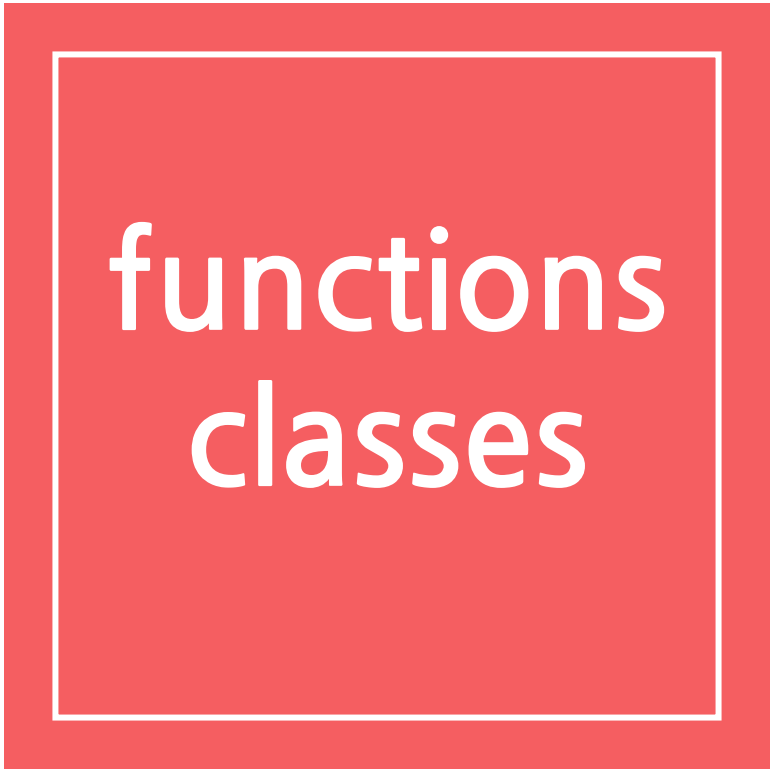

functions
classes

목차

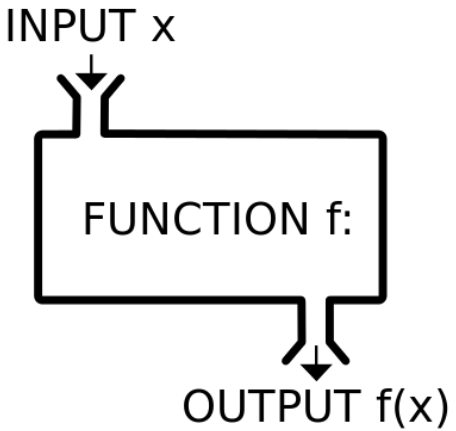
함수

함수

함수 '어떤 입력이 주어지면 어떤 결과를 돌려준다'

함수를 사용하는 이유: 코드를 재사용할 수 있음

반복적으로 사용되는 코드가 있는 경우 함수로 만들기



함수

함수 정의하기

```
def 함수명():
    실행문장
```

함수 호출하기

```
함수명()
```

함수

함수 정의하기

```
def my_func():
    print('함수가 실행되었습니다.')
```

함수 호출하기

```
my_func()
```

```
함수가 실행되었습니다.
```

함수

매개변수가 있는 경우

매개변수 함수에 입력으로 전달된 변수

```
def 함수명(매개변수, 매개변수, ...):
    실행문장
```

함수

```
def hello_n_times(n):
    for i in range(n):
        print('hello')
```

```
hello_n_times(1)
```

```
hello
```

```
hello_n_times(3)
```

```
hello
hello
hello
```

```
def print_n_times(string, n):
    for i in range(n):
        print(string)
```

```
print_n_times('안녕하세요', 2)
```

```
안녕하세요
안녕하세요
```

```
print_n_times('좋은 하루 보내세요', 3)
```

```
좋은 하루 보내세요
좋은 하루 보내세요
좋은 하루 보내세요
```

함수

return 함수를 실행했던 위치로 돌아가라는 의미

return 결과값 함수를 호출했던 위치로 돌아가면서 결과값을 전달할 수 있음

```
def func():
    print('A')
    return
    print('B')
```

```
print(func())
```

```
A
```

```
def func():
    print('A')
    return 'result'
    print('B')
```

```
print(func())
```

```
A
result
```

함수

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

```
add_and_mul(2, 3)
```

```
(5, 6)
```

함수의 결과값은 항상
하나로 반환됨

```
def str_times(string, n):
    return string*n
```

```
print(str_times('안녕',3))
```

```
안녕안녕안녕
```

함수

전역변수(global variable)

지역변수(local variable)

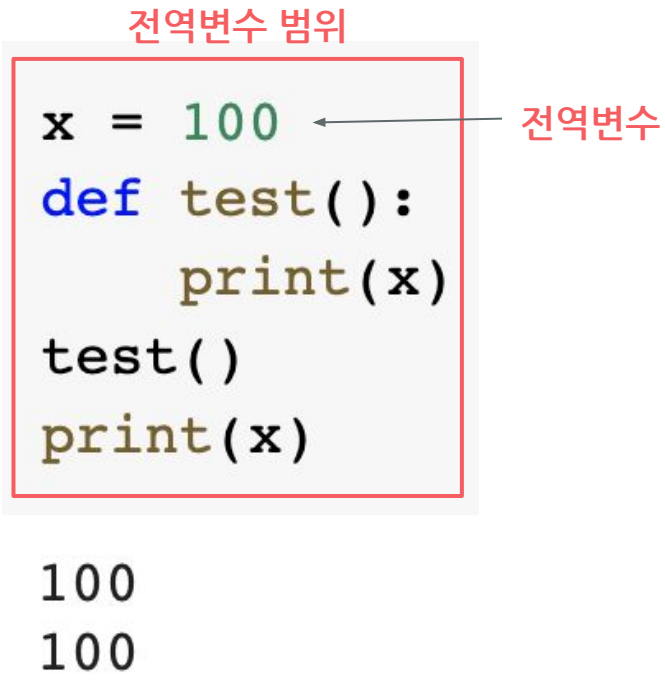
```
x = 100
def test():
    print(x)
test()
print(x)
```

```
100
100
```

함수

전역변수(global variable)

지역변수(local variable)



함수

전역변수(global variable)

지역변수(local variable)

x 는 test 내에서 사용할 수 있는 지역변수임.
함수 밖에서 사용하려고 하면 정의되지 않았다는 오류 발생

지역변수 범위

```
def test():
    x = 100
    print(x)
test()
print(x)
```

지역변수

```
100
---------------------------------
NameError
<ipython-input-99-71884d701b0a> in
      3     print(x)
      4 test()
----> 5 print(x)

NameError: name 'x' is not defined
```

함수

함수 안에서 전역 변수를 변경하려고 하면?

```
x = 100
def f():
    x = 200
f()
print(x)
```

```
100
```

```
x = 100
def f():
    x = 200
    return x
x = f()
print(x)
```

```
200
```

1. return 사용
   함수 내의 x와 함수 밖의 x는 다름.
   함수 내의 x를 리턴해서 함수 밖의 x에 할당

```
x = 100
def f():
    global x
    x = 200
    return x
f()
print(x)
```

```
200
```

2. global 사용
   global x : 함수 내에서 전역변수 x를 직접 사용하겠다는 의미
   권장하지는 않는 방법 (함수는 독립적으로 존재하는 것이 바람직)

함수

lambda

함수를 생성할 때 사용하는 예약어.

def 와 동일한 역할을 하나 lambda는 한줄로 간결하게 만들 때 사용.

return 명령어가 없어도 결과값을 전달해줌

```
함수명 = lambda 매개변수: 결과값
```

함수

lambda

```
f = lambda x: x**2+1000

print(f(0))
print(f(1))
print(f(2))
print(f(3))
```

```
1000
1001
1004
1009
```

```
f2 = lambda x1, x2: x1**x2+1000

print(f2(1, 2))
print(f2(1, 3))
print(f2(2, 3))
print(f2(3, 3))
```

```
1001
1001
1008
1027
```

클래스

클래스

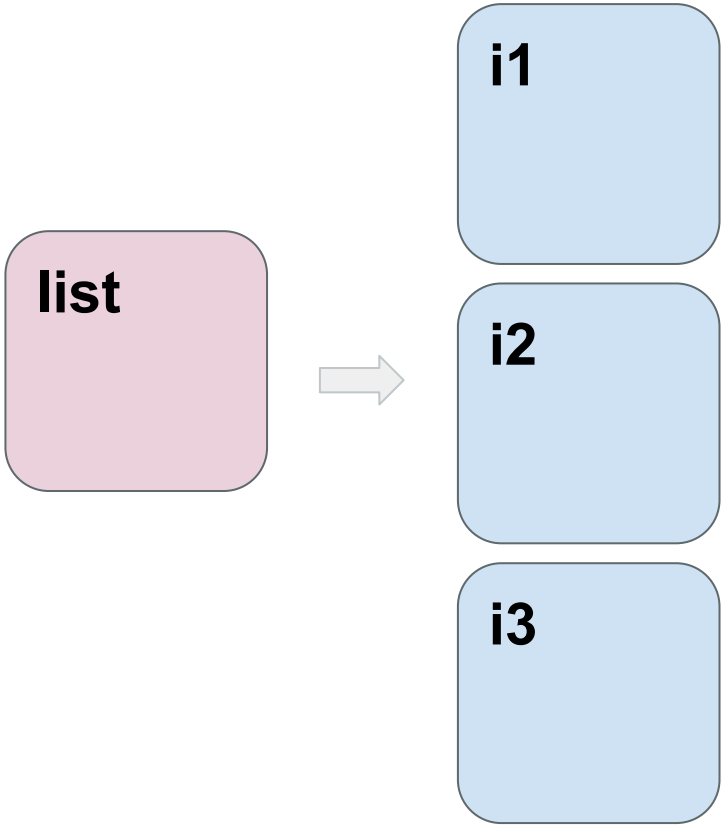
클래스(class)

인스턴스(instance)

```
print(type(list()))
```

```
<class 'list'>
```

```
i1, i2, i3 = [], [], []
```



리스트라는 클래스의 인스턴스 객체 i1, i2, i3 생성됨

클래스

클래스(class)

인스턴스(instance)

```
print(type(list()))
```

```
<class 'list'>
```

```
i1, i2, i3 = [], [], []
```

```
i1.append(10)
print(i1, i2, i3)
```

```
[10] [] []
```

```
i2.extend([10, 20])
print(i1, i2, i3)
```

```
[10] [10, 20] []
```

```
i2.remove(10)
print(i1, i2, i3)
```

```
[10] [20] []
```

클래스

클래스(class)

인스턴스(instance)

```
print(type(list()))
```

```
<class 'list'>
```

리스트도 클래스로 구현되어 있다!!

```
i1, i2, i3 = [], [], []
```

i1, i2, i3: 리스트라는 클래스의 인스턴스가 생성됨

```
i1.append(10)
print(i1, i2, i3)
```

```
[10] [] []
```

인스턴스 i1에 10 추가.
이는 i2, i3에 영향을 미치지 않음

```
i2.extend([10, 20])
print(i1, i2, i3)
```

```
[10] [10, 20] []
```

인스턴스 i2에 10, 20 추가.
이는 i1, i3에 영향을 미치지 않음

```
i2.remove(10)
print(i1, i2, i3)
```

```
[10] [20] []
```

인스턴스 i2에 10 제거.
이는 i1, i3에 영향을 미치지 않음

클래스

클래스(class)

인스턴스의 공장(factory). 객체를 쉽고 편리하게 생성하기 위해 만들어진 구문

인스턴스(instance)

클래스로 만들어진 구체적인 아이템. 클래스를 기반으로 생성한 객체

객체(object)

파이썬에서는 모든 것이 객체(예: 10, +, -)

클래스

왜 클래스를 사용해야 하는가?

상속 부모 클래스가 공통 속성을 구현하고, 자식 클래스는 이를 상속하여 재사용 가능

조합 하나의 클래스는 여러 조합으로 이루어져 있음

다중 인스턴스, 상속에 의한 커스터마이징, 연산자 오버로딩

클래스

클래스 만들기

```
class 클래스명():
    클래스에 관한 내용(속성, 메서드)
```

```
class MyClass():
    def method(self):
        self.x = 0
```

클래스

클래스 만들기

```
class 클래스명():
    클래스에 관한 내용(속성, 메서드)
```

```
class MyClass():
    def method(self):
        self.x = 0
```

클래스로 인스턴스 객체 만들기

```
인스턴스명 = 클래스명()
```

```
i1 = MyClass()
```

MyClass 클래스를 호출.
반환된 인스턴스 객체를 i1에 할당.

클래스

속성 변수. 객체에 저장된 상태 정보

메서드 함수. 객체에 적용할 행동, 행위

```
class Person:
    def __init__(self, name, job, dept, pay):
        self.name = name
        self.job = job
        self.dept = dept
        self.pay = pay
```

메서드

속성

클래스

__init__ 생성자. 클래스 호출시 자동으로 실행.

인스턴스 생성시 객체의 초기값을 정하는 용도로 사용.

```
class Person:
    def __init__(self, name, job, dept, pay):
        self.name = name
        self.job = job
        self.dept = dept
        self.pay = pay
```

```
id001 = Person('박땡땡', '디자이너', '디자인팀', 5000)
```

클래스

```
class Person:
    def __init__(self, name, job, dept, pay):
        self.name = name
        self.job = job
        self.dept = dept
        self.pay = pay

    def changeDept(self, dept):
        self.dept = dept

    def giveRaise(self, percent):
        self.pay = int(self.pay*(1+percent))
```

```
id001 = Person('박땡땡', '디자이너', '디자인팀', 5000)
id002 = Person('김땡땡', '영업사원', '영업팀', 5000)
```

```
print(id001.name, id001.job, id001.dept, id001.pay)
print(id002.name, id002.job, id002.dept, id002.pay)
```

```
박땡땡 디자이너 디자인팀 5000
김땡땡 영업사원 영업팀 5000
```

클래스

```
id002.giveRaise(0.10)
```

id002에 giveRaise 메서드를 통해 임금 인상

```
print(id001.pay, id002.pay)
```

id002만 인상되고 id001은 변함 없음

```
5000 5500
```

```
id002.giveRaise(0.05)
```

id002에 giveRaise 메서드를 통해 임금 인상

```
print(id001.pay, id002.pay)
```

id002만 인상되고 id001은 변함 없음

```
5000 5775
```

클래스

```
id001.changeDept('디자인2팀')
```

id001에 changeDept 메서드를 통해 부서 이동

```
print(id001.dept, id002.dept)
```

id001만 부서 이동되고 id002은 변함 없음

```
디자인2팀 영업팀
```

클래스

상속(inheritance)

부모클래스의 속성과 메서드를 물려받아 자식클래스를 생성하는 것

코드를 재사용

상속하면 다시 작성하지 않아도 부모 클래스의 모든 코드를 사용할 수 있음

필요한 것만 추가/변경하여 새로운 클래스 정의

클래스

상속

```
class Parent():
    클래스에 관한 내용(속성, 메서드)

class Child(Parent):
    클래스에 관한 내용(속성, 메서드)
```

부모(parent) 클래스
슈퍼(super) 클래스
베이스(base) 클래스

자식(child) 클래스
서브(sub) 클래스
파생된(derived) 클래스

클래스

상속

```
class Person:
    def __init__(self, name, job, dept, pay):
        self.name = name
        self.job = job
        self.dept = dept
        self.pay = pay

    def changeDept(self, dept):
        self.dept = dept

    def giveRaise(self, percent):
        self.pay = int(self.pay*(1+percent))
```

Person: 부모 클래스

```
class Manager(Person):
    def manage(self, dept):
        self.manageDepts = dept
```

Manager: 자식 클래스

클래스

상속

```
id100 = Manager('정땡땡', '디자이너', '디자인2팀', 9000)
```

Manager 클래스를 호출하여 id100 인스턴스 생성

```
print(id100.name, id100.job, id100.dept, id100.pay)
```

```
정땡땡 디자이너 디자인2팀 9000
```

부모 클래스에서 정의한 속성을 쓸 수 있음

```
id100.changeDept('디자인팀')
```

```
print(id100.dept)
```

```
디자인팀
```

Person 클래스에서 정의한 메서드를 사용. 즉, 자식 클래스로 생성된 인스턴스는 부모 클래스에서 정의한 메서드를 사용할 수 있음

```
id100.manage(['디자인팀', '디자인2팀', '디자인3팀'])
```

```
print(id100.manageDepts)
```

```
['디자인팀', '디자인2팀', '디자인3팀']
```

Manager에 있는 메서드 사용할 수 있음

클래스

상속

```
id001 = Person('박땡땡', '디자이너', '디자인팀', 5000)
```

Person 클래스를 호출하여 id001 인스턴스 생성

```
id001.giveRaise(0.1)
```

Person 클래스에서 정의한 메서드 사용 가능

```
print(id001.pay)
```

```
5500
```

```
id001.manage(['디자인팀'])
```

Manager 클래스에서 정의한 메서드 사용 불가능
부모 클래스로 생성된 인스턴스는 자식 클래스에서 정의한 메서드를 사용할 수 없음

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-120-2e1e11866dde> in <module>()
----> 1 id001.manage(['디자인팀'])

AttributeError: 'Person' object has no attribute 'manage'
```

클래스

오버라이딩 자식 클래스에서 부모 클래스의 메서드를 재정의하는 것

```
class Person:
    def __init__(self, name, job, dept, pay):
        self.name = name
        self.job = job
        self.dept = dept
        self.pay = pay

    def changeDept(self, dept):
        self.dept = dept

    def giveRaise(self, percent):
        self.pay = int(self.pay*(1+percent))
```

```
class Manager(Person):
    def manage(self, dept):
        self.manageDepts = dept

    def giveRaise(self, percent, bonus=0.05):
        Person.giveRaise(self, percent+bonus)
```

클래스

오버라이딩

```
id200 = Manager('윤땡땡', '엔지니어', '개발1팀', 10000)
```

```
id200.giveRaise(0.1, 0.1)
```

```
print(id200.pay)
```

```
12000
```

```
id300 = Manager('최땡땡', '총무', '관리팀', 10000)
```

```
id300.giveRaise(0.1)
```

```
print(id300.pay)
```

```
11500
```

giveRaise는 Manager 클래스에서 재정의 되었기 때문에 Manager 클래스에서 정의한 메서드가 적용된 것

Appendix

함수

인자를 주는 방법

위치 인자 값을 순서대로 상응하는 매개변수에 전달

키워드 인자 매개변수에 상응하는 이름을 인자에 지정해줌

위치 인자와 키워드 인자로 함수를 호출 시 위치 인자가 먼저 와야함

함수

인자를 주는 방법

위치 인자

```
def print_abc(a, b, c):
    print('a:', a)
    print('b:', b)
    print('c:', c)
```

```
print_abc(10, 20, 30)
```

```
a: 10
b: 20
c: 30
```

함수 호출
순서대로 인자를
전달받음

키워드 인자

```
print_abc(a=10, b=20, c=30)
```

```
a: 10
b: 20
c: 30
```

함수호출
키워드 인자로 전달

```
print_abc(c=30, b=20, a=10)
```

```
a: 10
b: 20
c: 30
```

함수 호출
키워드인자로 주면
순서는 상관 없음

함수

기본 매개변수값 설정

함수 호출시 기본값이 있는 매개변수는 입력값으로 주지 않으면 기본값을 사용

기본값이 있는 변수 뒤에는 기본값이 없는 일반 매개변수가 올 수 없음

```
def 함수명(일반매개변수, 기본매개변수=기본값):
    실행문장
```

함수

기본 매개변수값 설정

함수 호출시 기본값이 있는

매개변수는 입력값으로 주지

않으면 기본값을 사용

```
def func(x, y=0):
    print(x, y)
```

함수 선언

```
func(10, 10)
```

```
10 10
```

함수 호출
x=10, y=10 을 입력값으로 함
print(10, 10) 실행됨

```
func(20)
```

```
20 0
```

함수 호출
x=10을 입력값으로 함. y는
입력해주지 않았으니 기본값인 0
사용
print(20, 0) 실행됨

함수

기본 매개변수값 설정

기본값이 있는 변수 뒤에는 기본값이 없는 일반 매개변수가 올 수 없음

```
def func(x, y=0):
    print(x, y)
```

함수 선언
일반매개변수, 기본매개변수
순서대로 잘 선언함

```
def func(x=0, y):
    print(x, y)
```

```
  File "<ipython-input-26-0e0ca3565d62>", line 1
    def func(x=0, y):
                  ^
SyntaxError: non-default argument follows default argum
```

함수선언
기본매개변수 뒤에
일반매개변수가 와서 오류남
순서를 바꾸거나 y도 기본값을
주어 기본매개변수를 만들면
오류해결

함수

입력이 몇 개가 될지 모를 때는? 가변 매개변수

가변 매개변수 뒤에는 일반 매개변수가 올 수 없음

가변 매개변수는 하나만 사용할 수 있음

```
def 함수명(*매개변수):
    실행문장
```

함수

가변 매개변수

```
def add_func(*args):
    res = 0
    for i in args:
        res += i
    return res
```

```
print(add_func(0))
```

```
0
```

```
print(add_func(10, 20, 30))
```

```
60
```

함수 선언
add_func은 입력이 몇 개이든 상관 없음
*args 라고 매개변수 이름 앞에 *를 붙이면 입력값 전부를 모아 튜플로 만들어줌.
*args는 매개변수를 뜻하는 arguments의 약자. 관례적으로 많이 쓰임.

함수 호출
매개변수 1개

함수 호출
매개변수 3개

함수

가변 매개변수

가변 매개변수 뒤에는 일반 매개변수가 올 수 없음

```
def add_func(s, *args):
    print(s)
    print(sum(args))
```

함수 선언
일반매개변수, 가변매개변수
순서대로 잘 선언함

```
add_func('총합', 2, 3, 4)
```

함수 호출
s = '총합', args = (2, 3, 4)
로 입력됨

```
총합
9
```

함수

가변 매개변수

가변 매개변수 뒤에는 일반 매개변수가 올 수 없음

```
def add_func(*args, s):
    print(s)
    print(sum(args))
```

함수 선언
가변매개변수, 일반매개변수로 선언함

```
add_func(2, 3, 4, '총합')
```

함수 호출
args = (2, 3, 4, '총합')이 되고 일반 매개변수 s는 인자를 전달받지 못해 오류가 남
매개변수 선언시 순서를 바꾸거나 호출시 (2, 3, 4, s = '총합')이라고 s를 키워드 인자로 주면 오류 해결

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-47-c167f0cae2e3> in <module>()
----> 1 add_func(2, 3, 4, '총합')

TypeError: add_func() missing 1 required keyword-only argument: 's'
```

클래스

클래스 변수 클래스로 직접 접근 가능한 변수

클래스 메서드 클래스 영역에 존재하는 메서드. 클래스로 접근 가능(cls 파라메터 필수). @classmethod 데코레이터 사용.

클래스

클래스 메서드가 필요한 이유

인스턴스 대신에 클래스와 연관된 데이터를 처리할 필요가 있을 때 사용

예) 생성된 인스턴스의 개수를 추적하거나 현재 메모리에 있는 클래스 인스턴스의 전체 리스트를 관리하는 일

클래스

클래스 변수

클래스 변수 만들기

```
class 클래스명:
    클래스변수 = 값
```

클래스 변수에 접근하기

```
클래스명.클래스변수
```

```
class Student:
    count = 0
    students = []
    def __init__(self):
        Student.count += 1
        Student.students.append(self)
```

```
print(Student.count)
print(Student.students)
```

```
0
[]
```

클래스

클래스 변수

```
class Student:
    count = 0
    students = []
    def __init__(self):
        Student.count += 1
        Student.students.append(self)
```

클래스 변수 선언

인스턴스 메서드에서 클래스 변수 사용시
클래스명.클래스변수

```
print(Student.count, Student.students)
```

클래스변수에 접근하기
클래스명.클래스변수

```
0 []
```

```
id001 = Student()
id002 = Student()
```

```
print(Student.count)
print(Student.students)
```

```
2
[<__main__.Student object at 0x7fab94fa6eb8>, <__main__.Student object
```

클래스

클래스 메서드

클래스 메서드 만들기

```
class 클래스명:
    @classmethod
    def 클래스함수명(cls, 매개변수):
        함수내용
```

클래스 메서드 호출하기

```
클래스명.클래스함수명(매개변수)
```

```
class Student:
    count = 0
    students = []

    @classmethod
    def print(cls):
        print('*학생 목록*')
        for s in cls.students:
            print(str(s.name))

    def __init__(self, name):
        self.name = name
        Student.count += 1
        Student.students.append(self)
```

```
Student.print()
```

```
*학생 목록*
```

클래스

클래스 메서드

```
class Student:
    count = 0
    students = []

    @classmethod
    def print(cls):
        print('*학생 목록*')
        for s in cls.students:
            print(str(s.name))

    def __init__(self, name):
        self.name = name
        Student.count += 1
        Student.students.append(self)
```

```
Student.print()
```

```
*학생 목록*
```

```
id001 = Student('박땡땡')
id002 = Student('김땡땡')
id003 = Student('최땡땡')
```

```
Student.print()
```

```
*학생 목록*
박땡땡
김땡땡
최땡땡
```

모듈과 패키지

모듈(module) 함수, 변수, 클래스 등을 모아놓은 파일

- **표준 모듈** 파이썬에서 기본적으로 내장되어 있는 모듈

```
import 모듈
```

```
from 모듈 import 변수,함수,클래스
```

모듈과 패키지

모듈(module)

`import 모듈`

```
import math
```

```
math.pi
```

```
3.141592653589793
```

```
math.sin(1)
```

```
0.8414709848078965
```

`from 모듈 import 변수,함수,클래스`

```
from math import pi, sin
```

```
pi
```

```
3.141592653589793
```

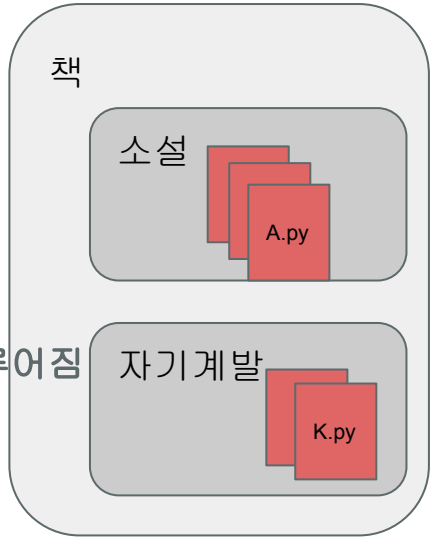
```
sin(1)
```

```
0.8414709848078965
```

모듈과 패키지

패키지(package) 모듈을 계층적으로 관리하는 상위 개념

공동작업이나 유지보수에 용이. 패키지 구조로 만들면 모듈명이 겹치더라도 안전하게 사용 가능.



파이썬 패키지는 디렉터리랑 모듈로 이루어짐
책, 소설, 자기계발은 디렉터리 이름
확장자가 .py인 것이 파이썬 모듈

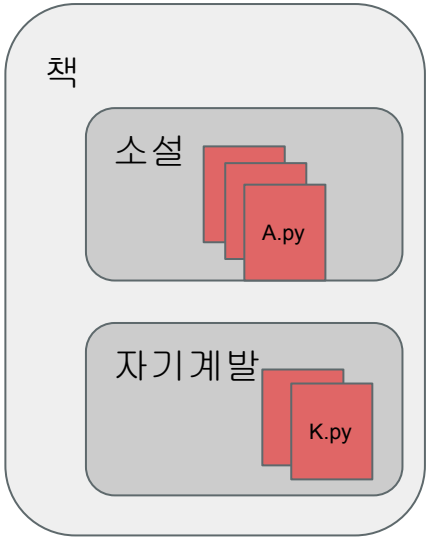
모듈과 패키지

패키지(package) 모듈을 계층적으로 관리하는 상위 개념

공동작업이나 유지보수에 용이. 패키지 구조로 만들면 모듈명이 겹치더라도 안전하게 사용 가능.

```
import 패키지
import 패키지.모듈
import 패키지.모듈.변수/함수/클래스

from 패키지 import 모듈
from 패키지.모듈 import 변수/함수/클래스
```



모듈과 패키지

as (alias) 다른 이름으로 모듈 임포트할 때 사용

```
import numpy as np
arr = np.array([10, 20])
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Reference

파이썬 공식 문서
러닝 파이썬
혼자 공부하는 파이썬
생활코딩 파이썬
처음 시작하는 파이썬
점프 투 파이썬