변수의 사용 범위 알아보기  
파이썬 스크립트에서 변수를 만들면 다음과 같이 함수 안에서도 사용할 수 있습니다.

|  |
| --- |
| x = 10 # 전역 변수 def foo():  print(x) # 전역 변수 출력 foo() print(x) # 전역 변수 출력 - 실행 결과 - 10 10 |

foo 함수에서 함수 바깥에 있는 변수 x의 값을 출력했습니다.  
함수를 포함하여 스크립트 전체에서 접근할 수 있는 변수를  
전역 변수(global variable)라고 부릅니다.  
  
특히 전역 변수에 접근할 수 있는 범위를  
전역 범위(global scope)라고 합니다.  
  
그럼 변수 x를 함수 foo 안에서 만들면 어떻게 될까요?

|  |
| --- |
| def foo():  x = 10 # foo의 지역 변수  print(x) # foo의 지역 변수 출력  foo() print(x) # 에러. foo의 지역 변수는 출력할 수 없음  - 실행 결과 - 10 Traceback (most recent call last):  File "C:\project\local\_variable.py", line 6, in <module>  print(x) # 에러. foo의 지역 변수는 출력할 수 없음 NameError: name 'x' is not defined |

실행을 해보면  
x가 정의되지 않았다는 에러가 발생합니다.  
변수 x는 함수 foo 안에서 만들었기 때문에  
foo의 지역 변수(local variable)입니다.  
  
지역 변수는  
변수를 만든 함수 안에서만 접근할 수 있고,  
함수 바깥에서는 접근할 수 없습니다.  
지역 변수를 접근할 수 있는 범위를  
지역 범위(local scope)라고 합니다.

함수 안에서 전역 변수 변경하기  
만약 함수 안에서 전역 변수의 값을 변경하면 어떻게 될까요?

|  |
| --- |
| global\_local\_variable.py x = 10 # 전역 변수 def foo():  x = 20 # x는 foo의 지역 변수  print(x) # foo의 지역 변수 출력  foo() print(x) # 전역 변수 출력  실행 결과 20 10 |

함수 foo 안에서 x = 20처럼 x의 값을 20으로 변경했습니다.  
하지만 함수 바깥에서 print로 x의 값을 출력해보면 10이 나옵니다.   
겉으로 보기에는 foo 안의 x는 전역 변수인 것 같지만   
실제로는 foo의 지역 변수입니다.   
전역 변수 x가 있고, foo에서 지역 변수 x를 새로 만들게 됩니다.   
이 둘은 이름만 같을 뿐 서로 다른 변수입니다.  
  
함수 안에서 전역 변수의 값을 변경하려면   
global 키워드를 사용해야 합니다.   
  
  
  
다음과 같이 함수 안에서 global에 전역 변수의 이름을 지정해줍니다.

|  |
| --- |
| global 전역변수  function\_global\_keyword.py  x = 10 # 전역 변수  def foo():  global x # 전역 변수 x를 사용하겠다고 설정  x = 20 # x는 전역 변수  print(x) # 전역 변수 출력    foo()  print(x) # 전역 변수 출력  실행 결과  20  20 |

이제 함수 안에서 x를 20으로 변경하면   
함수 바깥에서 x를 출력했을 때 20이 나옵니다.   
이렇게 함수 안에서 변수를 global로 지정하면   
전역 변수를 사용하게 됩니다.  
  
만약 전역 변수가 없을 때 함수 안에서 global을 사용하면 해당 변수는 전역 변수가 됩니다.

|  |
| --- |
| # 전역 변수 x가 없는 상태  def foo():  global x # x를 전역 변수로 만듦  x = 20 # x는 전역 변수  print(x) # 전역 변수 출력    foo()  print(x) # 전역 변수 출력 |

참고 | 네임스페이스

파이썬에서 변수는 네임스페이스(namespace, 이름공간)에 저장됩니다.   
다음과 같이 locals 함수를 사용하면 현재 네임스페이스를 딕셔너리 형태로 출력할 수 있습니다.  
x = 10  
locals()  
{'\_\_name\_\_': '\_\_main\_\_', '\_\_doc\_\_': None, '\_\_package\_\_': None, '\_\_loader\_\_': <class '\_frozen\_importlib.BuiltinImporter'>, '\_\_spec\_\_': None, '\_\_annotations\_\_': {}, '\_\_builtins\_\_': <module 'builtins' (built-in)>, 'x': 10}  
출력된 네임스페이스를 보면 'x': 10처럼 변수 x와 값 10이 저장되어 있습니다.  
여기서는 전역 범위에서 네임스페이스를 출력했으므로 전역 네임스페이스를 가져옵니다.

마찬가지로 함수 안에서 locals를 사용할 수도 있습니다.

def foo():  
 x = 10  
 print(locals())  
 foo()  
{'x': 10}

네임스페이스를 보면 'x': 10만 저장되어 있습니다. 이때는 지역 범위에서 네임스페이스를 출력했으므로 지역 네임스페이스를 가져옵니다.

변수(variable)

함수(function)

연산자(operator)

제어문(control statements)

자료형(data types)

함수(function)

def function\_name(parameter):

ststement

ststement

return x

입력값을 주로 인자(argument)

글자들의 한 묶음을 하나의 문자열(string)

변수를 정의(define)

들여쓰기 된 코드 블럭 indented block

변수의 스코프(scope)

전역 스코프(global scope)

지역 스코프(local scope)

인자(argument)를 받는 변수를 매개변수(parameter)

함수가 받을 매개변수의 기본값을 지정해 주면, 입력값이 없을 경우 해당 기본값을 사용하게 됩니다.

def say\_hi\_default(name='somebody'):

if 명제[비교 연산자(conditional operator)]

and 논리 연산자(logical operator)

반복문들을 루프(loop, 고리)라고 부릅니다.

반복문이 끝없이 계속 실행되는 경우를 무한 루프(infinite loop)

하나씩 순서대로 넘길(iterate) 수 있는 값들을 반복 가능한(iterable, 이터러블) 객체 라고 합니다

NoneType : None : 그냥 아무것도 없음을 표시하기 위해 만들어놓은 값

escape  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
파일에서 문자열 읽기

변수 = 파일객체.read()

-----

file = open('hello.txt', 'r') # hello.txt 파일을 읽기 모드(r)로 열기. 파일 객체 반환

s = file.read() # 파일에서 문자열 읽기

print(s) # Hello, world!

file.close() # 파일 객체 닫기

자동으로 파일 객체 닫기

with open(파일이름, 파일모드) as 파일객체:

코드

--------

with open('hello.txt', 'r') as file: # hello.txt 파일을 읽기 모드(r)로 열기

s = file.read() # 파일에서 문자열 읽기

print(s) # Hello, world!

문자열 여러 줄을 파일에 쓰기, 읽기

반복문으로 문자열 여러 줄을 파일에 쓰기

----------

with open('hello.txt', 'w') as file: # hello.txt 파일을 쓰기 모드(w)로 열기

for i in range(3):

file.write('Hello, world! {0}\n'.format(i))

문자열 끝에 개행 문자 \n를 지정해주어야 줄바꿈이 됩니다

리스트에 들어있는 문자열을 파일에 쓰기

파일객체.writelines(문자열리스트)

--------

lines = ['안녕하세요.\n', '파이썬\n', '코딩 도장입니다.\n']

with open('hello.txt', 'w') as file: # hello.txt 파일을 쓰기 모드(w)로 열기

file.writelines(lines)

--

writelines는 리스트에 들어있는 문자열을 파일에 씁니다.

특히 writelines를 사용할 때는 반드시 리스트의 각 문자열 끝에 개행 문자 \n을 붙여주어야 합니다

파일의 내용을 한 줄씩 리스트로 가져오기

변수 = 파일객체.readlines()

----------

with open('hello.txt', 'r') as file: # hello.txt 파일을 읽기 모드(r)로 열기

lines = file.readlines()

print(lines)

파일의 내용을 한 줄씩 읽기

변수 = 파일객체.readline()

-------

with open('hello.txt', 'r') as file: # hello.txt 파일을 읽기 모드(r)로 열기

line = None # 변수 line을 None으로 초기화

while line != '':

line = file.readline()

print(line.strip('\n')) # 파일에서 읽어온 문자열에서 \n 삭제하여 출력

파일에 문자열이 몇 줄이나 있는지 모르기 때문에 readline으로 파일을 읽을 때는 while 반복문을 활용해야 합니다.

while은 특정 조건이 만족할 때 계속 반복하므로 파일의 크기에 상관없이 문자열을 읽어올 수 있습니다.

readline은 더 이상 읽을 줄이 없을 때는 빈 문자열을 반환하는데, while에는 이런 특성을 이용하여 조건식을 만들어줍니다.

즉, line != ''와 같이 빈 문자열이 아닐 때 계속 반복하도록 만듭니다.

반복문 안에서는 line = file.readline()과 같이 문자열 한 줄을 읽어서 변수 line에 저장해주면 됩니다.

특히 변수 line은 while로 반복하기 전에 None으로 초기화해줍니다.

만약 변수 line을 만들지 않고 while을 실행하면 없는 변수와 빈 문자열 ''을 비교하게 되므로 에러가 발생합니다.

line을 None이 아닌 ''로 초기화하면 처음부터 line != ''는 거짓이 되므로 반복을 하지 않고 코드가 그냥 끝나버립니다.

while을 사용할 때는 이 부분을 주의해주세요.

---------

line = None # 변수 line을 None으로 초기화

while line != '':

문자열을 출력할 때는

------

print(line.strip('\n')와 같이 strip 메서드로 \n을 삭제했습니다.

왜냐하면 파일에서 읽어온 문자열에는 '안녕하세요.\n'와 같이 \n이 이미 들어있기 때문입니다.

만약 strip('\n')을 생략하면 문자열 한 줄을 출력할 때마다 빈 줄이 계속 출력됩니다.

즉, 문자열 안에 든 \n과 print가 출력하는 \n 때문에 줄바꿈이 두 번 일어납니다.

for 반복문으로 파일의 내용을 줄 단위로 읽기

------

with open('hello.txt', 'r') as file: # hello.txt 파일을 읽기 모드(r)로 열기

for line in file: # for에 파일 객체를 지정하면 파일의 내용을 한 줄씩 읽어서 변수에 저장함

print(line.strip('\n')) # 파일에서 읽어온 문자열에서 \n 삭제하여 출력

참고 | 파일 객체는 이터레이터

파일 객체는 이터레이터입니다.

따라서 변수 여러 개에 저장하는 언패킹(unpacking)도 가능합니다

-------

file = open('hello.txt', 'r')

a, b, c = file

a, b, c

('안녕하세요.\n', '파이썬\n', '코딩 도장입니다.\n')

물론 a, b, c = file과 같이 사용하려면 hello.txt에는 문자열 3줄이 들어있어야 합니다.

즉, 할당할 변수의 개수와 파일에 저장된 문자열의 줄 수가 일치해야 합니다

파이썬 객체를 파일에 저장하기, 가져오기

파이썬은 객체를 파일에 저장하는 pickle 모듈을 제공합니다.

다음과 같이 파이썬 객체를 파일에 저장하는 과정을 피클링(pickling)이라고 하고,

파일에서 객체를 읽어오는 과정을 언피클링(unpickling)이라고 합니다.

피클링 : 파이썬 객체를 파일에 저장하기

-------

import pickle

name = 'james'

age = 17

address = '서울시 서초구 반포동'

scores = {'korean': 90, 'english': 95, 'mathematics': 85, 'science': 82}

with open('james.p', 'wb') as file: # james.p 파일을 바이너리 쓰기 모드(wb)로 열기

pickle.dump(name, file)

pickle.dump(age, file)

pickle.dump(address, file)

pickle.dump(scores, file)

소스 코드를 실행하면 .py 파일이 있는 폴더에 james.p 파일이 생성됩니다. (다른 확장자를 사용해도 상관없습니다.)

특히 pickle.dump로 객체(값)를 저장할 때는 open('james.p', 'wb')와 같이 파일 모드를 'wb'로 지정해야 합니다.

b는 바이너리(binary)를 뜻하는데, 바이너리 파일은 컴퓨터가 처리하는 파일 형식입니다.

따라서 메모장 같은 텍스트 편집기로 열어도 사람이 알아보기 어렵습니다.

언피클링 : 파일에서 파이썬 객체 읽기

-------

import pickle

with open('james.p', 'rb') as file: # james.p 파일을 바이너리 읽기 모드(rb)로 열기

name = pickle.load(file)

age = pickle.load(file)

address = pickle.load(file)

scores = pickle.load(file)

print(name)

print(age)

print(address)

print(scores)

언피클링은 pickle 모듈의 load를 사용합니다.

언피클링을 할 때는 반드시 파일 모드를 바이너리 읽기 모드 'rb'로 지정해야 합니다

앞에서 james.p 파일을 저장할 때 pickle.dump를 네 번 사용했습니다.

마찬가지로 파일에서 객체(값)를 가져올 때도 pickle.load를 네 번 사용해야 합니다.

즉, name, age, address, scores 순으로 저장했으므로 가져올 때도 같은 순서로 가져오면 됩니다.

참고 | 다른 파일 모드는 없나요?

사실 파일 모드는 조합에 따라 여러 종류가 있습니다.

읽기 'r', 쓰기 'w' 이외에 추가 'a', 배타적 생성 'x'도 있습니다.

추가 모드는 이미 있는 파일에서 끝에 새로운 내용을 추가할 때 사용하고,

배타적 생성 모드는 파일이 이미 있으면 에러(FileExistsError)를 발생시키고 없으면 파일을 만듭니다.

'x'는 베타적 생성(exclusive creation)의 x입니다

또한, 파일의 형식도 함께 지정할 수 있는데, 텍스트 모드 't'와 바이너리 모드 'b'가 있습니다.

이 파일 형식과 읽기, 쓰기 모드를 조합한 텍스트 모드 'rt', 'wt'는 파일을 텍스트 모드로 엽니다.

특히 텍스트 모드는 생략할 수 있어서 그냥 'r', 'w'도 텍스트 모드입니다.

그리고 바이너리 모드 'rb', 'wb' 등은 피클링을 사용하거나 바이너리 데이터를 직접 저장할 때 사용합니다.

그다음에 '+'가 있는데 파일을 읽기/쓰기 모드로 엽니다.

이 모드는 'r+t', 'w+t', 'r+', 'w+', 'r+b', 'w+b' 등으로 조합할 수 있으며

읽기/쓰기 모드인 것은 같지만 파일 처리 방법이 조금씩 다릅니다.

함수 사용하기

파이썬은 함수(function)라는 기능을 제공하는데 특정 용도의 코드를 한 곳에 모아 놓은 것을 뜻합니다.

함수는 처음 한 번만 작성해 놓으면 나중에 필요할 때 계속 불러 쓸 수 있습니다.

print, input 등도 모두 파이썬에서 미리 만들어 둔 함수입니다.

코드의 용도를 구분할 수 있다.

코드를 재사용할 수 있다.

실수를 줄일 수 있다.

함수 만들기

def 함수이름():

코드

-----

Hello, world!'를 출력하는 함수

def hello():

print('Hello, world!')

함수 호출하기

함수()

-----

hello()

함수의 실행 순서(hello 함수는 다음과 같은 순서로 실행됩니다.)

---

파이썬 스크립트 최초 실행

hello 함수 호출

hello 함수 실행

print 함수 실행 및 'Hello, world!' 출력

hello 함수 종료

파이썬 스크립트 종료

빈 함수 만들기(내용이 없는 빈 함수를 만들 때는 코드 부분에 pass를 넣어줍니다.)

---------

def hello():

pass

pass는 아무 일을 하지 않아도 함수의 틀을 유지할 필요가 있을 때 사용합니다

def 함수이름(매개변수1, 매개변수2):

코드

---------

def 함수이름(매개변수):

"""독스트링"""

코드

def 함수이름(매개변수):

"""

여러 줄로 된

독스트링

"""

코드

-----------  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
print(...)

    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.

    Optional keyword arguments:

    file:  a file-like object (stream); defaults to the current sys.stdout.

    sep:   string inserted between values, default a space.

    end:   string appended after the last value, default a newline.

    flush: whether to forcibly flush the stream.

# 값을 여러 개 출력하기

print('문자 열')

print(값1, 값2, 값3)

print(변수1, 변수2, 변수3)

# sep로 값 사이에 문자 넣기(sep = separator)

# 제어 문자는 화면에 출력되지는 않지만 출력 결과를 제어하는 문자.

# 제어 문자는 \로 시작하는 이스케이프 시퀀스

# \n: 다음 줄로 이동하며 개행이라고도 부릅니다.

# \t: 탭 문자, 키보드의 Tab 키와 같으며 여러 칸을 띄웁니다.

# \\: \ 문자 자체를 출력할 때는 \를 두 번 써야 합니다.

print(값1, 값2, sep='문자 또는 문자열')

print(변수1, 변수2, sep='문자 또는 문자열')

print(1, 2, 3, sep='\n')

print('1\n2\n3')    # 문자열 안에 \n을 사용하여 줄바꿈

end 사용하기

print(값, end='문자 또는 문자열')

print(변수, end='문자 또는 문자열')

print(1, end='')    # end에 빈 문자열을 지정하면 다음 번 출력이 바로 뒤에 오게 됨

print(1, end=' ')    # end에 공백 한 칸 지정

문자열 사용하기

먼저 파이썬 프롬프트에서 문자열 'Hello, world!'를 출력해보겠습니다.

hello = 'Hello, world!'       # ' '(작은따옴표)로 문자열를 묶어서 입력.

hello = "Hello, world!"       # " "(큰따옴표)로 묶는 방법입니다.

hello = '''Hello, world!'''   # '''(작은따옴표 3개)로 묶을 수도 있습니다.

hello = """Hello, world!"""   # """(큰따옴표 3개)로 묶을 수도 있습니다.

hello

'Hello, world'                    # 'Hello, world!' 출력

여러 줄로 된 문자열(multiline string)을 사용.

'''로 시작하고 Hello, world!를 입력한 다음에 엔터 키를 누르면 다음 줄로 이동합니다.

이런 방식으로 문자열을 계속 입력하고 마지막 줄에서 '''로 닫은 뒤 엔터

hello = '''Hello, world!         # '''(""")로 시작

안녕하세요.                       # 문자열

Python입니다.'''                  # '''(""")로 끝냄.

print(hello)     # print 함수로 hello의 내용을 출력해보면 입력한 문자열 3줄이 출력

Hello, world!

안녕하세요.

Python입니다.

이처럼 여러 줄로 된 문자열은 '''로 시작하여 '''로 끝납니다.

물론 """로 시작하여 """로 끝내도 됩니다.

문자열을 표현할 때 작은따옴표, 큰따옴표 여러 가지 방식을 사용하는 이유

문자열을 사용하다 보면 문자열 안에 작은따옴표나 큰따옴표를 넣어야 할 경우

s = "Python isn't difficult", 'He said "Python is easy"'

참고 | 문자열에 따옴표를 포함하는 다른 방법 - 작은따옴표 앞에 \(역슬래시)를 붙이면 됩니다.

print('Python isn\'t difficult')      # "Python isn't difficult"

이스케이프(escape).

이처럼 문자열 안에 ', " 등의 특수 문자를 포함하기 위해 앞에 \를 붙이는 방법

참고 | 따옴표 세 개로 묶지 않고 여러 줄로 된 문자열 사용하기 - 문자열 안에 개행 문자(\n)

print('Hello\nPython')       # \n(개행 문자)

Hello

Python

참고 | 한글 문자열 출력이 안 될 때 - 한글 문자열을 넣었을 때 에러가 나는 경우.

C:\project>python string\_multiline\_quote.py

  File "string\_multiline\_quote.py", line 1

SyntaxError: Non-UTF-8 code starting with '\xbe' in file string\_multiline\_quote.py on line 1,

but no encoding declared; see http://python.org/dev/peps/pep-0263/ for details

에러가 나는 이유는 .py 파일을 UTF-8이 아닌 CP949로 저장했기 때문입니다.

이때는 스크립트 파일을 UTF-8로 저장하면 됩니다.

메모장에서 UTF-8로 저장하려면 파일(F) > 다른 이름으로 저장(A)… > 인코딩(E)에서 UTF-8을 선택한 뒤 저장.

인코딩 문제를 예방하려면 파이썬 IDLE, PyCharm 등 파이썬 전용 편집기나 개발 도구를 사용하면 됩니다.

파이썬 셸의 >>>에서 문자열을 그대로 출력하면 작은따옴표도 함께 출력됩니다

출력 결과가 문자열이라는 것을 정확하게 표현하기 위해 작은따옴표로 묶인 문자열이 출력됩니다.

*# 학습 목표*

*# 중복된 데이터를 찾아 제거할 수 있고, 결측치(missing data)를 제거하거나 채워 넣을 수 있다.*

*# 데이터를 정규화시킬 수 있다.*

*# 이상치(outlier)를 찾고, 이를 처리할 수 있다.*

*# 범주형 데이터를 원-핫 인코딩할 수 있다.*

*# 연속적인 데이터를 구간으로 나눠 범주형 데이터로 변환할 수 있다.*

*# 학습 목차*

*# 결측치(Missing Data)*

*# 중복된 데이터*

*# 이상치(Outlier)*

*# 정규화(Normalization)*

*# 원-핫 인코딩(One-Hot Encoding)*

*# 구간화(Binning)*

tuple : 여러 데이터를 저장 할 수 있는 컨테이너형 변수

- 특징 immutable, indexing, slicing, 시퀀스형

- 튜플은 요소를 일렬로 저장하지만, 요소를 변경, 추가, 삭제를 할 수 없습니다.

튜플 생성 : ( )에 각 값을  ,(콤마)로 구분, 또는 괄호로 묶지 않고 값만 콤마로 구분. 각 값을 요소(element)라함

튜플은 문자열, 정수, 실수, 불 등 모든 자료형을 저장할 수 있으며 자료형을 사용가능

보통 튜플은 요소가 절대 변경되지 않고 유지되어야 할 때 사용. 요소를 변경하게 되면 에러가 발생

튜플 = (값, 값, 값), 튜플 = 값, 값, 값

요소가 한 개인 튜플을 만들 때

( ) 안에 값 한 개를 넣고 ,(콤마)를 붙임. 또는, 괄호로 묶지 않고 값 한 개에 ,를 붙임.

튜플 = (값, ), 튜플 = 값,

range를 사용하여 튜플 만들기

튜플 = tuple(range(횟수))

튜플을 리스트로 만들고 리스트를 튜플로 만들기

튜플과 리스트는 요소를 변경, 추가, 삭제할 수 없는거만 다를 뿐 기능과 형태는 같다.

따라서 튜플을 리스트로 만들거나 리스트를 튜플로 만들 수도 있습니다.

tuple 안에 리스트를 넣으면 새 튜플이 생성.

a = [1, 2, 3]

tuple(a)            # (1, 2, 3) 튜플생성

반대로 list 안에 튜플을 넣으면 새 리스트가 생성.

b = (4, 5, 6)

list(b)             # [4, 5, 6] 리스트생성

참고  tuple(list)안에 문자열을 넣으면? 문자 튜플(문자 리스트)이 생성.

list('Hello')     # ['H', 'e', 'l', 'l', 'o']

tuple('Hello')    # ('H', 'e', 'l', 'l', 'o')

참고  튜플(리스트)을 사용헤 변수 여러 개를 한 번에 만들 수 있음. 변수의 개수와 튜플(리스트))의 요소 개수는 같아함.

a, b, c = [1, 2, 3]

print(a, b, c)        # 1 2 3

d, e, f = (4, 5, 6)

print(d, e, f)        # 4 5 6

튜플(리스트) 변수로도 변수 여러 개를 만들 수 있다. - 튜플 언패킹(tuple unpacking),

y = (4, 5, 6)        # 튜플 패킹

d, e, f = y          # 튜플 언패킹(tuple unpacking)

print(d, e, f)       # 4 5 6

x = [1, 2, 3]        # # 리스트 패킹

a, b, c = x          # 리스트 언패킹(list unpacking)

print(a, b, c)       # 1 2 3

튜플 패킹(tuple packing), 리스트 패킹(list packing)은 변수에 튜플(리스트)을 할당하는 과정을 뜻함.

a = [1, 2, 3]    # 리스트 패킹

b = (1, 2, 3)    # 튜플 패킹

c = 1, 2, 3      # 튜플 패킹

list : 여러 데이터를 저장 할 수 있는 컨테이너형 변수

- 특징 mutable, indexing, slicing, 시퀀스형

리스트 만들기

리스트 = [값, 값, 값]

리스트 생성 : [ ]에 각 값을  ,(콤마)로 구분, 또는 list()로 각 값은 ,(콤마)로 구분, 각 값을 요소(element)라함

리스트는 문자열, 정수, 실수, 불 등 모든 자료형을 저장할 수 있으며 자료형을 사용가능

빈 리스트 만들기

리스트 = []

리스트 = list()

빈 리스트를 만들 때는 [ ]만 지정하거나 list를 사용하면 됩니다.

빈 리스트는 쓸모가 없을 것 같지만, 보통 빈 리스트를 만들어 놓은 뒤에 새 값을 추가하는 방식으로 사용합니다.

range를 사용하여 리스트 만들기

리스트 = list(range(횟수)), list(range(시작, 끝, 증가폭))