

Python 및 전처리

Day 1

- Python 소개, 실습환경 셋팅
- Python 기본1
 - 자료구조
 - 조건문, 반복문
 - split, join, enumerate, zip
 - 함수, lambda, Map, Reduce
- Python 기본2
 - 클래스
 - 모듈, 패키지
 - 파일 I/O

Python 소개

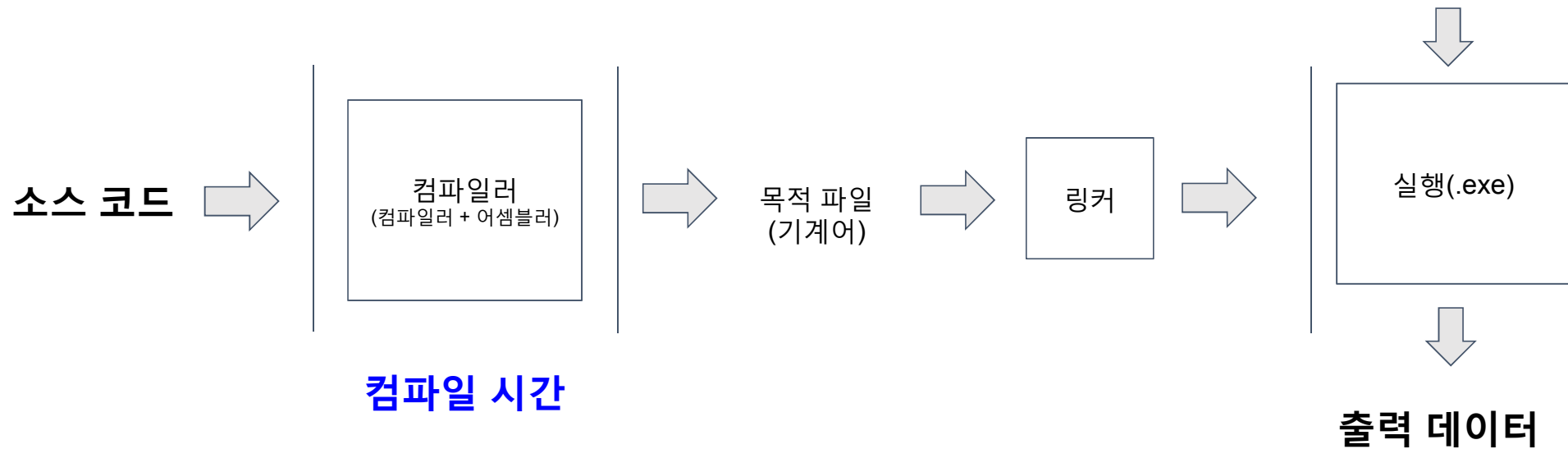
- 1991년 귀도 반 로섬(Guido van Rossum) 발표
- Python 특징
 - 인터프리터(interpreter) 언어
 - 간결하고 쉬운 문법 (Simplicity, Easy)
 - 가독성 (Readability)
 - 풍부한 라이브러리로 개발 생산성이 높음 (Efficiency)
- 활용 분야
 - 웹 프로그래밍, 수치연산 프로그래밍 등
 - Dropbox, Instagram, Youtube
(Google engineering motto: Python where we can, C++ we must do.)



Python 소개

1) 인터프리터 언어

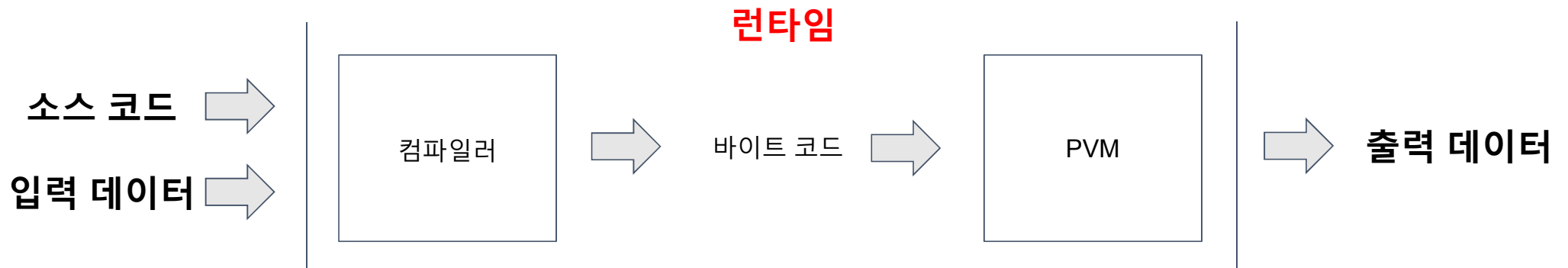
- 컴파일러 언어(C): 컴파일 시간과 런타임이 분리



Python 소개

1) 인터프리터 언어

- 인터프리터 언어(Python): 런타임에 동시에 분석과 실행



Python 소개

2) 간결한 코드

| C | JAVA | Python |
|--|--|---------------------------|
| <pre>int array[2]; array[0] = 1; array[1] = 2;</pre> | <pre>int[] array = new int[2]; array[0] = 1; array[1] = 2;</pre> | <pre>array = [1, 2]</pre> |

3) 들여쓰기 (Indentation)

| C | JAVA | Python |
|---|--|--|
| <pre>for (int a=0; a<3;a++) { print(a) }</pre> | <pre>for (int a=0; a<3;a++) { System.out.println(a) }</pre> | <pre>for a in range(3): print(a)</pre> |

Python PEP8

- Python Enhance Proposal 8 : Style Guide for Python Code
- 코드의 가독성, 일관성
 - <https://www.python.org/dev/peps/pep-0008/>
 - <https://spoqa.github.io/2012/08/03/about-python-coding-convention.html>
- **A Foolish Consistency is the Hobgoblin of Little Minds**
 - *멍청하게 일관성을 고집하는 것은 소인배의 발상이다*

Some other good reasons to ignore a particular guideline:

1. When applying the guideline would make the code less readable, even for someone who is used to reading code that follows this PEP.
2. To be consistent with surrounding code that also breaks it (maybe for historic reasons) -- although this is also an opportunity to clean up someone else's mess (in true XP style).

Python PEP8

- 들여쓰기(indent) : python에서 코드 블록 단위
 - 공백 2칸/4칸, tab 등을 사용
 - PEP 8 : 공백 4칸을 규정

C Language

```
1 if (condition_1) {  
2     print("result_1");  
3 } else if(condition_2){  
4     print("result_2");  
5 } else{  
6     print("result_3");  
7 }  
{} 를 사용하여 코드를 구분함
```

Python

```
1 if condition_1:  
2     print("result_1")  
3 elif condition_2:  
4     print("result_2")  
5 else:  
6     print("result_3")
```


Python PEP8

- 들여쓰기(indent) : python에서 코드 블록 단위

```
1 def indent_test():
2   print("Hllo world")
```

File "<ipython-input-11-c61e22956a4f>", line 2
 print("Hllo world")
 ^

IndentationError: expected an indented block

```
1 print("Hello World")
2   print("Hello World")
```

File "<ipython-input-8-874bdb4144ed>", line 2
 print("Hello World")
 ^

IndentationError: unexpected indent

```
1 for i in range(3):
2     print("Indet ###")
3     print("Indet Error")
```

File "<tokenize>", line 3
 print("Indet Error")
 ^

IndentationError: unindent does not match any outer indentation level

Python PEP8

- 들여쓰기(indent) : 좋은 예 vs 나쁜 예

좋은 예

```
# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                          var_three, var_four)

# More indentation included to distinguish this from the rest.
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)

# Hanging indents should add a level.
foo = long_function_name(
    var_one, var_two,
    var_three, var_four)
```

나쁜 예

```
# Arguments on first line forbidden when not using vertical alignment.
foo = long_function_name(var_one, var_two,
                          var_three, var_four)

# Further indentation required as indentation is not distinguishable.
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

Python PEP8

• 표현식, 구문에서의 공백문자 (Whitespace)

Example. 좋은 예 vs 나쁜 예

- Immediately inside parentheses, brackets or braces.

Yes: `spam(ham[1], {eggs: 2})`

No: `spam(ham[1], { eggs: 2 })`

- Between a trailing comma and a following close parenthesis.

Yes: `foo = (0,)`

No: `bar = (0,)`

- Immediately before a comma, semicolon, or colon:

Yes: `if x == 4: print x, y; x, y = y, x`

No: `if x == 4 : print x , y ; x , y = y , x`

- More than one space around an assignment (or other) operator to align it with another.

Yes:

`x = 1`

`y = 2`

`long_variable = 3`

No:

`x = 1`

`y = 2`

`long_variable = 3`

Python PEP8

- Naming Conventions
 - 1) Names to Avoid
 - 소문자 l, 대문자 O, 대문자 I
 - 2) Class Names : CapitalizedWord
 - 3) Function and Variable Name : lowercase_underscore
 - 4) Instance Variables
 - `_single_leading_underscore` (protected)
 - `__double_leading_underscore` (private)

Python IDE

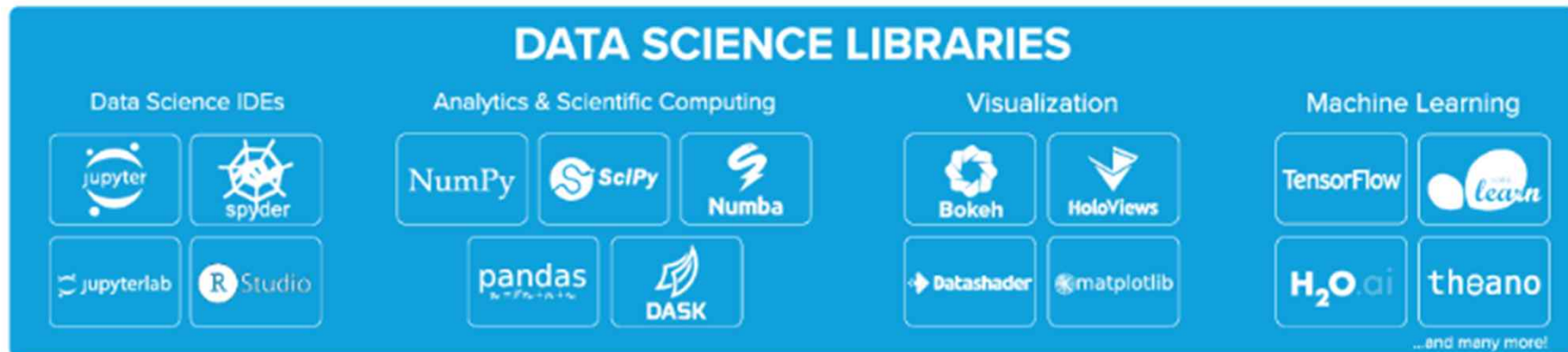


우리가 사용할 환경

python 개발 환경 구축

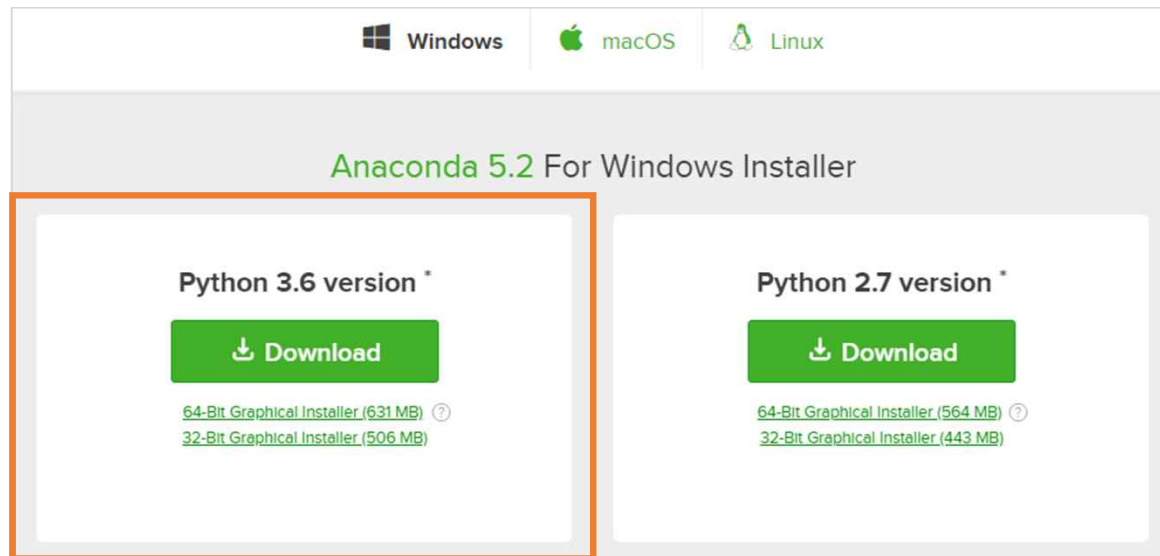
• ANACONDA.

- 데이터 사이언스 관련 패키지를 손쉽게 설치/관리할 수 있음
- 파이썬 버전별로 가상환경을 제공



python 설치하기 - Anaconda

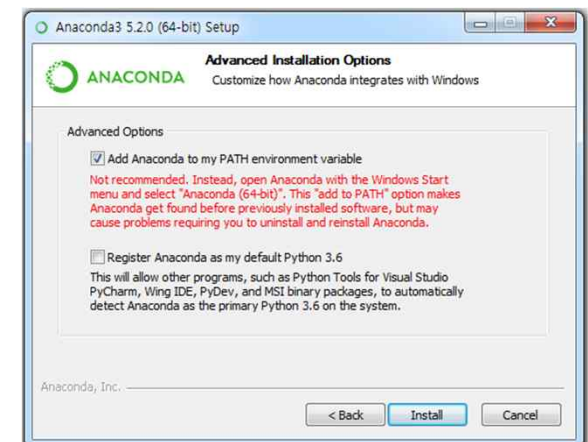
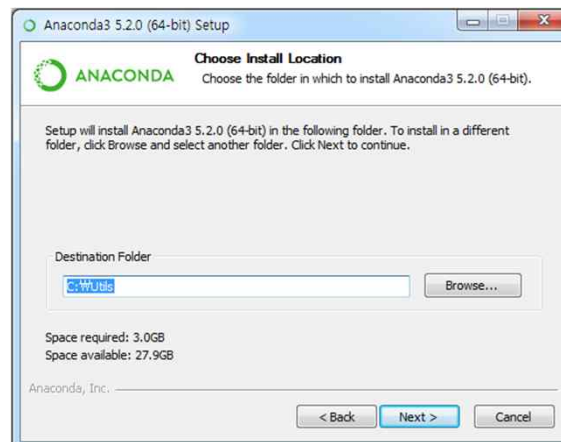
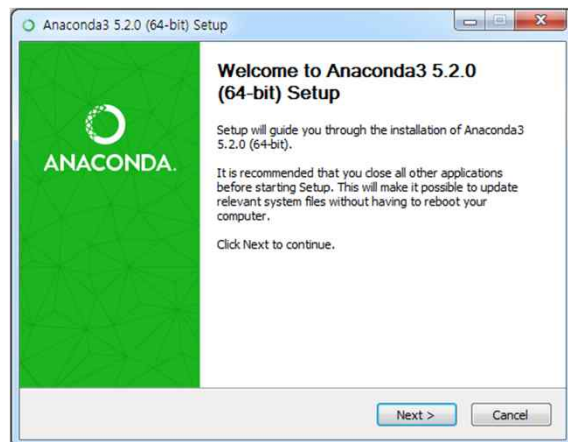
- Anaconda Download
 - <https://www.anaconda.com/download/>
 - Python version 3.6



*) '윈도우 > 컴퓨터 > 속성' 에서 32bit/64bit 확인 가능

python 설치하기 - Anaconda

- Anaconda Install
 - Anaconda3-5.2.0-Windows-x86_64.exe 실행
 - 라이선스 동의 및 설치 타입/경로 설정
 - [추가 옵션] 시스템 환경변수의 PATH 등록



python 설치하기 - Anaconda

- Anaconda Prompt 실행
 - python
 - “Hello World” 출력 확인

```
<base> C:\WUsers>python
Python 3.5.3 |Anaconda custom (64-bit)| <default, May 15 2017, 10:43:23> [MSC v.
1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print("Hello World !")
Hello World !
>>>
```

python 설치하기 - Anaconda

- 버전 확인 : `conda --version`
- 업데이트 : `conda update conda`
- 가상환경 생성
 - ex) `conda create --name {my_python} python=3.6`
 - ex) `activate {my_python}`
 - ex) `deactivate {my_python}`
- 가상환경 삭제
 - ex) `conda remove --name {my_python} -all`
- Jupyter Notebook 실행
 - 가상환경 activation 후, \$ **jupyter notebook** 입력

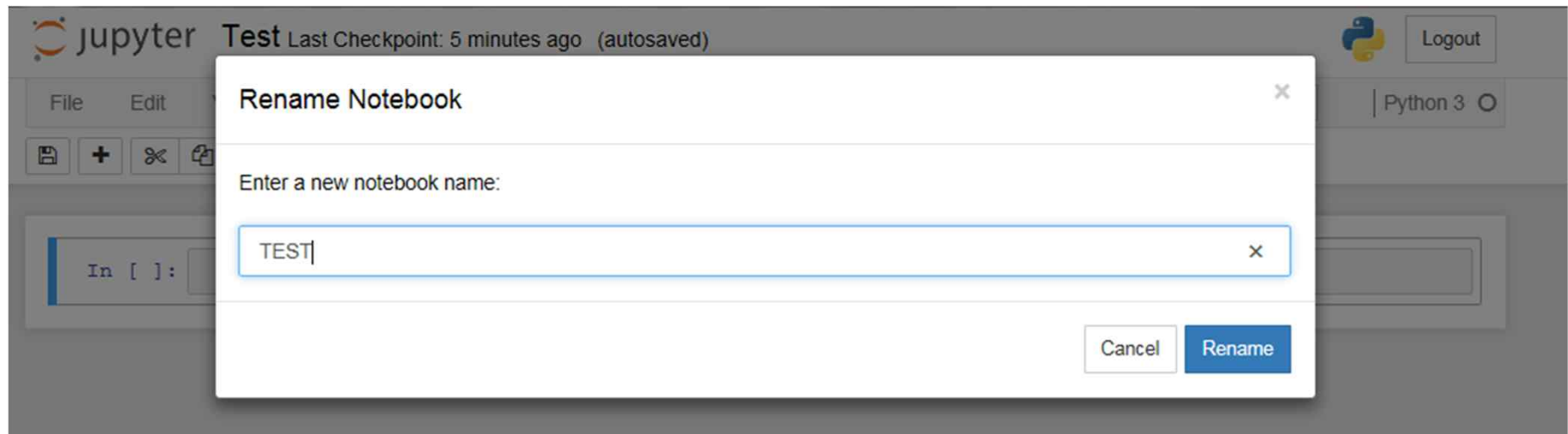
Jupyter Notebook 사용법

- 1. Python 코드 생성하기
 - New > Python 3



Jupyter Notebook 사용법

- 2. 파일 이름 변경하기



Jupyter Notebook 사용법

• 3. Cell 추가/삭제 단축키

```
In [1]: 1 # Edit Mode      : Enter
        2 # Command mode : ESC
```

Cell #1

```
In [ ]: 1 # Insert cell above : a
        2 # Insert cell below : b
```

Cell #2

```
In [ ]: 1 # Delete selected cell : dd
```

```
In [ ]: 1 # Copy selected cell : c
        2 # Cut  selected cell : x
```

```
In [ ]: 1 # paste cell above : Shift + v
        2 # paste cell below : v
```

Jupyter Notebook 사용법

- 4. Code 실행
 - 1) Shift + Enter : code 실행 후, cell 추가

```
In [10]: 1 a=3  
        2 b=5  
        3 a+b
```

```
Out[10]: 8
```

```
In [ ]: 1 |
```

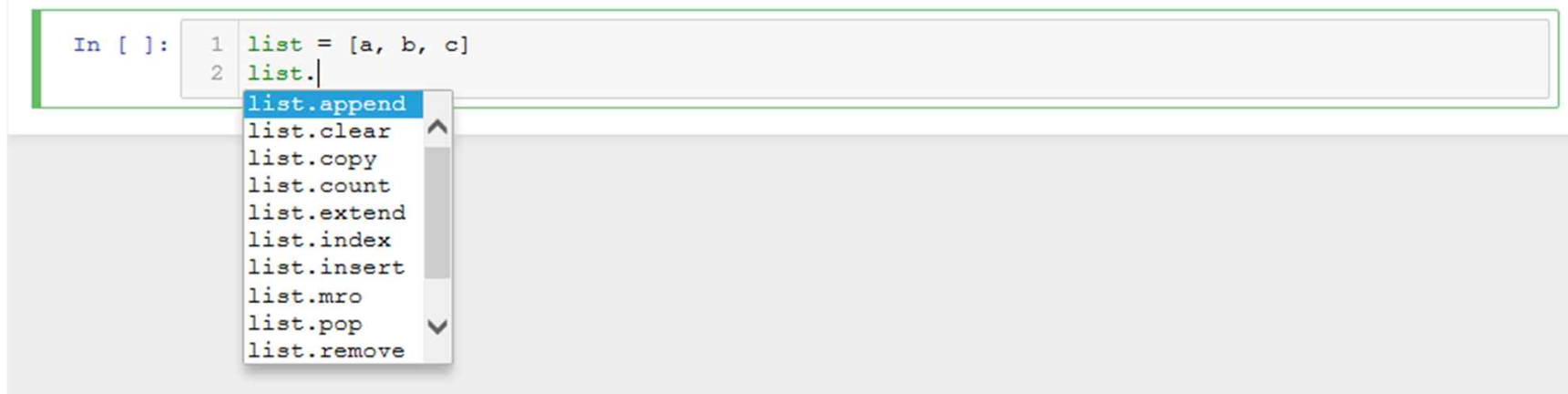
- 2) Ctrl + Enter : 선택된 code 실행

```
In [11]: 1 a=3  
        2 b=5  
        3 a+b|
```

```
Out[11]: 8
```

Jupyter Notebook 사용법

- 5. 코드 자동완성 : Tab



The screenshot shows a Jupyter Notebook cell with the following code:

```
In [ ]: 1 list = [a, b, c]
        2 list.|
```

A dropdown menu is displayed below the code, listing the following methods for `list`:

- `list.append` (highlighted)
- `list.clear`
- `list.copy`
- `list.count`
- `list.extend`
- `list.index`
- `list.insert`
- `list.mro`
- `list.pop`
- `list.remove`

Jupyter Notebook 사용법

- 6. Description : Shift + Tab, ?
 - 1) Shift + Tab

```
In [ ]: 1 list = [a, b, c]
        2 list
```

Init signature: list(self, /, *args, **kwargs)
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's items

- 2) ?

```
In [3]: list?
```

Init signature: list(self, /, *args, **kwargs)
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's items
Type: type

Chapter 1. 자료형

Chapter 1. 자료형

- Python 자료형

| 자료형 | Data Type |
|---------|--|
| 숫자형 | int (정수형), float (실수형) |
| Boolean | bool (True/False) |
| 군집형 | str (문자열) list (리스트) tuple (튜플) set (집합) dict (사전) |

Chapter 1. 자료형

• Python 자료형

- type 함수
- dir 함수

1. 파이썬에서 모든 것은 객체이다.

- type function

```
print(type(a))
```

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

- dir function

```
print(dir(a))
```

```
['_abs_', '_add_', '_and_', '_bool_', '_ceil_', '_class_', '_delattr_', '_dir_', '_divmod_', '_doc_',
'_eq_', '_float_', '_floor_', '_floordiv_', '_format_', '_ge_', '_getattr_', '_getnewargs_',
'_gt_', '_hash_', '_index_', '_init_', '_init_subclass_', '_int_', '_invert_', '_le_', '_lshift_',
'_lt_', '_mod_', '_mul_', '_ne_', '_neg_', '_new_', '_or_', '_pos_', '_pow_', '_radd_', '_rand_',
'_rdivmod_', '_reduce_', '_reduce_ex_', '_repr_', '_rfloordiv_', '_rlshift_', '_rmod_', '_rmul_',
'_ror_', '_round_', '_rpow_', '_rrshift_', '_rshift_', '_rsub_', '_rtruediv_', '_rxor_', '_setattr_',
'_sizeof_', '_str_', '_sub_', '_subclasshook_', '_truediv_', '_trunc_', '_xor_', 'bit_length',
'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```

dir은 객체가 자체적으로 가지고 있는 변수나 함수를 보여 준다.

Chapter 1. 자료형

- 1. List
 - List = [value1, value2, value3 ...]
 - 각 요소는 모든 자료형이 가능
 - 순서가 있고, 인덱싱을 통해 자료를 다루기 위한 자료형
 - List는 값의 생성, 삭제, 수정이 가능 (**mutable**)

- *Example*

```
1 list_char = ['a', 'b', 'c']  
2 list_str  = ['abc', 'bcd', 'cde']
```

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

```
1 list_float = [1.0, 2.0, 3.0, 4.0, 5.0]
```

```
1 list_bool = [True, False, True, True, False]
```

```
1 list_com = [1, 2, 3, "xyz", "abc"]
```

Chapter 1. 자료형

- 1-1) List Slicing
 - 여러 개의 데이터에 동시에 접근
 - *Example*

| 시작 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| A | 15 | 27 | 32 | 20 | 17 | 13 | 10 | 22 |
| 끝 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

- 리스트 이름[시작:끝] = 리스트의 시작 이상 끝 미만
- 리스트 이름[시작:] = 리스트의 시작 이상
- 리스트 이름[:끝] = 리스트의 끝 미만
- 리스트 이름[:] = 리스트 전체

Chapter 1. 자료형

- 1-1) List Slicing
 - 여러 개의 데이터에 동시에 접근
 - *Example*

| | | | | | | | | |
|----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 시작 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| A | 15 | 27 | 32 | 20 | 17 | 13 | 10 | 22 |
| 끝 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

```
>>>A = [15, 27, 32, 20, 17, 13, 10, 22]
```

```
>>> A[1:4]      [27, 32, 20]
```

```
>>> A[4:]       [17, 13, 10, 22]
```

```
>>> A[: -5]     [15, 27, 32]
```

```
>>> A[:]        [15, 27, 32, 20, 17, 13, 10, 22]
```

Chapter 1. 자료형

- 1-2) List 연결하기 (+)

- $LIST_1 + LIST_2 = [LIST_1 \text{의 요소}, LIST_2 \text{의 요소}]$

```
1 A = [1, 2, 3]
2 B = [[1, 2]]
3 A+B
```

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

- 1-3) List 반복하기 (*)

- $LIST_1 * n = [LIST_1 \text{의 요소}, LIST_1 \text{의 요소} \dots]$ (n번 반복)

```
1 A = [1, 2, 3]
2 A*3
```

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

Chapter 1. 자료형

• 1-4) List method

| 메소드 | 기능 |
|---------------|--------------------------------|
| append (x) | 데이터 x를 리스트 끝에 추가 |
| clear () | 리스트 비우기 |
| copy () | 리스트 복사 |
| count (x) | 데이터 값이 x인 요소의 개수 |
| extend (M) | 리스트에 리스트 M을 연결 |
| index (x) | 데이터 값이 x인 위치 출력 (여러 개인 경우 첫번째) |
| insert (i, x) | 데이터를 i번째 위치에 삽입 |
| pop () | 리스트의 지정한 값 한 개를 출력하고 삭제 |
| remove (x) | 리스트에서 데이터 값이 x인 것을 삭제 |
| reverse () | 리스트의 순서를 역순으로 바꿈 |
| sort () | 리스트 정렬 |

*) List method 조회

```
1 dir(list)
```

..... 중략

```
'__str__',
'__subclasshook__',
'append',
'clear',
'copy',
'count',
'extend',
'index',
'insert',
'pop',
'remove',
'reverse',
'sort']
```


Chapter 1. 자료형

• 1-4) 리스트 메소드 (method) – append, extend, insert

• Example

| | | |
|--|----------------------------------|---|
| <pre>1 A = [1, 2, 3] 2 B = [7, 8, 9]</pre> | <pre>1 A.append(4) 2 A</pre> | <div data-bbox="1821 837 2166 933"> append 데이터 x를 리스트 끝에 추가 </div> <div data-bbox="1821 1093 2166 1189"> extend 리스트에 리스트 M을 연결 </div> <div data-bbox="1821 1268 2166 1364"> insert (i, v) 데이터를 i 번째 위치에 삽입 </div> |
| | <pre>[1, 2, 3, 4]</pre> | |
| | <pre>1 A.append(B) 2 A</pre> | |
| | <pre>[1, 2, 3, [7, 8, 9]]</pre> | |
| | <pre>1 A.extend(B) 2 A</pre> | |
| | <pre>[1, 2, 3, 7, 8, 9]</pre> | |
| | <pre>1 A.insert(2, 10) 2 A</pre> | |
| | <pre>[1, 2, 10, 3]</pre> | |

Chapter 1. 자료형

• 1-4) 리스트 메소드 (method) – pop, remove, clear

• Example

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

```
1 A.pop(2)
2 A.pop()
3 A
```

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

pop

리스트의 지정한 값 한 개를
출력하고 삭제

```
1 A.remove(2)
2 A
```

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

remove

리스트에서 데이터 값이 x인 것을 삭제

```
1 A.clear()
2 A
```

```
[]
```

clear

리스트 비우기

Chapter 1. 자료형

• 1-4) 리스트 메소드 (method) – count, index, reverse, sort

• Example

| | | |
|----------------------------------|--------------------------------------|---|
| <pre>1 A = [1, 2, 3, 4, 1]</pre> | <pre>1 A.count(3) 2 A.count(1)</pre> | <div data-bbox="1668 687 1758 821" data-kind="parent" data-rs="2">}</div> <div data-bbox="1780 726 2163 821"> count 데이터 값이 x인 요소의 개수 </div> |
| | <pre>1 2</pre> | |
| | <pre>1 A.index(3) 2 A.index(1)</pre> | <div data-bbox="1668 869 1758 997" data-kind="parent" data-rs="2">}</div> <div data-bbox="1780 901 2163 1029"> index 데이터 값이 x인 위치 출력 (여러 개인 경우 첫번째) </div> |
| | <pre>2 0</pre> | |
| | <pre>1 A.reverse() 2 A</pre> | <div data-bbox="1668 1051 1758 1179" data-kind="parent" data-rs="2">}</div> <div data-bbox="1780 1093 2163 1189"> reverse 리스트의 순서를 역순으로 바꿈 </div> |
| | <pre>[1, 4, 3, 2, 1]</pre> | |
| | <pre>1 A.sort() 2 A</pre> | <div data-bbox="1668 1233 1758 1361" data-kind="parent" data-rs="2">}</div> <div data-bbox="1780 1268 2163 1364"> sort 리스트 정렬 </div> |
| | <pre>[1, 1, 2, 3, 4]</pre> | |

Chapter 1. 자료형

• 2. Tuple

- Tuple = (value1, value2, value3 ...)
- List와 같이 각 요소는 모든 자료형 가능
- List는 값의 생성/삭제/수정이 가능하지만 Tuple은 불가능 (**immutable**)

• Example

```
1 tuple_char = ('a', 'b', 'c')
2 tuple_str = ('abc', 'bcd', 'cde')
```

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

```
1 tuple_float = (1.0, 2.0, 3.0, 4.0, 5.0)
```

```
1 tuple_bool = (True, False, True, True, False)
```

```
1 tuple_1 = (1, 2, 'a', 'b')
2 tuple_1[0] = 'c'
```

```

TypeError                                 Traceback (most recent call last)
<ipython-input-9-c74b824c91ac> in <module>()
      1 tuple_1 = (1, 2, 'a', 'b')
----> 2 tuple_1[0] = 'c'

TypeError: 'tuple' object does not support item assignment

```

Chapter 1. 자료형

• 3. Set

- Set = { value1, value2, value3 ... }
- 집합에 관련된 연산을 쉽게 처리하기 위한 자료형
- 중복되지 않는 여러 개의 자료를 모아서 저장하는 경우
- 순서를 가지지 않음 (unordered)

• Example

```
1 set_int = {1, 2, 1, 2, 3}
2 print(set_int)
```

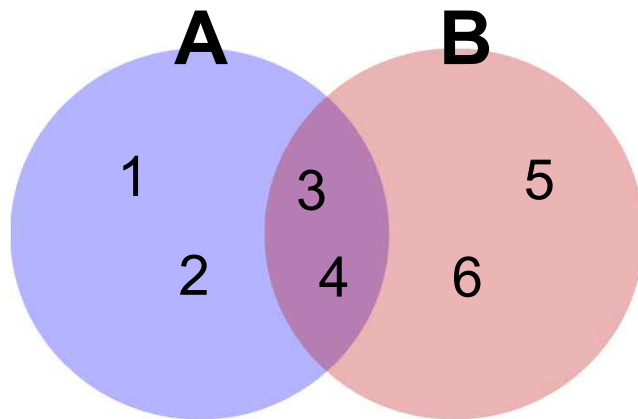
{1, 2, 3}

```
1 set_list = set(['a', 'b', 1, 2, 'a', 3])
2 print(set_list)
```

{'a', 1, 'b', 2, 3}

Chapter 1. 자료형

• 3-1). Set 연산



```
1 A = set([1, 2, 3, 4])
2 B = set([3, 4, 5, 6])
```

교집합

```
1 A & B
2 A.intersection(B)

{3, 4}
```

합집합

```
1 A | B
2 A.union(B)

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

차집합

```
1 A - B

{1, 2}

1 B.difference(A)

{5, 6}
```

Chapter 1. 자료형

- 4. dictionary
 - dictionary = { 'key1' : 'value1', 'key2' : 'value2', 'key3' : 'value3'... }
 - 대응관계를 가지고 있는 자료형
 - Key와 Value의 쌍으로 구성
 - key 정보를 통해서 Value에 접근

- *Example*

```
1 dict_int = {'key_1':1, 'key_2':2, 'key_3':3}
```

```
1 dict_float = {'key_1':1.0, 'key_2':2.0, 'key_3':3.0}
```

```
1 dict_str = {'key_1':'value_1', 'key_2':'value_2', 'key_3':'value_3'}
```

```
1 dict_1 = {'key_1':[1, 2], 'key_2':'value_2', 'key_3':3}
2 print(dict_1)
```

```
{'key_1': [1, 2], 'key_2': 'value_2', 'key_3': 3}
```

Chapter 1. 자료형

• 4-1) dictionary 값 추가, 수정, 삭제

• Example

```
1 pen = {'r' : 1000, 'g' : 500, 'b': 700}
```

```
1 pen['w'] = 300
2 pen
```

```
{'b': 700, 'g': 500, 'r': 1000, 'w': 300}
```

추가

```
1 pen['r'] = 650
2 pen
```

```
{'b': 700, 'g': 500, 'r': 650}
```

수정

```
1 del pen['g']
2 pen
```

```
{'b': 700, 'r': 1000}
```

삭제

Chapter 1. 자료형

• 4-2) dictionary method

| 메소드 | 기능 |
|------------|---------------------------------------|
| clear () | dictionary 비우기 |
| copy () | dictionary 복사 |
| items () | dictionary에 있는 모든 데이터 (Key, Value) 출력 |
| keys () | dictionary에 있는 Key만 반환 |
| values () | dictionary에 있는 Value만 반환 |
| update (D) | dictionary에 D를 추가 |

*) dict method 조회

```

1  dir(dict)

..... 중략
['__str__',
 '__subclasshook__',
 'clear',
 'copy',
 'fromkeys',
 'get',
 'items',
 'keys',
 'pop',
 'popitem',
 'setdefault',
 'update',
 'values']

```

Chapter 1. 자료형

• 4-2) dictionary method – copy, update, keys, values, items

• Example

| | | |
|--|--|-----------|
| <pre>1 pen = {'r': 1000, 'g': 500, 'b': 700}</pre> | <pre>1 pen2 = pen.copy() 2 print(pen2) 3 pen2['r']=1500 4 print(pen2) {'r': 1000, 'g': 500, 'b': 700} {'r': 1500, 'g': 500, 'b': 700}</pre> | } copy |
| <pre>1 pen2 = {'r': 300, 'w': 700} 2 pen.update(pen2) 3 print(pen) {'r': 300, 'b': 700, 'w': 700, 'g': 500}</pre> | } update | |
| <pre>1 pen.keys() dict_keys(['r', 'b', 'g'])</pre> | } keys values items | |
| <pre>1 pen.values() dict_values([1000, 700, 500])</pre> | | |
| <pre>1 pen.items() dict_items([('r', 1000), ('b', 700), ('g', 500)])</pre> | | |

Chapter2. 조건문과 반복문

Chapter2. 조건문과 반복문

• 1. if 문

```

1  if 조건문_1:
2      수행문1
3
4  elif 조건문_2:
5      수행문2
6
7  else:
8      수행문3

```

- } 수행문은 'indent' (들여쓰기)로 구분
- } 조건문이 여러 개인 경우, **elif**로 조건 추가

• Example

```

1  money = 2000
2
3  if money >= 3000:
4      print("Taxi")
5  elif money > 2000:
6      print("Bus")
7  else:
8      print("Walk")

```

Walk

수행문의 내용이 없는 경우 : **pass**

```

1  money = 2000
2
3  if money >= 3000:
4      print("Taxi")
5  elif money > 2000:
6      print("Bus")
7  else:
8      pass

```

Chapter2. 조건문과 반복문

- 2. in / not in
 - 있는지 없는지 확인할 때

- *Example 1.*

```
1 'a' in ['a', 'b', 'c']
```

True

```
1 'a' not in ['a', 'b', 'c']
```

False

- *Example 2.*

```
1 'j' in 'python'
```

False

```
1 'j' not in 'python'
```

True

```
1 'py' in 'python'
```

True

```
1 'py' not in 'python'
```

False

Chapter2. 조건문과 반복문

• 3. for 문

| | | |
|---|--|---|
| 1 | <code>for</code> 변수 <code>in</code> [...]: | } 문자열, List, Tuple, Set, dict, range() 함수 |
| 2 | 수행문 | |

| | |
|---|---|
| 1 | <code>for</code> f, s <code>in</code> [(1, 2), (3, 4), (5, 6)]: |
| 2 | <code>print</code> (f+s) |

3
7
11

| | |
|---|---|
| 1 | <code>for</code> i <code>in</code> <code>range</code> (1, 10, 3): |
| 2 | <code>print</code> (i) |

1
4
7

range(start, end, step)
 일정 간격의 정수 값을 가진 list를 생성해주는 함수

• 4. while 문

| | |
|---|-------------------------|
| 1 | <code>while</code> 조건문: |
| 2 | 수행문 |

| | |
|---|-------------------------------|
| 1 | count = 0 |
| 2 | <code>while</code> count < 3: |
| 3 | <code>print</code> (count) |
| 4 | count += 1 |

0
1
2

Chapter2. 조건문과 반복문

• 4-1. while문 Example

강제로 빠져나갈 때 : **break**

```
1 num = 0
2 while 1:
3     print (num)
4     if num == 10:
5         break
6     num += 1
```

처음으로 돌아갈 때 : **continue**

```
1 a = 0
2 while a < 10:
3     a += 1
4     if a%2 == 0: continue
5     print(a)
```

Chapter2. 조건문과 반복문

• 4-1. while문 Example

강제로 빠져나갈 때 : **break**

```

1 num = 0
2 while 1:
3     print (num)
4     if num == 10:
5         break
6     num += 1

```

0
1
2
3
4
5
6
7
8
9
10

처음으로 돌아갈 때 : **continue**

```

1 a = 0
2 while a < 10:
3     a += 1
4     if a%2 == 0: continue
5     print(a)

```

1
3
5
7
9

while문을 빠져나가지 않고, 조건문으로 다시 돌아가고 싶은 경우

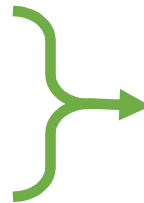
Chapter2. 조건문과 반복문

- 5. List Comprehension
 - 리스트 내에 for문을 포함시키는 것
 - 5-1) 반복문과 List Comprehension

```
1 [ <the_expression> for <the_element> in <the_iterable> ]
```

```
1 m = []
2 for n in range(5):
3     m.append(n * 3)
4 print(m)
```

[0, 3, 6, 9, 12]



```
1 m = [n*3 for n in range(5)]
2 print(m)
```

[0, 3, 6, 9, 12]

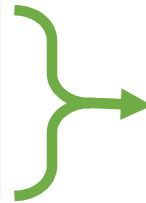
Chapter2. 조건문과 반복문

- 5. List Comprehension
 - 5-2) List Comprehension 안의 조건문

```
1 [ <expression> for <element> in <iterable> if <condition> ]
```

```
1 m = []
2 for n in range(10):
3     if n%2 == 0:
4         m.append(n)
5 print(m)
```

[0, 2, 4, 6, 8]



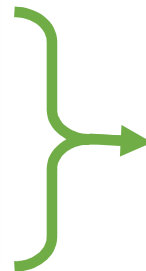
```
1 m = [n for n in range(10) if n%2 ==0]
2 print(m)
```

[0, 2, 4, 6, 8]

```
1 [ <expression_if> if <condition> else <expression_else> for <element> in <iterable> ]
```

```
1 m = []
2 for n in range(10):
3     if n%2 == 0:
4         m.append(n**2)
5     else:
6         m.append(n)
7 print(m)
```

[0, 1, 4, 3, 16, 5, 36, 7, 64, 9]



```
1 m = [n**2 if n%2 ==0 else n for n in range(10)]
2 print(m)
```

[0, 1, 4, 3, 16, 5, 36, 7, 64, 9]

Chapter3.

Split, Join, Enumerate, Zip

Chapter 3. Split, Join, Enumerate, Zip

- 1. Split
 - String Type을 특정 기준으로 나눠서 List 형태로 변환
 - default '공백'으로 처리
 - ex. String 데이터에서 특정 단어가 얼마나 자주 발생하였는가

```
1 words = 'one two three four five'
2
3 print(words.split(' '))
4 print(words.split())
```

```
['one', 'two', 'three', 'four', 'five']
['one', 'two', 'three', 'four', 'five']
```

```
1 print(type(words))
2 print(type(words.split()))
```

```
<class 'str'>
<class 'list'>
```

Chapter 3. Split, Join, Enumerate, Zip

- 2. Join
 - String List를 합쳐 하나의 String으로 변환

```
1 word_list = ['one', 'two', 'three', 'four', 'five']
2 print(' '.join(word_list))
3 print(type(word_list))
4 print(type(' '.join(word_list)))
```

one two three four five
<class 'list'>
<class 'str'>

Chapter 3. Split, Join, Enumerate, Zip

• 3. Enumerate

- List를 입력으로 받아 인덱스 값을 포함하는 enumerate 객체를 리턴
- iterator를 순회하면서 각 아이템의 인덱스를 얻어올 수 있음

```
1 num = 0
2 for v in word_list:
3     num += 1
4     print(num, v)
```

```
1 one
2 two
3 three
4 four
5 five
```

```
1 for i, v in enumerate(word_list):
2     print(i+1, v)
```

```
1 for i, v in enumerate(word_list, 1):
2     print(i, v)
```

```
1 one
2 two
3 three
4 four
5 five
```

Chapter 3. Split, Join, Enumerate, Zip

- 4. Zip
 - 두개의 list의 값을 병렬적으로 추출함

```
1 num_list=[1, 2, 3, 4, 5]
2 str_list=['one', 'two', 'three', 'four', 'five']
3
4 for n, s in zip(num_list, str_list):
5     print(n, s)
```

```
1 one
2 two
3 three
4 four
5 five
```

```
1 num_list=[1, 2, 3]
2 str_list=['one', 'two', 'three', 'four', 'five']
3
4 for n, s in zip(num_list, str_list):
5     print(n, s)
```

```
1 one
2 two
3 three
```

Chapter 4.

Lambda, Map, Reduce

Chapter 4. Lambda, Map, Reduce

- 1. function

- Example 1. 함수 선언

```

1 def hello_world():
2     print('Hello, World !')
3
4 hello_world()

```

Hello, World !

```

1 def hello_world():
2

```

File "<ipython-input-70-90f40b5bc5ac>", line 2

^

SyntaxError: unexpected EOF while parsing

SyntaxError: unexpected EOF while parsing

: 프로그램 구문이 잘못되었을 때 발생하는 예러

```

1 def hello_world():
2     pass

```

내용이 없는 빈 함수를 만드는 경우 **pass** 를 넣어
함수의 틀 유지

Chapter 4. Lambda, Map, Reduce

• 1. function

- *Example 2.* 인수가 여러 개인 함수 (인수 개수만큼 parameter 필요)

```

1 def average(x, y, z):
2     total = x + y + z
3     average = total/3
4     return average
5
6 avg = average(10, 20, 30)
7 print(avg)

```

20.0

- *Example 3.* 반환 값이 여러 개인 경우
 - 함수 반환 값이 2개 이상인 경우 튜플로 반환

```

1 def add_sub(x, y):
2     return x+y, x-y
3
4 a,b=add_sub(10, 20)
5 print(a, b)

```

30 -10

```

1 x = add_sub(10,20)
2 print(type(x))
3 print(x[0], x[1])

```

<class 'tuple'>
30 -10

Chapter 4. Lambda, Map, Reduce

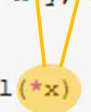
- 1. function
 - *Example 4. List unpacking*

```

1 def add_mul(x, y):
2     return x+y, x*y
3
4 x=[100, 3]
5 a,b=add_mul(*x)
6 print(a,b)

```

103 300



```

1 def add_mul(x, y):
2     return x+y, x*y
3
4 a,b=add_mul(*[100, 2, 30])
5 print(a,b)
6

```

함수 매개변수 개수와 리스트의 개수는 같아야 함

`TypeError: add_mul() takes 2 positional arguments but 3 were given`

Chapter 4. Lambda, Map, Reduce

• 1. function

- 함수의 default value를 선언
- 함수 호출 시, 매개변수를 넘겨주지 않는 경우 default value 값이 설정됨
- default value를 가진 매개변수가 먼저 올 수 없음

• Example 5. default Argument

```
1 def total(x, y, z=10):
2     return x+y+z
```

```
1 total(10, 20, 30)
```

60

```
1 total(10, 20)
```

40

```
1 def total(x=5, y, z):
2     return x+y+z
```

File "<ipython-input-32-5fdebf61356b>", line 1

```
def total(x=5, y, z):
    ^
```

SyntaxError: non-default argument follows default argument

Chapter 4. Lambda, Map, Reduce

• 1. function

• Example 5-1. Dictionary unpacking

```

1 p={'name':'John', 'age':30}
2 def person_info(name, age):
3     print("name : ", name)
4     print("age : ", age)
5
6 person_info(**p)

```

```

name : John
age : 30

```

함수의 매개변수와
dictionary의 Key 이름이 동일

*p : dictionary의 Key 사용

• Example 5-2. Dictionary unpacking

```

1 def person_info(**kwargs):
2     for kw, arg in kwargs.items():
3         print(kw, ": ", arg)
4
5 p={'name':'John', 'age':30}
6 person_info(**p)
7
8 print('-'*20)
9 p={'name':'Mike'}
10 person_info(**p)

```

```

name : John
age : 30
-----
name : Mike

```

정의되지 않은 키워드 처리 시,
dictionary의 Key, Value 값을 매개변수로 받음

Chapter 4. Lambda, Map, Reduce

• 2. Lambda

- 익명 함수, lambda expression
- 필요한 곳에서 정의하고 사용후에는 버림
- python3에서는 권장하지 않음
 - 그러나, 많이 사용 (Legacy library나 머신러닝 코드 등)

| [Define] | [lambda] |
|---|---|
| <pre>def add_10(x): return x+10</pre> | <pre>f = lambda x : x+10 f(1)</pre> |
| <pre>def add(x, y): return x+y</pre> | <pre>f = lambda x, y: x+y f(1,10)</pre> |

Chapter 4. Lambda, Map, Reduce

• 2. Lambda

- (lambda 매개변수 1, 매개변수 2: 반환 값) (a1, a2)
- 복잡한 코드인 경우 def 함수로 작성

```
1 def plus_one(x):  
2     return x+1
```

```
1 plus_one(1)
```

2

```
1 f=lambda x:x+1  
2 f(1)
```

2

```
1 (lambda x:x+1)(1)
```

2

Chapter 4. Lambda, Map, Reduce

- 3. Map

- list, str, tuple 등 자료형 각 element에 동일한 함수를 적용함

`Map` applies a function to all the items in an input_list. Here is the blueprint:

Blueprint

```
map(function_to_apply, list_of_inputs)
```

- *Example*


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

```
1 list(map(lambda x:x+1, data))
```

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


Chapter 4. Lambda, Map, Reduce

- 3. Map
 - *Example 1.*

| | | | | | |
|---------|--|---|---|----|----|
| items | 1 | 2 | 3 | 4 | 5 |
| |  lambda x: x**2 | | | | |
| squared | 1 | 4 | 9 | 16 | 25 |

```

1 for i in items:
2     squared.append(i ** 2)
3 print(squared)

```

[1, 4, 9, 16, 25]



```

1 squared = list(map(lambda x:x**2, items))
2 print(squared)

```

[1, 4, 9, 16, 25]

Chapter 4. Lambda, Map, Reduce

- 3. Map

- *Example 2*

- 두개 이상의 list에 적용

```
1 items = [1, 2, 3, 4, 5]
2 data  = [2, 2, 2, 3, 3]
3
4 list(map(lambda x,y:x+y, items, data))
```

[3, 4, 5, 7, 8]

- *Example 3*

- if 조건문 사용

```
1 items = [1, 2, 3, 4, 5]
2 list(map(lambda x:x**2 if x%2==0 else x, items))
```

[1, 4, 3, 16, 5]

Chapter 4. Lambda, Map, Reduce

- 4. Reduce

- list에 똑같은 함수를 적용하여 통합
- 하나의 결과 값을 가짐
- 코드의 직관성이 떨어져 Python3에서는 권장하지 않음
- *from functools import reduce*

- *Example*

```
1 from functools import reduce
2
3 items=[1, 2, 3, 4, 5]
4 reduce(lambda x, y: x+y, items)
```

Chapter 4. Lambda, Map, Reduce

- 5. Filter

- 필요 없는 데이터를 필터링하기 위한 함수
- 논리함수를 통해 나온 True/False에 따라 필터링을 수행

- **Example**

```
1 items = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2 list(filter(lambda x:x<6, items))
```

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

Chapter 5. Class

Chapter 5. Class

- 클래스는 객체(Object)를 만들기 위한 도구
- 클래스의 구성

| 구성 | Description |
|----------|---|
| 속성 | 객체를 구성하는 데이터 |
| 메소드 | 클래스 내에서 어떤 기능을 수행하는 함수 |
| 생성자, 소멸자 | 객체 생성 또는 소멸 시 자동으로 호출되는 함수 - 생성자 <code>def __init__(self, ...)</code> - 소멸자 <code>def __del__(self, ...)</code> |
| 연산자 중복 | 연산자(+, - 등) 기호를 이용하여 표현 가능 |

Chapter 5. Class

- OOP (Object Oriented Programming)
- Class Name은 CapWords 방식으로 명명 (ex. MyClass)

```

1 class ClassName:
2
3     variable_1 = 0
4     variable_2 = 0
5
6     def method_1(self):
7         pass
8
9     def method_2(self):
10        pass
  
```

class

```

1 c1 = ClassName()
2 c2 = ClassName()
3 c3 = ClassName()
  
```

Object

```

1 class ClassName
2     def method_first(self):
3         pass
  
```

File "<ipython-input-4-12de0c2c5d5a>", line 1
 class ClassName
 ^
 SyntaxError: invalid syntax

Chapter 5. Class

• 1) 멤버 변수, 멤버 메소드

| 구성 | 종류 |
|-----|----------------------------|
| 변수 | 클래스 변수(Class Variable) |
| | 인스턴스 변수(Instance Variable) |
| 메소드 | 클래스 메소드(Class Method) |
| | 인스턴스 메소드(Instance Method) |

```

1  class Class:
2
3      class_variable = 0
4
5      @classmethod
6      def class_method(cls):
7          print("Class method")
8
9      def insatnce_method(self):
10         self.instance_variable = 0
11         print("Instance method")

```

```

1  cs = Class()
2  Class.class_method()
3  cs.insatnce_method()

```

```

Class method
Instance method

```


Chapter 5. Class

• 1-1) variable

- 클래스 변수 (class variable)
 - 공용으로 사용되는 변수
- 인스턴스 변수 (instance variable)
 - 각 객체(instance) 마다 별도로 사용되는 변수

```

1 class Bag:
2
3     user_bag = [] 클래스 변수
4
5     def put_bag(self, stuff):
6         self.user_bag.append(stuff)

```

```

1 kim = Bag()
2 lee = Bag()
3
4 kim.put_bag("pencil")
5 lee.put_bag("Note")
6 print(Bag.user_bag)

```

['pencil', 'Note']

```

1 class Bag:
2
3     def __init__(self):
4         self.user_bag = [] 인스턴스 변수
5
6     def put_bag(self, stuff):
7         self.user_bag.append(stuff)

```

```

1 kim = Bag()
2 lee = Bag()
3
4 kim.put_bag("pencil")
5 lee.put_bag("Note")
6 print(kim.user_bag, lee.user_bag)

```

['pencil'] ['Note']

Chapter 5. Class

• 1-1) variable

• Example

```

1 class Account:
2     num_account = 0
3
4     def __init__(self, name):
5         self.name = name
6         Account.num_account += 1

```

클래스 변수

인스턴스 변수

```

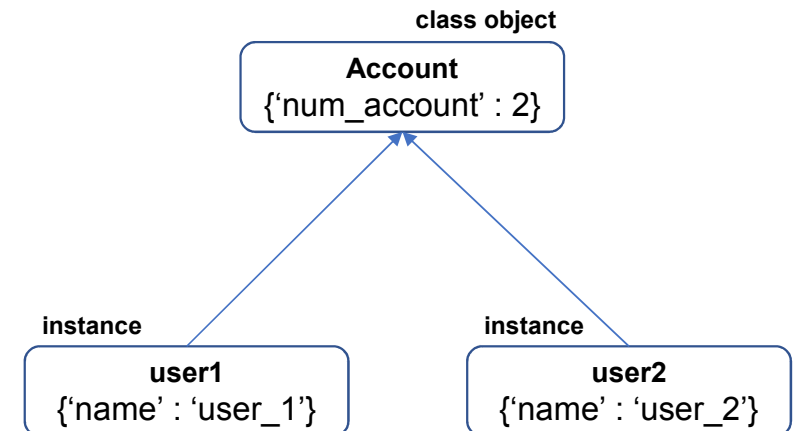
1 user1 = Account("user_1")
2 user2 = Account("user_2")
3
4 print(user1.name, user1.num_account)
5 print(user2.name, user2.num_account)
6
7 print(user1.__dict__)
8 print(user2.__dict__)

```

```

user_1 2
user_2 2
{'name': 'user_1'}
{'name': 'user_2'}

```



Chapter 5. Class

- 1-2) method
 - 클래스 메소드 (class method)

```
1 class Class:
2
3     @classmethod
4     def class_method(cls):
5         pass
```

- 인스턴스 메소드 (instance method)

```
1 class Class:
2
3     def instance_method(self):
4         pass
```

Chapter 5. Class

• 1-2) method

• `__init__`, `__del__`

- 생성자 : 객체 생성 시 자동으로 호출되는 method
- 소멸자 : 객체 소멸 시 자동으로 호출되는 method
- 인스턴스 메소드 : 첫번째 매개 변수는 생성되는 객체를 가리킴 (self)

매직 메소드: https://ziwon.github.io/posts/python_magic_methods/
<https://corikachu.github.io/articles/python/python-magic-method>

Example.

```

1 class ClassName:
2
3     def __init__(self, v1, v2):
4         self.v1 = v1
5         self.v2 = v2

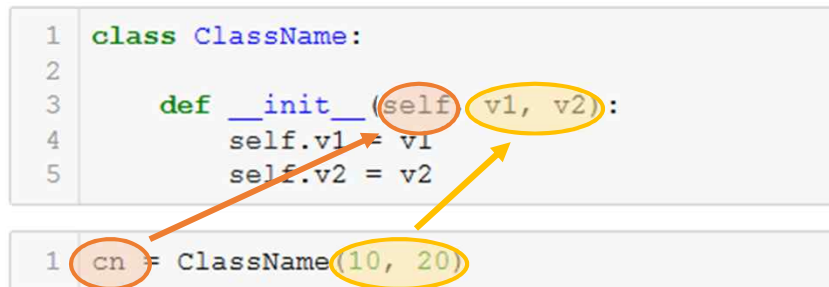
```



```

1 cn = ClassName(10, 20)

```



```

1 class ClassName:
2
3     def __init__(v1, v2):
4         print(v1, v2)
5

```

```

1 cn = ClassName(10, 20)

```

```

TypeError                                 Traceback (most recent call last)
<ipython-input-24-94bfcd20e614> in <module>()
----> 1 cn = ClassName(10, 20)

TypeError: __init__() takes 2 positional arguments but 3 were given

```

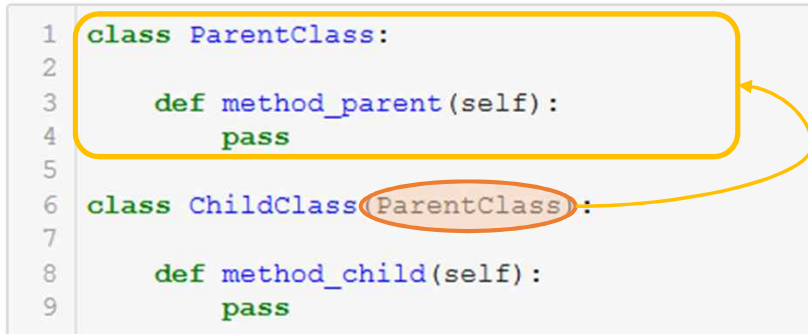
Chapter 5. Class

• 2) 클래스 상속 (Inheritance)

- 부모의 모든 것을 물려받는 것
- 기존 클래스를 변경하지 않고 기능을 추가/변경을 위해 사용
 - 기존 클래스가 라이브러리 형태로 제공되거나 수정이 허용되지 않는 경우

• Example

```
1 class ParentClass:
2
3     def method_parent(self):
4         pass
5
6 class ChildClass(ParentClass):
7
8     def method_child(self):
9         pass
```



ChildClass는 ParentClass를 상속받음

따라서, ChildClass는 ParentClass의 속성과 메소드를 모두 사용할 수 있음

Chapter 5. Class

• 2-1) 오버라이딩 (Overriding)

- 부모 클래스로부터 상속받은 특성을 자식 클래스에서 재정의
 - 1) 어떤 기능이 같은 메서드 이름을 계속 사용되어야 할 때
 - 2) 원래 기능을 유지하면서 새로운 기능을 덧붙일 때

```

1 class Person:
2
3     def greeting(self):
4         print("Hello")
5
6 class Employee(Person):
7
8     def greeting(self):
9         print("안녕하세요")

```

```

1 kim = Employee()
2 kim.greeting()

```

안녕하세요

```

1 class Person:
2
3     def greeting(self):
4         print("Hello")
5
6 class Employee(Person):
7
8     def greeting(self):
9         super().greeting()
10        print("Nice to meet you")

```

부모 클래스의 기능 유지
새로운 기능 추가

```

1 kim = Employee()
2 kim.greeting()

```

Hello
Nice to meet you

Chapter 6. 모듈과 패키지

Chapter 6. 모듈과 패키지

- 1. 모듈 (module)
 - 함수나 변수, 클래스 등을 가진 파일 (.py)
 - 모듈 안에는 함수, 클래스 또는 변수들이 정의
 - 파이썬은 많은 표준 라이브러리 모듈을 제공함
- 모듈 import
 - 1) **import** 모듈명
 - “모듈.함수명”
 - 2) **from** 모듈명 **import** 클래스/함수/변수
 - “함수명”

```
1 import math
2
3 n = math.factorial(5)
4 n
```

120

```
1 from math import factorial
2
3 n = factorial(5)
4 n
```

120

```
1 from math import factorial, pi
2
3 n = factorial(1) + pi
4 n
```

4.141592653589793

```
1 from math import *
2
3 n = cos(pi)
4 n
```

-1.0

Chapter 6. 모듈과 패키지

• 2. Packages

- 모듈을 계층적으로 관리하기 위한 모듈의 상위개념
- 공동 작업이나 코드의 유지 보수 등에서도 유리
 - 다른 모듈과 이름이 겹치더라도 해당 패키지 이름이 다르다면 문제가 없음

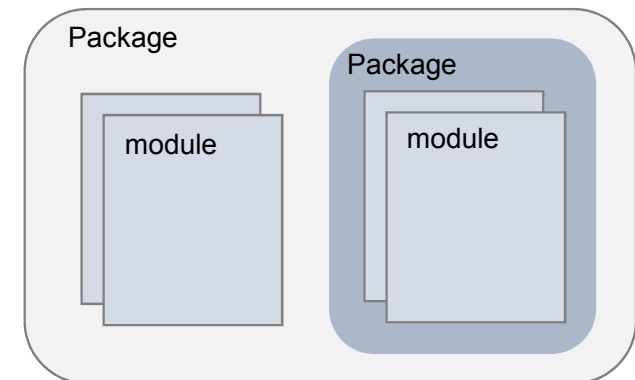
• 패키지의 모듈 import

```

1 import Package.module
2 import Package.module.변수
3 import Package.module.함수
4 import Package.module.클래스
  
```

```

1 from Package.module import 변수
2 from Package.module import 함수
3 from Package.module import 클래스
  
```



Chapter 7. 파일 읽고/쓰기

chapter 7. 파일 읽고 쓰기

- 1. 파일 열기
 - open (FILE_PATH, mode)

| mode | Description |
|------|---|
| 'r' | 파일을 읽기만 할 때 사용 해당 경로에 파일이 없는 경우 에러 |
| 'w' | 파일에 내용을 쓸 때 사용, 기존 내용은 삭제되고 처음부터 기록 해당 경로에 파일이 없는 경우 파일 생성 |
| 'a' | 파일 마지막에 내용을 추가할 때 사용 해당 경로에 파일이 없는 경우 파일 생성 |

- Example*

```
1 f=open('test_write.txt', 'w')
2 f.write("Test Write\n")
3 f.close()
```

```
1 f=open('test_write.txt', 'a')
2 f.write("Test Write_2\n")
3 f.close()
```

```
1 f=open('test_write.txt', 'r')
2 print(f.read())
3 f.close()
```

```
Test Write
Test Write_2
```

chapter 7. 파일 읽고 쓰기

• 2. 파일 읽기

| function | Description |
|---------------|---------------------------------|
| f.read() | 파일의 내용 전체를 한번에 읽음 |
| f.readline() | 호출 시 파일의 내용을 line 단위로 읽음 |
| f.readlines() | 파일의 모든 내용을 line 단위로 읽어 리스트로 반환함 |

• Example

```
1 f = open('test_file.txt' , 'r')
2 f.read()
```

```
'Hello World\nNice to meet you ! \n'
```

```
1 f = open('test_file.txt' , 'r')
2 f.readline().strip()
```

```
'Hello World'
```

chapter 7. 파일 읽고 쓰기

• 2. 파일 읽기

```
1 f = open('test_file.txt' , 'r')
2 for row in f:
3     print(row.strip())
4 f.close()
```

Hello World
Nice to meet you !

```
1 with open('test_file.txt' , 'r') as f:
2     for row in f:
3         print(row.strip())
4
```

Hello World
Nice to meet you !

• glob

- 특정 패턴과 일치하는 모든 경로명을 찾을 때 사용

```
1 import glob, os
2 for f in glob.glob(os.path.join('./', '*.txt')):
3     print(f)
```

.\test_file.txt
.\test_write.txt

*) os.path.join
파일명과 폴더의 이름을 합침

chapter 7. 파일 읽고 쓰기

- os.path

| function | description |
|----------------------|--------------------------|
| os.mkdir(DIR_PATH) | directory 생성 |
| os.rmdir(DIR_PATH) | directory 삭제 |
| os.path.exists(PATH) | directory 또는 File 존재 여부 |
| os.path.isfile(PATH) | 해당 경로에 존재하는 File 여부 |
| os.path.isdir(PATH) | 해당 경로에 존재하는 directory 여부 |

chapter 7. 파일 읽고 쓰기

• 3. 파일 쓰기

| mode | Description |
|------------------------|---------------------|
| f.write(STR) | 문자열 입력 |
| f.writelines(STR_LIST) | 리스트 등으로 된 여러 문장을 입력 |

• Example

```
1 f=open('test_write.txt', 'w')
2 f.write("Test Write")
3 f.close()
```

```
1 f=open('test_write.txt', 'r')
2 f.read()
```

'Test Write'

```
1 test_list=[]
2 for i in range(3):
3     test_list.append("Test Write {}".format(i))
4
5 f=open('test_write.txt', 'w')
6 f.writelines(test_list)
7 f.close()
```

```
1 f=open('test_write.txt', 'r')
2 f.read().strip().split('\n')
```

['Test Write 0', 'Test Write 1', 'Test Write 2']

chapter 7. 파일 읽고 쓰기

- 4. pickle
 - List, Class와 같은 자료형은 파일 입출력 방법으로 데이터를 저장할 수 없음

```
1 list = ['a', 'b', 'c']
2 with open('list_example.txt', 'wb') as f:
3     f.write(list)
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-97-8b1ff345e7e3> in <module>()
      1 list = ['a', 'b', 'c']
      2 with open('list_example.txt', 'wb') as f:
----> 3     f.write(list)

TypeError: a bytes-like object is required, not 'list'
```

- 텍스트 이외의 자료형을 저장하기 위한 모듈
- 데이터를 자료형 변경없이 파일 그대로 저장/로드할 수 있음
 - pickle.dump, pickle.load

chapter 7. 파일 읽고 쓰기

- 4. pickle
 - 1) module import
 - 2) dump : 데이터를 자료형 변경없이 파일로 저장
 - 3) load : 저장했던 데이터를 그대로 로드
- *Example*

```
1 import pickle
2
3 test_list = ['a', 'b', 'c']
4 with open('list_example.bin', 'wb') as f:
5     pickle.dump(test_list, f)
6
7 with open('list_example.bin', 'rb') as f:
8     results = pickle.load(f)
9
10 print(results)
```

['a', 'b', 'c']

Appendix

- 강의자료
 - edwith (<https://www.edwith.org/>)
 - 생활코딩 (<https://www.opentutorials.org/course/1750/10116>)
- 참고자료
 - <https://wikidocs.net/book/1>
 - <https://dojang.io/mod/page/view.php?id=1060>
 - <http://book.pythontips.com/en/latest/index.html>