

Sensor Programming

센서 프로그래밍

파이썬 기본 프로그래밍



RaspberryPi



RASPBIAN

파이썬 기본 프로그래밍

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

파이썬 프로그래밍 소개

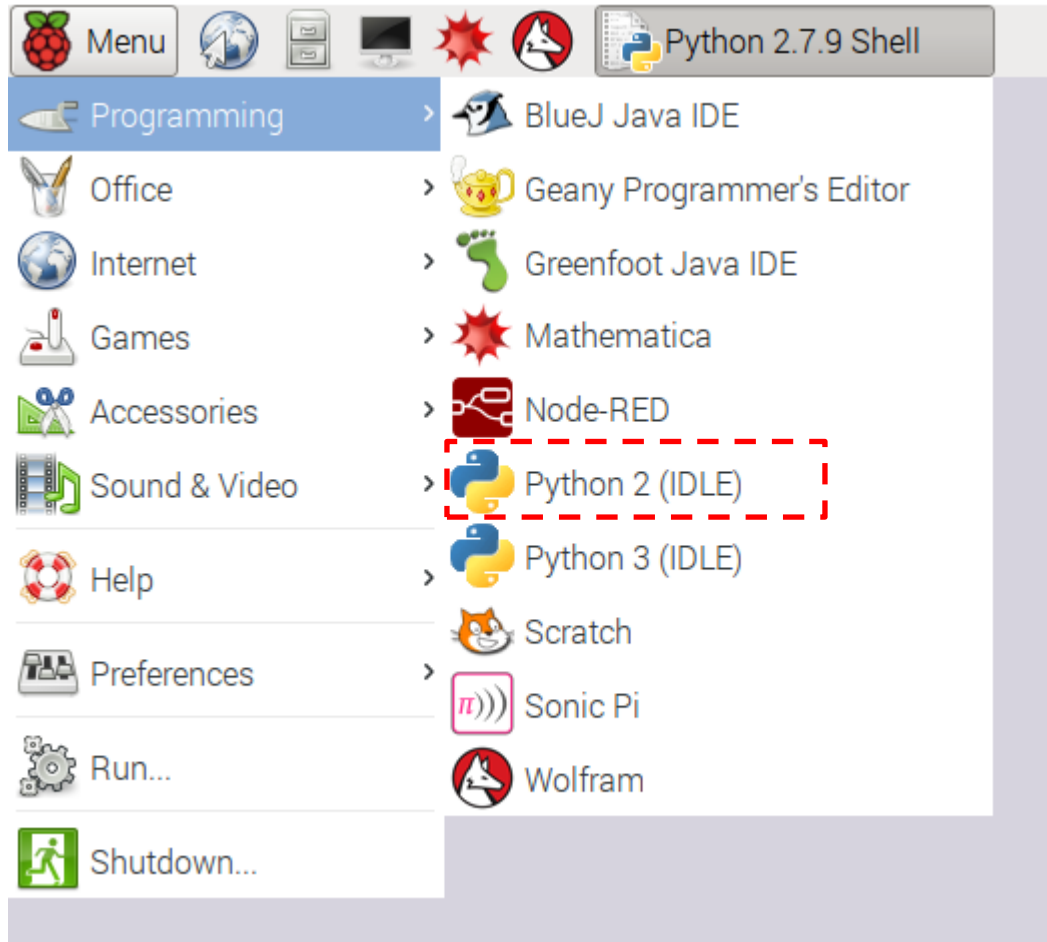
- 파이썬(Python)은 널리 쓰이는 범용, **고급언어**이다.
- 파이썬의 설계 철학은 **코드 가독성**에 중점을 두고 있으며 파이썬의 문법은 프로그래머가 (C와 같은 언어에서 표현 가능한 것보다도) **더 적은 코드로도 자신의 생각을 표현**하도록 한다.
- 파이썬은 프로그램의 크기에 상관없이 **명확**하게 프로그램 할 수 있는 구성 요소들을 제공한다.

– 위키피디아.

파이썬 프로그래밍 소개

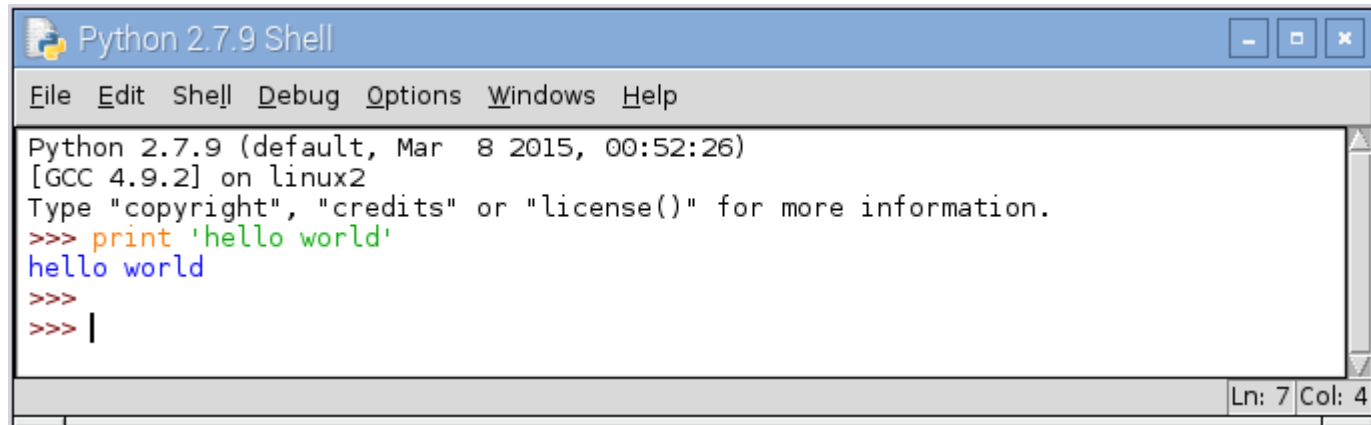
- 인터프리터 방식의 언어
 - vs 컴파일 방식
 - 가상머신으로 실행 (PVM : Python Virtual Machine)
 - 파일명.py는 실행시 바이트 코드 파일로 변환 (플랫폼 독립적)
- 코드의 간결성
 - 개발 생산성 향상과 쉬운 유지보수
 - TIME TO MARKET
- 객체지향을 지원
 - 코드의 재사용성 (수많은 라이브러리를 제공)
 - 유지보수의 용이성
- 파이썬은 고급 프로그래밍 언어이다.
- 속도를 개선하기 위해 C로 개발된 루틴을 외부 모듈로 사용 가능

파이썬 개발환경



파이썬 개발환경

➤ 파이썬 쉘에서 실행



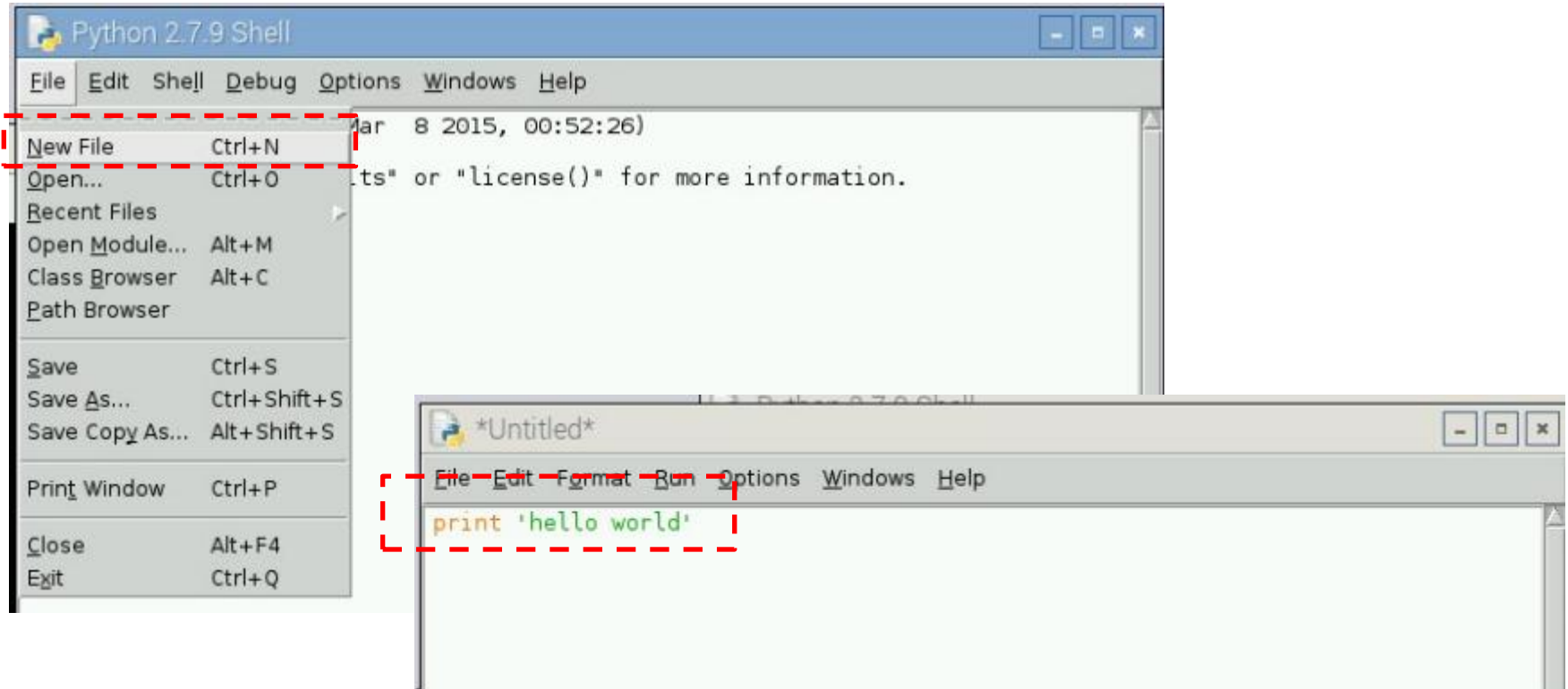
The image shows a screenshot of a Python 2.7.9 Shell window. The window has a title bar that says "Python 2.7.9 Shell" and standard window control buttons (minimize, maximize, close). Below the title bar is a menu bar with the following items: File, Edit, Shell, Debug, Options, Windows, and Help. The main area of the window is a text editor displaying the following text:

```
Python 2.7.9 (default, Mar  8 2015, 00:52:26)
[GCC 4.9.2] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> print 'hello world'
hello world
>>>
>>> |
```

At the bottom right of the window, there is a status bar that displays "Ln: 7 Col: 4".

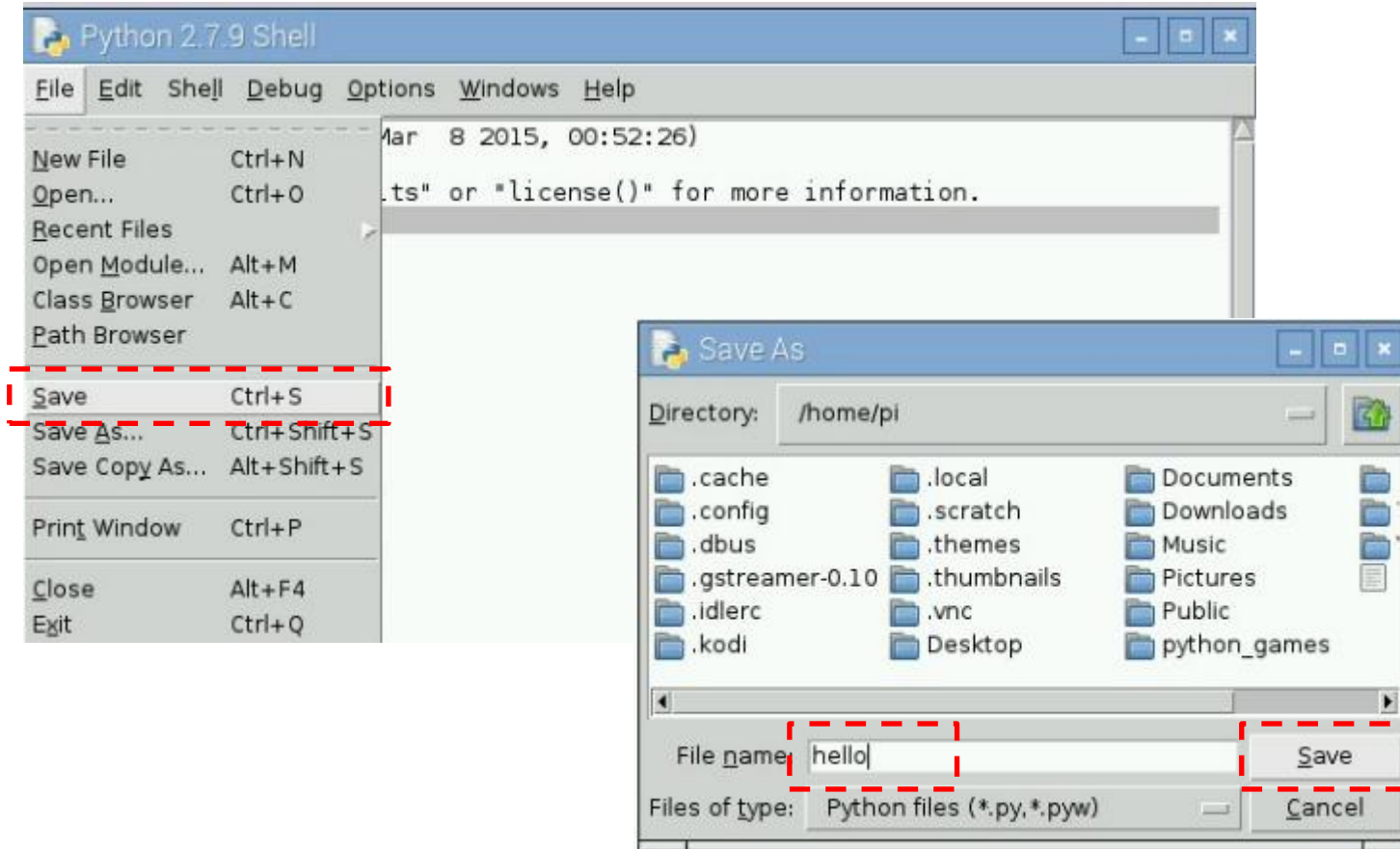
파이썬 개발환경

- 별도의 파일에 프로그램 작성



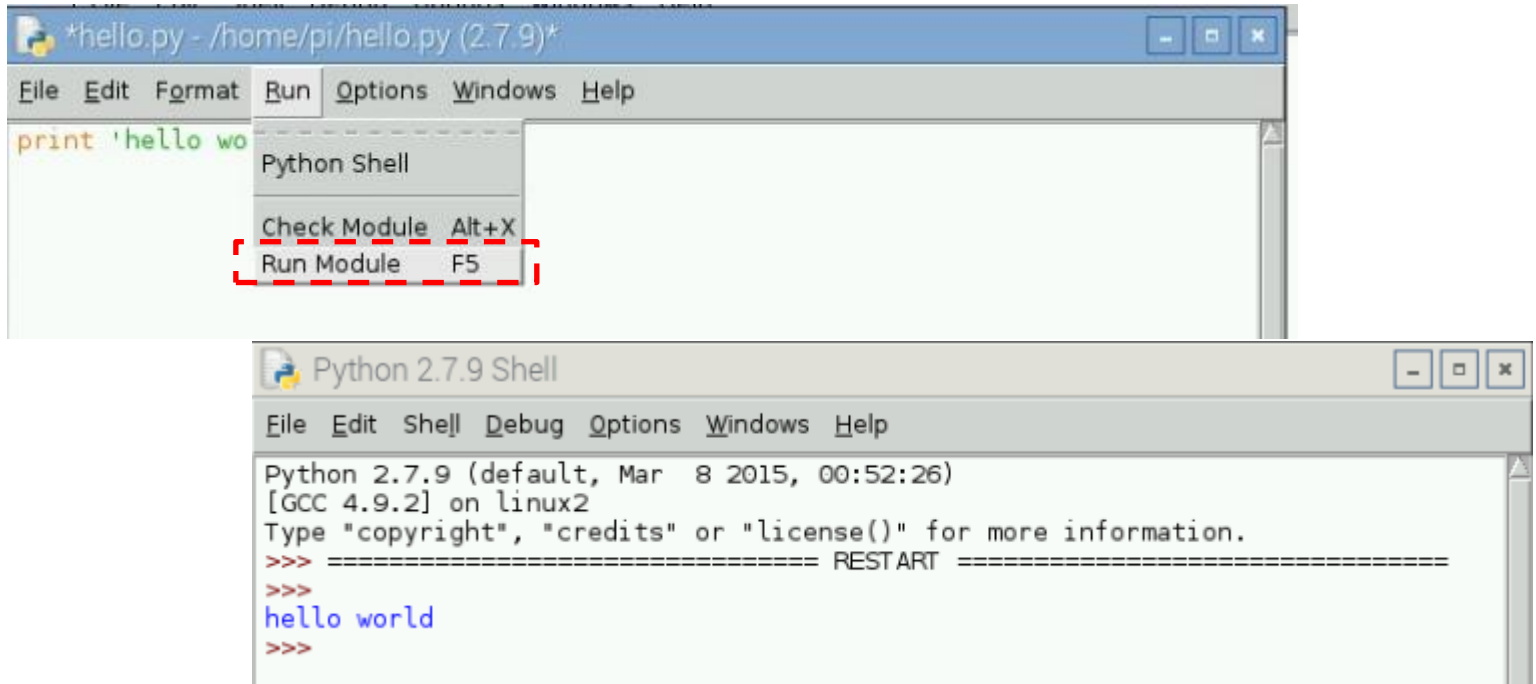
파이썬 개발환경

- 별도의 파일에 프로그램 작성



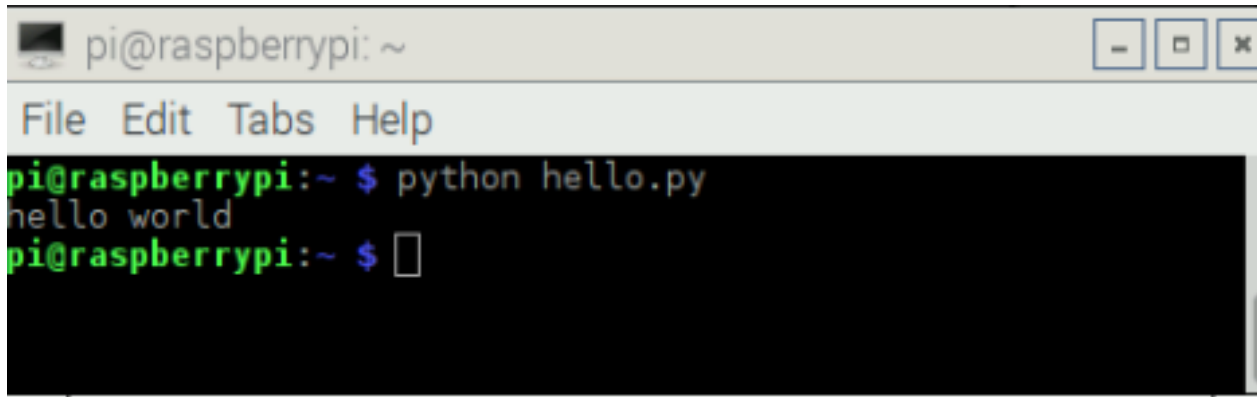
파이썬 개발환경

➤ 실행방법 1 : 파이썬 셸에서 실행



파이썬 개발환경

- 실행방법 2 : 터미널에서 실행
 - control + Alt + t
 - 소스코드 디렉토리로 이동
 - \$ cd /home/pi
 - \$ python hello.py



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ python hello.py  
hello world  
pi@raspberrypi:~ $
```

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. 형변환

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)
- 사전(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)

- 일상 생활에 사용하는 정수의 표현이 가능
- 표현 가능 범위 == 메모리 용량
- 정수의 진법 표현
 - 2진법: 0b로 시작하고 0과 1로 수를 표현한다.
 - 0b1010, 0b1110111, 0b10101010 등
 - 8진법: 0로 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
 - 31.4000000000000002
- 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 표현한다.

```
>>> "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.

```
>>> 'He shouted "Help me!"'
```

```
'He shouted "Help me!"'
```

```
>>> "He doesn't like it."
```

```
"He doesn't like 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"""
```

```
'Hello World\nHello Korea\n안녕 대한민국'
```

```
>>> ''' Hello World
```

```
... Hello Korea
```

```
... 안녕'''
```

```
'Hello World\nHello Korea\n안녕
```

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)
\v	수직 탭
\t	수평 탭
\r	캐리지 리턴
\f	폼피드
\a	비프음 (뽕 소리)
\b	백스페이스
\000	널문자
\\	"\" 문자
\'	단일 인용부호(')
\"	이중 인용부호(")

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, 'str']
```

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

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

```
[1, [2], ['str', 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 크기를 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 World"	"안녕 세상아"
[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 세상아"][-1]
```

```
'안녕 세상아'
```

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

```
'Hello World'
```

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

```
True
```

```
>>> [1, 2, 3.14, True, "Hello World", "안녕 세상아"][-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 (mutable하다고 한다)
 - 튜플의 경우 내용의 변경이 불가 (immutable하다고 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 크기를 구하기 위해 리스트와 같이 len 함수를 쓰면 된다.

기본 데이터 타입 : 튜플(tuple) 타입

```
>>> (1, 2, 3.14, True, "Hello World", "안녕 세상아")
(1, 2, 3.14, True, 'Hello World', '안녕 세상아')
>>> (1, 2, (1, 2), ("Hello World", True))
(1, 2, (1, 2), ("Hello World", True))
>>> x = [1, 2, 3.14, True, "Hello World", " 안녕 세상아 "]
>>> x[2] = 3
>>> print(x)
[1, 2, 3, True, "Hello World", "안녕 세상아"]
>>> y = (1, 2, 3.14, True, "Hello World", " 안녕 세상아 ")
```

```
>>> y[2] = 3
```

Traceback (most recent call last):

File "<stdin>", 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
```

```
>>> x
```

```
2
```

```
>>> y
```

```
1
```

기본 데이터 타입 : 사전(dict) 타입

- 순서가 없는 키-값의 쌍으로 된 집합이다.
- 키에 의해 데이터 값에 접근할 수 있다.
- 키-값의 쌍을 중괄호{ }로 묶어주면 된다.
- {키: 값, ... }

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

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 : 형변환

```
>>> 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 포함할수 있는 데이터 타입이다.

컨테이너 타입의 종류는 문자열, 리스트, 튜플, 집합, 사전이 있다.

```
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?

- 변수명은 데이터에 붙인 이름이다.
- 즉 데이터에 다양한 이름을 부여할 수 있다.

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

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

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

```
>>> i1
```

```
1
```

```
>>> f1
```

```
3.14
```

```
>>> s1
```

```
"Hello World"
```


함수, 객체지향

■ 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
- 사전
- 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 next이다.
 - next는 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, 사전 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 세상아")
```

```
>>> 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 "전체 요소의 개수 - 1"이 된다.
- 만약 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 세상아"
```

```
>>> s1
```

```
'Hello World 안녕세상아'
```

```
>>> 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
```

```
'Hello World 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)
'r'
>>> 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)
' '
>>> 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
```

```
15
```

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

```
>>> val
```

```
30
```

```
>>> 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 평가한다
수식1 != 수식2	수식1과 수식2 값이 같지 않음을 평가한다.
수식1 > 수식2	수식1의 값이 수식2의 값 보다 큰가를 평가한다.
수식1 < 수식2	수식2의 값이 수식1의 값 보다 큰가를 평가한다.
수식1 >= 수식2	수식1의 값이 수식2의 값 보다 같거나 큰가를 평가한다.
수식1 <= 수식2	수식2의 값이 수식1의 값 보다 같거나 큰가를 평가한다.

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.

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

입력 값		논리연산		
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

비트 연산자(Bitwise Operators)

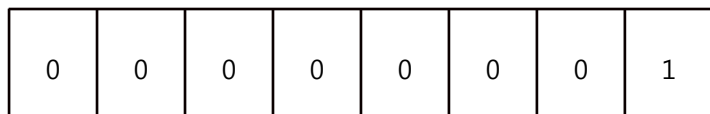
```
>>> data=0x12
>>> mask = 0x3
>>> data & mask
2
>>> data | mask
19
>>> data ^ mask
17
>>> ~data
-19
>>> data = 0
>>> ~data
-1
```

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

쉬프트 연산자

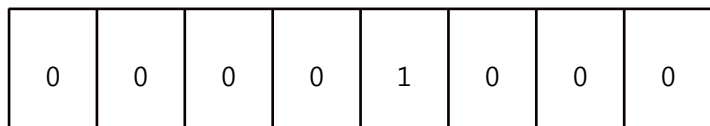
- 이진 데이터로 표현된 상태에서 왼쪽 혹은 오른쪽으로 정해진 만큼 이동시킨다.
- 좌측 쉬프트의 경우 오른쪽 빈 공백만큼은 0으로 채워진다. 이는 쉬프트 횟수만큼 2를 곱하는 효과가 난다.
- 우측 쉬프트의 경우 부호가 유지되면서 오른쪽으로 이동시킨다. 이는 쉬프트 횟수만큼 2로 나누는 효과가 난다.

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



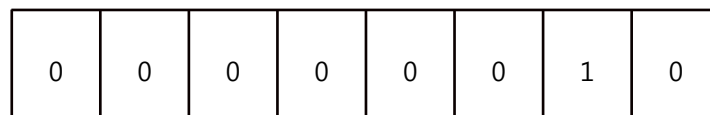
1

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 산출해낸다.

```
>>> "Hello " + "World"  
'Hello World'  
>>> "안녕 " + "세상아"  
'안녕 세상아'
```

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")
'jy loves sy'
>>> "%s loves No.%d" % ("yj", 7)
'yj loves No.5'
>>> "%d x %d = %d" % (6, 9, 54)
'6 x 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 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'
```

제어문

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

제어문

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

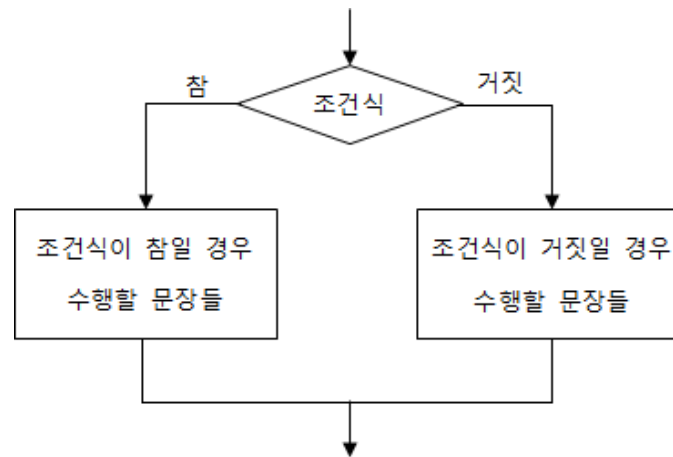
```
print("1")  
print("2")  
print("3")
```

조건문으로 흐름을 제어할 수 있다.

if else 문

문법:

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



제어문

데이터 입력 받기

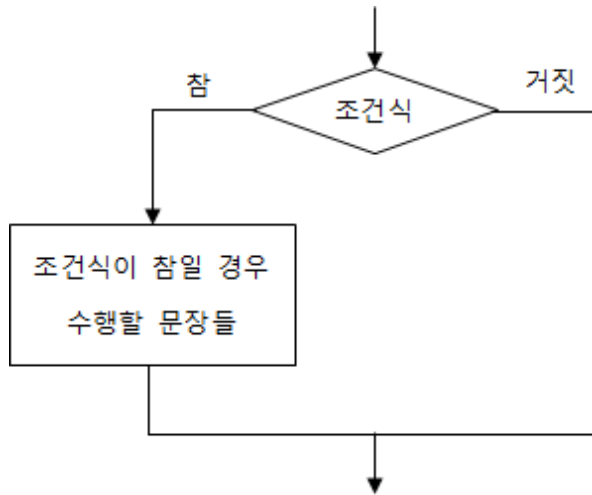
`input([프롬프트]) -> 문자열`

```
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!")
```

제어문

if 문만 있는 경우



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

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

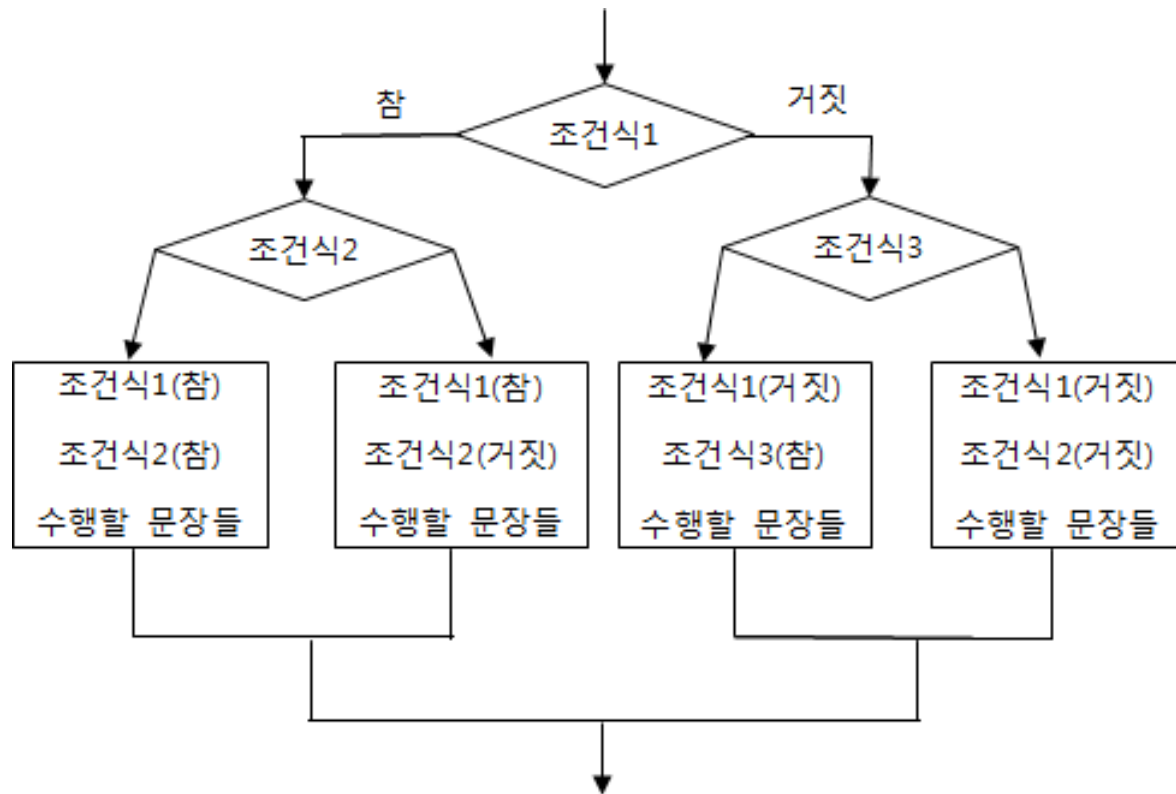
```
    print("당신은 장학금 수여 대상자 입니다.")
```

```
    print("축하합니다.")
```

```
print("공부 열심히 하세요.")
```


제어문

중첩된 if ~ else 문



제어문

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.")
```

제어문

중첩된 if ~ else 문

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

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

```
    print("수")
```

```
else:
```

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

```
        print("우")
```

```
    else:
```

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

```
            print("미")
```

```
        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:
```

```
    print("우")
```

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

```
    print("미")
```

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

```
    print("양")
```

```
else:
```

```
    print("가")
```

for문

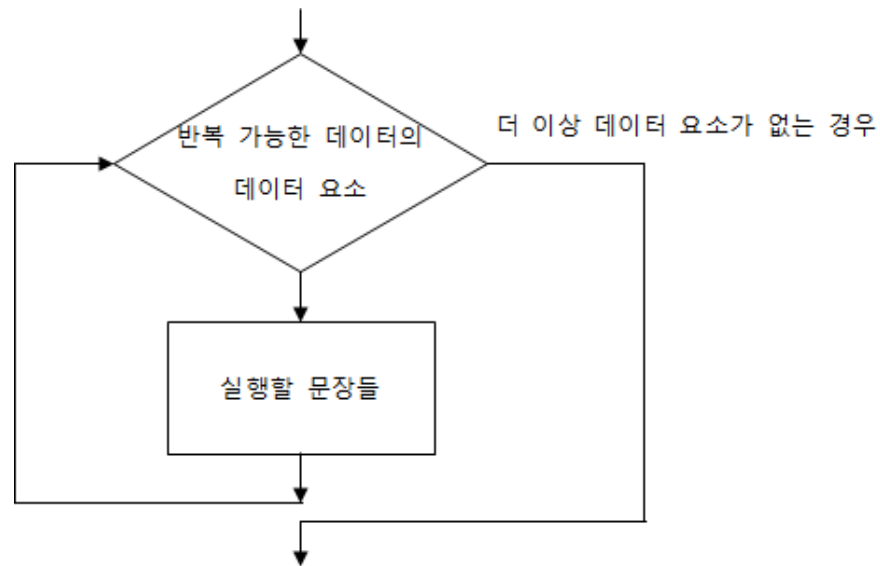
- for문의 문법

for 변수 in 반복_가능한_데이터:

수행할 문장들

[else:

수행할 문장]



for 문

```
message = "Hello!"
messages = ["Hello World", "안녕, 세상아"]
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")
```

range 함수

range 함수는 범위를 생성하는 함수이다.

```
>>> 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):
    total = total + item
print("1부터 10까지 합은", total, "입니다")
```

range 함수

range 함수는 범위를 생성하는 함수이다.

```
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")
```

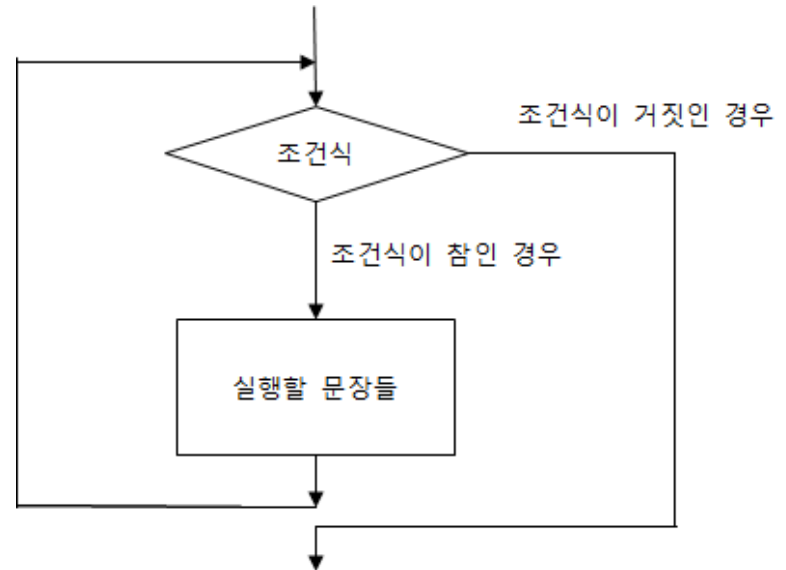
```
total = 0
for item in range(1, 101):
    if (item % 3) == 0 or (item % 7) == 0:
        total = total + item
print("1부터 100까지에서 3 혹은 7의 배수의 합은", total, "입니다")
```


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:
    print(count)
    count = count + 1
count = 1
```

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

```
count = 1
result = 0
while count <= 100:
    result = result + 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 수행한다.

- continue문

반복문에서 남은 문장을 수행하지 않고 다음 단계로 넘어간다.

```
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 경우이다.

```
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.")
```

함수

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 함수
- 기본 함수 혹은 built-in 함수
 - 기본 함수는 파이썬의 실행과 동시에 사용할 수 있는 함수들이다.
- 라이브러리 혹은 패키지 함수
 - 해당 라이브러리를 포함한 후에 사용할 수 있다.

파이썬 함수종류

■ 함수를 작성하는 이유

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

▪ 함수 문법

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

사용자 정의 함수

■ `__name__` 변수

- 파이썬 인터프리터가 파이썬 프로그램을 입력 받아서 실행하면 `__name__` 을 "`__main__`"으로 설정한다.

```
def hello_message( ):
    print("Hello World")

if __name__ == "__main__":
    hello_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

```
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

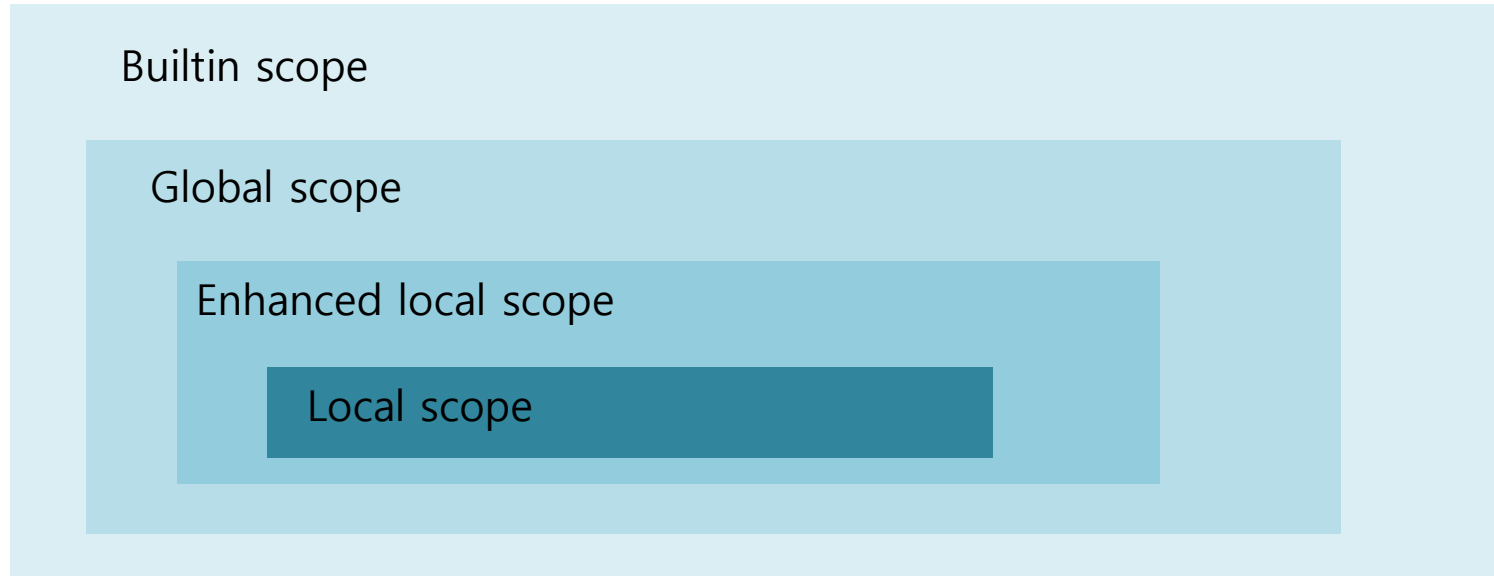
```
def circle_area(radius, print_format):  
    area = 3.14 * (radius ** 2)  
    print_format(area)
```

```
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():
    glob 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 디버깅 해야함
 - 전역 변수를 많이 참조 할수록 디버깅은 어려워짐
 - 함수 안의 범위만 쉽게 디버깅 하기 위해서 지역변수를 사용하자.

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 수정함으로써 오류를 제거한다.

예외

■ 예외

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

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

```
^CTraceback (most recent call last):  
  File "test.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
```

예외 처리 구문

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

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

```
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)
```