Elixirr

파이썬 기본 2

인선미

2029.01

튜플 자료형

- ✓ 리스트와 마찬가지로 여러 데이터를 하나로 묶어 저장하는 자료의 집합
- ✓ 인덱싱과 슬라이싱 가능
- ✓ **초기화 후 편집 불가**
- ✓ 소괄호 사이에 각각의 요소를 컴마로 구분해서 입력
- ✓ 생김새 :

  (90, 70, 'python', [1, 2, 3], 3.14)

- ✓ 소괄호 없이 값만 나열해도 가능
- ✓ 요소 하나의 튜플을 만드는 경우 값 뒤에 콤마를 넣어 튜플임을 나타냄

  예 : data_tuple01 = (3,) 또는 data_tuple01 = 3,

시퀀스 자료형

✓ 각각의 요소들이 연속적으로 이어진 자료형

- 리스트 `[0, 10, 20, 30, 40, 50]`

- 튜플 `(38, 76, 53, 62, 19)`

- range `range(5)`

- 문자열 `'Hello'`

시퀀스 자료형

✓ range로 생성된 숫자는 리스트나 튜플로 만들어서 사용가능

list(range(5)) : [0, 1, 2, 3, 4]

tuple(range(1, 6)) : (1, 2, 3, 4, 5)

list(range(5, 0, -1)) : [5, 4 3, 2, 1]

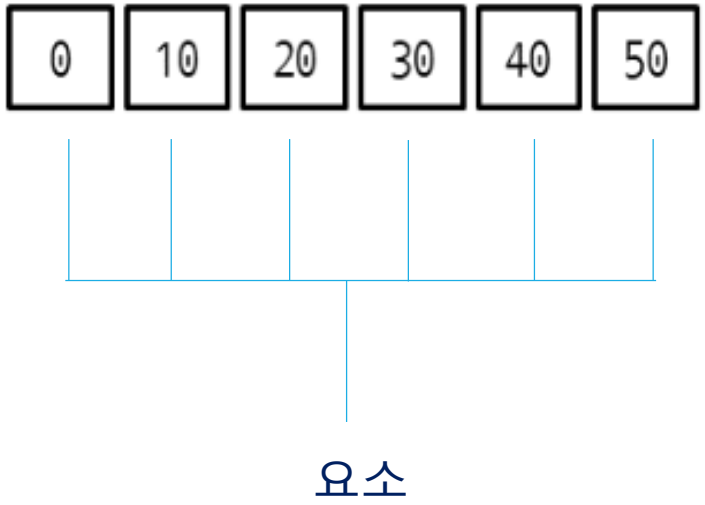
시퀀스 자료형

✓ 시퀀스 자료형은 공통된 동작과 기능을 제공

시퀀스 객체 : 시퀀스 자료형으로 만든 객체

요소(element) : 시퀀스 객체에 들어있는 각 값

```
[0, 10, 20, 30, 40, 50]
```



시퀀스 자료형의 공통기능 - in

✓ 시퀀스 객체 안에 특정 값이 있는지 확인하는 방법 - in 연산자

시퀀스 객체에 특정한 값이 있으면 True 반환

값 in 시퀀스 객체

```
>>> list01 = [ 10, 20, 30, 40, 50 ]
>>> 30 in list01
>>>
>>> 60 in list01
>>>
```

시퀀스 자료형의 공통기능 - not in

✓ 시퀀스 객체 안에 특정 값이 없는지 확인하는 방법 - not in 연산자

시퀀스 객체에 특정한 값이 없으면 True 반환

값 not in 시퀀스 객체

```
>>> 'apple' in 'orange and apple tree'
>>> 
>>> 30 not in (1, 2, 3, 4, 5)
>>> 
>>> 60 in range(100)
>>> 
```

딕셔너리 조작 함수

✓ **딕셔너리 키(key), 값(value) 반환** : keys() 함수, values() 함수, items( ) 함수

- 딕셔너리.items() 함수 : 키(key)와 값(value)의 쌍을 튜플로 묶은 값을 반환

```
>>> adic = {'a': 90, 'b': 80, 'c': 60, 'd': 70}
```

```
>>> for x in adic.keys():
        print(x)
a
b
c
d
```

키 리스트

```
>>> for x in adic.values():
        print(x)
90
80
60
70
```

값 리스트

```
>>> for x in adic.items():
        print(x)
'a', 90
'b', 80
'c', 60
'd', 70
```

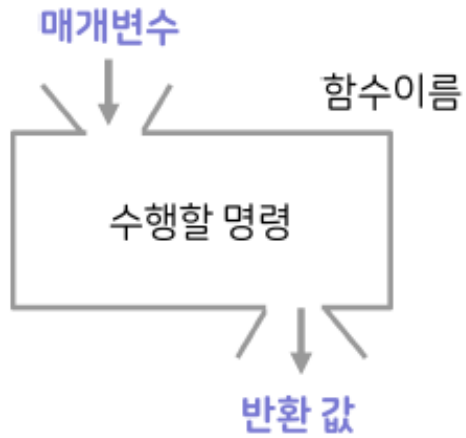
항목 리스트

함수와 클래스

함수 및 사용자정의 함수
모듈과 표준모듈
클래스/객체/메소드

함수

함수



특정 동작을 수행하는 일정한 코드의 모임

지금까지 파이썬의 다양한 내장함수를 사용해 왔음

print( ), input( ), int( )

내장함수와 사용자정의함수

함수를 정의하는 것은 재료(매개변수)를 받아서,
결과(반환 값)를 만들어주는 상자를 만드는 것

가독성: 파이썬 스크립트를 읽기 쉽게 해 줍니다.
재활용성: 반복되는 여러개의 명령을 하나로 묶어서 재활용성이 좋아집니다.

내장 함수

- **파이썬은 프로그래밍을 편리하게 할 수 있도록 미리 만들어진 함수를 제공**
- **별도의 설치과정 없이, 파이썬을 설치하면 바로 사용할 수 있는 함수**
- **이미 학습한 내장 함수들 및 자주 사용되는 함수**
  - print()
  - del()
  - type()
  - input()
  - range()
  - str()

내장 함수

```
for i in dir(__builtins__): print(i)
```

함수

		내장 함수		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

함수

▪ 내장 함수를 활용한 예제

- map 함수
  - **map(function, iterable)**
  - fucntion에 iterable의 item들을 인자로 전달하여 실행
  - function의 parameter 수 만큼 iterable이 전달되어야 함
  - 결과는 map 타입

함수

- 내장 함수를 활용한 예제

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

# for 반복문 이용
result1 = []
for val in myList:
    result1.append(val + 1)

print(f'result1 : {result1}')
```

```
# map 함수 이용
def add_one(n):
    return n + 1

result2 =
print(f'result2 : {result2}')
```

함수

▪ 내장 함수를 활용한 예제

- map
  - map을 사용하여 실수 값들의 일괄적 정수화(가까운 정수값으로)하기
  - tuple에 저장된 실수 값들을 가까운 정수값으로 변환한 정수 값을 인쇄하라(round 사용)

```
f = (3.14, -5.625, 100.4, 25.8)

# 코드 작성
```

함수

▪ 내장 함수를 활용한 예제

- zip 함수
  - zip(*iterable)
  - 각 iterable의 동일 인덱스의 item들을 튜플 쌍으로 반환
  - 결과는 zip 타입이며 iterable들의 길이는 달라도 무방

함수

▪ 내장 함수를 활용한 예제

- zip
  - 이미지의 색상별 데이터를 묶어서 24bpp RGB 이미지 만들기

[조건]

red, green, blue 픽셀 값이 각각의 튜플 r, g, b에 저장되어 있다
이 세 개의 튜플에서 같은 인덱스의 값들을 모아 튜플로 (r[0], g[0], b[0])로 구성한다
최종적으로 모아진 결과는 list로 만들고 이 내용들을 인쇄한다. 인쇄 결과를 참조한다
즉, 최종 결과 list가 img 라면 이 img는 [(r[0],g[0],b[0]), (r[1],g[1],b[1]), …] 형식이다

함수

▪ 내장 함수를 활용한 예제

- zip
  - 이미지의 색상별 데이터를 묶어서 24bpp RGB 이미지 만들기

```
1 r = (52, 255, 39, 132)
2 g = (19, 63, 227, 197)
3 b = (0, 68, 255, 187)
4
5 # 코드 작성
6
7 print(img)
```

함수

▪ 내장 함수를 활용한 예제

- enumerate 함수
  - enumerate(**iterable** [, start = 0])
  - start는 시작 번호(생략 시 0)이며 결과는 enumerate 타입
  - item들의 번호를 부여하여 (idx, value)의 튜플 쌍을 생성

함수

- enumerate

과일 품목별 일련번호 부여하기

튜플 f에 있는 과일들을 1번부터 차례대로 보기와 같이 인쇄되도록 번호를 부여하라
zip과 range를 이용하는 방법과 enumerate를 이용하는 방법이 가능하다

단, 결과는 dict로 생성되어야 한다(출력 예시 참조)
{1: 'apple', 2: 'orange', 3: 'banana', 4: 'mango'}

```
1 f = ('apple', 'orange', 'banana', 'mango')
2
3 # 코드 작성
4
5 print(tag)
```

<출력 결과>

```
{1: 'apple', 2: 'orange', 3: 'banana', 4: 'mango'}
```

함수

▪ 내장 함수를 활용한 예제 – filter 사용하기

- 반복 가능한 객체에서 특정 조건에 맞는 요소만 가져옴
- filter에 지정한 함수의 반환값이 True일 때만 해당 요소를 가져옴
- filter(함수, 반복가능한객체)

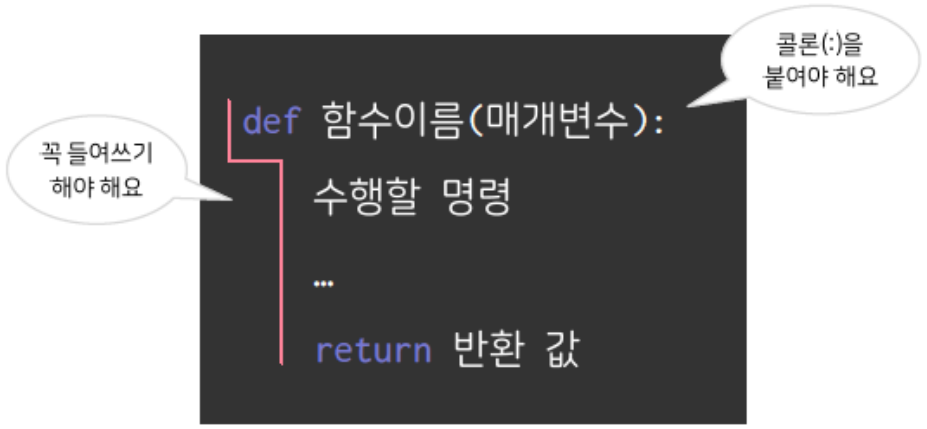
```
1 def f(x):
2     return x > 5 and x < 10
3
4 a = [8, 3, 2, 10, 15, 7, 1, 9, 0, 11]
5 list(filter(f, a))
```

함수

함수

함수 def

함수는 def* 키워드를 사용해서 정의할 수 있어요.



1. 함수 이름
2. 매개변수
3. 수행할 명령
4. 반환 값

*def : 영단어 define(정의하다)에서 비롯된 키워드

함수

매개변수와 인수

매개변수는 함수에 입력으로 전달된 값을 받는 변수

인수는 함수를 호출할 때 전달하는 입력 값

```
def add(a, b):  # a, b는 매개변수
    return a+b

print(add(3, 4))  # 3, 4는 인수
```

함수

기본 함수

입력값이 있고 리턴값이 있는 함수

```
def 함수_이름(매개변수):
    수행할_문장
    ...
    return 리턴값
```

```
def add(a, b):
    result = a + b
    return result
```

```
>>> a = add(3, 4)
>>> print(a)
7
```

함수

입력값이 없는 함수

```
>>> def say():
...     return 'Hi'
```

```
>>> a = say()
>>> print(a)
Hi
```

함수

리턴값이 없는 함수

```
>>> def add(a, b):
...     print("%d, %d의 합은 %d입니다." % (a, b, a+b))
```

```
>>> add(3, 4)
3, 4의 합은 7입니다.
```

함수

입력값도 리턴값도 없는 함수

```
>>> def say():
...     print('Hi')
```

```
>>> say()
Hi
```

함수

여러 개의 입력값을 받는 함수 만들기

함수 사용하기

```
>>> result = add_many(1,2,3)
>>> print(result)
6
>>> result = add_many(1,2,3,4,5,6,7,8,9,10)
>>> print(result)
55
```

함수

여러 개의 입력값을 받는 함수 만들기

여러 개의 입력값을 모두 더하는 함수
add_many(1, 2)이면 3
add_many(1, 2, 3)이면 6
add_many(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)이면 55를 리턴하는 함수

```
>>> def add_many(*args):
...
...
...
...
```

함수

여러 개의 입력값을 받는 함수 만들기

다음과 같이 호출하여 사용할 수 있도록 add_mul 함수를 만들어주세요

```
>>> result = add_mul('add', 1,2,3,4,5)
>>> print(result)
15
>>> result = add_mul('mul', 1,2,3,4,5)
>>> print(result)
120
```

함수

함수

```
>>> def add_and_mul(a,b):
...     return a+b, a*b
```

```
>>> result = add_and_mul(3,4)
```

```
>>> result1, result2 = add_and_mul(3, 4)
```

함수

return

함수의 종료
함수는 함수 블록이 끝나거나 return문을 만나면 종료
return문은 함수 블록 어디에든 올 수 있고 여러 번 있어도 OK

```
>>> def add_and_mul(a,b):
...     return a+b
...     return a*b
```

```
>>> def say_nick(nick):
...     if nick == "바보":
...         return
...     print("나의 별명은 %s 입니다." % nick)
```

함수

매개변수 초기화

```
def say_myself(name, age, man=True):
    print("나의 이름은 %s 입니다." % name)
    print("나이는 %d살입니다." % age)
    if man:
        print("남자입니다.")
    else:
        print("여자입니다.")
```

함수

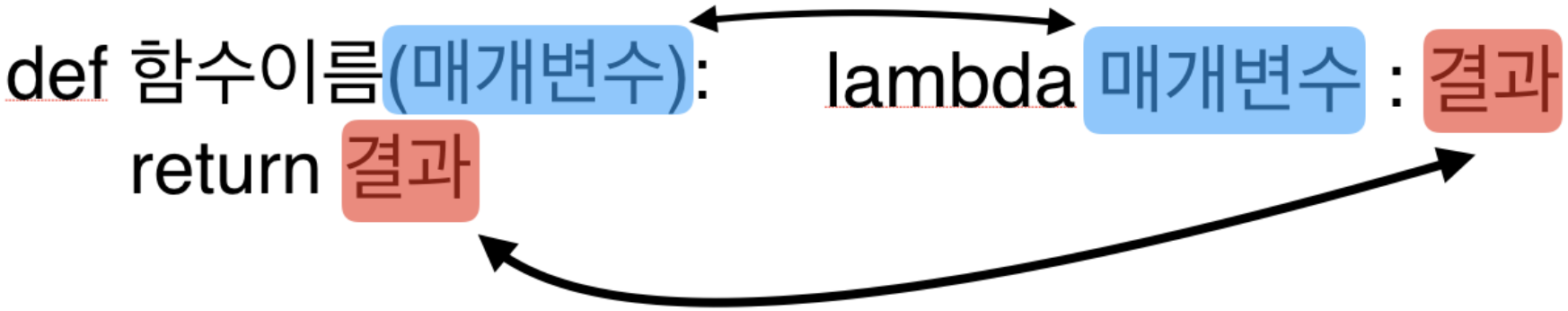
함수와 변수의 효력 범위

```
a = 1
def vartest(a):
    a = a +1

vartest(a)
print(a)
```

람다함수(익명함수)

- ✓ 이름이 없는 함수
- ✓ 람다함수의 장점 -> 코드의 간결함 메모리의 절약




```
>>> lambda x : x + 1
```

람다함수 사용 예

✓ 람다 함수의 존재 이유

✓ 익명 함수가 필요할 때(함수 이름이 필요 없는 경우).

✓ 코드의 가독성을 높이기 위해 짧은 함수 정의가 필요한 경우.

✓ 한 번만 사용하고 버릴 함수를 정의할 때.

```
f = lambda x, y: x + y
print(f(2, 3))
```

람다함수 사용 예

✓ 람다 함수의 사용 예

map: 시퀀스의 각 요소에 함수를 적용하여 새로운 시퀀스를 반환.

```
numbers = [1, 2, 3, 4]
squared =
print(squared)
```

출력: [1, 4, 9, 16]

람다함수 사용 예

✓ 람다 함수의 사용 예

filter: 시퀀스에서 특정 조건을 만족하는 요소만 필터링.

```
numbers = [1, 2, 3, 4, 5]
even_numbers =
print(even_numbers)
```

출력: [2, 4]

람다함수 사용 예

✓ 람다 함수의 사용 예

sorted 함수의 키로 사용하여 특정 조건에 따라 데이터를 정렬

```
points = [(1, 2), (3, 4), (5, 1)]
sorted_points =
print(sorted_points)
```

두 번째 값을 기준으로 정렬

출력: [(5, 1), (1, 2), (3, 4)]

모듈

모듈

모듈이란?

파이썬 코드로 이루어진 파일

함수나 변수, 클래스 등의 코드가 들어있음

직접 만들거나(사용자 정의 모듈)

다른 사람이 만들어 놓은 모듈(3rd party)을 가져와 사용

사용법

```
import 모듈명                    # 모듈 전체를 가져옴. 뒷부분에 as ... 를 넣어 긴 모듈명을 줄일 수 있음
from 모듈명 import 함수1, 함수2, ...    # 모듈 내 특정 함수만 가지고 오는 방법
```

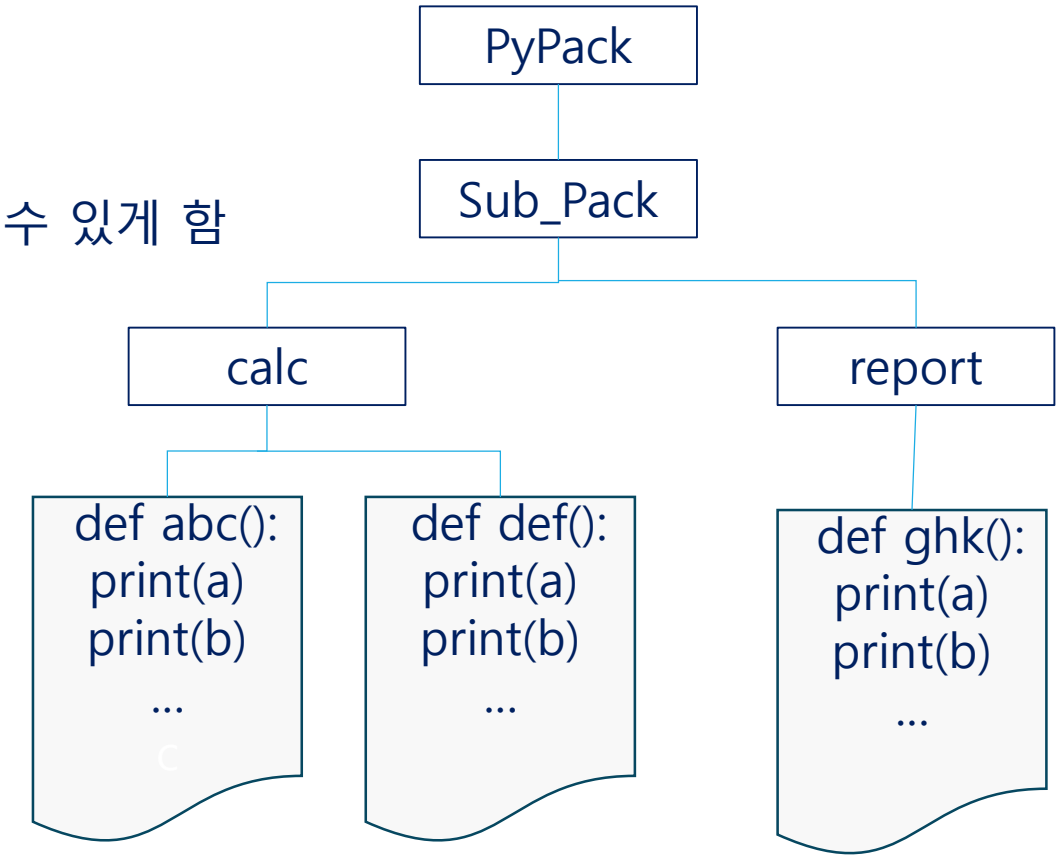
모듈

패키지란?

관련 있는 모듈의 집합

모듈을 담는 디렉토리

파이썬 모듈을 계층적인 디렉토리 구조로 관리할 수 있게 함



모듈

모듈 가져오기

import 모듈명(파일명)

- 모듈 전체 가져옴
- 모듈 내의 변수를 사용하기 위해 모듈.변수(함수 or 클래스) 형식으로 사용

from 모듈명 import 이름(함수명 or 클래스명)

- 모듈에서 특정 함수나 클래스만 사용하고 싶을 때 사용하는 방법
- import 하지 않은 함수, 클래스는 사용이 불가능

import 모듈명 as 이름(별명)

from 모듈명 import 이름(함수명 or 클래스명) as 이름(별명)

모듈

모듈 가져오기 예제

```
import math

num1 = math.pow(2, 3)
num2 = math.sqrt(9)

print(num1 + num2)
```

모듈

모듈 가져오기 예제

```
import math as m

num1 = m.pow(2, 3)
num2 = m.sqrt(9)

print(num1 + num2)
```

모듈

모듈 가져오기 예제 (from 모듈명 import *)

```
from math import *

num1 = pow(2, 3)
num2 = sqrt(9)

print(num1 + num2)
```

모듈

모듈 가져오기 예제 from 모듈명 import 이름(함수명 또는 클래스명)

```
from math import pow

num1 = pow(2, 3)
num2 = sqrt(9)

print(num1 + num2)
```

```
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    num2 = sqrt(9)
NameError: name 'sqrt' is not defined
```

모듈

모듈 가져오기 예제　　from 모듈명 import 이름(함수명) as 별명

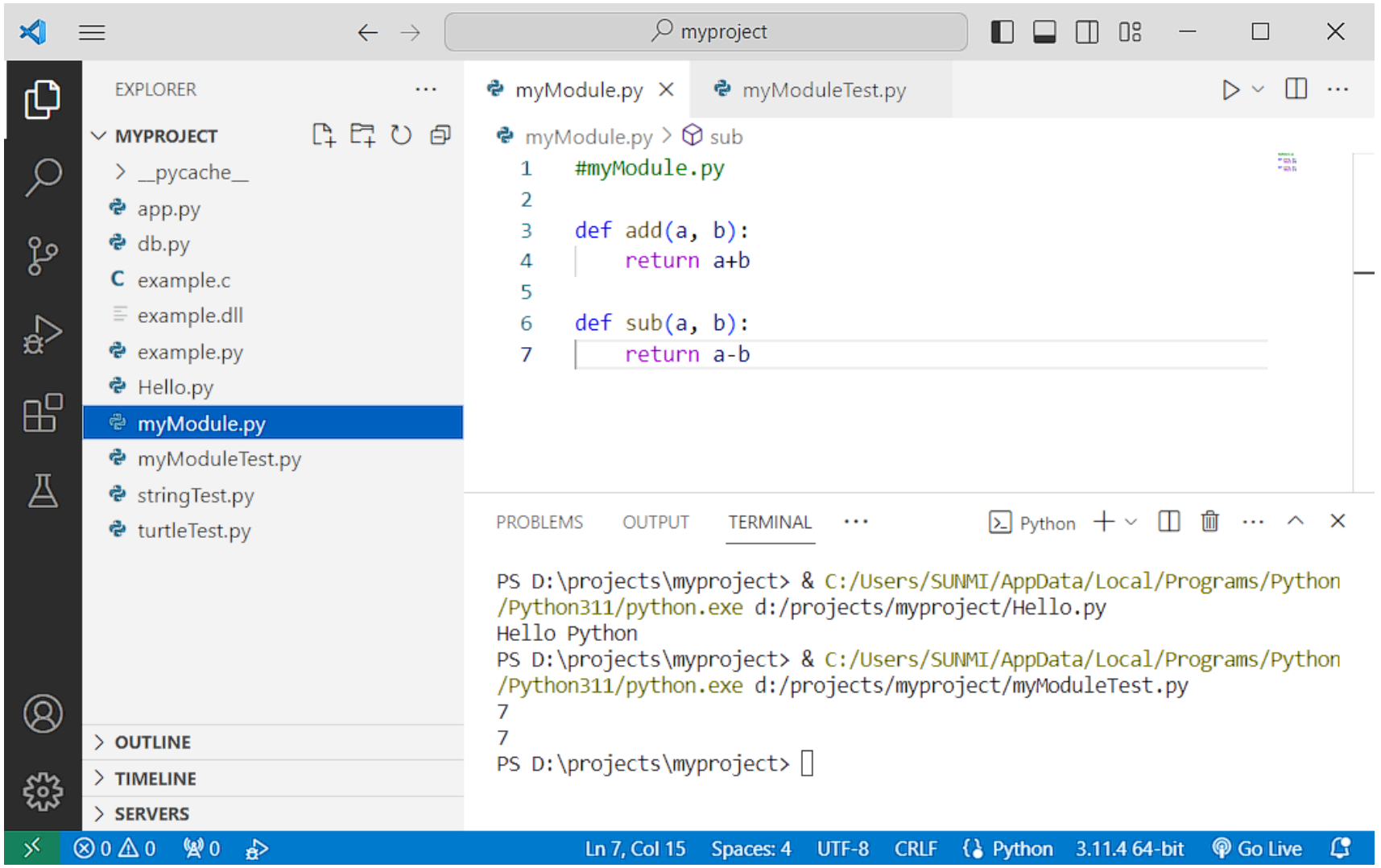
```
from math import pow as p, sqrt as sq

num1 = p(2, 3)
num2 = sq(9)

print(num1 + num2)
```

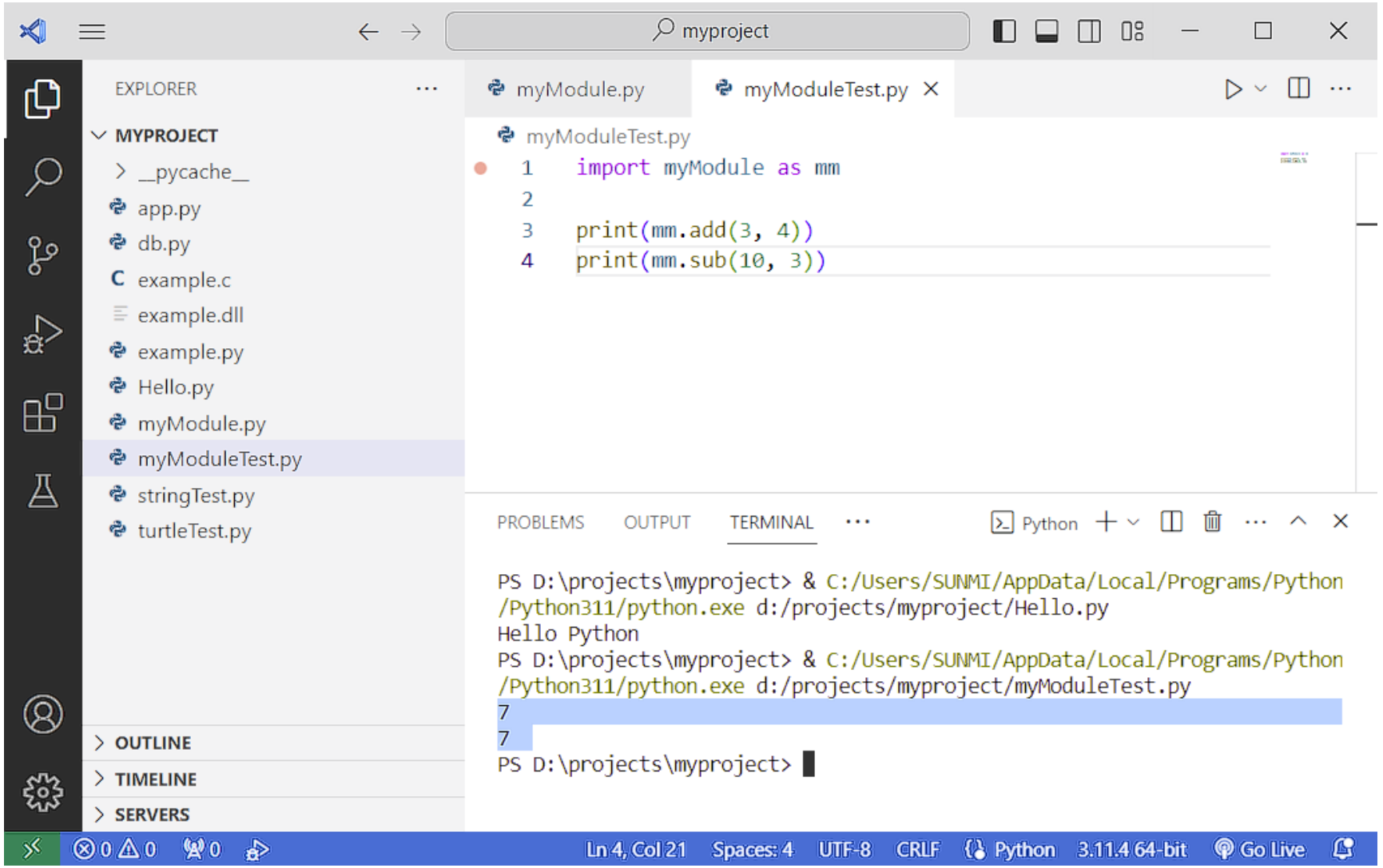
모듈

모듈 만들기



모듈

사용자 모듈 테스트



모듈

if __name__ == "__main__":의 의미

```
def add(a, b):
    return a+b

def sub(a, b):
    return a-b

if __name__ == "__main__":
    print(add(1, 4))
    print(sub(4, 2))
```

모듈

__name__ 변수

- 파이썬이 내부적으로 사용하는 특별한 변수 이름

- myModule.py파일을 실행할 경우, __name__ 변수에는 __main__값이 저장됨

- 그러나 다른 파이썬 모듈에서 myModule을 import할 경우에는 __name__ 변수에 myModule.py의 모듈 이름인 myModule 이 저장된다.

모듈

표준모듈

파이썬에서 기본으로 제공되는 모듈

자주 사용하는 기능이 표준 모듈로 설치되어 있음

현재 제공되는 모듈 목록 확인

https://docs.python.org/ko/3/library/index.html

주요 표준모듈

os 모듈 : 운영체제와 상호 작용을 위한 함수 제공

time 모듈 : 시간 관련 함수 제공

math 모듈 : 수학 관련 연산 함수 제공

random 모듈 : 다양한 랜덤(난수) 관련 함수 제공

모듈

random 모듈

난수 생성기능

예측 불가능한 무작위 동작 구현

```
import random

for I in range(5):
        print(random.random())
```

```
import random
for I in range(5):
        print(random.random())

0.5880766711219 94
0.15992254279600182
0.3560950068230381
0.8840809702784097
0.7763497918909065
for I in range(5):
        print(random.random())


0.20494234275731527
0.02896850820093 1664
0.2826757449097069
0.4380928846532256
0.12298579526160536
```

모듈

randint(begin, end) 함수

일정 범위의 내 임의의 정수 반환

begin 값과 end 값을 포함한 범위에서 추출

```
import random

for I in range(5):
        print(random.randint(1, 10))
```

```
for _ in range(5):
        print(random.randint(1, 10))

4
7
9
1
2
```

모듈

randrange(begin, end) 함수

일정 범위의 내 임의의 정수 반환

begin 값은 포함하지만 end 값은 포함하지

않는 범위에서 추출

```
import random

for I in range(5):
        print(random.randrange(1, 5))
```

```
for I in range(5):
        print(random.randrange(1, 5))

2
4
1
2
3
```

모듈

choice 함수

리스트에서 임의의 요소를 하나 반환

shuffle 함수

리스트의 요소를 무작위로 섞음

sample 함수

리스트 항목 중 n개를 무작위로 뽑아 새 리스트 생성

클래스

클래스

클래스란?

객체지향의 가장 기본적인 개념

객체를 표현하기 위한 문법

- 객체 : 체크박스, 스크롤바, 터틀객체 등 특정한 모양이나 개념이 존재하는 것.

현실세계 사물의 속성과 동작들을 코드로 구현하는 것

관련된 속성과 동작을 하나의 범주로 묶음

사물의 속성은 변수로, 동작은 함수(메소드)로 표현

- 속성은 매개변수를 받아 사용하기 위한 값을 정함
- 메소드는 만들어진 속성을 이용해 그에 따른 동작을 실행

멤버 : 클래스를 구성하는 변수와 함수

메소드 : 클래스에 속한 함수

클래스

▪클래스 (class)

▪똑같은 것을 계속 만들어낼 수 있는 틀, 설계도 등으로 정의하기도 함

예) 자동차 공장의 자동차 형태, 쿠키 틀 등

▪객체 (object)

▪클래스에서 찍어낸 형태

예) 자동차, 쿠키틀에서 찍어낸 쿠키

▪객체마다 고유한 성격을 가짐

예) 터틀객체가 각기 고유의 속성을 가질 수 있음 (색, 커서 모양 등)

▪클래스의 인스턴스(instance)라고 표현하기도 함

클래스

클래스 생성하기

클래스 선언 형식

```
class 이름:
    def __init__(self, 초기값):
        멤버 초기화
    메소드 정의
```

클래스

클래스 선언하기

```
class Human:
    def __init__(self, name, hobby):
        self.name = name
        self.hobby = hobby
    def intro(self):
        print("My name is %s. I like %s." % (self.name, self.hobby))
```

Human의 인스턴스 생성하기 (객체생성)

```
person01 = Human( " 안드로이드 ", "춤추기")
```

Intro 메소드 호출

```
person01.intro()
```

클래스

클래스

```
result1 = 0
result2 = 0

def add1(num):  # 계산기1
    global result1
    result1 += num
    return result1

def add2(num):  # 계산기2
    global result2
    result2 += num
    return result2

print(add1(3))
print(add1(4))
print(add2(3))
print(add2(7))
```

```
class Calculator:
    def __init__(self):
        self.result = 0

    def add(self, num):
        self.result += num
        return self.result

cal1 = Calculator()
cal2 = Calculator()

print(cal1.add(3))
print(cal1.add(4))
print(cal2.add(3))
print(cal2.add(7))
```

클래스

클래스 생성

```
class FourCal:
    def setdata(self, first, second):
        self.first = first
        self.second = second
    def add(self):
        result = self.first + self.second
        return result
    def mul(self):
        result = self.first * self.second
        return result
    def sub(self):
        result = self.first - self.second
        return result
    def div(self):
        result = self.first / self.second
        return result
```

클래스

클래스 생성

```
a = FourCal()
b = FourCal()

a.setdata(4, 2)
b.setdata(3, 8)

>>> a.add()
6
>>> a.mul()
8
>>> a.sub()
2
>>> b.add()
11
>>> b.mul()
24
```

클래스

클래스

- setdata 메서드를 수행하지 않고 add 메서드를 수행할 경우

```
>>> a = FourCal()
>>> a.add()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 6, in add
AttributeError: 'FourCal' object has no attribute 'first'
```

클래스

클래스

- 생성자 constructor
- 객체가 생성될 때 자동으로 호출되는 메서드를 의미
- 파이썬 메서드명으로 __init__를 사용

```
class FourCal:
def __init__(self, first, second):
        self.first = first
        self.second = second

    def setdata(self, first, second):
        self.first = first
        self.second = second
```

클래스

클래스

- 생성자 constructor
- 객체가 생성될 때 자동으로 호출되는 메서드를 의미
- 파이썬 메서드명으로 __init__를 사용

```
>>> a = FourCal(4, 2)
>>> a.first
4
>>> a.second
2
```

클래스

클래스의 상속

- 어떤 클래스를 만들 때 다른 클래스의 기능을 물려받을 수 있게 만드는 것
- FourCal 클래스를 상속하는 MoreFourCal 클래스
  - 기능 추가

```
class 클래스_이름(상속할_클래스_이름)
```

```
>>> class MoreFourCal(FourCal):
...     pass
```

클래스

클래스의 상속

- MoreFourCal 클래스는 FourCal 클래스를 상속했으므로 FourCal 클래스의 모든 기능을 사용할 수 있음

```
>>> a = MoreFourCal(4, 2)
>>> a.add()
6
>>> a.mul()
8
>>> a.sub()
2
>>> a.div()
2
```

클래스

클래스의 상속

- 기능추가

```
>>> class MoreFourCal(FourCal):
...     def pow(self):
...         result = self.first ** self.second
...         return result
```

클래스

클래스의 상속

- 기능추가 확인

```
>>> a = MoreFourCal(4, 2)
>>> a.pow()
16
>>> a.add()
6
```

클래스

클래스 – 메서드 오버라이딩

- 0으로 나눌 때 오류가 아닌 값 0을 리턴받고 싶다면?

```
>>> a = FourCal(4, 0)
>>> a.div()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    result = self.first / self.second
ZeroDivisionError: division by zero
```

클래스

클래스 – 메서드 오버라이딩

- 메서드 오버라이딩(method overriding)
- 부모 클래스(상속한 클래스)에 있는 메서드를 동일한 이름으로 다시 만드는 것
- 부모 클래스의 메서드 대신 오버라이딩한 메서드가 호출

```
>>> class SafeFourCal(FourCal):
...     def div(self):
...         if self.second == 0:  # 나누는 값이 0인 경우 0을 리턴하도록 수정
...             return 0
...         else:
...             return self.first / self.second
```

클래스

클래스 – 메서드 오버라이딩

- 0으로 나누었을 때의 ERR 해결
- FourCal 클래스와 달리 ZeroDivisionError가 발생하지 않고 의도한 대로 0이 리턴

```
>>> a = SafeFourCal(4, 0)
>>> a.div()
0
```

클래스

클래스 변수

- 클래스 안에 변수를 선언하여 생성
- 클래스_이름.클래스변수로 사용

```
>>> class Family:
...     lastname = "김"
```

```
>>> a = Family()
>>> b = Family()
>>> a.lastname
김
>>> b.lastname
김
```

클래스

클래스 변수

- 클래스변수는 객체변수와 달리 클래스로 만든 모든 객체에 공유된다.
- 실무에서 프로그래밍할 때도 클래스변수보다 객체변수를 사용하는 비율이 훨씬 높다.

```
>>> Family.lastname = "박"
>>> a.lastname
박
>>> b.lastname
박
```

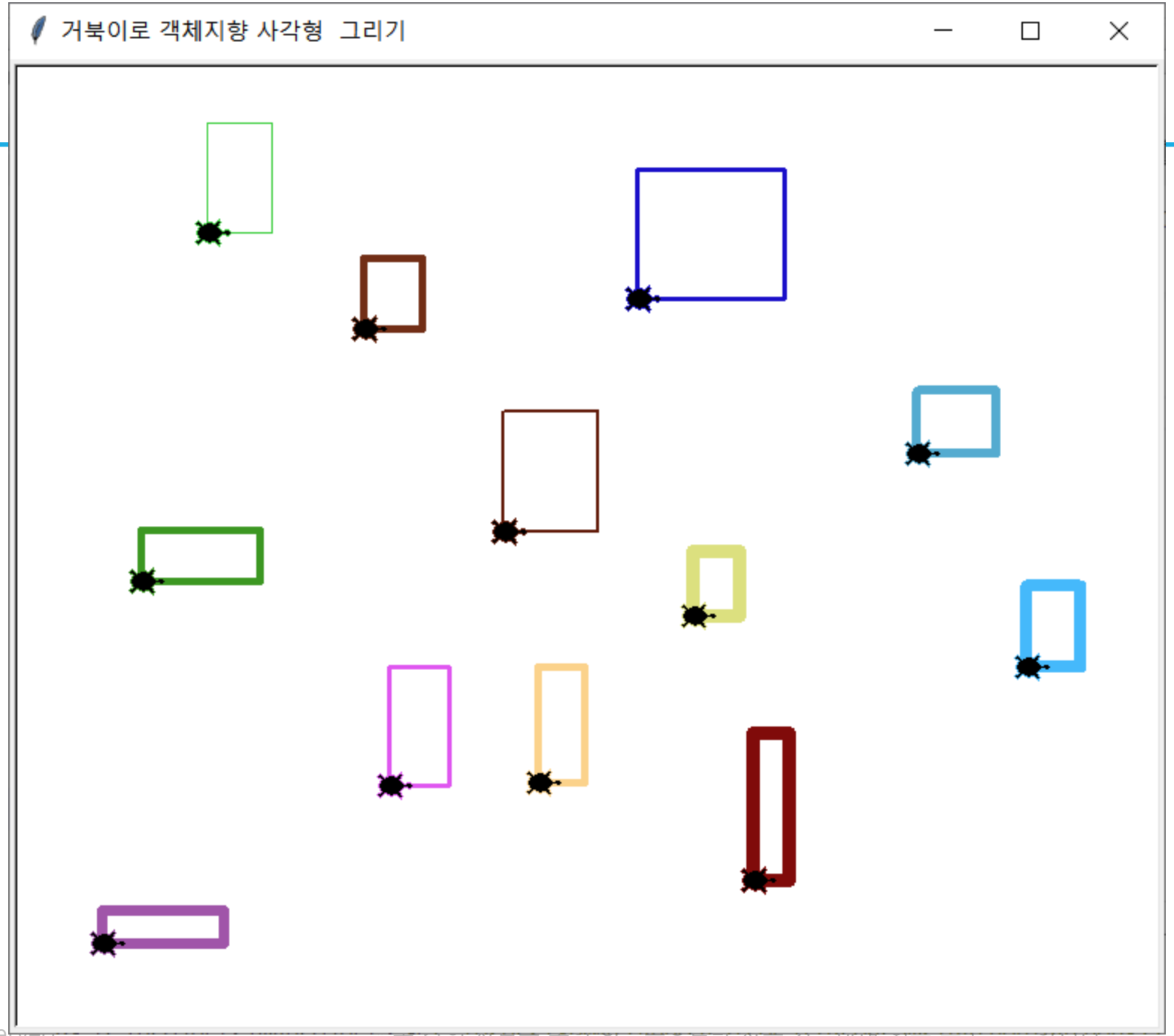
파이썬 터틀

▪ 파이썬 터틀이란?

- 파이썬에서 사용할 수 있는 그래픽 모듈
- LOGO 언어의 그래픽 라이브러리 (Seymour Papert, 1967)
- 터틀(거북이) 모양의 커서가 지나간 흔적을 이용해 화면에 선을 그림
- 그래픽 환경에서만 동작

https://docs.python.org/ko/3.10/library/turtle.html

클래스



감사합니다.