

Python 기초

*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

목차

파이썬 소개

파이썬의 데이터 타입

Number, Boolean, String

Python의 자료형

리스트, 튜플, 집합, 딕셔너리

조건문과 반복문

함수와 클래스

Python으로 파일 다루기

파일 읽기/쓰기/생성하기

예외처리

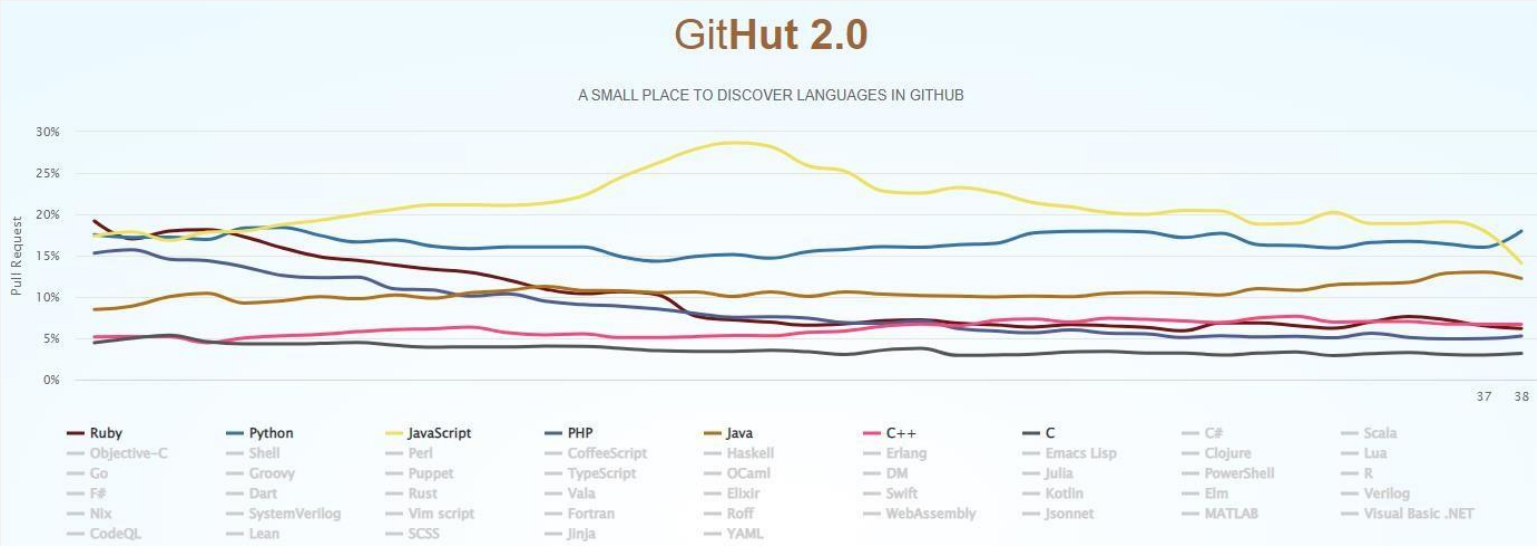
Python의 기초 라이브러리

Numpy, Pandas

1.

Python의 소개

Python 의 대중성



2021년 GitHub에서 사용된 프로그래밍 언어 순위

*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

Hello Python

- 주석 (comment)
- 변수 (variable)
- 데이터 (data)
- 함수 (function)

```
# 안녕하세요.  
say = "안녕하세요"  
print(say)
```

주석 (comment)

“#” 버튼 이후는 코드가
아닌 주석으로 처리되어
실행되지 않음

코드와 같은 줄이어도
상관 없음

```
# 안녕하세요.  
say = "안녕하세요"  
print(say)
```

변수 (variable)

- 알파벳, 숫자, 언더바 (_) 를 조합하여 사용
- 숫자로 시작할 수 없음
- 특수기호로 시작할 수 없음

```
# 안녕하세요.
```

```
say = "안녕하세요"
```

```
print(say)
```

함수 (function)

함수란 어떠한 로직 처리를 위해 미리 작성되어 모듈화 된 코드

파이썬에서의 `print()` 함수는 미리 구현되어 있는 내장함수로 변수의 값, 연산 결과 등을 확인 `print()` 출력 함수이고 입력 함수는 `input()`

```
# 안녕하세요.
```

```
say = "안녕하세요"
```

```
print(say)
```


2.

Python DataType

Number(숫자형), String(문자열)

Boolean(불리안, 참/거짓)

Number(숫자형)

숫자형의 종류

`int()`: 정수

`float()`: 실수

`complex()`: 복소수

2진수, 8진수, 16진수

`type()` 함수로 변수의 자료형을 파악할 수 있음

```
a = int(3)
b = float(3)
c = complex(3,2)
print(type(a))
<class 'int'>

print(type(b))
<class 'float'>

print(type(c))
<class 'complex'>
```

Number(숫자형)

비교 연산자는 값의
크기를 비교할 때
사용하는 연산자

비교 연산의 종류

결과값으로 **True** 또는
False 를 반환 같
은지 다른지, 크기
비교

연산	의미
$a > b$	a 가 b 보다 크다
$a < b$	a 가 b 보다 작다
$a \geq b$	a 가 b 보다 크거나 같다
$a \leq b$	a 가 b 보다 작거나 같다
$a != b$	a와 b는 다르다
$a == b$	a와 b는 같다

Number(숫자형)

산술 연산자는 값을 더하거나 빼는 등의 처리를 하는 연산자

산술 연산의 종류

덧셈 (+), 뺄셈 (-), 곱셈 (*), 제곱 (**)
나눗셈 (/), 나머지 (%), 몫 (//)

연산	의미
$a+b$	a 더하기 b
$a-b$	a 빼기 b
$a*b$	a 곱하기 b
a/b	a 나누기 b
$a//b$	a를 b로 나눈 몫
$a\%b$	a를 b로 나눈 나머지
$a**b$	a의 b제곱

Number(숫자형)

복합대입연산자

복합 대입연산자는

연산과 변수에 값

대입을 동시에 수

행하는 연산자

구문이 더 짧아 가독성이

좋아짐

연산	의미
$a += b$	$a = a + b$
$a -= b$	$a = a - b$
$a *= b$	$a = a * b$
$a /= b$	$a = a / b$
$a //= b$	$a = a // b$
$a \% = b$	$a = a \% b$
$a ** = b$	$a = a ** b$

String(문자열)

문자와 단어로 이루어진 데이터

큰 따옴표 (“”) 나 작은 따옴표 (‘’) 로 생성 파

이썬은 문자열에 대한 다양한 메소드를 지원

문자열끼리 더하기는 문자열을 결합함

문자열에 숫자를 곱하면 지정한 숫자만큼 문자열이 반복

```
a = 'thank'
b = "you"
a+b
'thankyou'
a * 2
'thankthank'
```

String(문자열)

print() 함수로 문자열을 표현할 때에는 “문자열”.format()의 형태

문자열 안에 나타내고 싶은 변수의 위치에 대괄호 {}를 삽입하여
변수 표현 가능

format() 안에는 나타내고 싶은 순서대로 변수가 위치해 ➊ 함

```
name = '이상훈'
email = 'lshlsh13579@gmail.com'
print('{}의 이메일 주소는 {} 입니다.'.format(name,email))
이상훈의 이메일 주소는 lshlsh13579@gmail.com 입니다.
```

String(문자열)

문자열 관련 주요 메소드(메소드란 변수 뒤에 ‘.함수’ 형태로 사용하는 함수)

파이썬에서는 다양한 메소드를 지원.

`split(x)`: 입력으로 들어간 `x` 문자를 구분자로 하여 문자열을 분리하여 리스트로 반환, 구분자를 지정하지 않으면 공백이 구분자가 됨

```
s = "안녕하세요. 제 이름은 이상훈 입니다."
result = s.split(' ') # 빈칸으로 쪼갬다
print(result)
['안녕하세요.', '제', '이름은', '이상훈', '입니다.']
```


String(문자열)

문자열 관련 주요 메소드(메소드란 변수 뒤에 ‘.함수’ 형태로 사용하는 함수)

파이썬에서는 다양한 메소드를 지원.

‘x’.join(y): y의 모든 요소를 x를 구분자로 하여 연결한 문자열을 반환. y 에는 리스트나 튜플과 같은 데이터 컨테이너가 들어가

① 합

```
s = "안녕하세요. 제 이름은 이상훈 입니다."
result = s.split(' ') # 빈칸으로 쪼갬
print(result)
['안녕하세요.', '제', '이름은', '이상훈', '입니다.']
result2 = ' '.join(result) # 빈칸을 활용하여 단어들로 구성된 result (리스트)를 문자열(string)으로 반환
print(result2)
안녕하세요. 제 이름은 이상훈 입니다.
```

String(문자열)

파이썬에서는 다양한 메소드를 지원.

upper(): 모든 문자를 대문자로 바꾼 문자열을 반환

lower(): 모든 문자를 소문자로 바꾼 문자열을 반환

capitalize(): 문자열 첫 글자는 대문자 나머지는 모두 소문자로
바꾼 문자열을 반환

```
email = 'lshlsh13579@gmail.com'
email.upper()
Out[31]: 'LSHLSH13579@GMAIL.COM'
email.lower()
Out[32]: 'lshlsh13579@gmail.com'
email.capitalize()
```

String(문자열) 메소드

메소드	기능
<code>capitalize</code>	문자열의 첫글자는 대문자로, 나머지는 소문자로 변환한다.
<code>center</code>	지정 문자로 문자열 가운데 정렬
<code>find</code>	문자열 처음 발견 위치 찾기/검색
<code>format</code>	지정 값을 지정 형식으로 포맷/변환
<code>join</code>	구분자 문자열로 연결/결합
<code>lower</code>	소문자로 변환
<code>replace</code>	문자열 일부를 지정 문자열로 대체
<code>split</code>	왼쪽부터 문자열을 지정 구분자 기준으로 쪼개기 후 List 생성
<code>strip</code>	앞뒤 공백 제거

*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

String(문자열) 메소드

메소드	기능
isalnum	모든 문자가 알파벳이나 숫자면 True 반환
isalpha	모든 문자가 알파벳이면 True 반환
isdecimal	모든 문자가 0~9면 True 반환
isdigit	모든 문자가 숫자 모양이면 True 반환
isidentifier	문자열이 식별자면 True 반환
islower	모든 문자가 소문자면 True 반환
isnumeric	모든 문자가 숫자이면 True 반환
isprintable	모든 문자가 인쇄 가능한 문자면 True 반환, \n 등은 불가
isspace	모든 문자가 공백이면 True 반환
istitle	모든 단어의 첫 글자가 대문자면 True 반환
isupper	모든 문자가 대문자면 True 반환

*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

Boolean(불리안, 참/거짓)

참과 거짓을 표현하는 데이터, 로직이 동작하게 하는 기본 요소로 논리 연산자를 이용하여 연산

A and B: A와 B가 모두 True 여 **True**

A or B: A와 B 둘 중 하나라도 True 면 True

not A: A의 논리 반대

```
t = True
f = False
print(type(t),type(f))
<class 'bool'><class 'bool'>
print(t and f)
False
print(t or f)
True
print(not f)
True
```

3.

Python의 자료형

리스트(List), 튜플(Tuple), 집합
(Set), 딕셔너리(Dictionary)

리스트(LIST)

파이썬의 대표적인 데이터 컨테이너 자료형

정수나 실수 등을 갖는 변수는 그 값이 하나이므로 많은 데이터 처리가 힘들다

여러 값을 요소로 가져 하나의 변수에 많은 데이터를 담을 수 있음

특징

대괄호 [] 안에 콤마로 구분해 요소 나열

어떠한 자료형도 요소로서 포함이 가능하고 중복될 수도 있음

요소가 순서를 가지므로 특정 위치의 요소나 범위의 요소 확인 가능

```
lst = [60, 30, "20"] # 숫자와 문자를 함께 넣을 수 있다.  
score = [85, 90, 80, 75, 95]
```

리스트(LIST)

일반 변수와의 비교

```
score = [85, 90, 80, 75, 95]
max(score)
95
min(score)
75
sum(score)
425
```

일반 변수를 사용하면 5명 성적을 관리하기 위해 5개 변수가 필요하고
50명이면 50개 변수 필요

리스트를 사용한다면 5명이든 50명이든 학생 성적을 리스트 하나로
관리할 수 있음

sum()/max() 와 같은 파이썬 내장함수를 쉽게 이용할 수 있음

리스트(LIST)

range() 함수로 리스트 만들기

연속된 정수로 리스트를 만들 때 range() 함수를 사용하면 편리

range(시작, 끝, 스텝)

range(n): 0부터 n-1 까지 1씩 증가하는 정수 범위

range(m, n): m부터 n-1 까지 1씩 증가하는 정수 범위

range(m, n, x): m부터 n-1 까지 x 만큼씩 증가하는 정수 범위

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

리스트(LIST)

리스트 인덱싱

리스트 안에서 특정 값을 찾을 때 사용하는 데이터를 인덱스 (Index) 라고 하며

인덱스로 특정 요소를 찾는 과정을 인덱싱 (Indexing) 이라고 함

대괄호 안에 인덱스를 지정함

파이썬의 정수는 0부터 시작, 0부터 시작하는 정수형 위치 인덱스를 사용

리스트의 크기를 벗어나는 인덱스는 에러 유발

인덱스의 역방향은 -1 부터

리스트(LIST)

리스트 인덱싱

```
score = [85, 90, 80, 75, 95]
```

```
score[0]
```

```
Out[48]: 85
```

```
score[1]
```

```
Out[49]: 90
```

```
score[2]
```

```
Out[50]: 80
```

```
score[3]
```

```
Out[51]: 75
```

```
score[4]
```

```
Out[52]: 95
```

```
score[-1]
```

```
Out[53]: 95
```

```
score[-5]
```

```
Out[54]: 85
```

*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

리스트(LIST)

리스트 슬라이싱

인덱스를 사용해 특정 범위의 요소를 찾는 과정이 슬라이싱
대괄호 안에 `[m:n]` 형태로 범위를 지정하여 요소를 확인
`m` 부터 `n-1` 까지, `n` 번째 요소는 결과에 포함되지 않음
`n`이 지정되지 않으면 리스트의 끝까지, `m`이 지정되지 않으면
처음부터

리스트(LIST)

리스트 슬라이싱

```
score = [85, 90, 80, 75, 95]
```

```
score[1:2]
```

```
Out[56]: [90]
```

```
score[1:3]
```

```
Out[57]: [90, 80]
```

```
score[:3]
```

```
Out[58]: [85, 90, 80]
```

```
score[2:]
```

```
Out[59]: [80, 75, 95]
```

리스트(LIST)

리스트 연산

리스트끼리 더하기 (=결합), 숫자와 곱하기 (=반복) 이 가능 문자열 연산과 비슷

```
score = [85, 90, 80, 75, 95]
score2 = [100, 100]
score + score2
Out[62]: [85, 90, 80, 75,
          95, 100, 100]
score * 2
Out[63]: [85, 90, 80, 75, 95,
          85, 90, 80, 75, 95]
```

리스트(LIST)

리스트 요소 추가

`append()` 메소드로 뒤에
요소 하나 추가

`extend()` 메소드로 뒤에
요소 추가

`append()` 와의 차이는
`append()` 는 요소 자체를
추가한다면
`extend()` 는 리스트를
풀어버림

```
score = [85, 90, 80, 75, 95]
score.append([100,100])
print(score)
[85, 90, 80, 75, 95, [100, 100]]
score.extend([99,99])
print(score)
[85, 90, 80, 75, 95, [100, 100],
                      99, 99]
```

리스트(LIST)

리스트 요소 삭제

del 문으로 요소 하나
지우기

```
score = [85, 90, 80, 75, 95]
del score[4]
print(score)
[85, 90, 80, 75]
```

범위로 여러 요소 지우기

```
score[2:] = []
print(score)
[85, 90]
```


리스트(LIST)

리스트 복사

LIST2 = LIST1

LIST2 변수에 LIST1 데이터를 대입한 것으로 이름만 다른 같은 리스트

LIST3 = LIST1[:]

LIST1 을 복사한 것으로 새로운 리스트를 만듦

```
nums = list(range(10)) #range 함수를 사용해서 0부터 9까지의 정수를 요소로 갖는 리스트 생성
nums1 = nums # nums1에 nums를 복사하는것 같지만 사실 같은 객체를 가리키는 것
nums2 = nums[:] # 복사
nums[0] = "a"
print(nums)
['a', 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(nums1)
['a', 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(nums2)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

List(문자열) 관련 메소드

메소드	기능
append	리스트 맨 뒤에 요소 추가
count	리스트 안에 지정한 요소가 몇 개 있는지 반환, 없으면 0
del	지정한 위치에 요소 삭제
extend	리스트에 새로운 리스트를 이어 붙임
index	지정한 값이 있는 요소의 위치 반환, 같은 요소가 없으면 에러
insert	지정한 위치에 요소 삽입
pop	지정한 위치의 요소를 반환하고, 리스트에서 그 요소를 삭제
remove	첫 번째 나타나는 지정한 요소 삭제, 없으면 에러
reverse	리스트 안의 요소를 뒤집음
sort	리스트 안의 요소 정렬

튜플(Tuple)

- 튜플도 리스트와 비슷한 데이터 컨테이너로 소괄호 () 를 사용해서 선언

- 튜플은 불변 객체, 한 번 만들면 요소의 값을 바꿀 수 없음 (Immutable)

- 그 외는 리스트와 같음, 요소 변경, 추가, 제거를 제외한 인덱싱, 슬라이싱은 지원

```
tpl = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
tpl[1] = 0
```

```
Traceback (most recent call last):
```

```
File "C:\Users\dudtj\AppData\Local\Temp\ipykernel_16000\4048645082.py",
```

```
line 1, in <module> tpl[1] = 0 TypeError: 'tuple' object does not support
```

```
item assignment
```

*모든 자료에 대한 권한은 매터코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

집합(SET)

- 집합은 중괄호 {} 를 사용해서 선언하는 자료형으로 중복을 허용하지 않음
- 집합 연산 (교집합, 합집합, 차집합, 대칭 차집합) 을 위한 자료형
- 중복된 원소는 하나만 제외하고 모두 무시됨
- 원소의 순서가 의미가 없으므로 인덱싱/슬라이싱 불가

집합(SET)

집합의 연산

합집합: + 또는 union 메소드 교집

합: & 또는 intersection 메소드

차집합: - 또는 difference 메소드 사용

대칭 차집합: ^ 또는 symmetric_difference 메소드 사용

집합(SET) 관련 메소드

메소드	기능
add	집합에 원소 하나를 추가
clear	집합에서 모든 원소 제거
discard	집합에서 특정 원소 제거, 지정한 원소를 찾지 못하면 무시
pop	집합에서 랜덤하게 원소 하나 제거 후 반환
remove	집합에서 특정 원소 제거, 지정한 원소를 찾지 못하면 에러 발생
update	집합에서 여러 원소 추가(리스트, 튜플, 딕셔너리는 분해되어 들어감)

딕셔너리(Dictionary)

{key1:value1, key2:value2}의 형태를 가지는 자료형으로
key에 대응하는 value를 가지는 자료형

딕셔너리는 요소의 순서가 의미가 없으며, 인덱싱과
슬라이싱이 불가능

딕셔너리는 key를 사용해 value를 확인

Key	'이름'	'성별 '	'나이 '	'전화번호'	'지역'	'키 '
Value	'이상훈'	'남'	33	'010-1234-5678'	'서울'	170

위 딕셔너리의 이름이 `diction` 이라면 *`diction['이름']` 입력시
'이상훈' 반환*

딕셔너리(Dictionary)

- .items()
- .keys()
- .values()

- 위 세 가지 메소드로
딕셔너리의 내용을
확인할 수 있음

```
information = {"이름" : "이상훈", "나이": "32",
               "키": "170"}

type(information)
dict
information.items()
dict_items([('이름', '이상훈'), ('나이', '32'),
            ('키', '170')])
information.keys() dict_keys(['이름', '나이', '키'])
information.values() dict_values(['이상훈', '32', '170'])
```


딕셔너리(Dictionary) 관련 메소드

메소드	기능
keys	모든 key를 요소로 갖는 dict_keys 개체 반환
values	모든 value를 요소로 갖는 dict_values 개체 반환
items	모든 key와 value를 (key, value)형태로 갖는 dict_items개체 반환
clear	모든 요소 삭제
get	key로 value를 찾을 때 없는 경우 대신 표시할 값 지정

```
information = {"이름" : "이상훈", "나이":"32", "키":"170"}
information['성별'] # 성별이라는 key가 없다
Traceback (most recent call last):
File "C:\Users\duadtj\AppData\Local\Temp\ipykernel_16000\2043907878.py", line 1, in
<module> information['성별'] # 성별이라는 key가 없다
KeyError: '성별'

information.get('성별') # 에러는 발생하지 않음 information.get('성별', "key error") # key가 없으니 원하는 출력 문자 설정 Out[102]: 'key error'
```

*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

4.

조건문과 반복문

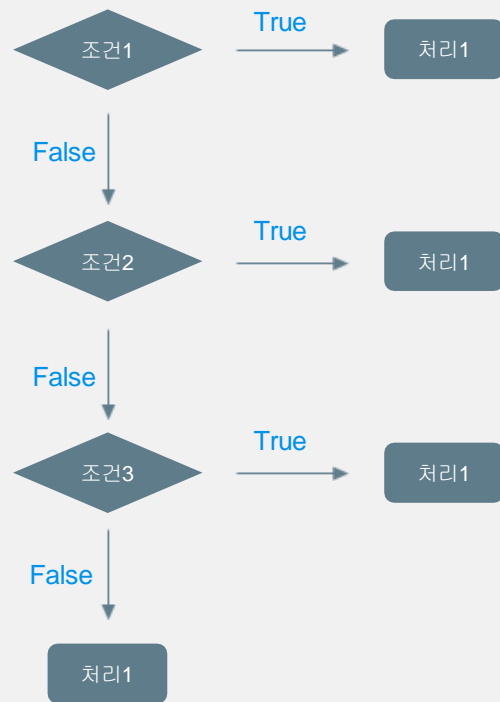
if, elif, else / for, while

조건문

if: 조건이 참이면 if 문
안의 로직

elif: if 조건이 틀리고
현재 조건이 맞으면

else: if/elif 조건이 다
틀렸을 때



조건문

조건문 안에 수행할
내용은 들여쓰기

```
score = 30
if score < 30:
    print("fail")

elif score >= 30:
    print("pass")

else :
    print("error")
pass
```

반복문(For 문)

조건이 참이냐 거짓이냐에 따른 로직 구성

for 변수 in 리스트, 튜플, 문자열 (반복 가능한 객체):
 처리할 구문

```
for i in range(3):  
    print(i)  
0  
1  
2  
for i in [0,1,2]:  
    print(i)  
0  
1  
2  
for i in ['apple', 'banana', 'cheery']:  
    print(i)  
apple  
banana  
cheery
```

*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

반복문(While 문)

조건문이 True 인 동안 while 문 안의 문장이 반복해서 실행됨

while 조건문:

처리

특정 상황에서 반복을 중지하기 while 문 안에서 조건을 제어함

```
i = 0
while i < 3:
    print(i)
    i+=1
0
1
```

반복문(While 문)

break

반복문 안에서 특정 조건이 되어 반복문을 빠져 나올 때 사용
for 문에도 똑같이 적용 가능

continue

반복문을 중단시키지 않고 다음 반복으로 넘어갈 때 사용

```
while True: # 구문 내에서 break이 되지 않으면 계속 반복된다.
    input_data = input("정수의 짝수만 알려드립니다. \n 종료하시려면 . 입력\n")
    if not input_data.isnumeric() :
        break
    elif int(input_data) % 2 == 0 :
        print("{}는 짝수입니다".format(input_data))
    elif int(input_data) % 2 == 1 :
        continue
```

반복문

리스트와 반복문 반복할 범위를 리스트(튜플)로 정하여 반복

`enumerate()`

몇 번째 반복인지 알려주는 인덱스도 함께 반환

```
name = ['이정재', '박해수', '오영수']
for i in name:
    print(i)
이정재 박해
수 오영수
```

```
for i, j in enumerate(name):
    print(i, j)
1 이정재
2 박해수
3 오영수
```


반복문

리스트와 반복문

복잡한 반복문을 작성하지 않고 조건에 맞는 데이터만 가져올 수 있다.

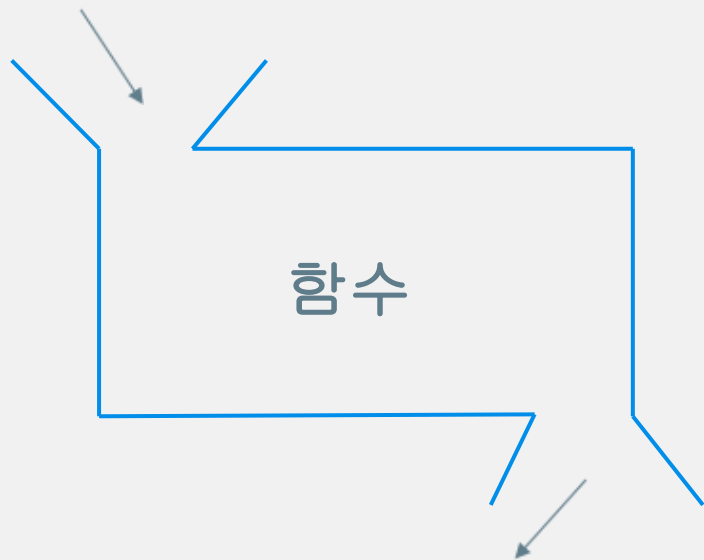
```
lst = list(range(9))
lst_odd = [x for x in lst if x % 2 == 1]
lst_even = [x for x in lst if x % 2 == 0]
print(lst_odd)
[1, 3, 5, 7]
print(lst_even)
[0, 2, 4, 6, 8]
```

5

함수와 클래스

함수

입력



출력

함수란 입력을 받아 무엇인가를 처리한 후 그 결과를 반환하는 것

프로그래밍에서 함수란 반복해 사용할 코드를 미리 정의해두고 필요할 때 사용하는 모듈로 같은 로직을 반복해서 코딩할 필요가 없어짐

*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

함수

함수는 `def` 문으로 정의

```
def 함수이름(매개변수):  
    처리할 로직  
    return 출력값
```

```
def sum1(a,b):  
    c = a+b  
    return c  
result = sum1(3,2)  
print(result)  
5
```

함수의 입력과 출력

대부분의 함수는 매개변수를 입력 받아 처리한 후 결과값을 반환

입력 매개변수가 없거나 반환되는 값이 없을 수 있음

```
def sum1(a,b):  
    c = a+b  
  
    return  
  
result = sum1(3,2)  
  
print(result)  
  
None
```

함수와 변수

기본적으로 함수는
함수 밖의 변수를
변경할 수 없음
def 구문 안에 있는 변수에만 권한이 있음

변경하려면 반환된 값을 받거나 **global** 키워드를 사용해 ㉠ 함

```
a = 1
def sum1(a):
    a+=1
sum1(a)
print(a)
1
```

```
a = 1
def sum1():
    global a
    a+=1
sum1()
print(a)
2
```

함수와 변수

- 여러 개의 값을 한 번에 받을 때, 함수가 반환하는 값이 여러 값일 경우 개수에 맞춰 변수를 할당해주어 ❶ 함
- 원하지 않는 값일 경우 '_' 표시로 반환을 생략할 수 있음
- 하나의 변수로 여러 값을 받을 때는 하나의 변수가 여러 값을 지닌 튜플이 됨

```
def sum1_1(a,b):
    return a, b, a+b

d,e,f = sum1_1(1,2) # 함수의 output이 3개일때 각각의 위치를 선정해주면 하나씩 받아진다.
print(d,e,f)
1 2 3

g = sum1_1(1,2) # 하나로 받아오면 tuple에 받아진다.
print(g)
(1, 2, 3)
```

함수와 변수

- 함수로 보내는 파라미터 개수가 정해져 있어 ❶ 하나? (가변 매개변수)
- 입력하는 변수가 10개일 때 10개를 전부 함수 input으로 적어주는 것이 아닌 *를 사용

```
def sum3(*args): # *arguments의 줄임말, 받으면 튜플로 받아들인다.  
    return sum(args), type(args)  
sum3(1,2,3,4,5,6,7,8,9,10)  
Out[149]: (55, tuple)
```


함수와 변수

함수의 input으로 딕셔너리를 사용하는 경우 *를 두개 사용하여 받음

```
def mean1(**kwargs): #kwargs : keyword argument의 줄임말
    print(type(kwargs))
    return sum(kwargs.values())

mean1(a=1,b=2)
<class 'dict'>
Out[150]: 3
```

클래스

함수만으로도 프로그램
만들기가 가능. 클래스
가 왜 필요할까? 계산한
잔고 유지하기 위해서
result 전역 변수
(**global**) 을 사용함
만일 2종류의 코인
잔고가 필요하다면?

```
result1 = 0
def eth_balance(num):
    global result1
    result1 += num
    return result1

result2 = 0
def ethc_balance(num):
    global result2
    result2 += num
    return result2

print(eth_balance(3))
print(eth_balance(6))
print(ethc_balance(11))
print(ethc_balance(6))
3
9
11
17
```

클래스

여러 개의 잔고가 필요
 할수록 전역 변수와 함수를 추가해 주어야 함
 클래스를 사용한다면 ?
 Calculator 클래스로 만든
 별개의 잔고 `balance1`,
`balance2` (객체)
 코인의 종류가 늘어나 계산해 하는 잔고가 늘어나더라도 클래스를 통해 객체만 생성하면 됨

```
class coin:
    def __init__(self, name):
        self.name = name
        self.shares = 0
    def buy(self, shares):
        self.shares += shares
        print(self.name +
              " 코인 {} 개 매수 완료".format(shares))

balance1 = coin('eth')
balance1.buy(3)
eth 코인 3 개 매수 완료
balance1.buy(4)
eth 코인 4 개 매수 완료
balance2 = coin('ethc')
balance2.buy(43)
ethc 코인 43 개 매수 완료
balance2.buy(45)
ethc 코인 45 개 매수 완료
```

클래스

```
balance1 = coin('eth')
```

```
balance1.buy(3)
```

```
class coin:
    def __init__(self,name):
        self.name = name
        self.shares = 0

    def buy(self,shares):
        self.shares+=shares
        print(self.name +
              " 코인 {} 개 매수 완료".format(shares))
```

클래스에서 사용되는 첫번째 매개변수는 관례적으로 **self**를 사용하는 것.

간단한 클래스

코인의 매수매도와 잔고를
계산하는 클래스를 만들어
본다.

매수/매도/잔고현황 확인

`def __init__(self,)`
클래스를 통해 객체 생
성할 때부터 변수
선언 가능

```
class coin:
    def __init__(self, name):
        self.name = name
        self.shares = 0

    def buy(self, shares):
        self.shares += shares
        print(self.name + " 코인 {} 개 매수 완료".format(shares))

    def sell(self, shares):
        self.shares -= shares
        print(self.name + " 코인 {} 개 매도 완료".format(shares))

    def balance(self):
        print("현재 "+self.name+
              " 코인은 {}개 보유중입니다.".format(self.shares))
```

간단한 클래스

```
eos = coin("EOS")
eos.buy(3)
EOS 코인 3 개 매수 완료
eos.balance()
현재 EOS 코인은 3개 보유중입니다.
```

클래스 내의 함수는 메소드라 부른다.

`eos` 객체를 만들고(이때 객체변수 `name`에 “eos”저장), `eos` 객체를 통해 `buy` 메소드와 `balance` 메소드를 호출.

클래스의 상속

상속은 보통 기존 클래스를 변경하지 않고 기능을 추가하거나 기존 기능을 변경할 때 사용

기존 클래스가 라이브러리 형태로 제공되거나 수정이 허용되지 않는 상황이라면 상속을 해 **㉠**함

```
class add_coin(coin):
    def sell_all(self):
        self.shares = 0
        print(self.name + " 코인 전량 매도 완료")
eos = add_coin("EOS")

eos.buy(3)
EOS 코인 3 개 매수 완료

eos.sell_all()
EOS 코인 전량 매도 완료

eos.balance()
현재 EOS 코인은 0개 보유중입니다.
```

6

Python으로 파일 다루기

파일 읽기/쓰기/생성하기

파일 생성하기

파일객체 = `open`(파일 이름, 파일 열기 모드)

파일 이름은 문자열

파일 열기 모드에는

‘r’: 읽기 모드, ‘w’: 쓰기 모드, ‘a’: 추가 모드

‘w’ 은 쓰기 모드로 파일 이름으로 된 파일이 현재 경로에 새로 생성

`f.close()` 를 통해 파일 객체를 닫아줌

```
f = open('test1.txt', 'w')  
f.close()
```

파일 읽기

readline()

파일을 한 줄씩 읽음

while True 조건문과 보통 같이 사용

```
f = open('정보.txt','r',encoding='utf8')
while True:
    line = f.readline()
    if not line:
        break
    print(line)
f.close()
이름:이상훈
지역:서울
나이:32 성
별:남
```

*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

파일 읽기

readlines()

파일의 모든 줄을 읽어서 각각의 줄을 요소로 갖는 리스트를 돌려줌
한글은 encoding = 'UTF-8'로 해주어 ㉠ 함

```
f = open('정보.txt','r',encoding='utf8')
lines = f.readlines()
for line in lines:
    print(line)
f.close()
이름:이상훈
지역:서울
나이:32 성
별:남
```

*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

파일 읽기

read()

파일 내용 전체를 문자열로 돌려줌

```
f = open('정보.txt','r',encoding='utf8')
lines = f.read()
print(lines)
f.close() 이
름:이상훈 지
역:서울 나이
:32 성별:남
```

파일 읽고 쓰기

파일에 새로운 내용 추가

같은 파일 이름으로 'w' 을 통해 열었을 때 이미 존재하는 파일의 내용이 사라짐

'a' 추가 모드를 통해 기존 파일에 내용을 추가할 수 있음

```
f = open('정보.txt', 'a', encoding='utf8')
f.write('\n키:170')
Out[197]: 6
f.close()
```

```
f = open('정보.txt', 'r', encoding='utf8')
lines = f.read()
print(lines)
f.close()
이름:이상훈
지역:서울
나이:32 성
별:남 키
:170
```

파일 읽고 쓰기

with 문

지금까지는 open -> close 의 구조

with 문 으로 f.close() 를 자동으로 처리

with 블록을 벗어나는 순간 파일 객체 f 가 자동으로 닫힘

open 이 반환하는 객체를 “as” 로 변수 선언

```
with open('정보.txt','r',encoding='utf8') as f:
    data = f.read()
print(data)
이름:이상훈
지역:서울
나이:32 성
별:남 키
:170
```

7

예외처리

Try. Except.

예외 처리

- 파이썬은 **try, except** 를 사용해서 예외적으로 오류를 처리할 수 있게 해줌
- 디렉토리 안에 없는 파일을 열려고 시도했을 때 발생하는 오류
FileNotFoundError

```
with open('정보1.txt','r',encoding='utf8') as f:
    data = f.read()
    print(data)
```

```
Traceback (most recent call last):
  File "C:\Users\dudtj\AppData\Local\Temp\ipykernel_16000\2901262461.py", line 1, in <module>
    with open('정보1.txt','r',encoding='utf8') as f:
FileNotFoundError: [Errno 2] No such file or directory: '정보1.txt'
```


예외 처리

try:

무엇인가를 하고

except [발생 오류[as 오류 메시지 변수]]:

try 블록에서 오류가 발생하면 여기서 처리

오류가 발생하지 않으면 **except** 블록이 수행되지 않음

try, except 에서 **except** 에 특정 오류만 처리할 수 있음

예외 처리

Try 구문에서 없는 txt 파일을 읽는 과정에서 에러가 발생하지만, except 구문에서 pass로 인해 프로그램이 멈추지 않고 진행 됨

pass : 다음 코드 실행

아래 코드는 try/except 구문이 아니었다면 정지될 코드

```
try:
    with open('정보1.txt','r',encoding='utf8') as f:
        data = f.read()
        print(data)
except :
    pass
print(test)
test
```

8

Python의 라이브러리

Numpy, Pandas

Numpy

- 빠른 수치 계산을 위해 C언어로 만들어진 파이썬 라이브러리
- 벡터와 행렬 연산을 위한 편리한 기능들을 제공
- **Array** 라는 단위로 데이터를 관리하고 연산을 수행



NumPy

Numpy

Array 용어 정리

Axis: 배열의 각 축

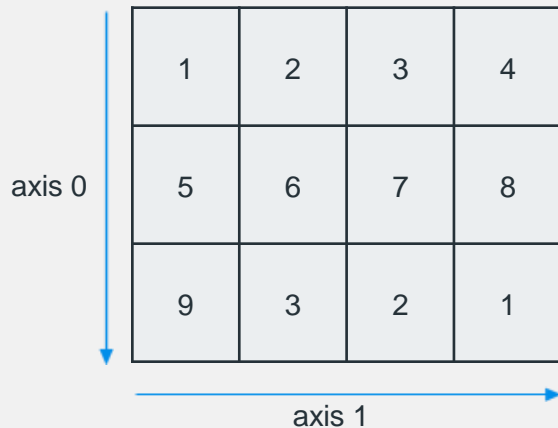
Rank: 축의 개수

Shape: 축의 길이

[3x4] 형태의 배열일 경우

Axis 0/1 을 가지는 2차원 배열.

Axis 0의 길이는 3, axis 1의 길이는 4



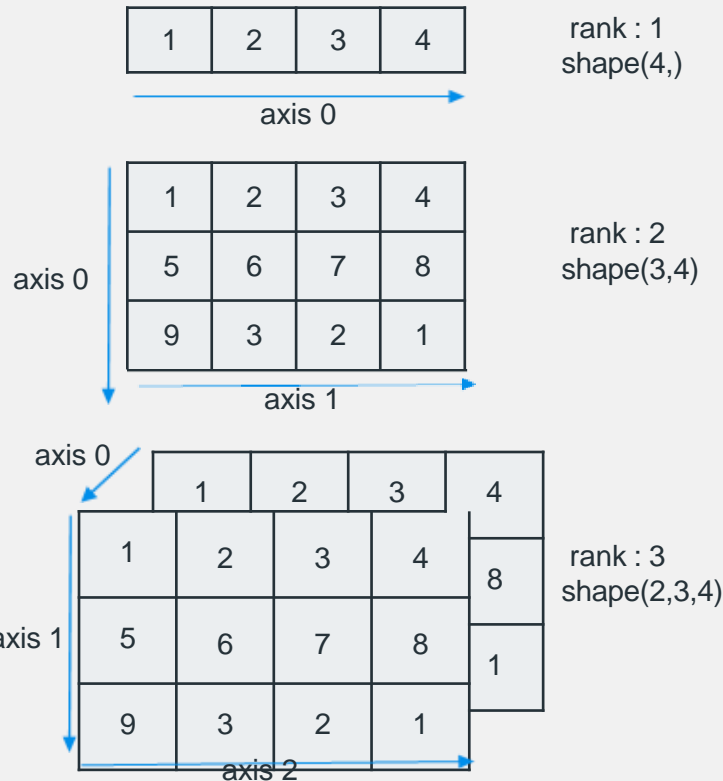
1	2	3	4
5	6	7	8
9	3	2	1

Numpy – Array(배열)

```
import numpy as np
np.array([1,2,3,4])

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

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



*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

Numpy – 배열을 만드는 여러 함수들

- `np.zeros()`: 0으로 채워진 배열
- `np.ones()`: 1로 채워진 배열
- `np.full()`: 특정 값으로 채워진 배열
- `np.eye()`: 정방향 행렬
- `np.random.random()`: 랜덤 값으로 채운 배열

```
import numpy as np
np.ones((2,2))
array([[1., 1.],
       [1., 1.]])

np.full((2,2),10)
array([[10, 10],
       [10, 10]])

np.eye((4))
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

Pandas

- Pandas 라이브러리는 주로 엑셀, csv 등에서 가져오는 데이터들을 처리하는데 사용
- 대표적으로 **Series(시리즈)**와 **DataFrame(데이터프레임)**이 사용됨

Pandas – Series(시리즈)

데이터프레임과 더불어 **pandas** 가 제공하는 자료구조 중 하나 인덱스와 값으로 구성되어 키와 값으로 구성되는 딕셔너리와 유사 열이 하나뿐이므로 열의 의미가 없어 그냥 값이라 부름

index	0
0	300
1	600
2	900
3	1200
4	1500

인덱스 미지정

index	0
2001	300
2002	600
2003	900
2004	1200
2005	1500

인덱스 지정

*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

Pandas – Series(시리즈)

- 인덱스는 언제든지 변경될 가능성이 있으므로 행 번호와 항상 일치하지는 않음
- 보이는 인덱스 값이 변해도 행 번호는 언제나 0, 1, 2, ...
- 필요하면 인덱스 값을 행 번호로 초기화 할 수 있음
- 데이터프레임에서도 같은 의미

index	0
2001	300
2002	600
2003	900
2004	1200
2005	1500

Pandas – Series(시리즈)

- `reset_index()` 메소드로 인덱스를 행 번호에 기반한 정수 값으로 초기화
- `drop=True` 옵션으로 기존 인덱스를 버림
- `drop=False` 옵션일 경우 데이터프레임으로 변환

index	0
2001	300
2002	600
2003	900
2004	1200
2005	1500

원본

index	0
0	300
1	600
2	900
3	1200
4	1500

drop = True

index	index	0
0	2001	300
1	2002	600
2	2003	900
3	2004	1200
4	2005	1500

drop = False

*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

Pandas – Series(시리즈)

pd.Series() 함수를 통해 다른 자료형 데이터를 시리즈로 만들 수 있음

```
lst = [1,2,3,4,5]
ex_1 = pd.Series(lst)
print(ex_1,type(ex_1))

0 1
1 2
2 3
3 4
4 5
dtype: int64
<class 'pandas.core.series.Series'>
```

List -> Series

```
dictionary = {"성별":"남자","이름": "
              이상훈","지역":"서울","나이":"32"}
ex_2 = pd.Series(dictionary)
print(ex_2)

성별  남자
이름  이상훈
지역  서울
나이  32
dtype: object
```

Dictionary -> Series

Pandas – Series(시리즈)

Series 의 인덱싱과 슬라이싱

조회	행 번호로 찾기	인덱스 이름으로 찾기
특정 값	<code>.iloc[행 번호]</code>	<code>.loc['행 이름']</code>
특정 행	<code>.iloc[[행 번호]]</code>	<code>.loc[['행 이름']]</code>
특정 범위	<code>.iloc[행 번호1 : 행 번호2]</code>	<code>.loc['행 이름1' : '행 이름2']</code>

Pandas – Series(시리즈)

Series 의 인덱싱

```
ex_2.loc['이름']
'이상훈'

ex_2.loc[['이름']]
이름 이상훈
dtype: object

ex_2.iloc[1]
'이상훈'

ex_2.iloc[[1]]
이름 이상훈
dtype: object
```

index	0
성별	남자
이름	이상훈
지역	서울
나이	32

Pandas – Series(시리즈)

Series 의 슬라이싱

```
ex_2.iloc[1:3]
이름 이상훈 지
역 서울 dtype:
object

ex_2.loc['이름':'나이']
이름 이상훈
지역 서울
나이 32
dtype: object
```

index	0
성별	남자
이름	이상훈
지역	서울
나이	32

Pandas – DataFrame(데이터프레임)

데이터 분석에서 가장 중요한 데이터 구조로 엑셀 시트와 같은 형태

	trd_date	co_nm	code	adj_prc	volume
0	2022-01-01	삼성전자	005930	70000	324598
1	2022-01-02	삼성전자	005930	70500	98709
2	2022-01-03	삼성전자	005930	69000	724598
3	2022-01-04	삼성전자	005930	65000	32435
4	2022-01-05	삼성전자	005930	67000	424598

Pandas – DataFrame(데이터프레임)

데이터프레임은 `pd.DataFrame()` 함수를 사용해 만들
여러 열을 가지므로 인덱스 외에 열 이름에 대한 설정이 필요
만들 때 열 이름을 지정하거나 나중에 변경할 수 있음

Pandas – DataFrame(데이터프레임)

numpy의 배열(array)를 dataframe으로 변경.

행과 열의 이름을 지정하지 않으면 행과 열의 번호가 곧
이름으로 지정

```
array = np.array([(3,2,31),(4,14,7)])  
ex_1 = pd.DataFrame(array)  
ex_1.head()  
  
  0  1  2  
0  3  2 31  
1  4 14  7
```

Pandas – DataFrame(데이터프레임)

pd.DataFrame(data, index, columns)에서 index와 columns를 활용하여 행과 열의 이름을 정할 수 있음

```
array = np.array([(3,2,31),(4,14,7)])  
ex_1 = pd.DataFrame(array, index = ['a','b'], columns = ['c','d','e'])  
  
ex_1.head()  
   c  d  e  
a  3  2 31  
b  4 14  7
```

Pandas – DataFrame(데이터프레임)

딕셔너리로 만들면 딕셔너리 키가 데이터프레임 열 이름이 됨

```
dictionary = {"성별": "남자", "이름": "이상훈", "지역": "서울", "나이": "32"}  
ex_2 = pd.DataFrame([dictionary])
```

```
ex_2.head()
```

```
  성별  이름  지역  나이  
0  남자  이상훈  서울   32
```

Pandas – DataFrame(데이터프레임)

특정 열의 이름 변경은

```
df.rename(columns={'바꿀열':'새이름'}, inplace=True)
```

Inplace=True를 하지 않으면 원본 df는 변경 되지 않음

```
dictionary = {"성별":"남자","이름":"이상훈","지역":"서울","나이":"32"}
ex_2 = pd.DataFrame([dictionary]) ex_2.rename(columns={'이름': 'name'}, inplace=True)
ex_2.head()
```

```
   성별  name  지역  나이
0  남자  이상훈  서울   32
```

Pandas – DataFrame(데이터프레임)

- `reset_index()` 메소드로 인덱스를 행 번호에 기반한 정수 값으로 초기화
- `drop=True` 옵션으로 기존 인덱스를 버림
- `drop=False` 옵션일 경우 인덱스 열을 일반 열로 유지

index	0
2001	300
2002	600
2003	900
2004	1200
2005	1500

원본

index	0
0	300
1	600
2	900
3	1200
4	1500

drop = True

index	index	0
0	2001	300
1	2002	600
2	2003	900
3	2004	1200
4	2005	1500

drop = True

*모든 자료에 대한 권한은 메타코드에 있으며, 무단으로 자료를 복제 및 배포 등 유료목적으로 활용하시면 별도의 조치가 들어갈 수 있습니다.

Pandas – DataFrame 인덱싱 / 슬라이싱

행(index) 조회

- 행 하나 조회 : `df.loc[행이름]`
- 행 여러개 조회 : `df.loc[[행이름1, 행이름2, ...]]`
- 행 범위 조회 : `df.loc[행이름1:행이름n]`

열(column) 조회

- 열 하나 조회 : `df[열이름]`
- 열 여러개 조회 : `df[[열이름1, 열이름2, ...]]`
- 열 범위 조회 : `df.loc[:, 열 이름1 : 열 이름n]`

Pandas – DataFrame 행(index) 조회

```
samsung.loc['2015-03-31']
co_nm 삼성전자
gicode A005930
adj_prc 28820.0
Name: 2015-03-31, dtype: object

samsung.loc[['2015-03-31','2015-05-29']]
           co_nm gicode adj_prc
trd_date
2015-03-31 삼성전자 A005930 28820.0
2015-05-29 삼성전자 A005930 26140.0

samsung.loc['2015-03-31':'2015-05-29']
           co_nm gicode adj_prc
trd_date
2015-03-31 삼성전자 A005930 28820.0
2015-04-30 삼성전자 A005930 28200.0
2015-05-29 삼성전자 A005930 26140.0
```

trd_date	co_nm	gicode	adj_prc
2015-01-30	삼성전자	A005930	27300
2015-02-27	삼성전자	A005930	27140
2015-03-31	삼성전자	A005930	28820
2015-04-30	삼성전자	A005930	28200
2015-05-29	삼성전자	A005930	26140
2015-06-30	삼성전자	A005930	25360
2015-07-31	삼성전자	A005930	23700
2015-08-31	삼성전자	A005930	21780
2015-09-30	삼성전자	A005930	22680
2015-10-30	삼성전자	A005930	27440

Pandas – DataFrame 열(column) 조회

```
samsung['co_nm']
trd_date
2015-01-30 삼성전자
2015-02-27 삼성전자
2015-03-31 삼성전자

samsung[['co_nm','adj_prc']]
      co_nm adj_prc
trd_date
2015-01-30 삼성전자    27300.0
2015-02-27 삼성전자    27140.0
2015-03-31 삼성전자    28820.0

samsung.loc[:, 'co_nm ':'adj_prc']
      co_nm gi code adj_prc
trd_date
2015-01-30 삼성전자    A005930 27300.0
2015-02-27 삼성전자    A005930 27140.0
2015-03-31 삼성전자    A005930 28820.0
```

trd date	co nm	ai code	adi prc
2015-01-30	삼성전자	A005930	27300
2015-02-27	삼성전자	A005930	27140
2015-03-31	삼성전자	A005930	28820
2015-04-30	삼성전자	A005930	28200
2015-05-29	삼성전자	A005930	26140
2015-06-30	삼성전자	A005930	25360
2015-07-31	삼성전자	A005930	23700
2015-08-31	삼성전자	A005930	21780
2015-09-30	삼성전자	A005930	22680
2015-10-30	삼성전자	A005930	27440

Pandas – DataFrame 인덱싱 / 슬라이싱

행열(DataFrame) 조회

- 특정 행과 특정 열 조회 : `df.loc[[행이름1,행이름2, ...], [열이름1,열이름2, ...]]`
- 행과 열의 범위로 조회 : `df.loc[행이름1:행이름n, 열이름1:열이름n]`

조건 조회

- 열 모두 조회 : `df.loc[조건], or df[조건]`
- 열 선택 조회 : `df.loc[조건,[열이름1, 열이름2, ...]]`

Pandas – DataFrame 행/열 조회

```
samsung.loc[['2015-03-31', '2015-05-29'],
             ['co_nm', 'adj_prc']]
```

```
co_nm adj_prc
trd_date
2015-03-31 삼성전자 28820.0
2015-05-29 삼성전자 26140.0
```

```
samsung.loc['2015-03-31':'2015-05-29',
            'co_nm':'adj_prc']
```

```
co_nm giccode adj_prc
trd_date
2015-03-31 삼성전자 A005930 28820.0
2015-04-30 삼성전자 A005930 28200.0
2015-05-29 삼성전자 A005930 26140.0
```

trd_date	co_nm	giccode	adj_prc
2015-01-30	삼성전자	A005930	27300
2015-02-27	삼성전자	A005930	27140
2015-03-31	삼성전자	A005930	28820
2015-04-30	삼성전자	A005930	28200
2015-05-29	삼성전자	A005930	26140
2015-06-30	삼성전자	A005930	25360
2015-07-31	삼성전자	A005930	23700
2015-08-31	삼성전자	A005930	21780
2015-09-30	삼성전자	A005930	22680
2015-10-30	삼성전자	A005930	27440

Pandas – DataFrame 조건 조회

```
samsung.loc[samsung.index < '2015-05-29',
            ['co_nm', 'adj_prc']]
```

```
      co_nm adj_prc
trd_date
2015-01-30 삼성전자 27300.0
2015-02-27 삼성전자 27140.0
2015-03-31 삼성전자 28820.0
2015-04-30 삼성전자 28200.0
```

trd_date	co_nm	aicode	adj_prc
2015-01-30	삼성전자	A005930	27300
2015-02-27	삼성전자	A005930	27140
2015-03-31	삼성전자	A005930	28820
2015-04-30	삼성전자	A005930	28200
2015-05-29	삼성전자	A005930	26140
2015-06-30	삼성전자	A005930	25360
2015-07-31	삼성전자	A005930	23700
2015-08-31	삼성전자	A005930	21780
2015-09-30	삼성전자	A005930	22680
2015-10-30	삼성전자	A005930	27440

Pandas – csv 파일 읽어오기

파일 읽기는 `pd.read_csv()`, 파일 쓰기는 `pd.to_csv()`

`pd.read_csv()` 주요 옵션

- `Sep`: 구분자 지정 (기본값=coma)
- `header`: 헤더가 될 행 번호 지정 (기본값=0)
- `Index_col`: 인덱스 열 지정 (기본값=False)
- `skiprows`: 처음 몇 줄을 무시할 것인지 지정

Pandas – csv 파일 읽어오기

대량의 데이터를 읽어 온 후 데이터를 탐색

`df.head(num)` : 앞쪽 데이터를 확인, `num` 은 위에서부터의 행 숫자, 생략 가능

`df.tail(num)` : 가장 마지막부터 데이터를 확인, `num`은 `head`와 같은 역할

`df.shape` : 데이터프레임의 (행수,열수)를 반환

`df.columns` : 데이터프레임의 모든 열 이름 확인

`df.index` : 데이터프레임의 모든 `index` 이름 확인