

2018-2

기초 파이썬

3기 김현세, 배민영

CONTENTS

1. 변수와 자료형
2. 제어문
3. 함수
4. 클래스
5. 파일 입출력
6. 모듈
7. 예외 처리
8. 기타 코드 작성 규칙
9. QUEST

변수와 자료형

a = 2

a라는 변수에 2를 할당

변수: 컴퓨터 메모리에 존재하는 값을 참조하기 위한 이름
→ 어떤 값을 담을 수 있는 그릇으로 이해!

변수 이름으로 사용할 수 있는 문자:

소문자, 대문자, 숫자(숫자로 시작할 수는 없음), 언더스코어

변수 이름으로 사용할 수 없는 문자:

예약어(ex.False, True, except, def etc..), 숫자로 시작하는 문자

* 예약어: 파이썬 내에서 이미 특정 의미(혹은 기능)를 가지고 사용되고 있는 단어

할당: 변수가 특정 객체의 메모리 주소를 갖도록 하는 것
→ 변수라는 그릇에 어떤 값을 담는 행위로 이해!

변수와 자료형

숫자 자료형

```
_integer = 123
_float = 1.23
_float2 = 1.23e10
_float3 = 1.23e-10
_octal = 0o123
_Hexadecimal = 0x8123f
```

123 ————— int형 (정수 자료형)

1.23
1.23 * 10¹⁰
1.23 * 10⁽⁻¹⁰⁾ } float형 (소수 자료형)

8진수 123(=83)
16진수 8123f } int형 (정수 자료형)
(=33059)

- 진법과 관계 없이 정수는 모두 int형, 소수는 모두 float형
- 8진수는 '0o'를 붙여서, 16진수는 '0x'를 붙여서 표기

자료형 변환 함수 int(), float()

```
print(float(3))
```

3.0

정수 자료인 3을 소수 자료인 3.0으로 바꿈

```
print(int(12.3444))
```

12

소수 자료인 12.3444를 정수 자료인 12로 바꿈
(소수점 아래는 무조건 버림)

※ print(): 각종 데이터를 출력해주는 함수

변수와 자료형

연산자

사칙연산: +, -, *, /

// 연산자: 정수 나누기(나머지를 버리고 몫을 구함)

% 연산자: 나누기 시 나머지

** 연산자: 제곱

대입 연산자:

$x += y \leftrightarrow x = x + y$

$x -= y \leftrightarrow x = x - y$

$x *= y \leftrightarrow x = x * y$

$x /= y \leftrightarrow x = x / y$

사칙연산

```
>>> 1 + 3
```

```
4
```

```
>>> 1 - 3
```

```
-2
```

```
>>> 2 * 3
```

```
6
```

```
>>> 3 / 5
```

```
0.6
```

기타연산

```
>>> 12 // 5
```

```
2
```

```
>>> 12 % 5
```

```
2
```

```
>>> 12 ** 5
```

```
248832
```

```
>>> x = 3
```

```
>>> x += 3
```

```
>>> x
```

```
6
```

12/5의 몫

12/5의 나머지

12^5

$x+3=6$ 을
다시 x에 할당

변수와 자료형

string (문자열 자료형) – ' 혹은 "를 사용해 생성

```
"Python Session"  
'Python Session'
```

문자열에 ' 또는 "를 포함하려면

```
"Python' Session"  
'Python" Session'
```

문자열 생성법에는 2가지 선택지가 있으므로, 다른 것을 이용함.

여러 줄에 걸쳐 문자열을 만들려면

```
'''  
Python Session  
i like Python  
'''  
  
"""Python Sessionni like Python"""
```

' 혹은 "를 세 번 사용해 괄호를 만듦.

자료형 변환: **str()**

```
>>> type(str(123))  
<class 'str'>  
>>> str(123)  
'123'
```

* type() 함수:
특정 자료의 자료형
을 알려주는 함수

int 자료형인 123을 string 자료형인 '123'으로 바꿈.

변수와 자료형

이스케이프 코드

I say "I" 라는 문자열을 만들기 위해서는, 해당 문자열을 작은 따옴표로 둘러싸야 했다.
다른 방법은 없을까?

```
>>> a = 'I say \'"\'I\'"'
>>> a
'I say \'"\'I\''
```

`\'`를 쓰면 된다. `\'`는 문자열 출력 시 `'`로 출력되기 때문이다.

이와 같이 특정 기능을 갖는 `\`(역슬래시)로 시작하는 문자들을 이스케이프 코드라 지칭한다.

<code>\n</code> : 개행 (줄바꿈)	<code>\v</code> : 수직 탭	<code>\t</code> : 수평 탭
<code>\r</code> : 캐리지 리턴	<code>\f</code> : 폼 피드	<code>\a</code> : 벨 소리
<code>\b</code> : 백 스페이스	<code>\000</code> : 널문자	<code>\\</code> : 문자 <code>"\"</code>
<code>\'</code> : 단일 인용부호(<code>'</code>)	<code>\"</code> : 이중 인용부호(<code>"</code>)	

`\n`, `\\`, `\'`, `\"` 정도만 알면 됨!

변수와 자료형

문자열 연산

```
>>> "Py" + "thon"
'Python'
>>> "---" * 20
'-----'
```

'+'를 하면 두 개의 문자열을 합침
'* n'을 하면 문자열을 n번 더함

콜론의 앞 또는 뒤를 비우면
극단까지 포함된다.

```
>>> a[:]
'I like Python'
>>> a[2:]
'like Python'
```

인덱싱: 문자열의 특정 문자를 선택
슬라이싱: ':'를 사용해 문자열의 부분을 선택

```
>>> a = "I like Python"
>>> a[0] 문자열의 0번째 문자(I)를 선택
'I'
>>> a[5] 문자열의 5번째 문자(e)를 선택
'e'
>>> a[0:5] 문자열의 0번째부터 4번째
          * 띄워쓰기도 1개 문자로 셈
          * 마지막 인덱스는 포함하지
          * 않고 직전 인덱스까지만 포함
'I lik'
>>> a[1:6] 문자열의 1번째부터 5번째
          * 마지막 인덱스는 포함하지
          * 않고 직전 인덱스까지만 포함
' like'
>>> a[-1] 문자열의 뒤에서 1번째 문자
          * 뒤에서 셀 때는 -1부터 셈
'n'
```

인덱싱

* 인덱스는 0부터 세는 것에 주의
즉, 0은 1번째 문자를 가리킴

슬라이싱

인덱싱

변수와 자료형

메서드 개념은 후술되므로,
일단 이런 기능이 있다는 점만 속지!

문자열 관련 주요 메서드

replace(A, B): 문자열 안에서 A를 찾아 모두 B로 바꿈.

split(A): A를 기준으로 문자열을 나누어 리스트를 반환함.

*빈칸일 경우에는 띄워쓰기를 기준으로 나눔.

```
>>> a = "I like Python"
>>> a.replace("like", "love")
'I love Python'
>>> a.split()
['I', 'like', 'Python']
>>> a.split('e')
['I lik', ' Python']
```

문자열 포매팅 메서드

```
>>> a = "딱다구리"
>>> print("{0}는 너무 시끄러워요".format(a))
딱다구리는 너무 시끄러워요
>>> b = "개구리"
>>> print("{0}, {1}는 너무 시끄러워요".format(a, b))
딱다구리, 개구리는 너무 시끄러워요
```

→ {0}, {1} 자리를 format 괄호 안의 원소들(a, b)로 대체한 문자열을 만들어 줌.

※ print(): 문자열을 출력해주는 함수

변수와 자료형

리스트 자료형: 여러 원소들을 저장하는 일종의 배열. 대괄호로 묶여서 표현되며 각 원소는 콤마로 구분됨.

```
_python = [1,2,3,4,5,"PYPY", ["ABC", (123, 111)]]
```

문자열과의 비교

```
>>> _python[0]
```

```
1
```

```
>>> _python[5]
```

```
'PYPY'
```

```
>>> _python[6][0]
```

```
'ABC'
```

문자열과 마찬가지로 0부터 인덱스를 매기며,
인덱싱/슬라이싱은 문자열과 동일.

단, 문자열은 인덱싱을 하여 수정할 수 없으나
리스트는 가능함.

리스트 안에 또 다른 리스트(혹은 iterable)가
있을 경우의 인덱싱은 이렇게.

```
>>> a = [1,2,3]
```

```
>>> b = [4,5,6]
```

```
>>> a+b
```

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

문자열과 마찬가지로 덧셈, 곱셈 연산이
가능하며 기능도 동일함.

```
>>> a = ["abc", "bcf"]
```

```
>>> a * 3
```

```
['abc', 'bcf', 'abc', 'bcf', 'abc', 'bcf']
```

`list()` 함수: 자료를 리스트로 변환해줌

가령, 문자열에 적용하면 문자들을 원소로 갖는 리스트가 반환됨.

변수와 자료형

리스트 수정/ 삭제

```
>>> a = [1, 2, 3, 4, 5]
>>> a[0] = 10
>>> a
[10, 2, 3, 4, 5]
```

◀ 인덱싱을 사용하여 리스트의 원소를 교체할 수 있음

변수 제거 시에는 **del** 변수명

del 예약어를 이용하면 리스트의 원소 또는 전체 리스트를 삭제할 수 있음 ▶

```
>>> del a[0]
>>> a
[2, 3, 4, 5]
```

리스트 관련 주요 메서드

```
>>> a = [1, 2, 3, 4, 5]
>>> a.append([6, 7])
>>> a
[1, 2, 3, 4, 5, [6, 7]]
```

append()

리스트의 새로운 원소를 추가

```
>>> a = [1, 2, 3, 4, 5]
>>> a.extend([6, 7])
>>> a
[1, 2, 3, 4, 5, 6, 7]
```

extend()

리스트에 리스트를 이어붙임

```
>>> a
[1, 2, 3, 4, 5, 6, 7]
>>> len(a)
7
```

len()

리스트의 길이를 반환

```
>>> a = [4, 2, 1, 3]
>>> a.sort()
>>> a
[1, 2, 3, 4]
```

sort()

리스트를 정렬

변수와 자료형

튜플 자료형: 리스트와 동일한 기능을 수행하나, 리스트와 달리 소괄호로 묶이며 변경이 불가함

```
>>> _tuple = (1,2,3)
>>> _tuple = (1,)
>>> _tuple = 1,2,3
```

소괄호를 사용하지 않더라도 튜플 데이터 생성이 가능함

인덱싱/슬라이싱 활용도 리스트와 동일

변수와 자료형

딕셔너리 자료형 {key1:value1, key2:value2, key3:value3}

파이썬의 특징적인 자료형.

리스트/튜플에서 "인덱스"가 원소들을 구분하는 이름의 역할을 했다면, 딕셔너리는 "키"가 그 역할을 대신함.

```
>>> _dictionary = {"a":1, "b":2, "c":3}
>>> _dictionary['b']
2
```

← 중괄호 안에 key값을 입력하면 매칭되는 value를 얻음.

```
>>> _dictionary["d"] = 4
>>> _dictionary
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
>>> del _dictionary["a"]
>>> _dictionary
{'b': 2, 'c': 3, 'd': 4}
```

← 딕셔너리에 데이터를 추가할 때는 별도의 메서드 없이, 이와 같이 새로운 key값에 대해 value를 할당하면 된다.

← 삭제는 리스트의 경우와 유사하다.

변수와 자료형

딕셔너리 관련 메서드

```
>>> "c" in _dictionary
True
>>> "f" in _dictionary
False
```

`in` 예약어를 사용하면 `key`값 존재 여부를 알 수 있다.

```
In [4]: dictionary.keys()
Out[4]: dict_keys(['b', 'c', 'd'])

In [5]: dictionary.values()
Out[5]: dict_values([2, 3, 4])

In [6]: dictionary.items()
Out[6]: dict_items([('b', 2), ('c', 3), ('d', 4)])
```

`keys()`, `values()`, `items()` 메서드 사용 시,
각각 해당 딕셔너리의 `key`값, `value`값, (`key`, `value`) 튜플의 목록
을 얻을 수 있다.

변수와 자료형

집합(set) 자료형

딕셔너리와 마찬가지로 중괄호를 이용해 생성하나, key, 콜론(:) 없이 value들만 입력됨.

```
In [7]: set = {1,2,3}
In [8]: set
Out[8]: {1, 2, 3}
```

```
>>> set_1 = set([1,2,3])
>>> set_2 = set([1,5,6])
>>> set_1 & set_2
{1}
>>> set_1 | set_2
{1, 2, 3, 5, 6}
>>> set_1 - set_2
{2, 3}
```

```
In [9]: set = {1,2,3,3}
In [10]: set
Out[10]: {1, 2, 3}
```

집합 자료형에서는 중복이 허용되지 않으며, 순서도 없음(즉, 인덱스가 없음) ▲

◀ &, |, -를 이용해 교집합, 합집합, 차집합을 구할 수 있음.

변수와 자료형

Bool 자료형: True 혹은 False만을 값으로 갖는 자료형

```
In [12]: a = True
In [13]: a
Out[13]: True
In [14]: type(a)
Out[14]: bool
```

모든 자료형은 Bool값과 호환(혹은 bool() 함수를 사용해 Bool값으로 변환)될 수 있다.

각 자료형에서 無를 의미하는 것(가령 int 자료형 0, string 자료형 "")은 False로 호환되며, 나머지는 True로 호환된다.

※ None: NULL 혹은 “없음” 을 의미하는 객체

```
>>> bool("python")
True
>>> bool("")
False
>>> bool([1,2,3])
True
>>> bool([])
False
>>> bool(1)
True
>>> bool(0)
False
>>> bool(None)
False
```


제어문

비교 연산자

$x < y \leftrightarrow x$ 가 y 보다 작음

$x > y \leftrightarrow x$ 가 y 보다 큼

$x \leq y \leftrightarrow x$ 가 y 보다 작거나 같음

$x \geq y \leftrightarrow x$ 가 y 보다 크거나 같음

$x == y \leftrightarrow x$ 와 y 가 같음

$x != y \leftrightarrow x$ 와 y 가 같지 않음

논리 연산자

$x \text{ and } y \leftrightarrow x$ 와 y 가 모두 참이면 참

$x \text{ or } y \leftrightarrow x$ 와 y 둘 중 하나만 참이면 참

$\text{not } x \leftrightarrow$ 논리 상태를 반전시킴

멤버 연산자

$x \text{ in } y \leftrightarrow y$ 에 x 가 포함되어 있으면 참

$x \text{ not in } y \leftrightarrow y$ 에 포함되어 있지 않으면 참

→ 각 연산자들이 참일 조건을 만족하면 True(bool 자료형)를 반환하고, 아니면 False를 반환

제어문

조건문(if)

```
>>> signal = True
>>> if signal == True:
...     print("signal is True")
... else:
...     print("signal is not True")
...
signal is True
```

이 경우에는 `signal == True`가 `True`를 반환하므로, `if`문의 코드가 실행됨

`if` 뒤의 값(`signal == True`)가 `True`일 경우
아래 블록의 코드가 실행됨
조건문이 `False`일 경우에는 `else:` 이하의
블록이 실행됨

※ 조건문 사용 시 주의사항

`if` 문에 속하는 실행문은 들여쓰기를 해야 함
(통상 `tab` 또는 `space 4번`으로 들여쓰기 함)

조건 뒤에 콜론(:) 이 필요함

`else`는 꼭 있을 필요는 없음

`if` 문을 중첩하여 사용할 수도 있음

제어문

elif문

```
>>> if count == 1:
...     print("count is 1")
... elif count == 2:
...     print("count is 2")
... else:
...     print("count is not 1 or 2")
...
count is 2
```

elif문은 if문이 거짓일 경우, 다음으로 실행되는 조건문임.
elif문은 여러 번 사용될 수 있음.

다양한 조건문 예시

```
>>> if True:
...     print("That's True!")
...
That's True!
```

if 뒤에 꼭 비교문만 올 수 있는 것은 아님.
True/False를 반환하는 코드라면 무엇이든 가능함.

```
>>> if 1:
...     print("It's also True!")
...
It's also True!
```

가령 int 자료형 1은 bool 자료형 True와 대응되므로,
이 조건문은 참으로 인식되어 실행된다.

제어문

조건부 표현식

조건문이_참인_경우 **if** 조건문 **else** 조건문이_거짓인_경우

위와 같은 양식의 조건부 표현식을 사용하면, 전체 조건문을 1줄로 축약할 수 있다.

e.g.

```
signal = True
```

```
if signal == True:
```

```
    print("Hi")
```

```
else:
```

```
    print("Bye")
```



```
print("Hi") if signal else print("Bye")
```

제어문

반복문 1. while문

while (조건문):
(실행될 코드)

조건문이 True인 동안
아래 코드가 계속 실행됨

e.g.

```
>>> while count < 10:  
...     count += 1  
...     print("count: {}".format(count))  
...  
count: 1  
count: 2  
count: 3  
count: 4  
count: 5  
count: 6  
count: 7  
count: 8  
count: 9  
count: 10
```

◀ 조건문 양식과
들여쓰기하는 것은
if문과 동일함

※ 무한 루프에 걸려 빠져나와야
할 때에는 Ctrl+C

break:
반복문을 빠져나감

흐름 제어를 위한
예약어들

continue:
반복문의 처음,
조건문으로 돌아감

```
>>> count = 0  
>>> while count < 10:  
...     count += 1  
...     print("count: {}".format(count))  
...     if count == 2:  
...         break  
...  
count: 1  
count: 2
```

```
>>> count = 0  
>>> while count < 10:  
...     count += 1  
...     if count % 3 != 0:  
...         continue  
...     print("count: {}".format(count))  
...  
count: 3  
count: 6  
count: 9
```

제어문

반복문 2. for문

반복가능 자료형(iterable):
리스트처럼 여러 데이터를 반환할 수 있는 자료형

for 원소의 이름 in 반복가능 자료형:
(실행될 코드)

이 이름은 for문 안에서 iterable이 반환하는 데이터를 호출하기 위해 사용되는 것으로, 임의로 지정하면 됨.

- while에서 반복 흐름 제어에 사용된 break, continue는 for에서도 동일한 기능으로 사용됨.
- 실행될 코드가 들여쓰기 되는 것도 동일함.

```
>>> _list = ["종윤", "상연", "건우"]
>>> for person in _list:
...     print(person)
...
종윤
상연
건우
```

◀ 이 반복문은 _list를 베이스로 실행되며, _list가 반환하는 자료가 3개인 바, 안의 코드가 3번 반복됨.

1번째 반복: person = "종윤"
2번째 반복: person = "상연"
3번째 반복: person = "건우"
로 동일한 코드가 3번 실행됨.

제어문

range(): 연속적인 값을 반환하는 자료를 생성

range(5): 0, 1, 2, 3, 4를 순차로 반환
(0부터) 5미만까지의 정수

range(2,6): 2, 3, 4, 5를 순차로 반환
2부터 6미만까지의 정수

range(1,6,2): 1, 3, 5를 순차로 반환
1부터 6미만까지 2씩 증가

* 리스트 슬라이싱과 마찬가지로 끝 숫자는 포함되지 않음

```
>>> a=range(5)
>>> a
range(0, 5)
>>> list(a)
[0, 1, 2, 3, 4]
```

range 함수를 이용해 iterable인 a을 만든 뒤, 내용물을 확인해보려 하면 0, 1, 2, 3, 4가 나타나지 않고 range(0, 5)가 나타남.
이 경우 list()를 이용해 a를 list로 바꿔주면 내용물들을 볼 수 있다.

```
>>> for i in range(0,5):
...     print(i)
...
0
1
2
3
4
```

제어문

리스트 내포

0부터 9까지 정수를 원소로 갖는 리스트를 만들고 싶다면?

```
temp = []  
for i in range(10):  
    temp.append(i)
```



```
temp = [i for i in range(10)]
```

리스트 내포 사용 시 1줄로 축약 가능!

반복문과 리스트 메서드 append를 활용

응용

```
temp = [i for i in range(10) if i%2 == 0]  
print(temp)
```

i가 짝수일 때만 list에 추가

[0,2,4,6,8]

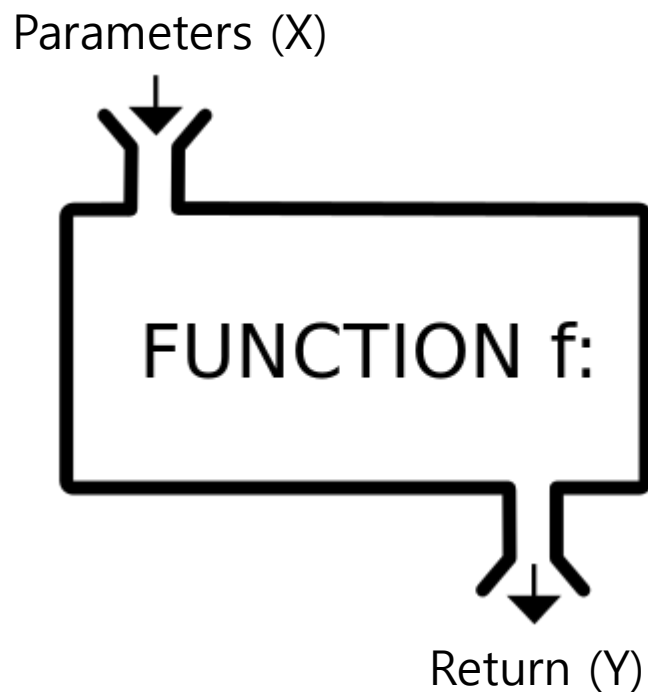
```
temp = [(i, j) for i in range(10) for j in range(10) if i%2 == 0]
```

반복문을 중첩해 사용할 수 있음.

```
for i in range(10):  
    for j in range(10):  
        if i%2 == 0:  
            temp.append((i,j))
```

← 이것과 대응됨

함수



함수: 인자(parameter)를 전달받아 코드로 짜여진 규칙을 수행해 연산된 결과값을 반환(return)하는 객체

e. g.

전달받을 인자가 2개이며, 함수 코드 내부에서 해당 인자들을 각각 a, b로 명명

```
def cal(a,b):
```

```
    return a * b
```

a*b를 반환

cal(2,3) 위에서 정의한 함수의 이름 cal과 소괄호에 2개의 인자(2와 3)를 넣어서 코드를 실행하면 2, 3이 인자로 전달된 채 cal 함수가 "호출"된다.
결과값: 6

함수

함수의 기본적인 포맷

`def` 함수명(입력 인자의 이름(들)):
(실행될 코드)
`return` 반환하고자 하는 값

함수를 정의하기 위한 코드도
꼭 들어쓰기가 되어야 한다.

1. 인자, 반환 값이 존재 (기본)

```
def cal(a,b):  
    return a * b
```

2. 인자만 존재

```
def cal(a,b):  
    print(a * b)
```

3. 반환 값만 존재

```
def cal():  
    return 777
```

4. 둘 다 존재하지 않음

```
def cal():  
    print(777)  
    return
```

목적에 맞게 인자 혹은 반환을 생략할 수 있다.



전자의 경우 함수의 이름 뒤의 소괄호를 비워두어야 하며
후자의 경우 return문을 작성하지 않거나, return을 쓰되 뒤에
반환값을 쓰지 않으면 된다.

함수

Keyword Parameter: 함수 정의에서 사전에 값이 할당되어 있는 parameter

```
>>> def twice(k):  
...     return k*2
```



```
>>> def twice(k=2):  
...     return k*2
```

Keyword parameter는 뒤에 오도록 해야 함

def fun(a, b, c=1) (O)

def fun(a, b=1, c) (X)

```
>>> twice()  
4
```

Keyword parameter를 별도로 지정하지 않고 함수를 호출하면 사전에 지정된 값을 기준으로 값이 반환되며,

```
>>> twice(k=4)  
8
```

인자를 바꾸기 위해서는 '인자 이름=값'의 형태로 함수를 호출한다.

함수

Return 활용

여러 값을 반환하고 싶으면, 콤마(,)를 이용하면 된다. 이 경우 여러 값이 묶인 튜플이 반환된다.

```
def mul_return():  
    return 1,2,3
```

```
print(mul_return())  
출력 값: (1, 2, 3)
```

또한 함수에서 return만 쓰일 수 있는데, 이는 return None, 즉 아무 것도 반환하지 않는 것과 같다. 이러한 용법은 반복문의 break처럼 함수를 빠져나가기 위한 용도로 사용될 수 있다. 모든 함수는 return이 실행되는 순간 종료되기 때문이다.

```
def 함수명(인자1=..., 인자2=...):  
    (실행1)  
    return
```

함수

재귀 함수

함수 안에서 함수 자신을 다시 호출하는 함수 (주로 return문에서 다시 호출되는 형태)

e. g. $n!$ 을 계산하는 함수

```
def factorial(n):  
    if n == 1:  
        return 1    함수 호출이 무한 루프에 빠지지 않도록, 재귀 호출이 멈추는 지점 필요  
    else:  
        return n * factorial(n-1)    반환문에서 함수 자신이 다시 호출되고 있음
```

가령, factorial(5)가 호출되었다고 하자.

$5 \neq 1$ 이므로 $5 * \text{factorial}(4)$ 가 반환되어야 하며, 이를 위해서는 factorial(4)가 호출되어야 한다. factorial(4)는 $4 * \text{factorial}(3)$ 을, factorial(3)은 $3 * \text{factorial}(2)$ 를, factorial(2)는 $2 * \text{factorial}(1)$ 을 순차적으로 반환해야 하게 된다.

factorial(1)에 이르면 $n == 1$ 이 성립하므로 1이 반환되며 더 이상의 함수는 호출되지 않는다.

그 결과 factorial(2)는 $2*1$ 을 반환할 수 있게 되고, factorial(3)은 $3*2*1$ 을, factorial(4)는 $4*3*2*1$ 을, factorial(5)는 $5*4*3*2*1$ 을 순차적으로 반환할 수 있게 된다.

재귀 함수를 구성할 때에는 반드시 언젠가는 함수의 재귀적 호출이 멈출 수 있도록 장치를 마련해 두어야 한다! (위 예시에서의 return 1과 같이)

함수

함수 내부 변수

```
>>> a = 0
>>> def test(a):
...     for i in range(5):
...         a += 1
...         print(a)
...
>>> test(a)
5
>>> print(a)
0
```

다름!

함수 내에서 a는 5이지만,
함수 바깥의 a는 여전히 0!

함수 내에서 인자의 이름이 'a'로 설정되면서 생긴 변수 a는,
함수 바깥에서 정의된 변수 a와는 독립적이다.

함수

각종 유용한 내장함수들

1. enumerate(): iterable을 인자로 받아 인덱스와 원소의 쌍을 리턴

```
for i, value in enumerate([종윤, 상연, 건우]):  
    print(i, value)
```

출력값: 0 종윤
1 상연
2 건우

2. lambda(): 간결하게 함수를 정의할 때 사용

```
mul = lambda 인자 x,y: 반환 값 x*y  
print(mul(5,5))
```

출력값: 25

3. zip(): 같은 개수를 지닌 자료형을 인자로 받아 차례대로 묶어줌

```
print(list(zip([1,2,3], [4,5,6])))
```

출력값: [(1, 4), (2, 5), (3, 6)]

클래스

클래스: 공통된 목적으로 사용하기 위한 변수와 함수(메소드)를 하나의 틀로 묶은 것
클래스를 정의하는 것은 사실상 새로운 자료형을 만드는 것과 같음

클래스 정의의 기본 틀

`class` 클래스명:
(실행 코드)

`def` 매서드1():
(실행 코드)

※ 매서드: 클래스 안에서 정의되는 함수

`def` 매서드2():
(실행 코드)

클래스

클래스의 예시

```
class data_analysis:
```

```
    blank = []
```

data_analysis 클래스 안에 blank라는 빈 리스트를 만듦

```
    def mean(self, a,b,c):
```

```
        _mean = (a+b+c)/3
```

```
    return _mean
```

data_analysis 클래스 안에 세 수의 평균을 반환하는 매서드를 만듦

매서드를 정의할 때에는 self가 인자로 항상 포함되어야 함.
이는 매서드를 호출함에 있어 클래스(정확히는 인스턴스) 자신이 인자로 전달됨을 의미.

test = data_analysis() 만들어진 클래스의 객체를 생성하는 방법은 함수 호출과 같음.

test.mean(1,2,3) → 2.0 모니터에 출력 매서드는 클래스 객체 뒤에 '.매서드(인자)'를 작성하면 호출되며,

print(test.blank) → [] 모니터에 출력 매서드 정의 시 self, a, b, c 총 4개의 인자가 있었으나, 실제 호출에는 a, b, c만 입력.

클래스 객체 내의 데이터를 불러 올 때에는 매서드와 비슷하게 '.자료명'을 쓰면 됨.
단, 매서드와 달리 인자가 없으므로 소괄호도 없음.

클래스

클래스와 객체



객체: 클래스에 의해 만들어진 실체

인스턴스: 객체를 클래스와의 관계로 부를 때 사용

test = data_analysis()

data_analysis 클래스의 인스턴스이자,
data_analysis 클래스에 의해 생성된 하나의 객체

인스턴스가 바로 매서드의 인자로 전달되는 self의 정체

print(type(123)) → <class 'int'>

123은 'int' 클래스에 속한 인스턴스

**우리가 알고 있던 모든 자료형 그리고 자료들은
사실 클래스와 객체였던 것!**

클래스

`__init__()`: 생성자, 인스턴스를 만들 때 자동으로 실행되는 매서드

class data_analysis: 인스턴스를 만들기 위해서는 3개의 인자가 필요하게 됨

def __init__(self, a, b, c):

self.변수명: 인스턴스 변수,
클래스 정의 코드 내에서 다른
매서드에서 참조될 수 있는 변수

self.a = a
self.b = b
self.c = c

blank = []

def mean(self):

_mean = (self.a+self.b+self.c)/3

return _mean

```
>>> test = data_analysis() 인자 3개를 주지 않으면 에러 발생
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: __init__() missing 3 required positional a
>>>
>>> test = data_analysis(2,3,4)
```

클래스

클래스 상속

기존 클래스의 변수와 매서드를 가져와 원하는 부분만 수정해 클래스를 만들 수 있음
만들고자 하는 클래스와 비슷한 클래스를 상속하여 효율적으로 코딩할 수 있음

형식: `class 클래스명(상속받는 클래스명):`
(코드)

매서드 오버라이딩

상속한 클래스의 매서드를 덮어 쓰는 것

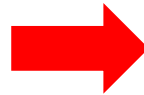
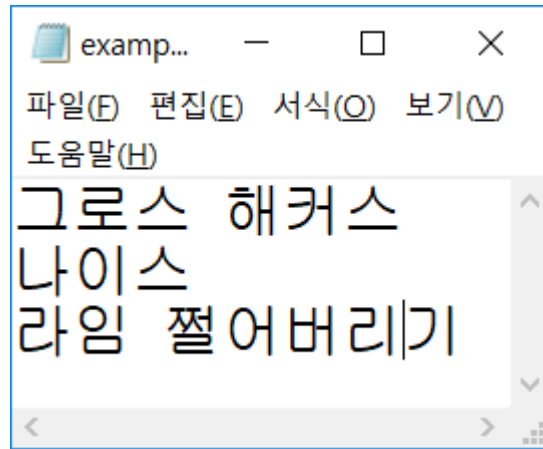
e. g.

```
class _inheritance(data_analysis):  
    def mean(self):  
        _mean = (self.a+self.b+self.c) // 3  
        return _mean
```

그냥 원 클래스에서의 것과 동일한 이름의 매서드를 새로 정의하면 됨.

파일 입출력

`open("파일 경로", "파일 열기 모드")`: 파일을 열 때 사용되는 내장함수



파일 경로를 쓸 때에 'w'는 'ww'로 표기해주어야 하는데, 이는 파이썬은 'ww'을 'w' 하나로 인식하기 때문이다.

```
>>> f = open('C:\\\\ex\\\\example.txt', 'r')
>>> f.readline()
'그로스 해커스\n' 'wn'은 줄 바꿈을 의미한다.
>>> f.readline()
'나이스\n'
>>> f.readline()
'라임 찢어버리기'
>>> f.close()
```

작업을 마친 뒤에는 `close()`로 파일을 항상 닫아주어야 함!

`readline()` 메서드: 열어둔 파일 객체에 사용하는 메서드로, 한 줄의 텍스트를 읽어 반환함.

`readlines()` 메서드: 위와 달리 모든 줄의 텍스트를 한 번에 읽어 리스트로 반환함.

```
>>> f.readlines()
['그로스 해커스\n', '나이스\n', '라임 찢어버리기']
```

파일 입출력

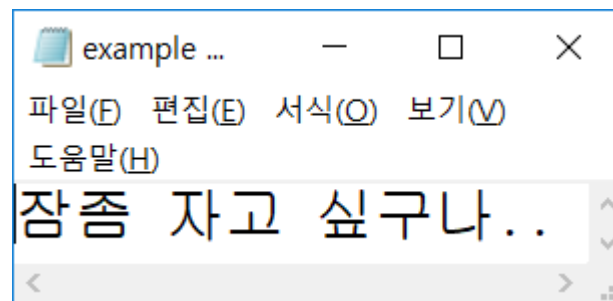
파일 열기 모드의 종류

“r”: 파일을 읽을 때 사용

“w”: 파일에 내용을 (새로) 쓸 때 사용

“a”: 파일의 마지막에 내용을 추가할 때 사용

```
>>> f = open('C:\\\\ex\\\\example.txt', 'w')
>>> f.write('잠 좀 자고 싶구나..')
11
>>> f.close()
```



write() 메서드: 파일에 내용을 입력해줌과 더불어, 입력한 텍스트의 글자 수를 반환함.

파일 입출력

with문

with문을 사용하면 close()를 할 필요가 없음

들여쓰기로 파일이 open되어 있는 구간(블록)이 설정되며,
해당 구간의 코드가 끝나면 파일은 자동으로 닫힘.

코드 예시

```
with open("C:\Users\WW\Ex.txt", 'r') as f:  
    lines = f.readlines()
```

모듈

모듈: 다른 파이썬 코드에서 사용할 수 있도록 클래스, 함수, 변수 등을 포함하고 있는 파일
라이브러리라고도 불리며, 이미 구현된 코드들을 가져와서 사용할 수 있어 작업의 효율성을 높여 줌

모듈 불러오기

- **import 모듈 명**: 모듈 내의 함수 / 객체 호출 시 '모듈 명.객체명'

```
>>> import pandas
>>> df = pandas.read_csv("test.csv")
```

pandas 라이브러리에 대해서는
다음 세션에서 다루겠습니다!

- **import 모듈 명 as 약칭**: 모듈 내의 함수 / 객체 호출 시 '약칭.객체명'

```
>>> import pandas as pd
>>> df = pd.read_csv("test.csv")
```

- **from 모듈 명 import 객체 또는 함수**: 모듈 내의 특정 객체 또는 함수만을 import하며, 해당 객체 호출 시에는 곧바로 '객체명'을 입력하면 됨

```
>>> from pandas import read_csv
>>> df = read_csv("test.csv")
```

앞으로 진행될 세션들에서 다양한 머신러닝 알고리즘 구현을 위한 모듈들에 대해서 학습할 예정!

예외 처리

try, except 구문

try 블록을 실행 중 에러가 발생하면 except 블록의 코드를 실행함.

에러를 직접 핸들링하거나 에러가 발생해도 코드 실행을 계속하고 싶을 때 주로 사용됨.

try:

(실행1)

(실행2)

* 들여쓰기에 신경 써야함

e. g.

except:

(실행1')

(실행2')

```
>>> try:
...     f = open('C:\#ex\#example.txt', r)
... except:
...     print('읽기 모드로 하고 싶으면 r이 아니라 "r"을 써야 합니다.')
읽기 모드로 하고 싶으면 r이 아니라 "r"을 써야 합니다.
```

예외 처리

특정 오류에 대해서만 예외처리를 하고 싶은 경우

try:

(실행1)

(실행2)

except 오류 이름:

(실행1')

(실행2')

```
>>> try:
...     f = open('C:###what.txt', r)
... except NameError:
...     print('으음..')
...
으음..
```

기타 코드 작성 규칙

```
95 # 스팸일 확률을 오름차순으로 정렬
96 classified.sort(key=lambda row: row[2])
97
98 # 스팸이 아닌 메시지 중에서 스팸일 확률이 가장 높은 메시지
99 spammiest_hams = list(filter(lambda row: not row[1], classified))[-5:]
100
101 # 스팸 중에서 스팸일 확률이 가장 낮은 메시지
102 hammiest_spams = list(filter(lambda row: row[1], classified))[:5]
```

에디터 내에서 #을 붙여 코드를 작성하면,
해당 줄은 코드 실행 시 무시됨.

이를 **주석**이라 하며,
코드에 대한 코멘트를 남길 때 주로 사용됨.

```
>>> 1 + 2 + 3 + #
...    4 + 5 + 6
21
```

한 줄의 코드를 여러 줄에 걸쳐 작성하고
싶으면 w(역슬래시)를 이용한다.

Quest (코드파일 또는 캡처한 실행결과를 깃헙에 PULL REQUEST)

1. <https://programmers.co.kr/learn/courses/30/lessons/12907>

위 링크의 문제를 풀어주세요.

2. 짝수인 원소에 대응하는 인덱스의 합을 리턴하는 함수 f4()를 만들어주세요. 단, 제어문(for, while, if)을 사용하지 마세요!

Hint : map, lambda, filter, enumerate

```
>>>f4([1,2,3,4])
```

```
2
```

```
>>>f4([1,2,3,4,5])
```

```
6
```

Quest (코드파일 또는 캡처한 실행결과를 깃헙에 PULL REQUEST)

3. diamonds_data.csv 의 열을 뒤집은 파일을 생성하는 코드를 만들어주세요.

(carat, depth, table, price, x, y, z) => (z, y, x, price, table, depth, carat)

- 경로나 파일 명이 틀리면 "wrong file path"를 출력하는 예외처리를 해주세요.
- 출력 파일의 확장자 명은 csv로 지정해 주세요.
- 열을 뒤집은 예시는 다음과 같습니다.

	A	B	C	D	E	F	G	H	I	J	K
1	z	y	x	price	table	depth	carat				
2	2.43	3.98	3.95	326	55	61.5	0.23	1			
3	2.31	3.84	3.89	326	61	59.8	0.21	2			
4	2.31	4.07	4.05	327	65	56.9	0.23	3			
5	2.63	4.23	4.2	334	58	62.4	0.29	4			
6	2.75	4.35	4.34	335	58	63.3	0.31	5			
7	2.48	3.96	3.94	336	57	62.8	0.24	6			

참고 자료

작성에 참고한 자료

- 박응용, 점프 투 파이썬, 위키독스
- 이영준·고병욱(GH 2기), 파이썬 세션 자료
- <https://blog.naver.com/hemahero/221209920743>

추천 보충/참고 자료

- Jump2python (<https://wikidocs.net/book/1>)
- Byte of python (http://byteofpython-korean.sourceforge.net/byte_of_python.html)
- 기초 PYTHON 프로그래밍 서강대 인터넷 강의 (https://www.edwith.org/sogang_python)

본 세션 학습에 어려움이 있으시다면 언제든지 문의 주세요 ☺

내용 문의: 김현세 / QUEST 문의: 배민영