파이썬 기초 강의

파이썬 소개

파이썬은 현재 전 세계에서 가장 인기 있는 프로그래밍 언어입니다. 파이썬은 많은 대학에서 필수 과목으로 학습되고 있으며, 일반 목적 프로그래밍 언어(다양한 영역에서 사용할 수 있는 프로그래밍 언어)로 받아들여지고 있습니다. 파이썬은 스크립팅(PC 어플 제어), PC 어플 개발, 웹 개발, 머신 러닝, 사물 인터넷, 데이터 사이언스, 그 외 다양한 분야에서 사용됩니다. 다음은 대표적인 사이트 3 군데에서 발표한 프로그래밍 언어 인기 순위입니다.

Programming Language	Ratings
Python	16.36%
C	16.26%
C++	12.91%
Java	12.21%
C#	5.73%

<TIOBE 지수>

Language	Share
Python	27.93 %
Java	16.78 %
JavaScript	9.63 %
C#	6.99 %
C/C++	6.9 %

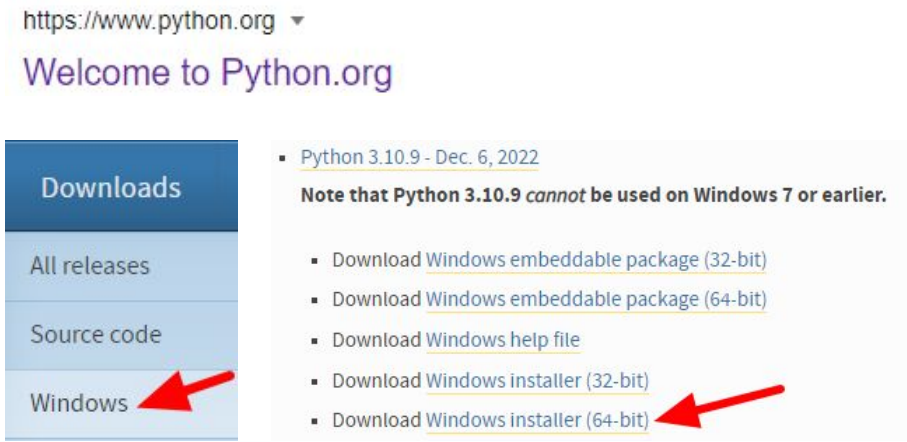
<PYPL 지수>

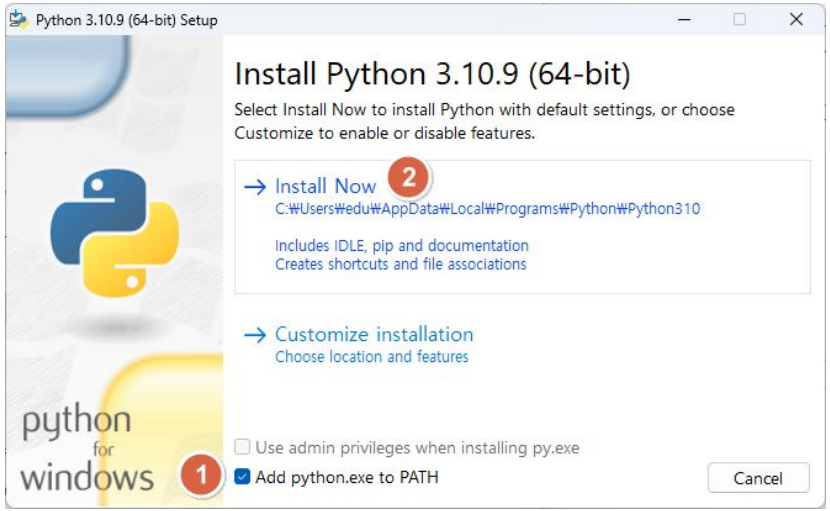
Programming Language	Percentage (YoY Change)
Python	16.756% (+0.793%)
Java	11.005% (-1.988%)
C++	10.254% (+3.563%)
Go	9.657% (+1.445%)
JavaScript	9.286% (-3.690%)

<GITHUB 지수>

파이썬 설치하기

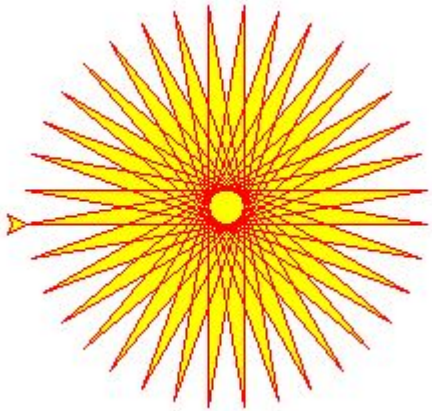
3.10.9를 설치합니다.





파이썬 맛보기 : turtle

```
>>> from turtle import *
>>> color('red','yellow')
>>> begin_fill()
>>> while True:
...     forward(200)
...     left(170)
...     if abs(pos())<1:
...         break
...
...
>>> end_fill()
>>> clear()
>>> bye()
```



파이썬 훑어보기

파이썬의 기본 재료 : 값

```
>>> 3
3
>>> 3.14
3.14
>>> '3.14'
'3.14'
```

값으로 하는 일 : 사칙 연산

```
>>> 3+4
7
>>> 3-4
-1
>>> 3*4
12
>>> 3/4
0.75
```

값으로 하는 일 : 몫과 나머지, 거듭 제곱

```
>>> 3//4
0
>>> 3%4
3
>>> 3**4
81
```

값으로 하는 일 : 비교 연산

```
>>> 3>4
False
>>> 3>=4
False
>>> 3<4
True
>>> 3<=4
True
>>> 3==4
False
>>> 3!=4
True
```

값으로 하는 일 : 논리 연산

```
>>> 3>4 and 5>6
False
>>> 3<4 and 5<6
True
>>> 3>4 or 5>6
False
>>> 3<4 or 5>6
True
>>> not 3<4
False
>>> not 3>4
True
```

값으로 하는 일 : 문자열 연산

```
>>> 'hello'+'world'
'helloworld'
>>> 'hello'*3
'hellohellohello'
```

값에 이름 붙이기 : 변수와 할당

```
>>> a=3
>>> b=4
>>> a+b
7
>>> c=a+b
>>> c
7
```

판단하기 1 : if

만약 나이가 19살 이상이면 성인이야.

```
>>> age=23
>>> if age>=19:
...     'adult'
...
...
'adult'
```

판단하기 2 : if-else

만약 나이가 19살 이상이면 성인이고 아니면 미성년자야.

```
>>> age=23
>>> if age>=19:
...     'adult'
... else :
...     'minor'
...
...
'adult'
```

판단하기 3 : if-elif-else

MZ 세대 : M 1981~1996, Z 1997~2012

만약 태어난 해가 1997~2012년 사이이면 Z세대이고

1981~1996년 사이이면 M세대야

```
>>> birth_year = 2004
>>> if 1981 <= birth_year <= 1996:
...     'M generation'
... elif 1997 <= birth_year <= 2012:
...     'Z generation'
...
...
'Z generation'
```

동작 반복하기 : while

파이썬 수준이 10이 될때까지 공부해!

```
>>> python_level=0
>>> while python_level<10:
...     'study python'
...     python_level += 1
...
...
'study python'
'study python'
'study python'
'study python'
'study python'
'study python'
'study python'
'study python'
'study python'
'study python'
>>> python_level
10
```

동작 반복하기 : while

파이썬 수준이 10이 될때까지 공부해!

```
>>> python_level = 0
>>> while python_level<10:
...     'study python'
...     python_level += 1
...     python_level
...
...
```

```
'study python'
1
'study python'
2
'study python'
3
'study python'
4
'study python'
5
'study python'
6
'study python'
7
'study python'
8
'study python'
9
'study python'
10
```

동작 반복하기 2 : while-if-break

```
>>> while True:
...     'study python'
...     python_level += 1
...     if python_level>=10:
...         break
...
...
```

```
'study python'
'study python'
'study python'
'study python'
'study python'
'study python'
'study python'
'study python'
'study python'
'study python'
```

값을 담는 주머니 : 리스트

계란 담기 : 직접 담기

‘계란을 한 바구니에 담지 마라’

우리는 담아보겠습니다.

```
>>> eggs=['egg0','egg1','egg2','egg3']
>>> eggs
['egg0', 'egg1', 'egg2', 'egg3']
```

값을 담는 주머니 : 리스트

계란 담기 : 하나씩 담기

```
>>> eggs=[]
>>> eggs.append('egg0')
>>> eggs.append('egg1')
>>> eggs.append('egg2')
>>> eggs.append('egg3')
>>> eggs
['egg0', 'egg1', 'egg2', 'egg3']
```

리스트 처리하기 : for-in

계란 확인하기

```
>>> eggs=['egg0','egg1','egg2','egg3']
>>> for egg in eggs:
...     egg
...
...

'egg0'
'egg1'
'egg2'
'egg3'
```

리스트 처리하기 : for-in

계란 삶기

```
>>> eggs=['egg0','egg1','egg2','egg3']
>>> for egg in eggs:
...     'boil ' + egg
...
...
'boil egg0'
'boil egg1'
'boil egg2'
'boil egg3'
```

동작 묶기 : 함수

```
>>> def add(a,b):
...     return a+b
...
>>> a=1
>>> b=2
>>> c=add(a,b)
>>> c
3
```

동작 묶기 : 함수

```
>>> def is_adult(age):
...     return age >= 19
...
>>> my_age = 20
>>> is_adult(my_age)
True
```

변수 함수 묶기 : 클래스

```
>>> class Student:
...     def __init__(instance, name, age):
...         instance.name = name
...         instance.age = age
...     def get_name(instance):
...         return instance.name
...     def get_age(instance):
...         return instance.age
...     def set_age(instance, age):
...         instance.age = age
...
...
>>> student_1 = Student('Brad', 21)
>>> student_1.name
'Brad'
>>> student_1.get_name()
'Brad'
```

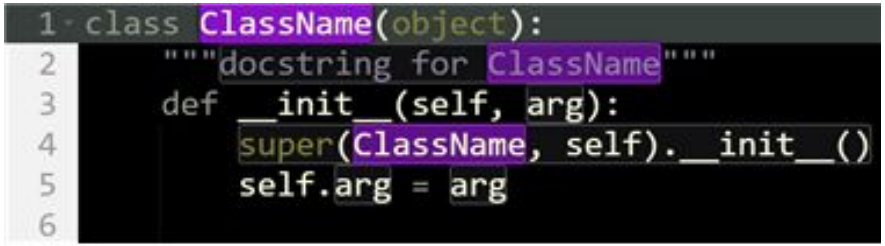
파이썬 툎아보기

def 문 정리하기

```
1 def name(arguments):
2     """description for name"""
3     # TODO: write code...
```

def 문은 이름이 있는 함수를 정의할 때 사용합니다. 그 함수는 0개 이상의 인자를 받습니다. def 문의 하위 문장은 같은 간격의 들여쓰기를 적용해 주어야 합니다.

class 문 정리하기



class 문은 어떤 목적을 갖는 변수와 함수를 묶을 때 사용합니다. class 문의 하위 문장은 같은 간격의 들여쓰기를 적용해 주어야 합니다.

try-except 문 정리하기

```
1 try:
2     # TODO: write code...
3 except Exception, e:
4     raise e
```

try-except 문은 오류 처리를 위한 제어문입니다. try 이하의 문장을 수행하는 동안에 오류가 발생하면 except 문장에서 처리합니다. try 문의 하위 문장은 같은 간격의 들여쓰기를 적용해 주어야 합니다.

with-as 문 정리하기

with-as 문은 이름에 할당된 어떤 것을 가지고 어떤 일을 수행합니다. 일반적으로 파일을 열거나 외부 자원을 얻을 때 사용합니다. with-as 문이 끝나면 자동으로 파일을 닫거나 외부 자원을 제거 또는 해제할 수 있습니다. with-as 문의 하위 문장은 같은 간격의 들여쓰기를 적용해 주어야 합니다.

예제 다시 보기 : turtle

```
>>> from turtle import *
>>> color('red','yellow')
>>> abs(pos())
0.0
>>> begin_fill()
>>> while True:
...     forward(200)
...     left(170)
...     abs(pos())
...     if abs(pos())<1:
...         break
...
...
```

```
200.0
34.862297099063255
193.9231012048832
68.66532094194501

193.923101204883
34.86229709906346
199.9999999999998
3.1166837572216805e-13

>>> abs(pos())
3.1166837572216805e-13
>>> end_fill()
>>> bye()
```

무작정 따라하기 1 : listener

```
from turtle import *

def animate():
    print('a')
    ontimer(animate, 1000)

def flick():
    print('f')

onkey(flick, 'space') # key listener
listen()
animate()

exitonclick() # GUI loop here
```

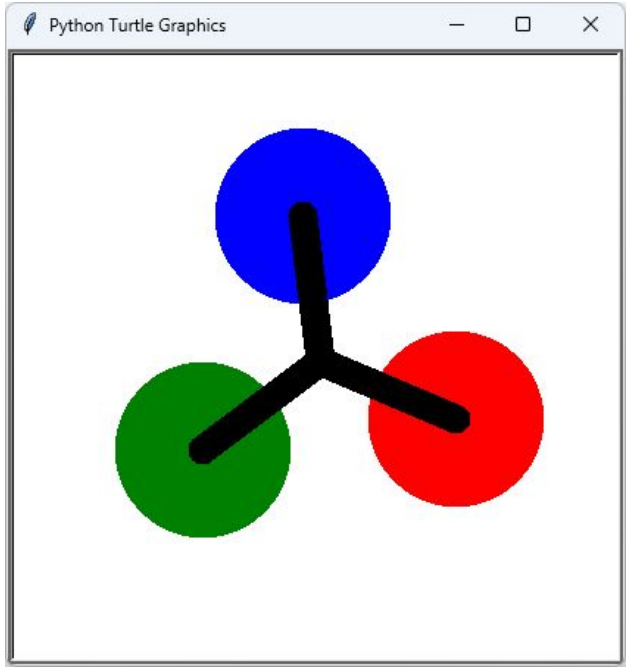
무작정 따라하기 2 : 피젯 스피너

```
from turtle import *
state = {'turn': 0}
def spinner():
    clear()
    angle = state['turn']/10
    right(angle)
    forward(100)
    dot(120, 'red')
    back(100)
    right(120)
    forward(100)
    dot(120, 'green')
    back(100)
    right(120)
    forward(100)
    dot(120, 'blue')
    back(100)
    right(120)
    update()
```

```
def animate():
    if state['turn']>0:
        state['turn']-=1

    spinner()
    ontimer(animate, 20)
def flick():
    state['turn']+=10

setup(420, 420, 370, 0)
hideturtle()
tracer(False)
width(20)
onkey(flick, 'space')
listen()
animate()
done()
```



자료형 살펴보기 1 : type

```
>>> type(3)
<class 'int'>
>>> type(3.14)
<class 'float'>
>>> type('3.14')
<class 'str'>
>>> type(True)
<class 'bool'>
>>> type([])
<class 'list'>
```

자료형 살펴보기 2 : __class__

```
>>> (3).__class__
<class 'int'>
>>> 3.14.__class__
<class 'float'>
>>> '3.14'.__class__
<class 'str'>
>>> True.__class__
<class 'bool'>
>>> [].__class__
<class 'list'>
```

자료형 살펴보기 3 : isinstance

```
>>> isinstance(3, int)
True
>>> isinstance(3.14, float)
True
>>> isinstance('3.14', str)
True
>>> isinstance(True, bool)
True
>>> isinstance([], list)
True
```

사칙 연산과 거듭 제곱 : 실수

```
>>> 3.14+2.72
5.86
>>> 3.14-2.72
0.41999999999999993
>>> 3.14*2.72
8.5408
>>> 3.14/2.72
1.1544117647058822
>>> 3.14**2.72
22.472357891492628
```

비교 연산 : 실수

```
>>> 3.14>2.72
True
>>> 3.14>=2.72
True
>>> 3.14<2.72
False
>>> 3.14<=2.72
False
>>> 3.14==2.72
False
>>> 3.14!=2.72
True
```

비교 연산은 if문 또는 while 문의 조건문으로 사용합니다.

한 문자열 접근 : indexing

```
>>> 'hello'[0]
'h'
>>> 'hello'[1]
'e'
>>> 'hello'[4]
'o'
>>> 'hello'[5]
Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
    'hello'[5]
IndexError: string index out of range
```

한 문자열 접근 : indexing

```
>>> 'hello'[-0]
'h'
>>> 'hello'[-1]
'o'
>>> 'hello'[-2]
'l'
>>> 'hello'[-5]
'h'
>>> 'hello'[-6]
Traceback (most recent call last):
  File "<pyshell#55>", line 1, in <module>
    'hello'[-6]
IndexError: string index out of range
```

부분 문자열 접근 : slicing

```
>>> 'hello'[0:1]
'h'
>>> 'hello'[0:2]
'he'
>>> 'hello'[1:3]
'el'
>>> 'hello'[:3]
'hel'
>>> 'hello'[3:]
'lo'
>>> 'hello'[:]
'hello'
>>> 'hello'[1:4:2]
'el'
>>> 'hello'[::2]
'hlo'
```

부분 문자열 접근 : slicing

```
>>> 'hello'[-3:-1]
'll'
>>> 'hello'[-3:]
'llo'
>>> 'hello'[:-3]
'he'
>>> 'hello'[-1:-3]
''
>>> 'hello'[4:2]
''
>>> 'hello'[-1:-3:-1]
'ol'
>>> 'hello'[4:2:-1]
'ol'
```

문자열 주요 함수 : len, join, split

```
>>> len('hello')
5
>>> ','.join(['hello','world'])
'hello,world'
>>> ','.join(['spiderman','superman','batman','aquaman'])
'spiderman,superman,batman,aquaman'
>>> 'hello,world'.split(',')
['hello', 'world']
>>> 'spiderman,superman,batman,aquaman'.split(',')
['spiderman', 'superman', 'batman', 'aquaman']
```

형식 문자열 1 : %

```
>>> '%s'%'hello world'
'hello world'
>>> '%d'%78
'78'
>>> '%f'%1.23456
'1.234560'
```

```
>>> '%d %x'%(78,78)
'78 4e'
>>> '%.0f'%1.23456
'1'
>>> '%.2f'%1.23456
'1.23'
>>> '%.4f'%1.23456
'1.2346'
```

형식 문자열 2 : str.format 함수

```
>>> '{}'.format('hello world')
'hello world'
>>> '{}'.format(78)
'78'
>>> '{}'.format(1.23456)
'1.23456'
```

```
>>> '{} {:x}'.format(78,78)
'78 4e'
>>> '{:.0f}'.format(1.23456)
'1'
>>> '{:.2f}'.format(1.23456)
'1.23'
>>> '{:.4f}'.format(1.23456)
'1.2346'
```

형식 문자열 3 : f-string

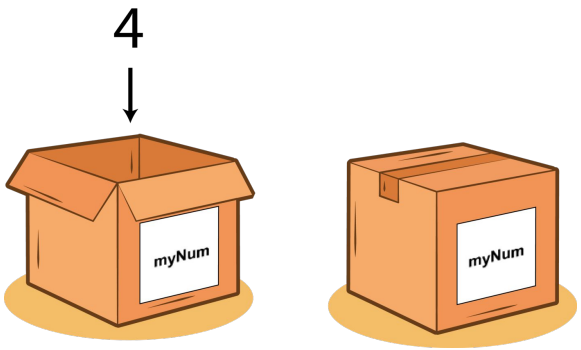
```
>>> a='hello world'
>>> b=78
>>> c=1.23456
>>>
>>> f'{a}'
'hello world'
>>> f'{b}'
'78'
>>> f'{c}'
'1.23456'
```

```
>>> a=78
>>> b=1.23456
>>>
>>> f'{a} {a:x}'
'78 4e'
>>> f'{b:.0f}'
'1'
>>> f'{b:.2f}'
'1.23'
>>> f'{b:.4f}'
'1.2346'
```

변수 톺아보기

변수, 객체 , 값 이해하기

```
>>> a=3
>>> b=4
>>> type(3)
<class 'int'>
>>> type(4)
<class 'int'>
>>> id(3)
2026110124336
>>> id(a)
2026110124336
>>> id(4)
2026110124368
>>> id(b)
2026110124368
```



값은 음식, 객체는 그릇, 변수는 이름표

if 문 톺아보기

if-elif-else 문 정리하기

```
1 if condition:
2     # TODO: write code...
```

if 문은 어떤 조건이 맞으면 어떤 동작을 수행하고자 할 때 사용합니다. 조건은 condition 부분에 넣습니다. if 문의 하위 문장은 같은 간격의 들여쓰기를 적용해 주어야 합니다.

```
1 elif condition:
2     # TODO: write code...
```

elif 문은 if 문과 함께 사용하며 이전 if 문이나 elif 문의 조건에 맞지 않고 다른 조건이 맞으면, 어떤 동작을 수행하고자 할 때 사용합니다. 조건은 condition 부분에 넣습니다. elif 문의 하위 문장은 같은 간격의 들여쓰기를 적용해 주어야 합니다.

```
1 else:
2     # TODO: write code...
```

else 문은 if 문과 함께 사용하며 이전 if 문이나 elif 문의 조건에 맞지 않으면 나머지 조건에 대해, 어떤 동작을 수행하고자 할 때 사용합니다. 나머지 모든 조건이기 때문에 조건을 따로 넣지 않습니다. else 문의 하위 문장은 같은 간격의 들여쓰기를 적용해 주어야 합니다.

달면 삼키고 쓰면 뱉는다

```
>>> sweet = True
>>> bitter = not sweet
>>>
>>> if sweet:
...     'swallow'
... elif bitter:
...     'spit'
...
...
'swallow'
```

사공이 많으면 배가 산으로 간다

```
>>> too_many_captains = True
>>>
>>> if too_many_captains:
...     'ship goes to the mountain'
...
...
'ship goes to the mountain'
```

십 년이면 강산도 변한다

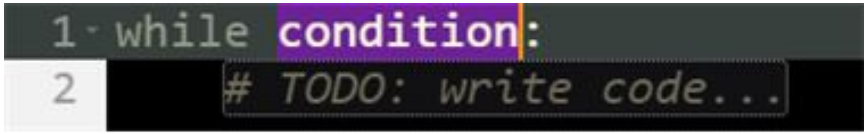
```
>>> past_year = 10
>>>
>>> if past_year == 10:
...     'rivers and mountains change'
...
...
'rivers and mountains change'
```

BMI 계산기

```
Height=float(input('Enter your height in centimeters: '))
Weight=float(input('Enter your Weight in Kg: '))
Height=Height/100
BMI=Weight/(Height*Height)
print(f'your Body Mass Index is: {BMI}')
if BMI>0:
    if BMI<=16:
        print('you are severely underweight')
    elif BMI<=18.5:
        print('you are underweight')
    elif BMI<=25:
        print('you are healthy')
    elif BMI<=30:
        print('you are overweight')
    else:
        print('you are severely overweight')
else:
    print('enter valid details')
```

while 문 톺아보기

while 문 정리하기



while 문은 어떤 동작을 계속 수행하고자 할 때 사용합니다. 예를 들어, 파이썬 프로그램을 계속 수행하고자 할 때 while 문을 사용합니다. while 문을 그치는 조건은 condition 부분에 넣습니다. while 문의 하위 문장은 같은 간격의 들여쓰기를 적용해 주어야 합니다.

천리길도 한 걸음부터 : while 활용하기

```
>>> way_to_go = 400*1000
>>> number_to_step = way_to_go*2.5
>>> step=0
>>> while step<number_to_step:
...     step += 1
...
...
>>> step
1000000
```

10리=약 4Km
1000리=약 400Km
1Km=1000m
1보=약 0.4m
1m당 약 2.5보

```
>>> step=0
>>> while step<number_to_step:
...     step += 1
...     if step % 20000 == 0:
...         print(step, '보')
...
...
```

천리길도 한 걸음부터 : while 활용하기

```
>>> way_to_go=400*1000
>>> number_to_step=way_to_go*2.5
>>> step=0
>>> while True:
...     step+=1
...     if step>=number_to_step:
...         break
...
...
>>> step
1000000
```

구슬이 서 말이라도 꿰어야 보배 : while 활용하기

```
>>> treasure = 54*1000
>>> pearl_threaded = 0
>>> while pearl_threaded<treasure:
...     pearl_threaded += 1
...
...
>>> pearl_threaded
54000
```

서 말 = 세 말
한 말 = 약 18 리터
세 말 = 약 54 리터
1 리터 = 10cm*10cm*10cm
구슬 하나 크기 = 1cm
1 리터 = 구슬 1000 개
54 리터 = 구슬 54*1000 개

구슬이 서 말이라도 꿰어야 보배 : while 활용하기

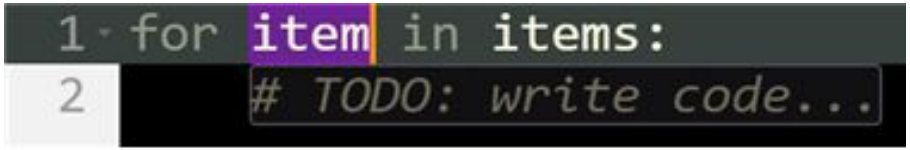
```
>>> while True:
...     pearl_threaded +=1
...     if pearl_threaded>=treasure:
...         break
...
...
>>> pearl_threaded
54000
```

티끌 모아 태산 : while 활용하기

```
>>> great_mountain=100000000
>>> dust_added=0
>>> while True:
...     dust_added += 1
...     if dust_added>=great_mountain:
...         break
...
...
>>> dust_added
100000000
```

for 문 톺아보기

for-in 문 정리하기



for-in 문은 같은 형태의 항목의 모임에 있는 각각의 항목에 대해 같은 형태의 동작을 반복하고자 할 때 사용합니다. 즉, 집합 형태의 데이터를 처리하기 위한 제어문입니다. for-in 문의 하위 문장은 같은 간격의 들여쓰기를 적용해 주어야 합니다.

for-in 형식 1 : range 함수

```
>>> for i in range(10):
...     i
...
...
0
1
2
3
4
5
6
7
8
9
```

```
>>> for i in range(0,10,2):
        i


0
2
4
6
8
>>> for i in range(10,0,-3):
        i


10
7
4
1
```

for-in : 정수 리스트 생성하기

```
>>> int_list = []
>>> for i in range(10):
...     int_list.append(i)
...
...
>>> int_list
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

리스트 내포 : 리스트 생성하기 편리

```
>>> [i for i in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> int_list_2 = [i for i in range(10)]
>>> int_list_2
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

임의 숫자 생성 : random.random(), random.randint(a,b)

```
>>> import random
>>>
>>> random.random()
0.02531693379458666
>>>
>>> random.randint(0,100)
42
>>>
>>> float_list=[]
>>> for _ in range(10):
...     float_list.append(random.random())
...
...
>>> float_list
[0.84544649816002, 0.5946760670502335, 0.8026071793669904, 0.414276105146452
86, 0.4110316554914315, 0.10950979641341718, 0.5856467039612103, 0.115771708
42056705, 0.8833798181117146, 0.42102967714488004]
>>>
>>> int_list_2=[]
>>> for _ in range(10):
...     int_list_2.append(random.randint(0,100))
...
...
>>> int_list_2
[97, 20, 86, 87, 74, 15, 59, 98, 66, 69]
```

임의 숫자 생성 : random.random(), random.randint(a,b)

```
>>> [random.random() for _ in range(10)]
[0.9440183851107024, 0.9044375712379684, 0.10666859536722595, 0.290131393192
15353, 0.12893935924153688, 0.5392454104727692, 0.24612589610794222, 0.64247
86931164383, 0.11459750001612234, 0.7440036934089492]
>>>
>>> [random.randint(0,100) for _ in range(10)]
[39, 23, 65, 33, 48, 47, 47, 39, 67, 11]
>>>
>>> float_list = [random.random() for _ in range(10)]
>>> float_list
[0.15864429014900394, 0.2836904513087716, 0.28959477517125565, 0.46410354877
539917, 0.02341394139473285, 0.48549061038723174, 0.1482282945282415, 0.7147
247628062257, 0.28496608116753863, 0.1414858340444237 2]
>>>
>>> int_list = [random.randint(0,100) for _ in range(10)]
>>> int_list
[81, 26, 99, 30, 28, 74, 66, 23, 50, 65]
```

임의 문자열 생성 : random.choices()

```
>>> import random
>>> import string
>>>
>>> string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>>
>>> random.choices(string.ascii_letters, k=10)
['v', 'K', 'r', 'i', 'J', 'V', 'I', 'U', 'P', 'M']
>>>
>>> ''.join(random.choices(string.ascii_letters, k=10))
'ukyooPncgk'
>>>
>>> str_list = []
>>> for _ in range(10):
...     str_list.append(''.join(random.choices(string.ascii_letters, k=10)))
...
...
>>> str_list
['CbdeHhAYmN', 'sRjvSmxcAL', 'gGGTHqWpHA', 'tPELqbfIJt', 'xVQKLNRRIx', 'gHXU
opEltB', 'WrXcYTsMoS', 'qARpzDOqcg', 'kDrcmCgnVb', 'AgPFZmJRvu']
```

임의 문자열 생성 : random.choices()

```
>>> [''.join(random.choices(string.ascii_letters, k=10)) for _ in range(10)]
['coBWtjqvEC', 'fcSIkRaZkd', 'yYQMhfdAbt', 'aHMIjxFAEn', 'PsdVSJpxCu', 'iQtVTAyWWV',
'QhYbeJjRni', 'RxsIniZdTk', 'kCgnxmtvYB', 'uVYHKUzIWc']
>>>
>>> str_list_2 = [''.join(random.choices(string.ascii_letters, k=10)) for _ in range(10)]
>>> str_list_2
['xahbVVmjxt', 'GyeeSRIyQP', 'IUKIoDCSzs', 'zHaAtZXXQZ', 'kjlAsytCLa', 'orFXUoKdhY',
'AffAJpDrlj', 'ECJxdntoLl', 'bLblkVplog', 'XbaYicJahq']
```

임의 숫자 생성 2 : numpy.random.random(), numpy.random.randint()

```
>>> import numpy as np
>>>
>>> np.random.random(10)
array([0.63577907, 0.50668027, 0.09034438, 0.42560984, 0.26787988,
       0.67003881, 0.40108399, 0.2926183 , 0.01391219, 0.74219088])
>>>
>>> list(np.random.random(10))
[0.7002399362137705, 0.49014648349932, 0.8655447041877247, 0.8941759192659277, 0.0155
14374672230602, 0.9655073446460648, 0.33319270934291545, 0.5869496516974557, 0.421097
12482773354, 0.7540008028948447]
>>>
>>> np.random.randint(0,100,10)
array([ 1, 31, 37,  7, 19, 12, 75,  1, 44, 96])
>>>
>>> list(np.random.randint(0,100,10))
[80, 60, 35, 94, 36, 81, 14, 85, 39, 50]
```

list 톺아보기

한 항목 접근 : indexing

```
>>> [0,1,2,3,4][0]
0
>>> [0,1,2,3,4][1]
1
>>> [0,1,2,3,4][4]
4
>>> [0,1,2,3,4][5]
Traceback (most recent call last):
  File "<pyshell#45>", line 1, in <module>
    [0,1,2,3,4][5]
IndexError: list index out of range
```

한 항목 접근 : indexing

```
>>> [0,1,2,3,4][-0]
0
>>> [0,1,2,3,4][-1]
4
>>> [0,1,2,3,4][-2]
3
>>> [0,1,2,3,4][-5]
0
>>> [0,1,2,3,4][-6]
Traceback (most recent call last):
  File "<pyshell#52>", line 1, in <module>
    [0,1,2,3,4][-6]
IndexError: list index out of range
```

부분 항목 접근 : slicing

```
>>> [0,1,2,3,4][0:1]
[0]
>>> [0,1,2,3,4][0:2]
[0, 1]
>>> [0,1,2,3,4][1:3]
[1, 2]
>>> [0,1,2,3,4][:3]
[0, 1, 2]
>>> [0,1,2,3,4][3:]
[3, 4]
>>> [0,1,2,3,4][:]
[0, 1, 2, 3, 4]
>>> [0,1,2,3,4][1:4:2]
[1, 3]
>>> [0,1,2,3,4][::2]
[0, 2, 4]
```

부분 항목 접근 : slicing

```
>>> [0,1,2,3,4][-3:-1]
[2, 3]
>>> [0,1,2,3,4][-3:]
[2, 3, 4]
>>> [0,1,2,3,4][:-3]
[0, 1]
>>> [0,1,2,3,4][-1:-3]
[]
>>> [0,1,2,3,4][4:2]
[]
>>> [0,1,2,3,4][-1:-3:-1]
[4, 3]
>>> [0,1,2,3,4][4:2:-1]
[4, 3]
```

리스트 주요 함수 : len, sort, reverse, random.shuffle

```
>>> import numpy as np
>>>
>>> np.random.seed(42)
>>>
>>> int_list = list(np.random.randint(0,100,10))
>>> int_list
[51, 92, 14, 71, 60, 20, 82, 86, 74, 74]
>>> len(int_list)
10
>>> int_list.sort()
>>> int_list
[14, 20, 51, 60, 71, 74, 74, 82, 86, 92]
>>> int_list.reverse()
>>> int_list
[92, 86, 82, 74, 74, 71, 60, 51, 20, 14]
>>> random.shuffle(int_list)
>>> int_list
[82, 74, 51, 86, 60, 92, 20, 74, 71, 14]
```

리스트 주요 함수 : append, pop

```
>>> int_list.sort()
>>> int_list
[14, 20, 51, 60, 71, 74, 74, 82, 86, 92]
>>> int_list.append(93)
>>> int_list.append(94)
>>> int_list
[14, 20, 51, 60, 71, 74, 74, 82, 86, 92, 93, 94]
>>> len(int_list)
12
>>> int_list.pop()
94
>>> int_list.pop()
93
>>> int_list
[14, 20, 51, 60, 71, 74, 74, 82, 86, 92]
```

리스트 주요 함수 : pop(0), insert

```
>>> int_list
[14, 20, 51, 60, 71, 74, 74, 82, 86, 92]
>>> int_list.pop(0)
14
>>> int_list.pop(0)
20
>>> int_list
[51, 60, 71, 74, 74, 82, 86, 92]
>>> int_list.insert(0,49)
>>> int_list.insert(0,48)
>>> int_list
[48, 49, 51, 60, 71, 74, 74, 82, 86, 92]
```

튜플, 사전, 집합 소개

```
>>> t = (1,2,3,4)
>>> d = { 'a':1, 'b':2, 'c':3, 'd':4 }
>>> s = {1,2,2,3,3,3,4,4,4,4}
>>> t
(1, 2, 3, 4)
>>> d
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
>>> s
{1, 2, 3, 4}
>>> type(t)
<class 'tuple'>
>>> type(d)
<class 'dict'>
>>> type(s)
<class 'set'>
```

```
>>> for i in t:
        i


1
2
3
4
>>> for i in d:
        i


'a'
'b'
'c'
'd'
>>> for i in s:
        i


1
2
3
4
```

튜플, 사전, 집합 : 내포와 생성자

```
>>> t = (i for i in range(10))
>>> d = {('key'+str(i)):i for i in range(10)}
>>> d
{'key0': 0, 'key1': 1, 'key2': 2, 'key3': 3, 'key4': 4, 'key5': 5,
'key6': 6, 'key7': 7, 'key8': 8, 'key9': 9}
>>> t = (i for i in range(10))
>>> d = {('key'+str(i)):i for i in range(10)}
>>> s = {i for i in range(10)}
>>> t
<generator object <genexpr> at 0x0000027AF1B653F0>
>>> d
{'key0': 0, 'key1': 1, 'key2': 2, 'key3': 3, 'key4': 4, 'key5': 5,
'key6': 6, 'key7': 7, 'key8': 8, 'key9': 9}
>>> s
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> type(t)
<class 'generator'>
>>> type(d)
<class 'dict'>
>>> type(s)
<class 'set'>
```

```
>>> for i in t:
        i


0
1
2
3
4
5
6
7
8
9

>>> for i in t:
        i
```

generator는 1회용

```
>>> for i in d:
        i

'key0'
'key1'
'key2'
'key3'
'key4'
'key5'
'key6'
'key7'
'key8'
'key9'
>>> for i in d:
        i

'key0'
'key1'
'key2'
'key3'
'key4'
'key5'
'key6'
'key7'
'key8'
'key9'
```

튜플 톺아보기

튜플의 기본적인 활용

```
>>> a = (1,2,3,4)
>>> a
(1, 2, 3, 4)
>>> type(a)
<class 'tuple'>
>>> b = (1)
>>> b
1
>>> type(b)
<class 'int'>
>>> c = (1,)
>>> type(c)
<class 'tuple'>
```

```
>>> d, e = 5, 6
>>> d
5
>>> e
6
>>> d, e = (5, 6)
>>> d
5
>>> e
6
```

```
>>> (d, e) = 5, 6
>>> d
5
>>> e
6
>>> (d, e) = (5, 6)
>>> d
5
>>> e
6
```

남은거 리스트로 묶기 : packing operater *

```
>>> d, e = 5, 6, 7
  Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
  ValueError: too many values to unpack (expected 2)
```

* 변수에 붙으면 리스트 묶기 연산자

```
>>> d, *e = 5, 6, 7
>>> d
5
>>> e
[6, 7]
>>> d, *e, f = 5, 6, 7
>>> e
[6]
>>> d, *e, f = 5, 6, 7, 8
>>> e
[6, 7]
```

```
>>> d, *e, f = 5, 6
>>> e
[]
>>> d, *e, f = 5
  Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
  TypeError: cannot unpack non-iterable int object
```

남은거 리스트로 묶기 : packing operater *

```
>>> *e, f = 5, 6
>>> e
[5]
>>> *e, = 5,
>>> e
[5]
>>> *e = 5,
  Traceback (most recent call last):
    File "<stdin>", line 1
  SyntaxError: starred assignment target must be in a list or tu
  ple
>>> *e = 5
  Traceback (most recent call last):
    File "<stdin>", line 1
  SyntaxError: starred assignment target must be in a list or tu
  ple
```

튜플과 함수

```
>>> def t_func():
        return 1, 2, 3

>>> a = t_func()
>>> a
(1, 2, 3)
>>> a, b, c = t_func()
>>> a
1
>>> b
2
>>> c
3
>>> a, *b = t_func()
>>> b
[2, 3]
```

dict 톺아보기

dict : 열쇠로 값 찾기

dict는 키와 값을 하나의 항목으로 하여 구성된 자료형

{key1:value1, key2:value2, key3:value3,...}

key는 문자열이나 정수

value는 문자열, 정수, 실수, 또는 객체

dict 생성하기

```
>>> shopping={}
>>> shopping['apple']=3
>>> shopping['pear']=2
>>> shopping['hanrabong']=5
>>> shopping['pine apple']=2
>>> shopping
{'apple': 3, 'pear': 2, 'hanrabong': 5, 'pine apple': 2}
```

dict 항목 읽기

```
>>> shopping
{'apple': 3, 'pear': 2, 'hanrabong': 5, 'pine apple': 2}
>>> shopping['apple']
3
>>> shopping['hanrabong']
5
```

dict 주요 함수 1

```
>>> shopping={'apple':3,'pear':2,'hanarabong':5,'pine apple':1}
>>> shopping.keys()
dict_keys(['apple', 'pear', 'hanarabong', 'pine apple'])
>>> shopping.values()
dict_values([3, 2, 5, 1])
>>> shopping.items()
dict_items([('apple', 3), ('pear', 2), ('hanarabong', 5), ('pine apple', 1)])
```

dict 풀어내기 1 : list

```
>>> list(shopping.keys())
['apple', 'pear', 'hanarabong', 'pine apple']
>>> list(shopping.values())
[3, 2, 5, 1]
>>> list(shopping.items())
[('apple', 3), ('pear', 2), ('hanarabong', 5), ('pine apple', 1)]
```

dict 풀어내기 2 : unpacking operator *

```
>>> [*shopping.keys()]
['apple', 'pear', 'hanarabong', 'pine apple']
>>> [*shopping.values()]
[3, 2, 5, 1]
>>> [*shopping.items()]
[('apple', 3), ('pear', 2), ('hanarabong', 5), ('pine apple', 1)]
```

dict 와 for-in 문

```
>>> for key in shopping.keys():
        key


'apple'
'pear'
'hanarabong'
'pine apple'
>>> for value in shopping.values():
        value


3
2
5
1
```

```
>>> for item in shopping.items():
        item


('apple', 3)
('pear', 2)
('hanarabong', 5)
('pine apple', 1)
```

dict 예제 : 장보기 1

```
>>> shopping={'apple':3, 'pear':2, 'hanrabong':5, 'pine apple':2}
>>> basket=[]
>>> for fruit, how_many in shopping.items():
        fruit,how_many
        plastic_bag=[]
        for _ in range(how_many):
            plastic_bag.append(fruit)
        basket.append(plastic_bag)


('apple', 3)
('pear', 2)
('hanrabong', 5)
('pine apple', 2)
>>> basket
[['apple', 'apple', 'apple'], ['pear', 'pear'], ['hanrabong', 'hanrabong', 'hanrabong
', 'hanrabong', 'hanrabong'], ['pine apple', 'pine apple']]
```

dict 예제 : 장보기 2

```
>>> shopping={'apple':3, 'pear':2, 'hanrabong':5, 'pine apple':2}
>>> basket=[]
>>> for fruit, how_many in shopping.items():
        fruit, how_many
        basket.append([fruit]*how_many)


('apple', 3)
('pear', 2)
('hanrabong', 5)
('pine apple', 2)
>>> basket
[['apple', 'apple', 'apple'], ['pear', 'pear'], ['hanrabong', 'hanrabong', 'hanrabo
', 'hanrabong', 'hanrabong'], ['pine apple', 'pine apple']]
```

dict 예제 : 장보기 3

```
>>> shopping={'apple':3,'pear':2,'hanrabong':5,'pine apple':2}
>>> basket=[[fruit]*how_many for fruit,how_many in shopping.it
    ems()]
>>> basket
[['apple', 'apple', 'apple'], ['pear', 'pear'], ['hanrabong',
'hanrabong', 'hanrabong', 'hanrabong', 'hanrabong'], ['pine ap
ple', 'pine apple']]
```

dict 예제 : 장보기 4

```
>>> shopping={'apple':3,'pear':2,'hanrabong':5,'pine apple':2}
>>> basket=[]
>>> while shopping:
        fruit,how_many = shopping.popitem()
        fruit,how_many
        basket.append([fruit]*how_many)


('pine apple', 2)
('hanrabong', 5)
('pear', 2)
('apple', 3)
>>> shopping
{}
>>> basket
[['pine apple', 'pine apple'], ['hanrabong', 'hanrabong', 'han
rabong', 'hanrabong', 'hanrabong'], ['pear', 'pear'], ['apple'
, 'apple', 'apple']]
```

함수 톺아보기

함수 : 자판기

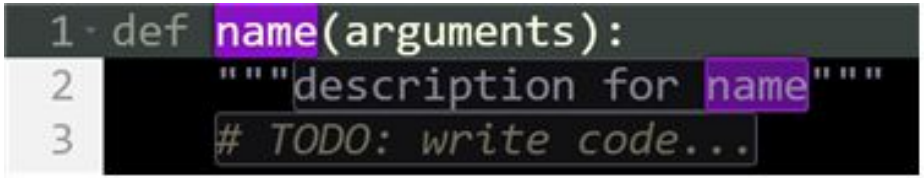
함수는 작은 동작을 묶어 큰 동작은 만드는 방법

큰 동작은 대표 동작

함수는 잘 정의하면 재사용할 수 있음(재사용성)

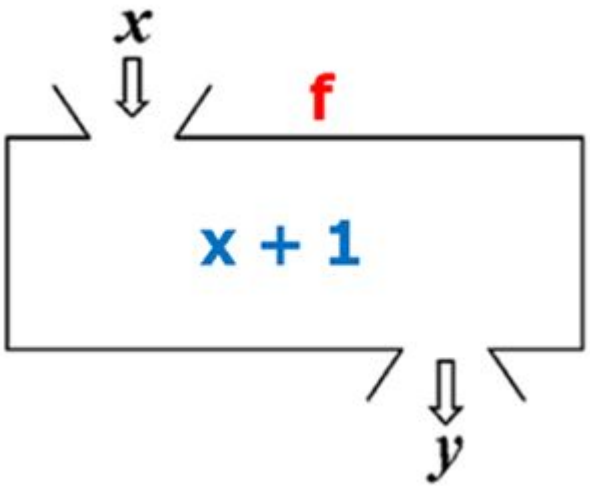
함수의 이름을 잘 정하면 가독성을 높일 수 있음(가독성)

함수 정의 하기 : def



def 문은 이름이 있는 함수를 정의할 때 사용합니다. 그 함수는 0개 이상의 인자를 받습니다. def 문의 하위 문장은 같은 간격의 들여쓰기를 적용해 주어야 합니다.

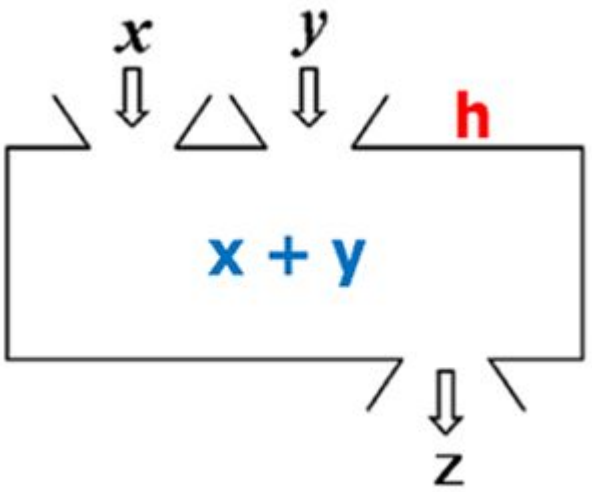
함수 정의하고 사용해 보기




```
def f (x):
    return x + 1

x = 2
y = f(x)
```

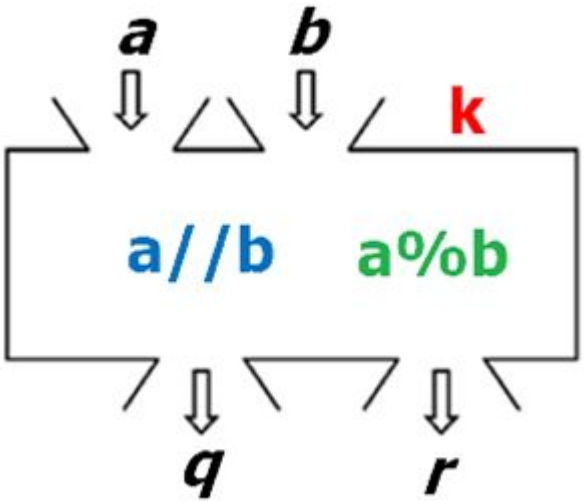
둘 이상의 함수 인자




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

```
x = 2
y = 2
z = h(x,y)
```

둘 이상의 함수 반환값




```
def k (a, b):
    q = a//b
    r = a%b
    return q, r
```

```
a = 19
b = 5
q, r = k(a, b)
```

위치 기반 인자, 키워드 기반 인자

```
>>> def func(a,b):
        a,b

>>> func(10,20)
(10, 20)
>>> func('hello','world')
('hello', 'world')
>>> func(b=100,a=3000)
(3000, 100)
```

함수와 커피 자판기

```
menu = {
    'americano': 1500,
    'caffe latte' : 2500,
    'espresso': 1500
}
total_money = 0
def coffee_machine(money, coffee_in):
    change, coffee_out = money, None
    if coffee_in in menu.keys():
        value = menu[coffee_in]
        if money>=value:
            change = money - value
            coffee_out = coffee_in
    return change, coffee_out

change, coffee = coffee_machine(5000, 'americano')
print(change, coffee)
```

*args 인자 이해하기

```
>>> def func(*args):
        args

>>> func(1,2,3,4)
(1, 2, 3, 4)
>>> l=[5,6,7,8]
>>> func(*l)
(5, 6, 7, 8)
```

```
>>> values = 1,2,3,4
>>> values
(1, 2, 3, 4)
>>> *values = 1,2,3,4
 Traceback (most recent call last):
   File "<stdin>", line 1
 SyntaxError: starred assignment target must be in a list or tuple
>>> *values, = 1,2,3,4
>>> values
[1, 2, 3, 4]
```

```
>>> values = 1,2,3,4
>>> values
(1, 2, 3, 4)
>>> *values
 Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
 TypeError: MainCPythonBackend._install_repl_helper.<locals>._handle_repl_valu
 e() takes 1 positional argument but 4 were given
>>> [*values]
[1, 2, 3, 4]
```

**kwargs

```
>>> def func(**kwargs):
        kwargs

>>> func(americano=1500,caffe_latte=2000,espresso=1500)
{'americano': 1500, 'caffe_latte': 2000, 'espresso': 1500}
```

```
>>> menu
{'americano': 1500, 'caffe latte': 2500, 'espresso': 1500}
>>> func(**menu)
{'americano': 1500, 'caffe latte': 2500, 'espresso': 1500}
```

함수와 for 문 1 : enumerate (리스트 항목 순서가 필요할 때)

```
>>> for i, coffee in enumerate(['americano', 'caffe latte', 'espresso']):
        i, coffee


(0, 'americano')
(1, 'caffe latte')
(2, 'espresso')
```

```
>>> enumerate(['americano', 'caffe latte', 'espresso'])
<enumerate object at 0x0000018FA9116F80>
>>> e=enumerate(['americano', 'caffe latte', 'espresso'])
>>> e
<enumerate object at 0x0000018FA9117BC0>
>>> [*e]
[(0, 'americano'), (1, 'caffe latte'), (2, 'espresso')]
```

함수와 for 문 2 : zip (리스트 항목별 묶기)

```
>>> l1=['americano', 'caffe latte', 'espresso', 'green tea']
>>> l2=[1500, 2500, 1600, 2200]
>>> z=zip(l1,l2)
>>> for coffee, price in z:
        coffee, price


('americano', 1500)
('caffe latte', 2500)
('espresso', 1600)
('green tea', 2200)
```

zip은 일회용!

```
>>> for coffee, price in z:
        coffee, price
```

zip으로 항목 별 묶고 풀기 1

```
>>> l1=['americano', 'caffe latte', 'espresso', 'green tea']
>>> l2=[1500, 2500, 1600, 2200]
>>> z=zip(l1,l2)
>>> l1_l2 = []
>>> for coffee, price in z:
        l1_l2.append((coffee, price))

>>> l1_l2
[('americano', 1500), ('caffe latte', 2500), ('espresso', 1600
), ('green tea', 2200)]
```

zip으로 항목 별 묶고 풀기 2

```
>>> z2=zip(*l1_l2)
>>> z2
<zip object at 0x0000018FA8E711C0>
>>> z3=[*z2]
>>> z3
[('americano', 'espresso', 'green tea', 'caffe latte'), (1500,
1600, 2200, 2500)]
```

```
>>> l1,l2=z3
>>> l1
('americano', 'espresso', 'green tea', 'caffe latte')
>>> l2
(1500, 1600, 2200, 2500)
```

zip으로 항목 별 묶고 풀기 3

```
>>> l1=[1,2,3,4]
>>> l2=[5,6,7,8]
>>> l3=[9,10,11,12]
>>> z=zip(l1,l2,l3)
>>> z
<zip object at 0x0000018FA8F12480>
>>> z=[*z]
>>> z
[(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]
```

zip으로 항목 별 묶고 풀기 4

```
>>> import random
>>> random.shuffle(z)
>>> z
[(4, 8, 12), (3, 7, 11), (2, 6, 10), (1, 5, 9)]
```

```
>>> z2=zip(*z)
>>> z2
<zip object at 0x0000018FA9249E80>
>>> z2=[*z2]
>>> z2
[(4, 3, 2, 1), (8, 7, 6, 5), (12, 11, 10, 9)]
>>> l1,l2,l3=z2
>>> l1
(4, 3, 2, 1)
```

무명 함수 : lambda

```
>>> f = lambda a,b : a * b
>>> f(10,20)
200
```

```
>>> f2 = lambda a : a**2
>>> f2(3)
9
>>> f3 = lambda a : a>2
>>> f3(2)
False
```

lambda, map, for-in 문의 활용

```
>>> f = lambda a,b : a * b
>>> f(10,20)
200
```

```
>>> m=map(lambda a: a**2, [1,2,3,4])
>>> m
<map object at 0x0000018FA913C310>
>>> m=[*m]
>>> m
[1, 4, 9, 16]
```

```
>>> f=lambda a:a**2
>>> l=[1,2,3,4]
>>> for m in map(f,l):
        m


1
4
9
16
```

lambda, filter, for-in 문의 활용

```
>>> f=filter(lambda a: a>2, [1,2,3,4])
>>> f
<filter object at 0x0000018FA9384D60>
>>> f=[*f]
>>> f
[3, 4]
```

```
>>> f=lambda a:a>2
>>> l=[1,2,3,4]
>>> for f in filter(f,l):
        f


3
4
```

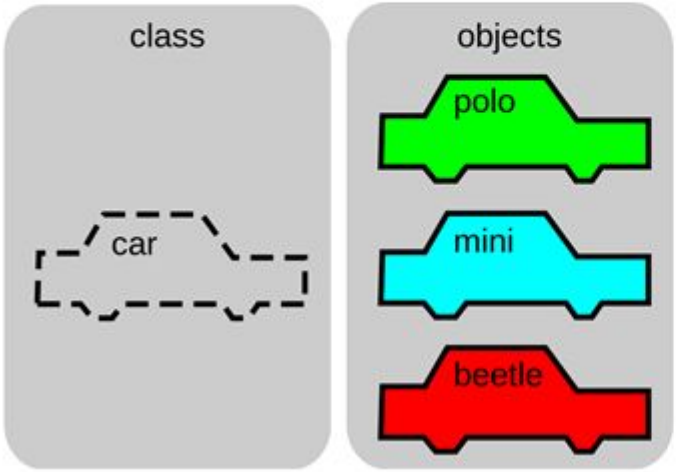
class 톺아보기

클래스와 객체 이해하기

초기에 컴퓨터 프로그램은 수식 계산에 초점이 맞추어져 있었습니다. 이때는 숫자 데이터를 중심으로 함수와 변수 위주로 프로그램이 작성되었습니다.

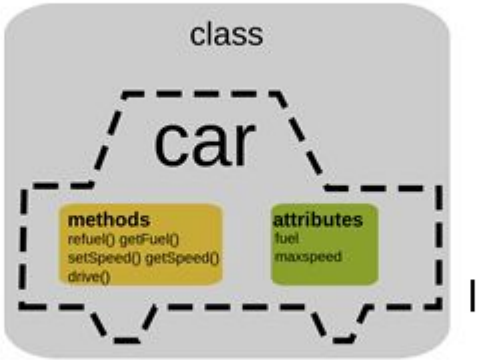
클래스와 객체 이해하기 2

컴퓨터 프로그램은 점차 여러 분야에 적용되었습니다. 예를 들어, 장부 등으로 기록하던 직원 관리, 학생 관리, 물품 관리 등도 컴퓨터 프로그램으로 대체되기 시작했습니다. 그래서 직원 관리 프로그램, 학생 관리 프로그램, 물품 관리 프로그램 등이 개발되었습니다. 이제 데이터의 중심이 숫자에서 사람, 물건 등으로 옮겨 갑니다. 프로그램도 함수와 변수 중심에서 객체와 클래스 중심으로 옮겨 갑니다.



클래스와 객체 이해하기 3

사람, 물건 하나하나는 객체라고 합니다. 이런 객체의 공통점을 뽑아 일반화한 것이 클래스입니다. 컴퓨터 프로그램에서 클래스는 객체를 만드는 틀이 됩니다. 객체는 속성과 방법을 갖습니다. 객체의 방법은 객체가 갖는 속성에 대한 접근 방법을 말합니다.



객체는 일반적으로 하나의 속성으로 결정되지 않습니다. 하나의 객체를 정의하기 위해서는 여러 개의 속성이 필요하며 그 속성들을 접근하기 위한 방법도 여러 개가 필요합니다. 그래서 객체는 여러 개의 속성과 방법을 하나로 묶는 역할을 합니다.

클래스 정의하기

이제 클래스와 객체에 대해 예제를 통해 구체적으로 살펴봅니다. 앞에서 함수는 어떤 목적을 갖는 하나 이상의 동작을 순차적으로 묶어 상위 동작을 만드는 개념이라고 소개하였습니다. 클래스는 어떤 목적을 갖는 하나 이상의 변수와 함수(==속성과 방법)를 묶어 상위 형태의 자료형을 만드는 개념입니다. 예를 들어, 파이썬에서 학생(Student)을 나타내는 자료형이 필요할 때, 다음과 같은 형태로 학생(Student) 자료 형을 정의할 수 있습니다.

```
1 : class Student:
2 :     pass
```

학생을 구성하는 요소로 이름(name), 나이(age), 성별(gender)이 필요하다고 할 때, 다음과 같이 속성 변수를 추가할 수 있습니다.

```
1 : class Student:
2 :     def __init__(self, name, age, gender):
3 :         self.name = name
4 :         self.age = age
5 :         self.gender = gender
```

객체 생성하기

학생 객체를 생성하여 변수로 가리킬 때는 다음과 같이 합니다.

```
1 : class Student:
2 :     def __init__(self, name, age, gender):
3 :         self.name = name
4 :         self.age = age
5 :         self.gender = gender
6 :
7 : student_1 = Student("홍길동", 23, "남자")
```

객체 사용하기

학생 객체의 정보를 출력하는 함수를 추가할 때는 다음과 같이

_12_class.py

```
01 : class Student:
02 :     def __init__(self, name, age, gender):
03 :         self.name = name
04 :         self.age = age
05 :         self.gender = gender
06 :
07 :     def Print(self):
08 :         print(self.name, self.age, self.gender)
09 :
10 : student_1 = Student("홍길동", 23, "남자")
11 : student_1.Print()
```

클래스 활용하기

```
>>> class RCCar:
        def __init__(self):
            self.dir = 'stop'
            self.spd = 0
        def go_forward(self):
            self.dir = 'forward'
        def go_backward(self):
            self.dir = 'backward'
        def turn_left(self):
            self.dir = 'left'
        def turn_right(self):
            self.dir = 'right'
        def set_speed(self, spd):
            self.spd = spd
        def stop(self):
            self.dir = 'stop'
            self.spd = 0
        def show_state(self):
            self.dir, self.spd
```

```
>>> mycar = RCCar()
>>> mycar.go_forward()
>>> mycar.set_speed(30)
>>> mycar.show_state()
('forward', 30)
```

클래스 활용하기 2

```
>>> class Remote:
        def __init__(self, car):
            self.car = car
        def go_forward(self, spd):
            self.car.go_forward()
            self.car.set_speed(spd)
        def stop(self):
            self.car.stop()
        def show_screen(self):
            self.car.show_state()
```

```
>>> remote = Remote(mycar)
>>> remote.go_forward(30)
>>> remote.show_screen()
('forward', 30)
>>> remote.stop()
>>> remote.show_screen()
('stop', 0)
```

클래스 활용하기 3

```
class CoffeeMachine:
    def __init__(self):
        self.menu = {}
        self.total_money = 0
    def set_menu(self,**menu):
        self.menu = menu
    def get_coffee(self, money_in, coffee_in):
        change, coffee_out = money_in, None
        if coffee_in in self.menu.keys():
            price = self.menu[coffee_in]
            if money_in>=price:
                change = money_in - price
                coffee_out = coffee_in
        return change, coffee_out
```

클래스 활용하기 3

```
cm = CoffeeMachine()
menu = {
    'americano':1500,
    'caffe latte':2500,
    'espresso':1500}
cm.set_menu(**menu)
print(cm.menu)
change, coffee = cm.get_coffee(5000, 'espresso')
print(change, coffee)
```