180604

파이썬 설치

https://www.python.org/downloads/windows/

다운로드 옆 Windows 누르면 bit별 설치할 수 있게 구분되어있음!

2.x / 3.x 문법 다름!! 지금 최신 버전은 3.6.5

+ Add Python 3.6 to PATH 체크 하고 바로 설치!

배열 연산 ~ 넘파이(like 패키지 in R) , 판다스 (like 데이터프레임 in R)

for 분석용, 데이터 핸들링 -> 아나콘다 사용 (주피터 + ?) ~ 64bit (다시 다운)

Chapter 2. 변수의 사용 - 함수(in R) -> method(in python)

쓰는 형태가 달라짐. method 형식으로 정의된 함수 형식 생소 => 변수 이름에 . 사용 불가!

print - print () : 함수의 서식

1. **"출력형식(서식)" % (값을 전달할 대상)** 형태로 사용!
2. **"순서+출력형식(서식)".format(값을 전달할 대상)** 형태로 사용!

1. print() 함수에서 사용할 수 있는 서식

%d : 정수타입

%f : 소수점(실수)

%c : 한 글자

%s : 두 글자 이상인 문자열

예시

```
>>> print("%d / %d = %d" % (100, 200, 0.5))
100 / 200 = 0                    # 입력한 값은 0.5인데 0으로 출력됨 (%d는 정수 표현이므로)
```

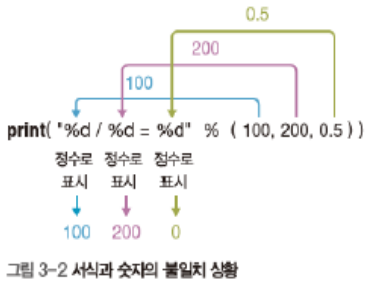


그림 3-2 서식과 숫자의 불일치 상황

```
>>> print("%d / %d = %5.1f" % (100, 200, 0.5)) # 따라서 다음과 같이 수정
100 / 200 =   0.5
```

서식 관련 예시

```
>>> print("%d" % 123)
123
>>> print("%5d" % 123)    # %5d : 정수 5자리로 표현, 나머지 빈 자리 수만큼 공백으로 채우기
  123
>>> print("%05d" % 123)   # %05d : 정수 5자리로 표현, 나머지만큼 0으로 표시하기
00123

>>> print("%f" % 123.45)   # %f : default 자리수 굉장히 많음, 따라서 자리수 지정 필요
123.450000
>>> print("%7.1f" % 123.45)        #%7.1f    - 7 : .포함 전체 자리수, 나머지 표현 공백
123.5                                          - .1 중 소수점 자리 수를 1개만 가져오기
>>> print("%07.1f" % 123.45)  # %07.1  .포함 수=7, 소수점 자리=1, 7자리 중 빈 자리는 모두 0
00123.5
```

```
>>> print("%7.3f" % 123.45) # %7.3f  .포함 7자리 수 채우되 소수점자리는 3자리, 나머지는 공백
123.450

>>> print("%s" % "Python")      # %s : 광범위한 문자 표현
Python
>>> print("%10s" % "Python")    #%10s : 열자리 문자열, 나머지는 공백
      Python
```

print() 함수를 사용한 다양한 출력 예시 - "출력형식(서식)" % (값 전달할 대상) 형태

```
>>> print("안녕하세요?")
안녕하세요?

>>> print("100")                # " " 안에 들어가면 문자 형태로 출력!
100

>>> print("%d" % 100)           # %d - 정수형태 => 숫자로 출력
100

>>> print("100+100")      # " " 안에 들어가면 문자 형태로 출력!
100+100

>>> print("%d" % (100 + 100))    # 숫자 100 + 숫자 100 = 숫자 200 출력
200

>>> print("%d" % (100, 200))      # %d가 하나밖에 없는데 숫자가 두 개 -> 오류
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in <module>
    print("%d" % (100, 200))
TypeError: not all arguments converted during string formatting

>>> print("%d %d" % (100)) # %d가 두 개인데 숫자가 하나 -> 오류
Traceback (most recent call last):
  File "<pyshell#62>", line 1, in <module>
    print("%d %d" % (100))
TypeError: not enough arguments for format string
```

```
>>> print("%d %d" % (100, 200))  # 올바른 형태로 수정!
100 200
```

그림 3-1 서식과 숫자의 대응

결과 확인

- **1. 형식이 순서대로 적용됨**
- **2. 값 전달할 대상 안에서 산술연산 가능**
- **3. 출력형식 반드시 쌍따옴표로 써줘야 함**

2. format 형태 사용

- { }를 함께 사용
- 값 전달할 대상 순서를 고려해 {0} {1} {2} ... 형태로 출력 순서 지정

"{1:5d} {0:d}".format(값을 전달할 대상1, 대상2))

↑

↑ 1st 순서 - d형식

2nd 순서 - 5d형식

 # 0 (in python) **: 최초를(1st) 의미하는 숫자**

예시

```
>>> print( "{0:d} {1:5d} {2:05d}".format(123, 123, 123))
123   123 00123
```

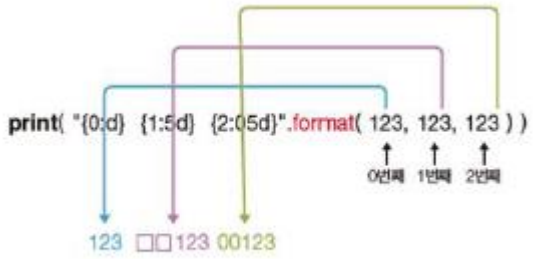


그림 3-6 format() 함수의 사용

이스케이프 문자 **for 특수기호 무력화**

이스케이프 문자	역할(설명)
₩n	새로운 줄로 이동 (Enter 효과) - 개행문자
₩t	다음 탭으로 이동 (Tab 효과)
₩b	뒤로 한 칸 이동 (Backspace 효과)
₩₩	₩ 출력 -역슬래시
₩'	' 출력
₩"	" 출력

예시

```
>>> print("₩n줄바꿈₩n연습")

줄바꿈
연습
>>> print("₩t탭키₩t연습")
        탭키    연습
>>> print("글자가 ₩"강조₩"되는 효과1")
글자가 "강조"되는 효과1
>>> print("글자가 ₩'강조₩'되는 효과2")
글자가 '강조'되는 효과2
>>> print("₩₩₩₩₩₩ 역슬래시 세 개 출력")
₩₩₩ 역슬래시 세 개 출력
>>> print(r"₩n ₩t ₩" ₩₩를 그대로 출력")
₩n ₩t ₩" ₩₩를 그대로 출력
```

#print(**r**"문자열")) 문자열 앞에 r을 넣으면 안에 있는 ₩n, ₩t, ₩b가 실행이 되지 않는 것 같다.

변수

변수란?

- 변수는 데이터를 담는 메모리 공간
- 변수에는 수, 텍스트, 목록, 이미지 데이터 등을 담을 수 있음

변수명 규칙

- 대, 소문자를 구분한다. (myVar와 MyVar는 다른 변수)
- 변수명에 . 쓸 수 있지만 활용 불가능 (for method 사용)
- 문자, 숫자, 언더바(_)를 포함할 수 있다. **하지만 숫자로 시작하면 안 된다.**
- 예약어는 변수명으로 쓰면 안 된다.

(예약어 : True, False, None, and, or, not, break, continue, return, if, else, elif, for, while, except, finally, global, import, try 등)

- 파이썬에서는 = 로 변수 입력

변수의 선언과 사용

-변수에 연속된 값을 대입

실습

In [1]: var1=var2=var3=3

In [2]: var1,var2,var3

Out[2]: (3, 3, 3)

자료의 유형-2진수, 8진수, 16진수

-2진수 접두사 : 0b (Binary number)

-8진수 접두사 : 0o (Octal number)

-16진수 접두사 : 0x (hexadecimal number)

실습

In [6]: A=0b100110

In [7]: B=0o107

In [8]: C=0x157

In [10]: A, B, C

Out[10]: (38, 71, 343)

In [11]: bin(38), oct(71), hex(343) #십진수 -> 각각 진수 형태로 나타내기

Out[11]: ('0b100110', '0107', '0x157')

수 다루기 - 정수

정수?

- 음의 정수, 0, 양의 정수
- 파이썬에서는 메모리가 허용하는 한, 무한대를 정수를 다룰 수 있음

실습

```
>>> d=-153435481351534385468463513513548646 8  # 변수에 음수입력
>>> e=-13513515313515843654312
>>> f=-3

>>> type(f)            # type으로 변수 형식 확인 (like class in R)
<class 'int'>
>>> type(e)
<class 'int'>
>>> type(d)
<class 'int'>
```

파이썬에서 제공하는 사칙 연산자

+ - * /	사칙연산
//	**나눗셈의 몫 구하기**
%	**나눗셈의 나머지 구하기**

수 다루기 - 실수

파이썬에서는 실수를 지원하기 위해 부동 소수형을 제공

- 부동 : 뜰 부, 움직일 동 -> 소수점을 움직여서 소수를 표현하는 자료형

부동 소수형의 특징

- 부동 소수형은 8바이트만을 이용해서 수를 표현한다. 즉, 한정된 범위의 수만 표현할 수 있다.
- 디지털 방식으로 소수를 표현해야 하기 때문에 정밀도의 한계가 있다.

예시

```
>>> b=22/7
>>> b
3.142857142857143
>>> type(b)
<class 'float'>

 # 22/7의 결과는 무리수지만 부동소수형은 소수점 이하 15자리만 표현
```

실수 계산오차

- 실수 저장방식 때문에 작은 오차 발생 가능성 존재
- 이러한 작은 오차는 반올림 등으로 처리

예시

```
In [35]: 0.1+0.1, 0.1+0.2, 3*0.1, 9.2-5.7
Out[35]: (0.2, 0.30000000000000004, 0.30000000000000004, 3.499999999999999)
```

수 다루기 – 복소수

- 파이썬에서는 허수 단위를 나타내는 부호로 i 대신 j 를 사용

예시

```
In [3]: x=2+3j
In [4]: x
Out[4]: (2+3j)
In [5]: x, x.real, x.imag, x.conjugate()     # x, 실수부, 허수부, 켤레복소수
Out[5]: ((2+3j), 2.0, 3.0, (2-3j))
```

수 다루기 - math 모듈을 이용한 계산

math 모듈을 사용하기 위한 import문

>>> import math

- 기본적으로 내장됨! (설치 X)
- 호출시 import 사용
- 모듈 안에서 제공되는 함수 / 모듈 이름 계속 쓰면서 method 사용해야 함 -> alias 가능

π와 e

```
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
```

파이썬 코드에서 **"."**은 "**~의**"로 해석

•math.pi는 math(모듈)의 pi

•math.e는 math(모듈)의 e

절대값, 버림과 반올림

abs() -- 절대값 계산 함수 -- 내장 함수

round() -- 반올림 계산 함수 -- 내장 함수

trunc() -- 버림 계산 함수 -- **math 모듈**

trunc - 기본 제공 함수 아님! (vs R)

예시

```
>>> abs(10)
10
>>> abs(-10)
10

>>> round(1.4)
1
>>> round(1.5)
2

>>> import math
>>> math.trunc(1.4)
1
>>> math.trunc(1.9)
1
```

팩토리얼 - from math 모듈

factorial()

```
>>> math.factorial(0)
1
>>> math.factorial(10)
3628800
>>> math.factorial(43)
604152630633738356373551320685139975072645120000000000
```

```
# 모듈 단어에 익숙해지기
# 사용자 모듈 설정 가능 (원하는 모듈만, 함수만 로딩해서 사용)
```

제곱과 제곱근

** -- 제곱 연산 -- 연산자

pow() -- 제곱 연산 -- **math 모듈**

sqrt() -- 제곱근 연산 -- **math 모듈**

```
>>> 3**3
27

>>> import math
>>> math.pow(3,3)
27.0
>>> math.sqrt(4)
2.0
>>> math.sqrt(100)
10.0

>>> 27**(1/3)
3.0
>>> math.pow(81,0.5)
9.0
```

 # 만일 했을 때 수학적 계산이 안 될 때?

import math 사용 or 계산 시작 전부터 math 모듈 불러오기 습관!

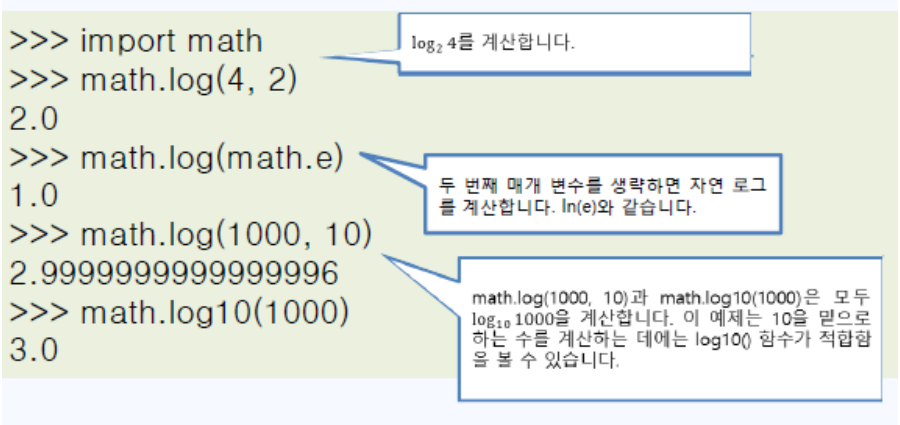
로그, 지수 from math 모듈

log() - 첫 번째 매개변수의 로그를 구함. 두 번째 매개변수는 밑 수, 생략시 자연수e가 밑 수

log10() - 밑수가 10인 로그를 계산함

exp() – e의 x제곱을 구함

예시



cf. 자연수 e를 사용할 때에도 **math.e**로 사용하는 것 잊지 말 것!

기타 함수 in math

ceil(x) - x보다 큰 정수 중 가장 작은 수

floor(x) - x보다 작은 정수 중 가장 큰 수

텍스트 다루기

줄 바꾸기

- 파이썬에서는 텍스트를 다루는 자료형으로 string을 제공
- 파이썬 코드에서는 문자열 데이터를 작은 따옴표 ' ' 또는 큰 따옴표 " "로 텍스트 표현
- 여러 줄로 이루어진 문자열은 작은 따옴표 세 개 (' ' ') 또는, 큰 따옴표 세 개 (" " ") 의 쌍으로 텍스트를 감싸서 표현 –> 여러 줄로 이루어진 텍스트 입력 가능

cf. 따옴표 한 번 쌍으로 쓰려면 ₩n로 다음 줄 설정

예시

```
>>> c='''어서와
파이썬은
처음이지?'''
>>> c
'어서와₩n파이썬은₩n처음이지?'
>>> d="""Welcome to
Python."""
>>> d
'Welcome to₩nPython.'
>>> type(d)
<class 'str'>
```

```
# print를 이용하지 않으면 다음 라인으로 설정되지 않음. 변수에 있는 내용만 보여줌
```

연결연산자 **★★★자주쓰임★★★** + : R과 다른 점 - 연결연산자 기능

- 문자열을 다룰 때 + 연산자는 두 문자열을 하나로 이어 붙임

cf. + 연산자가 문자열을 결합한다고 해서, - 연산자가 문자열을 분리하는 것은 아님

실습

```
>>> hello='Hello'
>>> world='World'
>>> hello_world=hello + ', ' + world
>>> hello_world
'Hello, World'
```

/ for문 이용~ 새로운 값 붙여서 쓸 때 해당되는 값에 + 쓰고 변수 쓰면 계속 사용될 것!

문자열 반복

- 문자열*n 또는 n*문자열
- 문자열 반복 후 대입 : *=n

```
In [32]: S="Hello" ; S+=', Python'
         S
Out[32]: 'Hello, Python'
```

```
In [4]: S*=3
        S
Out[4]: 'Hello, PythonHello, PythonHello, Python'
```

문자열 분리 (slicing)는 [ ] 연산자를 통해 수행함
- 다만 start값과 end값 의미하는 것 R과 다름!

x[a:b]

해석 순서 - 헷갈리기 쉬움 ! 주의 의미 잘 해석하기!!!!!!!!

1. 범위

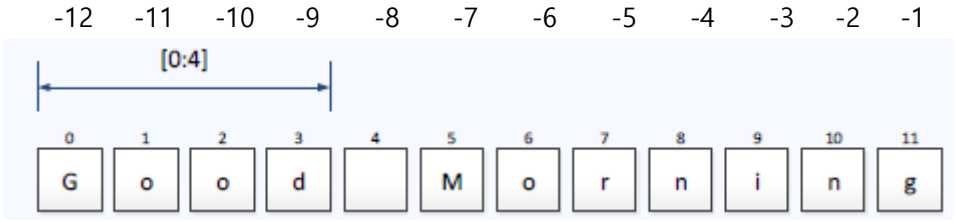
위치값 : a ~ 위치값 : b - 1

2. 순서

(실제로) a+1번째 순서 ~ b번째 순서

위치값을 a부터 b-1까지 출력,
- 파이썬에서 a의 위치 의미는 a+1th 선택 (왜냐하면 시작값이 0이니까)
- end값의 뜻은 end값 바로 전 까지라는 의미!

+마이너스 인덱스



ex.

4:6 실제 순서 5번째부터 6번째까지

0:4 실제 순서 - 1st부터 시작해서 3의 위치까지

5:12 실제 순서 6번째부터 12번째 까지

1:2 - 두 번째 컬럼만 선택됨! (1~1 = 2nd) 컬럼만 선택됨

```
>>> s='Good Morning'
>>> s[0:4]
'Good'
>>> a=s[5:12]
>>> a
'Morning'
```

특정 위치에 있는 문자를 참조하고 싶을 때 - 두 가지 슬라이싱 방법

1. a[:,1]
2. a[:,1:2]

둘 다 두 번째 행만 선택

----------**시험**

+ 한 개만 선택하면(정수색인) 데이터 프레임 구조 잃고 차원이 축소됨 (눕혀짐) in array 구조
but 연산 위해 세로로 유지하고 싶으면 1:2(슬라이스 색인) 형태로 1개의 컬럼 선택!
따라서, 데이터프레임 차원을 그대로 적용하고 싶으면 다음과 같은 슬라이스 색인 형태 사용

시험 ----------

예시 (간단)

```
>>> s
'Good Morning'
>>> s[0]
'G'
>>> s[8]
'n'
```

in 연산자

'문자열' in 변수명

원하는 부분이 문자열 안에 존재하는지 확인

패턴 포함 여부를 체크할 수 있다.
like 연산자처럼 (in SQL)
%in% 형태 (in R)
~ R처럼 in 사용해 T/F 하고 T만 포함된 거 가져올 수 있음

예시

```
>>> a='Good Morning'
>>> 'Good' in a
True
>>> 'Evening' in a
False
```

len() 함수

순서열 길이를 재는 함수. 문자열에도 사용 가능.

예시

```
>>> a='Good Morning'
>>> len(a)
12
```

아스키코드관련 함수

ord(c) - 하나의 문자에 대한 아스키코드

chr(c) - 아스키코드를 문자로 변경해줌

max(s) - 문자열 s 중 아스키코드가 가장 큰 문자

min(s) - 문자열 s 중 아스키코드가 가장 작은 문자

sorted(s) - 문자열 s 를 아스키코드 값 순으로 정렬하여 리스트로 반환

reversed(s) - 문자열 s 를 역순으로 저장, 반환시 list()함수를 써서 반환해야

문자열 함수

```
In [91]: ord("김")
Out[91]: 44608

In [92]: chr(110)
Out[92]: 'n'

In [93]: S2='파이썬 데이터 분석'
         len(S2), max(S2), min(S2), sorted(S2)
Out[93]: (10, '파', ' ', [' ', ' ', '데', '분', '석', '썬', '이', '이', '터', '파'])

In [98]: list(reversed(S2))
Out[98]: ['석', '분', ' ', '터', '이', '데', ' ', '썬', '이', '파']
```

텍스트 다루기 - 문자열 메소드method : 인자의 전달방식이 다름!

메소드	설명

startswith()	원본 문자열이 매개변수로 입력한 문자열로 시작되는지를 판단. True / False 반환
	>>> a='Hello'
	>>> a.startswith('He')
	True
	>>> a.startswith('lo')
	False

endswith()	원본 문자열이 매개변수로 입력한 문자열로 끝나는지를 판단. True / False 반환
	>>> a='Hello'
	>>> a.endswith('He')
	False
	>>> a.endswith('lo')
	True

find()	원본 문자열 안에 매개변수로 입력한 문자열이 존재하는 위치를 앞에서부터 찾음. 존재하지 않으면 -1을 결과로 내놓음 # 파이썬 순서인 걸 생각하기!
	>>> a='Hello'
	>>> a.find('K')
	-1

rfind()	원본 문자열 안에 매개변수로 입력한 문자열이 존재하는 위치를 뒤에서부터 찾음. 존재하지 않으면 -1을 결과로 내놓음 # 파이썬 순서인 걸 생각하기!
	>>> a.rfind('H')
	0
	>>> a.rfind('l')
	3
	>>> a.rfind('K')
	-1

count()	원본 문자열 안에 매개변수로 입력한 문자열이 몇 번 등장하는지 세기
	>>> a='Hello'
	>>> a.count('l')
	2

메소드	설명
lstrip()	원본 문자열 왼쪽에 있는 공백을 제거한다. >>> ' Left Strip'.lstrip() 'Left Strip'
rstrip()	원본 문자열 오른쪽에 있는 공백을 제거한다. >>> 'Right Strip '.rstrip() 'Right Strip'
strip()	원본 문자열 양쪽에 있는 공백을 제거한다. >>> ' Strip '.strip() 'Strip'

isalpha()	원본 문자열이 숫자와 기호를 제외한 알파벳(영문, 한글 등)으로만 구성되었는지 평가한다.
	>>> 'ABCdfds'.isalpha()
	True
	>>> 'ABC0908'.isalpha()
	False

isnumeric()	원본 문자열이 수로만 이루어져 있는지를 평가한다.
	>>> '1234'.isnumeric()
	True
	>>> 'ABC123'.isnumeric()
	False

isalnum()	원본 문자열이 알파벳과 수로만 이루어져 있는지 평가한다.
	>>> '1234ABC'.isalnum()
	True
	>>> '1234'.isalnum()
	True
	>>> 'ABC'.isalnum()
	True
	>>> '123 ABC'.isalnum()
	False # 공백이 들어가서 False 반환

★★★★★
replace()
★★★★★
	원본 문자열에서 찾고자 하는 문자열을 바꾸고자 하는 문자열로 변환한다.
	>>> a='Hello World'
	>>> b=a.replace('World', 'Korea')
	>>> a
	'Hello World'
	>>> b
	'Hello Korea'
	
	# 다 apply 형태 적용해야 함 (벡터연산 불가능) / 그동안은 for문 돌리기

split()	매개변수로 입력한 문자열을 기준으로 원본 문자열을 나누어 리스트를 만듦.
	(리스트=목록을 다루는 자료형)
	>>> a = 'Apple, Orange, Kiwi'
	>>> b = a.split(',')
	>>> b
	['Apple', ' Orange', ' Kiwi']
	>>> type(b)
	<class 'list'>
	
	# 자료구조가 list로 담김 - 1차원적 ~ array(배열) : 다차원이 됨.
	(list는 벡터(only in R)!랑 비슷 but list는 여러 데이터 타입을 같이 넣을 수 있음)

upper()	원본 문자열을 모두 대문자로 바꾼 문자열을 내놓는다.
	>>> a='lower case'
	>>> b=a.upper()
	>>> a
	'lower case'
	>>> b
	'LOWER CASE

lower()	원본 문자열을 모두 소문자로 바꾼 문자열을 내놓는다.
	>>> a='UPPER CASE'
	>>> b=a.lower()
	>>> a
	'UPPER CASE'
	>>> b
	'upper case'

format()	형식을 갖춘 문자열을 만들 때 사용.
	문자열 안에 { }로 데이터가 들어갈 자리를 만들어두고,
	format() 함수를 호출할 때 이 자리에 들어갈 데이터를 순서대로 넣어주면
	원하는 형식의 문자열을 만들어낼 수 있다.
	>>> a='My name is {0}. I am {1} years old.'.format('Mario',40)
	>>> a
	'My name is Mario. I am 40 years old.'
	>>> b='My name is {name}. I am {age} years old.'.format(name='Luigi', age=35)
	>>> b
	'My name is Luigi. I am 35 years old.'

적용시킬 대상은 method **앞에** 나와서 호출!
 # max(a) = a.max() - 둘 다 적용됨

딥러닝이 어려운 이유? 정형화된 데이터가 아니기 때문에!
대개 데이터 타입 - list / array 구조(벡터연산 안됨) -> for문 많이 쓰므로 어려움!

ex. a.startswith('He') ~ 안 되면 적용함수 써야 함 (apply...)

a. 쓰면 뒤에가능한 method잔뜩 나옴

+ 문자열 메서드 #dir(str) 로 확인

	메소드	설명
대소문자 변환	capitalize() title() upper() lower() swapcase()	첫 문자만 대문자, 나머지는 소문자 각 단어의 첫 문자만 대문자, 나머지는 소문자 문자열 전체를 대문자 문자열 전체를 소문자 문자열의 대문자는 소문자로, 소문자는 대문자로 변경
정렬	center(n,x) ljust(n,x) rjust(n,x)	가운데 정렬 (n : 전체 자리 수, x : 남는 자리 채울 문자) 왼쪽 정렬 (n : 전체 자리 수, x : 남는 자리 채울 문자) 오른쪽 정렬 (n : 전체 자리 수, x : 남는 자리 채울 문자)
특정 문자 개수 찾기	count(x,a,b)	문자열 x 의 개수 세기(a,b 는 각각 찾을 시작과 끝 인덱스)
특정 문자 인덱스 찾기	index() rindex() find() rfind()	문자열 x 가 시작하는 인덱스 (없는 문자열은 Error 발생) 문자열 x 가 시작하는 인덱스를 오른쪽부터 찾기 index 메소드와 동일하지만, 없는 문자열은 -1 반환 rindex 메소드와 동일하지만, 없는 문자열은 -1 반환
문자열의 시작과 끝	startswith(x) endswith(x)	문자열 x 로 시작하는지 확인 문자열 x 로 끝나는지 확인
출력 포맷	format() format_map()	문자열 출력 포맷을 지정 map()함수를 이용해서 문자열 출력 포맷을 지정
문자열 결합/분리	join(x) split(x,n) rsplit(x,n) splitlines()	x 의 각 문자 사이마다 문자열을 결합함. 문자열을 구분자 x 로 n 번 분리하여 리스트로 반환 split 과 동일하지만 우측부터 분리 '₩n'을 기준으로 문자열을 분리하여 리스트로 반환
공백제거	strip(x) lstrip(x) rstrip(x)	문자열 양 끝에 있는 문자열 x 제거 (기본값 : 공백 제거) 문자열 왼쪽에 있는 문자열 x 제거 (기본값 : 공백 제거) 문자열 오른쪽에 있는 문자열 x 제거 (기본값 : 공백 제거)
문자열 분할	partition(x) rpartition()	문자열 x 기준으로 세 부분으로 분할한 튜플을 반환 오른쪽으로부터 분할한 결과를 튜플로 반환
0 으로 채우기	zfill(n)	문자열 앞에 0 을 추가한 문자열을 반환 (n 은 전체 자리 수)
문자 대체	str.maketrans(x,y) translate(x) replace(x,y)	문자열 x 의 문자를 y 의 문자로 대체, 사전으로 반환 str.maketrans 를 이용해 문자 변환 문자열의 일부인 x 를 다른 문자 y 로 대체
탭키 조정	expandtabs()	탭의 크기를 조정
문자열 구성 확인	isalnum() isalpha()	문자열이 모두 알파벳과 숫자로만 이루어졌는지 확인 문자열이 모두 알파벳으로만 이루어졌는지 확인

	isdigit()	문자열이 모두 숫자로만 이루어졌는지 확인('3²' => True)
	isupper()	문자열이 모두 대문자로만 이루어졌는지 확인
	islower()	문자열이 모두 소문자로만 이루어졌는지 확인
	isspace()	문자열이 모두 공백으로만 이루어졌는지 확인
	istitle()	문자열의 모든 단어가 대문자로 시작하는지 확인
	isdecimal()	문자열이 모두 명확한 숫자로만 이루어졌는지 확인 ('3²' => False)
	isidentifier()	문자열이 identifier 로 사용할 수 있는지 확인
	isnumeric()	문자열이 모두 숫자값 표현에 해당하는 텍스트인지 확인

cf. isdigit() / isdecimal() / isnumeric()

isdigit()은 단일 글자가 '숫자' 모양으로 생겼으면 True 를 리턴한다.

isdecimal()은 단일 글자가 무조건 0-9 의 정수형태여야 한다.

isnumeric()은 숫자값 표현에 해당하는 텍스트까지 인정해준다. 예를 들어 "½" 이런 특수문자도 isnumeric()에서는 True 로 판정된다.

따라서 **int()**로 변환할 목적이라면, **isdecimal()**을 사용해 변환하는 것이 바람직.

('3²'.isdigit() => True 이지만 int('3²') Error 발생! / '3²'.isdecimal() => False, 가려낼 수 있다)

대소문자 변환

In [77]:
```
S='PYthon은 쉬워서 PROgramming하기 편해요'
S.capitalize(), S.title()
```

Out[77]: ('Python은 쉬워서 programming하기 편해요', 'Python은 쉬워서 Programming하기 편해요')

In [81]:
```
S.upper(),S.lower()
```

Out[81]: ('PYTHON은 쉬워서 PROGRAMMING하기 편해요', 'python은 쉬워서 programming하기 편해요')

In [82]:
```
S.swapcase()
```

Out[82]: 'pyTHON은 쉬워서 proGRAMMING하기 편해요'

정렬

In [83]:
```
S.center(35)
```

Out[83]: '   PYthon은 쉬워서 PROgramming하기 편해요   '

In [84]:
```
S.ljust(35,'_')
```

Out[84]: 'PYthon은 쉬워서 PROgramming하기 편해요_____'

In [85]:
```
S.rjust(35,'*')
```

Out[85]: '******PYthon은 쉬워서 PROgramming하기 편해요'

특정 문자 개수 찾기 + 인덱스 찾기

In [86]:
```
S.count('n'), S.count('on'), S.index('n'), S.rindex('n'), S.find('o')
```

Out[86]: (2, 1, 5, 21, 4)

In [87]:
```
S.index('b') # index는 없는 문자 에러 발생
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-87-f15e40d20244> in <module>
----> 1 S.index('b')

ValueError: substring not found
```

In [88]:
```
S.find('b')
```

Out[88]: -1

문자열의 시작과 끝

In [90]:
```
S.startswith('P'), S.startswith('요'), S.endswith('P'), S.endswith('요')
```

Out[90]: (True, False, False, True)

문자열 결합/분리

```
In [103]: chr="**"
          S.join(chr)
Out[103]: '*PYthon은 쉬워서 PROgramming하기 편해요*'

In [104]: chr='*'
          chr.join(S)
Out[104]: 'P*Y*t*h*o*n*은* *쉬*워*서* *P*R*O*g*r*a*m*m*i*n*g*하*기* *편*해*요'

In [108]: S3='    ₩tpython은 쉬워서 ₩nprogramming하기 편해요₩t₩t₩n'

In [110]: S3.splitlines()
Out[110]: ['    ₩tpython은 쉬워서 ', 'programming하기 편해요₩t₩t']
```

공백제거

```
In [111]: S3.strip()
Out[111]: 'python은 쉬워서 ₩nprogramming하기 편해요'

In [113]: S3.lstrip()
Out[113]: 'python은 쉬워서 ₩nprogramming하기 편해요₩t₩t₩n'

In [114]: S3.rstrip()
Out[114]: '    ₩tpython은 쉬워서 ₩nprogramming하기 편해요'

In [115]: S3='*********가나다라마바사****'

In [116]: S3.strip('*')
Out[116]: '가나다라마바사'
```

문자열 분할

```
In [117]: S="응용통계학과는 통계학, SPSS, R, SAS 등을 배웁니다"

In [118]: S.partition('는')
Out[118]: ('응용통계학과', '는', ' 통계학, SPSS, R, SAS 등을 배웁니다')

In [119]: S.partition('SPSS')
Out[119]: ('응용통계학과는 통계학, ', 'SPSS', ', R, SAS 등을 배웁니다')

In [120]: S.rpartition(' ')
Out[120]: ('응용통계학과는 통계학, SPSS, R, SAS 등을', ' ', '배웁니다')
```

0으로 채우기

```
In [137]: S.zfill(20)
Out[137]: '0000000000statistics'
```

문자 대체

```
In [129]: S='statistics'

In [131]: x=str.maketrans("st","ST") # {'s' : "S", "t" : "T"} 사전을 반환 (아스키코드)
          x
Out[131]: {115: 83, 116: 84}

In [133]: S.translate(x)
Out[133]: 'STaTiSTicS'

In [140]: a='Hello World'
          b=a.replace('World','Korea')
          a, b
Out[140]: ('Hello World', 'Hello Korea')
```

cf. 메서드를 사용하면 결과만 반환. 원 변수 변경 없음 (바꿔주는 메서드도 있음)
메서드를 사용한 결과를 저장하고 싶으면 새로운 변수로 지정해서 따로 저장

수에서 텍스트로, 텍스트에서 수로

input - 사용자의 입력을 기다림 cf. readLines in R

```
>>> input("input value : ")
input value : 5                    #5 내가 입력
'5'

# int() 정수 float() 소수 str() 문자열

>>> a = input("input value : ")
input value : 6

>>> type(a)
<class 'str'>

>>> b=input("input value : ")
input value : 9

>>> a*b
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    a*b
TypeError: can't multiply sequence by non-int of type 'str'
```

해결방법 **(숫자로 입력하고 싶을 때!)**

1) 처음 입력 전달할 때 숫자로 입력

```
>>> a = int(input("input value : "))
input value : 9
>>> type(a)
<class 'int'>
```

2) 계산시 숫자로 입력

```
>>> int(a) * int(b)
54
```

cf. input 은 문자열 입력 함수이기 때문에!! 기본 데이터 타입이 문자로 들어감
(for 숫자/문자 모두 들어오기 위해서)

```
>>> a=input()
9

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

print - 출력만 할 뿐 입력은 안 함.

print(" 첫 번째 수를 입력하세요. :")

cf. input - 출력과 동시에 오른쪽에 커서 나타남
print - 다음 라인에 커서 나타남

```
print("첫번째 수를 입력하세요. : ")
a = input()
print("두 번째 수를 입력하세요. : ")
b = input()
result = int(a) * int(b)
print("{0} * {1} = {2}".format(a, b, result))
# a, b 문자형태로 그냥 출력!
```

★★★★★★★★★★★아래 내용 잘 보기!★★★★★★★★★★★★★★★★★★★★

+ 마지막 줄 format method 말고 %d 등 표시형식 사용해서 6 * 8 = 48 나타내기

```
print("%d * %d" = %d" % (int(a), int(b), int(a)*int(b)))
```

cf. 실행 오류

```
print("%d * %d = %d" % (a, b, int(a)*int(b)))
```

a, b 가 정수형태이기 때문에 %d 로 안 먹음! ---- **앞 형식과 뒤의 형식 통일 신경써야 함**

 # 만일 a, b 로 그냥 쓰고 싶으면 s 로 쓰기!

```
print("%s * %s = %d" % (a, b, int(a)*int(b)))
```

=>

결론

1. 표시 형식과 전달할 형식 통일 신경쓰기

2. 연산은 미리 하고 변수 지정으로!

-result 변수 지정 자체에서 int(a) * int(b) 가능 (복잡해서 변수처리 하고 내려오는 것 선호)

프로그램 만들기!

메뉴에서 New File 선택 - 연달아 시행시키고 싶은 명령어 쓰기 - 저장한 후 실행시켜야 함 F5

```
a = int(input("input value : "))
b = int(input("input value : "))
print("%d * %d = %d" % (int(a), int(b), int(a)*int(b)))
```

---

cf. "x 곱하기 y 는 z" 형태로 프린트하기

```
a = int(input("input value : "))
b = int(input("input value : "))
print("%d 곱하기 %d 는 %d" % (a, b, int(a)*int(b)))
```

cf. format - 0,1,2 로 순서 지정해서 전달하니까 더 많이 사용

---

result=int(a)*int(b) -- 숫자

print("{0} * {1} = {2}".format(a, b, result))

여기서 a, b 는 문자임. 그냥 문자 형태로 출력하는 것

result 는 숫자이지만 {2}에서 포맷 지정되지 않았기 때문에 문자 형태로 출력

만일

print("{0:d} * {1:d} = {2:d}".format(a, b, result)) #[굳이 정수일 필요 없지만!]

이렇게 하면 **오류 발생**, 뒤에 format 에 반드시 뒤에 있는 타입은 정수여야 하므로

print("{0:d} * {1:d} = {2:d}".format(**int(a)**, **int(b)**, result))로 해야 함.!!!

★★★★★★★★★★★★!!!총!!!결!!!론!!!★★★★★★★★★★★★★★★★★★★★

순서대로 전달시 - (%든, format 이든) 앞 뒤 개수 일치 + 입.출력 데이터 타입 일치

!!!!!!!!!!제일 쉽게 하는 실수!!!!!!!!!!!(에러발생)

int() - 문자열 -> 숫자로 바꾸기

str() - 숫자 -> 문자열로 바꾸기

```
>>> import math

>>> type(math.pi)
<class 'float'>
>>> text="원주율은 " + str(math.pi) + " 입니다."
>>> text
'원주율은 3.141592653589793 입니다.'
```

Chapter 3 - 데이터 다루기

산술연산자

산술연산자의 종류

=

+

-

*

/

// 나누기 (몫)

% 나머지 값

** 제곱

이 뜻은 아래와 동일!

>>> a=5 ; b=3

>>> a, b = 5, 3

>>> a=3

>>> b=3

산술연산자의 우선순위

아무리 *, /이 +, -보다 우선순위더라도 괄호를 사용하는 것이 더 나은 코딩!

a = b + c * d 보다

a = b + (c * d) 가 더 나은 코딩!

산술 연산을 하는 문자열과 숫자의 상호 변환

- 문자열이 숫자로 구성되어 있을 때, int() 또는 float() 함수를 사용해서 정수/실수로 변환

```
>>> s1, s2, s3 = "100", "100.123", "99999999999"
>>> print(int(s1) + 1, float(s2) + 1, int(s3) +1)
101 101.123 100000000000
```

- 숫자를 문자열로 변환하려면 str() 함수 사용

```
>>> a = 100
>>> str(a) + "1"
'1001'
```

#문자의 결합은 확장 ! 1001 로 확장됨 (**not 연산**)

cf. print() 함수는 출력 결과 - 문자열인지 구분하기 어려워 사용하지 않음.

산술연산자와 대입 연산자

표 4-2 대입 연산자의 종류

연산자	사용 예	설명
+=	a += 3	a = a + 3과 동일
-=	a -= 3	a = a - 3과 동일
*=	a *= 3	a = a * 3과 동일
/=	a /= 3	a = a / 3과 동일
//=	a //= 3	a = a // 3과 동일
%=	a %= 3	a = a % 3과 동일
**=	a **= 3	a = a ** 3과 동일

cf. 파이썬에는 C/C++,자바 등에 있는 증가연산자 ++나 감소 연산자 --가 없음.

실습문제

돈(예로 777777)을 입력하면 5만 원, 1만 원, 5000원, 1000원 지폐로 교환하는 프로그램을 작성해 보자.


```
a = int(input("지폐로 교환할 돈은 얼마? "))
b = a // 50000
c = (a % 50000) // 10000
d = ((a % 50000) % 10000) // 5000
e= ((a % 50000) % 10000) % 5000 // 1000
f= ((a % 50000) % 10000) % 5000 % 1000

print("₩n ₩n 50000 원 짜리 ==> {0}장, ₩n 10000 원 짜리 ==> {1}장, ₩n 5000 원 짜리 ==> {2}장, ₩n 1000 원 짜리 ==> {3}장, ₩n 지폐로 바꾸지 못한 돈 ==> {4}원".format(b,c,d,e,f))
```

20180605

관계 연산자

관계 연산자의 개념

- 어떤 것이 크거나 작거나 같은지 비교한 것 (참 - True, 거짓 - False)
- 주로 조건문(if)나 반복문(while)에서 사용, 단독문에선 거의 사용하지 않음

표 4-3 관계 연산자의 종류

연산자	의미	설명
==	같다.	두 값이 동일하면 참
!=	같지 않다.	두 값이 다르면 참
〉	크다.	왼쪽이 크면 참
〈	작다.	왼쪽이 작으면 참
〉=	크거나 같다.	왼쪽이 크거나 같으면 참
〈=	작거나 같다.	왼쪽이 작거나 같으면 참

 # = 말고 == 사용하는 것 유의!

논리 연산자

논리연산자의 종류와 사용

- and, or, not 세 종류

ex. a 라는 값이 100 과 200 사이에 들어있어야 한다는 조건 표현

(a > 100) and (a < 100)

표 4-4 논리 연산자의 종류

연산자	의미	설명	사용 예
and(논리곱)	~이고, 그리고	둘 다 참이어야 참	(a 〉100) and (a 〈200)
or(논리합)	~이거나, 또는	둘 중 하나만 참이어도 참	(a == 100) or (a == 200)
not(논리부정)	~아니다, 부정	참이면 거짓, 거짓이면 참	not(a 〈100)

#각 조건을 괄호 형태를 통해서 사용한다 !

예시

```
>>> a = 99
>>> (a > 100) and (a < 200)
False
>>> (a > 100) or (a < 200)
True
>>> not (a==100)
True
>>> not (a > 100)
True
```

+ 숫자 형태로 했을 때, (0)은 False!

```
>>> if(1234) : print("참이면 보여요")      # 엔터 두 번 쳐야 함!(print)라서
참이면 보여요
>>> if(0) : print("거짓이면 안 보여요")    # 0 은 False, 그 외 숫자는 모두 True
```

cf. is, is not 연산자

- is, is not 연산자는 데이터의 값이 아닌, 데이터의 레퍼런스(저장되어 있는 주소)를 비교하는 연산자. 즉, 메모리에서의 물리적 위치가 같은지를 알아보는 연산자
- python 에서는 [-5, 256] 범위의 Integer 는 이미 메모리 주소를 갖고 있음.
- 이를 벗어나는 경우 같은 값이어도 레퍼런스가 다름.

```
In [25]: W=-5; X=-5; Y=257; Z=257
         id(W), id(X), id(-5), id(Y), id(Z), id(257)

Out[25]: (1576775712,
          1576775712,
          1576775712,
          1420522347408,
          1420522347888,
          1420522347920)
```

연산자의 우선 순위

우선순위	연산자	설명
1	( ) [ ] { }	괄호, 리스트, 딕셔너리, 세트 등
2	**	지수
3	+ - ~	단항 연산자
4	* / // %	산술 연산자
5	+ -	산술 연산자
6	<< >>	비트 시프트 연산자
7	&	비트 논리곱
8	^	비트 배타적 논리합
9	|	비트 논리합
10	< > > = < =	관계 연산자
11	== !=	동등 연산자
12	= %= /= //= -= += *= **=	대입 연산자
13	not	논리 연산자
14	and	논리 연산자
15	or	논리 연산자
16	if ~ else	비교식

Chapter 4

조건문

if 문

- 파이썬은 **들여쓰기**가 매우 중요. -> 동일한 수준의 문장들 들여쓰기 같게 잘 하기!
- **if 문 끝난 후 엔터 두 번 치는 것 권장!** -> **타 명령문과 if 문을 구분**
- if 문 다음에 '실행할 문장'은 if 문 다음 줄에서 들여쓰기를 해서 작성.
- 대화형 모드에서는 '실행할 문장' 모두 끝나고 Enter 2 번 눌러야 if 문이 끝나는 것으로 간주

 # : then 의 축소 기호

-> 들여쓰기에서 실수 제일 많이 발생!
깔끔하게 들여쓰기 잘 정리 해야 함.

+ if 문 끝나면 빈 라인 쓰는 것 권고.

예시

```
a = 99
if a < 100 :
    print("100 보다 작군요.")
```

출력 결과

```
100 보다 작군요.
```

조건이 참이고 실행할 문장이 두 개일 때

 # 들여쓰기 실수 예제

```
a = 200

if a < 100 :
    print("100 보다 작군요.")
print("거짓이므로 이 문장은 안 보이겠죠?")    # 들여쓰기 실수

print("프로그램 끝")
```

출력 결과

```
거짓이므로 이 문장은 안 보이겠죠?
프로그램 끝
```

 # 들여쓰기 실수 없을 때

```
a = 200

if a < 100 :
    print("100 보다 작군요.")
    print("거짓이므로 이 문장은 안 보이겠죠?")

print("프로그램 끝")
```

출력결과

```
프로그램 끝
```

if ~ else 문

- if , else 는 같은 위치에서 들여쓰기!
- 참, 거짓일 때 실행할 문장도 같은 들여쓰기로 해야 함!

예제

```
a = 200

if a < 100 :
    print("100 보다 작군요")
    print("참이면 이 문장도 보이겠죠?")
else :
    print("100 보다 크군요")
    print("거짓이면 이 문장도 보이겠죠?")

print("프로그램 끝")
```

출력결과

```
100 보다 크군요.
거짓이면 이 문장도 보이겠죠?
프로그램 끝
```

예제 : 입력 숫자가 짝수/홀수인지 계산

```
a = int(input("정수를 입력하세요 : "))

if a % 2 == 0 :
    print("짝수를 입력하셨군요")
else :
    print("홀수를 입력하셨군요")
```

출력 결과

```
정수를 입력하세요 : 125
홀수를 입력하셨군요
```

if ~ else ~ if ~ else 문 (중첩형태)

```
a = 75

if a > 50 :
    if a < 100 :                    # 중첩형태 !! if 안에 if문 다시
        print("50 보다 크고 100 보다 작군요")
    else :                          # 중첩형태 !! if 안에 있는 if문의 반대 else
        print("와 ~ 100 보다 크군요.")
else :
    print("에고 ~ 50 보다 작군요.")
```

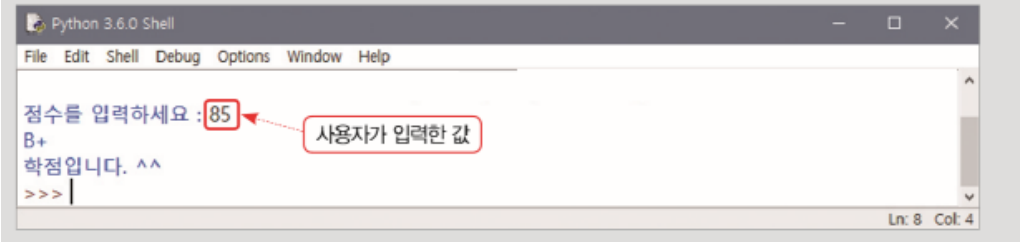
출력결과

```
50 보다 크고 100 보다 작군요.
```

else if 를 사용하고 싶으면 elif 를 쓰기!!!!

실습 문제

95점 이상 : A+, 90점 이상 : A0, 85점 이상 : B+, 80점 이상 : B0,
75점 이상 : C+, 70점 이상 : C0, 65점 이상 : D+, 60점 이상 : D0, 60점 미만 : F



나

```
a=int(input("점수를 입력하세요 : "))

if a >= 95 :
    print('A+')
elif a >= 90 :              # else if in R - elif !
    print('A0')
elif a>=85 :
    print('B+')
elif a>=80 :
    print('B0')
elif a>=75 :
    print('C+')
elif a>=70 :
    print('C0')
elif a>=65 :
    print('D+')
elif a>=60 :
    print('D0')
else :
    print('F')

print("학점입니다. ^^")
```

강사님

result = 'A+' 등으로 변수에 저장 후
print ("당신의 학점은 %s 입니다" % result)

i) int() 유의!

ii) 하위문법 들여쓰기!

삼항 연산자를 사용한 if 문

```
jumsu = 55
res=''

if jumsu >= 60 :
    res = '합격'
else :
    res = '불합격'

print(res)

# res='' 코드 없어도 되지만, 꼼꼼하게 하고 싶기 때문
(if 문 에러 났을 때 print(res)에서 출력되지 않으면 if 문에 문제가 있음을 알 수 있음)
```

if 문 줄임

참 출력형태 조건 거짓 출력형태

```
res = '합격' if jumsu >= 60 else '불합격'

# 조건, 출력 간단한 형식 ~ 람다를 통해 사용할 수 있다. 축약형 형태로 만들어야 할 때 !
# 조건문이 한 라인에 오면 해석할 수 있어야!

+
res='A' if jumsu>= 95 else 'A0' if jumsu>=90 else 'B+' if jumsu>=85 else 'B0' if jumsu>=80 else
'C+' if jumsu>=75 else 'C0' if jumsu >=70 else 'D+' if jumsu>=60 else 'F'
```

if 문 응용

리스트와 함께 사용

- 리스트 : 데이터 여러 개를 한 곳에 담아 놓은 것
- 방법 : 대괄호 [ ] 로 묶고, 그 안에 필요한 것들을 한꺼번에 넣음
- 예 : fruit 변수에 값 4 개를 리스트로 하나로 묶어 대입

```
#리스트
fruit = ['사과', '배', '딸기', '포도']
print(fruit)
```

결과

```
['사과', '배', '딸기', '포도']
```

```
# 추가
fruit.append('귤')        # append method 로 새로운 원소 추가
print(fruit)
```

결과

```
['사과', '배', '딸기', '포도', '귤']
```

```
# 리스트에 해당 항목 있는지 찾기
if '딸기' in fruit :                              # 이런 형태로 찾기!!!
    print("딸기가 있네요 ^^")
```

결과

```
딸기가 있네요 ^^
```

cf. append(method) -> for 문을 통해서 list 에 계속 추가! / 형식 조심! 낯설어서 실수 할 수도!

+

리스트 in python(vs 벡터 in R)

1. 벡터계산 X
2. 데이터타입 여러개 가능

연습문제

두 수와 연산자를 입력받아 입력받은 수와 연산 결과를 다음과 같이 출력.
단, 0 으로 나누려고 하면 에러 출력

#강사님 방법

```
v1=int(input("첫 번째 숫자를 입력하세요 : "))
op=input("계산을 입력하세요 (+, -, *, /, **)")
v2=int(input("두 번째 숫자를 입력하세요: "))
result=''
#if 문이 에러나도 result 살리게끔 ! 나중에 오류 나도 바로 찾게끔 습관들이기

if op=='+' :
    result=v1+v2
elif op=='-' :
    result=v1-v2
elif op=='*' :
    result=v1*v2
elif (op=='/') and not (v2==0) :
    result = v1 / v1
elif op=='**' :
    result=v1**v2

if (op=='/') and (v2==0) :
  print("0 으로는 나누면 안됩니다ㅠㅠ")
else :
    print("##계산기 : %d %s %d = %s" %(v1, op, v2, result))


# result 를 %s 를 쓴 이유?
  나누기 할 때 정수가 아닌 경우도 출력하기 위해서!
+ float 은 지저분해서
```

나

```
a=int(input("첫 번째 숫자를 입력하세요 : "))
pm=input("계산을 입력하세요 (+, -, *, /, **)")
b=int(input("두 번째 숫자를 입력하세요: "))

if pm== "/" :
    if b==0 :
        print("0 으로는 나누면 안 됩니다 ㅠ_ㅠ")
    else :
        print(a/b)
elif pm=="+" :
        print(a+b)
elif pm=="-" :
        print(a-b)
elif pm=="*" :
        print(a*b)
elif pm=="**" :
        print(a**b)
```

실수했던 것들

1. pm = "*", b=0

?
변수 지정 아님! 맞는지 확인하려면 = 두 개 쓰기!

→ pm == "*"
→ bm == 0

2. 마지막 줄 실수

```
a=int(input("첫 번째 숫자를 입력하세요 : "))
pm=input("계산을 입력하세요 (+, -, *, /, **)")
b=int(input("두 번째 숫자를 입력하세요: "))

if pm== "/" :
    if b==0 :
        print("0 으로는 나누면 안 됩니다 ㅠ_ㅠ")
    else :
        result = a/b
elif pm=="+" :
    result = a+b
elif pm=="-" :
    result = a-b
elif pm=="*" :
    result = a*b
elif pm=="**" :
    result = a**b

print("결과는 {0} 입니다.".format(result))
```

?
pm 이 /이고 b 가 0 일 때 result 변수가 없는데, 마지막 문장은 result 를 출력해야 함.

예 : 0 부터 **9 까지** 숫자 중에서 리스트 안에 없는 숫자 찾기

```
import random

numbers=[]                #빈벡터 생성

for num in range (0, 10) :          # 0 부터 9 까지 총 10 회를 반복
/ range 에서는 시작숫자 ~ 끝숫자 +1 !! 앞에서 색인 했던 거랑 비교
    numbers.append(random.randrange(0, 10))

print("생성된 리스트", numbers)

for num in range (0, 10) :          # 0 부터 9 까지 숫자를 num 에 넣음
    if num not in numbers :                 # 리스트 원소 하나씩 반복
        print("숫자 %d 는(은) 리스트에 없네요." % num)
```

출력 결과

```
생성된 리스트 [4, 3, 6, 5, 5, 9, 2, 1, 4, 5]
숫자 0 는(은) 리스트에 없네요.
숫자 7 는(은) 리스트에 없네요.
숫자 8 는(은) 리스트에 없네요.
```

cf. 비정형화 데이터 ~ 파이썬 ~ for 문을 많이 돌려야 함..!

연습문제

[프로그램 2]의 두 번째 기능처럼 두 숫자를 입력받고 두 숫자 사이의 합계를 구하는 프로그램을 만들어 보자. 단 1씩 증가하지 않고 증가하는 숫자도 입력받는다. 예를 들어 1, 100, 3을 입력하면 1+4+…+100의 합계를 구한다.

힌트 range(시작값, 끝값+1, 증가값) 형식으로 사용한다.


```
a = int(input("*** 첫 번째 숫자를 입력하세요 : "))
b = int(input("*** 두 번째 숫자를 입력하세요 : "))
c = int(input("*** 더할 숫자를 입력하세요 :  "))
sum = 0

for num in range(a, b+1, c) :
    sum = sum + num

#format method 로 할 때!
print("{0} + {1} + ... + {2}는 {3}입니다.".format(a, a+c, b, sum))

#표현 형태로 할 때!
print("%d +%d + ... + %d 는 %d 입니다." % (a, a+c, b, sum))
```

연습문제

이 장에서 만들 프로그램

- **[프로그램 2] 종합 계산기**
  - 기능이 두 가지인 종합 계산기 프로그램


```
cho = int(input("1. 입력한 수식 계산  2. 두 수 사이의 합계 : "))

if cho==1 :
    a1 = input("*** 수식을 입력하세요 : ")
    print(eval(a1))
else :
    v1 = int(input("*** 첫 번째 숫자를 입력하세요 : "))
    v2 = int(input("*** 두 번째 숫자를 입력하세요 : "))
    sum = 0
    for num in range(v1, v2+1) :
        sum = sum + num
    print("%d + ... + %d는 %d입니다." % (v1, v2, sum))
```

- eval( ) - 문자 형태로 입력된 값을 무조건 수행! (명령어 /수식 등)

- range 끝값+1 기억하기 !

강사님방법 -

```
cho = int(input("1. 입력한 수식 계산 #n2. 두 수 사이의 합계 #n메뉴 선택 : "))

if cho==1 :
    a1 = input("*** 수식을 입력하세요 : ")
    result = eval(a1)
    print("%s 결과는 %s 입니다." % (a1, result))
elif cho==2 :
    v1 = int(input("*** 첫 번째 숫자를 입력하세요 : "))
    v2 = int(input("*** 두 번째 숫자를 입력하세요 : "))
    sum = 0
    for num in range(v1, v2+1) :
        sum = sum + num

    print("%d + ... + %d는 %d입니다." % (v1, v2, sum))

else :
    print("잘못된 선택입니다.")
```

-1 번 a1 도, **result 도 %s 처리!**

(특히 result- 나누기일 경우 int - 정수, float - 형식 지저분)

0607

Chapter 05 반복문

기본 for 문

반복문의 개념과 필요성

```
#반복문 사용 X
print("안녕하세요? for 문을 공부 중입니다. ^^")
print("안녕하세요? for 문을 공부 중입니다. ^^")
print("안녕하세요? for 문을 공부 중입니다. ^^")
```

출력결과

```
안녕하세요? for 문을 공부 중입니다. ^^
안녕하세요? for 문을 공부 중입니다. ^^
안녕하세요? for 문을 공부 중입니다. ^^
```

```
# 반복문 사용
for i in range(0, 3, 1) :
    print("안녕하세요? for 문을 공부 중입니다. ^^")
```

출력결과

```
안녕하세요? for 문을 공부 중입니다. ^^
안녕하세요? for 문을 공부 중입니다. ^^
안녕하세요? for 문을 공부 중입니다. ^^
```

for 문의 개념

기본 형식

for 변수 in range(시작값, 끝값+1, 증가값) : #시작값 ~ 끝값 표시 색인 방법과 구분!

    _____ # 이 부분을 반복

예) range() 함수 사용과 내부적 변경

```
for i in range(0, 3, 1) :
    print("안녕하세요? for 문을 공부 중입니다. ^^")

for i in [0,1,2] :          # for 문 range 대신 list 사용 -> 리스트의 원소를 순서대로 가져감!
    print("안녕하세요? for 문을 공부 중입니다. ^^")
```

예) i 값 코드 내부 사용

```
for i in range(0, 3, 1) :
    print("%d : 안녕하세요? for 문을 공부 중입니다. ^^" % i)
```

출력결과

```
0 : 안녕하세요? for 문을 공부 중입니다. ^^
1 : 안녕하세요? for 문을 공부 중입니다. ^^
2 : 안녕하세요? for 문을 공부 중입니다. ^^
```

tip. for 문에서 i(변수) 지정 하지 않으려면 _(언더바) 사용

```
for _ in range(0,3,1) :
    print("안녕하세요? for 문을 공부중입니다. ^^")
```

예) range() 함수의 시작값을 2, i 값을 1 씩 줄여 (0 이 될 때까지) print() 함수 3 번 실행

```
for i in range (2, -1, -1) :
    print("%d : 안녕하세요? for 문을 공부중입니다. ^^" % i)
```

출력결과

```
2 : 안녕하세요? for 문을 공부 중입니다. ^^
1 : 안녕하세요? for 문을 공부 중입니다. ^^
0 : 안녕하세요? for 문을 공부 중입니다. ^^
```

 # 마이너스일 때는 또 다르군! abs(끝날 수) 에 +1 한 다음 마이너스 부호 붙이기 인듯..!
물론 마지막 세 번째 인수도 중요함!

예) 1~5 의 숫자들을 차례대로 출력

```
for i in range (1, 6, 1) :
    print(i, end=" ")
```

출력결과

```
1 2 3 4 5
```

 # end = ""
: 출력되는 것이 여럿인데, 그것들을 모두 한 라인에 올리고 싶을 때 !

★★★★★ **for i in l1**　　**각 리스트에 있는 값을 하나씩 가져옴**

```
>>> for i in l1 :
        print(i + 1) )
2
3
4
5
6
```

-----cf. 벡터연산 불가능 in python-----

```
>>> l1 = [1,2,3,4,5]
>>> l1 + 1
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    l1 + 1
TypeError: can only concatenate list (not "int") to list
```

★★★★새로운 리스트에 연산한 값 추가하기

```
>>> l2 = []        # 빈벡터 설정
>>> for i in l1 :
        l2.append(i+1)
>>> l2
[2, 3, 4, 5, 6]
```

 # 간단한 산술연산조차 for 문을 써야 함 in python(벡터연산 불가)

```
>>> for i in l1 :
        print(i, end="")
12345
```

 # 프린트 - default : 한줄 내려서 출력됨.
띄어쓰기로 반복출력하고 싶으면 end = " " (띄어쓰기) -- 구분기호

```
>>> for i in l1 :
        print(i, end=" ")      # 한 칸 띄어쓰기
1 2 3 4 5

>>> for i in l1 :
        print(i, end="  ")     # 두 칸 띄어쓰기

    1  2  3  4  5
```

★★ 맨 마지막에 새로 한 줄만 추가하고 싶을 때! ★★

---------?_? 안됨----------

```
l1 = [1,2,3,4,5]
for i in l1 :
        print(i, end=" ")

print("끝")
```

출력결과

```
1 2 3 4 5 끝      #연속적으로 이어짐,
```

★★★★★★★★!!!!! 해결 방법!!! ★★★★★★★★

방법 1 개행문자 삽입

```
l1 = [1,2,3,4,5]

for i in l1 :
        print(i, end=" ")

print("₩n 끝")
```

방법 2 프린트 명령어 한 번 더 사용

```
l1 = [1,2,3,4,5]

for i in l1 :
        print(i, end=" ")

print("")    #내려쓰고 싶다는 뜻
print("끝")
```

방법 1, 2 출력결과

```
1 2 3 4 5
끝                          ##조치 꼭 취해주어야 함!
```

★★리스트에 있는 문자원소를 upper method 적용시키고 싶을 때! ★★★

```
----------오류발생----------
>>> l2=['a','b','c']
>>> l2.upper()      # 리스트는 벡터연산 X!! // 하나하나씩 for 문 통해 전달 / apply 함수 사용
Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    l2.upper()
AttributeError: 'list' object has no attribute 'upper'
----------
```

l2 에 있는 a,b,c l3 에 대문자로 저장시키기 / for 문 사용

```
>>> l2
['a', 'b', 'c']
>>> l3
[]
>>> for i in l2 :
        l3.append(i.upper())
>>> l3
['A', 'B', 'C']
```

for 문을 활용한 합계 구하기

1.

- for 문을 배우기 전의 방식으로 1~10 의 합계를 구하는 프로그램 만들기

출력 결과

```
1에서 10까지의 합계 : 55
```

2.

0과 100 사이에 있는 7의 배수 합계를 구하도록 수정해 보자.

출력 결과

```
0과 100 사이에 있는 7의 배수 합계 : 735
```

풀이

```
#1
hap = 0
for i in range(1, 11) :
    hap = hap + i

print("1에서부터 10까지의 합계 : %d" % (hap))

#2
baesu = 0
for i in range(0, 101, 7) :
    baesu = baesu + i

print("0과 100 사이에 있는 7의 배수 합계 : {0}".format(baesu))
```

#7 의 배수만 하려면 0 / 7 로 시작해야 함!

cf. if 문으로도 풀어본 2 번

```
baesu=0
for i in range(0,101) :
    if i % 7 == 0 :
        baesu = baesu + i
```

속도면에서 range()를 사용한 것과 큰 차이는 안 나지만, 모든 i(1~100)를 사용하기 때문에 쪼금 느리고 좋지 않은 표현식.

3.

- 예: 시작값과 끝값, 증가값까지 사용자 입력

출력 결과

```
시작값을 입력하세요 : 2
끝값을 입력하세요 : 300
증가값을 입력하세요 : 3
2에서 300까지 3씩 증가시킨 값의 합계 : 15050
```

```
v1 = int(input("시작 값을 입력하세요 : "))
v2 = int(input("끝 값을 입력하세요 : "))
v3 = int(input("증가할 값을 입력하세요 : "))
hap1 = 0

for i in range(v1, v2+1, v3) :
    hap1 = hap1 + i

print("{0}에서 {1}까지 {2}씩 증가시킨 값의 합계 : {3}".format(v1, v2, v3, hap1))
```

★★★★★★★★★ range의 끝 범위 = 끝+1 꼭 신경쓰기!!!!!!! ★★★★★★★★★★★★

중첩 for 문

중첩 for 문의 기본 형식

```
for i in range (0, 3, 1) :
    for k in range(0,2,1) :
        print("파이썬은 꿀잼입니다. ^^ (i값 : %d, k값 : %d)" % (i,k))
```

출력결과

파이썬은 꿀잼입니다. ^^ (i 값 : 0, k 값 : 0)
파이썬은 꿀잼입니다. ^^ (i 값 : 0, k 값 : 1)
파이썬은 꿀잼입니다. ^^ (i 값 : 1, k 값 : 0)
파이썬은 꿀잼입니다. ^^ (i 값 : 1, k 값 : 1)
파이썬은 꿀잼입니다. ^^ (i 값 : 2, k 값 : 0)
파이썬은 꿀잼입니다. ^^ (i 값 : 2, k 값 : 1)

 # 외부변수보다 내부변수 먼저 처리!

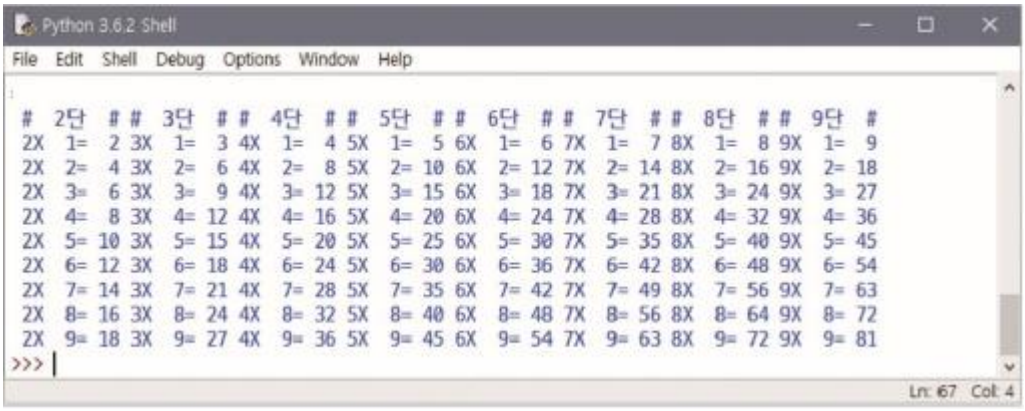
- 처리 순서
  - 외부 변수인 i는 계속 0, 1, 2로 변경된 후 끝나지만, 내부 변수인 k는 0과 1을 계속 반복

❶ 외부 for 문 1회 : i에 0을 대입
내부 for 문 1회 : k에 0을 대입 후 print() 함수 수행
내부 for 문 2회 : k에 1을 대입 후 print() 함수 수행

❷ 외부 for 문 2회 : i에 1을 대입
내부 for 문 1회 : k에 0을 대입 후 print() 함수 수행
내부 for 문 2회 : k에 1을 대입 후 print() 함수 수행

❸ 외부 for 문 3회 : i에 2를 대입
내부 for 문 1회 : k에 0을 대입 후 print() 함수 수행
내부 for 문 2회 : k에 1을 대입 후 print() 함수 수행

구구단 만들기




```
for a in range(2, 10) :
    print("# {0}단 #".format(a), end="₩t₩t")
print("")
for i in range(1,10) :
    for j in range(2, 10) :
        result = j * i
        print("%d x %d = %d" % (j, i, result), end="₩t")
    print("")
```

```
# 2단 #        # 3단 #        # 4단 #        # 5단 #        # 6단 #        # 7단 #        # 8단 #        # 9단 #
2 x 1 = 2      3 x 1 = 3      4 x 1 = 4      5 x 1 = 5      6 x 1 = 6      7 x 1 = 7      8 x 1 = 8      9 x 1 = 9
2 x 2 = 4      3 x 2 = 6      4 x 2 = 8      5 x 2 = 10     6 x 2 = 12     7 x 2 = 14     8 x 2 = 16     9 x 2 = 18
2 x 3 = 6      3 x 3 = 9      4 x 3 = 12     5 x 3 = 15     6 x 3 = 18     7 x 3 = 21     8 x 3 = 24     9 x 3 = 27
2 x 4 = 8      3 x 4 = 12     4 x 4 = 16     5 x 4 = 20     6 x 4 = 24     7 x 4 = 28     8 x 4 = 32     9 x 4 = 36
2 x 5 = 10     3 x 5 = 15     4 x 5 = 20     5 x 5 = 25     6 x 5 = 30     7 x 5 = 35     8 x 5 = 40     9 x 5 = 45
2 x 6 = 12     3 x 6 = 18     4 x 6 = 24     5 x 6 = 30     6 x 6 = 36     7 x 6 = 42     8 x 6 = 48     9 x 6 = 54
2 x 7 = 14     3 x 7 = 21     4 x 7 = 28     5 x 7 = 35     6 x 7 = 42     7 x 7 = 49     8 x 7 = 56     9 x 7 = 63
2 x 8 = 16     3 x 8 = 24     4 x 8 = 32     5 x 8 = 40     6 x 8 = 48     7 x 8 = 56     8 x 8 = 64     9 x 8 = 72
2 x 9 = 18     3 x 9 = 27     4 x 9 = 36     5 x 9 = 45     6 x 9 = 54     7 x 9 = 63     8 x 9 = 72     9 x 9 = 81
```

 # 어디를 먼저 내부 for 문으로 쓸 지 결정해야 함 !

강사님 - 띄어쓰기 맞추기 ~ %2d 로 표현!

```
for i in range(1, 10) :
    for j in range(2, 10) :
        result = j * i
        print("%d x %d = %2d" % (j, i, result), end="  ")

    print("")
```

```
2 x 1 =  2  3 x 1 =  3  4 x 1 =  4  5 x 1 =  5  6 x 1 =  6  7 x 1 =  7  8 x 1 =  8  9 x 1 =  9
2 x 2 =  4  3 x 2 =  6  4 x 2 =  8  5 x 2 = 10  6 x 2 = 12  7 x 2 = 14  8 x 2 = 16  9 x 2 = 18
2 x 3 =  6  3 x 3 =  9  4 x 3 = 12  5 x 3 = 15  6 x 3 = 18  7 x 3 = 21  8 x 3 = 24  9 x 3 = 27
2 x 4 =  8  3 x 4 = 12  4 x 4 = 16  5 x 4 = 20  6 x 4 = 24  7 x 4 = 28  8 x 4 = 32  9 x 4 = 36
2 x 5 = 10  3 x 5 = 15  4 x 5 = 20  5 x 5 = 25  6 x 5 = 30  7 x 5 = 35  8 x 5 = 40  9 x 5 = 45
2 x 6 = 12  3 x 6 = 18  4 x 6 = 24  5 x 6 = 30  6 x 6 = 36  7 x 6 = 42  8 x 6 = 48  9 x 6 = 54
2 x 7 = 14  3 x 7 = 21  4 x 7 = 28  5 x 7 = 35  6 x 7 = 42  7 x 7 = 49  8 x 7 = 56  9 x 7 = 63
2 x 8 = 16  3 x 8 = 24  4 x 8 = 32  5 x 8 = 40  6 x 8 = 48  7 x 8 = 56  8 x 8 = 64  9 x 8 = 72
2 x 9 = 18  3 x 9 = 27  4 x 9 = 36  5 x 9 = 45  6 x 9 = 54  7 x 9 = 63  8 x 9 = 72  9 x 9 = 81
```

while 문

for 문과 while 문 비교

while 문의 형식

```
변수 = 시작값
while 변수 < 끝값 :
    _______ # 이 부분을 반복
    변수 = 변수 + 증가값
```

while : **조건식**이므로 시작값 없음, 주어져야 함. + 변수를 스스로 추가시키지 못하므로 변수 증가값도 지정해줘야 함.

cf.

for 문은 반복할 횟수를 range() 함수에서 결정 후 그 횟수만큼 반복,
while 문은 반복 횟수를 결정하기보다는 조건식이 참일 때 반복하는 방식

예) : 문장을 3 회 반복하도록 하는 while 문

```
i = 0
while i < 3 :
    print("%d : 안녕하세요? while 문을 공부중입니다. ^^" % i )
    i = i+1
```

출력결과

```
0 : 안녕하세요? while 문을 공부중입니다. ^^
1 : 안녕하세요? while 문을 공부중입니다. ^^
2 : 안녕하세요? while 문을 공부중입니다. ^^
```

 # while 문 - 변수 시작값, 증가값 작성 + 조건식 지정!

예 : while 문으로 1 ~ 10 까지의 합계 구하기

```
i = 1
sum = 0

while i < 11 :
    sum = sum + i
    i = i + 1

print("1부터 10까지의 합계 : %d" % sum)
```

출력결과

1 부터 10 까지의 합계 : 55

무한루프를 하는 while 문

- while 조건식 : 에 들어가는 조건식을 True 로 지정

예시)

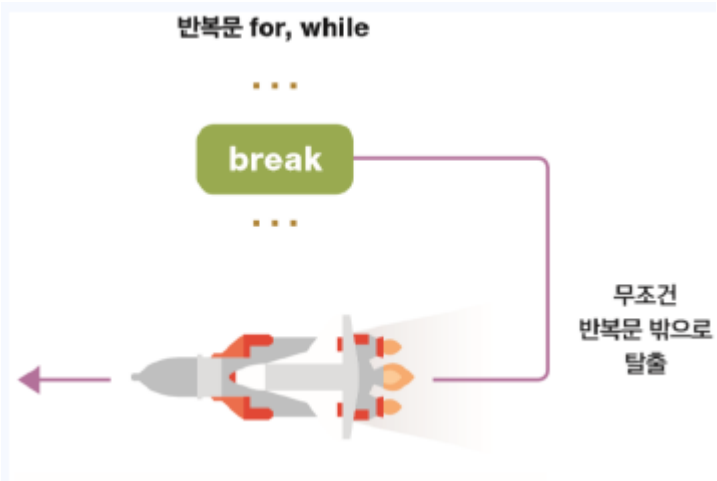
```
while True :
    print("ㅋ ", end=" ")
```

출력결과

ㅋ ㅋ ㅋ ㅋ ㅋ ㅋ ㅋ ㅋ ... 무한반복

break 문과 continue 문

break : 무조건 반복문 밖으로 탈출



cf. exit ~ 프로그램 종료

'$'를 입력하면 while 문을 빠져나가도록 수정해보자.

#while True (while 1): / 잘못된 값이 들어온 경우 로직이 반복되도록 함!

```
while 1 :

    cho = input("1. 입력한 수식 계산 ₩n2. 두 수 사이의 합계 ₩n메뉴 선택 : ")

    if cho=="1" :
        a1 = input("*** 수식을 입력하세요 : ")
        result = eval(a1)
        print("%s 결과는 %s 입니다." % (a1, result))
        break
    elif cho=="2" :
        v1 = int(input("*** 첫 번째 숫자를 입력하세요 : "))
        v2 = int(input("*** 두 번째 숫자를 출력하세요 : "))
        sum = 0

        for num in range(v1, v2+1) :
            sum = sum + num

        print("%d + ... + %d는 %d입니다." % (v1, v2, sum))
        break
    elif cho=="$" :
        break
    else :
        print("입력이 잘못되었습니다.")

print("프로그램 종료")
```

break 를 통해 빠져나가도록 ! else 의 경우에만 무한반복문으로 - !

전체 문장에 while True : 넣어주고, 각각의 경우마다 break 넣어주고 잘못된 값이 입력될 경우에만 반복문 처리되도록 할 수 있음.

누적 합계가 1000 이 되는 시작 지점 알기

1~100 의 합계를 최초로 1000 이 넘게 하는 숫자 : 45

```
sum = 0
for i in range(1, 101) :
    sum = sum + i
    if sum >= 1000 :
        break

print("1~100의 합계를 최초로 10000이 넘게 하는 숫자 : %d" % i)
```

출력결과

1~100 의 합계를 최초로 1000 이 넘게 하는 숫자 : 45

cf. while 로 하면 +1 이 된 경우가 됨 !

for 문으로 반복처리 하다가 if 문 해놓고 정지시키기!

0608

continue 문 : 반복문으로 다시 돌아가게 함

cf. in R - next

 # 해당 조건을 skip 하고 싶을 때!

```
hap = 0

for i in range(1, 101) :
    if i % 3 == 0 :
        continue

    hap += i

print("1~100의 합계(3의 배수 제외) : %d" % hap)
```

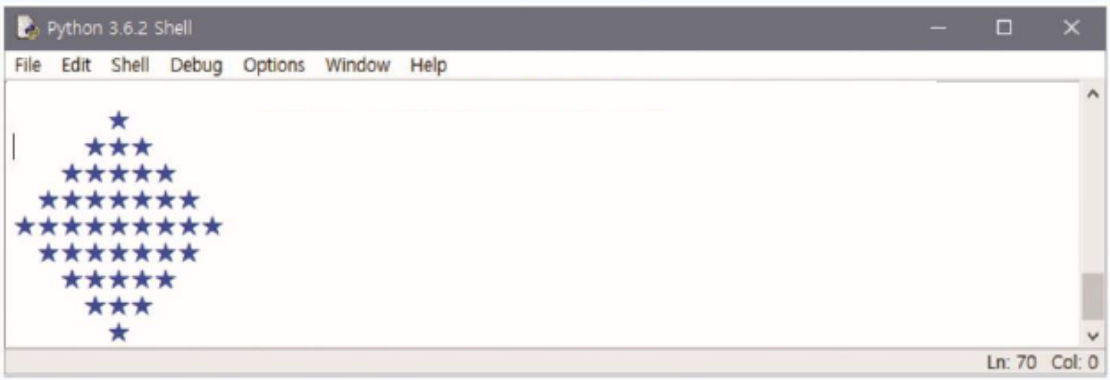
#skip 해야 할 조건이 먼저 나와야 함!

해당 조건이 continue 이후의 문장들을 반복하지 않으므로!! 순서 주의

연습문제

이 장에서 만들 프로그램

- [프로그램 2] 마름모 모양 출력
  - while 문 활용



- [프로그램 2]의 완성
  - 별 모양의 글자 출력하는 코드

```
print('\u2605')
```

  - While 문으로 구현

쫌 복잡하지만 풀긴 풀었다..

```
i=1
while i < 10 :
    if i <= 5 :
        print(" "*(10-2*i) +"#u2605"*(2*i-1))
    else :
        print(" "*(2*i-10) +"#u2605"*(10-(2*(i-5)+1)))

    i=i+1
```

강사님 방법

```
a="#u2605"
b="  "
i=1
while i < 10 :
    if i<6 :
        print(b*(5-i) + a*((2*i)-1))
    else :
        print(b*(i-5) + a*(((10-i)*2)-1))
    i=i+1
```

a 에 별 지정, b 에 공백 두 칸으로 지정 ! 곱셈이 먼저여도 보기 쉽게 괄호 써주기

꽉찬별 2605 안 찬 별 2606

힌트

print('₩u2605'+'₩u2605')

★★

print('₩u2605'*2)

★★

수열 계산

특정부분까지만 증가.. 분기점 필요

각 라인을 i로 표현, i를 통해 공백과 별 개수를 나타내는 방법 고민

5-i 2i-1

특정 조건을 만족했을 때 감소하게끔 가공하기

이중반복문을 쓸 필요는 없음

if :

else : 일케하면 되것다

Chapter 6 리스트, 튜플, 딕셔너리

cf. 리스트 ~ 벡터 in R

딕셔너리 ~ 리스트 in R [key - value 형태로 데이터 생성, key 만으로도 데이터 접근을 쉽게 함]

리스트 - 수정 가능 / 튜플 - 수정 불가능한 리스트

리스트의 기본

리스트의 일반적인 사용

빈리스트의 생성과 항목 추가

```
aa = []
aa.append(0)
aa.append(0)
aa.append(0)
aa.append(0)
print(aa)
```

출력결과

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

```
aa=[]
for i in range(0, 100) :
    aa.append(i)

len(aa)
```

출력결과

```
100
```

리스트의 생성과 초기화

```
aa=[]                          # 빈리스트 생성
bb=[10,20,30]                  # 정수로만 구성된 리스트 생성
cc=["파이썬","공부는","꿀잼"] # 문자열로만 구성된 리스트 생성
dd=[10,20,"파이썬"]            # 다양한 데이터 형을 섞어서 리스트 생성
```

리스트 값에 접근하는 다양한 방법

음수값으로 접근

aa=[10,20,30,40]

print("aa[-1]은 %d, aa[-2]는 %d" % (aa[-1], aa[-2]))

출력결과

aa[-1]은 40, aa[-2]는 30

 # 음수값 : 뒤에서부터 순서대로! -1 부터 차근차근 !! (not 0 부터 유의!)

리스트에 접근할 때 콜론(:)을 사용해 범위를 지정

```
     :     :
 ↑     ↑     ↑
시작  end   by
(시작의 range 와 end range 다르므로 유의해야 함 !)
```

1. 콜론의 앞이나 뒤 숫자 생략

aa=[10, 20, 30, 40]

aa[0:3] = 시작 range 0 부터 2 까지 -- 실제론 1 부터 3 까지

aa[2:] = 시작 range 2 부터 -- 실제론 3 부터 끝까지

aa[:2] =end range 1 까지 -- 실제론 처음부터 2 번째까지

 # 시작의 range 와 end range 다르므로 유의해야 함 !

2. 리스트의 항목 건너뛰며 추출 (by)

aa=[10,20,30,40,50,60,70]

aa[::2] = by 2 의 뜻 ! 0, 2, ...

aa[::-2] = (뒤의)처음에서부터 뒤로 거꾸로 2 개씩 띄어 추출

aa[::-1] = ★★ 역순출력 !!! ★★

3. 파이썬 첫 번째 컬럼 선택 a[: , 0] : = 전체 의미

(in R 첫 번째 컬럼 선택시 a[, 1] in 데이터 프레임(행부분 생략))

---

cf. ★★첫 번째 컬럼만 뽑는 경우 두 가지★★

aa[: ,0] // aa[: ,0:1] for 데이터프레임 형식, 차원축소 방지(2nd)

리스트끼리 덧셈, 곱셈 연산은 데이터 확장 (not 산술연산)

```
aaa=[10,20,30]
bbb=[40,50,60]
aaa+bbb
aaa*3
```

출력결과

```
[10,20,30,40,50,60]
[10,20,30,10,20,30,10,20,30]
```

리스트의 원소별 연산을 하려면 for 문 사용! / array 구조

리스트 값의 변경

두 번째 위치한 값을 1 개 변경하는 방법

```
aa = [10,20,30]
aa[1] = 200
aa
```

출력결과

```
[10, 200, 30]
```

두 번째 값인 20 을 200 과 201 이라는 값 2 개로 변경하는 방법

```
aa = [10,20,30]
aa[1:2] = [200, 201]
aa
```

출력결과

```
[10, 200, 201, 30]
```

슬라이스 색인으로, 특정 위치에 리스트를 삽입해도, 결과는 일반 리스트 !

aa[1:2] 대신 그냥 aa[1]을 사용해서 바꾼 경우

```
aa = [10,20,30]
aa[1] = [200, 201]
aa
```

출력결과

```
[10, [200, 201], 30]
```

정수색인시, 리스트가 중첩된 구조로 나타남...!

슬라이스 / 정수색인에 따라서 표현 결과 다름.

+ 특정 원소만 삭제 가능 (중간원소 삭제) (R 과 다른 점)

두 번째인 aa[1]의 항목 삭제

```
aa = [10,20,30]
del(aa[1])
aa
```

출력결과

```
[10, 30]
```

두 번째인 aa[1]에서 네 번째인 aa[3]까지 삭제

```
aa = [10,20,30,40,50]
aa[1:4] = []      #슬라이스 색인 사용!
aa
```

출력결과

```
[10, 50]
```

리스트 자체를 삭제하는 방법

```
aa=[]             # 빈리스트
aa=None           # 변수 존재 but 리스트 잃음
del(aa)           # 변수 자체를 삭제
```

출력 결과 (aa)

```
[]
```

아무 것도 안 나옴

오류 발생 (변수 자체가 없으므로)

#리스트 중간에 값 추가는 insert method, 마지막 위치에 추가는 append method

----------색인순서를 꼭 유의할 것!

리스트 메소드

메소드	설명

append()	리스트 끝에 새 요소를 추가한다.
	
	>>> a = [1,2,3]
	>>> a.append(4)
	>>> a
	[1,2,3,4]

extend()	기존 리스트에 다른 리스트를 이어 붙인다. + 연산자와 같은 기능
	
	>>> a = [1,2,3]
	>>> a.extend([4,5,6])
	>>> a
	[1,2,3,4,5,6]

insert()	첨자로 명시한 리스트 내의 위치에 새 요소를 삽입한다.
	insert(첨자, 데이터)의 형식으로 사용.
	
	>>> a = [2,4,5]
	>>> a.insert(0,1) #0 위치(첫 번째)에 데이터 1을 삽입함.
	>>> a
	[1,2,4,5]
	>>> a.insert(2,3) #2 위치(세 번째)에 데이터 3을 삽입함.
	>>> a
	[1,2,3,4,5]

remove()	매개 변수로 입력한 데이터를 리스트에서 찾아 발견한 첫 번째 요소를 제거함.
	
	>>> a=['BMW','BENZ','VOLKSWAGEN','AUDI']
	>>> a.remove('BENZ')
	>>> a
	['BMW', 'VOLKSWAGEN', 'AUDI']

remove() -- 위치는 모르지만 특정 값을 삭제하고 싶을 때
cf. del() -- 위치를 통해 삭제함

pop()	리스트의 마지막 요소를 뽑아내어 리스트에서 제거함.
	
	>>> a=[1,2,3,4,5]
	>>> a.pop()
	5
	>>> a
	[1, 2, 3, 4]
	>>> a.pop()
	4
	>>> a
	[1, 2, 3]
	# 마지막이 아닌 특정 요소를 제거하고 싶을 때 pop()에 인덱스 입력
	>>> a=[1,2,3,4,5]
	>>> a.pop(2) # 세 번째 요소 제거
	3
	>>> a
	[1, 2, 4, 5]

index()	리스트 내 매개변수로 입력한 데이터와 같은 첫 번째 요소의 첨자를 알려줌.
	찾고자 하는 데이터와 일치하는 요소가 없으면 오류를 일으킴.
	
	>>> a=['abc','def','ghi']
	>>> a.index('def')
	1
	>>> a.index('jkl')
	Traceback (most recent call last):
	  File "<pyshell#61>", line 1, in <module>
	  a.index('jkl')
	ValueError: 'jkl' is not in list

count()	매개변수로 입력한 데이터와 일치하는 요소가 몇 개 존재하는지 센다.
	
	>>> a=[1,100,2,100,3,100]
	>>> a.count(100)
	3
	>>> a.count(200)
	0

index () 해당하는 요소가 없으면 오류를 일으킴 (다른 함수/메소드와 다른 점)

count - 특정 데이터와 일치하는 요소가 몇 개 있는지 (빈도수)

-> **문자열 확인 시 count()나 in 연산자 사용이 더 바람직.** (index()는 오류발생)

/ count=0 이면 없다는 결론

sort()	리스트 내의 요소를 정렬한다. 매개변수로 reverse=True를 입력하면 내림차순, 아무 것도 입력하지 않으면 오름차순으로 정렬함.
	
	>>> a=[3,4,5,1,2]
	>>> a.sort()
	>>> a
	[1, 2, 3, 4, 5]
	>>> a.sort(reverse=True)
	>>> a
	[5, 4, 3, 2, 1]

reverse()	리스트 내 요소의 순서를 반대로 뒤집음 (like [::-1] 색인)
	
	>>> a=[3,4,5,1,2]
	>>> a.reverse()
	>>> a
	[2, 1, 5, 4, 3]
	>>> b=['안','녕','하','세','요']
	>>> b.reverse()
	>>> b
	['요', '세', '하', '녕', '안']

sort 는 재정렬한 값을 출력해줌 (in R) cf. decreasing, descending, reverse ... 문법 차이
데이터 타입별 sort 가 다를 수 있다.

리스트메소드 ~ for 문으로 해당 변수 해야 함.

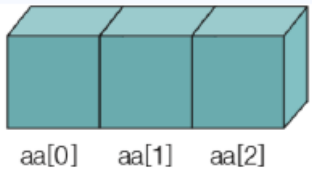
True - method 마다 매개변수 전달이 다름..!)! (T 나 True 나 TRUE 나 ...)

2 차원 리스트

2 차원 리스트의 개념

- 1 차원 리스트를 여러 개 연결한 것, 첨자를 2 개 사용

```
aa = [10, 20, 30]
```




```
aa = [[1, 2, 3, 4],
      [5, 6, 7, 8],
      [9, 10, 11, 12]]
```

예 : 중첩 for 문을 사용, 3 행 4 열짜리 리스트 생성 후 항목 1 ~ 12 를 입력하고 출력

```
list1=[]
list2=[]
value=1
for i in range(0, 3) :
    for k in range(0, 4) :
        list1.append(value)
        value +=1
    list2.append(list1)
    list1=[]

for i in range (0, 3) :
    for k in range(0, 4) :
        print("%3d" % list2[i][k], end= " ")
    print("")
```

#1,2 행 - 1 차원 리스트로 사용할 list1 과 2 차원 리스트로 사용할 list2 준비

#3 행 - value 는 리스트에 입력할 1 ~ 12 의 값으로 반복할 변수

4 ~ 9 행 - 리스트의 행 단위 만들기 위해 3 회 반복

5 ~ 7 행 - 4 회 반복해 항목이 4 개인 1 차원 리스트 생성, 처음엔 [1, 2, 3, 4] 만듦

8 행 - 2 차원 리스트에 추가

9 행 - 1 차원 리스트를 다시 비움

11 ~ 14 행 - 2 차원 리스트 출력

13 행 - '리스트명[행][열]' 방식으로 각 항목 출력

14 행 - 행별 출력 위해 (end = " " 남아있어서) print("")사

출력결과

1 2 3 4
5 6 7 8
9 10 11 12

◆ k 를 사용하지 않고 value 를 통해 값을 설정한 이유?

k in range (1,5)일 경우 i=0, 1, 2 일 때 모두 list1=[1,2,3,4]로만 반복되기 때문!
다음 행은 증가된 값이어야 하는데!
따라서 반복변수가 아닌 임의변수로 값을 할당하고 계속 값을 키운 것!

◆ 9th 문장 : 리스트 초기화 위해 필요

#####쓰지 않았을 때 !#################
i=0 ->list1=[1,2,3,4]
i=1 ->list1=[1,2,3,4,5,6,7,8]
i=2 ->list1=[1,2,3,4,5,6,7,8,9,10,11,12]

두 번째 for 문의 결과

1 2 3 4
1 2 3 4
1 2 3 4
####################################

.

연습문제

4행 5열의 2차원 리스트를 만들고, 0부터 3의 배수를 입력하고 출력하도록 수정해 보자.
는 다음과 같다.

출력 결과

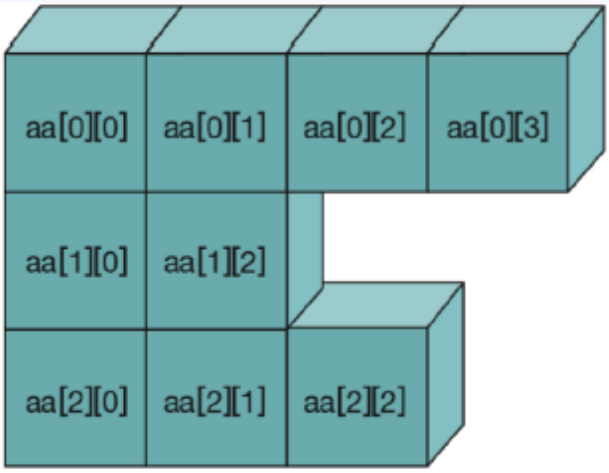
```
 0  3  6  9 12
15 18 21 24 27
30 33 36 39 42
45 48 51 54 57
```

```
list1=[]
list2=[]

value=0
for i in range(0, 4) :
    for k in range(0, 5) :
        list1.append(value)
        value +=3
    list2.append(list1)
    list1=[]

for i in range (0, 4) :
    for k in range(0, 5) :
        print("%3d" % list2[i][k], end= " ")
    print("")
```

1 차원인데 내부 리스트를 만든 것임 !-!
그래서 아래 for 문 통해서 list2[i][k] (행의 행 선택) 2 차원처럼 출력

▪ 불규칙한 크기의 2차원 리스트

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




```
list1=[]
list2=[]

v1=[4,2,3]

value=1
for i in range(0, len(v1)) :
    for k in range(0,v1[i]) :
        list1.append(value)
        value +=1
    list2.append(list1)
    list1=[]


for i in range (0, len(v1)) :
    for k in range(0,v1[i]) :
        print("%3d" % list2[i][k], end= " ")
    print("")
```

힌트였다!

행마다 다른 개수 ! 풀어보기

```
>>> l1=[[1,2,3,4],[1,2],[1,2,3]]
>>> l1[0]
[1, 2, 3, 4]
>>> len(l1)
3
>>> len(l1[0])
4
```

for 문을 적절히 활용하기 - for 문에 숫자를 len 을 활용한 변수로 바꾸기

프로그램 짜기

```
num=1
b=[]

while True :
    a = input("%d번째 생성할 리스트 수를 정해주세요 : " % num)
    if a == "z" :
        break

    num += 1
    b.append(int(a))

l1=[]
l2=[]
value=1
for i in range(0,len(b)) :
    for k in range(0, b[i]) :
        l1.append(value)
        value +=1
    l2.append(l1)
    l1=[]

for i in range(0, len(b)) :
    for k in range(0, b[i]) :
        print("%3d" % l2[i][k], end=" ")
    print("")
```

종료코드 : z

in for 문

0 부터 시작하고 싶음 -> value=0 바꾸기 / 3 씩 증가하고 싶음 -> value=3 으로 바꾸기

강사님 방법

```
i=1
value=0
list1=[]
list2=[]
while 1 :
    no = input("%d 번째 생성할 리스트 수를 입력하세요 : " % (i))
    if no == "z" :
        print ("입력을 종료하고 출력하겠습니다. ₩n")
        break
    for a in range (0, int(no)) :
        list1.append(value)
        value = value + 3
    list2.append(list1)
    list1=[]
    i=i+1

for a in range(0, len(list2)) :
    for b in range(0, len(list2[a])) :
        print("%3d" % (list2[a][b]), end="")
    print("")
print("")
```

아래 있는 for 문에서 for a in range(0, i-1) : 로 해도 됨 ! / while 문 - i+1 이 마지막이어서 잘 생각!

180611

튜플

튜플

- 리스트와 동일하나 수정이 안 됨(immutable)
- 대용량 데이터 set, 원본 데이터인 경우 튜플로 설정해두면 실수 예방 가능

튜플의 생성

- 리스트는 대괄호로 생성 [] , 튜플은 소괄호로 생성 ()
- 읽기 전용 자료를 저장할 때 사용
- 튜플은 소괄호()를 생략 가능, 항목이 하나인 튜플은 뒤에 쉼표를 꼭 붙여야 함 !

```
>>> tt1=(10,20,30) ; tt1
(10, 20, 30)
>>> tt2 = 10, 20, 30 ; tt2
(10, 20, 30)
>>> type(tt1) ; type(tt2)
<class 'tuple'>
<class 'tuple'>
>>> tt3 = (10) ; type(tt3)
<class 'int'>
>>> tt4 = 10 ; type(tt4)
<class 'int'>
>>> tt5 = (10, ) ; type(tt5
                        )
<class 'tuple'>
>>> tt6 = 10, ; type(tt6)
<class 'tuple'>
```

cf. tt5, tt6 개수 1 개 ! len 으로 원소 개수 확인

```
>>> len(tt5) ; len(tt6)
1
1
```

튜플 원소 수정/삭제는 불가능

튜플 삭제

- 튜플의 원소가 아닌, 튜플 자체 삭제는 가능

del(tt1)

del(tt2)

튜플의 사용 - 색인방식 / 리스트와 동일

튜플 항목에 접근

```
>>> tt1=(10,20,30,40)
>>> tt1[0]
10
>>> tt1[0]+tt1[1]+tt1[2]
60
```

튜플 범위에 접근

```
>>> tt1[1:3]
(20, 30)
>>> tt1[1:]
(20, 30, 40)
>>> tt1[:3]
(10, 20, 30)
>>> tt1[::2]
(10, 30)
```

튜플의 덧셈 및 곱셈 연산 : 데이터의 확장 ! (not 산술연산)

```
>>> tt2=('A', 'B')
>>> tt1 + tt2
(10, 20, 30, 40, 'A', 'B')
>>> tt2 * 3
('A', 'B', 'A', 'B', 'A', 'B')
```

연습문제

다음과 같이 2차원 튜플을 생성한 후 모든 값을 출력해 보자.

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

출력 결과

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

```
tt1=((1,2,3),
     (4,5,6),
     (7,8,9))


for i in range(0, 3) :
    for j in range (0, 3) :
        print(tt1[i][j], end=" ")
    print("")
```

튜플의 수정

불가피하게 수정해야 할 경우 **튜플 -> 리스트 -> 튜플 순으로 변환**

```
>>> myTuple = (10,20,30)
>>> myList = list(myTuple)
>>> myList.append(40)
>>> myTuple = tuple(myList)
>>> myTuple
(10, 20, 30, 40)
```

튜플의 패킹과 언패킹 (리스트도 사용 가능)

#패킹 : 여러 데이터를 튜플로 묶는 것

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

#언패킹 : 각 요소를 여러 개의 변수에 할당하는 것.

```
>>> one, two, three = a
>>> one
1
>>> two
2
>>> three
3
```

#뒤에 나와 있는 변수의 담긴 개수만큼 앞에 변수를 쓰면 알아서 개수만큼 할당됨

cf. 튜플의 원소는 수정 불가, but 튜플의 원소가 리스트일 때 리스트 원소는 수정 가능

```
In [9]: A=([1,2,3,4],)
        A[0][1:]=[5,6,7]
        A
Out[9]: ([1, 5, 6, 7],)
```

딕셔너리

딕셔너리의 개념 cf. 다른 프로그래밍 언어- hash, associative array 라 함.

딕셔너리 변수 = {키 1:값 1, 키 2:값 2, 키 3:값 3, ...}

- 쌍 두 개가 하나로 묶인 자료구조

ex. apple : 사과처럼 의미있는 두 값을 연결해 구성

- 중괄호 **{}**로 묶어 구성, 키, 값의 쌍으로 구성
- 사전의 키는 수정할 수 없고(삭제는 가능), 값만 수정 가능

딕셔너리의 생성

```
>>> dic1 = {1 : 'a' , 2 : 'b', 3 : 'c'} ; dic1
{1: 'a', 2: 'b', 3: 'c'}
>>> dic2 = {'a' : 1 , 'b' : 2 , 'c' : 3} ; dic2
{'a': 1, 'b': 2, 'c': 3}
```

- 키와 값 : 사용자가 지정
- **딕셔너리는 생성 순서대로 구성된다는 보장이 없음.**

여러 정보의 딕셔너리 표현

키	값
학번	1000
이름	홍길동
학과	컴퓨터학과

```
>>> student1 = {'학번' : 1000, '이름' : '홍길동', '학과' : '컴퓨터학과'} ; student1
{'학번': 1000, '이름': '홍길동', '학과': '컴퓨터학과'}
```

student1 에 연락처 추가

```
>>> student1['연락처'] = '010-1111-2222' ; student1
{'학번': 1000, '이름': '홍길동', '학과': '컴퓨터학과', '연락처': '010-1111-2222'}
```

학과 수정

```
>>> student1['학과'] = '파이썬학과' ; student1
{'학번': 1000, '이름': '홍길동', '학과': '파이썬학과', '연락처': '010-1111-2222'}
```

student1 의 학과 삭제

```
>>> del(student1['학과']) ; student1
{'학번': 1000, '이름': '홍길동', '연락처': '010-1111-2222'}
```

동일한 키를 갖는 딕셔너리를 생성하지 않고, 마지막에 있는 키만 적용

```
>>> student1 = {'학번' : 1000, '이름' : '홍길동', '학과' : '파이썬학과' , '학번' : 2000} ; student1
{'학번': 2000, '이름': '홍길동', '학과': '파이썬학과'}
```

딕셔너리의 사용

키로 값에 접근하는 코드

```
student1['학번']
student1['이름']
student1['학과']
```

출력결과

```
2000
'홍길동'
'파이썬학과'
```

딕셔너리명.get(키) 함수를 이용해 키로 값에 접근

```
student1.get('이름')
```

출력결과

```
'홍길동'
```

- 딕셔너리명[키]와 딕셔너리명.get(키)의 결과는 같다.
- 딕셔너리명[키]는 없는 키 호출시 오류 생성 /
  딕셔너리명.get(키)는 없는 키 호출시 아무 것도 반환하지 않음.
- 순서로 인덱싱 불가, 순서가 없으므로 슬라이싱도 불가능
- **따라서 없는 키를 찾을 때를 대비해서 딕셔너리명.get(키) 사용 권장 !**

#없는 키 호출시 딕셔너리명[키]와 딕셔너리명.get(키)의 차이점

```
>>> student1['주소']
Traceback (most recent call last):
  File "<pyshell#37>", line 1, in <module>
    student1['주소']
KeyError: '주소'

>>> student1.get('주소')
```

딕셔너리명.keys() - 딕셔너리의 모든 키 반환

```
student1.keys()
```

출력결과

```
dict_keys(['학번', '이름', '학과'])
```

list(딕셔너리명.keys()) 함수 사용 -> 출력결과 dict_keys 안 보이게

```
# list 구조를 활용해서 for 문 접근 ~ 연산에 필요
list(student1.keys())
```

출력결과

```
['학번', '이름', '학과']
```

딕셔너리명.values() - 딕셔너리의 모든 값을 리스트로 만들어 변환

```
student1.values()
```

출력결과

```
dict_values([2000, '홍길동', '파이썬학과'])
```

list(딕셔너리명.values()) 함수 사용 -> 출력결과 dict_values 안 보이게

```
list(student1.values())
```

출력결과

```
[2000, '홍길동', '파이썬학과']
```

딕셔너리명.items() 함수 사용 -> 튜플 형태로 구할 수 있음.

```
student1.items()
```

출력결과

```
dict_items([('학번', 2000), ('이름', '홍길동'), ('학과', '파이썬학과')])
```

딕셔너리 안에 해당 키가 있는지 없는지 in 을 사용해 확인 / True, False 로 반환

'이름' in student1

'주소' in student1

출력결과

True

False

for 문을 활용해 딕셔너리의 모든 값을 출력하는 코드

```
singer={}

singer['이름'] = '트와이스'
singer['구성원 수'] = 9
singer['데뷔'] = '서바이벌 식스틴'
singer['대표곡'] = 'SIGNAL'

for k in singer.keys() :
    print('%s --> %s' % (k, singer[k]))
```

출력결과

```
이름 --> 트와이스
구성원 수 --> 9
데뷔 --> 서바이벌 식스틴
대표곡 --> SIGNAL
```

딕셔너리에 딕셔너리 합하기 # .update()

```
In [61]: dict1={'서울':'02','경기':'031','세종':'044'}
         dict2={'세종':'044','제주':'064'}
         dict1.update(dict2)
         dict1
Out[61]: {'서울': '02', '경기': '031', '세종': '044', '제주': '064'}
```

컴프리헨션을 사용한 딕셔너리 만들기

```
In [62]: ls=['서울','경기','인천']
         dic={x:ls[x] for x in range(3)}
         dic
Out[62]: {0: '서울', 1: '경기', 2: '인천'}
```

딕셔너리의 정렬

- 키로 정렬한 후 딕셔너리 추출

```
import operator
trainDic, trainList = {}, []
trainDic = {'Thomas' : '토마스', 'Edward' : '에드워드', 'Henry' : '헨리', 'Gothen' : '고든', 'James' : '제임스' }
trainList = sorted(trainDic.items(), key = operator.itemgetter(0))
print(trainList)
```

#1 행 - import operator // 7 행의 **operator.itemgetter()**함수를 사용하려고 임포트

#3 행 - 빈 딕셔너리, 리스트 준비

#5 행 - 딕셔너리 작성

#7 행 1- 딕셔너리변수.items() -- 튜플형태로 (키,밸류) 리스트 형태로 나열

#7 행 2 - **키를 기준으로** 딕셔너리 정렬 // .items () - key + value 튜플형태 정렬

operator.itemgetter(0) - 0 :key 값 순

(1) - 1: value 값 순서

출력 결과

```
[('Edward', '에드워드'), ('Gothen', '고든'), ('Henry', '헨리'), ('James', '제임스'),
('Thomas', '토마스')]
```

cf. for k in singer.keys() sort -> ~~ for 문으로 호출해서 사용도 가능 함!

보통 dic.items() + operator.itemgetter(0/1) 형태로 쓰는가봄!!!!

리스트, 튜플, 딕셔너리의 심화 내용

세트

- 출력되는 값은 key 만 남은 딕셔너리 형태

- 딕셔너리의 키는 중복되면 안 되므로, 세트에 들어있는 값은 항상 유일한 원소들
- 집합에는 immutable 한 자료형만 저장이 가능(정수, 실수, 복소수, bool, 문자열, 튜플만 저장 가능)
- 세트를 생성하려면 딕셔너리처럼 중괄호 {}를 사용하지만, :없이 값을 입력

- 리스트 등의 원소 중복을 확인할 때에도 사용함

- 집합은 순서의 개념이 없기 때문에 인덱싱과 슬라이싱 불가능
- 집합은 mutable 객체. 인덱스와 슬라이싱을 사용할 수 없기 때문에 집합의 메서드를 사용
- **공집합**은 {}가 아닌, **set()**으로 생성
- 집합.sorted()의 결과는 리스트

```
mySet1={1,2,3,3,3,4} ; mySet1
```

출력결과

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

판매된 물품의 전체 수량이 아닌, 종류만 파악하고 싶을 때

```
salesList = ['삼각김밥', '바나나', '도시락', '삼각김밥', '삼각김밥', '도시락', '삼각김밥'];set(salesList)
```

출력결과

```
{'바나나', '삼각김밥', '도시락'}
```

두 세트 사이 교집합, 합집합, 차집합, 대칭 차집합을 구할 때

```
mySet1={1,2,3,4,5} ; mySet2={4,5,6,7}
mySet1 & mySet2     # 교집합
mySet1 | mySet2     # 합집합
mySet1 - mySet2     # 차집합
mySet1 ^ mySet2     # 대칭 차집합(배타적 차집합)
```

출력결과

```
{4, 5} / {1, 2, 3, 4, 5, 6, 7} / {1, 2, 3} / {1, 2, 3, 6, 7}
```

연산자 &, |, -, ^ 대신 메서드 사용

```
mySet1.intersection(mySet2) # 교집합
mySet1.union(mySet2)        # 합집합
mySet1.difference(mySet2)   # 차집합
mySet1.symmetric_difference(mySet2) # 대칭 차집합
```

+추가 ; True / False 반환

mySet1 <= mySet2 # 부분집합 - 왼쪽이 오른쪽의 부분집합인지 조사
mySet1 < mySet2 # 진부분집합 - 부분집합이면서 여분의 원소가 더 있음
mySet1 >= mySet2 # 포함(상위)집합 - 왼쪽이 오른쪽 집합을 포함하는지 조사
mySet1 > mySet2 # 진포함(상위)집합 - 포함집합이면서 여분의 원소가 더 있음
mySet1.issubset(mySet2) # 부분집합
mySet1.issuperset(mySet2) # 포함집합
mySet1.isdisjoint(mySet2) # 서로소

cf. frozenset 자료형
- 집합의 일종이지만 고정된 집합
- immutable 자료형으로 수정 및 삭제 불가.

컴프리헨션★★★★★ (삼항다항식)

리스트 = [수식 / for 항목 in range() / if 조건식]
(세 부분으로 이해 !)

- 최종 출력값의 형태를 변수로 정의, for 문 이전과 이후로 해석하면 된다.
- **값이 순차적인 리스트를 한 줄로 만드는 간단한 방법**
- for 문을 이용한 리스트의 생성
- **in 딥러닝**, 반복되는 변수 관련 중요 부분

ex. 1 - 5 까지 저장된 리스트

```
------for 문 사용 -----------
numlist = []
for num in range (1,6) :
    numlist.append(num)
numlist
```

출력결과

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

```
--------컴프리헨션으로 작성---------
numlist = [num for num in range(1,6)]
numlist
```

출력결과

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

```
.-------------------1~5 의 제곱으로 구성된 리스트-----
numList=[num*num for num in range(1,6)]
numList
```

출력결과

```
[1, 4, 9, 16, 25]
```

```
---------------- 1~20 숫자 중 3 의 배수로만 리스트를 구성
numList=[num for num in range (1,21) if num % 3 == 0]
numList
```

출력결과

```
[3, 6, 9, 12, 15, 18]
```

동시에 여러 리스트에 접근

- zip()를 사용해 동시에 여러 리스트에 접근

```
foods=['떡볶이','짜장면','라면','피자','맥주','치킨','삼겹살']
sides=['오뎅','단무지','김치']
for food, side in zip (foods, sides) :
    print(food, ' --> ', side)
```

출력결과

떡볶이 --> 오뎅

짜장면 --> 단무지

라면 --> 김치

zip 함수(in R mapply 참고! 동일하진 않지만 여러 항목을 입력한다는 면에서 비슷한 기능)

zip 함수를 이용해서 for 문에 여러 항목 입력 가능

--매칭되지 않으면 출력 X

-두 리스트를 튜플이나 딕셔너리로 짝지을 때 zip()사용

```
foods=['떡볶이','짜장면','라면','피자','맥주','치킨','삼겹살']
sides=['오뎅','단무지','김치']
tupList = list(zip(foods, sides))
dic = dict(zip(foods, sides))
tupList
dic
```

출력결과

[('떡볶이', '오뎅'), ('짜장면', '단무지'), ('라면', '김치')]

{'떡볶이': '오뎅', '짜장면': '단무지', '라면': '김치'}

 # dict 사용시 앞에 있는 게 key 값, 뒤가 value 값이므로 순서 잘 주의해서 쓰기 !

#zip()을

- list()로 묶으면 튜플 형태의 결과가 출력됨
- dict()로 묶으면 딕셔너리 형태의 결과가 출력됨

a=sorted(dic.items(), operator.itemgetter)를 통해 튜플 리스트로 정리한 후

첫 번째 원소만(foods) 따로 분리해서 저장...

첫 번째 원소끼리 정렬된 형태로 저장~0~!!!! for 문 쓰면 되겠다.

dict 구조로는 dict 과 zip 함수를 통해 저장 가능

리스트의 복사

얕은 복사

- default for 속도! (sorting, 연산 등을 위한 메모리 자원 남기기..!)
- newList=oldList 는 newList 와 oldList 가 동일한 메모리 공간 공유

깊은 복사 deep copy

- 얕은 복사의 방지
- newList = oldList[:]는 메모리의 공간을 복사해서 새로 만듦

```
In [90]: oldList=['짜장면','탕수육']
         newList=oldList #얕은 복사
         newList2=oldList.copy() #깊은 복사
         oldList[0]='짬뽕'
         oldList.append('깐풍기')
         print('old list={0}₩t₩t, ID={1}'.format(oldList,id(oldList)))
         print('new list={0}(얕은 복사), ID={1}'.format(newList, id(newList)))
         print('new list2={0}(깊은 복사)₩t, ID={1}'.format(newList2, id(newList2)))

         old list=['짬뽕', '탕수육', '깐풍기']          , ID=1324565002760
         new list=['짬뽕', '탕수육', '깐풍기'](얕은 복사), ID=1324565002760
         new list2=['짜장면', '탕수육'](깊은 복사)       , ID=1324566588616
```

(mutable 한 객체 -> .copy()로 ID 값 다르게 지정! 기본값 : 얕은 복사)

실습문제

- **[프로그램] 딕셔너리를 활용한 음식 궁합 출력**
  - 딕셔너리를 활용해 음식 궁합을 출력하는 프로그램


```
foods=['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살']
sides=['오뎅', '단무지', '김치', '피클', '소세지', '치킨무', '김치']

fsdic=dict(zip(foods,sides))

while 1 :
    v1 = input("{0} 중 좋아하는 음식은?".format(foods))
    if v1 in foods :
        print("<%s> 궁합 음식은 <%s> 입니다." % (v1, fsdic[v1]))
    elif v1 == "끝" :
        break
    else : print("그런 음식이 없습니다. 확인해 보세요.")
```

fsdic[v1] -> fsdic.get(v1)로 바꿔도 된다~~~

.get( ) 오류 발생 X / [ ] 없는 키 있으면 오류 발생

강사님 방법

```
foods=['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살']
sides=['오뎅', '단무지', '김치', '피클', '소세지', '치킨무', '김치']

fsdic=dict(zip(foods,sides))

while 1 :
    v1 = input("%s 중 좋아하는 음식은?(종료 : z) " % list(fsdic.keys()))
    if fsdic.get(v1) :
        print("<%s> 궁합 음식은 <%s> 입니다." % (v1, fsdic[v1]))
    elif v1 == "z" :
        print("프로그램을 종료합니다.")
        break
    else :
        print("그런 음식이 없습니다. 확인해보세요.")
```

in 연산자를 쓰지 않더라도, fsdic.get(v1) 에서 True / False 나뉘기 때문에 ! 사용함

```
['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살'] 중 좋아하는 음식은?치킨
<치킨> 궁합 음식은 <치킨무> 입니다.
['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살'] 중 좋아하는 음식은?라면
<라면> 궁합 음식은 <김치> 입니다.
['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살'] 중 좋아하는 음식은?짬뽕
그런 음식이 없습니다. 확인해 보세요.
['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살'] 중 좋아하는 음식은?끝
```

0612

Chapter 07 함수와 모듈

함수 기본

함수의 개념과 필요성

- 함수 : 반드시 값을 돌려주는 ! return 을 쓰지 않으면 함수 실행 시 객체 생성 안 됨.
- 함수와 메서드의 차이점
  1. 인자 전달방식 ~ 자료 구조
     함수 : 괄호 안에
     메서드 : 전달대상을 .메서드 앞에 (. 쓰고 조금 기다리면 그 타입에 맞는 메소드 나타남)
  2. 저장하는 방식
     함수 : 외부에 별도로 존재
     메서드 : 특정 클래스를 호출해야 가능함, 클래스 안에 존재

cf. method -- help 로 검색하기 (like in R)

- help(l1.copy)처럼 실제 쓰이는 형태를 통해 검색하면 도움말 뜸!
- help(copy)만 하면 안 뜸

함수 정의하기

def 함수이름(매개변수 목록) :

#코드블록

return 결과

return 바로 변수 쓰면 됨 ! - !

실습 1 (함수 정의)

```
def my_abs(arg) :
    if(arg < 0) :
        result=arg * -1
    else :
        result = arg

    return result
```

plus() 함수

```
def plus(v1, v2) :
    result = 0
    result = v1 + v2
    return result

hap = 0

hap = plus(100,200)
print("100 과 200 의 plus() 함수 결과는 %d" % hap)
```

result, hap = 0 을 쓴 이유?

오류가 발생해도 result, hap 값을 살려서 오류 발생을 알 수 있으므로! 습관적인 코드

출력결과

100 과 200 의 plus() 함수 결과는 300

덧셈, 뺄셈, 곱셈, 나눗셈을 하는 계산기 함수를 작성

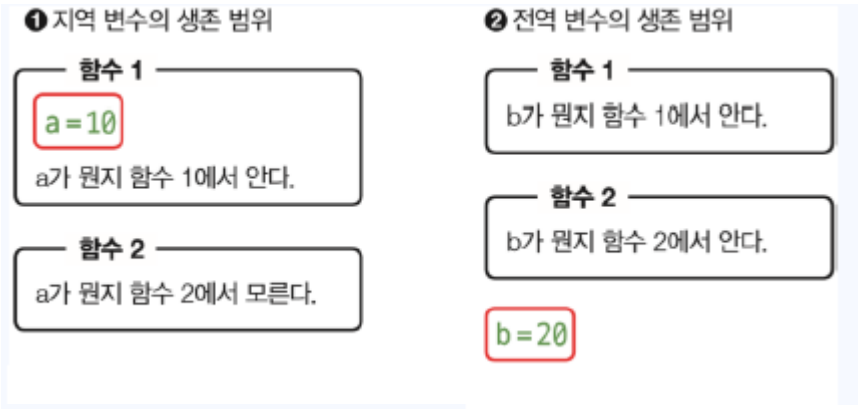
```
def calc(v1, v2, op) :
    result = 0
    if op == '+' :
        result = v1 + v2
    elif op == '-' :
        result = v1 - v2
    elif op == '*' :
        result = v1 * v2
    elif op == '/' :
        result = v1 / v2

    return result

res = 0
var1, var2, oper = 0, 0, ""

oper = input("계산을 입력하세요(+,-,*,/) : ")
var1 = int(input("첫 번째 수를 입력하세요 : "))
var2 = int(input("두 번째 수를 입력하세요 : "))

res = calc(var1, var2, oper)

print("##계산기 : %d %s %d = %d" % (var1, oper, var2, res))
```

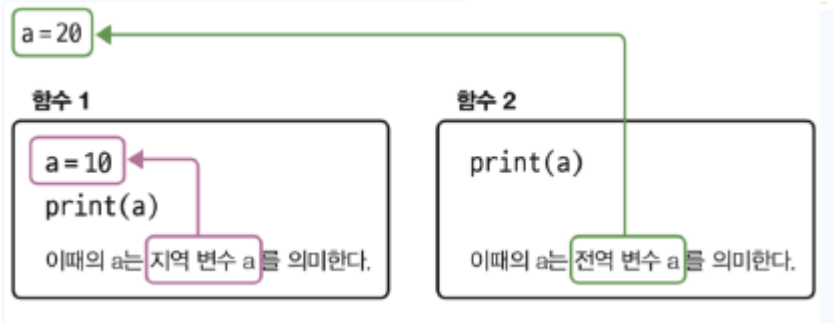
```
>>>
계산을 입력하세요(+,-,*,/) : *
첫 번째 수를 입력하세요 : 7
두 번째 수를 입력하세요 : 8
##계산기 : 7 * 8 = 56
```

지역 변수, 전역 변수

지역 변수와 전역 변수의 이해

- 지역변수 : 한정된 지역에서만 사용
- 전역변수 : 프로그램 전체에서 사용






```
def func1 () :
    a = 10
    print("func1()에서 a 값 %d" % a)

def func2() :
    print("func2()에서 a 값 %d" % a)

a=20

func1()
func2()
```

출력결과

```
func1()에서 a 값 10
func2()에서 a 값 20
```

a=20 이 없다면 func2 는?

출력결과

```
func1()에서 a 값 10
Traceback (most recent call last):
  File "C:/Users/kitcoop/AppData/Local/Programs/Python/Python36-32/함수 기본.py", line 10, in <module>
    func2()
  File "C:/Users/kitcoop/AppData/Local/Programs/Python/Python36-32/함수 기본.py", line 6, in func2
    print("func2()에서 a 값 %d" % a)
NameError: name 'a' is not defined
```

글로벌 예약어

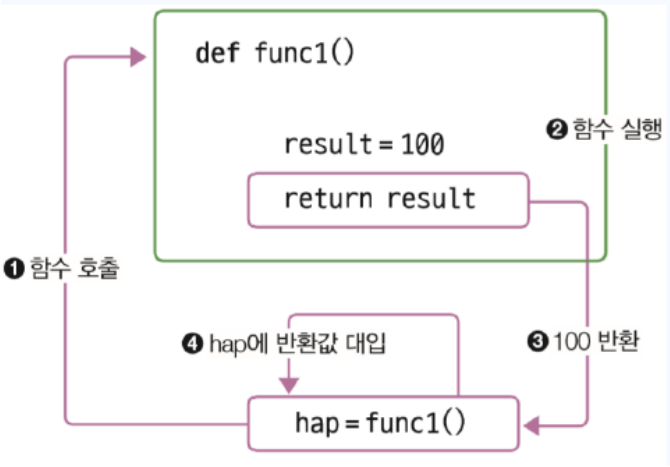
```
def func1() :
    global a          # 전역변수로 지정 명령어
    a = 10
    print("func1()에서 a 값 %d" % a)

def func2() :
    print("func2()에서 a 값 %d" % a)

func1()
func2()
```

출력 결과

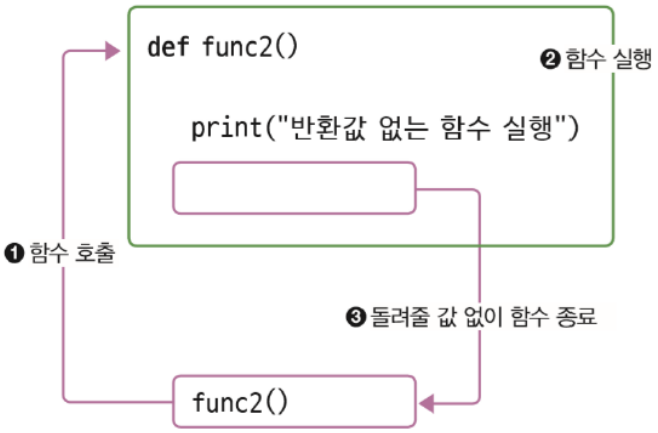
```
func1()에서 a 값 10
func2()에서 a 값 10
```

함수의 반환값과 매개변수

- 반환값이 있는 함수



- 반환값이 없는 함수



예시 - 반환 값이 없는 함수

```
def func1() :
    result = 100
    return result

def func2() :
    print("반환값이 없는 함수 실행")

#전역 변수 선언 부분#
hap = 0

#메인 코드 부분#
hap = func1()
print("func1()에서 돌려준 값 ==> %d" % hap)
func2()
```

출력 결과

```
func1()에서 돌려준 값 ==> 100
반환값이 없는 함수 실행
```

반환 값이 여러 개인 함수

```
def multi(v1, v2) :
    retList = []
    res1 = v1+v2
    res2 = v1-v2
    retList.append(res1)
    retList.append(res2)
    return retList

# 전역 변수 선언 부분 #
myList = []
hap, sub = 0, 0

#메인 코드 부분#
myList=multi(100,200)
hap=myList[0]
sub=myList[1]
print("multi에서 돌려준 값 ==> %d, %d" % (hap, sub))
```

출력 결과

multi 에서 돌려준 값 ==> 300, -100

 # return 을 여러 개 써도 마지막 return 만 실행함 !

따라서 리스트 등으로 return 후 색인해서 값 나타내기

cf. yield -> 설정하는 것 모두 출력됨

pass 예약어

```
def myFunc () :
    pass
```

- True 일 때 아무런 할 일이 없다고 빈줄로 두면 오류 발생!
  -> 넘어가려면 pass 사용(R 과의 차이점)

```
#오류발생#
if True :

else :
    print("거짓이네요")

#옳은 형태#
if True :
    pass
else :
    print("거짓이네요")
```

함수의 매개변수 전달

매개변수의 개수를 지정해 전달하는 방법

- 숫자 두 개의 합과 숫자 세 개의 합을 구하는 코드

```
def para2_func(v1, v2) :
    result = 0
    result = v1 + v2
    return result

def para3_func(v1, v2, v3) :
    result=0
    result = v1+v2+v3
    return result

##전역변수 선언 부분##
hap=0

##메인 코드 부분##

hap = para2_func(10,20)
print("매개변수가 2 개인 함수를 호출한 결과 ==> %d" % hap)
hap = para3_func(10,20,30)
print("매개변수가 3 개인 함수를 호출한 결과 ==> %d" % hap)
```

출력 결과

```
매개변수가 2 개인 함수를 호출한 결과 ==> 30
매개변수가 3 개인 함수를 호출한 결과 ==> 60
```

매개변수에 기본 값을 설정해놓고 전달하는 방법

기본값 처리 in python

R 과 다르게 앞의 변수에 default 를 지정하려면, 그 뒤의 모든 매개변수가 default 를 가져야 함.
그렇지만 뒤에 있는 변수만 default 는 가질 수 있음!

ex.

```
def plus(v1=0, v2=0)    # R 과 같은 형태
def plus (v1=0, v2)     # 불가능
def plus (v1, v2=0)       # 가능
```

함수 사용시 변수 순서 그대로 사용하거나,
변수 순서를 임의로 바꿀 때 매개변수 이름 지정해줘야 하는 점도 같음!

매개변수의 개수를 지정하지 않고 전달하는 방법

#in R 가변인자 ... 에 대체되는!

***para (or *args)**

```
def para_func(*para) :
    result=0
    for num in para :
        result = result + num

    return result
##전역변수 선언 부분##
hap=0

##메인 코드 부분##
hap = para_func(10,20)
print("매개변수가 2 개인 함수를 호출한 결과 ==> %d" % hap)
hap = para_func(10,20,30,40,50,60,70,80,90,100)
print("매개변수가 10 개인 함수를 호출한 결과 ==> %d" % hap)
```

출력 결과

```
매개변수가 2개인 함수를 호출한 결과 ==> 30
매개변수가 10개인 함수를 호출한 결과 ==> 550
```

함수 호출시 딕셔너리 형식의 매개변수를 키=값 형식으로 사용 + **para 로 사용 (or **kwargs)

```
def dic_func(**para) :
    for k in para.keys() :
        print("%s --> %d 명입니다." % (k, para[k]))

dic_func(트와이스=9, 소녀시대=7, 걸스데이=4, 블랙핑크=4)
```

출력 결과

```
트와이스 --> 9 명입니다.
소녀시대 --> 7 명입니다.
걸스데이 --> 4 명입니다.
블랙핑크 --> 4 명입니다
```

cf. **para 로 인수 입력시 키 위치에는 문자열 형태(' ')로 작성 불가능, 숫자로도 작성 불가능

연습문제

리스트를 전달받아 replace 할 수 있는 함수 만들기

```
def f_replace(data, v1=' ',v2='') :
    result=[]
    for a in data :
        result.append(a.replace(v1,v2))

    return result
```

replace 는 문자열 메소드

---나----------잘못풀었나봄...?0?

#리스트를 전달받아 각 원소별 두 번째 값마다 다음 순서의 값으로 치환하는 함수 만들기

```
def func1(l1) :
    for i in range(0 , len(l1)) :
        if (i < len(l1)-1) & (i % 2 == 1) :
            l1[i] = l1[i+1]
        elif (i == len(l1)-1) & (i % 2 == 1) :
            l1[i] = l1[i]

    return print(l1)


l2 = [1,3,88,1,3,82,4,6,8,4,7,9,10,11,12,13,14]

func1(l2)
```

```
def repl(l1) :
    for i in range(0, len(l1)) :
        l2[i] = str(l2[i])

        if (i < len(l1)-1) & (i%2==1) :
            l1[i]=l1[i].replace(l1[i],l1[i+1])
        elif (i == len(l1)-1) & (i%2==1) :
            pass

    return print(l1)
```

In [19]:

```
def f_nextreplace(data) :
    result=[]
    n=len(data)
    for i in range(0,n) :
        if i+1 == n :
            result.append(data[i])
        elif (i+1) % 2 == 0 :
            result.append(data[i+1])
        else :
            result.append(data[i])

    return result
```

실습문제 - 리스트 사용자 정의 함수 만들기
리스트를 전달받아 각 리스트의 원소에서
사용자가 지정한 분리 구분자로 분리된 각각의 데이터를 출력

['abc@naver.com','zz@daum.net']

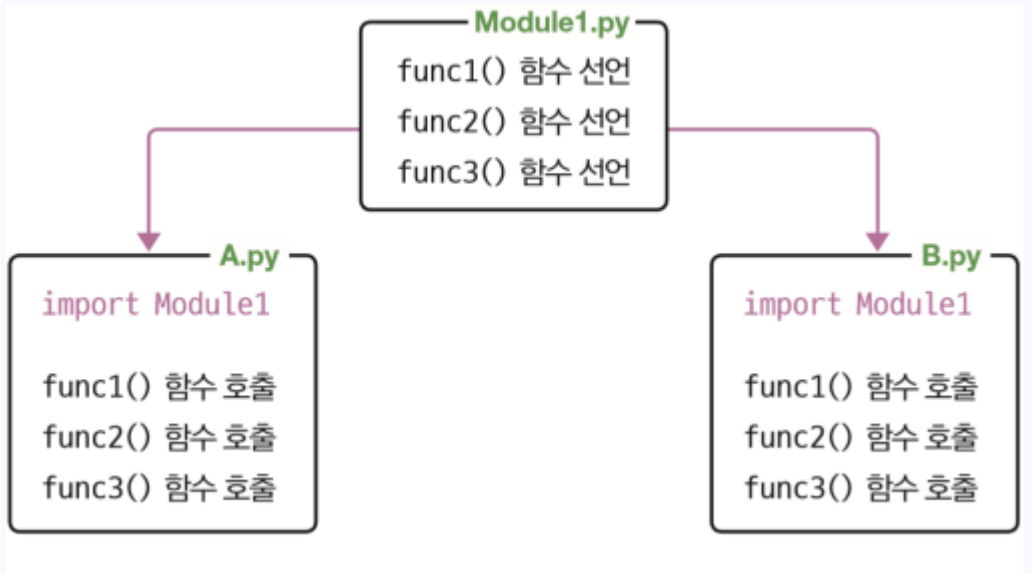
위의 데이터에서 sep 로 구분했을 때 num th 원소만 가져올 수 있도록?

```
def f_split(l1, sep=None, num=0) :
    result=[]
    for i in l1 :
        result.append(i.split(sep)[num])

    return result
```

#split - 문자열에만 적용가능한 method(list 안 됨), 리스트로 출력됨

모듈

모듈 : 함수의 집합



모듈의 생성과 사용

★ ★ ★ ★ 알리아스 형태로 사용 기억하기!! ★ ★ ★ ★

import Module1 as Alias

ex.

import numpy as np

import pandas as pd

호출을 할 때마다 모듈 이름/알리아스를 써야 함

모듈 이름 생략하고 함수 이름으로만 쓰고 싶을 때 사용방법

★ ★ ★ ★ 꼭 기억하기 !!! ★ ★ ★ ★

from 모듈명 import 해당함수 1, 해당함수 2, 해당함수 3

또는

from 모듈명 import *

```python
## 함수 선언 부분 ##
def func1() :
    print("Module1.py의 func1()이 호출됨.")

def func2() :
    print("Module1.py의 func2()가 호출됨.")

def func3() :
    print("Module1.py의 func3()이 호출됨.")
```

-> Module1 로 저장함

A.py

```python
import Module1

## 메인 코드 부분 ##
Module1.func1()
Module1.func2()
Module1.func3()
```

출력 결과

```
Module1.py의 func1()이 호출됨.
Module1.py의 func2()가 호출됨.
Module1.py의 func3()이 호출됨.
```

모듈의 종류

- 표준 모듈, 사용자 정의 모듈, 서드 파티 모듈로 구분

▪표준 모듈 : 파이썬에서 제공하는 모듈

▪사용자 정의 모듈 : 직접 만들어서 사용하는 모듈

▪서드 파티(3rd Party) 모듈 : 파이썬이 아닌 외부 회사나 단체에서 제공하는 모듈

•파이썬 표준 모듈이 모든 기능을 제공 않음

•서드 파티 모듈 덕분에 파이썬에서도 고급 프로그래밍 가능

•게임 개발 기능이 있는 pyGame, 윈도창을 제공 하는 PyGTK,
데이터베이스 기능의 SQLAlchemy 등

파이썬에서 제공하는 포준 모듈의 목록을 일부 확인

_(언더바) 포함 모듈 ; 기본적으로 불러옴

모듈을 호출하고 관리할 수 있음!

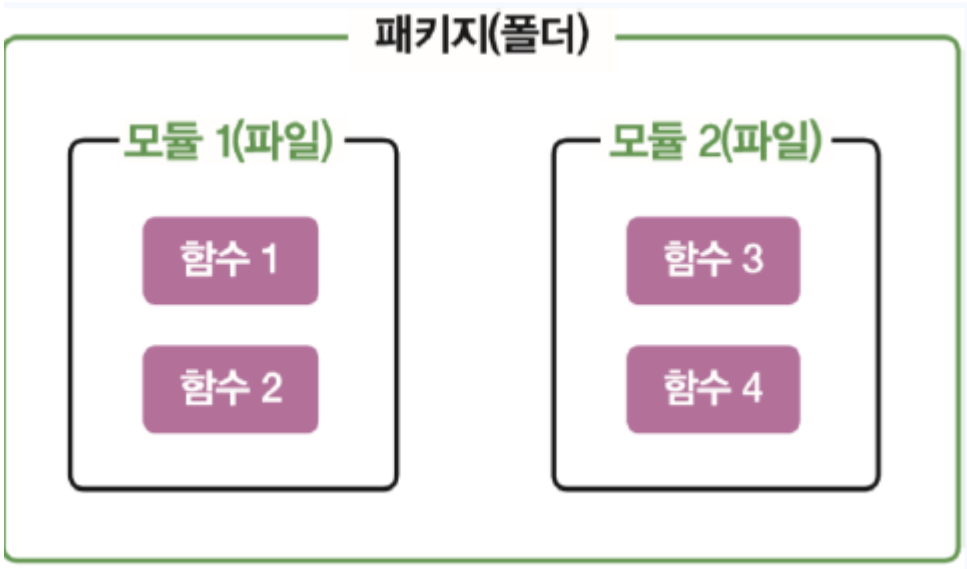
```
>>> import sys
>>> print(sys.builtin_module_names)
('_ast', '_bisect', '_blake2', '_codecs', '_codecs_cn', '_codecs_hk', '_codecs_iso2022', '_codecs_jp',
'_codecs_kr', '_codecs_tw', '_collections', '_csv', '_datetime', '_functools', '_heapq', '_imp', '_io', '_json',
'_locale', '_lsprof', '_md5', '_multibytecodec', '_opcode', '_operator', '_pickle', '_random', '_sha1',
'_sha256', '_sha3', '_sha512', '_signal', '_sre', '_stat', '_string', '_struct', '_symtable', '_thread',
'_tracemalloc', '_warnings', '_weakref', '_winapi', 'array', 'atexit', 'audioop', 'binascii', 'builtins', 'cmath',
'errno', 'faulthandler', 'gc', 'itertools', 'marshal', 'math', 'mmap', 'msvcrt', 'nt', 'parser', 'sys', 'time',
'winreg', 'xxsubtype', 'zipimport', 'zlib')
```

```
>>> import sys
>>> dir(__builtins__)        # 모듈에 있는 모든 함수 불러오기
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError',
'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError',
'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning',
'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError',
'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning',
'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt',
'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError',
'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning',
'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning',
'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning',
'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError',
'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError',
'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '_',
'__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__',
'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile',
'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter',
'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int',
'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next',
'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set',
'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial',
'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp',
'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'tau', 'trunc']
```

0614

패키지 - 폴더에 여러 모듈 넣기



- 임포트 형식

```
from 패키지명.모듈명 import 함수명
```

- 임포트 예제

```
from package.Module1 import *
```

내부함수　　내부함수 : 함수 안에 함수가 있는 형태 for 코드의 간편성

★★★★★lambda★★★★★

----굉장히 중요 !! , 딥러닝코드에서 삼항다항식처럼 많이 쓰임

함수 명= lambda / 인자 : / return

- 사용자 정의 함수의 축약형, **콜론 이전과 콜론 이후**로 나뉨 !

cf. 삼항다항식(컴프리헨션) - for 문 이전과 이후로 나뉨

cf. list 의 각 원소에 적용하고 싶을 때?

1. for 문을 통해 적용하는 방법　　**2.** 적용함수를 쓰는 방법

#lambda 는 꼭 적용함수와 써야 함 !

***람다 이해하기**

lambda _, _, _ :

↑ ↑ ↑

인자 각각엔 단 하나의 원소만 들어가도록 !

- lambda 내에서 반복문이 안 됨. 무조건 원소 하나만 써야 함. 리스트도 못들어감!

앞서 한 리스트의 원소를 split 하는 for 문 -> 람다형식으로 만들기를 통해 이해하기

```
ff_split = lambda x : x.split('@')[0]      # input x 값은 반드시 단 하나의 원소만 가능!
ff_split("abc@naver.com")

 ===> map() 적용함수 사용 for 리스트 원소별 적용
                          cf. 적용가능 함수 1. apply, 2. applymap, 3. map

list1=['abc@naver.com','sdf@google.com','asdf@daum.net']

map(ff_split, list1)        #값을 메모리 어디에 저장만 함, 출력 X
list(map(ff_split,list1))   #list 화

+ map 은 인자를 쌍으로 전달해야 함     (cf. mapply in R)
                                  # default 값이 있는 경우 숫자 안 맞춰줘도 됨 !
ff_split2=lambda x,y : x.split(y)
list(map(ff_split2,list1,'@')
#list1 -> 세 개, '@' -> 한 개, '@'가 하나뿐이므로 list1 의 1st 원소만 전달됨.

따라서
```

list(map(ff_split2,list1,['@','@','@'])

참고

l1=["abc@naver.com","abc.xxxx","xxxx,kjfd"]

l2=["@",".",","]

list(map(ff_split2,l1,l2))

출력 결과

['abc', 'abc', 'xxxx']

(각각 구분 기호가 다른 건 거의 드문 케이스, but 짝 맞춰서 하는 예시 측면에서!!)

람다 연습문제

l5=["1,234","3,234","9,999"]

#replace 로 "," 처리 한 후 정수로 바꾸기 (lambda 사용)

```
minus = lambda data, bf, af : int(data.replace(bf, af))
a1 = list(map(minus, l5, [",",",",",",",",],["","","",""]))
a1
```

출력결과

[1234, 3234, 9999]

(오류 예시)

```
list(map(minus, l5, ",",""))
```

출력결과

[]

잘 적용 안 됨

? map 은 쌍으로 전달받는 함수이므로 !

x	a	b
1,234	","	""
3,234		
9,999		

이렇게 전달된 것과 같다

---

cf

```
lambda x : x.replace(",", "").astype("int")
# astype 은 pandas 모듈 호출해서 쓰기
```

문제 2

["123.98775", "345.9876"] => [123.99, 345.99] 로!

ex. print 에서 "%.2f" 형태로 사용

프린트에서 쓰는 표현식을 람다에서도 쓸 수 있음

얘는 왜 문자열 형태로 출력될까? → 아마 print 형식으로 출력 = 문자열이어서?

```
l1=[1537.53157,453.153574,789.123]
f2 = lambda x1 : "%.2f" % x1
list(map(f2,l1))
['1537.53', '453.15', '789.12']

################################################

>>> l1=[1537.53157,453.153574,789.123]
>>> f2 = lambda x1 : float("%.2f" % x1)
>>> list(map(f2,l1))
[1537.53, 453.15, 789.12]
```

람다 심화문제

list1=[1,2,3,4,5]
list2=['서울','부산','대전','전주']
list3=['2007(1)','2007(2)','2007(3)','2007(4)']
list4=['345-4958','334-0948','394-9050','473-3853']

1. list1 의 값의 제곱을 출력

2. list2 의 값에 "시"를 붙여 출력

3. 데이터를 불러왔더니 컬럼명이 list3 과 같은 형태로 변형되었을 경우
각각 년도와 분기를 분리하여 출력

4. list4 는 지역번호를 제외한 전화번호이다. 각 지역을 나타내는 값은 list5 에 다음과 같이 생성되어 있고, area 에는 각 지역별 지역번호가 다음과 같은 딕셔너리 형태로 저장되어 있을때 이들을 사용하여 지역번호와 전화번호가 결합된 형태로 출력하여라

list5=['서울','경기','부산','제주']
area_no={'서울':"02",'경기':"031",'부산':"051" ,'제주':"064"}

#1.list1 의 값의 제곱을 출력
q1=lambda a : a**2
list(map(q1, list1))

#2.list2 의 값에 "시"를 붙여 출력
q2 = lambda b : "%s 시" % b
#cf.
q2 = lambda b : **b + "시"**
list(map(q2, list2))

#3.각각 년도와 분기를 분리하여 출력
q3_1 = lambda a : a.split("(")[0] # 첫번째 원소만 따로 빼기..'o'... 년도 컬럼만 새로 빼기
list(map(q3_1, list3))
q3_2 = lambda a : **a.split("(")[1] .replace(")","")**
list(map(q3_2, list3))
출력결과
['2007', '2007', '2007', '2007']
['1', '2', '3', '4']

cf. substring 사용!
[5:6] 등등

나
```
q3 = lambda a : a.replace(")","").split("(")
list(map(q3, list3))
```

출력결과
```
[['2007', '1'], ['2007', '2'], ['2007', '3'], ['2007', '4']]
```

#4. list5, area_no 를 사용하여 list4 를 전화번호가 결합된 형태로 출력
```
list5=['서울','경기','부산','제주']
area_no={'서울':"02",'경기':"031",'부산':"051" ,'제주':"064"}

q4=lambda k1, k2 : area_no[k1] + ')' + k2
list(map(q4, list5, list4))
```

cf. 만약에 lambda 에 area_no 도 새로 k3 으로 가져온다면 오류! 왜냐면 lambda 는 **한 원소만 허용하므로!**

+ 유용한 내장 함수

iter(x) - 자료 x 가 iterable 객체인지 판단 (Type Error 발생 시 iterable 객체가 아님)

enumerate(x,k) - 자료 x(iterable 객체)의 원소에 시작번호 k 로부터 번호를 붙여 enumerate 객체를 반환

filter(fun, x) - 자료 x(iterable 객체)의 원소에 fun 함수를 적용하여 결과가 True 인 원소만 반환.

map(fun, x) - 자료 x(iterable 객체)의 원소에 fun 함수를 적용하여 반환

(단 fun 함수의 결과는 논리값만 반환)

In [106]:
```
score=[90,39,20,10,50]
ls=list(enumerate(score)) # 기본값 : 0부터 시작
tp=tuple(enumerate(score))
dt=dict(enumerate(score))
print("{0}₩n{1}₩n{2}".format(ls,tp,dt))
```

```
[(0, 90), (1, 39), (2, 20), (3, 10), (4, 50)]
((0, 90), (1, 39), (2, 20), (3, 10), (4, 50))
{0: 90, 1: 39, 2: 20, 3: 10, 4: 50}
```

In [107]:
```
area=['서울','경기','인천']
dic=dict(enumerate(area,100))  # 키를 100부터 시작
dic
```

Out[107]:
```
{100: '서울', 101: '경기', 102: '인천'}
```

In [108]:
```
score=[9,3,5,6,2,1]
list(filter(lambda x : x % 2 == 0, score))
```

Out[108]:
```
[6, 2]
```

+ docstring

- 함수 선언문과 본체 사이에 작성하는 문자열
- 함수의 사용법, 인수의 의미, 주의사항 등 설명 작성(''' ''' 또는 """ """)

In [110]:
```
def calc(x=2, y=10) :
    '''
    두 수의 사칙연산과 나머지
    x, y에 두 수 입력
    '''
    add=x+y
    dif=x-y
    mul=x*y
    div=x/y
    rem=x%y
    return add, dif, mul, div, rem

help(calc)
```

```
Help on function calc in module __main__:

calc(x=2, y=10)
    두 수의 사칙연산과 나머지
    x, y에 두 수 입력
```

재귀함수

- 자신이 자신을 호출
- 반복문이 있는 것처럼 출력

ex1)

```
def selfCall() :
    print('하', end="")
    selfCall()

selfCall()
```

출력결과

하하하하하하하하하하하하.... (*무한)

ex2)

```
def count(num) :
    if num >=1 :
        print(num, end=' ')
        count(num-1)
    else :
        return
count(10)
count(20)
```

출력결과

10 9 8 7 6 5 4 3 2 1

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

ex3) 팩토리얼!

```
def factorial(num) :
    if num <= 1 :
        return num
    else :
        return num * factorial(num-1)
```

제너레이터와 yield

- yield ; 함수를 종결하지 않고 값을 계속 반환.
- 함수 내부에서 사용된 데이터들이 메모리에 그대로 유지됨
- 여러 개를 한꺼번에 출력 가능 (cf. return 은 마지막에 실행된 한 행만!)
- 제너레이터 : yield 가 포함된 함수
- 사이즈가 커져도 차지하는 메모리 사이즈가 동일, 대용량 처리시 적합. (next 함수를 통해 차례로 값에 접근할 때마다 메모리에 적재하는 방식)
- 컴프리헨션에 ( )를 씌워 사용하면 generator 생성

(참조 https://kkamikoon.tistory.com/90)

ex1)

```
def getFunc() :
    yield 1
    yield 2
    yield 3

print(list(getFunc()))
```

출력결과

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

ex2)

```
def getFunc(num) :
    for i in range(0, num) :
        yield i
        print('제너레이터 진행 중')
for data in getFunc(5) :
    print(data)
```

출력결과

```
0
제너레이터 진행 중
1
제너레이터 진행 중
2
제너레이터 진행 중
3
제너레이터 진행 중
4
제너레이터 진행 중
```

연습문제

#1. 로또생성기

힌트

import random

random.randrange(1,45) -- 하나만 생성됨, 중복된 값이 있으면 안 됨

나

```
In [*]: while 1 :
            b.append(random.randrange(1,45))
            if len(set(b))==6 :
                print(sorted(set(b)))
                break
            else :
                continue
```

→ 굉장히 무거운 방법 ㅠ-ㅠ

강사님 방법

```
l1=[]
from random import randrange   #

while len(l1) < 6 :
    l1=list(l1)    #
    l1.append(randrange(1,45))
    l1=set(l1)


l1=list(l1)   #
l1.sort()     #

print("** 로또 추첨을 시작합니다. ** ₩n₩n 추첨된 로또 번호 ==>", end=" ")

for i in l1 :         #
    print(i, end="  ")
```

#모듈명 사용해서 함수 쓰기 불편함!

-> randrange 만으로도 사용 가능하게 불러오기. 알리아스 작성도 가능.

 # set : 딕셔너리, append 불가, l1=set(l1) 때문에 생긴 행

#set !! 무조건 중복된 값 제거, 유일한 원소(distinct value)들만 나옴,
단점 -- set : key value 로만 저장되는 딕셔너리 구조,
데이터 추가, 수정은 append 가 아니라 집합 형태 연산으로 가능(합집합, 차집합 등등)

#set 로 결과 나온 값, list 로 바꿔주기

 # 출력이 아니라 정렬, 다시 l1 으로 정의 안 해도 알아서 정렬됨

#[,]인 리스트 형태로 말고, 값만 차례대로 나타내기 위해서 for 문

#2. 재귀함수를 이용해서 사용자가 입력한 값을 거꾸로 출력하기

강사님 방법

```
def f_reverse(w) :
    if w == "" :
        return w
    else :
        return w[-1] + f_reverse(w[0:-1])
```

 # -1=마지막값으로 설정해서 하는 a[0:-1]! 신기하군..
 # 재귀함수 - 본인이 본인을 불러올 때 다음 값 설정이 어려움
 # + 형태로 문자 중첩 가능! cf. *는 반복

참고

w 값	출력값
안녕하세요	요
안녕하세	요세
안녕하	요세하
안녕	요세하녕
안	요세하녕안
""	요세하녕안""

나

```
def bback(a) :
    a=str(a)
    i=len(a)
    if i >= 1 :
        print(a[i-1], end="")
        bback(a[0:i-1])
    else :
        return
```

이것도 됨

```
def bback(a) :
    a=str(a)
    i=len(a)
    if i == 0 :
        return
    print(a[i-1], end="")
    bback(a[0:i-1])
```

오류발생케이스-1

```
def bback(a) :
    a=str(a)
    i=len(a)
    print(a[i-1], end="")
    if i == 0 :
        return
    bback(a[0:i-1])
```

오류발생케이스-2

```
def bback(a) :
    a=str(a)
    i=len(a)
    print(a[i-1], end="")
    bback(a[0:i-1])
    if i == 0 :
        return
```

고민 끝에 내린 결론

사용자 정의함수를 print 하기 전에 if 로 print 할 조건을 걸러야 한다.
Print 다음에 거를 조건을 두면 오류남!

+ 이건 왜 안 되는가 !

```
def bback(a) :
    a=str(a)
    i=len(a)

    while 1 :
        if i == 0 :
            return
        else :
            print(a[i-1], end="")
            bback(a[0:i-1])
```

첫 글자가 계속계속계속계속 반복됨
-> 안 된 이유? 재귀함수 bback 에서 i==0 이 나오면 다시 while1 문 처음으로 돌아감
-> i=1 일 때가 계속계속 반복됨
->마무리를 지어야 한다

다시 고쳤다.

```
def bback(a) :
    a=str(a)
    i=len(a)
    while 1 :
        if i == 0 :
            return
        else :
            print(a[i-1], end="")
            bback(a[0:i-1])
            return
```

return 을 bback 뒤에 마무리해주니까 됐다.
답이 안 나올 때에 return 표시가 꼭 있어야 하는 것 같으다..

#3. 회문 판별 로직 : 앞으로 읽어도 뒤로 읽어도 똑같은 글자

강사님 방법

```
import math
w=input (" 회문을 판단할 단어를 입력하세요 : ")

cnt=math.trunc(len(w)/2)

if len(w) >=2 : ###
    for i in range(0,cnt) :   #
        if w[i] == w[-(i+1)] :
            continue #
        else :
            print("회문이 아닙니다.")
            exit(0)  ##

    print("회문입니다.")
else :
    print("Null값 또는 한자리 문자가 입력되었습니다.")
```

#반복횟수를 출력해서 한 번에 처리하기 -> 짝/홀수 상관 없이 하려면 math 모듈 trunc()사용

exit : stop point 를 단 한 번이라도 들어가면 나감, 다음 문장 실행 안 됨
python 프로그램 종료할 건지 물음
cf. break 는 뒤에 나오는 문장 사용됨

###한 글자여도 회문, 두 글자 이상일 때만 하기

#cf. pass - else 조건만 하고 싶을 때. 여기에서 써도 실행되지만, for 문의 반복의 의미로 continue 의 의미가 더 맞음

In [6]:
```
import math
w=input('회문을 판단할 단어를 입력하세요 : ')
w=str(w)

num=math.floor(len(w)/2)
l1=[]

for i in range(0,num) :
    if w[i] == w[-(i+1)] :
        l1.append(i)
    else :
        pass

if len(l1)==num :
    print('회문')
else :
    print('회문X')
```

```
회문을 판단할 단어를 입력하세요 : 무우루
회문X
```

+여러 가지 모듈

random 모듈

	함수	설명
임의의 정수 선택	randint(a,b) randrange(x,a,b)	[a,b] 사이의 임의의 정수 선택 range 함수의 결과 중 임의의 값 선택 [a,b-1]
임의의 실수 선택	random() uniform(a,b)	[0,1] 사이의 임의의 실수 선택 [a,b] 사이의 임의의 실수 선택
컨테이너 자료 x 에서 선택	choice(x) sample(x,k) shuffle(x)	x(문자열, 리스트, 튜플)에서 임의의 원소 한 개 선택 x(문자열, 리스트, 튜플, 집합)에서 k 개를 중복 없이 임의로 선택 x(리스트)의 데이터를 섞어서 리스트로 반환

임의의 정수/실수 선택

```
In [15]: import random
         print(random.randint(0,5))
```

```
5
```

```
In [58]: print('randrange(1 to 10 by 2) : {0}'.format(random.randrange(1,11,3)))
```

```
randrange(1 to 10 by 2) : 10
```

```
In [59]: print('random() : {0}'.format(random.random()))
```

```
random() : 0.3716164704025643
```

```
In [62]: print('uniform(1,10.5) : {0}'.format(random.uniform(1,10.5)))
```

```
uniform(1,10.5) : 2.7171426041078712
```

컨테이너 자료 x에서 선택 - choice

In [63]:
```
str='python program'
ls=[12,3,45,1]
tp=(1,2,5,3)
st={1,4,3,6,7}
dic={"서울":'02','경기도':'031','세종':'044'}

print('문자열 임의 선택 : {0}'.format(random.choice(str)))
print('리스트 임의 선택 : {0}'.format(random.choice(ls)))
print('튜플 임의 선택 : {0}'.format(random.choice(tp)))

New_ls=[random.choice(ls) for i in range(5)] # ls에서 5개의 원소 복원추출
New_ls
```

```
문자열 임의 선택 : m
리스트 임의 선택 : 12
튜플 임의 선택 : 5
```

Out[63]:
```
[45, 12, 12, 1, 12]
```

In [64]:
```
# choice 함수에서 집합, 사전은 error 발생
random.choice(st)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-64-333c01f9adb8> in <module>
      1 # choice 함수에서 집합, 사전은 error 발생
----> 2 random.choice(st)

C:\ProgramData\Anaconda3\envs\insun\lib\random.py in choice(self, seq)
    259         except ValueError:
    260             raise IndexError('Cannot choose from an empty sequence') from No
ne
--> 261         return seq[i]
    262
    263     def shuffle(self, x, random=None):

TypeError: 'set' object does not support indexing
```

- sample, shuffle

In [71]:
```
str='python program'
ls=[12,3,45,1]
tp=(1,2,5,3)
st={1,4,3,6,7}
dic={"서울":'02','경기도':'031','세종':'044'}

print('문자열 sample 선택 : {0}'.format(random.sample(str,4)))
print('list sample 선택 : {0}'.format(random.sample(ls,3))) # 비복원추출
print('튜플 sample 선택 : {0}'.format(random.sample(tp,3))) # 비복원추출
print('집합 sample 선택 : {0}'.format(random.sample(st,3))) # 비복원추출
print('')

random.shuffle(ls)
print('리스트 섞기 : {0}'.format(ls))
```

```
문자열 sample 선택 : ['h', 'p', 'o', 't']
list sample 선택 : [3, 12, 45]
튜플 sample 선택 : [2, 3, 5]
집합 sample 선택 : [7, 1, 4]

리스트 섞기 : [12, 3, 45, 1]
```

+itertools 모듈

함수	설명
chain()	인수로 지정한 리스트들을 연결
count(x,by)	인수 x 로 넣은 정수로부터 증분 by 인 수열을 무한히 생성
dropwhile(f,x)	iterable 자료 중 함수 f 결과가 처음 False 인 자료 반환
takewhile(f,x)	iterable 자료 중 함수 f 결과가 처음 True 인 자료 반환
groupby(x,f)	iterable 자료 중 함수 f 결과를 키로 하여 그룹화

In [74]:
```
import itertools
ls=[23,41,52,64,24]
str='python'
dic={"서울":'02','경기도':'031','세종':'044'}

print(list(itertools.chain(ls,str,dic)))
```

```
[23, 41, 52, 64, 24, 'p', 'y', 't', 'h', 'o', 'n', '서울', '경기도', '세종']
```

In [80]:
```
for x in itertools.count(70) :
    if x>90 :
        break

    print(x, end='₩t')

print('₩n')

for x in itertools.count(50,5) :
    if x > 100 :
        break

    print(x, end='₩t')
print('₩n')

print(list(zip(itertools.count(1),['a','b','c']))) #zip과 사용
```

```
70      71      72      73      74      75      76      77      78      79      80
81      82      83      84      85      86      87      88      89      90

50      55      60      65      70      75      80      85      90      95      100

[(1, 'a'), (2, 'b'), (3, 'c')]
```

In [87]:
```
ls=[3,2,5,6,1,7,2,5,5]

for x in itertools.dropwhile(lambda x : x<7, ls) :
    print(x, end="₩t")

print('₩n')

for x in itertools.takewhile(lambda x : x<7, ls) :
    print(x, end='₩t')
```

```
7       2       5       5

3       2       5       6       1
```

In [88]:
```
ls=['공기','책상','소년','책장','배움','백상예술대상','공집합','소나기','책걸상']
d={}
for word in sorted(ls) :   # 정렬한 형태의 list 사용
    if word[0] in d :
        d[word[0]].append(word)
    else :
        d[word[0]]=[]
        d[word[0]].append(word)

for k in d.keys() :
    d[k].sort()    # 키의 값(리스트) 정렬

for k, v in d.items() :
    print(k, v)
```

```
공 ['공기', '공집합']
배 ['배움']
백 ['백상예술대상']
소 ['소나기', '소년']
책 ['책걸상', '책상', '책장']
```

In [93]:
```
for k, group in itertools.groupby(sorted(ls),lambda w : w[0]) :
    print(k,list(group))
```

```
공 ['공기', '공집합']
배 ['배움']
백 ['백상예술대상']
소 ['소나기', '소년']
책 ['책걸상', '책상', '책장']
```

+ time 모듈
- 날짜와 시간 관련 기능 제공
- 에폭(Epoch) / 유닉스 시간
- 보다 편리한 형태로 조립하려면 localtime 함수
  / 지역시간 고려하여 현지 시간 구함
  / 시간 요소 멤버로 가지는 strut_time 형 객체 반환
  / 정보 분리하여 문자열로 조립
  / time.strftime() 함수를 이용하여 다양한 포맷 형식으로 출력
- 실행 시간 측정

코드	설명	예
%a	요일 줄임말	Sun, Mon, ... Sat
%A	요일	Sunday, Monday, ..., Saturday
%w	요일을 숫자로 표시, [0(일요일)~6]	0, 1, ..., 6
%d	일	01, 02, ..., 31
%b	월 줄임말	Jan, Feb, ..., Dec
%B	월	January, February, ..., December
%m	숫자 월	01, 02, ..., 12
%y	두 자릿수 연도	01, 02, ..., 99
%Y	네 자릿수 연도	0001, 0002, ..., 2017, 2018, 9999
%H	시간(24 시간)	00, 01, ..., 23
%I	시간(12 시간)	01, 02, ..., 12
%p	AM, PM	AM, PM
%M	분	00, 01, ..., 59
%S	초	00, 01, ..., 59
%Z	시간대	대한민국 표준시
%j	1 월 1 일부터 경과한 일수	001, 002, ..., 366
%U	1 년중 주차, 월요일이 한 주의 시작으로	00, 01, ..., 53
%W	1 년중 주차, 월요일이 한 주의 시작으로	00, 01, ..., 53
%c	날짜, 요일, 시간을 출력, 현재 시간대 기준	Sat May 19 11:14:27 2018
%x	날짜를 출력, 현재 시간대 기준	05/19/18
%X	시간을 출력, 현재 시간대 기준	'11:44:22'
%%	문자	%
%f	밀리세컨드	

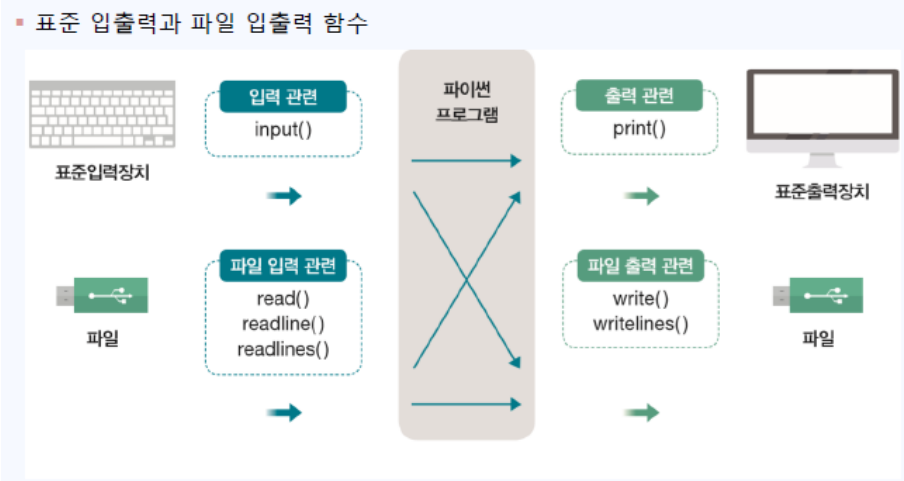
 # datetime.datetime.strptime(문자열, "형식") 으로 문자 → 시간 데이터로 변환

+ .date() - 날짜관련 정보만 나오게 / .time() - 시간관련 정보만 나오게

0615

Chapter 8 파일에 데이터 읽고 쓰기

파일 입출력의 개념



(for 정형화 데이터 X, 텍스트 파일 - 리스트 구조, R 이랑 다르게 for 문을 통해 하나하나 가져와야 함)

cf. 표준 출력을 화면에 뜨지 않게 변경한다. ex. a=~~ 변수설정

파일 입출력의 기본과정 -vs R 텍스트 부분 차이가 남

1 단계 -- open()

읽기용 **변수명 = open("파일명", "r")**

쓰기용 **변수명 = open("파일명", "w")**

모드 -- open()함수의 마지막 매개변수

종류	설명
r (or 생략)	읽기모드. (기본값)
w	쓰기모드. 기존에 파일이 있으면 덮어쓴다.
r+	읽기/쓰기 겸용모드
a	쓰기모드. 기존 파일이 있으면 이어서 씀 (append 약어)
t	텍스트 모드. 텍스트 파일을 처리한다. (기본값)
b	이진모드. 이진 파일을 처리함.

cf. 이진모드 - 이미지파일 등, 텍스트 파일이 아닌건 모두 이진모드

2 단계 -- 파일처리(패치)

3 단계 -- 파일 닫기 / 1 단계에서 open() 함수로 연 변수명

변수명.close()

 # 반드시 해줘야 함 ! 처리가 끝나면 메모리 누수가 없도록 (안 하면 열려있음) 꼭 써주기

텍스트파일 입출력

파일을 이용한 입력

한 행씩 읽어들이기

- readline() 메서드 사용

- 파일로 데이터 입력 후 이를 화면에 출력하는 프로그램

```
inFp = None
inStr = ""    #오류 방지용

inFp = open("ptc1.txt", "r")  #파일 열기

inStr = inFp.readline()  # inFp의 첫 행 읽어 inStr에 저장
print(inStr, end="") #출력

inStr = inFp.readline()  # inFp의 두 번째 행 읽어 inStr에 저장
print(inStr, end="")

inStr = inFp.readline()  # inFp의 세 번째 행 읽어 inStr에 저장
print(inStr, end="")

inFp.close()    # 파일 닫기!
```

@ readline 사용시 한 줄씩만 입력 -> 여러 번 반복 필요

- 텍스트 파일 내용 중 정확하게 행 세 개만 처리 가능.
- 텍스트 파일에 행이 1~2 개만 있으면 오류 발생
- 4 개 이상 있어도 3 개만 읽음
- print / **end="" 옵션**을 사용하지 않으면 값 사이에 ₩(엔터) 공백이 생김, 옵션 사용해야 바로 다음 라인에 출력.

텍스트 파일에 있는 모든 행 다 읽기

```
In [66]: inFp = None
         inStr = ""

         inFp = open("test1.txt", "r")
         inFp2= inFp.readline()

         while inFp2 :
             print(inFp2, end="")
             inFp2=inFp.readline()
         inFp.close()
```

```
1 2 3 4
5 2
1 5 2
```

readline --> 모듈로 사용자정의 함수 저장해서 한 번에 출력하게 하는 방법 권장 (뒤에 문제)

readline 자체가 한 줄 줄바꿈 옵션이 기본적으로 있기 때문에 end=""를 해도 옆에 안 뜸. 기본값은 값 사이에 엔터공백.

@

만약 for 문 이라면 총 줄 수 알아야 함

while True : ~ 공백일 때 넘어가도록 break 처리

readlines()

```
1  inFp = None
2  inList, inStr = [], ""
3
4  inFp = open("C:/Temp/data1.txt", "r")
5
6  inList = inFp.readlines()
7  for inStr in inList :
8       print(inStr, end = "")
9
10 inFp.close()
```

출력 결과

```
하하하하 파이썬을 공부합니다.
완전 재미있어요. ^^
파이썬을 공부하기 잘했네요~~
```

참조

readlines()는 모든 행을 원소 하나씩 가져와서 리스트로 출력하지만, 행의 끝에 ₩n 의 개행문자도 삽입된 형태로 저장됨.

6 행의 inList 출력시

['1:2:3:4₩n', '12:3:4₩n', '5:6₩n']

-> 7 행의 for 문을 통해 원소 하나하나를 print 하면서 ₩n 이 안 보이게 출력

#

readlines ; 반복 없이 데이터를 가져오지만, 표현을 위해서 for 문 필요(print), end="" 옵션 역시 또 사용. 중간에 빈 라인 없이 연속적 출력을 위해서 !

출력 프로그램

```
inFp = None
fName, inList, inStr="",[],""

fName=input("파일명을 입력하세요 : ")
inFp = open(fName, "r")

inList = inFp.readlines()
for inStr in in List :
    print(inStr, end="")

inFp.close()
```

텍스트 - 디폴트 디렉토리에 넣으면 파일 명으로만 불러올 수 있음. 경로 지정 안 하고!

파일이 없을 때, 오류가 발생하지 않게 하려면? os.path.exists(파일명) 형식 사용

```
import os                           # os 모듈 임포트

inFp=None
fName, inlist, inStr = "", [], ""

fName = input("파일명을 입력하세요 : ")

if os.path.exists(fName) :          # 파일이 있으면 True 반환
    inFp = open(fName, "r")

    inList = inFp.readlines()
    for inStr in inList :
        print(inStr, end="")

    inFp.close()
else :
    print("%s 파일이 없습니다." %fName)
```

사용자 정의함수 만들기 연습문제

#1. read_txt("위치") 로 하면 출력할 수 있도록 사용자 정의함수 만들기

```
def read_txt2(file) :
    fp=open(file,"r")
    while 1 :
        fstr=fp.readline()
        print(fstr, end="")
        if fstr=="" :
            break

    fp.close()
```

#2. 파일 라인 번호와 함께 출력하는 형태

다음과 같이 앞에 각 행 번호를 출력해 보자.

출력 결과

```
1: 하하하하 파이썬을 공부합니다.
2: 완전 재미있어요. ^^
3: 파이썬을 공부하기 잘했네요~~
```

```
def read_txt(x) :
    inFp = None
    inList, i = [], ""
    inFp=open(x, "r")
    inList=inFp.readlines()
    for i in range(0, len(inList)) :
        print("%d: %s" % (i+1, inList[i]), end="")
```

readline(), readlines() --**#함수의 결과 출력만 할 수 있음. 변수지정으로 해서 못 전달함**

자료 구조로 만들기 위해서
1,2,3,4,5 / 1,2,3 / 2,3,4 를 리스트 안에 리스트 구조인 2 차원 리스트로 담으려면?

#띄어쓰기 등등 sep 형태로 가져오기
#이중 for 문이 아니어도 풀 수 있음
#분리구분기호!

강사님

```
#    read_txt("file", sep=",") 구분기호 넣었을 때 2차원 list로 저장할 수 있는 사용자 정의함수

def read_txt3(file, sep=" ") :
    fp=open(file, "r")
    ll=[]
    while 1 :
        fstr=fp.readline()   #불필요한 공백 없애주기 !O! 내려쓰는 공백도 공백으로 처리함..!!!
        if fstr.strip()=="" :
            break
        ll.append(fstr.strip().split(sep))        #메소드는 연속적 호출 가능, 순차적으로 처리함 !

    fp.close()
    return ll
```

#strip 도 ₩n 개행문자 없애줌 !

#나중엔 정규식 표현을 활용해 replace 로 하기

---->치환, 읽어오는 것과 관련한 모듈 잘 정리하기

#split 결과 -> list

내가 푼 거

```
#---------------
#    read_txt("file", sep=",") 구분기호 넣었을 때 2차원 list로 저장할 수 있는 사용자 정의함수

def read_txt3(file, sep=",") :
    fp=open(file, "r")
    l1=[]
    l2=[]
    while 1 :
        fstr=fp.readline()
        if fstr=="" :
            break

        test=fstr.split(sep)
        for i in test :
            test2=i.replace("\n","")
            l1.append(test2)

        l2.append(l1)
        l1=[]
    fp.close()

    print(l2)

#\n을 어떻게 없애지?--> replace!
```

#split 결과 -> list, 바로 list 에 넣으면 됨! 근데 두 번 풀었다.
개행문자를 나는 replace 로 수정함..!

여기서 잠깐

with~as 문

프로그램을 만들다 보면 close() 함수를 호출하지 않고 프로그램을 종료해서 종종 문제가 발생한다. 아예 close() 함수를 사용하지 않으려면 다음과 같이 with~as 문으로 파일을 연다. Code11-03.py에서 4~9행을 다음과 같이 수정하면 된다. 즉 inFp.close()는 사용하지 않는다.

```
with open("C:/Temp/data1.txt", "r") as inFp :
    inList = inFp.readlines()
    print(inList)
```

한 행씩 파일에 입력

```
outFp = None
outStr = ""     #이후 오류방지 처리

outFp=open("text.txt", "w")

while True :
    outStr=input("내용 입력 : ")
    if outStr != "" :
        outFp.writelines(outStr + "₩n")      #개행문자 입력 X -> 붙여서 입력됨
    else :
        break

outFp.break
print("--정상적으로 출력--")
```

wrtie_text 만들어보기

In [ ]:

```
def wrtie_txt(file) :
    of1=open(file, 'w')
    w=input("저장할 문장을 입력하세요 : ")
    while True :
        if w!="" :
            of1.writelines(w+'₩n')

        else :
            break

    of1.close()
    print('----정상적으로 파일에 씀------')
```

이중리스트 파일에 입력

첫번째 라인으로 1,2,3,4,5 로 저장될 것 (기본 csv)

띄어쓰기로 저장하고 싶을 때 옵션 넣기..? -> 원소 하나씩 분리해서

1 2 3 4 5

1 2 3

1 2 3 4

강사님 방법

```
# 리스트 저장하기
def twrite(file, list, sep= " ") :
    fp=open(file, "w")
    #행의 수 구하기
    if type(list)==type(list[0]) :
        rownum = len(list)
    else :
        rownum = 1

    for a in range(0, rownum) :
        for b in range (0, len(list[a])) :
            fp.writelines(str(list[a][b]) + sep)
        fp.writelines("₩n")
    fp.close()
```

?

근데 이것도 list 가 1 차원인 경우에는 오류가 발생함.

if 문 -> 이중리스트인지 확인하는 차원

강사님 방법 수정 !

```
# 리스트 저장하기
def twrite(file, list, sep= " ") :
    fp=open(file, "w")
    #행의 수 구하기
    if type(list)==type(list[0]) :
        rownum = len(list)
        for a in range(0, rownum) :
            for b in range (0, len(list[a])) :
                fp.writelines(str(list[a][b]) + sep)
            fp.writelines("₩n")
    else :
        for a in range(0,len(list)) :
            fp.writelines(str(list[a]) + sep)

    fp.close()
```

```
# 리스트 저장하기 수정한 코드
def twrite(file, list, sep= " ") :
    fp=open(file, "w")
    #행의 수 구하기
    if type(list)==type(list[0]) :
        rownum = len(list)
        for a in range(0, rownum) :
            for b in range (0, len(list[a])) :
                fp.writelines(str(list[a][b]) + sep)
            fp.writelines("₩n")
    else :
        for a in range(0,len(list)) :
            fp.writelines(str(list[a]) + sep)

    fp.close()
```

cf. 모듈로 저장하고 필요할 때마다 쓰기!

내가 한 것!

```
a=[['1', '2', '3', '4', '5'], ['1', '3', '5'], ['7', '8']]
def write_txt(file, l1, sep=" ") :
    of1=open(file, "w")
    result=[]
    for i in range(0, len(l1)) :
        for j in range(0, len(l1[i])) :
            of1.writelines(l1[i][j] + sep)
        of1.writelines("₩n")

    of1.close()
    print("---정상적으로 파일에 씀---")
```

평가

이차원인 리스트에는 적합하나, 리스트가 일차원인 경우 전달이 불가능하다.

ex. a=[1,2,3,4] / a[0][0] -- 오류

why? type(a[0]) => 'int'

그래도 잘 접근했당

#마지막을 sep 안 쓰게 하는 방법은 ?0?

```
# 리스트 저장하기
def twrite(file, list, sep= " ") :
    fp=open(file, "w")
    #행의 수 구하기
    if type(list)==type(list[0]) :
        rownum = len(list)
        for a in range(0, rownum) :
            for b in range (0, len(list[a])-1) :
                fp.writelines(str(list[a][b]) + sep)
            fp.writelines(str(list[a][len(list[a])-1]))
            fp.writelines("₩n")
    else :
        for a in range(0,len(list)-1) :
            fp.writelines(str(list[a]) + sep)
        fp.writelines(str(list[len(list)-1]))

    fp.close()
```

마지막 거는 따로 뽑도록 했음!