CITYEYELAB

Business Data Analysis

ICAIS	Step	Define	Outputs
Identify Problem (문제 규명)	1. 과제선정	경영 상의 이슈를 도출하여, 프로젝트로 선정	프로젝트 실행 계획서
	2. 과제의 선정 배경	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. 분석 모델 개념도	과제 선정단계에서 1차적인 분석 모델을 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 수정보완 함(2차 모델)	분석 모델 개념도
	9. 데이터 수집	선정된 데이터의 수집계획을 마련하고, 데이터를 수집함	데이터 수집 계획서, Raw Data
	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

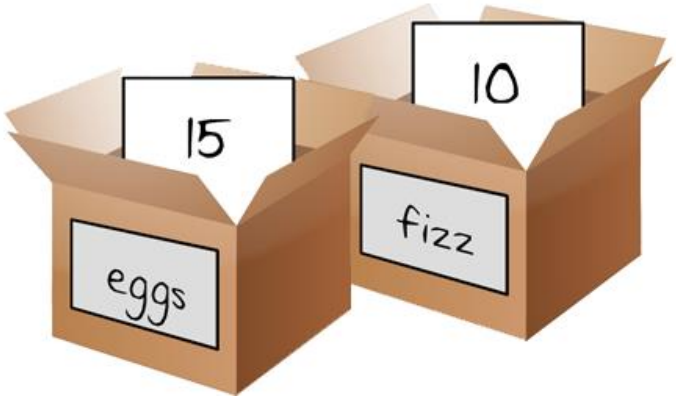
CITYEYELAB

Data Type

변수와 자료형

변수(Variables)란

- 변수(Variable)는 값을 저장하는 박스
- 데이터를 특정 이름을 붙여 저장함
- 변수를 만든다는 것은 숫자나 문자열과 같은 데이터를 저장할 수 있는 공간을 마련하는 것
- 연산을 쉽게 처리하거나, 큰 데이터를 한 번에 불러올 수 있음
- 여러 형태의 데이터를 하나 이상의 변수로 선언 가능



자료형

- 데이터 자료형을 규명하는 것은 프로그래밍에 있어서 가장 기본적이고 중요함
- Python에서는 데이터의 자료형을 변수 선언 시, 자동적으로 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.x 형태로 출력*
    - *문자형(str): 영어나 기호 또는 숫자로 구성된 문자*

CITYEYELAB

Data Type

```
a = 10
print(a)
print(type(a))
```

```
10
<class 'int'>
```

▶ 정수형 10 데이터가 a 에 선언

```
a = "B"
print(a)
print(type(a))
```

```
B
<class 'str'>
```

▶ 문자형 B 데이터가 a 에 선언

```
num = 23.1
print(num)
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, print() 함수를 사용하지 않고, 변수명만 입력해도 데이터가 출력됨
- Python에서 등호(=) 기호는 '같다'를 의미하는 것이 아닌, '선언하다'라는 의미가 있음

CITYEYELAB

Data Type

```
a = 10
b = "B"

a+b
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-5-9fd1f8fd3d20> in <module>()
      2 b = "B"
      3
----> 4 a+b

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

▶ 자료형이 일치하지 않으면 오류가 발생함

```
a = 10
type(a)
```

```
int
```

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

```
float
```

▶ 임의로 데이터의 타입을 바꾸고 싶은 경우

자료형

- 자료형이 일치하지 않으면, 오류가 발생함
- 자료형을 임의로 바꾸려면, 데이터 앞에 타입을 입력하여 변수를 선언함

CITYEYELAB

String

문자열 자료형(string)

- **프로그래밍 뿐만 아니라, 데이터 분석에 있어서 문자를 처리하는 것이 중요함**
- **문자열: 문자, 단어 등으로 구성된 문자들의 집합**

  "단어" / "A" / "Python Programming" / "123"
- **문자열에는 모두 큰따옴표(") 또는 작은따옴표(')를 이용해 선언함**

  → 문자열 내 큰따옴표나 작은따옴표가 쓰일 경우를 대비해, 두 가지 형태로 선언 가능
- **여러 줄의 문자열을 대입할 경우 큰따옴표 또는 작은따옴표 세 개를 사용함**

  """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

String

▶ 문자열을 그대로 작성하면 오류가 발생함

```
안녕하세요
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-8-041a565a82b3> in <module>()
----> 1 안녕하세요

NameError: name '안녕하세요' is not defined
```

```
'안녕하세요'
```

```
'안녕하세요'
```

```
"안녕하세요"
```

```
'안녕하세요'
```

```
'''안녕하세요'''
```

```
'안녕하세요'
```

```
"""안녕하세요"""
```

```
'안녕하세요'
```

▶ 여러 문장을 사용할 경우

```
"""
여러문장의 경우
따옴표를 세 개를 사용해 주면 됩니다.
감사합니다.
"""
```

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

```
a = '안녕하세요'
b = """저는 문자형 입니다.
반갑습니다"""

print(a)
print(b)
type(a),type(b)
```

```
안녕하세요
저는 문자형 입니다.
반갑습니다

(str, str)
```

문자형

- 따옴표를 활용하여 문자형을 문자열 자료형으로 사용할 수 있음
- 여러 개의 문장의 경우, 세 개의 따옴표를 이용함
- 문자열도 변수로 저장할 수 있음

CITYEYELAB

String

▶ 연산자를 이용해 문자열 연산이 가능함

```
str1 = "Python 실무 데이터 분석"
str2 = "에 오신 여러분을 환영합니다."

str1 + str2
```

'Python 실무 데이터 분석에 오신 여러분을 환영합니다.'

```
str3 = '감사합니다'

str3 * 3
```

'감사합니다감사합니다감사합니다'

▶ len() 함수를 이용해 문자열의 길이를 계산함

```
str4 = "이 문자열의 길이는 len함수를 이용해 구할 수 있습니다."

len(str4)
```

32

```
str5 = '감사합니다.'
len(str5)
```

6

문자형

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

CITYEYELAB

String indexing

▶ 인덱싱(indexing)과 연산의 활용

```
str5 = "너의 삶은 짧지만, 계속 구하면 가치있는 것을 찾으리라"

str5[0]
```

'너'

```
str5[14]
```

'구'

```
str5[-2]
```

'리'

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

'너구리'

▶ indexing 결과를 새로운 변수에 선언

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

<class 'str'>

'너구리'

문자형

- Indexing: 특정 위치의 글자를 추출해 올 수 있음
- 대괄호를 이용해 추출하며, 0부터 순서가 매겨진다.
- 뒷부분부터 셀 땐, 음수를 붙여 세어줄 수 있다.
- 실무에서 복잡한 문자 데이터를 다룰 때, 설문/제품명/항목명 등에 주로 사용된다.

CITYEYELAB

String slicing

▶ 슬라이싱(slicing)과 연산의 활용

```
str5 = "너의 삶은 짧지만, 계속 구하면 가치있는 것을 찾으리라"

str5[0:2]
```

'너의'

```
str5[:11]
```

'너의 삶은 짧지만, '

```
str5[18:]
```

'가치있는 것을 찾으리라'

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

'너의 삶은 짧지만, 가치있는 것을 찾으리라'

```
str5[:]
```

'너의 삶은 짧지만, 계속 구하면 가치있는 것을 찾으리라'

▶ 슬라이싱(slicing) 사용

```
str7 = "2021Python실무"
day = str7[:4]

lesson = str7[4:]

print(day)
print(lesson)
```

2021
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 m번째 글자 전까지 추출해 온다.
- str[n:]: n번째 글자부터 맨 끝 글자까지 추출해 온다.
- 처음번호와 끝 번호를 생략하면 모든 문자열이 출력된다.

CITYEYELAB

String Function

문자열 함수

- **Python에서 여러 가지 문자열 함수를 지원함**

- **count():** 특정 문자의 개수를 출력
- **find() / index():** 특정 문자가 어디에 위치해 있는지 출력
- **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 리스트 형태로 나누어 출력
  (쌍반점(:) 기호 사용 시, 전체 문자열이 리스트에 하나의 객체로 출력)

List

리스트(List)

- **리스트(list): 여러 개의 데이터를 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 담겨 있음
- **C 언어의 배열(array)과 유사한 List를 이용**
- **서로 다른 데이터 타입의 값들도 하나의 List에 넣을 수 있음**
- **리스트 내 데이터 간 순서가 존재함**
- **리스트 내 데이터를 삭제하거나 추가, 수정이 가능함**

  append(): 추가

  insert(): 삽입

  remove(): 삭제
- **Size가 정해져 있지 않고 유동성이 존재한다.**

CITYEYELAB

List

▶ 리스트(list) 선언과 형태

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

```
[100, 200, 300, 400, 500]
```

```
type(list1)
```

```
list
```

```
list2 = ['정종우',72,177,'경기 분당']
list2
```

```
['정종우', 72, 177, '경기 분당']
```

```
type(list2)
```

```
list
```

▶ 리스트의 특정 위치에 값을 추출할 때

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

```
100
```

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

```
(200, 300, 400)
```

리스트

- 문자열도 리스트로 선언이 가능하다.
- 리스트에서 특정 값을 추출(indexing)
  - 1) 리스트 이름을 입력 후, 대괄호를 이용해 위치를 호출
  - 2) 순서는 0부터 시작함

CITYEYELAB

List

▶ 리스트(list)의 여러가지 함수

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

```
['국어', '영어', '수학', '과학']
```

```
sub.append('사회')
sub
```

```
['국어', '영어', '수학', '과학', '사회']
```

```
sub.insert(0,'한국사')
sub
```

```
['한국사', '국어', '영어', '수학', '과학', '사회']
```

```
sub.remove('수학')
sub
```

```
['한국사', '국어', '영어', '과학', '사회']
```

```
len(sub)
```

```
5
```

▶ 리스트의 연산

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

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

```
score * 3
```

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

```
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.

CITYEYELAB

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 불가능함**
- **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 선언하며, 변수에 각 데이터를 선언
- **함수와 반복문 같이 중요한 하이퍼파라미터(Hyper Parameter)들을 보호할 때 사용함**

  하이퍼파라미터: 수식 내 값이 변하지 않는 인자나 상수

Tuple

▶ 튜플(tuple) 선언과 형태

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

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

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

```
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'
```

```
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

```
t2 = (1,2,'삼',4,5)
t2
```

```
(1, 2, '삼', 4, 5)
```

```
t2[2]
```

```
'삼'
```

```
t2[-1]
```

```
5
```

튜플

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

Tuple

▶ packing과 unpacking

** Packing **

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

```
tuple
```

```
pack1
```

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

** Unpacking **

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

```
int
```

```
a
```

```
1
```

▶ 튜플 함수 및 연산

** 튜플 연산 **

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

```
10
```

```
a[-1]
```

```
50
```

```
a[2:]
```

```
(30, 40, 50)
```

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

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

```
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.
- 기본 연산자를 이용한 연산이 가능하다.

CITYEYELAB

Set

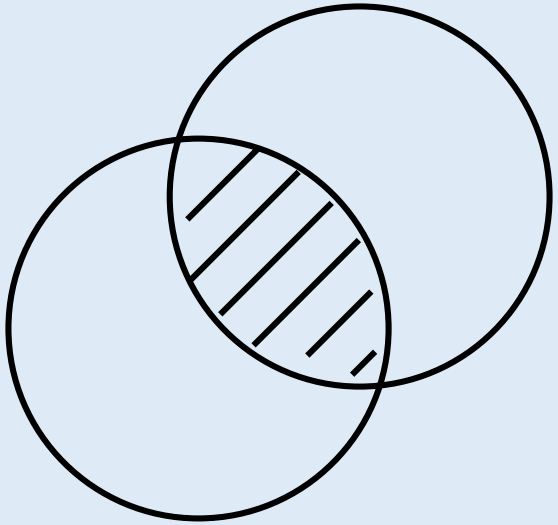
세트(Set)

- **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 순서가 없으며 중복을 허락하지 않음**
- **집합 내 데이터 변경이 가능함**
- **집합 연산이 가능함(Van Diagram 개념)**

집합 연산	
A&B	A와 B의 교집합 연산
A|B	A와 B의 합집합 연산
A-B	A 집합에서 B 집합의 원소를 제외한 나머지
A^B	A 집합과 B 집합의 교집합을 제외한 나머지



CITYEYELAB

Set

▶ 세트(set) 선언과 형태

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

executed in 16ms, finished 22:22:21 2021-10-01

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

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

```
{10}
```

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

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

▶ 세트의 집합 연산

** Set의 집합연산 **

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

# 교집합 연산
print(C&D)
# 합집합 연산
print(C|D)
# 차집합 연산 ; C 집합에서 D 집합의 원소를 뺀 나머지
print(C-D)
# 대상 차집합 연산 ; 교집합을 제외한 나머지 원소 출력
print(C^D)
```

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

세트

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

CITYEYELAB

Set

▶ 세트(set)에서 사용하는 여러 함수

```
print(C.intersection(D))
print(C.union(D))
print(C.difference(D))
```

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

```
C.add(100)
C
```

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

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

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

```
C.remove(30)
C
```

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

세트

- 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

CITYEYELAB

Dictionary

딕셔너리(Dictionary)

- **딕셔너리(Dictionary): 데이터를 key와 value의 pair 형태로 하나의 변수에 선언**
- **중괄호를 이용하여, key-value pair 형태로 묶어줌**

```
d = {"name":'"Jung", "value":100}
# Jung이라는 데이터가 name이라는 키 값과 쌍을 이룸
# 100이라는 데이터가 value라는 키 값과 쌍을 이룸
```

- **매우 많이 사용되는 데이터 자료형 중 하나이다.**
- **데이터를 구조적으로 다룰 수 있음**
- **Key 값은 중복되지 않음**
- **Pandas 라이브러리의 Series와 비슷한 개념**

Dictionary

▶ 딕셔너리(dictionary) 선언과 형태

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

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

▶ 딕셔너리 Indexing

** Indexig **

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

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

```
dict1['이름']
```

```
'안기모'
```

```
dict1.get('소속')
```

```
'빅데이터 그룹'
```

```
dict1.keys()
```

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

```
dict1.values()
```

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

딕셔너리

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

CITYEYELAB

Dictionary

▶ 딕셔너리(dictionary)와 리스트(list)의 사용

```
dict1['이름'] = ['나천재','안기모']
dict1['소속'] = ['빅데이터 그룹','경영혁신 그룹']
dict1['월 수익'] = [1500,2000]

dict1
```

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

```
name = ['안기모']
group = ['빅데이터 그룹']
income = [2000]

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

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

```
name.append('나천재')
group.append('경영혁신 그룹')
income.append(1500)
```

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

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

딕셔너리

- 딕셔너리의 Value 값도 List 형태로 넣을 수 있다.
  → 여러 형태의 데이터를 Key – Value 형태로 선언하고 관리할 수 있다.
- 딕셔너리와 리스트를 이용하여, 해당 Key 값에 Value를 계속 추가해 줄 수 있다.

CITYEYELAB

Operator

기본 연산자(Operator)

- Python에서는 기본적인 수학 연산을 지원한다.

▪ 산술 연산자

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

▪ 비교 연산자

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

▪ 논리 연산자

- And / or / not: ~이고, 그리고 ~ / ~이거나, 또는 / ~이 아니다.

산술 연산자		비교 연산자	
+ / -	덧셈/뺄셈	A > B	A가 B보다 크다
*	곱셈	A >= B	A가 B보다 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**

CITYEYELAB

Operator

▶ 산술 연산자 연습

```
a = 100
b = 35
```

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

```
(135, 65)
```

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

```
(3500, 2.857142857142857)
```

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

```
(2, 30)
```

```
a//b
```

```
2
```

▶ 비교 연산자 연습

```
a = 50
b = 67

a>b ,a<b
```

```
(False, True)
```

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

```
(False, True)
```

```
a==b
```

```
False
```

```
a!=b
```

```
True
```

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

```
bool
```

기본 연산

- 쉼표(,)를 이용해 여러 개의 결과를 동시에 출력할 수 있다.
- 결과에 대한 자료형을 type() 기능을 이용해 파악할 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)

CITYEYELAB

Operator

▶ 논리 연산자 연습

```
a = 100
b = 20

(a>b) and (a<b)
```

```
False
```

```
(a!=b) and (a>b)
```

```
True
```

```
(a!=b) or (a<b)
```

```
True
```

기본 연산

- 쉼표(,)를 이용해 여러 개의 결과를 동시에 출력할 수 있다.
- 결과에 대한 자료형을 type() 기능을 이용해 파악할 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)

CITYEYELAB

Conditional Statement

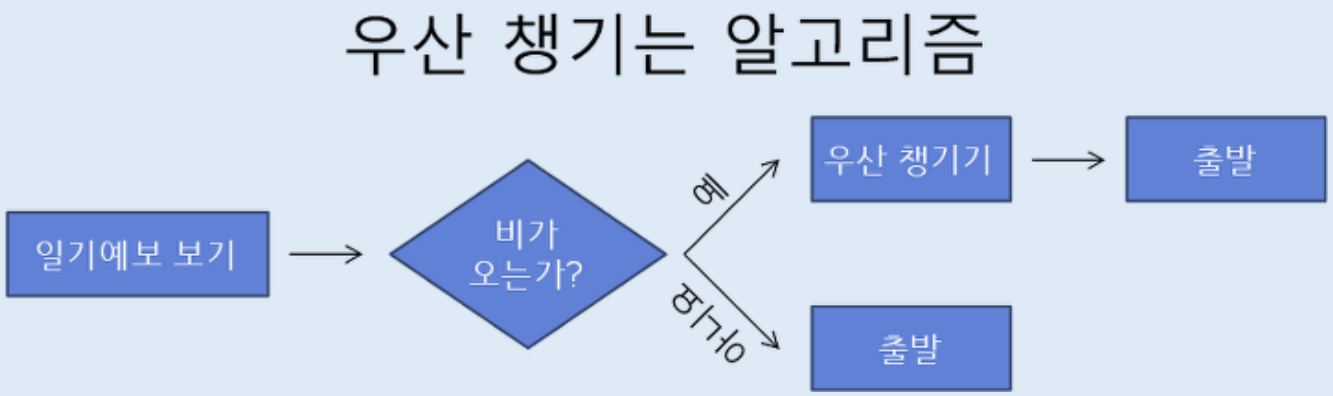
조건문(if)

- 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

If문의 기본 구조

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

- "조건1"을 만족하면, "수행할 문장1"이 실행됨
- "조건1"을 만족하지 않으면, "수행할 문장2"가 실행됨



CITYEYELAB

If Statement

▶ if 조건문을 이용한 연속형 변수 비교

```
A = 100
B = 200

if A > B :
    print("Hi")
else:
    print("안녕하세요")
```

```
안녕하세요
```

* 들여쓰기를 이용해 문장의 종속관계를 표현함

▶ 입력 함수를 이용하여 숫자를 비교하는 프로그램

```
C = int(input("첫번 째 정수를 입력하세요 : "))
D = int(input("두번 째 정수를 입력하세요 : "))

if C>D:
    print("첫번째 정수 ",C,"가 더 큽니다.")
else:
    print("두번째 정수 ",D,"가 더 큽니다.")

if C==D:
    print("첫번째 정수와 두번째 정수가 같습니다.")
else:
    print("첫번째 정수와 두번째 정수가 다릅니다.")
```

```
첫번 째 정수를 입력하세요 : 10
두번 째 정수를 입력하세요 : 5
첫번째 정수  10 가 더 큽니다.
첫번째 정수와 두번째 정수가 다릅니다.
```

If 조건문

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

CITYEYELAB

If Statement

▶ if 조건문을 이용한 연속형 변수 비교

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

- "조건1"을 만족하면, "수행할 문장1"이 실행됨
- "조건2"을 만족하면, "수행할 문장2"가 실행됨

▶ if 문을 이용한 성적처리 프로그램

```
score = int(input("성적을 입력하세요! : "))

if score >= 95 :
    print("학생의 성적은 A+ 입니다.")
elif score >= 90:
    print("학생의 성적은 A 입니다.")
elif score >= 80:
    print("학생의 성적은 B 입니다")
elif score >= 70:
    print("학생의 성적은 C 입니다")
else:
    print("학생의 성적은 F 입니다")
```

```
성적을 입력하세요! : 15
학생의 성적은 F 입니다
```

▶ pass 구문의 이용

```
A, B = 100, 20

if A > B :
    pass
else:
    print("Error")
```

If 조건문

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

CITYEYELAB

Loop Statement

반복문(Loop Statement)

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

while문의 기본 구조

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

- "조건1"을 만족하면, "수행할 문장1"이 실행됨
- "조건1"을 만족하면, "수행할 문장2"가 실행됨

for문의 기본 구조

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

- 0에서 부터, i번째 직전까지 n값이 수행 문장 변수에 대입되어 반복 수행됨

CITYEYELAB

While Statement

▶ while 반복문

```
i = 0

while i <5 :
    print("Hi Python~!")
    i = i + 1

print("반복종료")
```

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

▶ break문을 이용해 while문을 중단

```
i = 1
while i > 0:
    print("Hello")
    i = i + 1

    if i > 3 :
        print("강제 탈출 ")
        break

print("반복종료 ")
```

```
Hello
Hello
Hello
강제 탈출
반복종료
```

While 반복문

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

CITYEYELAB

For Statement

▶ for 반복문

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

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

▶ 반복의 범위를 변수를 이용해 잡아 준 경우

```
n = 10
for i in range(0,n+1):
    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, 자동적으로 수행문이 종료된다.

CITYEYELAB

For Statement

▶ 여러 가지 자료형을 이용한 for 문

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

```
a
b
c
d
```

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

```
3
7
11
```

▶ for 문과 if문을 같이 사용한 경우

```
score = [100,23,30,76,34,50]
num = 0

for i in score:
    num = num + 1
    if i > 60 :
        print("합격입니다.")
    else:
        print("불합격입니다.")
```

```
합격입니다.
불합격입니다.
불합격입니다.
합격입니다.
불합격입니다.
불합격입니다.
```

▶ 1부터 n까지 더하는 프로그램 vs 가우스 수열 이용

** 1부터 100까지 더하는 프로그램 **

```
sum = 0

for i in range(1,101):
    sum = sum + i

print(sum)
```

```
5050
```

** 100000000 회 반복시 **

```
sum = 0

for i in range(1,100000001):
    sum = sum + i

print(sum)
```

```
5000000050000000
```

** 가우스 **

```
n = 100

sum = n*(n+1)/2

print(sum)
```

```
5050.0
```

```
n = 100000000

sum = n*(n+1)/2

print(sum)
```

```
5000000050000000.0
```

for 반복문

- 리스트나 튜플 같은 자료형을 이용해 for 반복문을 실행 시킬 수 있다.
- If 문과 같이 쓰여, 자동업무 구현에 자주 활용된다.
- 계속 같은 수행문이 반복되기 때문에 때에 따라서는 처리속도가 매우 느려질 수 있다.
- 최대한 for 문을 피하여, 간단하게 코딩하는 것이 좋다.

CITYEYELAB

Function

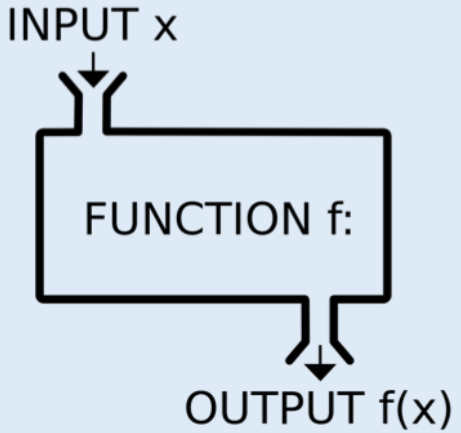
함수(Function)

- 함수: 입력 X값에 대한 결과 Y값을 반환(Return)하는 기능을 하나의 묶음으로 선언
- Input X와 Output Y로 구성됨
- 반복적으로 사용되는 특별한 기능을 함수로 선언함
- 여러 함수들의 집합을 '모듈(module)'이라고 부름
- 복잡한 데이터를 전처리할 때, 특정 기능을 반복적으로 구현하고 싶을 때 사용함

함수(Function)의 기본 구조

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

- 코드에서 Function1의 함수를 input값 X를 넣어 실행하면, 함수 내 수행문이 실행된 후 결과 값 Y가 반환된다.



CITYEYELAB

Function

▶ 입력 받은 수의 합을 구하는 함수

```
def sum(num):
    for i in range(1,num+1):
        sum = num*(num+1)/2
    return print(sum)

num = int(input("정수를 입력하시오 : "))
sum(num)
```

```
정수를 입력하시오 : 132
8778.0
```

1

2

코드가 처음으로 시작되는 부분

함수의 이름이 나올 때, 비로소 함수가 적용됨

▶ 한번 선언된 함수는 계속 사용이 가능함

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

```
data = 1000
circle_area(data)
```

```
3140000.0
```

- 한번 선언된 함수는 다른 코드 입력 창에서 입력을 해도 사용이 가능함
- 입력 받는 변수가 바뀌어도 함수 입력 값은 동일하게 들어감

함수

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

CITYEYELAB

Function

▶ 두 개의 입력 값을 받아 하나의 결과 값을 출력

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

```
num1 = 1000
num2 = 2000

cylinder_volume(num1,num2)
```

```
6280000000.0
```

▶ 날짜 데이터를 함수를 이용해 처리

```
def datetime(date):
    date = str(date)
    result1 = date[0:4]+'-'+date[4:6]+'-'+date[6:]
    return result1

datetime(20190724)
```

```
'2019-07-24'
```

▶ 하나의 값을 입력해 여러 개의 결과 값을 출력

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

timetable(5)
```

```
(5, 10, 15)
```

▶ 입력 값과 결과 값이 없는 경우

```
def function1( ):
    print("hi")

function1()
```

```
hi
```

함수

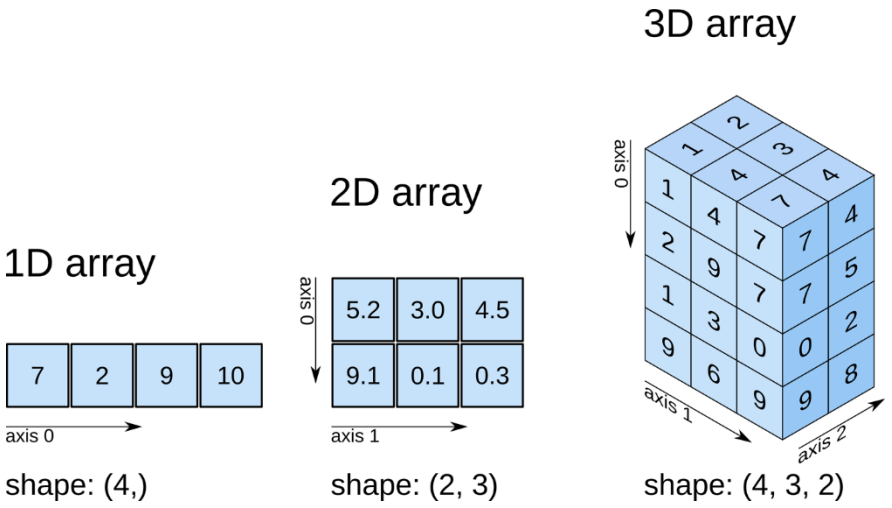
- 입력 값에는 여러 가지 데이터를 넣을 수 있다.
- 출력되는 데이터의 형태도 여러 형태로 출력할 수 있다.
- 입력 값 도는 결과 값이 없는 형태로도 출력할 수 있다.
- 데이터 전처리에서 매우 많이 사용된다.

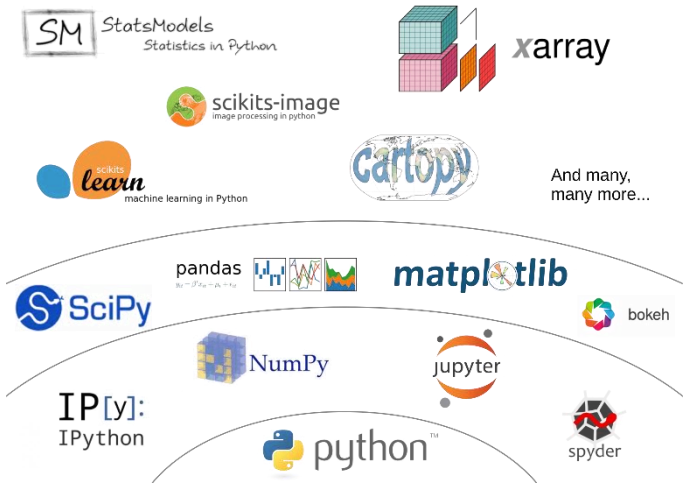
CITYEYELAB

Library

라이브러리(Library)

- 모듈: 여러 함수들을 모아놓은 집합
- 라이브러리: 특정 기능을 지원하는 모듈 및 함수를 모아놓은 집합
- 기능에 따라 여러 가지 라이브러리가 존재 (시각화, 웹, 데이터 분석, 머신러닝 등)
- 필요에 따라 라이브러리를 직접 만들기도 함
- Anaconda를 설치하면, 데이터분석에 필요한 기본 라이브러리들이 포함되어 설치됨
- 데이터 분석에서 기초적으로 Numpy와 Pandas를 많이 사용함





CITYEYELAB

Numpy

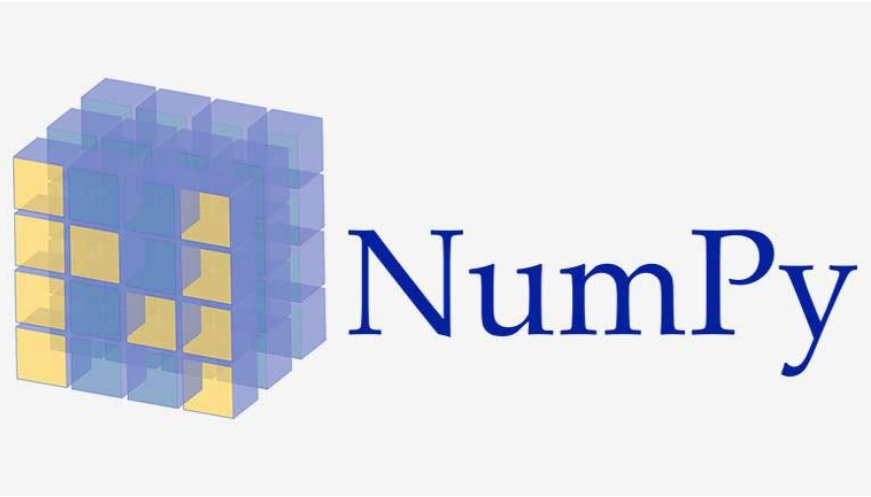
- **Numpy: Numeric + Python의 약자, 수학 및 과학 연산 라이브러리**
- **배열이나 행렬 계산에 필요한 함수 제공**
- **수열 데이터를 다룰 때 용이, 이후에 Pandas에서 DataFrame 형태로 사용함**
- **다차원 배열(Array)을 다룰 때 주로 사용함 (인공 신경망, 비정형 데이터 처리, 자연어 처리 등)**

Numpy의 라이브러리 호출

import numpy as np

- Numpy 라이브러리를 호출하여, np라는 약자로 사용한다.
- Numpy 내에 함수를 사용할 때, 반드시 np 라는 문자를 붙인다.

공식 홈페이지 / 도큐먼트 https://numpy.org/



CITYEYELAB

Numpy

▶ Numpy 라이브러리 호출

```
import numpy as np
```

```
data  = [6,5.7,3,4,2]   # 일반적인 python data list 형태
data.mean()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-49-c45326716449> in <module>()
      1 data  = [6,5.7,3,4,2]   # 일반적인 python data list 형태
----> 2 data.mean()

AttributeError: 'list' object has no attribute 'mean'
```

```
arr = np.array(data)
arr.mean()
```

```
4.14
```

▶ 리스트와 배열의 데이터 타입

```
print(type(data))
print(type(arr))
```

```
<class 'list'>
<class 'numpy.ndarray'>
```

Numpy

- ndarray: Numpy의 데이터 타입 중 하나로, 리스트와 비슷한 형식의 구조적인 데이터
- 이중 구조의 배열이나 행렬(Matrix)처럼 사용이 가능하며, 수학/과학 연산을 모두 지원한다.

CITYEYELAB

Numpy

▶ Numpy array 내부 데이터 타입

** numpy array 의 데이터 타입 **

```
arr1 = np.array([1,2,3,4,5])
arr1.dtype
```

```
dtype('int32')
```

```
arr2 = np.array([1.1, 2.3, 3.4, 4.5, 5.3])
arr2.dtype
```

```
dtype('float64')
```

```
arr3 = np.array(['A','B','C','D','E'])
arr4 = np.array(['김재윤','이철호','석도형','이한진','김수봄'])
print(arr3)
print(arr4)
print(arr3.dtype)
print(arr4.dtype)
```

```
['A' 'B' 'C' 'D' 'E']
['김재윤' '이철호' '석도형' '이한진' '김수봄']
<U1
<U3
```

▶ 3X3 행렬의 구현

** 이중 Matrix **

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

matrix = np.array(data)
print(type(matrix))
matrix
```

```
<class 'numpy.ndarray'>

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

Numpy

- ndarray: 배열 내부에는 여러 행태의 데이터 타입이 올 수 있으며,
  변수 선언 시 자동으로 데이터 타입이 결정 된다.
- 이중 리스트를 이용해 행렬(matrix)을 구현 할 수 있다.

CITYEYELAB

Numpy

▶ 행렬간 사칙연산

** Matrix의 사칙연산 **

```
matrix = np.array([[1,2,],[2,3],[19,30]])
matrix
```

```
array([[ 1,  2],
       [ 2,  3],
       [19, 30]])
```

```
matrix_2 = np.array([[3,4],[5,6],[3,5]])
matrix_2
```

```
array([[3, 4],
       [5, 6],
       [3, 5]])
```

```
print(matrix + matrix_2)
print('================')
print(matrix - matrix_2)
```

```
[[ 4  6]
 [ 7  9]
 [22 35]]
================
[[-2 -2]
 [-3 -3]
 [16 25]]
```

▶ 행렬의 제곱 연산

```
matrix **2
```

```
array([[  1,   4],
       [  4,   9],
       [361, 900]], dtype=int32)
```

▶ 행렬의 형태를 나타내는 함수

** 배열의 형태 **

```
arr_b = np.array([[1,2,],[2,3]])
arr_b.shape
```

```
(2, 2)
```

2x2 행렬을 나타낸다.

Numpy

- 행렬 간 사칙연산도 가능하다.
- shape: 행렬의 구조를 출력해준다. N개의 행의 m개의 열로 구성된 행렬은 (n,m)으로 출력된다.

CITYEYELAB

Numpy

▶ arange 함수를 이용한 데이터 자동 생성

** arange 함수 **

```
arr5 = np.arange(10)
arr5
```

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

```
arr6 = np.arange(1,6)
arr6
```

```
array([1, 2, 3, 4, 5])
```

```
arr7 = np.arange(1,10,2)
arr7
```

```
array([1, 3, 5, 7, 9])
```

▶ random data 생성

** random 함수 **

```
arr8 = np.random.randn(10)
arr8
```

```
array([ 0.14976628, -0.02257515,  0.04339561, -1.41898028, -0.58434544,
       -1.21515686, -0.17158701, -0.00979046, -0.31444517, -1.87297737])
```

```
arr9 = np.random.randn(10,5)
arr9
```

```
array([[ 0.40626083,  2.42619763, -0.31567582, -0.26100318,  0.81180358],
       [ 0.20287436, -0.19651704,  1.28342093,  0.78822009,  0.08333664],
       [ 0.199186  ,  1.25866321,  2.68502678,  1.32202813,  0.04838755],
       [ 1.73507176, -0.97925007, -0.74601245,  1.59260179,  0.57900297],
       [-0.06966016, -0.45476161,  0.83927609, -0.15315803, -0.3517723 ],
       [-1.71420445,  1.2544422 ,  0.06903204, -0.14948529, -1.15113748],
       [-1.89532021, -1.73015901,  0.94799112,  1.35773413,  1.56900441],
       [-1.4292201 ,  0.18517878, -0.18391198,  0.43691554, -0.93377357],
       [-0.00681251,  0.29786217, -0.59442625,  0.55825449, -0.38147473],
       [-0.75924897, -0.55560701,  1.67937484, -0.28819802, -1.13880699]])
```

Numpy

- np.arange(n): 0부터 n까지의 값을 가지는 배열을 생성한다.
- np.arange(n,m): n부터 m-1까지의 값을 갖는 배열을 생성한다.
- np.arange(n,m,k): n부터 m-1까지 k씩 증가시켜 배열을 생성한다.
- np.random.randn(n): 평균이 0이고 표준편차가 1인 n개의 데이터를 무작위로 생성한다.
- np.random.randn(n): 평균이 0이고 표준편차가 1인 n개의 데이터를 nxm 행렬의 형태로 생성한다.

CITYEYELAB

Numpy

▶ arange 함수를 이용한 데이터 자동 생성

```
arr = np.array([1,2,3,4,5])
print(arr.dtype)
```

```
int32
```

```
arr = arr_int.astype(np.float)
print(arr.dtype)
arr
```

```
float64

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

리스트와 같은 방식으로 indexing(데이터 탐색)을 할 수 있다.

▶ array indexing

```
arr_1 = np.arange(1,11)
arr_1
```

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

```
arr_1[0]
```

```
1
```

```
arr_1[-1]
```

```
10
```

```
arr_1 [:5]
```

```
array([1, 2, 3, 4, 5])
```

```
arr_1 [5:]
```

```
array([ 6,  7,  8,  9, 10])
```

```
arr_1 [:]
```

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

Numpy

- np.astype(): 현재 array 내 데이터를 원하는 형태의 데이터 타입으로 변경
- indexing(색인)은 리스트와 동일하게 작용한다.
  - * arr[n]: n번째 요소 출력
  - * arr[:n]: 처음부터 n전 까지 데이터 요소 출력
  - * arr[:]: 모든 데이터 출력
  - * arr[-m]: 뒤에서 m번째 요소 출력
  - * arr[m:]: 뒤에서 부터 m까지 데이터 요소 출력

CITYEYELAB

Numpy

▶ 2차원 array에서 indexing

```
** 2D - Indexing **
```

```
arr_1 = np.array([[1,2,5],[4,5,9],[6,7,34]])
arr_1
```

```
array([[ 1,  2,  5],
       [ 4,  5,  9],
       [ 6,  7, 34]])
```

```
arr_1[0,0]
```

```
1
```

```
arr_1[2,1]
```

```
7
```

```
arr_1[0,:]
```

```
array([1, 2, 5])
```

```
arr_1[:,0]
```

```
array([1, 4, 6])
```

▶ 행 또는 열 형태로 출력

```
arr_1[0:2,:]
```

```
array([[1, 2, 5],
       [4, 5, 9]])
```

```
arr_1[:,0:2]
```

```
array([[1, 2],
       [4, 5],
       [6, 7]])
```

```
arr_1[:,:]
```

```
array([[ 1,  2,  5],
       [ 4,  5,  9],
       [ 6,  7, 34]])
```

Numpy

- ndarray는 2차원 형태도 indexing이 가능하다.
- 쉼표(,)를 기준으로 앞에는 행, 뒤에는 열에 대한 정보를 얻을 수 있다.
  * array[1,2]: 1행 2열의 데이터
  * array[:,1]: 행 전체와 1열의 데이터

CITYEYELAB

Numpy

▶ array 내부의 데이터 변환하기

```
** 값 바꾸기 **

arr1 = np.array([3,4,5,6,7])
arr1

array([3, 4, 5, 6, 7])

arr1[0] =100
arr1

array([100,   4,   5,   6,   7])
```

▶ 열 데이터 변환하기

```
arr2[:,:]=1988
arr2

array([[1988, 1988],
       [1988, 1988],
       [1988, 1988],
       [1988, 1988],
       [1988, 1988]])
```

▶ 전체 데이터 변환하기

```
arr2 = np.arange(1,11).reshape(5,2)
arr2

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

arr2[0,:] = 200
arr2

array([[200, 200],
       [  3,   4],
       [  5,   6],
       [  7,   8],
       [  9,  10]])
```

▶ 값 초기화 하기

```
** 초기값 세팅 **

arr_z = np.zeros((2,2))
arr_z

array([[0., 0.],
       [0., 0.]])

arr_x = np.zeros(10)
arr_x

array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

arr_c = np.ones(10)
arr_c

array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

▶ 배열 합 구하기(전체 합, 행 방향 합, 열 방향 합)

```
** Array Operation **

arr1 = np.arange(1,11).reshape(5,2)
arr1

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

arr1.sum()

55

arr1.sum(axis=1)

array([ 3,  7, 11, 15, 19])

arr1.sum(axis=0)

array([25, 30])
```

CITYEYELAB

Pandas

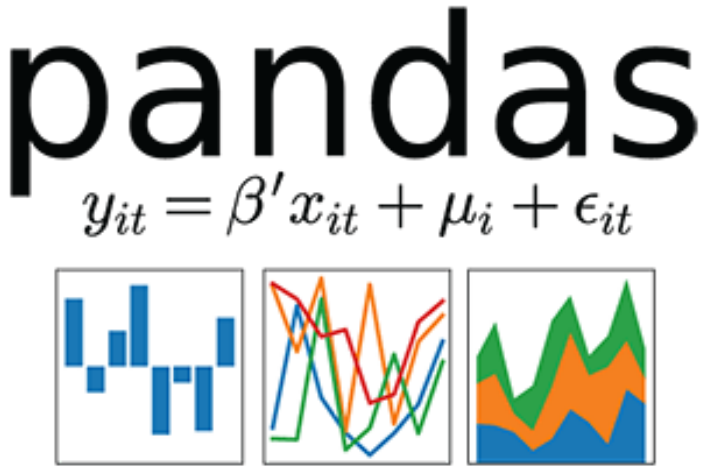
- Pandas: 데이터 과학에 사용되는 라이브러리로 데이터 불러오기, 전처리, 통계 분석에 사용됨
- Numpy를 기반으로 작성된 라이브러리
- Series 형태와 DataFrame 형태가 존재함 (DataFrame 형태는 CH.3에서 자세하게 다룰 예정)
- 실무에서 접하는 엑셀 파일이 DataFrame 형태임
- 데이터 분석에 있어서 필수적인 라이브러리이며, 전처리의 대부분은 pandas 라이브러리를 활용함

Pandas 라이브러리 호출

```
import pandas as pd
import numpy as np
```

- Pandas 라이브러리를 호출하여, pd라는 약자로 사용한다.
- 주로 numpy 라이브러리와 함께 쓰인다.
- 데이터 분석 시 두 가지 라이브러리는 필수적으로 호출해야한다.

공식 홈페이지 / 도큐먼트 https://pandas.pydata.org/



CITYEYELAB

Series

▶ Series의 기본적인 구조

```
import numpy as np
import pandas as pd
```

```
s1 = pd.Series([1,2,56,77,93])
s1
```

```
0     1
1     2
2    56
3    77
4    93
dtype: int64
```

```
s1.sum()
```

```
229
```

```
s1.mean()
```

```
45.8
```

▶ 리스트와 딕셔너리 형태로 Series를 생성함

```
s2 = pd.Series(['홍길동','성춘향','이몽룡','이길동'], name='출석부',
               index = ['하나','둘','셋','넷'])
s2
```

```
하나    홍길동
둘     성춘향
셋     이몽룡
넷     이길동
Name: 출석부, dtype: object
```

```
myDict = {'name':'Sam', 'age':20, 'gender':'male', 'job':'student'}
person = pd.Series(myDict, name='Person')
person
```

```
name          Sam
age            20
gender       male
job       student
Name: Person, dtype: object
```

Series

- Series는 값(value)과 순서(index)로 구성되어 있다.
- Series는 통계적 연산이 가능하다.
- Series는 여러 가지 Python 데이터 타입으로 생성이 가능하다.
- Series 자체에 name 기능을 이용해, 이름을 부여할 수 있다.

CITYEYELAB

Series

▶ Series의 indexing

** Indexing **

```
myDict = {'name':'Sam', 'age':20, 'gender':'male', 'job':'student'}
person = pd.Series(myDict, name='Person')
person
```

```
name          Sam
age            20
gender       male
job       student
Name: Person, dtype: object
```

```
person[0]
```

```
'Sam'
```

```
person['job']
```

```
'student'
```

```
person[:]
```

```
name          Sam
age            20
gender       male
job       student
Name: Person, dtype: object
```

▶ 비교 연산자를 이용한 indexing

```
person == 'male'
```

```
name      False
age       False
gender     True
job       False
Name: Person, dtype: bool
```

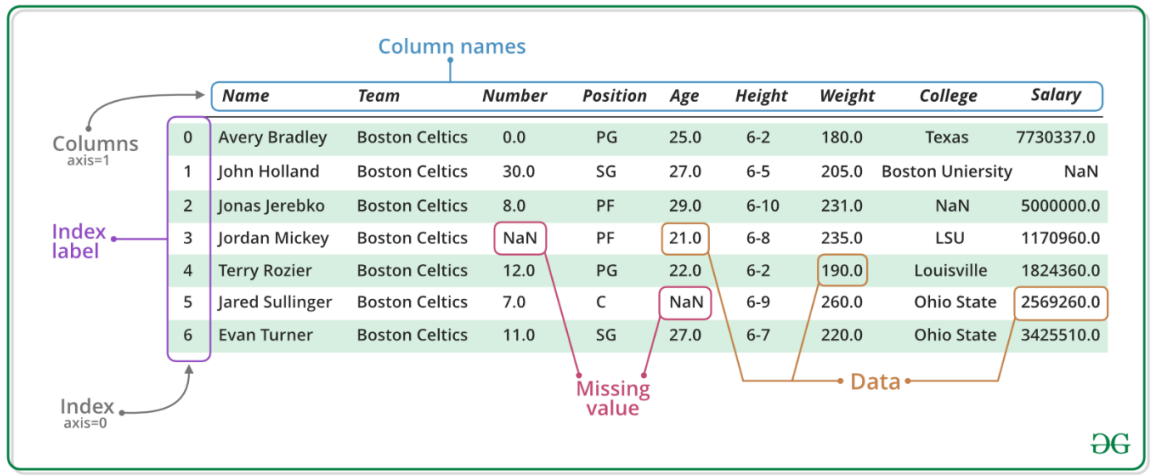
```
person[person == 'male']
```

```
gender    male
Name: Person, dtype: object
```

Series

- Series도 [] 기호를 이용해 indexing이 가능하다.
  * index 기준으로 값(value)를 찾아오는 형태
- 비교 연산자를 이용해, 특정 데이터의 존재 여부를 알 수 있다.

DataFrame Handling



	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston Uniersity	NaN
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0	6-8	235.0	LSU	1170960.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN	6-9	260.0	Ohio State	2569260.0
6	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0

✓ **Chapter 3. DataFrame Handling**

1) DataFrame Introduction
2) Table 기본 정보
3) DataFrame 기본 연산
4) Data 불러오기
5) Data 전처리 실습

Pandas DataFrame

데이터프레임(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 DataFrame 1GB~100GB**
  - **→ 복잡한 기능을 구현하기 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 이용할 수 있다.**
- **가장 구조적인 데이터 형태로써 직관적이고 설명하기 쉬우며, csv나 excel 형태로 저장할 수 있다.**

CITYEYELAB

DataFrame Introduction

▶ 리스트를 이용한 DataFrame 생성

```
import pandas as pd
```

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

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

```
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.

DataFrame Introduction

▶ 이중 리스트를 이용한 데이터프레임 생성

```
table1 = [
    ['2019-01-01',1000,'False','gum'],
    ['2019-01-04',3000,'True','snack'],
    ['2019-01-06',2000,'True','beverage'],
    ['2019-01-06',1000,'True','gum']
]
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

▶ columns 함수를 통해 컬럼명 변경

```
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

```
table2 = {
    '일자':['2019-01-01','2019-01-04','2019-01-06','2019-01-06'],
    '가격':[1000,3000,2000,1000],
    '구매여부':['False','True','True','True'],
    '제품':['gum','snack','beverage','gum']
}
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

```
name = ["A","B","C","D"]

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

Table Information

데이터프레임(DataFrame)

- DataFrame의 3가지 구성 요소: 인덱스(index), 변수(column), 값(value)

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

column

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

value

- 데이터 분석 전 데이터를 파악할 때, 이 세가지 요소에 대해 가장 먼저 확인해 봐야 한다.
- Index와 column은 pandas core 형태로 구성, value는 numpy의 ndarray 형태로 구성되어 있다.

```
type(df1.columns)
```

```
pandas.core.indexes.base.Index
```

```
type(df1.index)
```

```
pandas.core.indexes.base.Index
```

```
type(df1.values)
```

```
numpy.ndarray
```

```
df1.columns
```

```
Index(['일자', '가격', '구매여부', '제품'], dtype='object')
```

```
df1.index
```

```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

```
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)
```

Table Information

▶ 데이터 상위 5개 값만 출력 / 하위 5개 값만 출력

** 데이터 탐색 **

```
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

```
df1.tail()
```

	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

```
df1.head(2)
```

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

```
df1.tail(2)
```

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

▶ 특정 column에 대해서 확인

```
df1['제품']
```

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

```
df1['가격'].head()
```

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

▲ 특정 변수의 값만 head(), tail()함수로 출력

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

DataFrame

- dataframe.head(): 데이터프레임의 상위 5개 값을 출력한다.
- dataframe.tail(): 데이터프레임의 하위 5개 값을 출력한다.

Sorting

▶ 데이터를 index 기준 순서로 정렬하고 싶을 때

```
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 column을 기준으로 정렬할 때

```
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.

```
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

```
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

Operation

▶ 연속형 데이터에 대한 기본 통계 연산

```
df1['가격'].mean()
```
1750.0

```
df1['가격'].sum()
```
7000

```
df1['가격'].min()
```
1000

```
df1['가격'].max()
```
3000

```
df1['가격'].std()
```
957.427107756338

▶ 범주형 데이터에 대한 연산

```
df1['제품'].unique()
```
```
array(['gum', 'snack', 'beverage'], dtype=object)
```

```
df1['제품'].value_counts()
```
```
gum         2
beverage    1
snack       1
Name: 제품, dtype: int64
```

▶ describe() 함수를 이용한 요약 정리

```
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 요약 통계량을 출력한다.
- dataframe['col'].unique(): dataframe에 명목형 column의 항목을 출력한다.
- dataframe['col'].value_counts(): dataframe의 명목형 column의 항목 개수를 출력한다.

CITYEYELAB

Indexing

▶ 특정 변수에 대해 데이터 값을 확인

```
df1['제품']
```

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

```
df1[['제품','가격']]
```

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

* 여러 column을 하나의 프레임형태로 출력

▶ 범주형 데이터에 대한 연산

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

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

▶ 대괄호의 개수에 따라 Dataframe/Series 형태가 결정

```
type(df1['가격'])
```

```
pandas.core.series.Series
```

```
type(df1[['가격']])
```

```
pandas.core.frame.DataFrame
```

Indexing

- 실무에서는 불필요한 column이 많아 원하는 column만 추출해 올 필요가 있다.
- Dataframe['column1']: dataframe의 column1 데이터만 가져온다.
- Datatrame[['column1, 'column2']]: dataframe의 column1과 column2의 데이터를 가져온다.
- Column을 두 개 이상 가져올때 대괄호를 두 개 붙여줘야 한다. (series 형태가 불가능하기 때문)

CITYEYELAB

Indexing

▶ loc 함수를 이용한 indexing

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

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

```
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

```
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

```
df1["제품"]["B"]
```

```
'snack'
```

```
df1.loc["B","제품"]
```

```
'snack'
```

```
df1.at["B","제품"]
```

```
'snack'
```

Indexing

- Datafrmae.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 함수가 더 적절하다.

Indexing

▶ loc 함수는 여러 데이터를 불러올 수 있다.

```
df1.loc[["B","A"],"제품"]
```

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

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

	제품	가격
A	gum	1000
B	snack	3000

```
ind = ["A","B"]
col = ["제품","가격"]

df1.loc[ind,col]
```

	제품	가격
A	gum	1000
B	snack	3000

▶ at 함수는 하나의 데이터만 불러올 수 있다.

```
df1.at[["B","A"],"제품"]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-5-f694ce6353f9> in <module>
----> 1 df1.at[["B","A"],"제품"]

~₩anaconda3₩lib₩site-packages₩pandas₩core₩indexing.py in __getitem__(self, key)
   2078             return self.obj.loc[key]
   2079
-> 2080         return super().__getitem__(key)
   2081
   2082     def __setitem__(self, key, value):

~₩anaconda3₩lib₩site-packages₩pandas₩core₩indexing.py in __getitem__(self, key)
   2025
   2026         key = self._convert_key(key)
-> 2027         return self.obj._get_value(*key, takeable=self._takeable)
   2028
   2029     def __setitem__(self, key, value):

~₩anaconda3₩lib₩site-packages₩pandas₩core₩frame.py in _get_value(self, index, col, takea
ble)
   3008
   3009         try:
-> 3010             loc = engine.get_loc(index)
   3011             return series._values[loc]
   3012         except KeyError:

pandas₩_libs₩index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas₩_libs₩index.pyx in pandas._libs.index.IndexEngine.get_loc()

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

Indexing

- Datafrmae.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 함수가 더 적절하다.

Indexing

▶ 비교 연산자를 이용한 indexing

```
df1['가격']>2000
```

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

```
df1[df1['가격']>2000]
```

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

* 특정 column의 조건을 걸어 bool 형태로 표현할 수 있다.
* Dataframe을 대괄호 밖에 한번 더 언급해주면, 해당 조건을 만족하는 데이터가 출력된다.

```
df1['제품'] == 'gum'
```

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

```
df1[df1['제품'] == 'gum']
```

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

* 특정 항목의 일치 여부는 등호를 두 번 써서(==) 확인할 수 있다.

```
df1['제품'] != 'gum'
```

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

```
df1[df1['제품'] != 'gum']
```

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

* '!=' 기호를 이용해 불일치 여부도 확인할 수 있다.

CITYEYELAB

Indexing

▶ 비교 연산자를 이용한 indexing

```
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

* isin() 함수를 이용해 특정 column에 있는 여러 데이터를 확인할 수 있다.

```
data = ['gum','snack']
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

* 확인하고자 하는 데이터를 리스트 형태로 사용할 수 있다.

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

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

* '~' 기호를 이용하면, 해당 조건의 반대가 되는 값을 출력한다.

Add / Delete

▶ 데이터를 새로운 변수에 추가

```
df1['환불여부'] = "정상"
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

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

```
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

```
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

```
cond = ['구매여부','환불여부']
df1.drop(cond,axis=1)
```

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

Add / Delete

- 데이터를 새로운 column에 추가하려면 위와 같이 등호 기호를 이용하여 생성
- Drop(): 기존의 데이터를 index나 column을 중심으로 삭제(axis = 0: index / axis = 1: column)

Change

▶ 문자열 Replace 함수를 이용한 특정 값 변경

```
df1['판매여부'].replace(True,1)
```

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

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

```
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 데이터 테이블에 선언하여, 지속적으로 사용이 가능

Change

▶ 문자열 Replace 함수를 이어 사용하여 특정 값 변경

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

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

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

- replace함수를 비롯한, 대부분의 pandas 함수가 뒤에 이어 붙여 사용할 수 있다.
- 범주형 데이터에서 여러 값을 동시에 바꿀 때, 주로 사용한다.
- 변경된 값들을 새로운 변수에 선언할 수 있다.
- 데이터 분석 전 데이터 전처리 단계에서 매우 많이 사용된다.

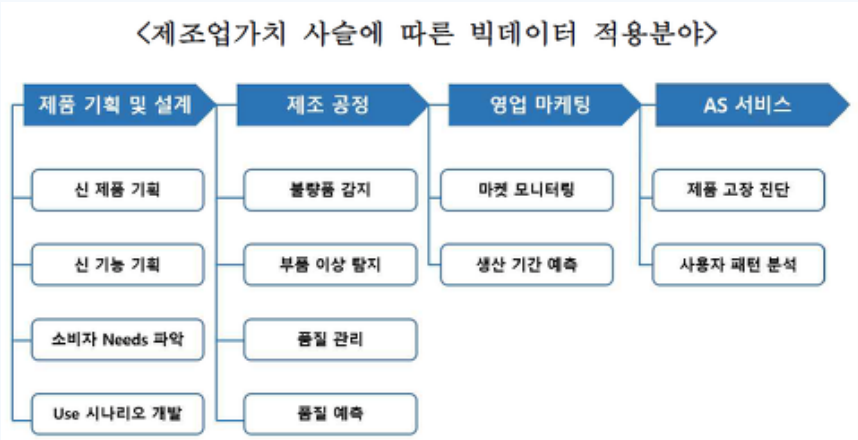
CITYEYELAB

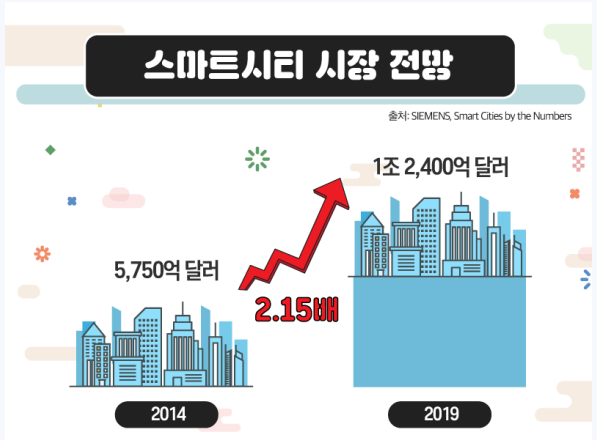
Data Load

Data 불러오기

- Pandas를 이용해 csv 혹은 excel 형식의 데이터 파일을 불러올 수 있음
- 필요한 경우에는 데이터 파일 간 병합(join)을 수행해야 함
- read_csv() 명령어를 이용하여, 데이터를 불러옴
- 데이터를 불러올 때, 데이터와 쥬피터노트북 파일이 같은 폴더에 위치 해야함

  * 위치가 다를 경우 상대 경로를 지정해주어야 한다.
- 기업에서 사용하는 데이터는 대부분 csv 나 excel 형태로 추출이 가능함
- 공공데이터포털이나 기상청 등 국가기관에서 공공데이터를 Open API 형태로 제공해주고 있음





Data Load

▶ 데이터 불러오기(Open API)

```
data = pd.read_csv('cards_data.csv')
```

```
data = pd.read_csv('C:/Users/USER/Desktop/강의진행/211004_국민카드임직원강의/강의자료/data/cards_data.csv')
```

- **Open API 주소**
  https://data.go.kr/tcs/dss/selectApiDataDetailView.do?publicDataPk=15061308
- **URL 주소**
  http://46.101.230.157/dilan/pandas_tutorial_read.csv

▶ URL 데이터 불러오기

```
url_data = 'http://46.101.230.157/dilan/pandas_tutorial_read.csv'
table1 = pd.read_csv(url_data)
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;...

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

Data Load

▶ 데이터 불러오기

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

▶ header를 생략하고 싶을 때

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

▶ column 명을 names 함수를 이용해 지정

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

▶ column 명을 제외하고, 첫번째 열(row) 데이터를 삭제

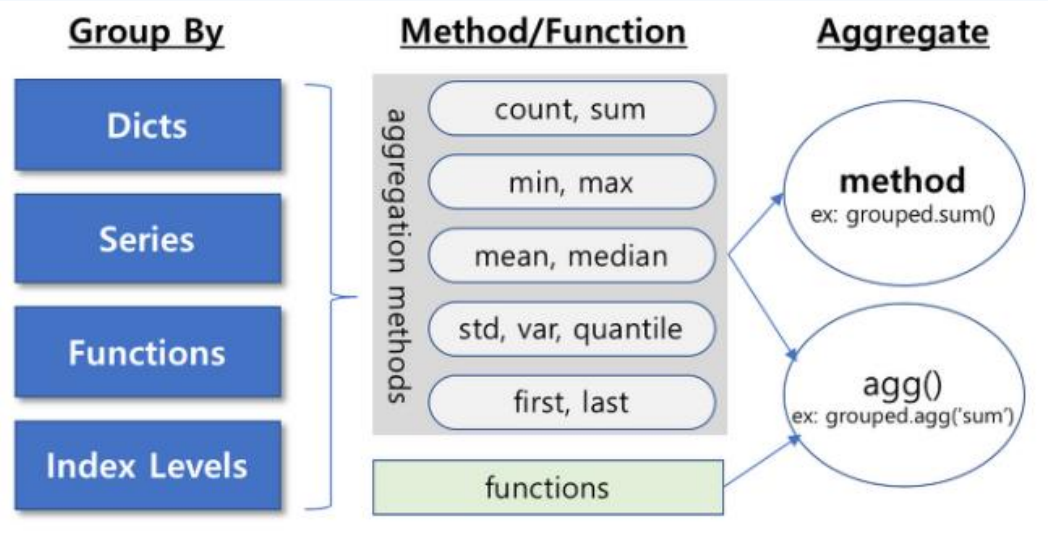
CITYEYELAB

Group by & Pivoting

Group by & Pivoting

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

  * Group by: 항목 별 값에 대한 결과를 series 형태로 출력한다.
  * Pivot table: 항목 별 값에 대한 결과를 DataFrame 형태로 출력한다.



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

Group by

▶ store_market_data.csv 데이터 불러오기

```
import pandas as pd
import numpy as np
```

```
data = pd.read_csv('store_market_data.csv')
print(data.shape)
```

```
(705571, 20)
```

```
data.head()
```

	공급일자	요일	공급월	공급주차	회원번호	조합원상태	물품대분류	물품중분류	물품소분류	물품명	구매수량	주소-구
0	2018-01-02	화	1	1	272369856	정상회원	과실	과일	사과	사과/유(1.5kg)	1.0	수지구
	2018-					정상			느타리	느타리버섯		수지

```
data.groupby('요일').mean()
```

요일	공급월	공급주차	회원번호	구매수량	연령	구매금액	반품_원거래일자
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 column 별 group by

```
data.groupby('요일')['구매금액'].mean()
```

```
요일
금    11064.344340
목    10857.069337
수    11062.047300
월    10542.528917
일    10939.249736
토    11046.589839
화    11002.352378
Name: 구매금액, dtype: float64
```

```
data.groupby('요일')['구매금액'].sum()
```

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

Pivot Table

▶ Pivot Table을 사용, T 함수를 이용해 결과를 row 형태로 정렬

```
pd.pivot_table(data = data, 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 사용할 수 있다.

```
pd.pivot_table(data = data, 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 계산

```
pd.pivot_table(data = data, 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 테이블을 생성한다.
- aggfunc 함수를 이용하여, 최대 / 최소 / 표준편차 등을 구할 수 있다.

CITYEYELAB

연습문제

Example 3-1. Pandas 라이브러리를 호출하고 예제 데이터(store_market_data.csv) 불러와 상위 7개 데이터와 하위 7개 데이터를 확인하기

Example 3-2. 예제 데이터의 '물품대분류'의 카테고리 수와 '물품대분류'에 따른 '구매금액'의 합을 pivot table을 이용해 구하기

Example 3-3. 예제 데이터의 모든 연속형 자료에 대한 요약 통계량 확인하기

Example 3-4. 예제 데이터의 성별을 1과 0으로 라벨링 처리 하기

Example 3-5. 구매 수량이 음수(-)인 값들만 따로 추출해 새로운 테이블로 선언

CITYEYELAB

연습문제

Example 3-6. 예제 데이터에서, '회원 번호'를 기준으로 '구매 금액'의 합을 계산하고,
가장 많이 구매를 한 고객 5명을 확인하기

Example 3-7. 예제 데이터에서 '요일'별로 '물품대분류'의 '구매금액'의 합을 계산하고,
판매량이 가장 높은 요일과, 해당 요일에 가장 많이 판매된 품목을 확인하기

Example 3-8. 예제 데이터에서 '연령대'별로 '구매금액'의 평균을 계산하고,
구매금액의 평균이 가장 높은 연령대를 확인하기

Example 3-9. 예제 데이터에서 일별로 '구매금액'의 총합을 계산하고,
가장 적게 팔린 날을 확인하기

Example 3-10. 예제 데이터에서 '공급주차'별로 '구매금액'의 총합을 계산하고,
가장 많이 팔린 '공급주차'를 확인하기