Python Basics

윤 길 배

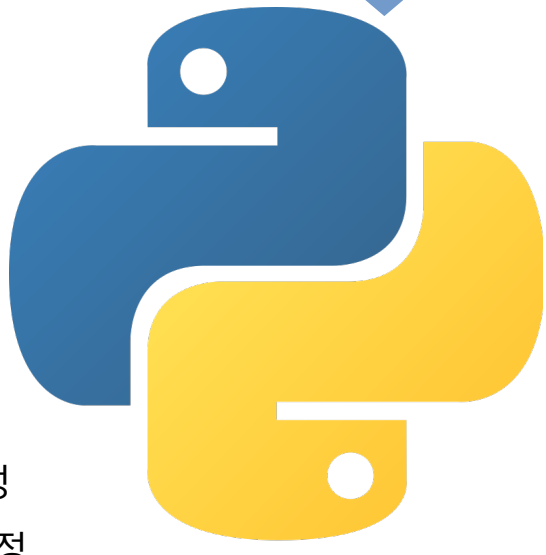
팔복기술주식회사

2022

차 례

1. Python 소개

- 창시자 : 휘도 판로쉼(Guido van Rossum , 1956~)
- 1989년은 성탄절이 월요일 완성. 1991년 발표.
- 읽고 쓰기 쉬운 프로그래밍 언어.



초보자에게 적합

- 프로그래머에게 훌륭한 튜토리얼 풍부함.
- 데이터 유형을 식별할 필요가 없음. 파이썬은 상황에 따라 그것이 정수인지, 부동 소수점 값인지, 부울 값인지, 아니면 다른 무엇인지 결정
- 파이썬의 문법은 영어 문법과 유사 – 읽기 쉬움

우수한 생산성

- 다른 프로그래밍 언어보다 적은 줄로 프로그램을 작성할 수 있는 구문.
- 개발자들의 모든 필요에 대응하는 라이브러리 완비 : Numpy, Scipy, Matplotlib, pandas
- 인터프리터 언어 - 코드가 작성되는 즉시 실행. 빠른 프로토타이핑 가능.

1. Python 소개

사용분야

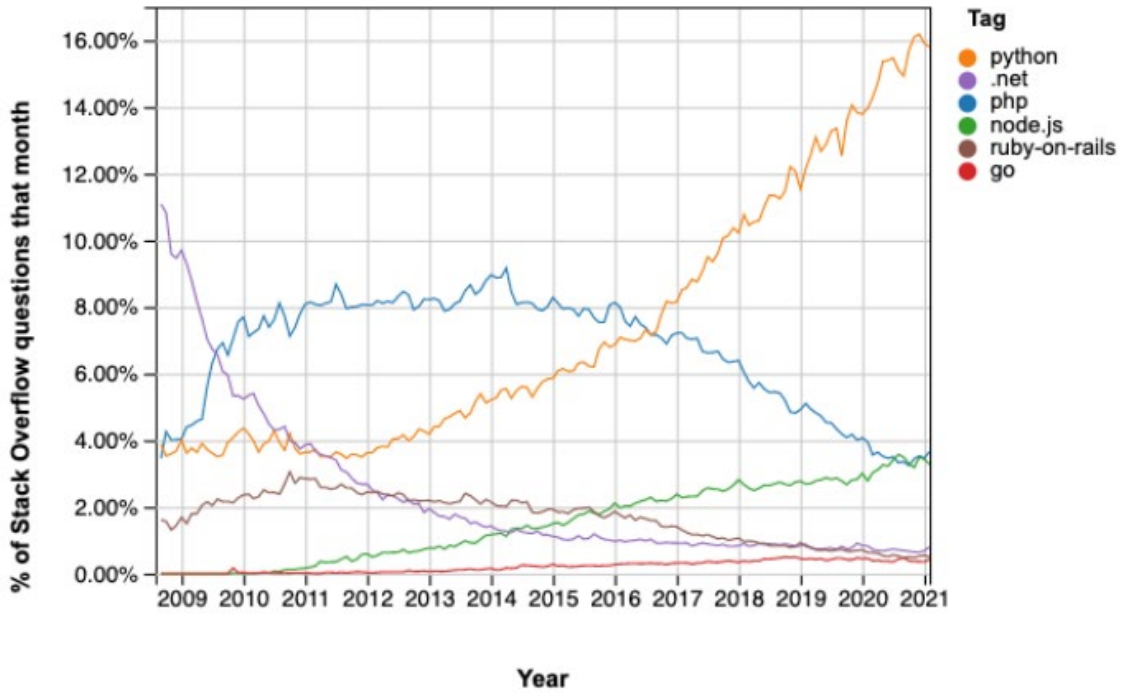
- 머신러닝 기본언어
- 웹 개발(서버 측),
- 소프트웨어 개발,
- 수학,
- 시스템 스크립팅.

Python 구문의 특징

- 가독성을 위해 영어와 유사한 구문구조.
- 세미콜론이나 괄호를 자주 사용하는 다른 프로그래밍 언어와 달리 새 줄로 명령완성
- {} 를 사용하지 않고 공백을 사용하여 루프, 함수 및 클래스 등 코드의 범위를 정의.

1. Python 소개

- StackOverflow 문의 중 16%가 "파이썬".



Stack Overflow Trends (March 2021)

1. Python 소개 - 설치 및 기본실행

설치

- 명령 프롬프트에서,

✓ > python --version

- https://www.python.org/

실행

- 파일로 실행
  - ✓ helloworld.py 작성
  - ✓ 내용 : print("Hello, World!")
  - ✓ C:\Users\Your Name>python helloworld.py

- 명령줄 실행
  - ✓ > python
  - ✓ Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
  - ✓ Type "help", "copyright", "credits" or "license" for more information.
  - ✓ >>> print("Hello, World!")
  - ✓ "Hello, World!"
  - ✓ exit()

1. Python 소개 - 들여쓰기

들여쓰기

- Python은 들여쓰기를 사용하여 코드 블록을 나타냄.
- 동일한 코드 블록에서 동일한 수의 공백을 사용해야 함.
  - ✓ if 5 > 2:
  - ✓ print("Five is greater than two!")

- 들여쓰기를 생략하면?
  - ✓ if 5 > 2:
  - ✓ print("Five is greater than two!")
  - ✓ if 5 > 2:
  - ✓ print("Five is greater than two!")

변수

- 파이썬에는 변수 선언을 위한 명령이 없음.
- Python에서 변수는 값을 할당할 때 생성.
  - ✓ x = 5
  - ✓ y = "Hello, World!"

1. Python 소개 - 주석

- 주석은 #으로 시작, 코드를 설명하는 데 사용.
- 코드를 테스트할 때 실행을 방지하기 위해 사용.

```
#This is a comment.
print("Hello, World!")

print("Hello, World!") #This is a comment

#print("Hello, World!")
print("Cheers, Mate!")

# 여러 줄 주석

#This is a comment
#written in
#more than just one line
print("Hello, World!")

"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

2. 변수

- 많은 프로그래밍 언어에서 변수는 데이터를 담는 콘테이너(container) 또는 버킷(bucket).

  ex) in C : int x = 4

- 파이썬에서는 변수는 콘테이너보다는 포인터.
- 변수명에는 데이타유형 정보가 없고, 생성된 데이터 객체에 유형정보가 있음.
- 변수는 처음 값을 할당하는 순간 생성.

```
x = [1, 2, 3]
y = x
print(y)
x.append(4)
print(y)
```

- 설정된 후 유형 변경가능.

```
x = 4       # x is of type int
x = " Sally "  # x is now of type str
print(x)
```

- 캐스팅

```
x = str(3)    # x will be ' 3 '
z = float(3)  # z will be 3.0
```

- 유형 확인 : type()함수

```
x = 5
y = " John "
print(type(y))
```

2. 변수 - 변수이름

- 문자열 변수는 작은따옴표나 큰따옴표를 사용하여 선언할 수 있음.
- 변수 이름은 대소문자를 구분함.

```
x = "John"
# is the same as
x = 'John'
a = 4
A = "Sally"
#A will not overwrite a
```

- 변수 이름 규칙

✓ 변수 이름은 문자 또는 밑줄 문자로 시작.
✓ 변수 이름은 숫자로 시작할 수 없음.
✓ 변수 이름은 영숫자 문자와 밑줄(Az, 0-9 및 _)만 포함할 수 있다.
✓ 변수 이름은 대소문자를 구분.

```
•myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

- 가독성을 높이기 위한 규칙

✓ Camel case : 첫 번째 단어를 제외한 각 단어는 대문자로 시작.
myVariableName = "John"

✓ Pascal case : 각 단어는 대문자로 시작
MyVariableName = "John"

✓ Snake case : : 각 단어는 밑줄 문자로 구분.
my_variable_name = "John"

2. 변수 - 변수값 할당

- 여러 변수 - 여러 값 할당

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

- 여러 변수에 하나의 값

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

- 컬렉션 압축 풀기

✓ 목록, 튜플 등에 값 모음이 있는 경우 값을 변수로 추출.

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

2. 변수 - 변수출력

- 텍스트와 변수를 결합하기 위해 +문자를 사용 .

```
x = "awesome"
print("Python is " + x)

x = "Python is "
y = "awesome"
z =  x + y
print(z)
```

- 숫자의 경우 +문자는 수학 연산자로 작동.

```
x = 5
y = 10
print(x + y)
```

- 문자열과 숫자를 결합하려고 하면 Python에서 오류가 발생합니다.

```
x = 5
y = "John"
print(x + y)
```

2. 변수 - 변수 - 전역변수

- 함수 외부에서 생성된 변수는 전역 변수.
- 전역 변수는 함수 내부와 외부 모두에서 모든 사람이 사용할 수 있음.

```
x = "awesome"
def myfunc():
  print("Python is " + x)

myfunc()
```

- 함수 내에서 같은 이름의 변수를 생성하면?
  - ✓ 지역 변수가 되며 함수 내에서만 참조.
  - ✓ 이름이 같은 전역 변수는 원래 값과 전역 변수가 그대로 유지됨.

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()
print("Python is " + x)
```

2. 변수 - 전역변수

- global
- 함수 내부에 전역 변수를 생성하려면 global키워드 사용.

```
def myfunc():
  global x
  x = "fantastic"

myfunc()
print("Python is " + x)
```

- 함수 내 전역 변수의 값을 변경할 때

```
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

3. 변수유형 – 숫자

- 파이썬에는 3 종류의 숫자형이 있음.
  - ✓ int
  - ✓ float
  - ✓ complex

```
# Example
x = 1     # int
y = 2.8   # float
print(type(x))
print(type(y))

# Float 표현방식
x = 35e3
y = 12E4
z = -87.7e100
print(type(x))
print(type(y))
print(type(z))
```

- 숫자의 형변환

```
X = 1     # int
Y = 2.8   # float

#convert from int to float:
A = float(x)

#convert from float to int:
B = int(y)

Print(a)
Print(b)
Print(type(a))
Print(type(b))
```

- 난수

```
import random
print(random.randrange(1, 10))  # 1 ~ 9
```

3. 변수유형 – 부동소숫점 문제

```
0.1 + 0.2 == 0.3
→?

print("0.125 = {0:.17f}".format(0.125))

print("0.1 = {0:.17f}".format(0.1))
print("0.2 = {0:.17f}".format(0.2))
print("0.3 = {0:.17f}".format(0.3))
```

- 부동소수를 사용하는 모든 프로그래밍 언어는 한정된 개수의 비트에 저장.
- 일부 숫자는 근사적으로 표현됨.

- $1/3=0.333333333\cdots_{10}$

$0.125 = 0.001_2$

$1/10=0.0001100110011\cdots_2$

https://www.rapidtables.com/convert/number/decimal-to-binary.html

3. 변수유형 – 숫자변수 유형지정

- 변수 유형 지정
  ✓ 클래스 생성자 함수를 사용하여 데이터 유형을 정의 가능.

- 정수:

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

- 부동소숫점:

```
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2
```

- 문자열:

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

3. 변수유형 – 문자열

- 작은따옴표(')나 큰따옴표(")사용 가능.

```
print("Hello")
print('Hello')

a = "Hello"
print(a)
```

- 여러 줄 문자열 : 세 개의 따옴표를 사용

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

- 대괄호를 사용하여 문자열의 요소참조.

```
a = "Hello, World!"
print(a[1])
```

- 문자열 반복

```
for x in "banana":
  print(x)
```

- 문자열 길이

```
a = "Hello, World!"
print(len(a))
```

- 문자열 있는지 확인

```
txt = "The best things in life are free!"
print("free" in txt)
```

- 문자열 없는지 확인

```
txt = "The best things in life are free!"
if "expensive" not in txt:
  print("No, 'expensive' is NOT present.")
```

3. 변수유형 – 문자열 - 문자열선택 - 슬라이싱

- 슬라이싱
  - ✓ 슬라이스 구문을 사용하여 문자 범위를 반환.
  - ✓ 문자열의 일부를 반환하려면 시작 인덱스와 끝 인덱스를 콜론으로 구분하여 지정.

```
b = "Hello, World!"
print(b[2:5])
```

- 처음부터 슬라이스

```
b = "Hello, World!"
print(b[:5])
```

- 끝까지 슬라이스

```
b = "Hello, World!"
print(b[2:])
```

- 네거티브 인덱싱
  - ✓ 문자열 끝에서 슬라이스를 시작.

From : "World!"의 "o" (위치 -5)

To : "World!"의 "d" (위치 -2):

```
b = "Hello, World!"
print(b[-5:-2])
```

3. 변수유형 –문자열 - 문자열가공

- 대문자/소문자로 변환

```
a = "Hello, World!"
print(a.upper())
print(a.lower())
```

- 앞뒤 공백 제거

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

- 문자열 바꾸기

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

- 문자열 분할
  - ✓ 지정된 구분 기호 사이의 텍스트가 목록 항목이 되는 목록을 반환.

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

- 문자열 연결 : + 연산자를 사용.

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

사이에 공백을 추가하려면.

```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

3. 변수유형 – 문자열 - 문자열가공 - format

- 문자열과 숫자를 결합하는 방법

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

- format() : 전달된 인수를 가져와 문자열의 {} 위치에 배치.

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

- 무제한의 인수 사용 가능.

```
Quantity = 3
Itemno = 567
Price = 49.95
Myorder =  " I want {} pieces of item {} for {} dollars. "
Print(myorder.format(quantity, itemno, price))
```

- 인덱스 번호 {0}를 사용 하여 인수위치 수동 지정 가능

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

3. 변수유형 – 문자열 - 문자열가공 – 탈출문자

- 문자열에 특수문자를 삽입하고자 할 때 사용.
- 백슬래시(₩) 다음에 특수문자를 놓으면 일반문자로 인식.

  ✓ 큰따옴표로 묶인 문자열 안에 큰따옴표를 사용하면 오류 발생.

Txt = " We are the so-called " Vikings " from the north. "

Txt = " We are the so-called ₩ " Vikings₩ " from the north. "

- 탈출 문자

  ✓ Python에서 사용되는 이스케이프 문자:

Code	content
₩'	Single Quote
₩₩	Backslash
₩n	New Line
₩r	Carriage Return
₩t	Tab
₩b	Backspace
₩f	Form Feed
₩ooo	Octal value
₩xhh	Hex value

3. 변수유형 – 문자열 - 문자열가공 – 내장 문자열함수

내장함수	설명
count()	문자(열) 개수 세기
find()	문자 위치 알려주기 문자열 안에 존재하지 않는 문자를 찾으면 -1 반환
join()	문자열 결합
'separator'.join()	문자열을 결합하는데 사용되는 separator
upper()	소문자를 대문자로 바꾸기
isupper()	해당 문자열이 대문자 인지 확인
lower()	대문자를 소문자로 바꾸기
islower()	해당 문자열이 소문자인지 확인
capitalize()	첫 문자를 대문자로 변환
isalpha()	문자로만 구성된 문자열에 True 반환
isalnum()	문자와 숫자로 구성된 경우 True를 반환
isdecimal()	숫자로만 구성된 경우 True를 반환

내장함수	설명
isspace()	공백인 경우 True를 반환
swapcase()	대문자는 소문자로, 소문자는 대문자로 변환
title()	각 단어의 첫 문자를 대문자로 변환
istitle()	각 단어의 첫 문자가 대문자인 경우 True를 반환
lstrip()	왼쪽 공백 지우기
rstrip()	오른쪽 공백 지우기
strip()	양쪽 공백 지우기
replace('바뀌게 될 문자열', '바꿀 문자열')	문자열 바꾸기
split('separator')	separator를 기준으로 문자열 나누기
split()	공백 문자를 기준으로 문자열을 분리
partition()	문자열을 partition() 메서드의 첫 번째 파라미터로 분리하여 그 앞 부분(prefix), partition 분리자(separator), 뒷부분 (suffix) 등 3개의 값 반환
startwith('문자열')	특정 문자열로 시작하면 True를 반환
endwith('문자열')	특정 문자열로 끝나면 True를 반환

3. 변수유형 – list

- 단일 변수에 여러 항목을 저장.

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi",
"melon", "mango"]
print(thislist)
```

- 항목은 순서가 지정되고 변경 가능하며 중복 값을 허용.
- list 항목은 모든 데이터 유형 가능 .

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
list4 = ["abc", 34, True, 40, "male"]
list5 = [1, 'two', 3.14, [0, 3, 5]]
```

- 유형확인.

```
print(type(mylist))
```

- list() 생성자로 생성.

```
l = list(("apple", "banana", "cherry"))
```

- 항목 접근방법

✓ 인덱스 번호를 참조하여 액세스.
- Thislist[2]

✓ 끝에서부터 인덱싱.
- print(thislist[-1])

✓ 범위로 인덱싱
- thislist[2:4]

✓ 첫번째부터 참조
- thislist[:4]

✓ 마지막부터 참조
- thislist[-4:]

연습

목록에 "apple"이 있는지 확인.

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

3. 변수유형 – list – 항목값 다루기

- 특정 항목 변경:
  thislist[1] = "nut"

- 범위 이용하여 변경
  thislist[1:3] = ("nut", "watermelon"]

- 대체대상보다 더 많은 항목을 할당하면?
  thislist = ["apple", "banana", "cherry"]
  thislist[1:2] = ["blackcurrant", "watermelon"]

- 대체대상보다 적은 수의 항목을 할당하면?
  thislist = ["apple", "banana", "cherry"]
  thislist[1:3] = ["watermelon"]

- 항목 삽입
  thislist = ["apple", "banana", "cherry"]
  thislist.insert(2, "watermelon")

- 항목 추가
  thislist = ["apple", "banana", "cherry"]
  thislist.append("orange")

- 목록 확장
  tropical = ["mango", "pineapple", "papaya"]
  thislist.extend(tropical)

- 모든 이터러블 추가
  ✓ 반복 가능한 개체(튜플, 집합, 사전 등)를 추가.

  thislist = ["apple", "banana", "cherry"]
  thistuple = ("kiwi", "orange")
  thislist.extend(thistuple)

- 지정된 항목 제거
  thislist = ["apple", "banana", "cherry"]
  thislist.remove("banana")

- 지정된 인덱스 제거
  thislist = ["apple", "banana", "cherry"]
  thislist.pop(1)

3. 변수유형 – list – loop 이용한 참조

- for 이용하여 모든 항목을 참조.

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

- Index 번호 이용

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

- while 루프 사용

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
```

- List Comprehension 이용

```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

3. 변수유형 – list 기타 기능

- list.sort()

✓ 목록을 정렬 하는 메서드.

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()

thislist = [100, 50, 65, 82, 23]
thislist.sort()
thislist.sort(reverse = True)

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
```

- 정렬 기능 사용자 정의

✓ 예) 숫자가 50에 얼마나 가까운지를 기준으로 정렬

```
def myfunc(n):
  return abs(n - 50)

thislist = [100, 50, 65, 82, 23]
thislist.sort(key = myfunc)
```

- list.reverse() : 조건없이 역순으로,

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
```

- list.copy()

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
```

- copy() 대신 list() 이용도 가능

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
```

- 두 목록 합치기

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
list3 = list1 + list2

list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
```

3. 변수유형 – tuple

- 순서가 있고 변경할 수 없는 데이타집합 .
- 중복값 허용함.
- 튜플 항목은 모든 데이터 유형 가능.

```
thistuple = ("apple", "banana", "cherry")
thistuple = "apple", "banana", "cherry"
print(thistuple)

print(len(thistuple))
```

- 하나의 항목으로 튜플 만들기
  - 항목 뒤에 쉼표를 추가해야 함.

```
thistuple = ("apple",)
print(type(thistuple))

thistuple = ("apple") #NOT a tuple
print(type(thistuple))

tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
tuple1 = ("abc", 34, True, 40, "male")
```

- 튜플 항목 참조

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi",
"melon", "mango")
print(thistuple[1])
print(thistuple[-1])
print(thistuple[2:5])
print(thistuple[:4])
print(thistuple[2:])
print(thistuple[-4:-1])
```

- 아이템이 존재하는 지 확인하는 구문

```
thistuple = ("apple", "banana", "cherry")

if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

3. 변수유형 – tuple - 다루기

- 튜플 변경방법
  ✓ 튜플은 변경할 수 없음
  ✓ 즉, 생성되면 항목을 변경, 추가 또는 제거할 수 없다.

- 해결 방법
  ✓ tuple → list → 변경 → tuple

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
```

✓ 튜플 + 튜플 방법

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y
```

- 튜플 풀기
  ✓ 튜플을 생성한 후 다시 변수로 추출

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
```

- * 사용(임의의 갯수)
  ✓ 변수의 수가 값의 수보다 적은 경우
  ✓ 변수 이름에 '*' 추가할 수 있다.

✓ 나머지 값을 "빨간색"이라는 목록으로 할당.

```
fruits = ("apple", "banana", "cherry", "strawberry",
"raspberry")
(green, yellow, *red) = fruits
```

```
fruits = ("apple", "mango", "papaya", "pineapple",
"cherry")
(green, *tropic, red) = fruits
```

3. 변수유형 – tuple - 다루기

- 튜플 변경방법
  ✓ 튜플은 변경할 수 없음
  ✓ 즉, 생성되면 항목을 변경, 추가 또는 제거할 수 없다.

- 해결 방법
  ✓ tuple → list → 변경 → tuple

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
```

✓ 튜플 + 튜플 방법

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y
```

- 튜플 풀기
  ✓ 튜플을 생성한 후 다시 변수로 추출

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
```

- * 사용(임의의 개수 지정가능)
  ✓ 변수의 수가 값의 수보다 적은 경우
  ✓ 변수 이름에 '*' 추가할 수 있다.

✓ 나머지 값을 "빨간색"이라는 목록으로 할당.

```
fruits = ("apple", "banana", "cherry", "strawberry",
"raspberry")
(green, yellow, *red) = fruits
```

```
fruits = ("apple", "mango", "papaya", "pineapple",
"cherry")
(green, *tropic, red) = fruits
```

3. 변수유형 – tuple - 다루기

- 반복을 통한 참조방법

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

- range() 사용 방법

```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
  print(thistuple[i])
```

- while 사용 방법

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

- 튜플 합치기

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
```

- 튜플 곱하기

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

- 튜플 메서드

```
count()    특정 값의 빈도
index()    특정 값의 위치
```

3. 변수유형 – set

- 단일 변수에 여러 항목을 저장.
- 항목은 순서가 없고, 변경(update)할 수 없으며, 중복 값을 허용하지 않음.
- 집합은 중괄호로 생성.

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

- 무순 → 인덱스나 키로 참조할 수 없음.

```
thisset = {"apple", "banana", "cherry", "apple"}
print(thisset)
print(len(thisset))
```

- 항목 설정 - 데이터 유형

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
set1 = {"abc", 34, True, 40, "male"}
```

- 항목 참조방법

✓ 인덱스나 키 사용불가.
✓ For, in 사용하여 참조 가능

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)

print("banana" in thisset)
```

- 항목 추가

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

- 세트 추가

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
```

- 모든 이터러블 추가(list 예)

```
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]
thisset.update(mylist)
```

3. 변수유형 – set

- 아이템 제거

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")

thisset = {"apple", "banana", "cherry"}
x = thisset.pop()  # 마지막 항목

thisset = {"apple", "banana", "cherry"}
thisset.clear()
print(thisset)

thisset = {"apple", "banana", "cherry"}
del thisset
print(thisset)
```

- set 결합

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3)

set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set1.update(set2)  # no return
Print(set1)
```

3. 변수유형 – dict

- 키:값 쌍으로 데이터 값을 저장하는 데 사용.
- 순서가 지정되고(버전 3.7부터), 변경 가능, 중복 안됨.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

- 중복 값은 기존 값을 덮어쓴다.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}

print(len(thisdict))
```

- 데이터 유형

```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

- 항목 액세스

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car["model"]
x = car.get("model")
```

- 키 가져오기

```
x = car.keys()
print(x) #before the change
car["color"] = "white"
print(x) #after the change
```

- 값 가져오기

```
x = car.values()
print(x) #before the change
```

- 키:값 쌍 목록 가져오기

```
x = car.items()
```

- dict에 "model"이 있는지 확인.

```
if "model" in thisdict:
  print("Yes, 'model' exists...")
```

3. 변수유형 – dict

- 값 변경

```
thisdict["year"] = 2018
```

✓ update() : iterable 이용 가능.

```
thisdict.update({"year": 2020})
```

- 항목 추가

```
thisdict["color"] = "red"
thisdict.update({"color": "red"})
```

- 항목 제거

```
car.pop("model")
car.clear()
del car["model"]
```

- 사전 복사

```
dict2 = car
mydict1 = car.copy()
mydict2 = dict(car)
print(mydict)
```

- loop 참조

```
for x in car:  # 모든 키 이름, 값 인쇄
  print(x)
  print(car[x])

for x in car.values():
  print(x)
for x in thisdict.keys():
  print(x)
for x, y in thisdict.items():
  print(x, y)
```

- 중첩된 dict

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
```

3. 변수유형 – 부울형

- True, False 유형.
- 두 값의 표현식을 평가하여 참 거짓을 알고싶을 때 사용.

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

- 조건이 있는지 여부에 따라 메시지 인쇄 True 또는 False

```
a = 200
b = 33

if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

- 문자열과 숫자도 평가함.
- 빈문자열, 0 제외하고 대부분 True로 평가됨.
- 모든 목록, 튜플, 집합 및 사전은 빈 항목을 제외하고 True.

```
print(bool("Hello"))
print(bool(15))

bool("abc")
bool(123)
bool(["apple", "cherry", "banana"])
```

- False를 반환데이타 값들.

```
bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})
```

3. 변수유형 – 파이썬 연산자

- 산술 연산자

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

- 할당 연산자 : 변수에 값을 할당하는 데 사용.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

- 비교 연산자

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

- 논리 연산자 : 조건문을 결합하는 데 사용.

Operator	Example
and	x < 5 and x < 10
or	x < 5 or x < 4
not	not(x < 5 and x < 10)

3. 변수유형 – 파이썬 연산자

- is, 파이썬 동일성 연산자
- 개체 비교. 메모리 위치가 동일해야 함.

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is z)  # True
print(x is y) # False
print(x == y) # True, 같은 내용
cf. id(x)
```

- 파이썬 멤버십 연산자

```
x = ["apple", "banana"]
print("banana" in x)
```

- 파이썬 비트 연산자

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	
>>	Signed right shift	

4. 흐름제어 - 조건 및 if 문

- 조건 및 If 문
- 들여쓰기를 사용하여 코드의 범위를 정의

```
a = 33
b = 200
if b > a:
  print("b is greater than a")

if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

- 한 줄 if 문:

```
if a > b: print("a is greater than b")
print("A") if a > b else print("B")
print("A") if a > b else print("=") if a == b else print("B")
```

- 복합조건문

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

- 중첩된 if

```
x = 41
if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

- Pass 이용

```
a = 33
b = 200

if b > a:
  pass
```

4. 흐름제어 - while loop

- while loop

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

- break 문

✓ while 조건이 true인 경우에도 루프를 나옴.

```
i = 1

while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

- continue

✓ 현재 반복을 중지하고 다음 반복을 계속.

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

- else 문

✓ 조건이 더 이상 참이 아닐 때 코드 블록을 한 번 실행.

✓ break 는 조건이 참인 상태에서 나오는 차이 있음.

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

4. 흐름제어 - for loop

- list, 튜플, set 등의 각 항목에 대해 한 번씩 일련의 명령문을 실행.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)

for x in "banana":    # 문자열 반복
  print(x)
```

- Break 사용

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

- Continue 사용

```
fruits = ["apple", "banana", "cherry"]

# 'banana'를 인쇄하지 않음.
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

- range() 함수

✓ 지정된 횟수만큼 코드 세트를 반복

```
for x in range(6):
  print(x)

for x in range(2, 6):
  print(x)

for x in range(2, 30, 3):
  print(x)
```

- 중첩 구조

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

- pass

```
for x in [0, 1, 2]:
  pass
```

4. 흐름제어 - 반복자(iterator)

- 모든 값을 순회할 수 있는 반복될 수 있는 개체
- 목록, 튜플, 사전 및 집합은 모두 반복 가능한 개체
- iter()를 이용하여 반복자를 생성, next()를 이용하여 단위 객체 추출.

- for루프를 사용하여 반복

```
mytuple = ("apple", "banana", "cherry")
for x in mytuple:
  print(x)
```

- iter 사용하여 반복

```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)
print(next(myit))
print(next(myit))

mystr = "banana "    # 문자열도 가능
myit = iter(mystr)

print(next(myit))
print(next(myit))
```

- enumerate

✓ 값과 인덱스를 반복해야할 때 사용

```
L = [2, 4, 6, 8, 10]
for i in range(len(L)):
    print(i, L[i])
```

➔→

```
for i, val in enumerate(L):
    print(i, val)
```

- map

✓ 반복자의 값에 함수 적용

```
# 10까지 수를 제곱
square = lambda x: x ** 2
for val in map(square, range(10)):
    print(val, end=' ')
```

- filter

✓ 함수가 참으로 평가한 값만 통과

```
# 짝수만 출력
is_even = lambda x: x % 2 == 0
for val in filter(is_even, range(10)):
    print(val, end=' ')
```

4. 흐름제어 - 반복자(iterator)

- zip
  - ✓ 여러개의 리스트를 동시에 반복

```
L = [2, 4, 6, 8, 10]
R = [3, 6, 9, 12, 15]
for lval, rval in zip(L, R):
    print(lval, rval)

L1 = (1, 2, 3, 4)
L2 = ('a', 'b', 'c', 'd')

z = zip(L1, L2)
print(*z)
```

  - ✓ 여러 개의 반복자를 묶을 수 있음.
  - ✓ zip → zip : unzip

```
z = zip(L1, L2)
new_L1, new_L2 = zip(*z)
print(new_L1, new_L2)
```

4. 흐름제어 – list comprehension

- 리스트를 만드는 for 루프를 짧고 읽기 쉬운 한 라인으로 압축하는 기술

✓ 구문 : [expr for var in iterable]

```
L = []
for n in range(12):
    L.append(n ** 2)
L
```

→ →

```
[n ** 2 for n in range(12)]
```

✓ 중복반복

```
[(i, j) for i in range(2) for j in range(3)]
```

✓ 조건부 반복

```
[val for val in range(20) if val % 3 > 0]
```

✓ set comprehension -- 중복제거

```
{n**2 for n in range(12)}
```

✓ dict comprehension

```
{n:n**2 for n in range(6)}
```

4. 흐름제어 – generator

- 리스트는 값들의 모음인 반면, 제너레이터는 값을 만들어내는 방식.
  - ✓ 리스트는 실제 값들을 생성하므로 메모리차지하나, 제너레이터는 방법만 정의하므로 메모리 소모 없음.
  - ✓ 메모리를 효율적으로 사용할 수 있고 계산 비용도 절감.
  - ✓ 리스트의 크기는 가용 메모리 범위로 제한되지만 제너레이터의 크기는 제한이 없음.

```
(n ** 2 for n in range(12))
```

  - ✓ 출력하려면 리스트 생성자함수 이용.

```
G = (n ** 2 for n in range(12))
list(G)

G = (n ** 2 for n in range(12))
for val in G:
    print(val, end=' ')
```

  - ✓ 제너레이터 표현식은 한번만 사용됨.

```
G = (n ** 2 for n in range(12))
list(G)
list(G)
```

  - ✓ 또한 현재위치를 기억하고 있음.
  - ✓ 배치 작업과정 중 이전에 중단됐던 지점을 기억

```
G = (n**2 for n in range(12))
for n in G:
    print(n, end=' ')
    if n > 30: break

print("₩nDo something else.")
for n in G:
    print(n, end=' ')
```

- 제너레이터 만드는 방법

```
G1 = (n ** 2 for n in range(12))
```

  - ✓ - 복잡한 구조인 경우 yield 사용

```
def gen():
    for n in range(12):
        yield n ** 2

G2 = gen()
print(*G1)
print(*G2)
```

4. 흐름제어 – generator – 에라토스테네스의 체

- Sieve of Eratosthenes
- 소수를 찾는 알고리즘
- 2부터 시작하여 자신의 배수를 모두 없애 나가는 방식.

```
# 2 ~ 40범위 소수 찾기
L = [n for n in range(2, 40)]
print(L)

# 첫 번째 값(2)의 모든 배수 제거
L = [n for n in L if n == L[0] or n % L[0] > 0]
print(L)

# 두 번째 값(3)의 모든 배수 제거
L = [n for n in L if n == L[1] or n % L[1] > 0]
print(L)

# 세 번째 값(5)의 모든 배수 제거
L = [n for n in L if n == L[2] or n % L[2] > 0]
print(L)
```

- generator 를 이용했을 때

✓ 배치 작업과정 중 이전에 중단됐던 지점을 기억

```
def gen_primes(N):
    """N까지의 소수를 생성합니다"""
    primes = set()
    for n in range(2, N):
        if all(n % p > 0 for p in primes):
            primes.add(n)
            yield n

print(*gen_primes(100))
```

- https://ko.wikipedia.org/wiki/%EC%97%90%EB%9D%BC%ED%86%A0%EC%8A%A4%ED%85%8C%EB%84%A4%EC%8A%A4%EC%9D%98_%EC%B2%B4

5. 함수(function)

- 함수는 호출될 때만 실행되는 코드 블록

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
```

- return : 반환 값

```
def my_function(x):
  return 5 * x

print(my_function(3))
```

- 기본 매개변수 값

✓ 인수 없이 함수를 호출하면 기본값 사용.

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("India")
my_function()
my_function("Brazil")
```

- 목록을 인수로 전달

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

- recursive : 재귀

✓ 정의된 함수가 스스로를 호출.
✓ 종료되지 않는 함수나 과도한 양의 메모리 사용 주의

```
def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(k, result)
  else:
    result = 0
  return result

tri_recursion(6)
```

5. 함수(function)

- *args와 **kwargs
  - ✓ 받을 매개변수 개수가 가변적일 때.
  - ✓ * : 임의 개수의 일반 인수.
  - ✓ ** : 임의 개수의 키워드 인수.
  - ✓ 순서는 일반 인수 먼저 표시, 키워드 인수는 뒤에 표시해야 함.

```
def catch_all(*args, **kwargs):
    print( " args = " , args)
    print( " kwargs =  " , kwargs)

catch_all(1, 2, 3, a=4, b=5)
catch_all('a', keyword=2)
```

- ✓ 호출시에도 *, ** 사용가능.

```
inputs = (1, 2, 3)
keywords = {'pi': 3.14}

catch_all(*inputs, **keywords)
```

- 람다 함수
  - ✓ 여러 인수를 사용할 수 있지만 표현식은 하나만 가지는 익명 함수.
  - ✓ lambda arguments : expression

```
x = lambda a, b : a * b
print(x(5, 6))
```

- Lambda 함수 사용예

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)  # 인수를 2배로 만드는 함수 생성
print(mydoubler(11))

#================================
data = [{'first':'Guido', 'last':'Van Rossum', 'YOB':1956},
        {'first':'Grace', 'last':'Hopper',      'YOB':1906},
        {'first':'Alan',  'last':'Turing',      'YOB':1912}]

# 이름 기준으로 정렬
sorted(data, key=lambda item: item[ ' first ' ])
# 생년기준으로 정렬
sorted(data, key=lambda item: item['YOB'])

def f(item): return item['YOB']
```

5. 함수(function) - 모듈(module)

- 모듈은 코드 라이브러리.
- 모듈 만들기
  - ✓ py 확장자를 가진 파일에 원하는 코드를 저장
  - ✓ == in mymodule.py ==

```
def greeting(name):
  print("Hello, " + name)
```

  - ✓ == 호출 파일에서 ==

```
import mymodule
mymodule.greeting("Jonathan")
```

- 모듈의 변수
  - ✓ 함수뿐 아니라 모든 유형(배열, 사전, 개체 등)의 변수도 포함 가능
  - ✓ == In mymodule.py ==

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

  - ✓ == 호출 파일에서 ==

```
import mymodule
a = mymodule.person1["age"]
print(a)
```

- 모듈 이름 바꾸기

```
import numpy as np
Import matplotlib.pyplot as plt
```

- 내장 모듈
  - ✓ 설치하지 않고 바로 사용할 수 있는 모듈

```
import platform
x = platform.system()
print(x)
```

- dir() 함수 사용
  - ✓ 모듈의 모든 함수 이름(또는 변수 이름)을 나열하는 내장 함수

```
dir(platform)
```

- 모듈에서 부분 가져오기

```
import matplotlib as mpl
dir(mpl)

import matplotlib.pyplot as plt
dir(plt)
```

6. 클래스(class)

- 클래스 만들기

✓ __init__() : 개체 생성시 항상 실행되는 함수

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

- 개체 속성 수정

```
p1.age = 40
```

- 개체 속성 삭제

```
del p1.age
```

- 객체 삭제

```
del p1
```

- pass

```
class Person:
  pass
```

- self

✓ 현재 클래스의 인스턴스에 대한 참조, 및 클래스에 속한 액세스 변수로 사용.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

✓ 대신 원하는 이름을 사용할 수도 있음.

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

6. 클래스(class) - 상속

- 상속
  - ✓ 다른 클래스의 모든 메서드와 속성을 상속하는 클래스를 정의.
  - ✓ 부모 클래스 : 상속하는 클래스.
  - ✓ 자식 클래스 : 상속받는 클래스.

- 부모 클래스

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

x = Person("John", "Doe")
x.printname()
```

- < 자식 클래스 >
  - ✓ 자식 클래스를 만들 때 상속받을 부모 클래스를 매개 변수로 하여 생성.

```
class Student(Person):
  def __init__(self, fname, lname):
    #add properties etc.
```

- __init__()
  - ✓ __init__()함수를 추가하면 자식 클래스는 부모의 __init__()기능을 상속하지 않음.
  - ✓ 부모 __init__() 함수를 상속하려면 자식의 __init__() 에 추가.

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)

x = Student("Mike", "Olsen")
x.printname()
```

6. 클래스(class) - 상속

- super() 함수 사용

✓ 자식 클래스가 부모로부터 모든 메서드와 속성을 상속 하도록 하는 함수.

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)

x = Student("Mike", "Olsen")
x.printname()
```

6. 클래스(class) – 속성/메소드 추가

- 클래스의 속성으로 추가

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    self.graduationyear = 2019

x = Student("Mike", "Olsen")
print(x.graduationyear)
```

- 2. 객체 생성시 추가

```
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

x = Student("Mike", "Olsen", 2019)
print(x.graduationyear)
```

- 메소드 추가

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class
of", self.graduationyear)

x = Student("Mike", "Olsen", 2019)
x.welcome()
```

7. 기타기능 - 파일 처리

- 파일 열기 : open()

```
f = open("demofile.txt", "rt")
```

- 파일을 여는 네 가지 모드
  - ✓ "r"- 읽기 – 기본값. 파일이 없으면 오류.
  - ✓ "a" - 추가 - 존재하지 않는 경우 파일을 생성.
  - ✓ "w" - 쓰기 - 파일이 없으면 생성.
  - ✓ "x" - 만들기 - 지정된 파일을 만들고 파일이 있으면 오류.

- 바이너리 모드 / 텍스트 모드
  - ✓ "t"- 텍스트 - 기본값입니다
  - ✓ "b" - 바이너리 - 예: 이미지

- 파일 닫기

```
f.close()
```

- 파일 읽기
  - ✓ == 현위치에 demofile.txt 생성 후 ==

```
f = open("demofile.txt", "r")
Print(f.read())
```

  - ✓ 파일의 일부만 읽기

```
pint(f.read(5))  # 첫 번째 5자를 반환
print(f.readline())  # 파일의 한 줄을 읽기
```

  - ✓ # 파일을 한 줄씩 반복읽기

```
for x in f:  print(x)
```

- 파이썬 파일 쓰기

```
f = open("demofile.txt", "a")
f.write("Now the file has more content!")
f.close()
f = open("demofile3.txt", "w")
f.write("new content!")
f.close()
```

- 파일 삭제

```
import os
if os.path.exists("demofile.txt"):  # 파일 있는지 확인후 삭제
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

7. 기타기능 - 날짜/시간

- 날짜정보 : 년, 월, 일, 시, 분, 초 및 마이크로초.

```
import datetime

x = datetime.datetime.now()
print(x)
```

- 요일의 연도와 이름을 반환

```
import datetime
x = datetime.datetime.now()
print(x.year)
print(x.strftime("%A"))
```

- 날짜 객체 생성

```
import datetime
x = datetime.datetime(2020, 5, 17)  # datetime(년, 월, 일)
print(x)
```

- strftime() 메서드

✓ datetime개체를 읽을 때 포맷하는 방법.

✓ 월 이름 표시:

```
import datetime
x = datetime.datetime(2018, 6, 1)
print(x.strftime("%B"))
```

strftime() format

Directive	Description	Example
%a	Weekday	Wed
%A	Weekday	Wednesday
%w	Weekday number 0-6,	0 is Sunday
%d	Day of month 01-31	31
%b	Month name	Dec
%B	Month name	December
%m	Month number	12
%y	Year	18
%Y	Year	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59 41	
%S	Second 00-59 08	
%f	Microsecond 548513	
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year	365
%U	Week number of year	52
%W	Week number of year	52
%c	date and time Mon Dec 31 17:41:00 2018	
%C	Century	20
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character %	
%G	ISO 8601 year 2018	
%u	ISO 8601 weekday	1
%V	ISO 8601 weeknumber	01

7. 기타기능 - 수학 기능

- 수학 작업을 수행할 수 있는 광범위한 수학 함수 세트 내장.
- https://docs.python.org/3/library/functions.html
- https://docs.python.org/3/library/math.html
- https://docs.python.org/ko/3/library/math.html

- 내장함수 이용

```
x = min(5, 10, 25)
y = max(5, 10, 25)
x = abs(-7.25)
x = pow(4, 3)
```

- math 모듈 이용

```
import math

x = math.sqrt(64)
x = math.ceil(1.4)
y = math.floor(1.4)
x = math.pi
```

- 사용자 입력

```
username = input("Enter username:")
print("Username is: " + username)
```

7. 기타기능 - 오류 다루기

- try-except
  - ✓ try 블록 : 오류 코드 블록을 테스트 목적 구문.
  - ✓ Except블록 : 오류를 처리 하는 블록.
  - ✓ finally블록 : 오류여부와 관계없이 실행되는 코드블록.

- try-except-finally 구문

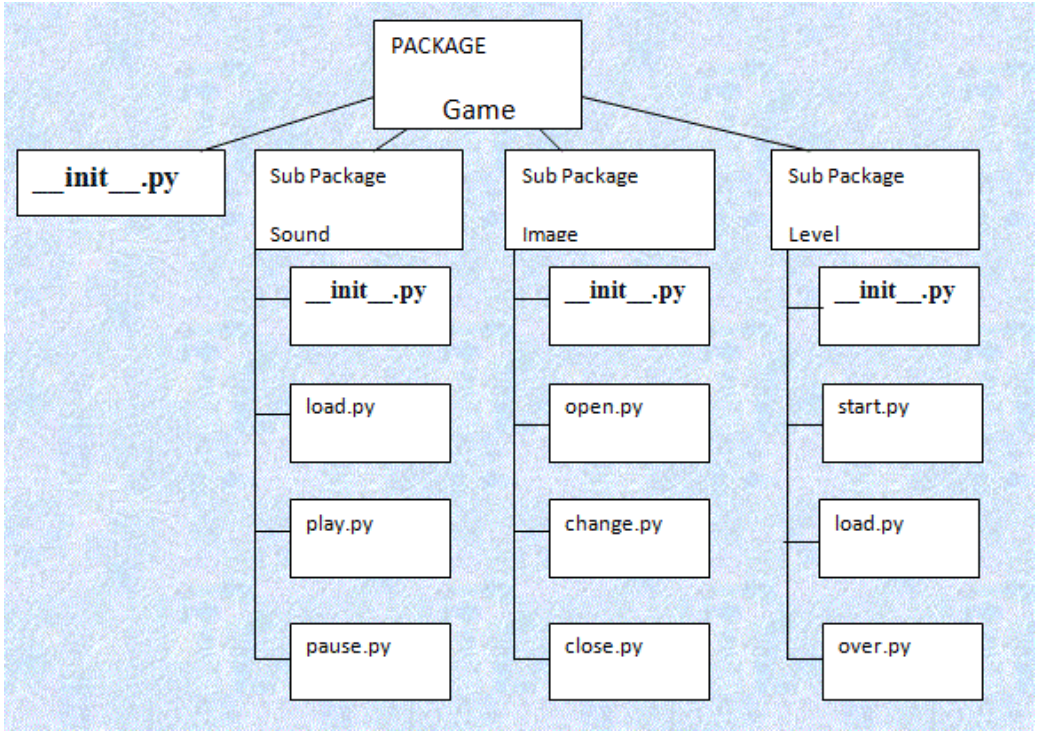
```
try:
    print(x)
except:
    print("Something went wrong")
finally:
    print("The 'try except' is finished")

print(x)
```

- 중첩 구문

```
try:
  f = open("demofile.txt")
  try:
    f.write("Lorum Ipsum")
  except:
    print("Something went wrong when writing to the file")
  finally:
    f.close()
except:
  print("Something went wrong when opening the file")
```

7. 기타기능 – pip(pip installs packages)

- 파이썬 표준저장소인 PyPI(http://pypi.python.org/)로부터 필요패키지를 다운로드 및 설치해줌
- 패키지에는 모듈에 필요한 모든 파일이 포함되어 있음.
- 보통 파이썬이 설치될 때 자동설치됨.



- PIP 버전 확인:

  > pip --version

- PIP 설치(업그레이드)

  > python –m pip install pip

- 패키지 설치

  > pip install camelcase

- 패키지 사용

```
import camelcase
c = camelcase.CamelCase()
txt = "hello world"
print(c.hump(txt))
```

- 패키지 제거

  > pip uninstall camelcase

- 패키지 나열

  > pip list