



# Python Programming

# 파이썬 기본 프로그래밍

1. 파이썬 프로그래밍 소개
2. 개발환경
3. 기본 데이터 타입
4. 컨테이너 데이터 타입
5. 연산자
6. 제어문
7. 함수

- 파이썬(Python)은 널리 쓰이는 범용, **고급언어**이다.
- 네덜란드의 **귀도 반 로섬(Guido van Rossum)**이 개발한 프로그래밍 언어
- 귀도는 1989년 크리스마스가 있던 주에 자신이 출근하던 연구실의 문이 닫혀 있어서 취미삼아 파이썬을 만들었다
- 파이썬이라는 이름은 귀도가 즐겨 봤던 코미디 프로그램인 몬티 파이썬의 날아다니는 써커스 <Monty Python's Flying Circus>에서 인용
- 파이썬의 설계 철학은 **코드 가독성**에 중점을 두고 있으며 파이썬의 문법은 프로그래머가 (C와 같은 언어에서 표현 가능한 것보다도) **더 적은 코드로도 자신의 생각을 표현**하도록 한다.
- 파이썬은 프로그램의 크기에 상관없이 **명확**하게 프로그램 할수 있는 구성 요소들을 제공한다.

- 위키피디아.

# 파이썬 프로그래밍 소개

## ➤ 인터프리터 방식의 언어

- vs 컴파일 방식
- 가상머신으로 실행 (PVM : Python Virtual Machine)
- 파일명.py는 실행시 바이트 코드 파일로 변환 (플랫폼 독립적)

## ➤ 코드의 간결성

- 개발 생산성 향상과 쉬운 유지보수
- TIME TO MARKET

## ➤ 객체지향을 지원

- 코드의 재사용성
- 유지보수의 용이성
















➤ 파이썬은 고급 프로그래밍 언어이다.

➤ 다양한 형식의 라이브러리가 많이 공유되고 있음-pip

➤ 개발에 비용이 발생하지 않는 언어(무료)

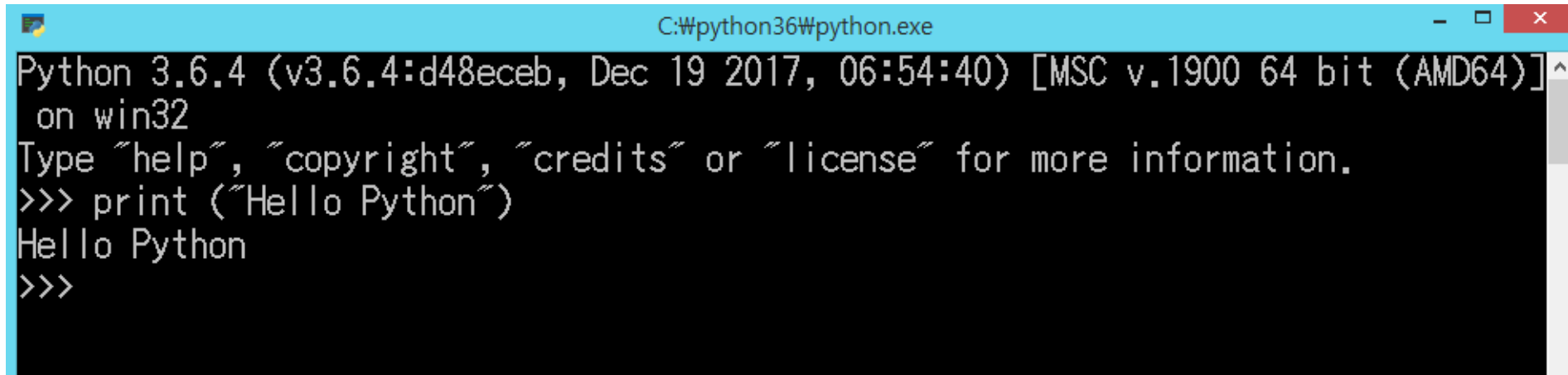
# 파이썬 개발환경

PC > 로컬 디스크 (C:) > python36

이름	수정한 날짜	유형	크기
 DLLs	2018-01-14 오전...	파일 폴더	
 Doc	2018-01-14 오전...	파일 폴더	
 include	2018-01-14 오전...	파일 폴더	
 Lib	2018-01-14 오전...	파일 폴더	
 libs	2018-01-14 오전...	파일 폴더	
 Scripts	2018-01-14 오전...	파일 폴더	
 tcl	2018-01-14 오전...	파일 폴더	
 Tools	2018-01-14 오전...	파일 폴더	
 LICENSE	2017-12-19 오전...	텍스트 문서	30KB
 NEWS	2017-12-19 오전...	텍스트 문서	371KB
 python	2017-12-19 오전...	응용 프로그램	99KB
 python3.dll	2017-12-19 오전...	응용 프로그램 확장	58KB
 python36.dll	2017-12-19 오전...	응용 프로그램 확장	3,527KB
 pythonw	2017-12-19 오전...	응용 프로그램	97KB
 vcruntime140.dll	2016-06-09 오후...	응용 프로그램 확장	86KB

# 파이썬 개발환경

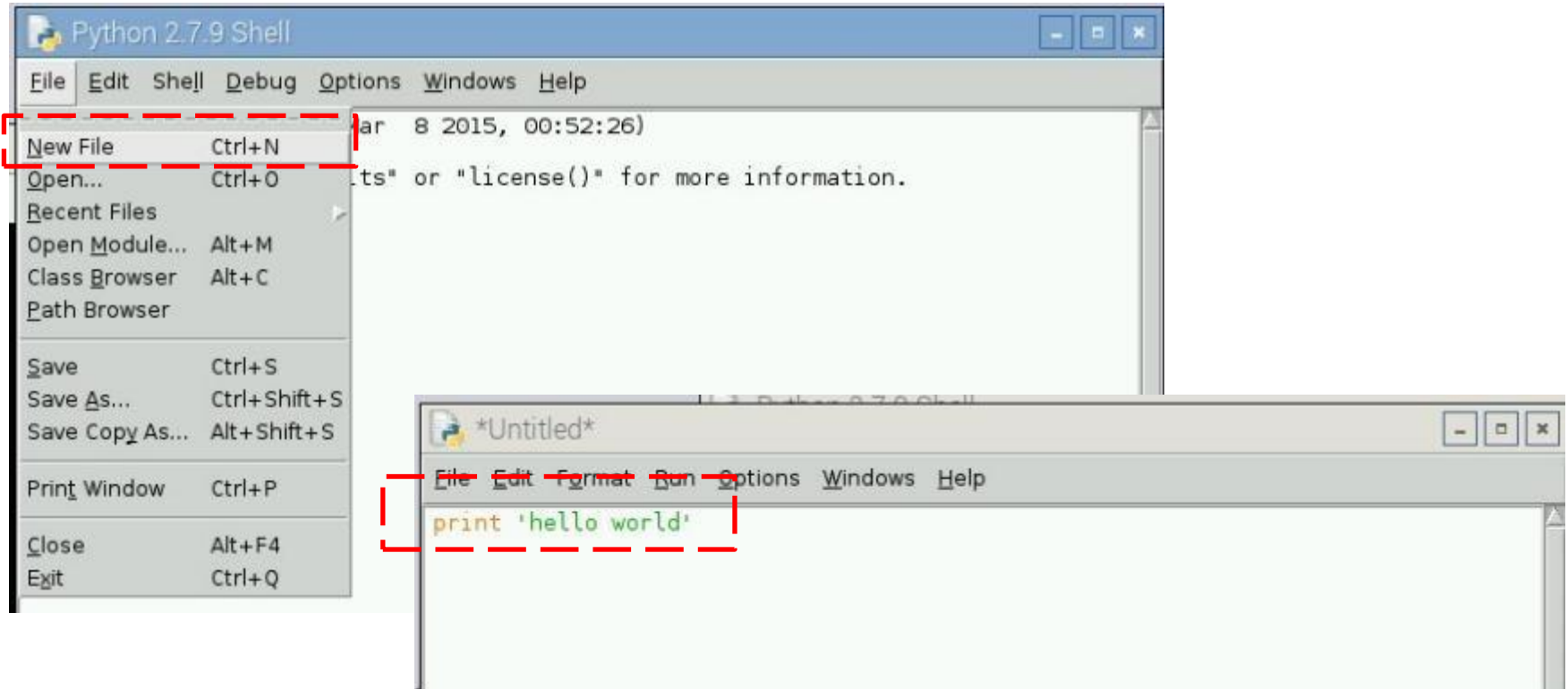
## ➤ 파이썬 셸에서 실행



```
C:\python36\python.exe
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello Python")
Hello Python
>>>
```

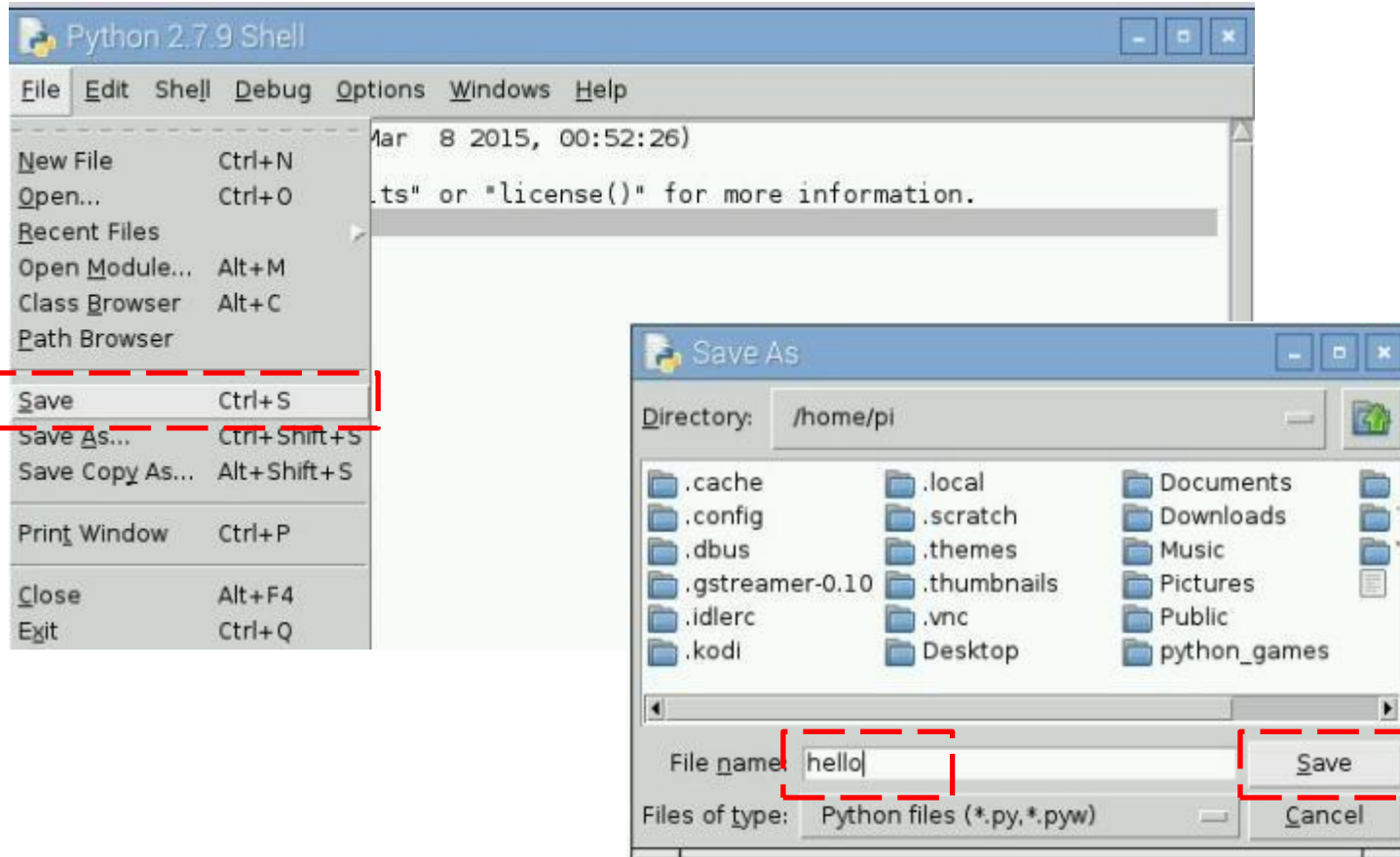
# 파이썬 개발 환경

- 별도의 파일에 프로그램 작성



# 파이썬 개발환경

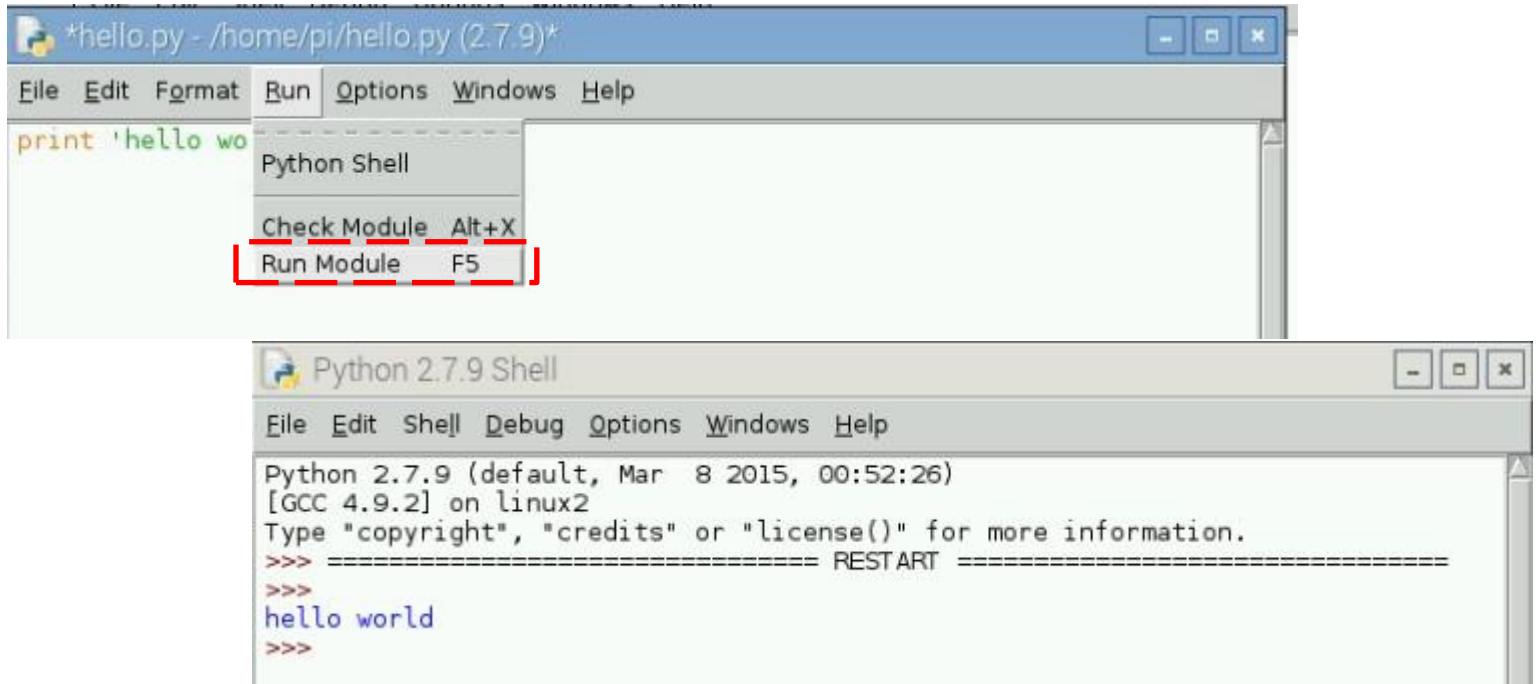
## ➤ 별도의 파일에 프로그램 작성





# 파이썬 개발환경

## ➤ 실행방법 1:파이썬 셸에서 실행



## 기본 데이터 타입

1. 표준 출력
2. 정수(int)
3. 실수(float) 타입
4. 불린(boolean) 타입
5. None 타입
6. 문자열(str) 타입
7. 리스트(list) 타입
8. 튜플(tuple) 타입
9. 사전(dict) 타입
10. 집합(set) 타입
11. 형 변환

# 표준 출력

## 코드

```
print(10 + 5)  
print(10 - 5)  
print(10 * 5)  
print(10 / 5)
```

## 실행 결과

15

5

50

2.0

# 기본 데이터 타입

## C와 동일 타입

- 정수(int)
- 실수(float)
- 불린(boolean)
- 문자열(str)

```
>>> type(1)
<type 'int'>
>>> type(3.14)
<type 'float'>
>>> type(True)
<type 'bool'>
>>> type('hello world')
<type 'str'>
```

## 파이썬 유일 타입

- 리스트(list)
- 튜플(tuple)
- 사궤(dict)
- 집합(Set)

```
>>> type([1,2,3])
<type 'list'>
>>> type((1,2,3))
<type 'tuple'>
>>> type({'name': '홍길동', 'age': 20})
<type 'dict'>
>>> type({1,2,3,4})
<type 'set'>
```

# 기본 데이터 타입 : 정수(int)

- 프로그램에서 기본적으로 사용하는 정수의 표현이 가능
- 표현 가능 범위 == 메모리 용량
- 정수의 진법 표현
  - 2진법: 0b로 시작하고 0과 1로 수를 표현한다.
  - 0b1010, 0b1110111, 0b10101010 등
  - 8진법: 0o로 시작하고 0, 1, 2, 3, 4, 5, 6, 7로 수를 표현한다.
  - 01234567
  - 16진법: 0x로 시작하고 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f로 수를 표현한다.
  - (대문자 A, B, C, D, E, F로 표현해도 된다.)
  - 0xff, 0x1, 0x0f

# 기본 데이터 타입 : 실수(float) 타입

- 소수점을 포함하고 있는 수

- `>>> 3.14`
- `3.14`
- `>>> 2 * 3.14 * 5`
- `31.400000000000002`

- 가수e지수 혹은 가수E지수

- `>>> 3.1415e2`
- `314.15`
- `>>> 2.7182E2`
- `271.82`
- `>>> 0.3e-2`
- `0.003`

# 기본 데이터 타입 : 불린(boolean) 타입과 None 타입

## ■ 불린 타입

- True와 False 두개의 불린(boolean)형 데이터를 제공.
  - True는 참을 표현
  - False는 거짓을 표현
- ```
>>> True True
>>> True and True True
>>> True and False False
```

## ■ None 타입

- None 타입의 예는 None 데이터 하나뿐이다.
- None은 값이 없음을 표현하는 데이터이다.
- >>> x = None

# 기본 데이터 타입 : 문자열(str) 타입

- 더블 쿼터 문자들을 감싸서 문자열을 표현 한다.

```
>>> "Hello World"
```

```
Hello World
```

```
>>> "한글 입력"
```

```
"한글 입력"
```

```
>>> "3.14"
```

```
3.14
```

- 문자열 내에 인용부호 포함시키기

- 만약 단일 쿼터가 포함된 문자열의 경우는 더블 쿼터(")로 감싼다.
- 만약 더블 쿼터가 포함된 문자열의 경우는 단일 쿼터(')로 감싼다.

```
>>> "Heshouted "Help me!"
```

```
Heshouted "Help me!"
```

```
>>> "He doesn'tlike it." "He doesn'tlike it."
```



# 기본 데이터 타입 : 문자열(str) 타입

## ■ 여러 줄의 문자열 표현하기

- 싱글 쿼터('')나 더블쿼터(“ ”) 세 개를 사용하여 문자열로 사용

```
>>> """Hello World
```

```
...Hello Korea
```

```
...안녕 대한민국"""
```

```
"HelloWorld\nHello Korea\n안녕 대한민국"
```

.py 파일을 UTF-8이 아닌 CP949로 저장해서 그렇습니다. 이때는 스크립트 파일을 UTF-8로 저장하면 됩니다. 보통 메모장을 사용하면 기본 인코딩이 CP949라서 이런 문제가 종종 생깁니다. 메모장에서 UTF-8로 저장하려면 **파일(F) > 다른 이름으로 저장(A)... > 인코딩(E)**에서 **UTF-8**을 선택한 뒤 저장하면 됩니다. 이런 문제를 예방하려면 파이썬 IDLE, PyCharm 등 파이썬 전용 편집기나 개발 도구를 사용하면 됩니다. 이들 편집기, 개발 도구는 기본 인코딩이 UTF-8

# 기본 데이터 타입 : 문자열(str) 타입

## ■ 문자열 내 이스케이프 문자 사용하기

- 이스케이프 문자는 역 슬러시 문자(\)를 이용해서 표현한 특수문자이다.
- 이는 출력물을 보기 좋게 정렬하거나 그 외의 특별한 용도로 자주 이용된다.

| 이스케이프 문자 | 설명         |
|----------|------------|
| \n       | 줄바꿈(개행)    |
| \t       | 수평 탭       |
| \a       | 비프음 (뽕 소리) |
| \000     | 널문자        |
| \\       | “\” 문자     |
| \        | 단일 인용부호(,) |
| \”       | 이중 인용부호(“) |

# 기본 데이터 타입 : 문자열(str) 타입

## ■ 문자열 format() 함수 활용

- 문자열 내의 { } 사이의 인자값을 출력
- 다양한 양식을 포맷을 지정하여 출력 가능.

```
age = 20
```

```
name = 'Swaroop'
```

```
print '{0} was {1} years old when he wrote this book'.format(name, age)
print 'Why is {0} playing with that python?'.format(name)
```

### # {} 안의 숫자는 생략 가능

```
print '{} was {} years old when he wrote this book'.format(name, age)
print 'Why is {} playing with that python?'.format(name)
```

### # 소수점 이하 셋째 자리까지 부동 소숫점 숫자 표기 (0.333)

```
print '{0:.3f}'.format(1.0/3)
```

### # 밑줄(\_)로 11칸을 채우고 가운데 정렬(^)하기 (\_\_hello\_\_)

```
print '{0:_^11}'.format('hello')
```

### # 사용자 지정 키워드를 이용해 (kim wrote little Python) 표기

```
print '{name} wrote {book}'.format(name='kim', book='little f Python')
```

# 기본 데이터 타입 : 리스트(list) 타입

- 다양한 데이터 타입들을 순서에 따라 저장할 수 있는 데이터 타입이다.
- 리스트 데이터를 만드는 방법은 대괄호[ ] 안에 데이터들을 넣어 주면 된다.

```
>>> [1, 3.14, True, "문자열"]
```

```
[1, 3.14, True, "문자열"]
```

```
>>> [1, [2], [False, True]]
```

```
[1, [2], [False, True]]
```

# 기본 데이터 타입 : 리스트(list) 타입

- 리스트 타입은 순서화 된 데이터 타입
- 0으로부터 시작하는 인덱스 값으로 각 요소들을 참조할 수 있다.

```
>>> [1, 3.14, True, 'str'][0]
```

```
1
```

```
>>> [1, 3.14, True, 'str'][2]
```

```
True
```

```
>>> len([1, 3.14, True, 'str'])
```

```
4
```

```
>>> len([1, 3, 5])
```

```
3
```

- 리스트 타입의 갯수를 구하기 위해서는 len 함수를 이용하면 된다.

# 기본 데이터 타입 : 리스트(list) 타입

- 리스트 인덱스의 시작이 1이 아니라 0
- 따라서 리스트의 마지막 인덱스는 “리스트의\_크기 - 1”이 된다.

|      |      |      |      |                  |          |
|------|------|------|------|------------------|----------|
| 1    | 2    | 3.14 | True | “Hello<br>World” | “안녕 파이썬” |
| [0]  | [1]  | [2]  | [3]  | [4]              | [5]      |
| [-6] | [-5] | [-4] | [-3] | [-2]             | [-1]     |

- 리스트 인덱스는 마지막 요소부터 접근이 가능하다
- **리스트의 마지막 요소에 대한 인덱스는 “-1”이다.**
- 역순으로 내려오면서 인덱스 값이 줄어든다.

# 기본 데이터 타입 : 리스트(list) 타입

```
>>> [1, 2, 3.14, True, "Hello World", "안녕 파이썬"][-1]
```

```
„안녕 파이썬“
```

```
>>> [1, 2, 3.14, True, "Hello World", "안녕 파이썬"][-2]
```

```
„Hello World“
```

```
>>> [1, 2, 3.14, True, "Hello World", "안녕 파이썬"][-3]
```

```
True
```

```
>>> [1, 2, 3.14, True, "Hello World", "안녕 파이썬"][-6]
```

```
1
```

# 기본 데이터 타입 : 튜플(tuple) 타입

- 다양한 데이터 타입들을 주어진 순서에 따라 저장할 수 있는 데이터 타입이다.
- 튜플 데이터 타입을 만드는 방법은 괄호 ()안에 데이터를 넣어 주면 된다.
- 리스트와 튜플 타입의 차이점
  - **리스트 타입은 내용의 변경이 가능 (mutable하다라고 한다)**
  - **튜플의 경우 내용의 변경이 불가 (immutable하다라고 한다).**
- **속도 면에서 튜플이 좀 더 빠르다.**
- 튜플의 인덱싱도 리스트의 인덱싱과 같은 방식이다.
  - 즉 인덱스 값은 0부터 시작하며 마지막 인덱스 값은 “튜플의\_크기 -1”이다.
  - 역순으로의 인덱싱도 가능하다.
  - 즉 마지막 요소에 대해 인덱스 값을 “-1”로 해서 접근해도 된다.
- 튜플의 갯수를 구하기 위해 리스트와 같이 **len 함수**를 쓰면 된다.



# 기본 데이터 타입 : 튜플(tuple) 타입

```
>>> (1, 2, 3.14, True, "Hello World", "안녕 파이썬")
```

```
(1, 2, 3.14, True, "Hello World", "안녕 파이썬")
```

```
>>> (1, 2, (1, 2), ("Hello World", True))
```

```
(1, 2, (1, 2), ("Hello World", True))
```

```
>>> x = [1, 2, 3.14, True, "Hello World", "안녕 파이썬 "]
```

```
>>> x[2] = 3
```

```
>>> print(x)
```

```
[1, 2, 3, True, "Hello World", "안녕 파이썬"]
```

```
>>> y = (1, 2, 3.14, True, "Hello World", "안녕 파이썬 ")
```

```
>>> y[2] = 3    # 튜플은 읽기전용이라 변경 불가
```

```
Traceback (most recent call last): File "<stdi
```

```
n>", line 1, in <module>
```

```
TypeError: 'tuple' object does not support item assignment
```

# 기본 데이터 타입 : 튜플(tuple) 타입

## ■ 팩킹(packing)

- 콤마(,)로 분리된 데이터 값들은 튜플로 인식한다.

```
>>> 1, 2, 3
```

```
(1, 2, 3)
```

```
>>> 1, "Hello", True, "안녕"
```

```
(1, "Hello", True, "안녕")
```

## ■ 언팩킹(unpacking)

- 변수에 한번에 저장 가능, tmp 변수 없이 값 교환 가능

```
>>> x, y = (1, 2)
```

```
>>> y, x = x, y # 교환 알고리즘 구현 가능
```

```
>>> x
```

```
>>> y
```

```
1
```

# 기본 데이터 타입 : 사전(dict) 타입 -1

- 사전과 비슷한 형태의 자료형
- 순서가 없는 키-값의 쌍으로 된 집합이다.
- 키에 의해 데이터 값에 접근할 수 있다.
- 키-값의 쌍을 중괄호{ }로 묶어주면 된다.
- 빈 딕셔너리를 만들 때는 {}만 지정하거나 dict를 사용

- **x = {}**

- **y = dict()**

- {키: 값, ...}

```
>>> d = {'name':'홍길동', 'age':20, 'addr':'seoul'}
```

```
>>> d
```

```
{'addr': 'seoul', 'age': 20, 'name': '홍길동'}
```

```
>>> d['name'], 또는 d.get('name', 'default value')
```

```
'홍길동'
```

```
>>> d['addr']
```

```
'seoul'
```

```
>>> d['age']
```

```
20
```

# 기본 데이터 타입 : 사전(dict) 타입 - 2

- dict에서 zip 함수를 이용하여 리스트 두 개를 딕셔너리로 만드는 방법(많이 활용)
  - **x2 = dict(zip(['a', 'b', 'c'], [10, 20, 30]))**
  - **>>> x2** {'a': 10, 'b': 20, 'c': 30}
- 리스트 안에 (키, 값) 형식의 튜플을 나열하는 방법으로 dic 생성
  - **x3 = dict([('a', 10), ('b', 20), ('c', 30)])** # (키, 값) 형식의 튜플로 딕셔너리를 만들
  - **>>> print(x3)**
  - {'a': 10, 'b': 20, 'c': 30}
- dict.fromkeys함수로기본 dic 생성
  - **keys = ['a', 'b', 'c', 'd']**
  - **x = dict.fromkeys(keys)**
  - **print(x)**
  - {'a': None, 'b': None, 'c': None, 'd': None}
  - **y = dict.fromkeys(keys, 100)**
  - **print(y)**
  - {'a': 100, 'b': 100, 'c': 100, 'd': 100}

# 기본 데이터 타입 : 사전(dict) 타입 -3

- **update(키=값)**은 이름 그대로 딕셔너리에서 키의 값을 수정 및 추가

```
a = {}  
a.update(name="kim")  
print(a)  
a.update(name="lee")  
print(a)
```

```
{'name': 'kim'}  
{'name': 'lee'}
```

- **pop(키)** 또는 **pop(키, 기본값)**은 특정 키-값 쌍을 삭제한 뒤 값을 반환

```
x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}  
print(x.pop('a')) # del x['a']  
print(x)  
print(x.pop('z', 0))
```

```
10  
{'b': 20, 'c': 30, 'd': 40}  
0
```

- **setdefault(키)** 또는 **setdefault(키, 기본값)**은 딕셔너리에 키와 값을 추가( 단, setdefault로 딕셔너리에 이미 들어있는 키의 값은 수정할 수 없음)

```
x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}  
x.setdefault('e')  
x.setdefault('f', 100)  
print(x)  
x.setdefault('f', 999)  
print(x)
```

```
{'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': None, 'f': 100}  
{'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': None, 'f': 100}
```

# 기본 데이터 타입 : 사전(dict) 타입 - 4

- keys()는 딕셔너리 a의 Key만을 모아서 dict\_keys라는 객체를 리턴

```
a = {'name': 'park', 'phone': '01012345678', 'birth': '0104'}  
b = a.keys()  
print(b)  
print(type(b))  
c = list(b)  
print(type(c))
```

```
dict_keys(['name', 'phone', 'birth'])  
<class 'dict_keys'>  
<class 'list'>
```

- values()는 딕셔너리 a의 values만을 모아서 dict\_keys라는 객체를 리턴

```
a = {'name': 'park', 'phone': '01012345678', 'birth': '0104'}  
b = a.values()  
print(b)  
print(type(b))
```

```
dict_values(['park', '01012345678',  
            '0104'])  
<class 'dict_values'>
```

- items()는 딕셔너리 키-값 쌍을 모두 가져온다
- clear() 함수는 모든 항목을 삭제

```
a.clear()
```

- 해당 Key가 딕셔너리 안에 있는지 판단하는 경우 in 연산자 활용

```
a = {'name': 'park', 'phone': '01012345678', 'birth': '0104'}  
print("name" in a)  
print("sex" in a)
```

```
True  
False
```

# 기본 데이터 타입 : 집합(set) 타입

- 집합은 순서화 되지 않게 유일한 값을 담는 데이터 타입이다.
- 집합 타입 데이터의 생성은 값들을 중괄호 {}로 묶어주면 된다.
- {값, ...}

```
>>> x = {2, 3, 2, 3, 4, 5, 1}
```

```
>>> x
```

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

```
>>> y = {3, 4, 5, 6, 7}
```

```
>>> x | y
```

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

```
>>> x & y
```

```
{3, 4, 5}
```

```
>>> x - y
```

```
{1, 2}
```

```
>>> x ^ y
```

```
{1, 2, 6, 7}
```

| 연산자 | 의미     |
|-----|--------|
|     | 합집합    |
| &   | 교집합    |
| -   | 차집합    |
| ^   | 대칭 차집합 |

# 기본 데이터 타입 : 형 변환

```
>>> float(1)
```

```
1.0
```

```
>>> int(3.14)
```

```
3
```

```
>>> str(3.14)
```

```
'3.14'
```

```
>>> tuple([1,2,3])
```

```
(1, 2, 3)
```

```
>>> list((1, 2, 3))
```

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

```
>>> float("3.14")
```

```
3.14
```

```
>>> float("Hello World")
```

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

```
File "<stdin>", line 1, in <module>
```

```
ValueError: could not convert string to float: „Hello World“
```



# 컨테이너 타입의 활용

1. 컨테이너 타입 소개
2. 순차 데이터 타입과 비순차 데이터 타입
- 3 가변 (Mutable) 데이터 타입과 불변(Immutable) 데이터 타입
- 4 인덱싱 (Indexing)
- 5 슬라이싱 (Slicing)
- 6 덧셈과 곱셈
- 7 요소 확인
- 8 추가 기능들

# 컨테이너 타입 소개

여러 데이터를 포함할 수 있는 데이터 타입이다.

컨테이너 타입의 종류는 문자열, 리스트, 튜플, 집합, 사적이 있

다. `str = "abcd"`

`list = [1,2,3,4]`

`Tuple = (1,2,3,4)`

`dict = {'name':'홍길동', 'age':20, 'addr':'seoul'}` `set =`

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

컨테이너 타입은 프로그램의 활용도가 높음으로 중요하다.

# 변수란(Identifier)?

- 변수명은 데이터에 붙인 이름이다.
- 즉 데이터에 다양한 이름을 부여할 수 있다.
- 문자, 숫자, 밑줄(\_)로 구성되어 있다.
- 키워드로는 구성할 수 없다.

```
>>> i1 = 1
```

```
>>> f1 = 3.14
```

```
>>> s1 = "Hello World"
```

```
>>> i1 1
```

```
>>> f1 3.14
```

```
>>> s1
```

```
"Hello World"
```

# 함수, 객체 지향

## ■ 함수

- 특정 행위(기능)를 수행하는 일정 코드 부분을 의미한다.
- 함수의 사용법은 수학에서 배운 함수와 동일하다.
- 예를 들어서 사인 함수는  $\sin(0)$ 와 같이 사용한다.
  - $\sin$ : 함수명,
  - 0:  $\sin$  함수에 입력한 인자 값
  - $\sin(0)$ 의 결과 값은 0이다

## ■ 객체 지향

- 객체는 속성과 행위(메서드, 기능)를 가지는 대상체이다.
- 속성은 객체가 가지는 값이며 행위는 객체가 수행할 수 있는 기능을 말한다.

```
>>> a = [1, 2, 3, 4]
```

```
>>> a
```

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

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

# 순차 데이터 타입과 비순차 데이터 타입

## ■ 순차 타입

- 인덱스 값을 이용해서 각 데이터 요소를 접근할 수 있는 데이터 타입
- 순차 데이터 타입의 인덱스 시작 값은 0 이다.
  - 문자열
  - 리스트
  - 튜플

## ■ 비순차 데이터 타입

- 요소의 순서화가 없다
- 인덱스 값을 이용해서 각 데이터 요소를 접근할 수 없는 데이터 타입
  - Dic 타입
  - 집합 타입

# 순차 타입

```
>>> s1 = "Hello World"
```

```
>>> s1[0]
```

```
H
```

```
>>> lst = [1, 3.14, True, "Hello World"]
```

```
>>> lst[0]
```

```
1
```

```
>>> lst[3][0]
```

```
H
```

```
>>> tpl = ("학생", 90, 95, 95)
```

```
>>> tpl[0]
```

```
학생
```

# 비순차 데이터 타입

```
>>> poly1 = {"triangle": 2, "rectangle": 3, "line": 1}
>>> poly1
{"line": 1, "triangle": 2, "rectangle": 3}
>>> poly1["line"]
1
>>> poly1["triangle"]
2
>>> poly2 = {1: "line", 2: "triangle", 3: "rectangle"}
>>> poly2[1]
"line"
```

# 반복 가능한(Iterable) 데이터와 반복자(Iterator)

## ■ 반복 가능한(Iterable) 데이터와 반복자(Iterator)

- 반복 가능한 데이터
  - 포함하고 있는 요소를 순차적으로 반복해서 접근할 수 있는 데이터를 말한다.
  - 순차 데이터 타입 : 문자열, 리스트, 튜플
- 반복자(Iterator)
  - 순차 데이터 타입의 요소를 순차적으로 접근할 수 있는 방법을 제공하는 객체이다.
  - 반복자 객체의 주요 사용 함수는 `next`이다.
  - `next`는 반복자 객체가 가리키는 요소들을 하나씩 순차적으로 접근 가능하게 한다.
- 순서를 다한 후에 반복자를 이용해서 데이터 요소에 접근하려고 하면 `StopIteration` 에러가 발생한다.



# 반복 가능한(Iterable) 데이터와 반복자(Iterator)

```
>>> s1 = {"line", "rectangle", "triangle"}
```

```
>>> s1
```

```
{'line', 'triangle', 'rectangle'}
```

```
>>> s1_iter = iter(s1)
```

```
>>> type(s1_iter)
```

```
<class 'set_iterator'>
```

```
>>> next(s1_iter)
```

```
'line'
```

```
>>> next(s1_iter)
```

```
'triangle'
```

```
>>> next(s1_iter)
```

```
'rectangle'
```

```
>>> next(s1_iter)
```

```
Traceback (most recent call last): File "<stdin>", line 1, in <module> StopIteration
```

# 가변(Mutable) 데이터 타입과 불변(Immutable) 데이터 타입

## ■ 가변 데이터 타입

- 내용이 변할 수 있는 데이터 타입
- 리스트, 집합, dic 데이터 타입

## ■ 불변 데이터 타입

- 내용이 변할 수 없는 데이터 타입이다.
- 가변을 제외한 7가지 타입

# 가변(Mutable) 데이터 타입

```
>>> lst = [2, 2, 3]
>>> lst
[2, 2, 3]
>>> lst[0] = 1
>>> lst
[1, 2, 3]
>>> lst.append(4)
>>> lst
[1, 2, 3, 4]
>>> lst.remove(4)
>>> lst
[1, 2, 3]

>>> poly1 = {"triangle": 2, "rectangle": 3, "line": 0}
>>> poly1
{'line': 0, 'triangle': 2, 'rectangle': 3}
>>> poly1["line"] = 1
>>> poly1
{'line': 1, 'triangle': 2, 'rectangle': 3}
```

# 불변(Immutable) 데이터 타입

- 기본 데이터 타입에서 문자열, 튜플 등은 불변 데이터 타입이다.
- 즉 문자열이나 튜플의 내용을 변경시키려는 행위는 오류를 발생시킨다.

```
>>> s1 = "Hello World"
```

```
>>> s1[0] = 'H'
```

```
Traceback (most recent call last): File "<stdin>", line 1, in <module>
```

```
TypeError: 'str' object does not support item assignment
```

```
>>> t1 = (1, 2, "Hello World", "안녕 파이썬")
```

```
>>> t1[0] = 3
```

```
Traceback (most recent call last): File "<stdin>", line 1, in <module>
```

```
TypeError: 'tuple' object does not support item assignment
```

# 인덱싱(Indexing)

- 순차 데이터 타입은 인덱스 값을 통하여 요소들을 접근할 수 있다.
- 첫 번째 요소의 인덱스 값은 0부터 시작한다.
- 따라서 마지막 요소의 인덱스 값은 “적체 요소의 개수 - 1”이 된다.
- 전체 개수가  $n$ 개라면 인덱스 값은 0부터  $n-1$ 까지가 된다. (순방향 인덱싱)

|     |     |     |     |           |
|-----|-----|-----|-----|-----------|
| [0] | [1] | [2] | ... | [ $n-1$ ] |
|-----|-----|-----|-----|-----------|

- 순차 데이터 타입의 순방향 인덱스 접근 뿐만 아니라 역순 인덱스 접근도 가능하다.
- 마지막 요소의 인덱스 값은 -1이고 이를 기준으로 역순 요소의 인덱스 값은 감소한다. (역방향 인덱싱)
- 결국 처음 요소는  $-n$ 의 값으로 접근이 가능하다.

|          |            |            |     |      |
|----------|------------|------------|-----|------|
| [ $-n$ ] | [ $-n+1$ ] | [ $-n+2$ ] | ... | [-1] |
|----------|------------|------------|-----|------|

# 인덱싱(Indexing)

```
>>> s1 = "Hello World"
```

```
>>> s1[0]
```

```
H
```

```
>>> s1[1]
```

```
e
```

```
>>> len(s1)
```

```
11
```

```
>>> s1[10]
```

```
d
```

```
>>> s1[-1]
```

```
d
```

```
>>> s1[-2]
```

```
l
```

```
>>> s1[-11]
```

```
H
```

# 슬라이싱(Slicing)

- 인덱스 범위 값을 이용해서 순차 데이터 타입의 일부를 참조하는 것을 말한다.
- 순차\_데이터[ [시작] :[끝]:[단계] ]
- 슬라이싱의 결과에서 끝 인덱스 요소는 포함되지 않는다.

```
>>> lst1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
>>> lst1[1:9:1]
```

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

```
>>> lst1[1:9]
```

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

```
>>> lst1[1:9:2]
```

```
[1, 3, 5, 7]
```

```
>>> lst1[9:1:-2]
```

```
[9, 7, 5, 3]
```

```
>>> lst1[-1:-11:-2]
```

```
[10, 8, 6, 4, 2]
```

# 덧셈과 곱셈

덧셈의 경우 두 순차 데이터를 연결시킨다.  
곱셈은 곱하는 만큼 연결시킨다.

```
>>> s1 = "Hello World" + "안녕 파이썬"
```

```
>>> s1
```

```
„HelloWorld 안녕파이썬“
```

```
>>> lst1 = [1, 2, 3] + [4, 5, 6]
```

```
>>> lst1
```

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

```
>>> tpl1 = (1, 2, 3) + (4, 5, 6)
```

```
>>> tpl1
```

```
(1, 2, 3, 4, 5, 6)
```

```
>>> s1 = "Hello World " * 3
```

```
>>> s1
```

```
„HelloWorld Hello World Hello World“
```

```
>>> lst1 = [1, 2, 3] * 3
```

```
>>> lst1
```

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

```
>>> tpl1 = (1, 2, 3) * 4
```

```
>>> tpl1
```

```
(1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3)
```



# 요소 확인

- 요소 확인을 위해서 in 과 not in 이 이용된다.
- in은 요소가 포함되어 있는 지를 확인하는데 이용된다.
- not in은 요소가 포함되어 있지 않음을 확인하는데 이용된다.

```
>>> s1 = "Hello World"
>>> "W" in s1 True
>>> "w" in s1 False
>>> lst1 = [1, 2, 3, 4, 5]
>>> 3 in lst1 True
>>> 6 not in lst1 True
>>> polygon = {"triangle": 2, "rectangle": 3, "line": 0}
>>> "line" in polygon True
>>> 0 in polygon
False
```

# 길이, 최대값, 최소값 확인

- len: 컨테이너 데이터의 요소 개수를 구한다.
- max: 컨테이너 데이터의 최대값을 구한다.
- min: 컨테이너 데이터의 최소값을 구한다.

```
>>> s1 = "Hello World"
>>> len(s1) 11
>>> lst1 = [1, 2, 3, 4, 5]
>>> len(lst1) 5
>>> polygon = {"triangle": 2, "rectangle": 3, "line": 0}
>>> len(polygon) 3
>>> tpl1 = (1, 2, 3, 4, 5)
>>> len(tpl1) 5
>>> st1 = {"red", "green", "blue"}
>>> len(st1) 3
```

# 길이, 최대값, 최소값 확인

```
>>> s1 = "Hello World"
```

```
>>> max(s1)
```

```
W
```

```
>>> lst1 = [1, 2, 3, 4, 5]
```

```
>>> max(lst1)
```

```
5
```

```
>>> tpl1 = (1, 2, 3, 4, 5)
```

```
>>> max(tpl1)
```

```
5
```

```
>>> st1 = {"red", "green", "blue"}
```

```
>>> max(st1)
```

```
red
```

```
>>> s1 = "Hello World"
```

```
>>> min(s1)
```

```
H
```

```
>>> lst1 = [1, 2, 3, 4, 5]
```

```
>>> min(lst1)
```

```
1
```

```
>>> tpl1 = (1, 2, 3, 4, 5)
```

```
>>> min(tpl1)
```

```
1
```

```
>>> st1 = {"red", "green", "blue"}
```

```
>>> min(st1)
```

```
blue
```

# 연산자

1. 수치 연산자 (Arithmetic Operators)
2. 대입 연산자 (Assignment Operators)
3. 비교 연산자 (Comparison Operators)
4. 논리 연산자 (Logical Operators)
5. 비트 연산자 (Bitwise Operators)
6. 식별 연산자 (Identity Operators)
7. 구성원 연산자(Membership Operators)
8. 문자열 연산자(String Operators)

# 수치 연산자(Arithmetic Operators)

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
>>> 2 ** 3
8
>>> 3.14 ** 2
9.8596
>>> 2 ** 0.5
1.4142135623730951
>>> 3 // 2
1
>>> 3 % 2
1
```

| 수치연산자 | 설명             |
|-------|----------------|
| +     | 덧셈을 수행한다.      |
| -     | 뺄셈을 수행한다.      |
| *     | 곱셈을 수행한다.      |
| /     | 나눗셈을 수행한다.     |
| **    | 거듭제곱을 수행한다.    |
| //    | 나눗셈의 몫을 구한다.   |
| %     | 나눗셈의 나머지를 구한다. |

# 수치 연산자(Arithmetic Operators)

- +=, -=, \*=, /=, \*\*=, //=, %=은 복합 대입연산자이다.

a = a + b

a += b

```
>>> val = 20
```

```
>>> val += 5
```

```
>>> val
```

```
25
```

```
>>> val -= 10
```

```
>>> val
```

```
>>> val *= 2
```

```
>>> val
```

```
>>> val /= 3
```

```
10.0
```

# 비교 연산자(Comparison Operators)

```
>>> var = 95
>>> var == 90
False
>>> var != 90
True
>>> var > 90
True
>>> var < 90
False
>>> var >= 90
True
>>> var <= 90
False
```

| 비교 연산자     | 설명                           |
|------------|------------------------------|
| 수식1 == 수식2 | 수식1과 수식2 값이 같음을 평가           |
| 수식1 != 수식2 | 수식1과 수식2 값이 같지 않음을 평가        |
| 수식1 > 수식2  | 수식1의 값이 수식2의 값 보다 큰가를 평가     |
| 수식1 < 수식2  | 수식2의 값이 수식1의 값 보다 큰가를 평가     |
| 수식1 >= 수식2 | 수식1의 값이 수식2의 값 보다 같거나 큰가를 평가 |
| 수식1 <= 수식2 | 수식2의 값이 수식1의 값 보다 같거나 큰가를 평가 |

범위 확인은 일반적인 수학 표현을  
사용 할 수 있다.

```
>>> var = 95
>>> 90 <= var <= 100
True
>>> var = 70
>>> 90 <= var <= 100
False
>>> var = 110
>>> 90 <= var <= 100
False
```

# 논리 연산자(Logical Operators)

| 논리 연산자 | 설명                   |
|--------|----------------------|
| and    | 논리곱 (and) 연산을 수행한다.  |
| or     | 논리합 (or) 연산을 수행한다.   |
| not    | 논리 부정(not) 연산을 수행한다. |

| 입력 값  |       | 논리 연산   |        |       |
|-------|-------|---------|--------|-------|
| A     | B     | A and B | A or B | not A |
| True  | True  | True    | True   | False |
| True  | False | False   | True   | False |
| False | True  | False   | True   | True  |
| False | False | False   | False  | True  |

```
>>> grade = 4.3
>>> register = 7
>>> (4.0 <= grade) and (register >= 5)
True
```



# 비트 연산자(Bitwise Operators)

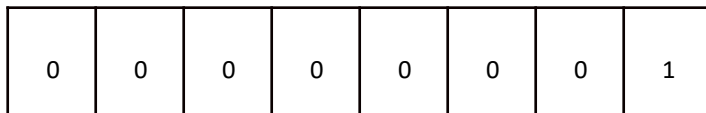
```
>>> data=0x12
>>> mask= 0x3
>>> data & mask
2
>>> data | mask
9
>>> data ^ mask
7
>>> ~data
-19
>>> data = 0
>>> ~data
-1
```

| 비트 연산자 | 설명                    |
|--------|-----------------------|
| &      | 비트 논리곱(and) 연산을 수행한다. |
|        | 비트 논리합(or) 연산을 수행한다.  |
| ^      | 비트 xor 연산을 수행한다.      |
| ~      | 비트 논리 부정을 수행한다.       |

# 쉬프트 연산자

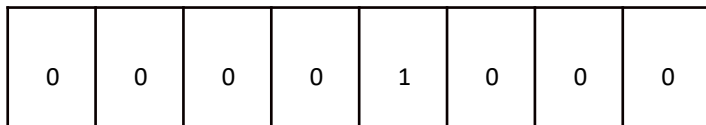
- 이진 데이터로 표현된 상태에서 왼쪽 혹은 오른쪽으로 정해진 만큼 이동시킨다.
- 좌측 쉬프트의 경우 오른쪽 빈 공백만큼은 0으로 채워진다. 이는 쉬프트 횟수만큼 2를 곱하는 효과가 난다.
- 우측 쉬프트의 경우 부호가 유지되면서 오른쪽으로 이동시킨다. 이는 쉬프트 횟수만큼 2로 나누는 효과가 난다.

| 비트 연산자 | 설명                      |
|--------|-------------------------|
| <<     | 좌측 쉬프트(shift) 연산을 수행한다. |
| >>     | 우측 쉬프트(shift) 연산을 수행한다. |



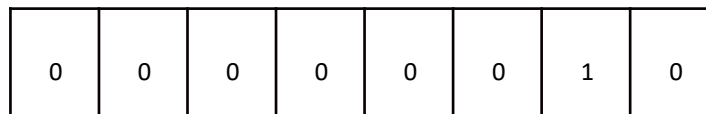
# 1

1 << 3



# 8

8 >> 2



# 2



# 식별 연산자(Identity Operators)

| 비교 연산자 | 설명                        |
|--------|---------------------------|
| is     | 두 대상체가 동일한 대상체인지를 체크한다.   |
| is not | 두 대상체가 서로 다른 대상체인지를 체크한다. |

```
>>> lst = [ 1, 2, "Hello World"]
>>> lst1 = [ 1, 2, "Hello World"]
>>> lst is lst1 False
>>> id(lst) == id(lst1) False
>>> lst == lst1
True
>>> lst2 = lst
>>> lst is lst2
True
```

# 구성원 연산자(Membership Operators)

| 비교 연산자 | 설명                   |
|--------|----------------------|
| in     | 컨테이너에 요소가 있는지를 확인한다. |
| not in | 컨테이너에 요소가 없는지를 확인한다. |

```
>>> lst = [1, 2, "Hello World"]
>>> 1 in lst
True
>>> 3 in lst
False
>>> polygon = {"triangle": 2, "rectangle": 3, "line": 1}
>>> "line" in polygon
True
>>> "circle" in polygon
False
>>> 1 in polygon
False
```

# 문자열 연산자(String Operators)

- 파이썬에서 +는 두 문자열을 연결시키는 용도로도 사용된다.
- “Hello “+ “World”는 “Hello World”를 산출해 낸다.

```
>>> "Hello " + "World"
„Hello World“
>>> "안녕 " + "파이썬"
„안녕 파이썬“
```

## 문자열 포매팅 연산자

- 문자열을 주어진 양식에 맞추어서 생성하는 것을 말한다.
- 연산자로서 %와 문자열 포맷 코드를 제공한다.

| 포맷 코드 | 설명                    |
|-------|-----------------------|
| %s    | 문자열 (string)          |
| %c    | 문자 (character)        |
| %d    | 정수 (integer)          |
| %f    | 부동소수 (floating-point) |
| %o    | 8진수                   |
| %x    | 16진수(소문자)             |
| %X    | 16진수(대문자)             |
| %e    | 과학적 수치 표현(소문자)        |
| %E    | 과학적 수치 표현(대문자)        |

# 문자열 연산자(String Operators)

```
>>> "Hello %s" % "World"
„Hello World
>>> "%d" % 5
5
>>> "%f" % 3.14
„3.140000“
>>> "%e" % 3.14
„3.140000e+00
>>> "%s loves %s" % ("yj", "sy")
„jylves s“
>>> "%s loves No.%d" % ("yj", 7)
„jylves No5“
>>> "%d x %d = %d" % (6, 9, 54)
„6x 9 = 54“
>>> "2 x %f x %d = %f" % (3.14, 3, 2 * 3.14 * 3)
```

# 문자열 연산자(String Operators)

| 양식 지정 코드 | 설명                                        |
|----------|-------------------------------------------|
| n        | 총 길이를 나타낸다.                               |
| -        | 좌측 정렬을 한다.                                |
| +        | 부호를 표시 한다.                                |
| 0        | 스페이스를 0으로 대체한다.                           |
| m.n      | m은 최소 전체 길이를 나타내고<br>n은 소수점 이하의 길이를 나타낸다. |

```
>>> "%10s"%"Hi!" '
      Hi!'
>>> "%-10s"%"Hi!" 'Hi
!      '
>>> "%010d" % 500000
„0000500000“
```

```
>>> "pi is %010.3f" %3.14
'pi is 000003.140'
>>> "pi %.2f" % 3.14
„pi 3.14“
```

## 제 어 문

1. 흐름과 흐름 제어
2. 선택 흐름과 if 문
3. for 문
4. while 문
5. break 문과 continue 문
6. pass 문
7. 무한 반복



# 제 어 문

프로그램의 첫 줄부터 마지막까지 한 줄씩 수행하는 흐름(Flow)이 있다.

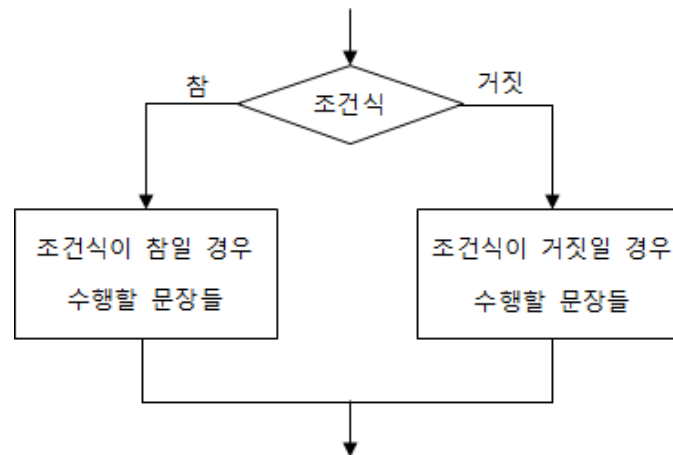
```
print("1")  
print("2")  
print("3")
```

조건문으로 흐름을 제어할 수 있다.

if else 문

문법:

```
if 조건식:  
    문장  
else:  
    문장들
```



# 제 어 문

## 데이터 입력 받기

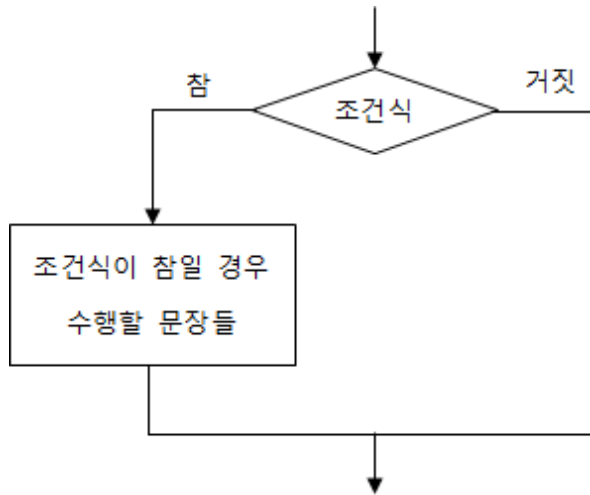
`input([프롬프트]) -> 문자열`

```
data = int(input("숫자를 입력하시오: "))
if data % 2 == 0:
    print("입력된 값은 짝수입니다.")
else:
    print("입력된 값은 홀수입니다.")
```

```
score = int(input("점수를 입력하시오: "))
if score >= 70:
    print("당신은 시험을 통과했습니다.")
else:
    print("당신은 시험을 통과하지 못했습니다.")
    print("공부 열심히 하세요!")
```

# 제 어 문

if 문만 있는 경우



아무 명령문도 입력하지 않고자  
하는 경우에 **pass** 문을 입력함

```
grade = float(input("총 평점을 입력해 주세요: "))
```

```
if grade >= 4.3:
```

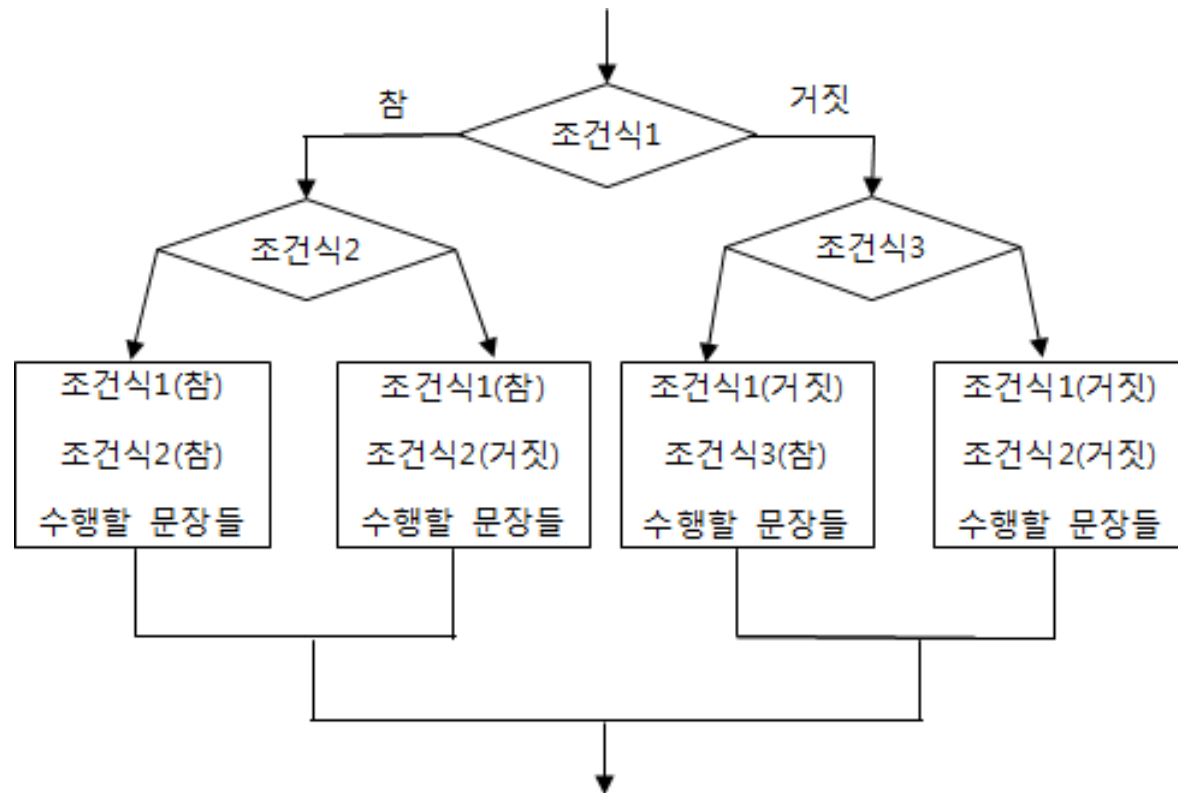
```
    print("당식은 장학금 수여 대상자 입니다.")
```

```
    print("축하합니다.")
```

```
print("공부 열심히 하세요.")
```

# 제 어 문

중첩된 if ~ else 문



# 제 어 문

중첩된 if ~ else 문

```
age = int(input("나이를 입력 하시오"))
height = int(input("키를 입력 하시오"))

if age >= 40:
    if height >= 170:
        print("40이상 : 키가 보통 이상 입니다.")
    else:
        print("40이상 : 키가 보통입니다.")
else:
    if height >= 175:
        print("40미만 : 키가 보통 이상 입니다.")
    else:
        print("40미만 : 키가 보통입니다.")
```

# 제 어 문

## 중첩된 if ~ elif ~ else 문

```
score = int(input("총점을 입력해 주세요: "))
```

```
if score >= 90:
```

```
    print("수")
```

```
else:
```

```
    if 80 <= score < 90: pr
```

```
        int("우")
```

```
else:
```

```
    if 70 <= score < 80: pr
```

```
        int("ㅁ")
```

```
else:
```

```
    if 60 <= score < 70:
```

```
        print("양")
```

```
    else:
```

```
        print("가")
```

```
score = int(input("총점을 입력해 주세요: "))
```

```
if score >= 90:
```

```
    print("수")
```

```
elif 80 <= score < 90: pri
```

```
    nt("우")
```

```
elif 70 <= score < 80:
```

```
    print("ㅁ")
```

```
elif 60 <= score < 70: pri
```

```
    nt("양")
```

```
else:
```

```
    print("가")
```

# for문

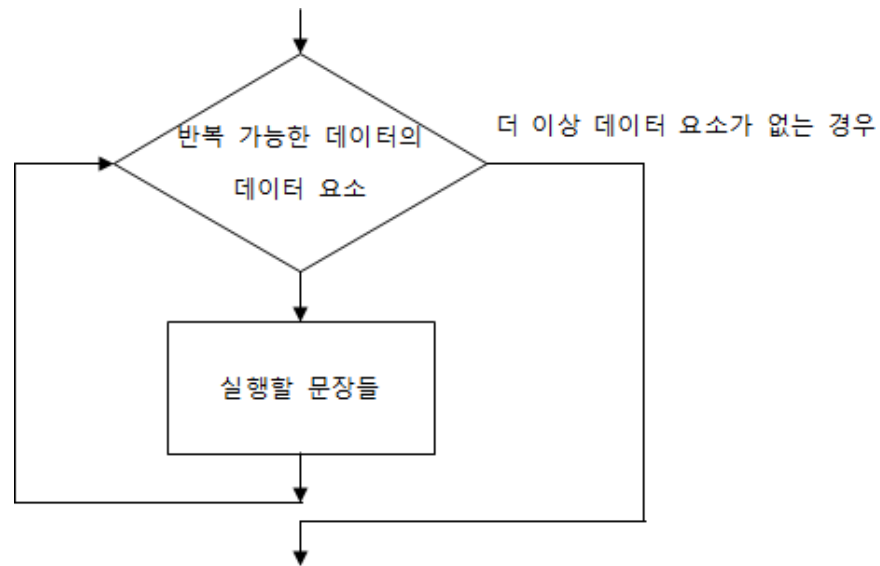
- for문의 문법

for 변수 in 반복\_가능한\_데이터 :

수행할 문장들

[else:

수행할 문장들]



# for 문

```
message = "Hello!"
messages = ["Hello World", "안녕, 파이썬"]
numbers = (1, 2, 3)
polygon = {"triangle": 2, "rectangle": 3, "line": 1}
color = {"red", "green", "blue"}

for item in message:
    print(item)
print( )

for item in messages:
    print(item)
print( )

for item in numbers:
    print(item)
print( )

for item in polygon:
    print(item)
print( )

for item in color:
    print(item)
```



## for 문

```
total = 0  
for item in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:  
    total = total + item  
print("1부터 10까지 합은", total, "입니다")
```

```
a = [(1,2), (3,4), (5,6)]  
for (first, last) in a:  
    print(first + last)
```

# range 함수

range 함수는 범위를 생성하는 함수

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

```
total = 0
for item in range(1, 11): tota
    l = total + item
print("1부터 10까지 합은", total, "입니다")
```

# range 함수

```
total = 0
for item in range(1, 101):
    total = total + item
print("1부터 100까지 합은", total, "입니다")
```

```
total = 0
for item in range(1, 101, 2):
    total = total + item
print("1부터 100까지 짝수 합은", total, "입니다")
```

```
total = 0
for item in range(1, 101):
    if (item % 3) == 0 or (item % 7) == 0:
        total = total + item
print("1부터 100까지에서 3 혹은 7의 배수의 합은", total, "입니다")
```

# list에 내포된 for 구문(list comprehension)

- 리스트 구문 안에 for문을 직접 구현하여 쉽게 리스트를 생성할 수 있음
- 방법
  - `arr = [표현식 for 항목 in 반복가능객체]`
  - `arr = [표현식 for 항목1 in 반복가능객체1  
for 항목2 in 반복가능객체2]`
  - `arr = [표현식 for 항목 in 반복가능객체 if 조건]`
  - `arr = [표현식 for 항목1 in 반복가능객체1 if 조건  
for 항목2 in 반복가능객체2 if 조건]`

```
arr = [1,2,3,4]
result = []
for num in arr:
    result.append(num*3)
print(result)

result = [num * 3 for num in arr]
print(result)
```

## list에 내포된 for 구문(list comprehension)

- `arr = [표현식 for 항목1 in 반복가능객체1  
for 항목2 in 반복가능객체2]`

```
arr = [1,2,3,4]
brr = [10,20,30]
```

```
result = [x * y for x in arr
           for y in brr]
print(result)
```

- `arr = [표현식 for 항목 in 반복가능객체 if 조건]`

```
arr = [1,2,3,4,5,6,7,8,9,10]
```

```
result = [x * 2 for x in arr if x % 2 == 0]
print(result)
```

## list에 내포된 for 구문(list comprehension)

- `arr = [표현식 for 항목1 in 반복가능객체1 if 조건  
for 항목2 in 반복가능객체2 if 조건]`

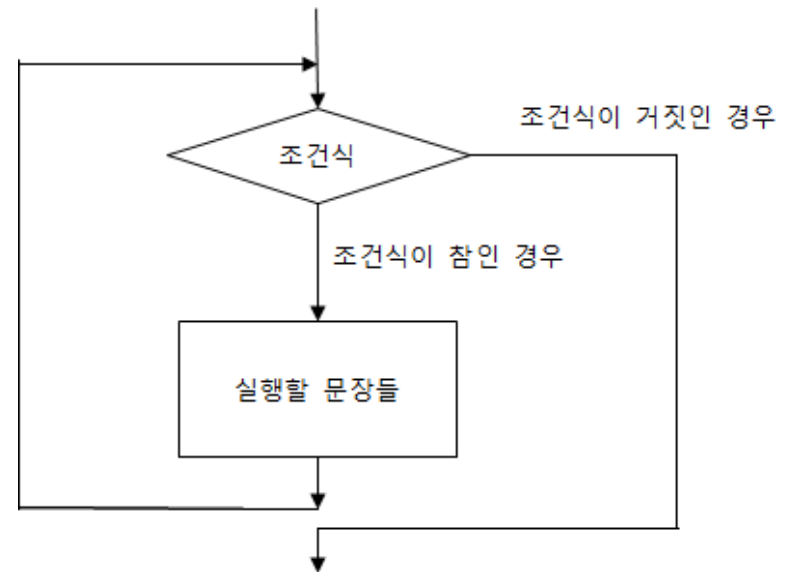
```
arr = [1,2,3,4,5,6,7,8,9,10]
brr = [1,2,3,4,5,6,7,8,9,10]
result = [x * y for x in arr if x % 2 == 0
          for y in brr if y % 3 == 0]
print(result)
```

# while 문

- while 문은 조건식을 검사하여 참인 경우까지 블록 내부 문장을 반복 수행한다

while문의 문법은 다음과 같다

```
while 조건식: 실행할 문장들
[else: 실행할 문장들]
```



# while 문

```
count = 1
while count <= 10: pr
    int(count)
    count = count + 1
count = 1
```

```
total = 0
count = 1
while count <= 100: total =
    total + count count = c
    ount + 1
print("1부터 100까지의 합은:", total)
```

```
count = 1
result = 0
while count <= 100: result = r
    esult + count
    count = count + 1
else:
    print("덧셈이 작업 완료 되었습니다.")
print("1부터 100까지의 합은:", result)
```



# break문과 continue 문

- break 문

현재 수행하고 있는 반복문을 빠져 나와서 다음 단계의 문장을 수행

- continue문

반복문에서 남은 문장을 수행하지 않고 다음 단계로 넘어감

```
break_letter = input("중단할 문자를 입력하시오: ")
for letter in "python":
    if letter == break_letter: break
    print(letter)
else:
    print("모든 문자 출력 완료!")
```

```
continue_letter = input("건너뛸 문자를 입력하시오: ")
for letter in "python":
    if letter == continue_letter: continue
    print(letter)
```

# 무한 반복

반복문을 빠져 나오지 못하고 특정 문장들을 계속 반복하는 경우.

```
choice = None
while True:
    print("1. 원 그리기")
    print("2. 사각형 그리기")
    print("3. 선 그리기")
    print("4. 종료")
    choice = input("메뉴를 선택하시오: ")
    if choice == "1":
        print("원 그리기를 선택했습니다.")
    elif choice == "2":
        print("사각형 그리기를 선택했습니다.")
    elif choice == "3":
        print("선 그리기를 선택했습니다.")
    elif choice == "4":
        print("종료합니다.")
        break
    else:
        print("잘못된 선택을 했습니다.")
```

## Function

1. 파이썬 함수종류
2. 사용자 정의 함수
3. 함수의 호출과 흐름
4. 함수의 인자와 반환값
5. 함수를 인자로 전달하기
6. 기본 함수(Built-in 함수)
7. 라이브러리(패키지) 함수

# 파이썬 함수종류

## ■ 함수의 세 가지 종류

### ■ 사용자 정의 함수

- 사용자가 자신이 필요로 하는 기능을 수행하는 함수를 작성한 함수들

### ■ 기본 함수 혹은 built-in 함수

- 기본 함수는 파이썬의 실행과 동시에 사용할 수 있는 함수들.

- `len()`, `sum()`, `min()`, `max()`, `sorted()`

- `id()`, `type()`

### ■ 라이브러리 혹은 패키지 함수

- 해당 라이브러리를 포함한 후에 사용할 수 함수들.

- Import 라이브러리 : `import time`

- From 라이브러리 import 함수명 as 축약 : `os.getcwd()`

# 파이썬 함수종류

## ■ random 모듈 활용

- `>>> import random`
- `>>> random.random()`
- `0.90389642027948769`
- `>>> random.randrange(1,7)` # 1에서 7보다 작은 사이의 난수 발생
- `6`
- `>>> random.randrange(1,7)`
- `2`
- `>>> abc = ['a', 'b', 'c', 'd', 'e']`
- `>>> random.shuffle(abc)` # container 안의 내용을 랜덤하게 섞기
- `>>> abc`
- `['a', 'd', 'e', 'b', 'c']`
- `>>> random.shuffle(abc)`
- `>>> abc`
- `['e', 'd', 'a', 'c', 'b']`
- `>>> menu = '짬면', '육계장', '비빔밥'`
- `>>> random.choice(menu)`
- `'짬면'`
- # 1부터 48 사이의 숫자 중 아무거나 뽑기
- # A부터 z 사이의 문자 중에서 아무거나 뽑기
- # True, False 둘 중에 하나 뽑기

# 파이썬 함수종류

## ■ 함수를 작성하는 이유

- 특정의 기능을 수행하는 코드들을 하나의 묶음으로 사용
- 재사용성을 높이고 코드의 통일된 관리를 하기 위해 함수를 작성함.

## ▪ 함수 문법

```
def 함수명 ([인자1, 인자2, ...]):  
    수행할 문장들  
    return 반환값
```

# 사용자 정의 함수

## ■ 함수의 인자전달 시 값 전달과 객체 전달 차이 이해

```
def call_by_value(num, mlist) :  
    num = num + 1  
    mlist.append("add 1")
```

```
num = 10  
mlist = [1,2,3]
```

```
print(num, mlist)  
call_by_value(num, mlist)  
print(num, mlist)
```

# 사용자 정의 함수

## ■ name 변수

- 파이썬 인터프리터가 파이썬 프로그램을 입력 받아서 실행하면 `__name__`을 “`__main__`”으로 설정됨.

```
def hello_message( ): pri
    nt("Hello World")

if __name__ == "__main__": hell
    o_message( )
```



# 사용자 정의 함수

## ■ 함수의 인자와 반환값

- 함수를 실행할 때 외부로부터 인자를 받아서 처리할 수 있다.
- 외부로부터 넘어온 값은 함수 내부에서 자유롭게 사용이 가능하다.
- 함수는 작업을 마친 후 호출한 지점으로 돌아갈 때 반환값을 되돌려 줄 수 있다.
- return 문
  - return           # 제어를 되돌리고 None 값을 반환
  - return 반환값   # 제어를 되돌리고 반환 값을 반환

# 사용자 정의 함수

## 함수의 인자와 반환값

```
def hello_message( repeat_count ) :  
    for item in range(repeat_count) :  
        print("Hello World")  
if __name__ == "__main__":  
    hello_message(1)  
    hello_message(2)
```

```
def circle_area(radius, pi):  
    area = pi * (radius ** 2)  
    return area  
if __name__ == "__main__":  
    print("반지름:", 3, "PI:", 3.14, "면적:", circle_area(3, 3.14))  
    print("반지름:", 3, "PI:", 3.1415, "면적:", circle_area(3, 3.1415))
```

# 사용자 정의 함수

## 함수의 인자와 여러 개의 반환값

```
def circle_area_circumference(radius, pi):  
    area = pi * (radius ** 2)  
    circumference = 2 * pi * radius  
    return area, circumference  
  
if __name__ == "__main__":  
    result = circle_area_circumference(3, 3.14)  
    print("반지름:", 3, "면적과 둘레:", result)  
    res1, res2 = circle_area_circumference(3, 3.14)  
    print("반지름:", 3, "면적:", res1, "둘레: ", res2)
```

# 사용자 정의 함수(인자)

- 기본인자(Default Argument)

값이 전달되지 않은 경우 기본값이 전달된 것으로 인식함  
다른 언어의 오버로딩 효과를 나타낼 수 있다.

```
def circle_area(radius, pi=3.14 ):
```

```
    area = pi * (radius ** 2)
```

```
    return area
```

```
if __name__ == "__main__":
```

```
    print("반지름 :", 3, "면적 :", circle_area(3,3.14))
```

```
    print("반지름 :", 3, "면적 :", circle_area(3) )
```

# 사용자 정의 함수(인자)

키워드 사용 인자 전달

( 키워드 인자 사용 후 기본 인자를 사용하면 인식이 되지 않는다. )

```
def circle_area(radius, pi):  
    area = pi * (radius ** 2)  
  
if __name__ == "__main__":  
    print("반지름 : ", 3, "면적 : ", circle_area(3, pi = 3.14))  
    print("반지름 : ", 3, "면적 : ", circle_area(radius=3, pi=3.14))  
    print("반지름 : ", 3, "면적 : ", circle_area(pi=3.14, radius=3))  
    # print("반지름 : ", 3, "면적 : ", circle_area(radius=3, 3.14))
```

# 사용자 정의 함수(인자)

- 가변인자(Arbitrary Argument)

개수가 지정되지 않고 여러 개가 들어오는 인자를 받고자 하는 인자  
변수명 앞에 \* 한 개를 사용한다  
전달된 내용은 내부적으로 tuple로 적용됨  
Java 언어의 ... 변수와 같은 역할

```
def do_any_sum( *num ) :
```

```
    sum = 0
```

```
    for su in num :
```

```
        sum += su
```

```
    return sum
```

```
if __name__ == "__main__" :
```

```
    print("합계 : ", do_any_sum(1))
```

```
    print("합계 : ", do_any_sum(1,2,3) )
```

```
    print("합계 : ", do_any_sum(1,4,5,6,7))
```

```
    # print(" 합계: ", do_any_sum( (1,2,3,4,5) )) tuple 전달은 일반변수로 받아야 함
```

# 사용자 정의 함수(인자)

- 정의되지 않은 인자 전달

정의되지 않는 인자 전달이 가능하며, 이는 dic 타입으로 전달됨  
일반인자->가변인자->정의되지 않은 인자 순으로 전달되어야 함

```
def circle_area( radius, *pi, **info ):
    for item in pi :
        area = item * (radius ** 2 )
        print("반지름 : " , radius, "PI : ", item, "면적 : ", area)

    for key in info :
        print(key , " : ", info[key])

if __name__ == "__main__" :
    circle_area(3, 3.14, 3.1456, 3.141592)
    print()
    circle_area( 3, 3.14, 3.1456, line_color="red", area_color="green )
```

# 사용자 정의 함수

함수의 인자로 함수 전달하기(고차함수)

round함수 : 숫자 자릿수 처리함수

```
def circle_area(radius, print_format):
```

```
    area = 3.14 * (radius ** 2)
```

```
    print_format(area)
```

```
def precise_low(value):
```

```
    print("결과값:", round(value, 1)) # 반올림해서 소수점 한자리까지 출력
```

```
def precise_high(value):
```

```
    print("결과값:", round(value, 2)) # 반올림해서 소수점 둘째자리까지 출력
```

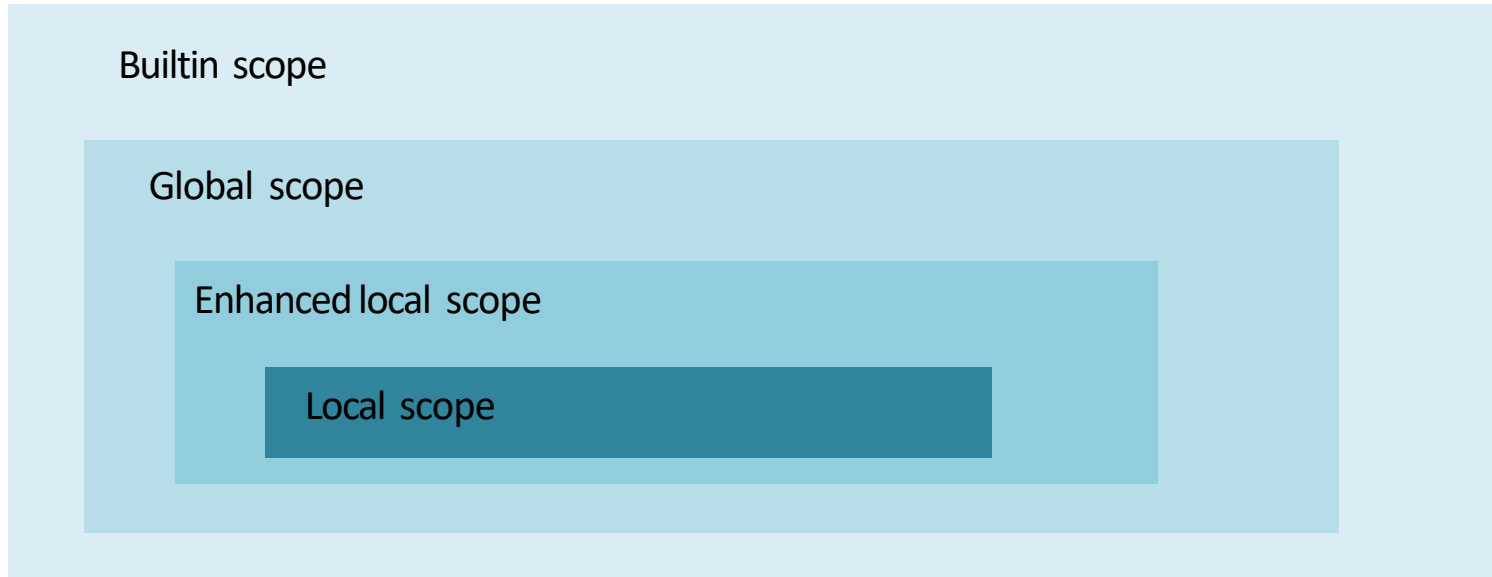
```
if __name__ == "__main__":
```

```
    circle_area(3, precise_low)
```

```
    circle_area(3, precise_high)
```



# 함수의 스코프



- Builtin scope : 파이썬이 제공하는 내장모듈 공간
- Global scope : 함수안에 포함되지 않은 전역 공간
- Enhanced local scope : 파이썬은 함수를 중첩가능함. 중첩된 영역의 공간
- Local scope : 함수 내에 정의된 지역 공간

# 함수의 스코프

```
g_val = 3
def foo():
    l_val = 2
foo()
```

- g\_val은 프로그램 전역공간에서 사용가능
- l\_val은 foo함수 안에서만 사용가능
- foo 함수 밖에서 l\_val을 사용 한다면?

```
g_val = 3
def foo():
    g_val = 2
foo()
print g_val
```

- foo함수 밖의 g\_val은 전역변수
- foo함수 안의 g\_val은 지역변수 (l-value)
- 출력 결과는?

```
g_val = 3
def foo():
    l_val = g_val
    print l_val
foo()
```

- foo 함수 안의 g\_val은 전역변수 (r-value)
- 따라서 foo함수 안에서 사용 가능
- 출력 결과는?

# 함수의 스코프

```
g_val = 3
def foo():
    global g_val
    g_val = 2
foo()
print g_val
```

- foo 함수 안에서 g\_val의 값이 l-value에 오면 지역변수
- foo 함수 안에서 전역변수 g\_val의 값을 바꾸려면?
  - global 키워드를 사용해야 함
  - global 키워드 이후로는 g\_val을 전역변수로 인식함
- 출력 결과는?

- 전역변수를 남발하면?
  - 문제 발생시 전역변수의 값을 참조하는 모든 코드를 디버깅해야함
  - 전역변수를 많이 참조 할수록 디버깅은 어려워짐
  - 함수 안의 범위만 쉽게 디버깅 하기 위해서 지역변수를 사용을 권고함.

# 함수의 스코프

```
g_val = 3
def foo():
    global g_val
    g_val = 2
foo()
print g_val
```

- foo 함수 안에서 g\_val의 값이 l-value에 오면 지역변수
- foo 함수 안에서 전역변수 g\_val의 값을 바꾸려면?
  - global 키워드를 사용해야 함
  - global 키워드 이후로는 g\_val을 전역변수로 인식함
- 출력 결과는?

- 전역변수를 남발하면?
  - 문제 발생시 전역변수의 값을 참조하는 모든 코드를 디버깅해야함
  - 전역변수를 많이 참조 할수록 디버깅은 어려워짐
  - 함수 안의 범위만 쉽게 디버깅 하기 위해서 지역변수를 사용을 권고함.

# 함수의 스코프 - 예제

```
x = "global"
def foo():
    print("x inside :", x)
foo()
print("x outside:", x)
```

```
x = "global"
def foo():
    x = x * 2  # 오류 발생
    print(x)
foo()
```

```
def foo():
    y = "local"
foo()
print(y) #오류 발생
```

```
x = 5
def foo():
    x = 10
    print("local x:", x)
foo()
print("global x:", x)
```

# 함수의 스코프

## ■ nonlocal vs global

```
x = 'global'
def outer():
    x = "local"

    def inner():
        nonlocal x
        x = "nonlocal"
        print("inner:", x)

    inner()
    print("outer:", x)
outer()
print("global : ", x)
```

```
def scope_test():
    def do_local():
        spam = "local spam"
        print("do local", spam)
    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"
    def do_global():
        global spam
        spam = "global spam"
    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)
scope_test()
print("In global scope:", spam)
```

# 람다(Lamda) 함수 == 익명함수

## Lambda 인자들 : 표현식

- 아주 작으며 이름이 없는 함수
- 문법적으로는 하나의 표현식만 사용되어야 함

```
def hap(x, y):  
    return x + y
```

```
print(hap(10, 20))  
print(( lambda x,y: x + y)(10, 20) )
```

# 람다(Lamda) 함수 == 익명함수

## 람다 함수를 인자로 전달

```
pairs = [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]
```

```
def shortKey(pair) :  
    return pair[1]
```

```
pairs.sort(key=shortKey)
```

```
# pairs.sort(key=lambda pair: pair[0])  lambdar로 함수 전달
```

```
print(pairs)
```



# 람다(Lamda) 함수 == 익명함수

## 함수를 반환하는 함수

```
def make_incrementor(n):  
    print("n", n)  
    return lambda x: x + n
```

```
f = make_incrementor(42) # f 함수를 생성함
```

```
print(make_incrementor(0))  
print(f)  
print(f(0))
```

```
print(f(1))
```

```
print(make_incrementor(65)(1))
```

# 람다(Lamda) 함수 == 익명함수

## 활용 예제

### map()

**map(함수, 리스트)**

리스트로부터 원소를 하나씩 꺼내서 함수를 적용시킨 다음, 그 결과를 새로운 목록을 map 객체로 반환하는 함수

```
arr = [1,2,3,4,5]
brr = map(lambda x : x**2, arr)
print(brr)
crr = list(brr)
print(crr)
```

### filter()

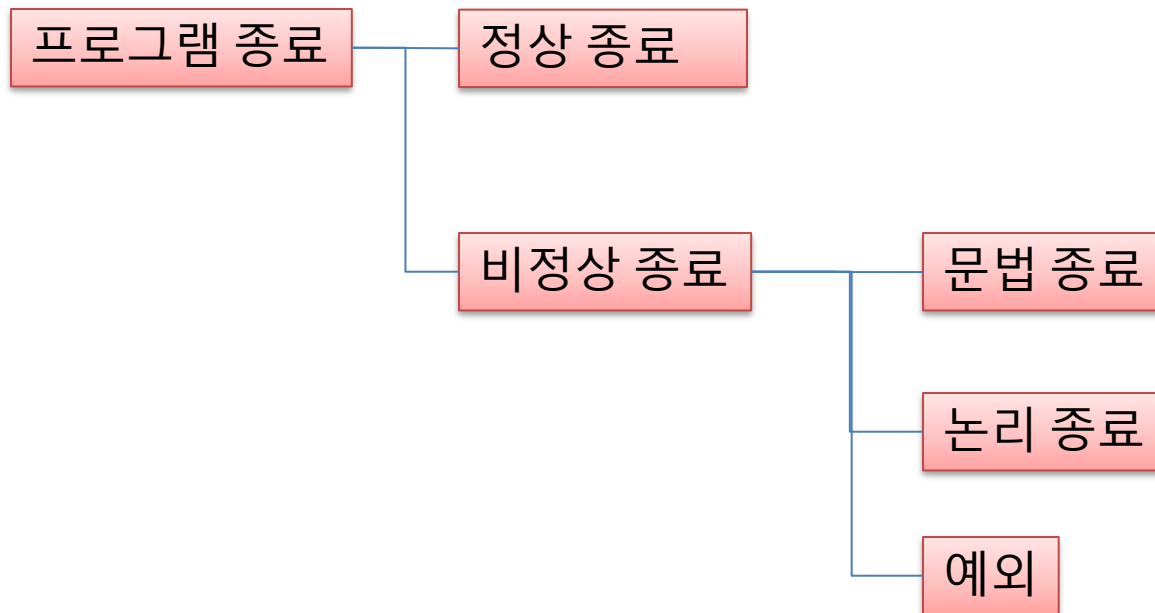
**filter(함수, 리스트)**

리스트로부터 원소를 하나씩 꺼내서 함수를 적용시킨 다음, 조건에 만족하는 원소만 새로운 목록으로 반환하는 함수

```
arr = [1,2,3,4,5,6,7,8,9]
brr = filter(lambda x : x%3 == 0, arr)
print(brr)
#print(next(brr))
for item in brr :
    print(item, end="")
```

# 예외

1. 예외
2. 예외 처리 구문



# 예외

## ■ 정상 종료

- 프로그램이 수행 중에 어떤 오류도 없었고 결과 또한 아무런 문제가 없는 경우이다.

## ■ 문법 오류

- 문법 오류는 프로그램 내에 파이썬 문법에 어긋나는 코드가 있는 경우에 발생한다.
- 파이썬 인터프리터가 문법 오류를 출력함
- 출력된 결과를 바탕으로 오류를 수정해야 함

## ■ 논리 오류

- 논리 오류는 프로그램이 잘못된 결과를 산출하는 경우를 말한다.
- 디버거를 이용하여 실행 과정을 추적하고 잘못된 데이터를 산출하는 부분을 수정함으로 써 오류를 제거한다.

# 예 외

## ■ 예 외

- 문법적인 오류나 논리적 오류가 없지막 오류를 발생시키는 경우를 말한다.
- 0으로 나누기, 리스트 데이터에 인덱스 범위를 넘어서서 접근하려는 경우
- 프로그램 수행 도중 ctrl+c 를 입력하는 경우 등

```
Traceback (most recent call last): File "test.p
y", line 4, in <module>
    print a[4]
IndexError: list index out of range
```

```
^CTraceback (most recent call last): File "te
st.py", line 3, in <module>
    time.sleep(1)
KeyboardInterrupt
```

```
>>> 3/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

## ■ 파이썬의 Exception 상속도.

```
BaseException
├── SystemExit
├── KeyboardInterrupt
├── Exception
│   ├── ArithmeticError
│   │   └── ZeroDivisionError
│   ├── AssertionError
│   ├── AttributeError
│   ├── EOFError
│   ├── ImportError
│   │   └── ModuleNotFoundError
│   ├── LookupError
│   │   ├── IndexError
│   │   └── KeyError
│   ├── NameError
│   ├── OSError
│   │   ├── ChildProcessError
│   │   ├── FileExistsError
│   │   ├── FileNotFoundError
│   │   ├── IsADirectoryError
│   │   ├── NotADirectoryError
│   │   ├── PermissionError
│   │   └── TimeoutError
│   ├── ReferenceError
│   ├── RuntimeError
│   │   ├── NotImplementedError
│   │   └── RecursionError
│   ├── SyntaxError
│   │   └── IndentationError
│   │       └── TabError
│   ├── TypeError
│   ├── ValueError
│   │   └── UnicodeError
│   └── Warning
```

## ■ 파이썬의 Exception 종류 - 1

| 예외 클래스              | 의미 또는 예외 발생 원인                           |
|---------------------|------------------------------------------|
| BaseException       | 모든 예외의 최상위 예외                            |
| SystemExit          | 프로그램을 종료하는 명령이 실행되었을 때                   |
| KeyboardInterrupt   | Control-C 키가 입력되었을 때                     |
| Exception           | (시스템 종료를 제외한) 대부분의 예외의 상위 예외             |
| ArithmeticError     | 수의 연산과 관련된 문제                            |
| ZeroDivisionError   | 수를 0으로 나누려 할 때                           |
| AssertionError      | assert 문( <b>9.4.6 assert 문으로 검증하기</b> ) |
| AttributeError      | (모듈, 클래스, 인스턴스에서) 잘못된 속성을 가리킬 때          |
| EOFError            | (파일, 스트림 등에서) 읽어들이 데이터가 더이상 없을 때         |
| ImportError         | 모듈을 임포트할 수 없을 때                          |
| ModuleNotFoundError | 임포트할 모듈을 찾을 수 없을 때                       |
| LookupError         | (시퀀스, 매핑에서) 잘못된 인덱스, 키를 인덱싱할 때           |
| IndexError          | (시퀀스에서) 잘못된 인덱스를 인덱싱할 때                  |
| KeyError            | (매핑에서) 잘못된 키를 인덱싱할 때                     |
| NameError           | 잘못된 변수(이름)를 가리킬 때                        |

## ■ 파이썬의 Exception 종류 - 2

| 예외 클래스              | 의미 또는 예외 발생 원인                        |
|---------------------|---------------------------------------|
| OSError             | 컴퓨터 시스템(운영 체제)의 동작과 관련된 다양한 문제        |
| ChildProcessError   | 하위 프로세스(프로그램이 실행한 별도의 프로그램)에서 오류 발생   |
| FileExistsError     | 이미 존재하는 파일/디렉토리를 새로 생성하려 할 때          |
| FileNotFoundError   | 존재하지 않는 파일/디렉토리에 접근하려 할 때             |
| IsADirectoryError   | 파일을 위한 명령을 디렉토리에 실행할 때                |
| NotADirectoryError  | 디렉토리를 위한 명령을 파일에 실행할 때                |
| PermissionError     | 명령을 실행할 권한이 없을 때                      |
| TimeoutError        | 명령이 수행되는 시간이 시스템이 허용한 기준을 초과했을 때      |
| RuntimeError        | 다른 분류에 속하지 않는 실행시간 오류                 |
| NotImplementedError | 내용 없는 메서드가 호출되었을 때                    |
| RecursionError      | 함수의 재귀 호출 단계가 허용한 깊이를 초과했을 때          |
| SyntaxError         | 구문 오류                                 |
| IndentationError    | 들여쓰기가 잘못되었을 때                         |
| TabError            | 들여쓰기에 탭과 스페이스를 번갈아가며 사용했을 때           |
| TypeError           | 연산/함수가 계산해야 할 데이터가 잘못된 유형일 때          |
| ValueError          | 연산/함수가 계산해야 할 데이터가 유형은 올바르나 값이 부적절할 때 |
| UnicodeError        | 유니코드와 관련된 오류                          |
| Warning             | 심각한 오류는 아니나 주의가 필요한 사항에 관한 경고         |



# 예외 처리 구문

- 파이썬에서 예외 처리는 예외가 발생 시에 프로그램이 비정상적으로 종료하는 하는 것을 방지하고 예외에 대한 알맞은

try:

    코드 블록

except [예외\_타입 [ as 예외\_변수]]

    예외 처리 코드

[else:

    예외가 발생하지 않은 경우 수행할 코드

finally:

    예외가 발생하든 하지 않든 try 블록 이후 수행할 코드]

# 예외 처리 구문

## ■ except 문

- 예외처리 방식에 따라 다음의 세 가지 방식으로 작성할 수 있다.
- 특정 타입의 예외를 처리할 경우
  - except 예외\_타입:
- 특정 타입의 예외 객체를 예외\_변수로 받아서 예외 처리에 사용할 경우
  - except 예외\_타입 as 예외\_변수:
- 모든 타입의 예외를 처리할 경우
  - except:

# 예외 처리 구문

## ■ except 문

```
def get(key, dataset):  
    try:  
        value = dataset[key]  
    except IndexError: # 인덱스가 잘못된 예외  
        return None  
    except KeyError: # 키가 잘못된 예외  
        return None  
    else:  
        return value  
  
print(get(3, (1, 2, 3))) # 범위를 벗어난 인덱스를 인덱싱  
print(get('age', {'name': '홍길동', 'sex': True})) # 사전에 없는 키 인덱싱
```

# 예외 처리 구문

## ■ 여러 오류 동일하게 처리하기

```
def get(key, dataset):  
    try:  
        value = dataset[key]  
    except (IndexError, KeyError): # 두 오류 동일 처리처리  
        return None  
    else:  
        return value  
  
print(get(3, (1, 2, 3))) # 범위를 벗어난 인덱스를 인덱싱  
print(get('age', {'name': '홍길동', 'sex': True})) # 사전에 없는 키 인덱싱
```

# 예외 처리 구문

## ■ 모든 오류 처리하기

```
def get(key, dataset):
```

```
    try:
```

```
        value = dataset[key]
```

```
#    except : 전체 오류 처리
```

```
    except BaseException : # 최상위 오류 처리
```

```
        return None
```

```
    else:
```

```
        return value
```

```
print(get(3, (1, 2, 3))) # 범위를 벗어난 인덱스를 인덱싱
```

```
print(get('age', {'name': '홍길동', 'sex': True})) # 사전에 없는 키 인덱싱
```

# 예외 처리 구문

## ■ 오류 종류를 전달 받아서 처리하기

**try:**

```
a = [1,2]
print(a[3])
4/0
print("end")
except ZeroDivisionError as e:
    print(e)
except IndexError as e:
    print(e)
```

**try:**

```
a = [1,2]
print(a[3])
4/0
print("end")
except (ZeroDivisionError, IndexError) as e: # 위소스와 동일하게 처리됨
    print(e)
```

# 예외 처리 구문

## ■ 오류 발생시키기

오류를 일부러 발생시키기 위해서 사용되는 구문은 `raise` 이다.

```
def myfun( jum ) :  
    if 0 <= jum <= 100 :  
        print(jum, "은 정상적인 점수입니다.")  
    else :  
        raise ValueError(jum, "정상적인 점수가 아닙니다")  
  
try :  
    jum = 100  
    myfun(jum)  
    jum = 200  
    myfun(jum)  
    print("end")  
except ValueError as ve:  
    print (ve.args[0], ve.args[1])  
  
print("main end")
```

# 예외 처리 구문

## ■ 오류 발생시키기

상속에서 하위 메소드에서 강제로 구현하지 않으면 오류 생기게 하는 오류 발생.

```
class Bird:
    def fly(self):
        raise NotImplementedError
```

```
class Eagle(Bird):
    pass
```

```
# class Eagle(Bird):
#     def fly(self):
#         print("very fast")
```

```
eagle = Eagle()
eagle.fly()
```



# 예외 처리 구문

## ■ 사용자 정의 예외 구현

- **Exception**을 상속 받아서 예외 클래스를 생성한다.
- **raise** 구문을 활용하여 사용자 정의 예제를 발생시킬 수 있다.

```
class AgeException(Exception) :
    def __init__(self, msg):
        self._message = msg
    def input_age() :
        age = int(input("나이를 입력하세요 "))
        if age < 0 :
            raise AgeException("나이는 음수 나이가 존재하지 않습니다")
        elif age >= 200 :
            raise AgeException("나이가 %d는 유효한 나이가 아닙니다" % age)
        else:
            return age
try:
    age = input_age()
except AgeException as e :
    print("age 오류 :", e.args[0])
else:
    print("입력받은 나이는 ", age , "입니다")
```

# 예외 처리 구문

```
def divide(m, n):  
    try:  
        result = m / n  
    except ZeroDivisionError:  
        print("0으로 나눌 수 없습니다")  
    except:  
        print("all error")  
    else:  
        return result  
    finally:  
        print("나눗셈 연산입니다.")
```

```
if __name__ == "__main__":  
    res = divide(3, 2)  
    print(res)  
    print()  
  
    res = divide(3, 0)  
    print(res)  
    print()  
  
    res = divide(None, 2)  
    print(res)
```



# Python Programming

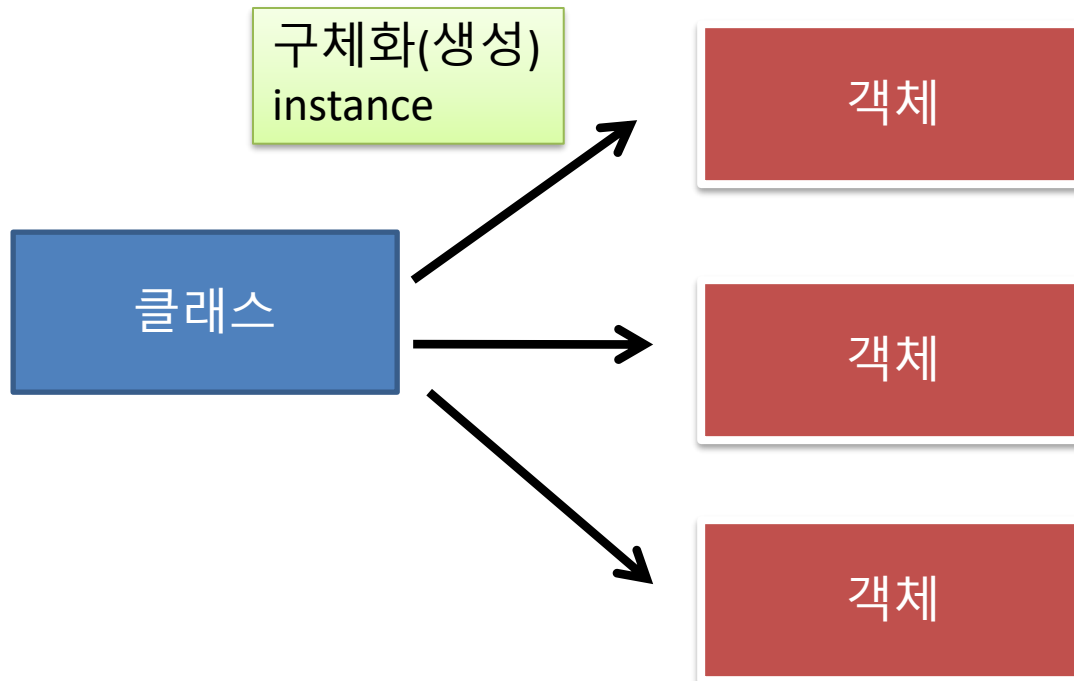
객체지향프로그래밍

# 객체지향프로그래밍

- 객체지향
- Self
- 모듈과 클래스
- 클래스와 데이터 타입
- 캡슐화(encapsulation)와 접근제한
- Property
- 상속 및 오버라이딩
- 클래스 속성
- 특별메소드
- 패키지

## ■ 객체 지향

- 객체는 속성과 행위(메서드, 기능)를 가지는 대상체이다.
- 속성은 객체가 가지는 값이며 행위는 객체가 수행할 수 있는 기능을 말한다.



## ■ 파이썬의 클래스.

```
class 클래스이름 :  
    클래스 본체
```

```
클래스변수 = 클래스이름()
```

```
class Car :  
    pass
```

```
car1 = Car()  
print(car1)
```

| 구분     | 이름         | 역할        |
|--------|------------|-----------|
| 클래스 이름 | Car        | 자동차 클래스이름 |
| 속성     | _speed     | 차량의 속도변수  |
| 기능     | get_speed  | 속도값 얻어오기  |
|        | start      | 출발        |
|        | stop       | 멈춤        |
|        | accelerate | 속도를 높이다   |

- self
  - 객체 자신을 가리키는 특수한 키워드
  - 다른 언어의 this 키워드와 유사한 개념을 가짐

```
class Sam (object):  
    name = "korea"  
    def get_name(self):  
        return self.name
```

```
    def getInfo(self):  
        self.get_name()
```

```
s1 = Sam()  
s2 = Sam()  
s3 = Sam()
```

```
s3.name = "japan"  
print(s1.name)  
print(s2.name)  
print(s3.get_name())
```

- `__init__(self)`
  - 객체가 생성될 때 마다 호출되는 특수 메소드
  - 생성자 함수라고도 한다

```
class Student :  
    def __init__(self,name="홍길동",age=20):  
        self._name = name  
        self._age = age  
        print(name, "student 객체가 생성됨")  
  
    def showInfo(self):  
        print(self._name,self._age)  
  
s1 = Student()  
s2 = Student("장나라")  
s3 = Student(age=30) # s3 = Student(30)은 어떤의미  
s4 = Student("임꺽정",40)  
  
s1.showInfo()  
s2.showInfo()  
s3.showInfo()  
s4.showInfo()
```



## ■ 모듈

- 함수 또는 클래스를 포함하고 있는 파이썬 파일
- import 구문으로 삽입하여 사용한다.

```
# sam.py
def aa():
    print("aa")

class Car:
    def __init__(self):
        self._name = "홍길동"
    def show_info(self):
        print(self._name)
```

```
# sam1.py
import sam

sam.aa()
s = sam.Car()
s.show_info()
```

## ■ 모듈

- 함수 또는 클래스를 포함하고 있는 파이썬 파일
- import 구문으로 삽입하여 사용한다.

```
# vehicle.py - 1
```

```
def change_km_to_mile(km):  
    return km * 0.621371
```

```
def change_mile_to_km(mile):  
    return mile * 1.609344
```

```
class Car:
```

```
    def __init__(self):  
        self._speed = 0
```

```
    def get_speed(self):  
        return self._speed
```

```
    def start(self):
```

```
        self._speed = 20
```

```
    def accelerate(self):
```

```
        self._speed = self._speed + 30
```

```
    def stop(self):
```

```
        self._speed = 0
```

## ■ 모듈

- 함수 또는 클래스를 포함하고 있는 파이썬 파일
- import 구문으로 삽입하여 사용한다.

```
# vehicle.py - 2
```

```
class Truck:
```

```
    def __init__(self):
```

```
        self._speed = 0
```

```
    def get_speed(self):
```

```
        return self._speed
```

```
    def start(self):
```

```
        self._speed = 10
```

```
    def accelerate(self):
```

```
        self._speed = self._speed + 20
```

```
    def stop(self):
```

```
        self._speed = 0
```

## ■ 모듈

- 함수 또는 클래스를 포함하고 있는 파이썬 파일
- import 구문으로 삽입하여 사용한다.

```
import vehicle      # vehicle 모듈 import
```

```
if __name__ == "__main__":  
    my_car = vehicle.Car( )  
    my_car.start( )  
    my_car.accelerate( )  
    speed_mile = vehicle.change_km_to_mile(my_car.get_speed( ))  
    print("속도:", speed_mile, "mile")  
    my_car.stop( )
```

```
from vehicle import Car, change_km_to_mile
```

```
# from vehicle import *
```

```
if __name__ == "__main__":  
    my_car = Car( )  
    my_car.start( )  
    my_car.accelerate( )  
    speed_mile = change_km_to_mile(my_car.get_speed( ))  
    print("속도:", speed_mile, "mile")  
    my_car.stop( )
```

## ■ 모듈

- 객체 변수가 실제 클래스로 생성된 것인가 부울린으로 반환하는 함수
- Instance(객체변수, 클래스이름)

```
print(type(1))
print(isinstance(1, int))
print(isinstance(1.0, int))
print(type("kor"))
print(isinstance("kor",str))
class Sam1:
    pass
s1 = Sam1()
print("s1", isinstance(s1, Sam1))
print(isinstance("kor",object))
print(isinstance({}, dict))
```

## ■ 캡슐화

- 변수, 또는 메소드 앞에 밑줄(\_)를 붙여서 구분함
- 밑줄이 없으면 공개 모드로 접근할 수 있음
- 밑줄이 하나 있으면 보호모드로 객체 외부에서는 접근하지 말아야 한다.(접근은 됨, 권장사항)
- 밑줄이 두개 있으면 비공개모드로 외부에서 접근이 되지 않는다.

```
class Car:
```

```
    def __init__(self):
```

```
        self.price = 2000
```

```
        self._speed = 0
```

```
        self.__color = "red"
```

```
if __name__ == "__main__":
```

```
    my_car = Car( )
```

```
    print(my_car.price)
```

```
    print(my_car._speed)
```

```
    # print(my_car.__color)
```

- 객체의 속성을 읽거나 설정하는데 사용되는 변수
- 접근하는 것은 외부에서는 변수를 직접적으로 접근하는 것으로 보이나 내부적으로는 메소드로 접근할 수 있도록 설정된 값
- 방법 - 1
  - `@property` : 어노테이션을 이용하여 읽기속성 접근 가능(get메소드 구현)
  - `@메소드이름.setter` : 어노테이션을 이용하여 쓰기속성값을 사용 가능(set메소드 구현)

```
@property
def price(self):
    return self._price
@price.setter
def price(self, value):
    self._price = value
@property
def speed(self):
    return self._speed
@speed.setter
def speed(self, value):
    self._speed = value
```

- 방법 - 2
  - 파이썬 2.6버전 이후에서 지원되는 기능
  - property 클래스를 활용해서 설정
  - `price = property(get_price, set_price)`

```
def get_price(self):  
    return self._price
```

```
def set_price(self, value):  
    self._price = value
```

```
price = property(get_price, set_price)
```

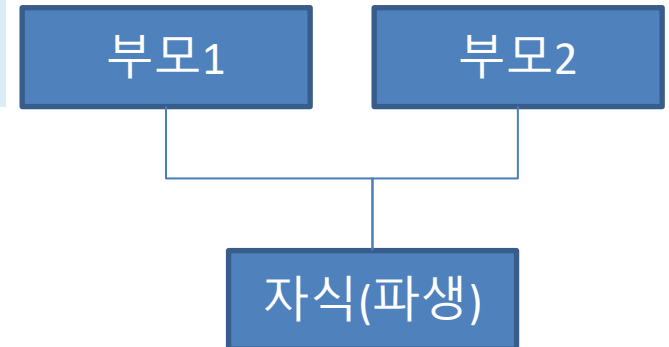


- 기존 클래스와 유사한 클래스를 생성하는 경우 상속하는 것이 유리(is 관계)
- 부모 클래스의 속성과 메소드를 자식 객체가 활용 가능
- 파이썬은 기본적으로 최상위 클래스로 object 클래스를 상속받고 있음
- 파이썬은 다중 상속을 지원함

```
class 클래스이름(부모클래스):  
    클래스 본체 내용
```



```
class 클래스이름(부모클래스1,부모클래스2):  
    클래스 본체 내용
```



```
class My:
    def __init__(self):
        self._name = "korea"
        print("My")

    def get_name(self):
        print("self._name", self._name)
        return self._name

class SubMy(My):
    def __init__(self):
        super().__init__()
        print("SubMy")
    def sub_print(self):
        print("sub_print", self.get_name())

s = SubMy()
print(s._name)
print("메소드", s.get_name())
print("-----")
s.sub_print()
```

```
class SportCar(object):  
    def __init__(self):  
        self._speed = 0  
  
    def get_speed(self):  
        return self._speed  
  
    def start(self):  
        self._speed = 20  
  
    def accelerate(self):  
        self._speed = self._speed + 40  
  
    def turbocharge(self):  
        self._speed = self._speed + 70  
  
    def stop(self):  
        self._speed = 0
```

```
if __name__ == "__main__":  
    my_sportcar = SportCar()  
    my_sportcar.start()  
    print("속도:", my_sportcar.get_speed())  
    my_sportcar.accelerate()  
    print("속도:", my_sportcar.get_speed())  
    my_sportcar.turbocharge()  
    print("속도:", my_sportcar.get_speed())  
    my_sportcar.stop()
```

```
class Car(object):  
    def __init__(self):  
        self._speed = 0  
  
    @property  
    def speed(self):  
        return self._speed  
  
    def start(self):  
        self._speed = 20  
  
    def accelerate(self):  
        self._speed = self._speed + 30  
  
    def stop(self):  
        self._speed = 0
```

```
class SportCar(Car):  
    def __init__(self):  
        super().__init__(self)  
        self._color = "red"  
    def accelerate(self):  
        self._speed = self._speed + 40  
    def turbocharge(self):  
        self._speed = self._speed + 70  
    @property  
    def color(self):  
        return self._color  
  
if __name__ == "__main__":  
    my_sportcar = SportCar()  
    print("색상:", my_sportcar.color)  
    my_sportcar.start()  
    print("속도:", my_sportcar.speed)  
    my_sportcar.accelerate()  
    print("속도:", my_sportcar.speed)  
    my_sportcar.turbocharge()  
    print("속도:", my_sportcar.speed)  
    my_sportcar.stop()
```

- 부모클래스에서 정의된 메소드를 재정의해서 사용한다
- 서브 클래스에서 서로 다른 용도로 사용하기 위해서 사용됨

```
class A:  
    def func1(self):  
        print ("super")  
  
class SubA1(A):  
    def func1(self):  
        print ("SubA1")  
  
class SubA2(A):  
    def func1(self):  
        print ("SubA2")  
  
def printA(a):  
    a.func1()
```

```
a = A()  
a1 = SubA1()  
a2 = SubA2()  
# a.func1()  
# a1.func1()  
# a2.func1()  
printA(a)  
printA(a1)  
printA(a2)
```

# 오버라이딩(Overriding) - sample



*# ariforce.py 파일*

```
class AirForce(object):  
    def take_off(self):  
        pass  
  
    def fly(self):  
        pass  
  
    def attack(self):  
        pass  
  
    def land(self):  
        pass
```

*# bomber.py 파일*

```
from airforce import AirForce
```

```
class Bomber(AirForce):  
    def __init__(self, bomb_num):  
        self._bomb_num = bomb_num  
  
    def take_off(self):  
        print("폭격기 발진")  
  
    def fly(self):  
        print("폭격기 목적지로 출격")  
  
    def attack(self):  
        for i in range(self._bomb_num):  
            print("폭탄 투하")  
            self._bomb_num = self._bomb_num - 1  
  
    def land(self):  
        print("폭격기 착륙")
```

# 오버라이딩(Overriding) - sample



*# fighter.py 파일*

```
from airforce import AirForce
```

```
class Fighter(AirForce):
```

```
    def __init__(self, weapon_num):
        self._missile_num = weapon_num
```

```
    def take_off(self):
        print("전투기 발진")
```

```
    def fly(self):
        print("전투기가 목표지로 출격")
```

```
    def attack(self):
        for i in range(self._missile_num):
            print("미사일 발사")
            self._missile_num =
self._missile_num - 1
```

```
    def land(self):
        print("전투기 착륙")
```

*# sam.py 파일*

```
from fighter import Fighter
from bomber import Bomber
```

```
def war_game(airforce):
    airforce.take_off()
    airforce.fly()
    airforce.attack()
    airforce.land()
```

```
if __name__ == "__main__":
    f15 = Fighter(3)
    war_game(f15)
    print()
```

```
b29 = Bomber(3)
war_game(b29)
```

- 생성되는 객체마다 별도로 메모리가 할당되는 변수 => **인스턴스 속성**
- 모든 객체마다 같은 메모리를 참조하여 하나만 할당되는 변수 => **클래스 속성**

```
class STest:
    age = 10

s1 = STest()
s2 = STest()
s1.age = 30
s2.age = 50
# s2.sex = True
# print(s2.sex)
# print(s1.sex)
print(s1.age)
print(s2.age)
# print(STest.age)
# print(STest.age)
```



```
class Circle(object):
    PI = 3.14
    def __init__(self, radius):
        self._radius = radius
    @property
    def radius(self):
        return self._radius
    def get_area(self):
        area = Circle.PI * (self._radius ** 2)
        return round(area, 2)
    def get_circumference(self):
        circumference = 2 * Circle.PI * self._radius
        return round(circumference, 2)

if __name__ == "__main__":
    circle1 = Circle(3)
    print("원주율: ", Circle.PI)
    print("반지름: ", circle1.radius, "면적: ", circle1.get_area())
    print("반지름: ", circle1.radius, "둘레: ", circle1.get_circumference())
    circle2 = Circle(4)
    print("반지름: ", circle2.radius, "면적: ", circle2.get_area())
    print("반지름: ", circle2.radius, "둘레: ", circle2.get_circumference())
```

- 객체를 생성한 후 접근할 수 있는 메소드 => **인스턴스 메소드**
- 객체를 생성하지 않고도 사용할 수 있는 메소드 => **클래스 메소드**
  - 클래스 메소드에서는 인스턴스 속성을 접근할 수 없다.
  - 클래스 메소드는 첫번째인자 self를 사용하지 않는다.(대신 cls를 통한 클래스 인자 사용)
  - cls는 클래스 자체 객체를 가리킨다.
  - 클래스 메소드 선언시 @classmethod 데코레이터를 사용해야 함
- 객체를 생성하지 않고 접근할 수 있는 메소드 => **static 메소드**
  - 클래스 메소드와 같이 클래스 이름으로 접근한다
  - 메소드 선언 시 cls를 사용하지 않는다
  - 메소드 선언시 @staticmethod 데코레이터를 사용해야 함

**class Sam:**

data = "python"

**def** \_\_init\_\_(self):

self.\_data = "korea"

**def** instance\_method(self):

print("instance method", self.\_data)

@classmethod

**def** class\_method(cls):

print("class method", cls.data)

# print("class method", cls.\_data)

@staticmethod

**def** static\_method():

print("static method", Sam.data)

s = Sam()

s.instance\_method()

s.class\_method()

Sam.class\_method()

Sam.static\_method()

```
class CircleCalculator(object):
```

```
    __PI = 3.14
```

```
    @classmethod
```

```
    def calculate_area(cls, radius):
```

```
        area = cls.__PI * (radius ** 2)
```

```
        return round(area, 2)
```

```
    @classmethod
```

```
    def calculate_circumference(cls, radius):
```

```
        circumference = 2 * cls.__PI * radius
```

```
        return round(circumference, 2)
```

```
class CircleCalculator(object):
```

```
    @staticmethod
```

```
    def calculate_area(radius, pi):
```

```
        area = pi * (radius ** 2)
```

```
        return round(area, 2)
```

```
    @staticmethod
```

```
    def calculate_circumference(radius, pi):
```

```
        circumference = 2 * pi * radius
```

```
        return round(circumference, 2)
```

```
if __name__ == "__main__":
```

```
    print("반지름:", 3, "면적:", CircleCalculator.calculate_area(3, 3.14))
```

```
    print("반지름:", 3, "둘레:", CircleCalculator.calculate_circumference(3, 3.14))
```

- 특별한 연산 혹은 문법으로 자동 호출되는 메소드(==매직메소드)

## 기본 특별메소드

| 코드        | 파이썬 내부 호출    | 의미                            |
|-----------|--------------|-------------------------------|
| x = Sam() | x.__init__() | 객체를 하나 생성할 때 마다 실행하는 메소드      |
| repr(x)   | x.__repr__() | 객체의 공식적 표현을 문자열로 표현(내부-인터프리터) |
| str(x)    | x.__str__()  | 객체의 비공식적 표현을 문자열로 표현(일반)      |

```
a = "python is programming"
print(str(a))
# 'python is programming'
print(repr(a))
# "'python is programming'"
```

```
a = "python is programming"
a1 = str(a)
a2 = repr(a)

a3 = eval(a2)
# a4 = eval(a1)
print("end")
```

```
class Car(object):
    def __init__(self, name):
        self._name = name
    def __str__(self):
        return "Type: Car, Name: " + self._name
    def __repr__(self):
        return "Car('" + self._name + "')"

if __name__ == "__main__":
    my_car = Car("Sonata")
    my_car_string = str(my_car)
    print(my_car_string)
    print(my_car)
    my_car_repr = repr(my_car)
    print(my_car_repr)
    my_car1 = eval(my_car_repr)
    print(isinstance(my_car, Car))
    print(my_car is my_car1)
```

# Car 클래스 객체로 만들어졌는가?  
# 같은 객체인가

## 객체의 함수화 관련 메소드

| 코드        | 파이썬 내부 호출          | 의미                     |
|-----------|--------------------|------------------------|
| my_call() | my_call.__call__() | 객체 호출을 마치 함수처럼 호출하는 의미 |

```
import random
```

```
class Dice(object):
```

```
    def __init__(self, start, end):
```

```
        print("init")
```

```
        self._start = start
```

```
        self._end = end
```

```
    def __call__(self):
```

```
        return random.randint(self._start, self._end)
```

```
if __name__ == "__main__":
```

```
    dice = Dice(1, 6)
```

```
    print(dice( ))
```

```
    print(dice())
```

```
    print(dice())
```

```
    print(dice())
```

## 집합 메소드

| 코드                  | 파이썬 내부 호출                      | 의미                             |
|---------------------|--------------------------------|--------------------------------|
| <code>len(x)</code> | <code>x.__len__()</code>       | x 내부의 데이터 개수를 반환하는 함수 호출       |
| <code>d in x</code> | <code>x.__contains__(d)</code> | x 내부 데이터에 d가 존재하는가를 확인하는 함수 호출 |

```
class StudentScores(object):
```

```
    def __init__(self, data):
        self._data = data
```

```
    def __len__(self):
        return len(self._data)
```

```
    def __contains__(self, d):
        if d in self._data:
            return True
        else:
            return False
```

```
if __name__ == "__main__":
    student_scores = StudentScores([90, 95, 100])
    print(len(student_scores))
    print( 90 in student_scores)
    print( 80 in student_scores)
```

## 산술연산 및 관계 연산 메소드

| 코드                        | 파이썬 내부 호출                      | 의미                                                    |
|---------------------------|--------------------------------|-------------------------------------------------------|
| $x + y$                   | <code>x.__add__(y)</code>      | x 객체의 메소드에서 <code>__add__</code> 메소드를 실행(y 인자전달)      |
| $x - y$                   | <code>x.__sub__(y)</code>      | x 객체의 메소드에서 <code>__sub__</code> 메소드를 실행(y 인자전달)      |
| $x * y$                   | <code>x.__mul__(y)</code>      | x 객체의 메소드에서 <code>__mul__</code> 메소드를 실행(y 인자전달)      |
| $x / y$                   | <code>x.__truediv__(y)</code>  | x 객체의 메소드에서 <code>__truediv__</code> 메소드를 실행(y 인자전달)  |
| $x // y$                  | <code>x.__floordiv__(y)</code> | x 객체의 메소드에서 <code>__floordiv__</code> 메소드를 실행(y 인자전달) |
| $x \% y$                  | <code>x.__mod__(y)</code>      | x 객체의 메소드에서 <code>__mod__</code> 메소드를 실행(y 인자전달)      |
| $x ** y$                  | <code>x.__pow__(y)</code>      | x 객체의 메소드에서 <code>__pow__</code> 메소드를 실행(y 인자전달)      |
| <code>divide(x, y)</code> | <code>x.__divmod__(y)</code>   | x 객체의 메소드에서 <code>__divmod__</code> 메소드를 실행(y 인자전달)   |
| $x \& y$                  | <code>x.__and__(y)</code>      | x 객체의 메소드에서 <code>__and__</code> 메소드를 실행(y 인자전달)      |
| $x   y$                   | <code>x.__or__(y)</code>       | x 객체의 메소드에서 <code>__or__</code> 메소드를 실행(y 인자전달)       |
| $x \wedge y$              | <code>x.__xor__()</code>       | x 객체의 메소드에서 <code>__xor__</code> 메소드를 실행(y 인자전달)      |



## 산술연산 및 관계 연산 메소드-예제

```
class Sam(object):  
    def __init__(self, num):  
        self._num = num  
  
    def __add__(self, other):  
        return self._num + other._num  
  
    def __divmod__(self, other):  
        mok = self._num // other._num  
        nam = self._num % other._num  
        return mok, nam
```

```
s1 = Sam(7)  
s2 = Sam(3)  
# num = s1 + s2  
num = divmod(7, 3)  
num = divmod(s1, s2)  
print(num)
```

## 비교 관계 연산 메소드

| 코드                     | 파이썬 내부 호출                 | 의미                                               |
|------------------------|---------------------------|--------------------------------------------------|
| <code>x == y</code>    | <code>x.__eq__(y)</code>  | x 객체의 메소드에서 <code>__eq__</code> 메소드를 실행(y 인자전달)  |
| <code>x != y</code>    | <code>x.__ne__(y)</code>  | x 객체의 메소드에서 <code>__ne__</code> 메소드를 실행(y 인자전달)  |
| <code>x &lt; y</code>  | <code>x.__lt__(y)</code>  | x 객체의 메소드에서 <code>__lt__</code> 메소드를 실행(y 인자전달)  |
| <code>x &gt; y</code>  | <code>x.__gt__(y)</code>  | x 객체의 메소드에서 <code>__gt__</code> 메소드를 실행(y 인자전달)  |
| <code>x &lt;= y</code> | <code>x.__le__(y)</code>  | x 객체의 메소드에서 <code>__le__</code> 메소드를 실행(y 인자전달)  |
| <code>x &gt;= y</code> | <code>x.__ge__(y)</code>  | x 객체의 메소드에서 <code>__ge__</code> 메소드를 실행(y 인자전달)  |
| <code>if x:</code>     | <code>x.__bool__()</code> | x 객체의 메소드에서 <code>__bool__</code> 메소드를 실행 부울린 반환 |

## 비교 관계 연산 메소드-예제

```
class Sam(object):
    def __init__(self, num):
        self._num = num

    # def __eq__(self, other):
    #     return self._num == other._num

    def __bool__(self):
        return self._num > 5

s1 = Sam(7)
s2 = Sam(3)
s3 = Sam(7)
print(s1 == s3)
if s2:
    print("크다")
else:
    print("작다")
```

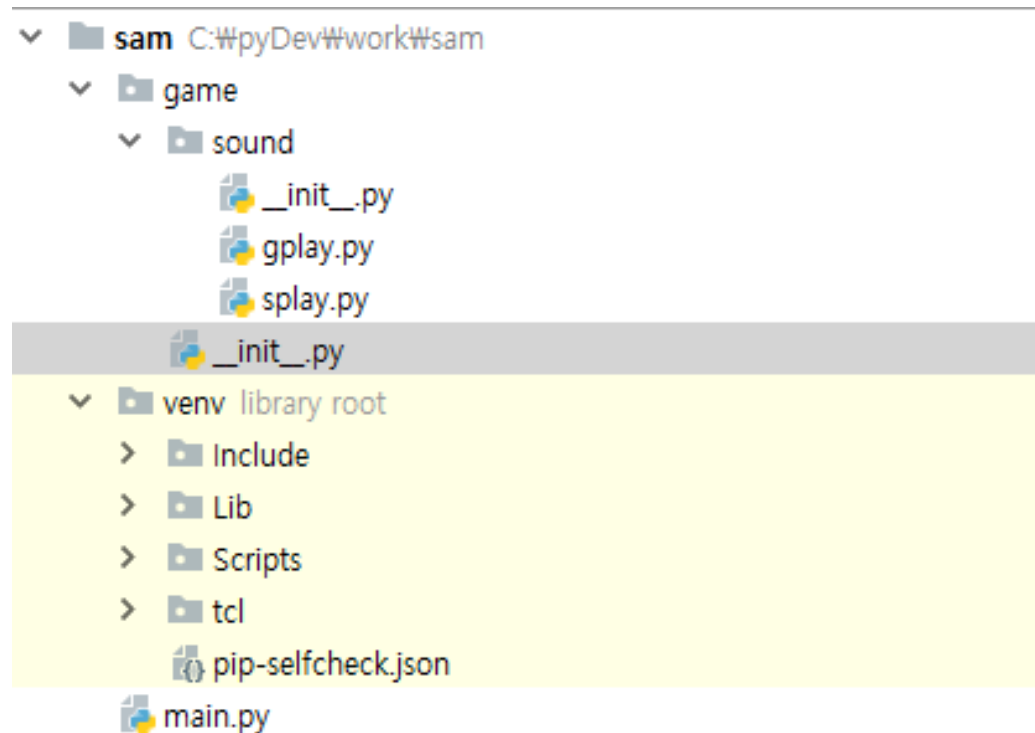
## 비교 관계 연산 메소드-예제

```
class StudentScores(object):  
    def __init__(self, data):  
        self._data = data  
        self._total = sum(self._data)  
  
    def __eq__(self, y):  
        return self._total == y._total  
  
    def __lt__(self, y):  
        return self._total < y._total  
  
    def __le__(self, y):  
        return self._total <= y._total
```

```
if __name__ == "__main__":  
    student_a = StudentScores([90, 95, 100])  
    student_b = StudentScores([80, 85, 90])  
  
    print( student_a == student_b)  
    print( student_a < student_b)
```

## ■ 패키지

- 관련된 모듈은 모아서 관리하는 개념
- 전체 프로젝트 진행 시 관련된 파이썬 파일을 폴더 별 관리가 가능함
- 패키지 폴더는 `__init__.py` 파일이 폴더에 있어야 함
- `Import *` 사용하는 경우 `__all__` 속성값에 파이썬 파일명 지정
- new - 패키지 생성



- sound 폴더에 파이썬 파일 생성

# slay.py 파일

```
def sound_play():  
    print("sound play")
```

- main.py에서 sound\_play 함수 호출

# main.py 파일

```
import game.sound.splay
```

```
game.sound.splay.sound_play()
```

# main.py 파일

```
from game.sound.splay import sound_play
```

```
# from game.sound.splay import *
```

```
sound_play()
```

- 하위 모듈의 삽입 접근 오류

```
# main.py 파일
```

```
from game.sound import *
```

```
splay.sound_play()
```

- sound 폴더 안에 있는 \_\_init\_\_.py 파일에 아래 내용 삽입

```
# game/sound/__init__.py 파일
```

```
__all__ = ["splay"]
```

- /game/sound/gplay.py 파일 추가

**# gplay.py 파일**

```
def effect_play(x):  
    print("effect", x)
```

- sound 폴더 안에 있는 \_\_init\_\_.py 파일에 추가 내용 삽입

**# game/sound/\_\_init\_\_.py 파일**

```
__all__ = ["splay", "gplay"]
```

- main.py 파일에서 gplay.py 파일에 있는 내용 실행

**# main.py 파일**

```
from game.sound import *
```

```
splay.sound_play()  
gplay.effect_play(3)
```





# Python Programming

## 객체지향프로그래밍

- 유용한 함수
- 유용한 모듈
- 파일처리
- JSON
- 정규표현식

- `abs(v)` : 어떤 숫자를 입력으로 받았을 때, 그 숫자의 절대값을 돌려주는 함수
- `chr(v)` : 아스키(ASCII) 코드값을 입력 받아 그 코드에 해당하는 문자를 출력하는 함수
- `ord(v)` : 문자자의 아스키 코드값을 리턴하는 함수
- `divmod(a, b)` : 2개의 숫자를 입력으로 받는다. 그리고 a를 b로 나눈 몫과 나머지를 튜플 형태로 리턴하는 함수
- `enumerate(list)` : 순서가 있는 자료형(리스트, 튜플, 문자열)을 입력으로 받아 인덱스 값을 포함하는 enumerate 객체를 리턴하는 함수

```
for i, name in enumerate(['python', 'java', 'Node', 'Scala']):  
    print(i, name)
```

- `eval(expression)` : 실행 가능한 문자열(`1+2`, `'hi' + 'a'` 같은 것)을 입력으로 받아 문자열을 실행한 결과값을 리턴하는 함수

```
print(eval('1+2'))  
print(eval("'py' + 'thon'"))  
print(eval('divmod(4, 3)'))  
print(eval("'python'"))
```

- `pow(x, y)` : x의 y 제곱한 결과값을 리턴하는 함수
- `sorted(iterable)` : 입력값을 정렬한 후 그 결과를 리스트로 리턴하는 함수
- `zip(iterable*)` : 동일한 개수로 이루어진 자료형을 묶어 주는 역할을 하는 함수

```
print(list(zip(["kor", "eng", "mat"], [40, 50, 60])))
```

## OS

- Os(운영체제)가 제공하는 정보나 기능을 구현할 수 있는 모듈
- 파일 입출력 및 관리를 할 수 있는 기능 제공(open, fileinput, shutil 등)

```
import os
print(os.name) # 운영체제의 이름 출력
print(os.getcwd()) # 실행중인 현재 디렉토리
os.chdir("..") # 상위 디렉토리로 이동
print(os.getcwd()) # 실행중인 현재 디렉토리
path = os.path.join(os.getcwd(),"down") # 현재 디렉토리 밑에 하위 디렉토리 연결
os.mkdir(path,0o777) # 지정된 경로로 하위 디렉토리 생성하기
# print(path)
print(os.listdir(os.getcwd())) # 지정된 폴더의 하위 목록얻어오기
print(os.getcwd())
os.rmdir(path = os.path.join(os.getcwd(),"down")) # 지정된 폴더 삭제하기
print(os.listdir()) # 지정된 폴더의 하위 목록얻어오기
f = open("sam.txt", 'w') # 파일 생성
f.close()
print(os.listdir()) # 지정된 폴더의 하위 목록얻어오기
os.remove(os.path.join(os.getcwd(),"sam.txt")) # 지정된 파일 삭제하기
print(os.listdir()) # 지정된 폴더의 하위 목록얻어오기
os.system("dir")
```

## ■ sys

- 파이썬 인터프리터와 관련된 정보를 얻을 수 있는 모듈
- 실행 인자 값 및 강제 종료 실행

```
import sys
print(dir(sys))           # sys 라이브러리의 모든 함수
print(sys.version)        # 파이썬 인터프리터 버전
print(sys.maxsize)        # int의 최대값
print(len(sys.argv))      # 파이썬 실행시 전달받은 인자리스트
# python sam.py 1 "korea Japan" 333 444
print(sys.argv)
print(sys.exit(0))        # 파이썬 강제 종료
```

## ■ math

- 수학과 관련된 모듈

```
import math
print(math.ceil(3.2))     # 강제 올림
print(math.sin(math.radians(30))) # sin 30도 값
print(math.pi)           # pi 값
print(math.sqrt(9))       # 9의 제곱근 값
```

## ■ time, datetime, calendar

- 날짜 및 시간과 관련된 다양한 함수를 지원하는 모듈

| 포맷코드 | 설명                  | 예                    |
|------|---------------------|----------------------|
| %a   | 요일 줄임말              | Mon                  |
| %A   | 요일                  | Monday               |
| %b   | 달 줄임말               | Jan                  |
| %B   | 달                   | January              |
| %c   | 날짜와 시간을 출력함         | 06/01/01<br>17:22:21 |
| %d   | 날(day)              | [00,31]              |
| %H   | 시간(hour)-24시간 출력 형태 | [00,23]              |
| %I   | 시간(hour)-12시간 출력 형태 | [01,12]              |
| %j   | 1년 중 누적 날짜          | [001,366]            |
| %m   | 달                   | [01,12]              |
| %M   | 분                   | [01,59]              |
| %p   | AM or PM            | AM                   |
| %S   | 초                   | [00,61]              |

| 포맷코드 | 설명                    | 예          |
|------|-----------------------|------------|
| %U   | 1년 중 누적 주-일요일을 시작으로   | [00,53]    |
| %w   | 숫자로 된 요일              | [0(일요일),6] |
| %W   | 1년 중 누적 주-월요일을 시작으로   | [00,53]    |
| %x   | 현재 설정된 로케일에 기반한 날짜 출력 | 06/01/01   |
| %X   | 현재 설정된 로케일에 기반한 시간 출력 | 17:22:21   |
| %Y   | 년도 출력                 | 2001       |
| %Z   | 시간대 출력                | 대한민국 표준시   |
| %%   | 문자                    | %          |
| %y   | 세기부분을 제외한 년도 출력       | 01         |

## ■ time, datetime, calendar

```
import time
import calendar
import datetime

print(time.time())                # 1970.1.1.0.0.0초 기준으로 밀리초 반환
print(time.localtime(time.time())) # 지정된 시간을 년월일 시분초 관리객체로 반환
print(time.localtime(time.time()).tm_hour) # 특정 날짜의 시간을 반환
print(time.asctime(time.localtime(time.time()))) # 특정 날짜를 형식적으로 출력
print(time.ctime())                # 시스템 현재 날짜를 형식적으로 출력
# time.strftime('출력할 형식 포맷 코드', time.localtime(time.time()))
print(time.strftime('%x', time.localtime(time.time()))) # 날짜
print(time.strftime('%X', time.localtime(time.time()))) # 시간
time.sleep(1)                # 실행을 일시 정지하며 인자는 초단위
now = datetime.date.today()  # 시스템의 오늘 날짜 얻기
dday = datetime.date(2018,12,4) # 특정한 년월일 설정
resday = dday - now
print("남은 날 ",resday.days) # dday 날짜수 구하기
print(calendar.weekday(2018,12,4)) #요일 반환
# 월요일은 0, 화요일은 1, 수요일은 2, 목요일은 3, 금요일은 4, 토요일은 5, 일요일은 6
print(calendar.monthrange(2018,5)) # 지정된 날의 시작요일과 마지막 날(튜플로반환)
```

## ■ collections 모듈

- 파이썬 dic 타입에서는 입력된 순서로 반환된다는 것을 보장할 수 없다.
- Collections.OrderedDic() 를 활용하면 입력된 순서로 반환하는 것을 보장함
- dic 클래스는 key 입력순서가 달라도 같은 것으로 보나 OrderedDic은 입력 순서도 같아야 같은 객체로 봄

```
d = {}  
d['first'] = "kim"  
d['second'] = "jae"  
d['third'] = "ung"  
for k, v in d.items():  
    print(k, v)
```

```
import collections  
d = collections.OrderedDict()  
d['first'] = "kim"  
d['second'] = "jae"  
d['third'] = "ung"  
  
for k, v in d.items():  
    print(k, v)
```



## ■ 파일 open 및 쓰기

■ `fs = open(파일이름, 파일열기모드, [encoding=인코딩방식])`

| 파일열기모드 | 설명                                 |
|--------|------------------------------------|
| r      | 읽기모드 - 파일을 읽기만 할 때 사용(기본)          |
| w      | 쓰기모드 - 파일에 내용을 쓸 때 사용              |
| a      | 추가모드 - 파일의 마지막에 새로운 내용을 추가 시킬 때 사용 |
| 'b'    | 이진모드로 이미지나 동영상 파일 저장할 때 사용         |
| 't'    | 텍스트 모드로 문자열 출력하는 경우 사용(기본)         |

```
if __name__ == "__main__":  
    fp = open("c:\\text.txt", "wt", encoding="utf-8")  
    fp.write("%10d\n" % 1)  
    fp.write("%.2f\n" % 3.14)  
    fp.write("Hello World\n")  
    fp.write("안녕 파이썬!")  
    fp.close()
```

## ■ 파일 open 및 읽기

- `fs.readline()` : 한 줄씩 읽어서 문자열로 반환
- `fs.readlines()` : 전체를 한번에 읽어서 문자열로 리스트로 반환
- `fs.read()` : 전체를 읽어서 하나의 문자열로 반환

```
if __name__ == "__main__":  
    fp = open("c:\\text.txt", "rt", encoding="utf-8")  
    line = fp.readline()  
    print(line.strip())  
    line = fp.readline()  
    print(line.strip())  
    line = fp.readline()  
    print(line.strip())  
    line = fp.readline()  
    print(line.strip())  
    # lines = fp.readlines() #전체 읽어서 문자열 리스트 반환  
    # print(lines)  
    # contents = fp.read() #전체 읽어서 하나의 문자열 반환  
    # print(contents)  
    fp.close()
```

## ■ 파일 open 및 읽기

- `fs.readline()` : 한 줄씩 읽어서 문자열로 반환
- `fs.readlines()` : 전체를 한번에 읽어서 문자열로 리스트로 반환
- `fs.read()` : 전체를 읽어서 하나의 문자열로 반환

```
if __name__ == "__main__":  
    fp = open("c:\\text.txt", "rt", encoding="utf-8")  
    line = fp.readline()  
    print(line.strip())  
    line = fp.readline()  
    print(line.strip())  
    line = fp.readline()  
    print(line.strip())  
    line = fp.readline()  
    print(line.strip())  
    # lines = fp.readlines() #전체 읽어서 문자열 리스트 반환  
    # print(lines)  
    # contents = fp.read() #전체 읽어서 하나의 문자열 반환  
    # print(contents)  
    fp.close()
```

- with 구문 활용
  - 파일 작업은 반드시 닫기 작업을 진행함
  - With 구문을 작업블럭을 보화하고 자동으로 닫아주는 역할 진행

```
if __name__ == "__main__":
```

```
    with open("text.txt", "rt", encoding="utf-8") as fp:
```

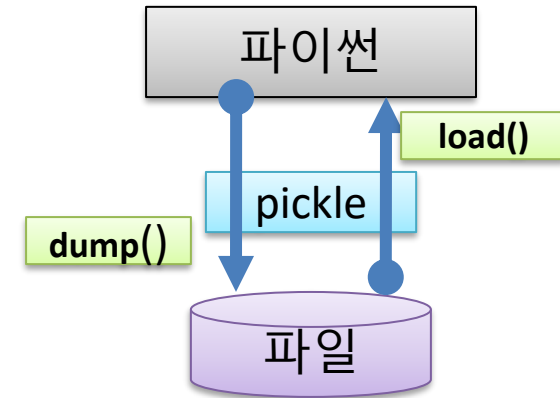
```
        lines = fp.readlines()
```

```
        for line in lines:
```

```
            print(line.strip())
```

```
print("Done!")
```

- 직렬화/역직렬화(Serialization)
  - 파이썬 자체 데이터 타입의 입출력을 위해서는 바이트 단위 입출력 필요
  - 파이썬 타입을 기본 byte 단위로 만들기(직렬화)
  - Btye 데이터를 파이썬 타입으로 변환 (역직렬화)
  - pickle 모듈 활용



```
import pickle
```

```
# pickle.dump(파이썬 데이터 타입", 파일객체) 파일에 저장
```

```
if __name__ == "__main__":
```

```
    with open("binary.dat", "wb") as fp:
```

```
        pickle.dump(1, fp)
```

```
        pickle.dump(3.14, fp)
```

```
        pickle.dump("Hello World", fp)
```

```
        pickle.dump("안녕 파이썬!", fp)
```

```
        pickle.dump([1, 2, 3], fp)
```

```
        pickle.dump((1, 2, 3), fp)
```

```
        pickle.dump({"line": 0, "rectangle":1, "triangle": 2}, fp)
```

```
    print("save end")
```

- 직렬화/역직렬화(Serialization)

[illegible]

## ■ 직렬화/역직렬화(Serialization) - 예제

```
import pickle
```

```
class Person:
```

```
    def __init__(self, name, age,sex):
```

```
        self._name = name
```

```
        self._age = age
```

```
        self._sex = sex
```

```
    def __str__(self): # 파이썬 3.6 이상에서 사용할 수 있는 f 문자열 포매팅 활용
```

```
        return f'이름: {self._name}, 나이:{self._age}, 성별: {self._sex} 입니다'
```

```
p1 = Person('홍길동', 23, True)
```

```
with open('person.pickle', 'wb') as f:
```

```
    pickle.dump(p1, f) # 파일로 저장함
```

```
print(p1) # Person {_name: 홍길동, _age: 23, _sex: True}
```

```
with open('person.pickle', 'rb') as f:
```

```
    p2 = pickle.load(f) # 파일에서 읽어옴
```

```
print(p2) # Person {_name: 홍길동, _age: 23, _sex: True}
```

## ■ 아래 데이터를 읽어드려서 자료를 처리하세요

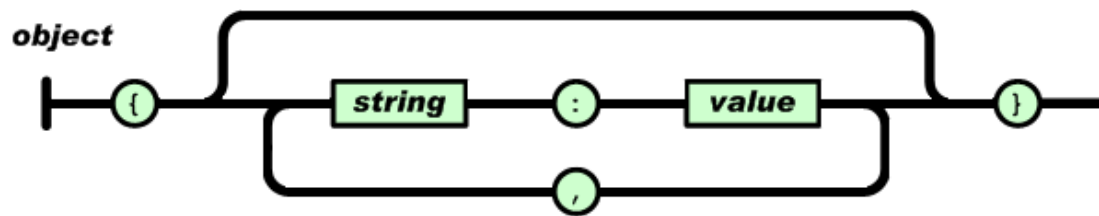
```
홍길동 30 1 70 80 90
임꺽정 20 2 90 90 70
장나라 30 1 80 80 30
홍명보 32 1 87 79 88
아이유 25 2 90 78 94
정해인 34 1 85 98 66
김제동 43 1 87 94 100
김호인 33 1 76 89 83
이순옥 34 2 89 64 81
```



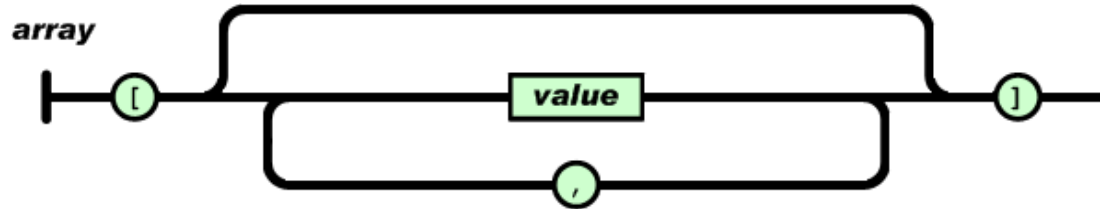
- JSON (JavaScript Object Notation)은 인터넷상에서 데이터를 주고 받을 때 사용하는 경량의 DATA-교환 형식이다. 이 형식은 프로그램의 변수값을 표현하는데 적합하고, 사람이 읽고 쓰기에 용이하며, 기계가 분석하고 생성함에도 용이

JSON은 두개의 구조를 기본으로 두고 있다:

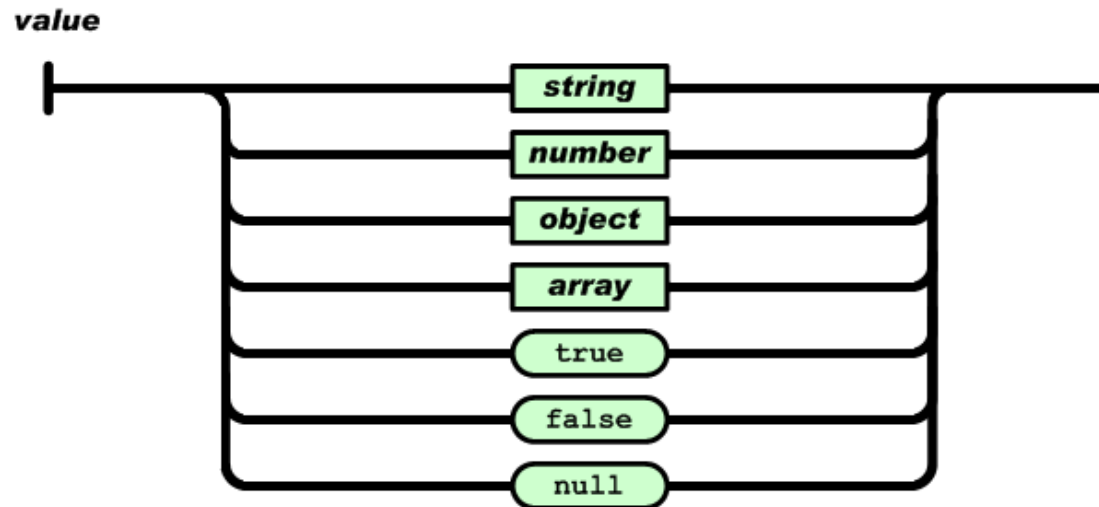
- name/value 형태의 쌍으로 collection 타입. 다양한 언어들에서, 이는 *object*, record, struct(구조체), dictionary, hash table, 키가 있는 list, 또는 연상배열로서 실현 되었다.
- 값들의 순서화된 리스트. 대부분의 언어들에서, 이는 *array*, vector, list, 또는 sequence로서 실현 되었다.



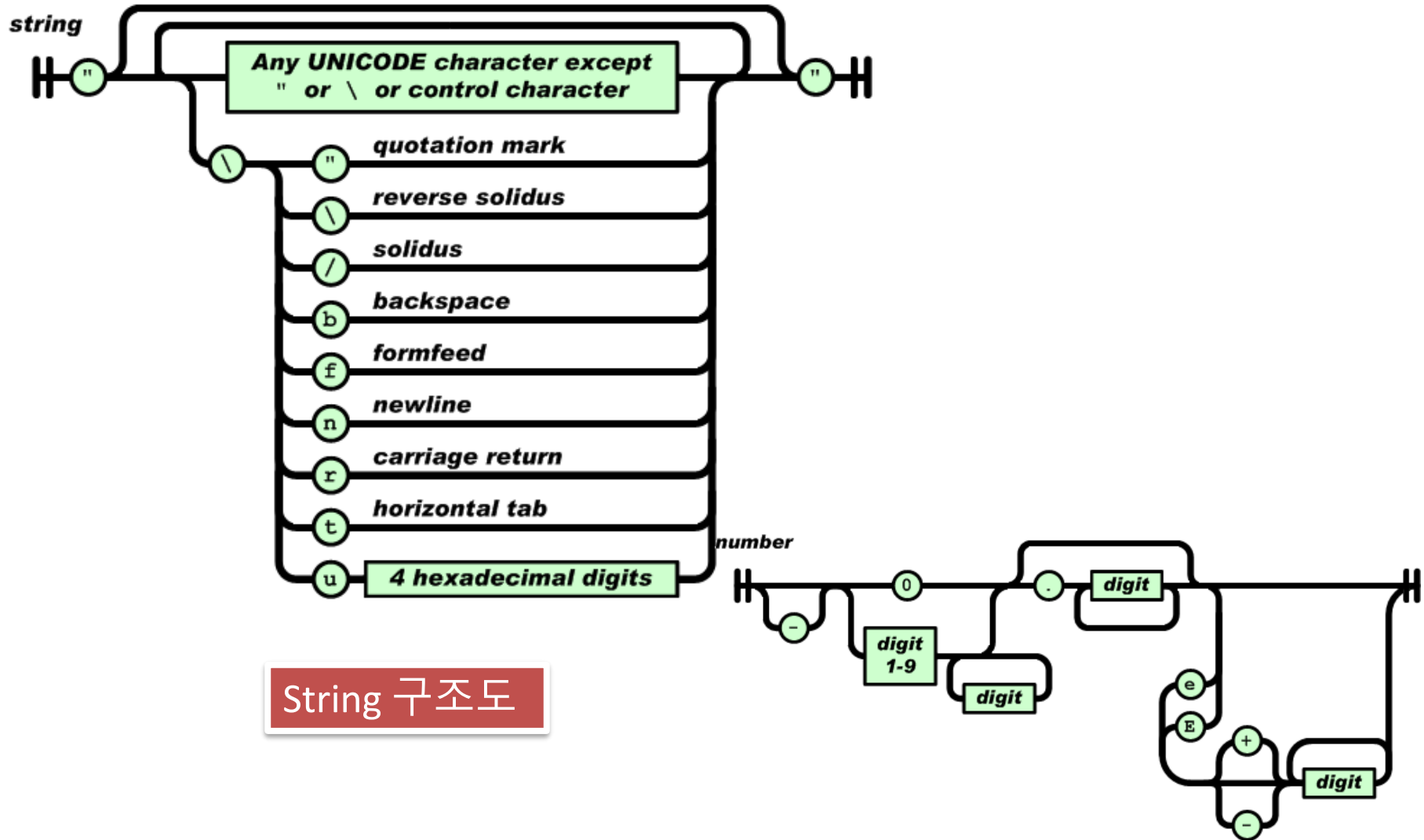
Object 구조도



Array 구조도



Value 구조도



String 구조도

Number 구조도

## ▪◎ JSON 문법

- JSON 문법은 자바스크립트 표준인 ECMA-262 3판의 객체 문법에 바탕
- 인코딩은 유니코드
- 표현할 수 있는 자료형에는 수, 문자열, 참/거짓이 있고, 또 배열과 객체도 표현

```
[10, {"v": 20}, [30, "마흔"]]
```

```
{"name2": 50, "name3": "값3", "name1": true}
```

```
{name2: 50, name3: "값3", name1: true}
```

```
{  
  "name": "홍길동",  
  "age": 30,  
  "sex": true,  
  "marriage": true,  
  "address": "서울특별시 송파구 석천동",  
  "hobby": ["축구", "독서"],  
  "family": {"#": 2, "아버지": "홍판서", "어머니": "이복순"},  
  "job": "웹서버 개발자"  
}
```

## ▪◎ JSON 문법

- JSON 문법은 자바스크립트 표준인 ECMA-262 3판의 객체 문법에 바탕
- 인코딩은 유니코드
- 표현할 수 있는 자료형에는 수, 문자열, 참/거짓이 있고, 또 배열과 객체도 표현

```
[10, {"v": 20}, [30, "마흔"]]
```

```
{"name2": 50, "name3": "값3", "name1": true}
```

```
{name2: 50, name3: "값3", name1: true}
```

- 파이썬에서는 Json 처리를 위해서는 json 모듈을 import 함
- json문자열을 만들기 위해서는 json 모듈의 dumps 함수
- json문자열을 보기좋게 정렬하려면 다음처럼 indent 옵션
- json 파일에 한글이 포함된 경우 ensure\_ascii=False 파라미터를 활용함
- json모듈의 loads 함수를 이용하여 만들어진 json 문자열을 파이썬 객체로 다시 변경

## #JSON making 샘플

```
import json
student = {
    'id': 2002324,
    'name': '홍길동',
    'history': [
        {'date': '2018-03-11', 'lang': 'java'},
        {'date': '2018-07-23', 'lang': 'python'},
    ]
}
# JSON making
jsonString = json.dumps(student, ensure_ascii=False, indent=4)
print(jsonString)
print(type(jsonString))
```

```
import json
sJson = '{"id": 2002324, "name": "홍길동", "history": [{"date": "2018-03-11",
"lang": "java"}, {"date": "2018-07-23", "lang": "python"}]}'

# JSON parsing
student = json.loads(sJson)

# Type체크
print(type(student))
print(student["name"])
for his in student['history']:
    print(his['date'], his['lang'])
print(dict['name'])
for h in dict['history']:
    print(h['date'], h['lang'])
```

data.json 파일

```
{"list":[
  {"name": "홍길동", "birth": "0304", "age": 30, "jum":[90,80,60]},
  {"name": "박호선", "birth": "0611", "age": 23, "jum":[60,40,90]},
  {"name": "이홍선", "birth": "0908", "age": 33, "jum":[70,40,90]},
  {"name": "박허용", "birth": "0706", "age": 31, "jum":[90,90,70]},
  {"name": "최남호", "birth": "0801", "age": 19, "jum":[80,70,80]}
]}
```

Jsonsamle.py 파일

```
import json
with open('./data.json',encoding='UTF8') as f:
    data = json.load(f)

print(type(data))

print(data)
```



netjsonsamle1.py 파일

```
import json
import urllib.request

url = "http://ip.jsontest.com" # URL

d = {'name': '홍길동', 'lang': 'python', 'age': 25}
params = json.dumps(d).encode("utf-8") # encode: 문자열을 바이트로 변환
print(params)
req = urllib.request.Request(url, data=params,
                             headers={'content-type': 'application/json'})
response = urllib.request.urlopen(req)
print(response.read().decode('utf8')) # decode: 바이트를 문자열로 변환
```

netjsonsamle2.py 파일

```
import requests
import json
from collections import OrderedDict

resp =
requests.get('https://www.govtrack.us/data/congress/113/votes/2013/s11/dat
a.json')
# resp.raise_for_status() 웹 응답코드 확인
if (resp.status_code == requests.codes.ok):
    html = resp.text
    print(html)
# data = json.loads(html, object_pairs_hook=OrderedDict) 입력순서대로 반환
보장
    data = json.loads(html)
    print("category", data["category"])
else:
    print(resp.status_code, '접속에 실패함')
```

# 정규표현식(Regular Expressions) : 정규식



- 복잡한 문자열을 처리할 때 사용하는 기법
- 정규식을 활용하면 많은 문자열 정보를 쉽게 관리가 가능
- 아래 응용 예제를 통한 필요성 이해

## re.py 파일

```
import re

data = """
park 800905-1049118
kim 700905-1059119
LEE 980603-2032072
"""

pat = re.compile("(\\d{6})[-]\\d{7}")
print(pat.sub("\\g<1>-*****", data))
```

## meta.py 파일

```
import re
p = re.compile( '[a-z]+' ) # 메타 문자열(정규표현식)
m = p.match("python")
print(m)
```

- 메타 문자열로 문자열에서 찾고자 하는 문자열 지정
- 메타 문자열 : . ^ \$ \* + ? { } [ ] \ | ( )

## 문자 클래스 [ ]

- 문자 클래스로 만들어진 정규식은 "[와 ] 사이의 문자들과 매치"라는 의미
- 문자 클래스를 만드는 메타 문자인 [와 ] 사이에는 어떤 문자도 들어갈 수 있음
- 정규 표현식이 [abc]라면 이 표현식의 의미는 "a, b, c 중 한 개의 문자와 매치"를 의미
- **샘플 [abc]와 매치 테스트**
  - "a"는 정규식과 일치하는 문자인 "a"가 있으므로 매치
  - "before"는 정규식과 일치하는 문자인 "b"가 있으므로 매치
  - "dude"는 정규식과 일치하는 문자인 a, b, c 중 어느 하나도 포함하고 있지 않으므로 매치되지 않음

```
import re
```

```
p = re.compile( '[abc]' )
```

```
print(p.match("a"))
```

```
print(p.match("before"))
```

```
print(p.match("test"))
```

```
# 객체 반환
```

```
# 객체 반환
```

```
# None 반환
```

## 문자 클래스 [ ] - 계속

- [ ] 안의 두 문자 사이에 하이픈(-)을 사용하면 두 문자 사이의 범위(From - To)를 의미
  - [a-c]라는 정규 표현식은 [abc]와 동일, [0-5]는 [012345]와 동일
  - [a-zA-Z] : 알파벳 모두, [0-9] : 숫자
  - ^ 메타 문자가 사용될 경우에는 반대(not)라는 의미
  - 예를 들어 [^0-9]라는 정규 표현식은 숫자가 아닌 문자만 매치
- [자주 사용하는 문자 클래스]

| 문자 | 의미                                                                     |
|----|------------------------------------------------------------------------|
| \d | 숫자와 매치, [0-9]와 동일한 표현식                                                 |
| \D | 숫자가 아닌 것과 매치, [^0-9]와 동일한 표현식                                          |
| \s | whitespace 문자와 매치, [ \t\n\r\f\v]와 동일한 표현식이다. 맨 앞의 빈 칸은 공백문자(space)를 의미 |
| \S | whitespace 문자가 아닌 것과 매치, [^ \t\n\r\f\v]와 동일한 표현식                       |
| \w | 문자+숫자(alphanumeric)와 매치, [a-zA-Z0-9_]와 동일한 표현식                         |
| \W | 문자+숫자(alphanumeric)가 아닌 문자와 매치, [^a-zA-Z0-9_]와 동일한 표현식                 |

## Dot(.)

- \n를 제외한 모든 문자와 매치됨을 의미
- a.b : a 모든문자 b
  - “aab”는 가운데 문자 “a”가 모든 문자를 의미하는 .과 일치하므로 정규식과 매치
  - “a0b”는 가운데 문자 “0”가 모든 문자를 의미하는 .과 일치하므로 정규식과 매치
  - “abc”는 “a”문자와 “b”문자 사이에 어떤 문자라도 하나는 있어야 하는 이 정규식과 일치하지 않으므로 매치 안됨
  - a[.]b 는 [] 사이라 그냥 일반문자 . 일때 만 매치

```
import re
p = re.compile( 'a.b' )
print(p.match("aab"))      # 객체 반환
print(p.match("a0b"))      # 객체 반환
print(p.match("abc"))      # None 반환

p = re.compile( 'a[.]b' )
print(p.match("a.b"))      # 객체 반환
print(p.match("aob"))      # None 반환
```

## 반복( \* )

- \* 앞의 문자가 0개에서 무한개까지 반복되는 문자와 매치
- 예제
  - “ct”는 가운데 문자 “a”가 0개 있으므로 정규식과 매치
  - “cat”는 가운데 문자 “a”가 1개 있으므로 정규식과 매치
  - “caaaat”는 “a”가 여러개 있으므로 정규식과 매치
  - “ca\*t”는 \* 표가 일반 문자가 a가 아니므로 매치 안됨

```
import re
p = re.compile( 'ca*t' )
print(p.match("ct"))           # 객체 반환
print(p.match("cat"))          # 객체 반환
print(p.match("caaaaat"))      # None 반환

print(p.match("ca*t"))          # None 반환
```

## 반복( + )

- \* 앞의 문자가 1 개에서 무한개까지 반복되는 문자와 매치
- 예제
  - “ct”는 가운데 문자 “a”가 0개 있으므로 정규식과 매치 안됨
  - “cat”는 가운데 문자 “a”가 1개 있으므로 정규식과 매치
  - “caaaat”는 “a”가 여러개 있으므로 정규식과 매치
  - “ca+t”는 + 표가 일반 문자가 a가 아니므로 매치 안됨

```
import re
p = re.compile( 'ca+t' )
print(p.match("ct"))           # None 반환
print(p.match("cat"))          # 객체 반환
print(p.match("caaaaat"))      # None 반환

print(p.match("ca+t"))         # None 반환
```



## 반복( {m,n} )

- {m, n} 정규식을 사용하면 반복 횟수가 m부터 n까지인 것을 매치
- {3,} 처럼 사용하면 반복 횟수가 3 이상인 경우 매치
- {,3} 처럼 사용하면 반복 횟수가 3 이하인 것을 의미
- {1,}은 +와 동일하며 {0,}은 \*와 동일
- {2} 는 반드시 지정된 횟수만 일치

| 정규식    | 문자열  | Match 여부 | 설명                    |
|--------|------|----------|-----------------------|
| ca{2}t | cat  | No       | "a"가 1번만 반복되어 매치되지 않음 |
| ca{2}t | caat | Yes      | "a"가 2번 반복되어 매치       |

```
import re
p = re.compile( 'ca{2}t' )
print(p.match("cat"))      # None 반환
print(p.match("caat"))     # 객체반환
```

## 반복( {m,n} )

- `ca{2,5}t` : "c + a(2~5회 반복) + t"

| 정규식                   | 문자열     | Match 여부 | 설명                    |
|-----------------------|---------|----------|-----------------------|
| <code>ca{2,5}t</code> | cat     | No       | "a"가 1번만 반복되어 매치되지 않음 |
| <code>ca{2,5}t</code> | caat    | Yes      | "a"가 2번 반복되어 매치       |
| <code>ca{2,5}t</code> | caaaaat | Yes      | "a"가 5번 반복되어 매치       |

```
import re
p = re.compile( 'ca{2,5}t' )
print(p.match("cat"))           # None 반환
print(p.match("caat"))          # 객체 반환
print(p.match("caaaaat"))       # 객체 반환
print(p.match("caaaaaaaat"))    # None 반환
```

## 반복( ? )

- ? 메타문자가 의미하는 것은 {0, 1}
- `ab?c` : "a + b(있어도 되고 없어도 된다) + c"
- {m.n} 은 쉽고 표현도 간결한 \*, +, ? 메타문자를 사용하는 것이 좋음

| 정규식               | 문자열               | Match 여부 | 설명                 |
|-------------------|-------------------|----------|--------------------|
| <code>ab?c</code> | <code>ac</code>   | Yes      | "b"가 0번 사용되어 매치    |
| <code>ab?c</code> | <code>abc</code>  | Yes      | "b"가 1번 사용되어 매치    |
| <code>ab?c</code> | <code>abbc</code> | No       | "b"가 2번 사용되어 매치 안됨 |

```
import re
```

```
p = re.compile( 'ab?c' )
```

```
print(p.match("ac"))
```

# 객체 반환

```
print(p.match("abc"))
```

# 객체 반환

```
print(p.match("abbc"))
```

# None 반환

## 반복( ^, \$ )

- ^, \$는 시작과 끝을 나타내는 메타문자
- ^는 시작문자로 매치
- \$는 마지막 문자로 매치

| 정규식   | 문자열       | Match 여부 | 설명                      |
|-------|-----------|----------|-------------------------|
| ^abc  | abcdefgh  | Yes      | 시작문자가 abc로 사용되어 매치      |
| ^abc  | zabcdefgh | No       | 시작문자가 abc로 사용되지않아 매치 안됨 |
| abc\$ | zyxabc    | Yes      | 끝문자가 abc로 사용되어 매치       |
| abc\$ | zxyabcd   | No       | 끝문자가 abc로 사용되지않아 매치 안됨  |

```
import re
p = re.compile('^abc')
print(p.search('abcdefgh'))
print(p.search('zabcd'))
p = re.compile('abc$')
print(p.search('zyxabc'))
print(p.search('zyxabcd'))
```

## 정규식 관련 메소드

- re 모듈을 활용하여 정규식을 표현함
- match, search는 정규식과 매치될 때에는 match 객체를 반환하고 매치되지 않을 경우에는 None을 리턴

| Method     | 목적                                           |
|------------|----------------------------------------------|
| match()    | 문자열의 처음부터 정규식과 매치되는지 판단.                     |
| search()   | 문자열 전체를 검색하여 정규식과 매치되는지 판단.                  |
| findall()  | 정규식과 매치되는 모든 문자열(substring)을 리스트로 반환         |
| finditer() | 정규식과 매치되는 모든 문자열(substring)을 iterator 객체로 반환 |

## 정규식 관련 메소드 - match

- 처음부터 정규식과 매치되는지 판단하여 match 또는 None 반환

```
import re
p = re.compile('[a-z]+')           # 영문자 소문자 판단
m = p.match("python")             # 객체 반환
print(m)
m = p.match("3 python")
print(m)                          # None 반환
```

## 정규식 관련 메소드 - search

- 문자열 전체에서 정규식과 매치되는지 판단하여 match 또는 None 반환

```
import re
p = re.compile('[a-z]+')
m = p.search("python java")
print(m)
print(m.group())
m = p.search("3 python java")
print(m)
print(m.group())
```

*# 영문자 소문자로 시작하는지 판단*

*# 객체 반환*

*# 객체 반환*

## 정규식 관련 메소드 - findall

- 문자열 전체에서 정규식과 매치되는 문자열을 리스트로 반환

```
import re
p = re.compile('[a-z]+')
result = p.findall("life is too short")
print(result)
result = p.findall("life is too short3long")
print(result)
```

*# 영문자 소문자 판단*



## 정규식 관련 메소드 - finditer

- 반복 가능한 객체(iterator object)를 리턴
- 반복 가능한 객체가 포함하는 각각의 요소는 match 객체

```
import re
p = re.compile('[a-z]+')
result = p.finditer("life is too short")
print(result)
result = p.finditer("life is too short3long")
print(result)
```

*# 영문자 소문자 판단*

## match 객체 메소드

- match, search 메소드 반환되는 객체가 match
- 매치가 된 실제적인 정보를 관리하는 객체

| method  | 의미                              |
|---------|---------------------------------|
| group() | 매치된 문자열을 리턴.                    |
| start() | 매치된 문자열의 시작 위치를 리턴.             |
| end()   | 매치된 문자열의 끝 위치를 리턴.              |
| span()  | 매치된 문자열의 (시작, 끝) 에 해당되는 튜플을 리턴. |

```
import re
p = re.compile('[a-z]+')
m = p.match("python java")
print(m.group())
print(m.start())
print(m.end())
print(m.span())
```

```
import re

m = re.search('[a-z]+', '3 python')
print(m.group())
print(m.start())
print(m.end())
print(m.span())
```

## 매치 옵션

- 정규식 실행시 옵션에 의해 다르게 매치됨
- 축약어 및 풀단어 옵션값 설정

| 옵션            | 의미                                       |
|---------------|------------------------------------------|
| DOTALL(S)     | . 이 줄바꿈 문자(\n)를 포함하여 모든 문자와 매치할 수 있도록 함. |
| IGNORECASE(I) | 대소문자에 관계없이 매치할 수 있도록 함.                  |
| MULTILINE(M)  | 여러줄과 매치할 수 있도록 함. (^, \$)                |
| VERBOSE(X)    | verbose 모드를 사용할 수 있도록 함.                 |

## 매치 옵션(DOTALL)

```
import re
p = re.compile('a.b')
m = p.match('a\nb')
print(m)
```

```
import re
p = re.compile('a.b', re.DOTALL)
m = p.match('a\nb')
print(m)
```

## 매치 옵션 IGNORECASE, I

```
import re
p = re.compile('[a-z]+', re.IGNORECASE)
print(p.match('python'))
print(p.match('python'))
print(p.match('PYTHON'))
print(p.match('3python'))
```

## 매치 옵션(MULTILINE, M)

- ^, \$메타 문자를 문자열의 각 라인마다 적용해 줌

```
import re
p = re.compile("^python\s\w+")
```

```
data = """python one
life is too short
python two
you need python
python three"""
```

```
print(p.findall(data))
```

```
import re
p = re.compile("^python\s\w+", re.M)
```

```
data = """python one
life is too short
python two
you need python
python three"""
```

```
print(p.findall(data))
```



## \문자열 처리와 r 문자열 포맷

- 정규표현식에서 \문자처리 시 주의점 (\d, \D, \s, \S, \w, \W, \t, \v 등)
- \를 정규식 표현 시 \\로 표현하여야 함
- 그냥 \ 하나로 처리하려고자 하는 경우 r 문자열로 표시함

```
import re
```

```
# \sport 문자열 서치 예제
```

```
p = re.compile("sport")
```

```
print(p.search('music\sport\reader')) # 그냥 sport 검색
```

```
p = re.compile('\sport')
```

```
print(p.search('music\s port\reader')) # 그냥 sport 검색 공백 port 검색
```

```
p = re.compile('\\sport')
```

```
print(p.search('music\sport\reader')) # \sport 검색
```

```
p = re.compile(r'\sport')
```

```
print(p.search('music\sport\reader')) # \sport 검색 이 방식을 추천
```

## 특수 메타문자( | )

- "or"의 의미와 동일
- A|B 라는 정규식이 있다면 이것은 A 또는 B라는 의미

```
import re
p = re.compile('Crow|Servo')
print(p.match('Crow Hello'))
print(p.match('Servo Hello'))
print(p.match('pytho nHello'))
```



## 특수 메타문자 ( \b, \B )

- \b 단어 구분자로 공백으로 처리됨
- 완벽한 단어 검색하는 경우 사용되는 옵션
- \B : \b의 반대로 해석됨

```
import re
```

```
p = re.compile(r'\bclass\b')
```

```
print(p.search('no class at all'))
```

# 객체 반환

```
print(p.search('the declassified algorithm'))
```

# None 반환

```
p = re.compile(r'\Bclass\B')
```

```
print(p.search('no class at all'))
```

# None 반환

```
print(p.search('the declassified algorithm'))
```

# 객체 반환

## 그룹핑( grouping )

- 반복문자열을 검색하고자 하는 메타문자
- ( 와 ) 를 활용해서 작성함
- 반복되는 문자열 검색

```
import re
p = re.compile('ABC')
m = p.search('python ABCABCABC OK ?')
print(m)
print(m.group())           # ABC 출력
p = re.compile('(ABC)+')
m = p.search('python ABCABCABC OK ?')
print(m)
print(m.group())           #ABCABCABC 출력
p = re.compile('ABC')
print(p.findall('ABC bpython ABCABCABC OK ?'))    # ['ABC', 'ABC', 'ABC', 'ABC']
p = re.compile('(ABC)+')
print(p.findall('ABC bpython ABCABCABC OK ?'))    # ['ABC', 'ABC']
```

## 그룹핑( grouping ) - 예제

```
import re
p = re.compile(r"\w+\s\d+[-]\d+[-]\d+")
m = p.search("park 010-1234-1234 python")
print("m.group()", m.group())      # 검색된 문자열에서 부분 접근 불가
p = re.compile(r"(\w+)\s+(\d+)[-](\d+)[-](\d+)")
m = p.search("park 010-1234-1234 python")
print("m.group()", m.group())
print("m.group(0)", m.group(0))    # 검색된 문자열 전체
print("m.group(1)", m.group(1))    # 검색된 문자열 첫번째 그룹 해당 문자열
print("m.group(2)", m.group(2))    # 검색된 문자열 두번째 그룹 해당 문자열
print("m.group(3)", m.group(3))    # 검색된 문자열 세번째 그룹 해당 문자열
```

## 그룹핑( grouping ) - 예제

```
import re
p = re.compile(r"(\w+)\s+(\d+)[-](\d+)[-](\d+)")
# p = re.compile(r"(\w+)\s+((\d+)[-]\d+[-]\d+)")
data = """
park 010-1234-1234 python
kim 019-1234-2234 java
lee 017-1234-3234 python
kang 018-1234-4234 node
"""

mList = p.findall(data)
print(mList)
```

## 출력결과

```
[('park', '010', '1234', '1234'), ('kim', '019', '1234', '2234'), ('lee', '017', '1234', '3234'), ('kang', '018', '1234', '4234')]
```

## 그룹핑( grouping ) – 내부 연속문자열 검색

- `(\b\w+)\s+\1` : (그룹1) + " " + "그룹1과 동일한 단어" 와 매치됨을 의미
- `\1` 은 그룹 1, `\2`는 그룹2 참조

```
import re
```

```
p = re.compile(r'(\b\w+)\s+\1')
```

```
mgroup = p.search('Paris in the the spring python is very very good  
language').group()
```

```
print(mgroup)
```

```
mlist = p.findall('Paris in the the spring python is very very good language ')
```

```
print(mlist)
```

## 출력결과

```
the the
```

```
['the', 'very']
```

## 그룹핑( grouping ) – 그룹핑에 이름 지정

- 그룹의 참조를 인덱스가 아님 이름으로 참조가 가능
- (?P<name>\w+)\s+((\d+)[-]\d+[-]\d+) : (\w+) --> (?P<name>\w+)
- m.group("name") 방식으로 참조 가능

```
import re
```

```
p = re.compile(r"(?P<name>\w+)\s+((\d+)[-]\d+[-]\d+)")  
m = p.search("park 010-1234-1234")  
print(m.group("name"))
```

```
import re
```

```
p = re.compile(r'(?P<word>\b\w+)\s+(?P=word)')  
print(p.search('Paris in the the spring').group())
```

## 그룹핑( grouping ) – 그룹핑에 이름 지정

- 그룹의 참조를 인덱스가 아님 이름으로 참조가 가능
- (?P<name>\w+)\s+((\d+)[-]\d+[-]\d+) : (\w+) --> (?P<name>\w+)
- m.group("name") 방식으로 참조 가능

```
import re
```

```
p = re.compile(r"(?P<name>\w+)\s+((\d+)[-]\d+[-]\d+)")  
m = p.search("park 010-1234-1234")  
print(m.group("name"))
```

```
import re
```

```
p = re.compile(r'(?P<word>\b\w+)\s+(?P=word)')  
print(p.search('Paris in the the spring').group())
```

## 찾은 문자열 바꾸기

- sub 메서드를 이용하면 정규식과 매치되는 부분을 다른 문자로 쉽게 바꿈
- p.sub("새로운문자열", "원데이터")

```
data = 'python is very good python is pythons python is perfect'
```

```
p = re.compile(r'\bpython\b')
```

```
new_data = p.sub('파이썬', data)
```

```
print(data)
```

```
print(new_data)
```

```
p = re.compile('(blue|white|red)')
```

```
c_str = p.sub('colour', 'blue socks and red shoes')
```

```
print(c_str)
```

## 출력결과

```
python is very good python is pythons python is perfect
```

```
파이썬 is very good 파이썬 is pythons 파이썬 is perfect
```

```
colour socks and colour shoes
```



## 그룹이름명으로 찾아바꾸기

- 내부의 그룹명으로 서로 내용 교체하기

```
import re
```

```
data = """
```

```
park 010-1234-1234
```

```
kim 019-1234-2234
```

```
lee 017-1234-3234
```

```
kang 018-1234-4234
```

```
"""
```

```
p = re.compile(r"(?P<name>\w+)\s+(?P<phone>(\d+)[-]\d+[-]\d+)"
```

```
print(p.sub("\g<phone> \g<name>", data ))
```

## 출력결과

010-1234-1234 park

019-1234-2234 kim

017-1234-3234 lee

018-1234-4234 kang

## 함수를 sub의 새로운 내용으로 삽입 - 1

```
import re
def hexrepl(match):
    """ return the hex string for a decimal number """
    value = int(match.group())
    return hex(value)

p = re.compile(r'\d+')
data = 'Call 65490 for printing, 49152 for user code.'
print(data)
result = p.sub(hexrepl, data)
print(result)
```

## 출력결과

```
Call 65490 for printing, 49152 for user code.
Call 0xffd2 for printing, 0xc000 for user code.
```

## 함수를 sub의 새로운 내용으로 삽입 - 2

```
import re
fNames = {"park":"박", "kang":"강", "lee":"이", "kim":"김"}
def exchang(match):
    key = match.group(1).lower()
    return fNames[key] + " "

p = re.compile(r'([a-z]+\s+)', re.IGNORECASE)
data = """
park 010-1234-1234
KIM 019-1234-2234
lee 017-1234-3234
kang 018-1234-4234
kim 010-1234-4234
"""
print(data)
result = p.sub(exchang, data)
print(result)
```

## 출력결과

```
park 010-1234-1234
KIM 019-1234-2234
lee 017-1234-3234
kang 018-1234-4234
kim 010-1234-4234

박 010-1234-1234
김 019-1234-2234
이 017-1234-3234
강 018-1234-4234
김 010-1234-4234
```

## Greedy vs Non-Greedy

- non-greedy 문자인 ?을 사용하면 가능한 한 가장 최소한의 반복을 수행

```
import re
import re
s = '<html><head><title>Title</title>'
print(re.match('<.*>', s).span())
print(re.match('<.*>', s).group())

print(re.match('<.*?>', s).span())
print(re.match('<.*?>', s).group())

taglist = re.findall('<.*?>', s)
print(taglist)
taglist = re.findall('<[a-z]*?>', s)
print(taglist)
```

## 출력결과

```
(0, 32)
<html><head><title>Title</title>
(0, 6)
<html>
['<html>', '<head>', '<title>', '</title>']
['<html>', '<head>', '<title>']
```