

▼ 데이터의 상위 5개의 값만 출력

1 df1.head()

	일자	가격	구매여부	제품
A	2019-01-01	1000	False	gum
B	2019-01-04	3000	True	snack
C	2019-01-06	2000	True	beverage
D	2019-01-06	1000	True	gum

1 df1.head(2)

	일자	가격	구매여부	제품
A	2019-01-01	1000	False	gum
B	2019-01-04	3000	True	snack

1 df1.tail(2)

	일자	가격	구매여부	제품
C	2019-01-06	2000	True	beverage
D	2019-01-06	1000	True	gum

▼ 특정 Column에 대해 확인

1 df1['제품']

```
A    gum
B    snack
C    beverage
D    gum
Name: 제품, dtype: object
```

1 df1['가격'].head()

```
A    1000
B    3000
C    2000
D    1000
Name: 가격, dtype: int64
```

▲ 특정 변수의 값만 head() 와 tail() 함수로 출력할 수 있다.

◀ head() 함수와, tail() 함수 내부에 숫자를 넣어주면, 해당 숫자만큼 데이터가 출력된다.

- dataframe.head() : 데이터 프레임의 상위 5개의 값을 출력한다.
- dataframe.tail() : 데이터 프레임의 하위 5개의 값을 출력한다.

2. DataFrame Handling - Sorting

▼ 데이터를 Index 기준 순서로 정렬하고 싶을 때

```
1 df1.sort_index()
```

	일자	가격	구매여부	제품
A	2019-01-01	1000	False	gum
B	2019-01-04	3000	True	snack
C	2019-01-06	2000	True	beverage
D	2019-01-06	1000	True	gum

▼ 데이터를 특정 Column 을 기준으로 정렬할 때

```
1 df1.sort_values(by='일자')
```

	일자	가격	구매여부	제품
A	2019-01-01	1000	False	gum
B	2019-01-04	3000	True	snack
C	2019-01-06	2000	True	beverage
D	2019-01-06	1000	True	gum

▼ ascending 함수를 이용하면, 오름차순/내림차순 정렬을 할 수 있다.

```
1 df1.sort_index(ascending=False)
```

	일자	가격	구매여부	제품
D	2019-01-06	1000	True	gum
C	2019-01-06	2000	True	beverage
B	2019-01-04	3000	True	snack
A	2019-01-01	1000	False	gum

▼ 리스트를 이용해 두 column을 기준으로 정렬

```
1 df1.sort_values(by=['일자', '가격'])
```

	일자	가격	구매여부	제품
A	2019-01-01	1000	False	gum
B	2019-01-04	3000	True	snack
D	2019-01-06	1000	True	gum
C	2019-01-06	2000	True	beverage

Sorting

- dataframe.sort_index() : dataframe을 인덱스를 기준으로 정렬한다.
⇒ ascending : 오름차순 / 내림차순 정렬
- dataframe.sort_value() : dataframe을 특정 값을 기준으로 정렬한다.
⇒ by 함수 : 정렬하고자 하는 Column을 입력

2. DataFrame Handling - Operation

▼ 연속형 데이터에 대한 기본 통계 연산

```
1 df1['가격'].mean()
```

1750.0

```
1 df1['가격'].sum()
```

7000

```
1 df1['가격'].min()
```

1000

```
1 df1['가격'].max()
```

3000

```
1 df1['가격'].std()
```

957.427107756338

▼ 범주형 데이터에 대한 연산

```
1 df1['제품'].unique()
```

array(['gum', 'snack', 'beverage'], dtype=object)

```
1 df1['제품'].value_counts()
```

```
gum      2
beverage 1
snack     1
Name: 제품, dtype: int64
```

▼ Describe 함수를 이용한 요약 정리

```
1 df1['가격'].describe()
```

```
count      4.000000
mean       1750.000000
std         957.427108
min         1000.000000
25%         1000.000000
50%         1500.000000
75%         2250.000000
max         3000.000000
Name: 가격, dtype: float64
```

Operation

- dataframe.describe() : dataframe에 대한 모든 연속형 column에 대해 요약통계량을 출력한다.
- dataframe['col1'].unique() : dataframe에 명목형 column의 항목을 출력한다.
- dataframe['col1'].value_counts() : dataframe의 명목형 column의 항목 개수를 출력한다.

2. DataFrame Handling - Indexing

▼ 특정 변수에 대해 데이터 값을 확인

```
1 df1['제품']
```

A gum
B snack
C beverage
D gum
Name: 제품, dtype: object

```
1 df1[['제품', '가격']]
```

	제품	가격
A	gum	1000
B	snack	3000
C	beverage	2000
D	gum	1000

▲ 위와 같이 여러 column을 하나의 프레임형태로 출력할 수 있다.

▼ 범주형 데이터에 대한 연산

```
1 table2 = df1[['제품', '가격']]
2 table2
```

	제품	가격
A	gum	1000
B	snack	3000
C	beverage	2000
D	gum	1000

▼ 대괄호의 개수에 따라 Dataframe / Series 형식이 결정된다.

```
1 type(df1['가격'])
```

pandas.core.series.Series

```
1 type(df1[['가격']])
```

pandas.core.frame.DataFrame

Indexing

- 실무에서는 불필요한 column이 많아, 원하는 column만 추출해 올 필요가 있다.
- `dataframe['column1']` : dataframe의 column1 데이터만 가져온다.
- `dataframe[['column1','column2']]` : dataframe의 column1과 column2의 데이터를 가져온다.
- Column을 두 개 이상 가져올때, 대괄호를 두 개 붙여줘야 한다. (Series형식이 불가능 하기 때문)

2. DataFrame Handling - Indexing

▼ loc 함수를 이용한 Indexing

```
1 df1.loc['A']
```

일자 2019-01-01
가격 1000
구매여부 False
제품 gum
Name: A, dtype: object

```
1 df1.loc[['A','D']]
```

	일자	가격	구매여부	제품
A	2019-01-01	1000	False	gum
D	2019-01-06	1000	True	gum

▼ 범위를 지정하여 Indexing 가능

```
1 df1.loc['A':'C']
```

	일자	가격	구매여부	제품
A	2019-01-01	1000	False	gum
B	2019-01-04	3000	True	snack
C	2019-01-06	2000	True	beverage

```
1 df1["제품"] ["B"]
```

'snack'

```
1 df1.loc["B", "제품"]
```

'snack'

```
1 df1.at["B", "제품"]
```

'snack'

여러 다른 형태로도 Indexing 가능

Indexing

- `dataframe.loc[]` : 해당 Index의 row 데이터를 불러온다.
- 특정 위치의 데이터를 Indexing 할 때, 많이 쓰이는 함수이다. (location의 약자)
- `at` 함수도 같은 기능을 하지만, 여러 데이터를 불러올 땐, `loc` 함수만 작동한다.

2. DataFrame Handling - Indexing

▼ loc 함수는 여러 데이터를 불러 올 수 있다

```
1 df1.loc[["B", "A"], "제품"]
```

```
B    snack
A      gum
Name: 제품, dtype: object
```

```
1 df1.loc[["A", "B"], ["제품", "가격"]]
```

	제품	가격
A	gum	1000
B	snack	3000

```
1 ind = ["A", "B"]
2 col = ["제품", "가격"]
3
4 df1.loc[ind, col]
```

	제품	가격
A	gum	1000
B	snack	3000

▼ at 함수는 하나의 데이터만 불러올 수 있다

```
1 df1.at[["B", "A"], "제품"]
```

```
TypeError                                Traceback (most recent call last)
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in _get_value(self,
index, col, takeable)
    2535         try:
-> 2536             return engine.get_value(series._values, index)
    2537         except (TypeError, ValueError):

pandas\libs\index.pyx in pandas._libs.index.IndexEngine.get_value()

pandas\libs\index.pyx in pandas._libs.index.IndexEngine.get_value()

pandas\libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

TypeError: '['B', 'A']' is an invalid key
```

Indexing

- `dataframe.loc[]` : 해당 Index의 row 데이터를 불러온다.
- 특정 위치의 데이터를 Indexing 할 때, 많이 쓰이는 함수이다. (location의 약자)
- `at` 함수도 같은 기능을 하지만, 여러 데이터를 불러올 땐, `loc` 함수만 작동한다.

2. DataFrame Handling - Indexing

▼ 비교 연산자를 이용한 Indexing

```
1 df1['가격'] > 2000
```

```
A    False
B     True
C    False
D    False
Name: 가격, dtype: bool
```

```
1 df1['제품'] == 'gum'
```

```
A     True
B    False
C    False
D     True
Name: 제품, dtype: bool
```

```
1 df1['제품'] != 'gum'
```

```
A    False
B     True
C     True
D    False
Name: 제품, dtype: bool
```

```
1 df1[df1['가격'] > 2000]
```

	일자	가격	구매여부	제품
B	2019-01-04	3000	True	snack

```
1 df1[df1['제품'] == 'gum']
```

	일자	가격	구매여부	제품
A	2019-01-01	1000	False	gum
D	2019-01-06	1000	True	gum

```
1 df1[df1['제품'] != 'gum']
```

	일자	가격	구매여부	제품
B	2019-01-04	3000	True	snack
C	2019-01-06	2000	True	beverage

◀◀ 특정 column의 조건을 걸어 bool형태로 표현 할 수 있다.

◀ dataframe을 대괄호 밖에 한번 더 언급해주면, 해당 조건을 만족하는 데이터가 출력된다.

◀ 특정 항목이 일치 여부는 등호를 두 번 써서(==) 확인할 수 있다.

◀ “!=” 기호를 이용해 불일치 여부도 확인 할 수 있다.

2. DataFrame Handling - Indexing

▼ 비교 연산자를 이용한 Indexing

```
1 df1[df1['제품'].isin(['snack', 'gum'])]
```

	일자	가격	구매여부	제품
A	2019-01-01	1000	False	gum
B	2019-01-04	3000	True	snack
D	2019-01-06	1000	True	gum

```
1 data = ['gum', 'snack']
2 df1[df1['제품'].isin(data)]
```

	일자	가격	구매여부	제품
A	2019-01-01	1000	False	gum
B	2019-01-04	3000	True	snack
D	2019-01-06	1000	True	gum

```
1 data = ['gum', 'snack']
2 df1[~df1['제품'].isin(data)]
```

	일자	가격	구매여부	제품
C	2019-01-06	2000	True	beverage

◀ isin() 함수를 이용해, 특정 Column에 있는 여러 데이터를 확인할 수 있다.

◀ 확인하고자 하는 데이터를 리스트 형태로 사용할 수 있다.

◀ “~” 기호를 이용하면, 해당 조건의 반대가 되는 값을 출력한다.

2. DataFrame Handling - Add / Change / Delete

▼ 데이터를 새로운 변수에 추가

```
1 df1['환불여부'] = "정상"
2 df1
```

	일자	가격	구매여부	제품	환불여부
A	2019-01-01	1000	False	gum	정상
B	2019-01-04	3000	True	snack	정상
C	2019-01-06	2000	True	beverage	정상
D	2019-01-06	1000	True	gum	정상

```
1 df1['환불여부'] = ['정상', '환불', '정상', '환불']
2 df1
```

	일자	가격	구매여부	제품	환불여부
A	2019-01-01	1000	False	gum	정상
B	2019-01-04	3000	True	snack	환불
C	2019-01-06	2000	True	beverage	정상
D	2019-01-06	1000	True	gum	환불

▼ 데이터를 삭제

```
1 df1.drop("B",axis=0)
```

	일자	가격	구매여부	제품	환불여부	판매여부
A	2019-01-01	1000	False	gum	정상	True
C	2019-01-06	2000	True	beverage	정상	True
D	2019-01-06	1000	True	gum	환불	False

```
1 df1.drop("구매여부",axis=1)
```

	일자	가격	제품	환불여부	판매여부
A	2019-01-01	1000	gum	정상	True
B	2019-01-04	3000	snack	환불	False
C	2019-01-06	2000	beverage	정상	True
D	2019-01-06	1000	gum	환불	False

Add / Delete

- 데이터를 새로운 Column에 추가하려면, 위와 같이 등호 기호를 이용해 생성
- drop() : 기존의 데이터를 Index나 Column 중심으로 삭제 (axis=0 : index / axis=1 : column)

2. DataFrame Handling - Add / Change / Delete

▼ 문자열 Replace 함수를 이용한 특정 값 변경

```
1 df1['판매여부'].replace(True,1)
```

```
A    1.0
B    0.0
C    1.0
D    0.0
Name: 판매여부, dtype: float64
```

```
1 df1['판매여부(replace)'] = df1['판매여부'].replace(True,1)
2 df1
```

	일자	가격	구매여부	제품	환불여부	판매여부	판매여부(replace)
A	2019-01-01	1000	False	gum	정상	True	1.0
B	2019-01-04	3000	True	snack	환불	False	0.0
C	2019-01-06	2000	True	beverage	정상	True	1.0
D	2019-01-06	1000	True	gum	환불	False	0.0

```
1 df1['제품'].replace('snack','스낵')
```

```
A      gum
B      스낵
C  beverage
D      gum
Name: 제품, dtype: object
```

Change

- `dataframe['Column'].replace('A','B')` : dataframe의 column에서, A의 모든 항목값을 B로 변환
- 변환된 내용을 새로운 테이블에 선언하여, 지속적 사용 가능

2. DataFrame Handling - Add / Change / Delete

▼ 문자열 Replace 함수를 이어 사용하여 특정 값 변경

```
1 df1['제품'].replace('snack', '스낵').replace('gum', '껌').replace('beverage', '음료')
```

```
A    껌
B    스낵
C    음료
D    껌
Name: 제품, dtype: object
```

```
1 df1['제품(replace)'] = df1['제품'].replace('snack', '스낵').replace('gum', '껌').replace('beverage', '음료')
2 df1
```

	일자	가격	구매여부	제품	환불여부	판매여부	판매여부(replace)	제품(replace)
A	2019-01-01	1000	False	gum	정상	True	1.0	껌
B	2019-01-04	3000	True	snack	환불	False	0.0	스낵
C	2019-01-06	2000	True	beverage	정상	True	1.0	음료
D	2019-01-06	1000	True	gum	환불	False	0.0	껌

Change

- replace 함수를 비롯한, 대부분의 Pandas 함수가 뒤에 이어 붙여 사용할 수 있다.
- 범주형 데이터에서 여러 값을 동시에 바꿀 때, 주로 사용된다.
- 변경된 값들을 새로운 변수에 선언 할 수 있다.
- 데이터 분석 전 데이터 정제단계에서 매우 많이 사용된다.

2. DataFrame Handling - Data Load

Data 불러오기

- Pandas를 이용해 CSV 나 Excel 형식의 데이터 파일을 불러 올 수 있음
- 필요한 경우에는 데이터 파일간 병합(Join)을 수행 해야 함
- read_csv() 명령어를 이용하여, 데이터를 불러옴
- 데이터를 불러올 때, 데이터와 주피터노트북 파일이 같은 폴더에 위치해야 함
(위치가 다를 경우, 상대 경로를 지정해 주어야 한다.)
- 기업에서 사용하는 데이터는 대부분 CSV나 Excel 형태로 추출이 가능
- 공공데이터포털이나, 기상청 등 국가기관에서 공공데이터를 Open

CRM Data : 회사에서 사용되는 고객과 거래 또는 고객과 회사의 관계를 나타내는 모든 데이터

```

1 data = pd.read_csv('store_market_data.csv')
2 data

```

	공급 일자	요 일	공 급 월	공급 주자	회원번호	조합 원상 태	물품 대분 류	물품중 분류	물품소 분류	물품명	구매 수량	주 소 구	주 소 동	성 별	연령	연령 대	구매 금액	구 매 매 장	반품_원거 래일자	구매 시각
0	2018-01-02	화	1	1	272369856	정상 회원	과실	과일	사과	사과/유(1.5kg)	1.0	수 지 구	풍 곡 동	여	45.0	40 대	22207	매 장 C	NaN	10:04
1	2018-01-02	화	1	1	1506656256	정상 회원	채소	버섯	느타리 버섯	느타리버섯(300g)	1.0	수 지 구	풍 곡 동	여	36.0	30 대 이 하	4977	매 장 C	NaN	10:05
2	2018-01-02	화	1	1	1506656256	정상 회원	축산 물	알	유정란	유정란/매장용(10알/ 국내산)	1.0	수 지 구	풍 곡 동	여	36.0	30 대 이 하	7083	매 장 C	NaN	10:05
3	2018-01-02	화	1	1	1023108864	정상 회원	반찬	두부/유 부	두부	연두부(100g)	1.0	수 지 구	풍 곡 동	여	36.0	30 대 이 하	766	매 장 C	NaN	10:08

2. DataFrame Handling - Data Load

▼ 데이터 불러오기

```
1 data = pd.read_csv('store_market_data.csv')
1 data = pd.read_csv('C:\Users\ai0001\Desktop\Jupyter File\store_market_data.csv')
```

URL 주소 :
http://46.101.230.157/dilan/pandas_tutorial_read.csv

▼ URL 데이터 불러오기

```
1 url_data = 'http://46.101.230.157/dilan/pandas_tutorial_read.csv'
2 table1 = pd.read_csv(url_data)
3 table1.head()
```

2018-01-01 00:01:01;read;country_7;2458151261;SEO;North America

0	2018-01-01 00:03:20;read;country_7;2458151262;...
1	2018-01-01 00:04:01;read;country_7;2458151263;...
2	2018-01-01 00:04:02;read;country_7;2458151264;...
3	2018-01-01 00:05:03;read;country_8;2458151265;...
4	2018-01-01 00:05:42;read;country_6;2458151266;...

```
1 table1 = pd.read_csv(url_data, delimiter=';')
2 table1.head()
```

	2018-01-01 00:01:01	read	country_7	2458151261	SEO	North America
0	2018-01-01 00:03:20	read	country_7	2458151262	SEO	South America
1	2018-01-01 00:04:01	read	country_7	2458151263	AdWords	Africa
2	2018-01-01 00:04:02	read	country_7	2458151264	AdWords	Europe
3	2018-01-01 00:05:03	read	country_8	2458151265	Reddit	North America
4	2018-01-01 00:05:42	read	country_6	2458151266	Reddit	North America

Data Load

- `pd.read_csv(" ")` : 해당 CSV 파일을 불러온다.
- URL 링크의 데이터도 하나의 변수로 선언해 불러올 수 있다.
- CSV 형식에 맞게, 구분자를 넣어 불러올 수 있다. (`delimiter` 함수)

2. DataFrame Handling - Data Load

▼ 데이터 불러오기

```
1 table1 = pd.read_csv(url_data, delimiter=';', header=None)
2 table1.head()
```

	0	1	2	3	4	5
0	2018-01-01 00:01:01	read	country_7	2458151261	SEO	North America
1	2018-01-01 00:03:20	read	country_7	2458151262	SEO	South America
2	2018-01-01 00:04:01	read	country_7	2458151263	AdWords	Africa
3	2018-01-01 00:04:02	read	country_7	2458151264	AdWords	Europe
4	2018-01-01 00:05:03	read	country_8	2458151265	Reddit	North America

```
1 table1 = pd.read_csv(url_data, delimiter=';',
2                       names = ['my_datetime', 'event', 'country', 'user_id', 'source', 'topic'])
3 table1.head()
```

	my_datetime	event	country	user_id	source	topic
0	2018-01-01 00:01:01	read	country_7	2458151261	SEO	North America
1	2018-01-01 00:03:20	read	country_7	2458151262	SEO	South America
2	2018-01-01 00:04:01	read	country_7	2458151263	AdWords	Africa
3	2018-01-01 00:04:02	read	country_7	2458151264	AdWords	Europe
4	2018-01-01 00:05:03	read	country_8	2458151265	Reddit	North America

```
1 table1 = pd.read_csv(url_data, delimiter=';', names=name_1, skiprows=1)
2 table1.head()
```

	my_datetime	event	country	user_id	source	topic
0	2018-01-01 00:03:20	read	country_7	2458151262	SEO	South America
1	2018-01-01 00:04:01	read	country_7	2458151263	AdWords	Africa
2	2018-01-01 00:04:02	read	country_7	2458151264	AdWords	Europe
3	2018-01-01 00:05:03	read	country_8	2458151265	Reddit	North America
4	2018-01-01 00:05:42	read	country_6	2458151266	Reddit	North America

URL 주소 :

http://46.101.230.157/dilan/pandas_tutorial_read.csv

◀ header를 생략하고 싶을 때

◀ Column명을 names함수를 이용해 지정할 수 있다

◀ Column명을 제외하고, 첫번째 열(row)데이터를 삭제할 수 있다

2. DataFrame Handling - Pivoting

Pivoting

- 데이터 분석에 가장 많이 사용되는 함수로, 데이터를 특정 항목에 대해 요약정리 해준다.
- 월별 매출액 / 항목별 판매량 / 라인 별 불량률 등 원하는 목표변수에 대한 지표를 직관적으로 볼 수 있다.
- Group by와 Pivot table의 두 가지 기능으로 구현 할 수 있다.

Group by : 항목 별 값에 대한 결과를 Series 형태로 출력

Pivot table : 항목 별 값에 대한 결과를 DataFrame 형태로 출력

Pivot

df

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

df.pivot(index='foo', columns='bar', values='baz')

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

2. DataFrame Handling - Group By

▼ store_market_data.csv 데이터 불러오기

```
1 table1 = pd.read_csv('store_market_data.csv')
2 table1.head()
```

	공급 일자	요일	공급 월	공급 주자	회원번호	조합 원상 태	물품 대분 류	물품 중분 류	물품소 분류	물품명	구매 수량	주 수	주 상	연령 별	연령 대	구매 금액	구매 매장	반품_ 원거래 일자	구매 시간	
0	2018-01-02	화	1	1	272369856	정상 회원	과실	과일	사과	사과/유 (1.5kg)	1.0	수 지 구	하 위 연 장 자	여	45.0	40 대	22207	매 장 C	NaN	10:04
1	2018-01-02	화	1	1	1506656256	정상 회원	채소	버섯	느타리 버섯	느타리버섯 (300g)	1.0	수 지 구	하 위 연 장 자	여	36.0	30 대 이 하	4977	매 장 C	NaN	10:05
2	2018-01-02	화	1	1	1506656256	정상 회원	축산 물	알	유정란	유정란/매장용 (10알/국내산)	1.0	수 지 구	하 위 연 장 자	여	36.0	30 대 이 하	7083	매 장 C	NaN	10:05
3	2018-01-02	화	1	1	1023108864	정상 회원	반찬	두부/ 유부	두부	연두부(100g)	1.0	수 지 구	하 위 연 장 자	여	36.0	30 대 이 하	766	매 장 C	NaN	10:08
4	2018-01-02	화	1	1	1476143616	정상 회원	간식	빵	식빵, 식사대 용	아침빵 (240g/8개)매장	1.0	수 지 구	하 위 연 장 자	여	34.0	30 대 이 하	4403	매 장 C	NaN	10:09

```
1 table1.groupby('요일').mean()
```

	공급월	공급주자	회원번호	구매수량	연령	구매금액	반품_원거래일자
요일							
금	3.679024	13.909721	7.366242e+08	1.138407	50.968497	11064.344340	2.018037e+07
목	3.570806	13.614572	7.317310e+08	1.129928	51.099647	10857.069337	2.018032e+07
수	3.292904	12.790732	7.434917e+08	1.144800	50.733620	11062.047300	2.018035e+07
월	3.521324	14.006212	7.508196e+08	1.131904	50.381287	10542.528917	2.018038e+07
일	3.572221	13.222756	7.566981e+08	1.144827	50.118775	10939.249736	2.018030e+07
토	3.558841	13.502375	7.203877e+08	1.148571	50.972849	11046.589839	2.018036e+07
화	3.410518	13.226825	7.388714e+08	1.134223	50.783786	11002.352378	2.018030e+07

▼ 특정 Column별 Group by

```
1 table1.groupby('요일')['구매금액'].mean()
```

```
요일
구매금액
수    11062.047300
월    10542.528917
일    10939.249736
토    11046.589839
화    11002.352378
Name: 구매금액, dtype: float64
```

```
1 table1.groupby('요일')['구매금액'].sum()
```

```
요일
구매금액
수    1215650577
월    1130698629
일    1161404346
토    1305186165
화    591561808
목    1076943090
화    1223417575
Name: 구매금액, dtype: int64
```

```
1 table1.groupby(['요일', '연령대'])['구매금액'].sum()
```

```
요일  연령대
금    30대이하    158434038
      40대      449759043
      50대      319196997
      60대      170808002
      70대이상    117390184
목    30대이하    150301936
      40대      416901325
      50대      293524482
      60대      157682379
      70대이상    112148947
수    30대이하    156665842
      40대      436644537
      50대      298765854
      60대      168640554
      70대이상    100567716
월    30대이하    195332433
      40대      599999999
```

2. DataFrame Handling - Pivot Table

▼ Pivot table 사용, T 함수를 이용해 결과를 row 형태로 정렬

```
1 pd.pivot_table(data=table1, index='요일', values='구매금액').T
```

	요일	금	목	수	월	일	토	화
구매금액	11064.34434	10857.069337	11062.0473	10542.528917	10939.249736	11046.589839	11002.352378	

▼ column에 값을 채워 사용 할 수 있다

```
1 pd.pivot_table(data=table1, index='요일', values='구매금액', columns='연령대')
```

	연령대	30대이하	40대	50대	60대	70대이상
요일						
금	9858.994275	10714.160822	11557.989535	11792.060891	12084.639078	
목	9832.009943	10611.956549	11494.536419	11125.547097	11378.748681	
수	9939.464662	10848.850552	11600.304950	11392.322772	11958.111296	
월	9759.789797	10208.078631	11029.804540	11177.027319	11331.251843	
일	9674.993523	10645.606520	11986.038184	11352.549424	11058.976769	
토	10055.889145	10817.761716	11859.119222	11005.123723	11159.196517	
화	9814.932993	10680.950792	11766.328357	11530.118531	11443.887802	

▼ aggfunc 함수를 이용해, 특정 통계량에 대해 계산할 수 있다

```
1 pd.pivot_table(data=table1, index='요일', values='구매금액', aggfunc='sum')
```

	구매금액
요일	
금	1215650577
목	1130698629
수	1161404346
월	1305186165
일	591561808
토	1076943090
화	1223417575

Pivot Table

- `pivot_table(data= df , value= 'col1', index= 'col2')`
=> df 데이터 테이블에, col1을 값으로, col2를 index로 하는 테이블 생성
- aggfunc 함수를 이용해, 합이나, 최대,최솟값, 표준편차 등을 구할 수 있다.

3. Data Preprocessing - Missing Value

결측값 (Missing Value)

- **결측값** : 데이터가 수집/저장/처리/분석 되는 과정에서 오류나 누락으로 인해 데이터에 발생한 공백
- **표현 방법** : NA (Not Available) : 기록 되지 않음
 - NaN (Not a Number) : 수학적으로 정의되지 않음
 - Inf (Infinite) : 무한대
 - Null : 값이 없음
- **처리 방법**
 - **제거** : 결측치가 있는 행 또는 열 자체를 제거
 - **대치** : 결측치를 특정 값으로 대치
- 비조건 부 평균 대치법 : 단순 평균값이나 중앙값을 대치
- 조건부 평균 대치법 : 회귀식을 이용하여 값을 대치
- 보간법 : 실측값 사이의 간격에 따라 결측값을 대치
- 단순 확률 대치법 :
 - Hot Deck : 표본조사에서 흔히 사용되며, 비슷한 값들에 확률을 부여해 대치
 - Cold Deck : 외부의 출처나 이전의 연구에서 인용된 자료를 가져와 대치
 - Mix (혼합방법) : 조건부평균 대치법을 이용해 값을 가져오고, Hot Deck 을 이용해 잔차를 더함

3. Data Preprocessing - Missing Value

▼ 결측값 확인

```
import pandas as pd
```

```
df1 = pd.read_csv('01_Contract_Data.csv')
```

```
df1.isnull().sum() # 특정 항목별 결측치의 개수를 확인
```

```
Index          0
Member_ID      0
Sales_Type     0
Contract_Type  0
Channel        0
Datetime       0
Term           0
Payment_Type   0
Product_Type   0
Amount_Month   0
Customer_Type  2
Age           6972
Address1       2
Address2       2
State          0
Overdue_count  0
Overdue_Type   0
Gender         0
Credit_Rank    8781
Bank          2759
dtype: int64
```

3. Data Preprocessing - Missing Value

▼ 결측값 제거

```
df2 = df1.dropna() # 결측치가 있는 행을 찾아서 제거
```

```
df2.isnull().sum()
```

```
Index          0
Member_ID      0
Sales_Type     0
Contract_Type  0
Channel        0
Datetime       0
Term          0
Payment_Type   0
Product_Type   0
Amount_Month   0
Customer_Type  0
Age           0
Address1       0
Address2       0
State         0
Overdue_count  0
Overdue_Type   0
Gender        0
Credit_Rank   0
Bank          0
dtype: int64
```

3. Data Preprocessing - Missing Value

▼ 결측값 대체

```
df1['Credit_Rank'].describe()
```

```
count    42520.000000
mean       3.428810
std        2.213453
min         0.000000
25%         1.000000
50%         3.000000
75%         5.000000
max        10.000000
Name: Credit_Rank, dtype: float64
```

```
# 결측치를 평균값으로 대체한 다음 새로운 항목으로 선언
df1['Credit_Rank(clean)'] = df1['Credit_Rank'].fillna(3.429)
```

```
df1['Credit_Rank(clean)'].describe()
```

```
count    51301.000000
mean       3.428842
std        2.015130
min         0.000000
25%         2.000000
50%         3.429000
75%         5.000000
max        10.000000
Name: Credit_Rank(clean), dtype: float64
```

3. Data Preprocessing - Date Time

Date Time

- 데이터의 '시간'의 개념을 다루는 것은 실무에서 매우 중요한 Point
- Time Series와 같은 시계열 분석을 수행하거나, 연도/월/요일 별 통계량을 계산할 때에도 필요
- Pandas 라이브러리에서 Datetime 형태 데이터 타입을 지원
- 날짜 형식에 따라, 날짜데이터를 Datetime Type으로 변환하여 사용

▼ 서로 다른 형태의 날짜 Format을 맞추어 변환

```
df1 = pd.read_csv('03_Delivery.csv', encoding='cp949')
```

```
df1.head(3)
```

	결제 수단	배송번호	배송시작일	배송완료일	상품구매 금액	상품번 호	수 량	주문 경로	주문일
0	무통 장입 금	D-20181227- 0000648-00	2019-01-07 오전 8:56	2019-01-11 오전 5:32	71450	1077.0	1	PC쇼 핑몰	20181227
1	무통 장입 금	D-20181229- 0000119-00	2019-01-03 오전 9:30	2019-01-07 오전 5:32	141240	42.0	2	모바 일웹	20181229
2	무통 장입 금	D-20181230- 0000100-00	2019-01-03 오전 9:30	2019-01-07 오전 5:32	13910	1271.0	1	모바 일웹	20181230

3. Data Preprocessing - Date Time

▼ Format을 이용해 날짜 데이터 변환

```
df1['주문일(datetime)'] = pd.to_datetime(df1['주문일'], format='%Y%m%d')
```

```
df1['주문일(datetime)'].describe(datetime_is_numeric=True)
```

```
count          46749
mean    2018-10-27 17:26:07.477379328
min          2017-01-01 00:00:00
25%          2018-07-27 00:00:00
50%          2018-11-18 00:00:00
75%          2019-02-25 00:00:00
max          2019-06-19 00:00:00
Name: 주문일(datetime), dtype: object
```

```
df1['주문연도'] = df1['주문일(datetime)'].dt.year
df1['주문월'] = df1['주문일(datetime)'].dt.month
df1['주문주차'] = df1['주문일(datetime)'].dt.isocalendar().week
df1['주문요일'] = df1['주문일(datetime)'].dt.day_name()
```

```
df1[['주문일(datetime)', '주문연도', '주문월', '주문주차', '주문요일']].head(3)
```

	주문일(datetime)	주문연도	주문월	주문주차	주문요일
0	2018-12-27	2018	12	52	Thursday
1	2018-12-29	2018	12	52	Saturday
2	2018-12-30	2018	12	52	Sunday

3. Data Preprocessing - Date Time

▼ 요일 데이터를 이용해, 주중/주말 데이터를 나누어 분석

```
list_week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']  
list_weekend = ['Saturday', 'Sunday']
```

```
cond1 = df1['주문요일'].isin(list_week)  
cond2 = df1['주문요일'].isin(list_weekend)
```

```
df1.loc[cond1, '주중/주말'] = '주중'  
df1.loc[cond2, '주중/주말'] = '주말'
```

```
df1['주중/주말'].value_counts()
```

```
주중    42406
```

```
주말     4343
```

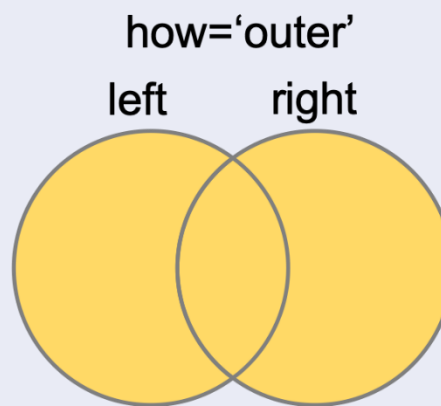
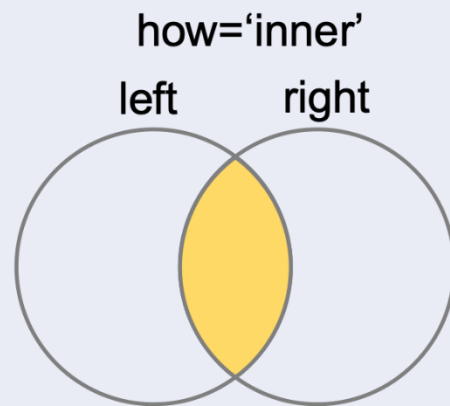
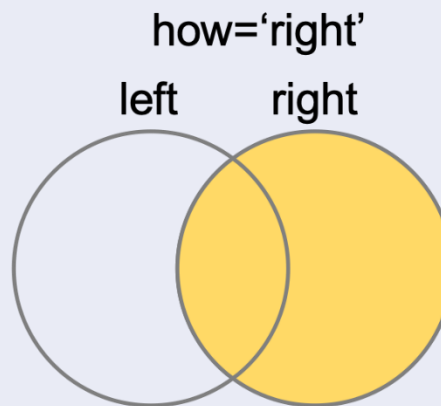
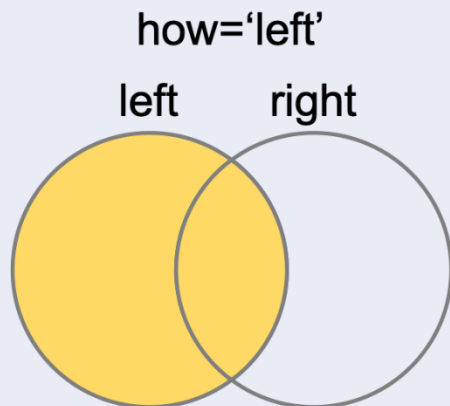
```
Name: 주중/주말, dtype: int64
```

- **isin()** : 특정 리스트 내 데이터가 존재하면 True / 존재하지 않으면 False를 출력
- loc 함수를 isin 함수와 같이 활용해, list_week에 있는 항목은 '주중', list_weekend에 있는 값은 '주말'로 변환

4. DataFrame Join

Table Join

- 서로 다른 데이터 테이블을 공통된 Column을 기준으로 병합하는 작업
- 크게 4가지 병합 기법이 존재



4. DataFrame Join

▼ 아래의 두 Table을 '이름'의 공통 항목(Key Column)으로 병합

이름	나이	성별
홍길동	30	남
성춘향	25	여
이몽룡	45	남

이름	부서	지역
홍길동	A	서울
성춘향	B	서울
변사또	B	제주
허준	C	강원

▼ Inner Join : Key Column의 공통 값만 가져와 병합

이름	나이	성별	부서	지역
홍길동	30	남	A	서울
성춘향	25	여	B	서울

▼ Outer Join : Key Column의 모든 값을 가져와 병합

이름	나이	성별	부서	지역
홍길동	30	남	A	서울
성춘향	25	여	B	서울
이몽룡	45	남		
변사또			B	제주
허준			C	강원

4. DataFrame Join

▼ 아래의 두 Table을 '이름'의 공통 항목(Key Column)으로 병합

```
df1 = pd.read_csv('04_store_member.csv')
print(df1.shape)
df1.head(2)
```

(4396, 6)

	회원번호	회원상태	성별	결혼유무	주소	생년
0	2101	정상회원	여	기혼	서울 강북구 미아동 134-	1967.0
1	2102	정상회원	여	NaN	경기 용인시 수지구 동천동	1947.0

```
df2 = pd.read_csv('04_store_product_1.csv')
print(df2.shape)
df2.head(2)
```

(130893, 11)

	공급일자	물품코드	물품 대분류	물품 중분류	물품 소분류	물품명	구매 수량	구매금액	구매 매장	반품_원 거래일자	회원 번호
0	20170201	50301001	채소	과일 채소	딸기	딸기(1kg)	1.0	13600	매장 1	NaN	2102
1	20170201	80201053	간식	떡	가래 떡	현미가래 떡(400g)	1.0	99999999	매장 1	NaN	2102

4. DataFrame Join

▼ df2의 '회원번호'를 기준으로 '구매금액'과 '구매수량'을 계산

```
df3 = df2.pivot_table(index='회원번호', values=['구매금액', '구매수량'],
                      aggfunc='sum').reset_index()
df3.head(3)
```

	회원번호	구매금액	구매수량
0	2102	102093159	472.5
1	2103	7904400	1586.4
2	2104	389040	112.0

▼ df3의 '회원번호'와 df1의 '회원번호'를 Key Columns 으로 공통된 값에 대해 병합 (Inner Join)

```
pd.merge(df3, df1, on='회원번호', how='inner')
```

	회원번호	구매금액	구매수량	회원상태	성별	결혼유무	주소	생년
0	2102	102093159	472.5	정상회원	여	NaN	경기 용인시 수지구 동천동	1947.0
1	2103	7904400	1586.4	정상회원	여	기혼	경기 용인시 수지구 동천동	1972.0