



# 05장 파이썬 날개달기

---

클래스, 모듈, 예외 처리

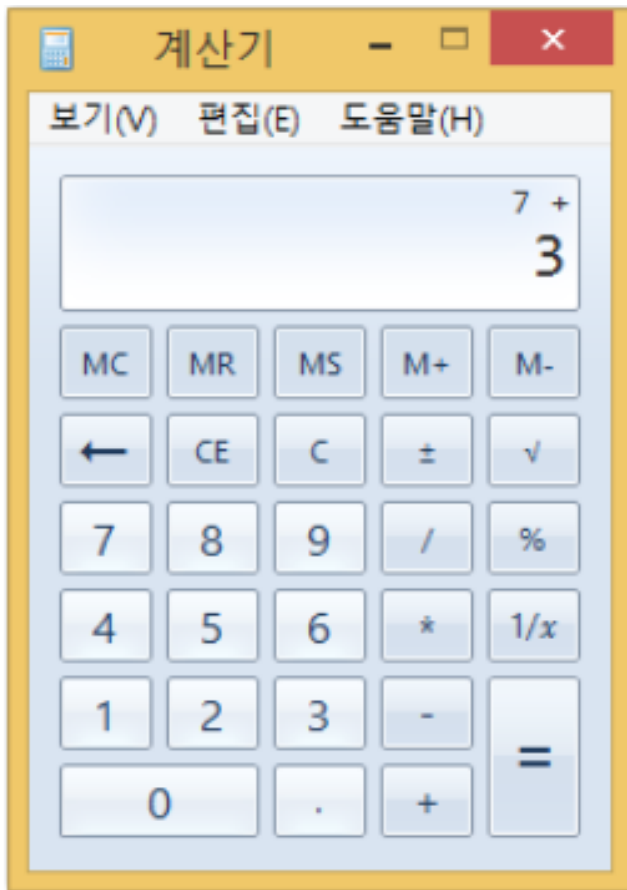


# 클래스는 왜 필요한가?

---

반복되는 변수나 메서드(함수)를  
미리 정해 놓은 틀(설계도)

# 클래스는 왜 필요한가?



$$3+4 = 7$$

$$7+3 = 10$$



# 05-1 파이썬 프로그래밍의 핵심, 클래스

클래스는 도대체 왜 필요한가?

```
result = 0
```

```
def add(num):
```

```
    global result → 이전에 계산된 결과값 유지를 위해 전역변수 사용
```

```
    result += num
```

```
    return result
```

→ Add 함수는 입력 인수로 num을 받으면 이전에 계산된 결과값에 더한 후 출력하는 함수이다.

→ `print(add(3))`

→ `Print(add(4))`

→ 3

→ 7

# 05-1 파이썬 프로그래밍의 핵심, 클래스

[한 프로그램에서 두 개의 계산기가 필요하다면?] - 함수를 각각 따로

```
result1 = 0
result2 = 0

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

def add2(num):
    global result2
    result2 += num
    return result2
```

- print(add1(3))      → 계산기 1의 결과값이 계산기 2에 아무런 영향을 끼치지 x
- print(add1(4))      → 계산기가 3개, 5개, 10개... 필요하다면? 전역변수와 함수를 추가??
- print(add2(3))
- print(add2(7))



# 클래스는 왜 필요한가?

---

반복되는 변수나 메서드(함수)를  
미리 정해 놓은 틀(설계도)

- 반복되는 변수 : result
- 메서드(함수) : add

### 클래스의 개념

#### 클래스의 모양과 생성

```
class 클래스명 :  
    # 이 부분에 관련 코드 구현
```

클래스를 쓰는 방법

1. class 입력하고
2. **대문자**로 시작하는 클래스의 이름을 작성
3. 안에 들어갈 함수와 변수 설정

# 05-1 파이썬 프로그래밍의 핵심, 클래스

## 클래스를 이용한 계산기

```
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))

- 클래스를 이용하면 계산기의 개수가 늘어나더라도 **인스턴스**를 생성하기만 하면 되기 때문에 함수를 사용하는 경우와 달리 매우 간단





# 05-1 파이썬 프로그래밍의 핵심, 클래스

## 클래스를 이용한 계산기

---

빼기 기능 추가

```
def sub(self, num):  
    self.result -= num  
    return self.result
```

# 클래스와 객체

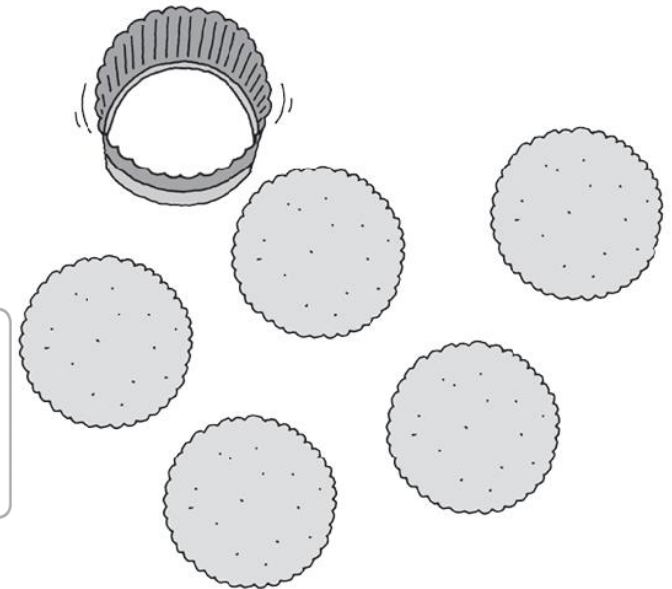
## 클래스와 객체

- 클래스(Class)

- 똑같은 무엇인가를 계속해서 만들어 낼 수 있는 설계 도면

- 객체(Object)

- 클래스로 만든 피조물



- 과자 틀 → 클래스(class)
- 과자 틀을 사용해 만든 과자 → 객체(object)

과자 틀(클래스)과 이 틀로 찍어 만든 과자(객체)

# 클래스와 객체

## 클래스와 객체

- 잘 만든 붕어빵틀이 있다면 새로운 종류의 다양한 붕어빵을 만들 수 있다.
- 즉, 잘 만든 클래스 코드가 있다면 이 코드로 다양한 종류의 인스턴스를 생성할 수 있다.



### [클래스와 객체의 관계]

과자 틀 → 클래스(class)

과자 틀을 사용해 만든 과자 → 객체(object)

# 05-1 파이썬 프로그래밍의 핵심, 클래스

아무 기능도 없는 껍질뿐인 클래스도 객체 생성이 가능

```
class Cookie:  
    pass
```

```
>>> a = Cookie()
```

```
>>> b = Cookie()
```

- ➔ `a = Cookie()` : `a`는 객체
- ➔ `a` 객체는 `Cookie`의 인스턴스이다. (관계)



# 생성자

---

- 생성자의 개념 : 인스턴스를 생성하면서 필드값을 초기화시키는 함수
  - 생성자의 기본
    - 생성자의 기본 형태 : `__init__()`라는 이름
- Tip** • `__init__()`는 앞뒤에 언더바(\_)가 2개씩, `init` 는 Initialize 의 약자로 초기화 의미  
언더바가 2 개 붙은 것은 파이썬에서 예약해 놓은 것, 프로그램을 작성시 이 이름을 사용하여 새로운 함수나 변수명을 만들지 말 것

```
class 클래스명 :  
    def __init__(self) :  
        # 이 부분에 초기화할 코드 입력
```



# 05-1 파이썬 프로그래밍의 핵심, 클래스

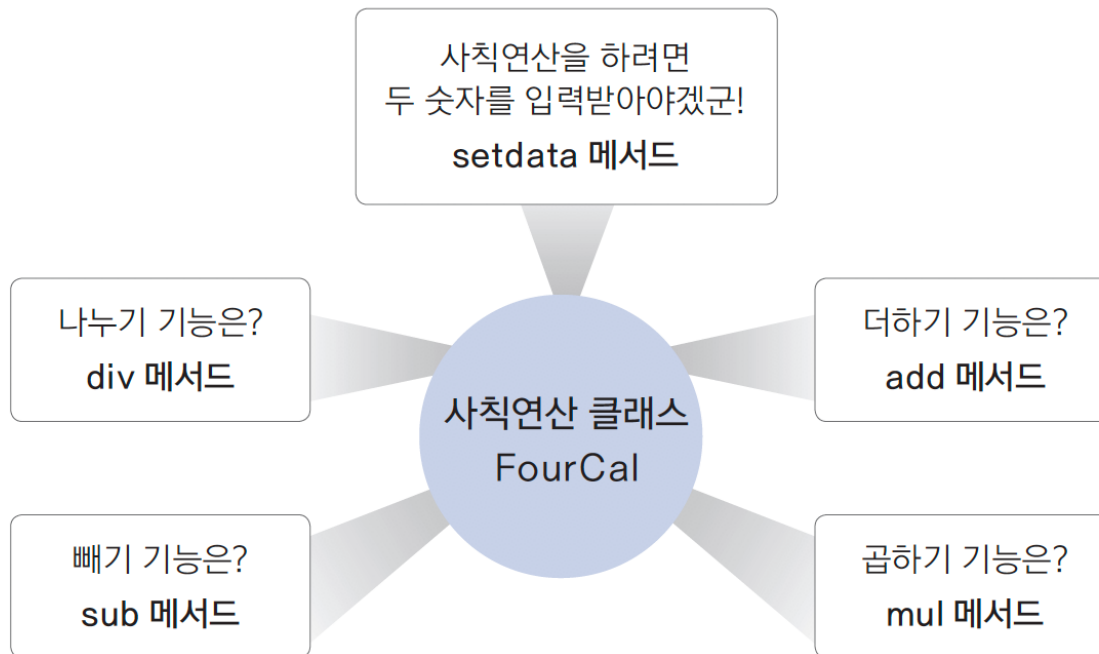
## 클래스의 구조

```
class 클래스이름[(상속 클래스명)]:  
    <클래스 변수 1>  
    <클래스 변수 2>  
    ...  
    def 클래스함수1(self[, 인수1, 인수2,,,]):  
        <수행할 문장 1>  
        <수행할 문장 2>  
        ...  
    def 클래스함수2(self[, 인수1, 인수2,,,]):  
        <수행할 문장1>  
        <수행할 문장2>  
        ...
```

- **class**라는 키워드는 클래스를 만들 때 사용되는 예약어
- 바로 뒤에 클래스 이름을 입력한다. 상속할 클래스가 있다면 ()안에 입력한다.
- 클래스 내부에는 클래스 변수와 클래스 함수들이 있다.

# 사칙연산 클래스 만들기

어떤 클래스를 만들지 대충 그림을 그려 보자





# 05-1 파이썬 프로그래밍의 핵심, 클래스

a = FourCal()를 입력해서 a라는 객체를 만든다.

```
>>> a = FourCal()
```

a.setdata(4,2)처럼 숫자 4와 2를 a에 지정해 주고

```
>>> a.setdata(4,2)
```

```
>>> print(a.add())
```

```
6
```

```
>>> print(a.mul())
```

```
8
```

```
>>> print(a.sub())
```

```
2
```

```
>>> print(a.div())
```

```
2
```





# 05-1 파이썬 프로그래밍의 핵심, 클래스

---

## 사칙연산 클래스 1

```
>>> class FourCal:  
...     pass  
...  
>>>
```

```
>>> a = FourCal()  
>>> type(a)  
<class '__main__.FourCal'> ➔ 객체 a의 타입은 FourCal 클래스이다.
```

➔ 객체 a가 FourCal 클래스의 인스턴스임을 알 수 있다.



# 05-1 파이썬 프로그래밍의 핵심, 클래스

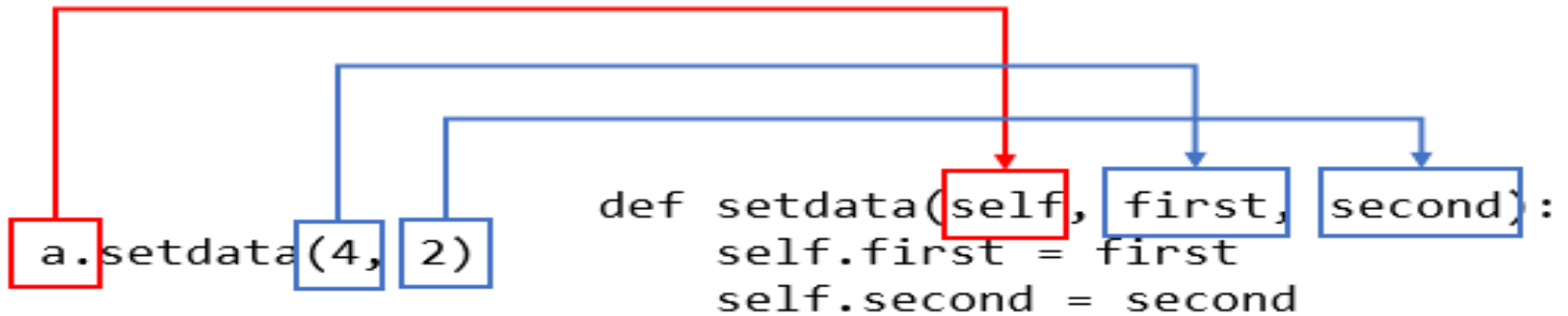
## 사칙연산 클래스 2

```
>>> a.setdata(4, 2)
```

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

- ➔ 클래스 안에 구현된 함수를 **메서드(Method)**라 부른다.
- ➔ setdata 메서드의 입력 인수는 self, first, second라는 3개의 입력값을 받는다.
- ➔ set 메서드의 첫 번째 인수에는 자동으로 a라는 인스턴스가 입력으로 들어가게 된다.

# 05-1 파이썬 프로그래밍의 핵심, 클래스



➔파이썬 메서드의 첫 번째 매개변수 이름은 관례적으로 **self**를 사용한다.

```
self.first = 4  
self.second = 2 해석됨.
```

Self가 전달된 객체 a이므로 다시 아래와 같이 해석됨.

```
a.first = 4  
a.second = 2
```



## 05-1 파이썬 프로그래밍의 핵심, 클래스

```
>>> a = FourCal()
>>> a.setdata(4, 2)
>>> print(a.first)
4
>>> print(a.second)
2
##a, b객체를 만들어 보자
>>> a = FourCal()
>>> b = FourCal()
>>> a.setdata(4,2)
>>> print(a.first)
4
>>> b.setdata(3,7)
>>> print(b.first)
3
>>> print(a.first)
4
```



# 05-1 파이썬 프로그래밍의 핵심, 클래스

---

## 사칙연산 클래스 3

```
>>> a = FourCal()  
>>> b = FourCal()  
>>> a.setdata(4, 2)  
>>> b.setdata(3, 7)  
>>> id(a.first)  
>>> id(b.first)
```

➔ 객체 변수는 다른 객체들 영향 받지 않고 독립적으로 그 값을 유지한다.



# 05-1 파이썬 프로그래밍의 핵심, 클래스

## 사칙연산 클래스 4

```
>>> class FourCal:
...     def setdata(self, first, second):
...         self.first = first
...         self.second = second
...     def add(self):
...         result = self.first + self.second
...         return result
...
>>>
```

- 새롭게 추가된 것은 add이라는 메서드이다.
- 입력으로 받는 값은 self밖에 없고, 돌려주는 값은 result이다.
- a.add()처럼 수행하면 add 함수에 자동으로 객체 a가 첫번째 입력 인수로 들어가된다. result = self.first + self.second



## 05-1 파이썬 프로그래밍의 핵심, 클래스

```
>>> a = FourCal()  
>>> a.setdata(4, 2)  
>>> print(a.add())  
6
```

즉, add메서드의 매개변수는 self이고 반환값은 result이다.

```
>>> result = self.first + self.second  
>>> result = a.first + a.second  
>>> result = 4+2  
>>> print(a.add())  
6
```

곱하기. 빼기. 나누기 기능도 만들어 보자.

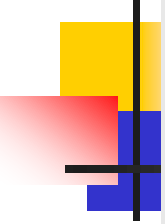


## 05-1 파이썬 프로그래밍의 핵심, 클래스

```
>>> 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
...
>>>
```



# 05-1 파이썬 프로그래밍의 핵심, 클래스



```
>>> a = FourCal()
>>> b = FourCal()
>>> a.setdata(4, 2)
>>> b.setdata(3, 8)
>>> a.add()
6
>>> a.mul()
8
>>> a.sub()
2
>>> a.div()
2
>>> b.add()
11
>>> b.mul()
24
>>> b.sub()
-5
>>> b.div()
0.375
```



## 05-1 파이썬 프로그래밍의 핵심, 클래스

---

### 생성자(Constructor)

- AttributeError 오류
  - FourCal 클래스의 객체 a에 setdata 메서드를 수행하지 않고 add 메서드를 수행하면, 'AttributeError: 'FourCal' object has no attribute 'first' 오류 발생
  - setdata 메서드를 수행해야 객체 a의 객체변수 first와 second가 생성되기 때문

```
>>> 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'
```



# 생성자

- 생성자의 개념 : 인스턴스를 생성하면서 필드값을 초기화시키는 함수
  - 생성자의 기본
    - 생성자의 기본 형태 : `__init__()`라는 이름
- Tip** • `__init__()`는 앞뒤에 언더바(\_)가 2개씩, `init` 는 Initialize 의 약자로 초기화 의미  
언더바가 2 개 붙은 것은 파이썬에서 예약해 놓은 것, 프로그램을 작성시 이 이름을 사용하여 새로운 함수나 변수명을 만들지 말 것

```
class 클래스명 :  
    def __init__(self) :  
        # 이 부분에 초기화할 코드 입력
```

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

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

```
>>> a = FourCal(4, 2)
>>> a.add()
6
>>> a.div()
2.0
```



# 05-1 파이썬 프로그래밍의 핵심, 클래스

클래스의 상속(Inheritance- '물려받다')

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

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



# 05-1 파이썬 프로그래밍의 핵심, 클래스

---

## 클래스 상속

- 상속(Inheritance)
  - '물려받다'라는 뜻
  - 어떤 클래스를 만들 때 다른 클래스의 기능을 물려받을 수 있게 만드는 것
- FourCal 클래스를 상속하는 MoreFourCal 클래스

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



# 05-1 파이썬 프로그래밍의 핵심, 클래스

---

## 클래스 상속

- MoreFourCal 클래스
  - FourCal 클래스를 상속했으므로 FourCal 클래스의 모든 기능을 사용할 수 있어야 함

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



# 05-1 파이썬 프로그래밍의 핵심, 클래스

---

- 클래스 상속
- a의 b제곱( $a^b$ )을 계산하는 MoreFourCal 클래스

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

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





# 05-1 파이썬 프로그래밍의 핵심, 클래스

---

## 메서드 오버라이딩

```
>>> a = FourCal(4, 0)
>>> a.div()
Traceback (most recent call last):

ZeroDivisionError: division by zero
```

- FourCal 클래스의 객체 a에 4와 0값을 설정하고 div 메서드를 호출하면 4/0 오류 발생
- 오류가 아닌 0을 돌려주도록 만들고 싶다면...
- FourCal 클래스를 상속하는 SafeFourCal 클래스를 만듦



# 05-1 파이썬 프로그래밍의 핵심, 클래스

---

## 메서드 오버라이딩

- ➔ 상속받을 대상인 클래스의 메서드와 이름은 같지만 그 행동을 다르게 해야 할 때
- ➔ 메서드 이름을 동일하게 다시 구현하는 것을 메서드 오버라이딩(Overriding)

```
>>> class SafeFourCal(FourCal):
...     def div(self):
...         if self.second == 0:
...             return 0
...         else:
...             return self.first / self.second
...

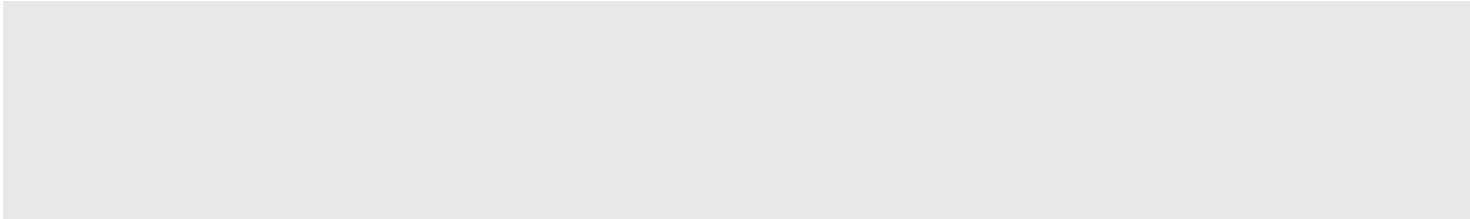
>>> a = SafeFourCal(4, 0)
>>> a.div()
0
```



## 나 혼자 코딩

---

곱하는 값이 0일 경우 'Fail' 문자열을 출력하는 클래스 FailFourCal을 위에서 만든 FourCal클래스를 상속해서 만드시오.



## 05-2 모듈

- 모듈이란 함수나 변수 또는 클래스 들을 모아 놓은 파일
- 모듈은 다른 파이썬 프로그램에서 불러와 사용할 수 있게끔 만들어진 파이썬 파일

### 모듈 만들고 불러 보기

```
# C:\doit\mod1.py
def add(a, b):
    return a + b

def sub(a, b):
    retrun a - b
```

```
cd C:\doit
```

- import는 이미 만들어진 파이썬 모듈을 사용할 수 있게 해 주는 명령어이다.

```
import 모듈이름
```

```
>>> import mod1
>>> print(mod1.add(3,4))
7
```



## 05-2 모듈

---

from 모듈이름 import 모듈함수

```
>>> from mod1 import add
```

```
>>> add(3, 4)
```

```
7
```

```
>>>from mod1 import add, sub
```

```
from mod1 import *
```

### ■ 모듈의 생성과 사용

Module1.py

```
1  ## 함수 선언 부분 ##
2  def func1() :
3      print("Module1.py의 func1()이 호출됨.")
4
5  def func2() :
6      print("Module1.py의 func2()가 호출됨.")
7
8  def func3() :
9      print("Module1.py의 func3()이 호출됨.")
```

## 05-2 모듈

A.py

```
1 import Module1
2
3 ## 메인 코드 부분 ##
4 Module1.func1()
5 Module1.func2()
6 Module1.func3()
```

A.py의 4~6행에서 호출할 때  
Module1.함수명() 형식으로\_모듈명을 앞에 붙였다.

### 출력 결과

Module1.py의 func1()이 호출됨.  
Module1.py의 func2()가 호출됨.  
Module1.py의 func3()이 호출됨.

- 모듈명을 생략하고 함수명만 쓸 때 1행 형식

```
from 모듈명 import 함수1, 함수2, 함수3
또는
from 모듈명 import *
```

## 05-2 모듈

- 4~6행 함수명으로만 호출

B.py

```
1 from Module1 import func1, func2, func3    # 또는 from Module1 import *
2
3 ## 메인 코드 부분 ##
4 func1()
5 func2()
6 func3()
```



# 선택적 import

- 모듈을 import 할 때 선택적 실행 관리 가능
  - » `__name__` 은 파이썬 파일을 직접 실행했을 때 사용되는 변수
    - `python abc.py` 로 실행한 경우 `__name__` 에는 `"__main__"` 이 저장됨
- 아래 구문은 파일을 직접 실행한 경우에만 실행되도록 조건 처리 (임포트 한 경우에는 실행되지 않음)

```
if __name__ == "__main__":  
    print(safe_sum('a', 1))  
    print(safe_sum(1, 4))  
    print(sum(10, 10.4))
```

```
>>> import mod1  
>>>
```

## 05-2 모듈

- if `__name__ == "__main__"`: 의 의미

모듈 만들고 불러 보기

```
# C:\Users\a\mod1.py
def add(a, b):
    return a + b

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

print(add(1, 4))
print(sub(4, 2))
```

import 모듈이름

```
> Python mod1.py
5
2
```

## 05-2 모듈

### ■ if \_\_name\_\_ == "\_\_main\_\_": 의 의미

```
>>> import mod1  
5  
2
```

- ➔ if \_\_name\_\_ == "\_\_main\_\_"을 사용하면 C:\python>python mod1.py처럼 직접 이 파일을 실행했을 때는 \_\_name\_\_ == "\_\_main\_\_"이 참이 되어 if문 다음 문장이 수행된다.
- ➔ 반대로 대화형 인터프리터나 다른 파일에서 이 모듈을 불러서 사용할 때는 \_\_name\_\_ == "\_\_main\_\_"이 거짓이 되어 if문 다음 문장이 수행되지 않는다..

## 05-2 모듈

- if `__name__ == "__main__"`: 의 의미

모듈 만들고 불러 보기

```
# C:\Users\a\mod1.py
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))
```

```
>>> import mod1
>>>
```

# 클래스나 변수를 포함한 모듈

## ■ 클래스 또는 변수를 포함한 모듈 생성

```
PI = 3.141592

class Math:
    def solv(self, r):
        return PI * (r ** 2)

def sum(a, b):
    return a+b

if __name__ == "__main__":
    print(PI)
    a = Math()
    print(a.solv(2))
    print(sum(PI, 4.4))
```

파이썬의 `__name__` 변수는 파이썬이 내부적으로 사용하는 특별한 변수명이다.

만약 C:\>>>python mod1.py처럼 직접 mod1.py 파일을 실행시킬 경우 mod2.py의 `__name__` 변수에는 `__main__` 이라는 값이 저장된다.

하지만 파이썬 셸이나 다른 파이썬 모듈에서 mod2를 import 할 경우에는 mod2.py의 `__name__` 변수에는 "mod1"이라는 mod1.py의 모듈이름 값이 저장된다.

```
C:\Python>python mod2.py
3.141592
12.566368
7.541592
```

직접 실행한 경우 실행 결과 출력

```
C:\Python>python
>>> import mod2
>>>
```

import 한 경우 실행되지 않음

# 클래스나 변수를 포함한 모듈

## ■ 모듈에 포함된 변수, 클래스, 함수 사용하기

```
>>> print(mod2.PI)  
3.141592
```

```
>>> a = mod2.Math()  
>>> print(a.solve(2))  
12.566368
```



## 05-2 모듈

---

### sys.path.append

```
>>> import sys      sys 모듈은 파이썬을 설치할 때 함께 설치되는 라이브러리 모듈이다.  
>>> sys.path  
['', 'C:\\\\Windows\\SYSTEM32\\python35.zip',  
'c:\\Python35\\DLLs',  
'c:\\Python35\\lib', 'c:\\Python35',  
'c:\\Python35\\lib\\site-packages']
```

```
sys.path.append("C:/Python/Mymodules")
```



## 05-2 모듈

---

sys.path.append

```
>>> import sys
>>> sys.path
['', 'C:\\Windows\\SYSTEM32\\python35.zip',
'c:\\Python35\\DLLs',
'c:\\Python35\\lib', 'c:\\Python35',
'c:\\Python35\\lib\\site-packages']
```

```
sys.path.append("C:/doit")
```





## 05-3 패키지

- 패키지는 파이썬 모듈을 계층적으로 관리하는 도구
  - » 디렉터리와 파이썬 모듈로 구성됨/각 디렉터리와 모듈은 .(dot) 를 사용해서 연결

가상의 game 패키지 예

```
game/  
  __init__.py  
  sound/  
    __init__.py  
    echo.py  
    wav.py  
  graphic/  
    __init__.py  
    screen.py  
    render.py  
  play/  
    __init__.py  
    run.py  
    test.py
```



## 05-3 패키지

---

### 테스트를 위해 패키지 만들기

```
C:/Users/a/game/__init__.py
C:/Users/a/game/sound/__init__.py
C:/Users/a/game/sound/echo.py
C:/Users/a/game/graphic/__init__.py
C:/Users/a/game/graphic/render.py
```

```
# echo.py
def echo_test():
    print ("echo")
```

```
# render.py
def render_test():
    print ("render")
```



## 05-3 패키지

---

### 패키지 안의 함수 실행하기 1

```
C:\> set PYTHONPATH=C:/Python → 안해도 됨
```

```
C:\> python
```

```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:54:25) [MSC  
v.1900 64 bit (AM...
```

```
Type "help", "copyright", "credits" or "license" for more  
information.
```

```
>>>
```



## 05-3 패키지

---

### 패키지 안의 함수 실행하기 2

```
>>> import game.sound.echo
>>> game.sound.echo.echo_test()
echo
```

```
>>> from game.sound import echo
>>> echo.echo_test()
echo
```

```
>>> from game.sound.echo import echo_test
>>> echo_test()
echo
```



## 05-3 패키지

### 패키지 안의 함수 실행하기-불가능 한 경우

- 다음과 같이 echo\_test 함수를 사용하는 것은 불가능

```
>>> import game
>>> game.sound.echo.echo_test()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module> AttributeError:
'module' object has no attribute 'sound'
```

- 다음과 같이 echo\_test 함수를 사용하는 것은 불가능

```
>>> import game.sound.echo.echo_test
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named echo_test
```

- `import a.b.c`처럼 import할 때 가장 마지막 항목인 `c`는 반드시 모듈 또는 패키지여야함



## 05-3 패키지

---

### `__init__.py` 의 용도

- 해당 디렉터리가 패키지의 일부임을 알려주는 역할
- game, sound, graphic 등 패키지에 포함된 디렉터리에 `__init__.py` 파일이 없다면 패키지로 인식되지 않는다



## 05-3 패키지

---

`__all__`

```
>>> from game.sound import *
>>> echo.echo_test()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'echo' is not defined
```

```
# C:/Python/game/sound/__init__.py
__all__ = ['echo']
```



## 05-3 패키지

---

### relative 패키지

```
# render.py
from game.sound.echo import echo_test

def render_test():
    print ("render")
    echo_test()
```

➔ graphic 디렉터리의 render.py 모듈이 sound 디렉터리의 echo.py 모듈을 사용하고 싶다면 render.py를 수정하면 된다.





## 05-4 예외처리

### 프로그램 실행 중 다양한 형태의 오류 발생

```
>>> f = open("나없는파일", 'r')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory:
'나없는파일'
```

```
>>> 4 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> a = [1,2,3]
>>> a[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module> IndexError: list
index out of range
```



## 05-4 예외 처리

---

### ■ 예외 처리 기법

#### ■ try, except문

- 오류 처리를 위한 구문
- try 블록 수행 중 오류가 발생하면 except 블록 수행  
try 블록에서 오류가 발생하지 않으면 except 블록 미수행

```
try:  
...  
  
except [발생 오류[as 오류 메시지 변수]]:  
...
```

#### ■ except 구문

```
except [발생 오류 [as 오류 메시지 변수]]:
```

#### ■ [ ] 기호

- 괄호 안의 내용을 생략할 수 있다는 관례 표기 기법



## 05-4 예외처리

---

■ `except` 구문은 다음 3가지 방법으로 사용

1) `try, except` 만 쓰는 방법

```
try:
    ...
except:
    ...
```

2) 발생 오류만 포함한 `except`문

```
try:
    ...
except 발생오류:
    ...
```



## 05-4 예외처리

- 3)발생 오류와 오류 메시지 변수까지 포함한 except문
  - 파이썬은 try, except 를 통해서 오류 처리 구문 지원

```
try:
    ...
except [발생 오류[as 오류 메시지 변수]]:
    ...
```

```
try:
    4 / 0
except ZeroDivisionError as e:
    print(e)
```



## 05-4 예외처리

---

try .. else

```
try:
    f = open('foo.txt', 'r')
except FileNotFoundError as e:
    print(str(e))
else:
    data = f.read()
    f.close()
```

➔ foo.txt라는 파일이 없다면 except절이 수행되고 foo.txt 파일이 있다면 else 절이 수행됨



## 05-4 예외처리

---

try .. Finally

➔ Finally절은 try문 수행 도중 예외 발생 여부에 상관없이 항상 수행된다.

```
f = open('foo.txt', 'w')
try:
    # 무언가를 수행한다.
finally:
    f.close()
```



## 05-4 예외처리

---

### 여러 개의 오류 처리하기

➔ 0으로 나누는 오류와 인덱싱 오류를 다음과 같이 처리할 수 있다.

```
try:
    a = [1,2]
    print(a[3])
    4/0
except ZeroDivisionError:
    print("0으로 나눌 수 없습니다.")
except IndexError:
    print("인덱싱할 수 없습니다.")
```



## 05-4 예외처리

---

### 오류 회피하기

발생한 오류에 대해 특별한 처리 없이 정상 흐름으로 돌리는 방법

```
try:
    f = open("나없는파일", 'r')
except FileNotFoundError:
    pass
```





## 05-4 예외처리

---

### 오류 일부러 발생시키기

```
class Bird:
    def fly(self):
        raise NotImplementedError
```

```
class Eagle(Bird):
    def fly(self):
        print("very fast")
```

```
eagle = Eagle()
eagle.fly()
```



## 05-5 내장함수

---

파이썬에서 기본적으로 포함하고 있는 함수

예) `print()`, `type()`



## 05-5 내장함수

---

abs

```
>>> abs(3)
3
>>> abs(-3)
3
>>> abs(-1.2)
1.2
```



## 05-5 내장함수

---

All → 모두 참인지 검사

```
>>> all([1, 2, 3])  
True  
>>> all([1, 2, 3, 0])  
False
```

→ 0은 거짓이므로 false



## 05-5 내장함수

---

any → 하나라도 참이 있을 경우 True를 리턴

```
>>> any([1, 2, 3, 0])
```

```
True
```

```
>>> any([0, ""])
```

```
False
```



## 05-5 내장함수

---

chr → ASCII 코드를 입력받아 문자를 출력

```
>>> chr(97)
'a'
>>> chr(48)
'0'
```

ASCII(아스키 코드) : 0~127사이의 숫자를 각 문자에 대응



## 05-5 내장함수

---

dir → 자체적으로 가지고 있는 변수나 함수를 보여준다.

```
>>> dir([1, 2, 3])
['append', 'count', 'extend', 'index', 'insert', 'pop',...]
>>> dir({'1':'a'})
['clear', 'copy', 'get', 'has_key', 'items', 'keys',...]
```



## 05-5 내장함수

---

divmod → 몫과 나머지를 튜플 형태로

```
>>> divmod(7, 3)
(2, 1)
>>> divmod(1.3, 0.2)
(6.0, 0.099999999999999978)
```





## 05-5 내장함수

---

### enumerate

```
>>> for i, name in enumerate(['body', 'foo', 'bar']):  
...     print(i, name)  
...  
0 body  
1 foo  
2 bar
```

- For문처럼 반복되는 구간에서 객체가 현재 어느 위치에 있는지 알려주는 인덱스 값이 필요할 때 enumerate함수를 사용하면 유용하다.



## 05-5 내장함수

---

### eval

➔ 실행 가능한 문자열을 입력으로 받아 문자열을 실행한 결과를 리턴하는 함수

```
>>> eval('1+2')
3
>>> eval("'hi' + 'a'")
'hia'
>>> eval('divmod(4, 3)')
(1, 1)
```



## 05-5 내장함수

---

### filter

```
def positive(x):  
    return x > 0
```

```
print(list(filter(positive, [1, -3, 2, 0, -5, 6])))
```

- positive 함수는 리스트를 입력 값으로 받아 양수 값만 리턴하는 함수
- lambda 인자 : 표현식

```
>>> print(list(filter(lambda x: x > 0, [1, -3, 2, 0, -5, 6])))
```



## 05-5 내장함수

---

hex → 정수값을 입력받아 16진수(hexadecimal) 변환하여 리턴하는 함수

```
>>> hex(234)
```

```
'0xea'
```

```
>>> hex(3)
```

```
'0x3'
```



## 05-5 내장함수(id)

---

id(object)는 객체를 입력받아 객체의 고유 주소값(레퍼런스)을 리턴하는 함수이다.

```
>>> a = 3
>>> id(3)
135072304
>>> id(a)
135072304
>>> b = a
>>> id(b)
135072304
```



## 05-5 내장함수

---

input

```
>>> a = input()
hi
>>> a
'hi'
>>> b = input("Enter: ")
Enter: hi
```

```
>>> b
'hi'
```



## 05-5 내장함수

---

int

```
>>> int('3')
```

```
3
```

```
>>> int(3.4)
```

```
3
```

```
>>> int('11', 2) ➔ 2진수로 표현된 '11'의 10진수 값은 3
```

```
3
```

```
>>> int('1A', 16) ➔ 16진수로 표현된 1A의 10진수 값은 26
```

```
26
```



## 05-5 내장함수

---

### isinstance

```
>>> class Person: pass    → 아무 기능이 없는 Person 클래스 생성
...
>>> a = Person()          → Person 클래스의 인스턴스 a 생성
>>> isinstance(a, Person) → a가 Person 클래스의 인스턴스인지 확인
True
```

```
>>> b = 3
>>> isinstance(b, Person) → b가 Person 클래스의 인스턴스인지 확인
False
```





## 05-5 내장함수

### lambda

➔ 함수를 한 줄로 간결하게 만들때 사용한다.

lambda 인자 : 표현식

```
>>> sum = lambda a, b: a+b
```

```
>>> sum(3,4)
```

```
7
```

```
>>> myList = [lambda a,b:a+b, lambda a,b:a*b]
```

```
>>> myList
```

```
[at 0x811eb2c>, at 0x811eb64>] ➔ 리스트 myList에 람다 함수가 2개 추가됨
```

```
>>>myList[0](3,4)
```

```
7
```

```
>>>myList[1](3,4)
```

```
12
```



## 05-5 내장함수

---

`len(s)` → 입력값 `s`의 길이를 리턴하는 함수

```
>>> len("python")
```

```
6
```

```
>>> len([1,2,3])
```

```
3
```

```
>>> len((1, 'a'))
```

```
2
```



## 05-5 내장함수

---

`list(s)` → 반복 가능한 자료형 `s`를 입력받아 리스트로 만들어 리턴하는 함수

```
>>> list("python")
['p', 'y', 't', 'h', 'o', 'n']
>>> list((1,2,3))
[1, 2, 3]
```

```
>>> a = [1, 2, 3]
>>> b = list(a)
>>> b
[1, 2, 3]
```



## 05-5 내장함수

---

### filter

```
def positive(x):  
    return x > 0
```

```
print(list(filter(positive, [1, -3, 2, 0, -5, 6])))
```

- positive 함수는 리스트를 입력 값으로 받아 양수 값만 리턴하는 함수
- lambda 인자 : 표현식

```
>>> print(list(filter(lambda x: x > 0, [1, -3, 2, 0, -5, 6])))
```



## 05-5 내장함수

---

`map(f, iterable)`

→ 함수(f)와 반복 가능한(iterable)자료형을 입력 받는다.

→ map은 입력받은 자료형의 각 요소가 함수 f에 의해 수행된 결과를 묶어서 리턴하는 함수

```
>>> def two_times(x): return x*2
```

```
>>> list(map(two_times, [1, 2, 3, 4]))  
[2, 4, 6, 8]
```

```
>>> list(map(lambda a: a*2, [1, 2, 3, 4]))  
[2, 4, 6, 8]
```



## 05-5 내장함수

---

max

```
>>> max([1, 2, 3])
3
>>> max("python")
'y'
```

min

```
>>> min([1, 2, 3])
1
>>> min("python")
'h'
```



## 05-5 내장함수

---

oct → 정수 형태의 숫자를 8진수 문자열로 바꾸어 리턴하는 함수

```
>>> oct(34)
'0o42'
>>> oct(12345)
'0o30071'
```



## 05-5 내장함수

---

open

mode	설명
w	쓰기 모드로 파일 열기
r	읽기 모드로 파일 열기
a	추가 모드로 파일 열기
b	바이너리 모드로 파일 열기

```
>>> f = open("binary_file", "rb")
```





## 05-5 내장함수

---

ord : ord(c)는 문자의 아스키 코드값을 리턴하는 함수이다.  
chr 함수와 반대

```
>>> ord('a')  
97  
>>> ord('0')  
48
```



## 05-5 내장함수

---

pow

```
>>> pow(2, 4)
```

```
16
```

```
>>> pow(3, 3)
```

```
27
```



## 05-5 내장함수

---

range

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

```
>>> list(range(5, 10))  
[5, 6, 7, 8, 9]
```

```
>>> list(range(1, 10, 2))  
[1, 3, 5, 7, 9]  
>>> list(range(0, -10, -1))  
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```



## 05-5 내장함수

---

sorted

```
>>> sorted([3, 1, 2])  
[1, 2, 3]  
>>> sorted(['a', 'c', 'b'])  
['a', 'b', 'c']  
>>> sorted("zero")  
['e', 'o', 'r', 'z']  
>>> sorted((3, 2, 1))  
[1, 2, 3]
```



## 05-5 내장함수

---

str

```
>>> str(3)
'3'
>>> str('hi')
'hi'
>>> str('hi'.upper())
'HI'
```



## 05-5 내장함수

---

tuple

```
>>> tuple("abc")
('a', 'b', 'c')
>>> tuple([1, 2, 3])
(1, 2, 3)
>>> tuple((1, 2, 3))
(1, 2, 3)
```



## 05-5 내장함수

---

`type` : `type(object)`은 입력값의 자료형이 무엇인지 알려주는 함수이다.

```
>>> type("abc")
<class 'str'>
>>> type([ ])
<class 'list'>
```



## 05-5 내장함수

---

zip → 동일한 개수로 이루어진 자료형을 묶어 주는 역할을 하는 함수

```
>>> list(zip([1, 2, 3], [4, 5, 6]))
[(1, 4), (2, 5), (3, 6)]
>>> list(zip([1, 2, 3], [4, 5, 6], [7, 8, 9]))
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
>>> list(zip("abc", "def"))
[('a', 'd'), ('b', 'e'), ('c', 'f')]
```





## 05-6 외장함수

---

라이브러리 함수

Import 해서 사용하는 것



## 05-6 외장함수

---

### sys.argv

- sys모듈은 파이썬 인터프리터가 제공하는 변수와 함수를 직접 제어할 수 있게 해 주는 모듈이다.
- 명령 행에서 인수 전달하기 -sys.argv

```
# argv_test.py
import sys
print(sys.argv)
```

```
C:/Python/Mymodules>python argv_test.py you need python
['argv_test.py', 'you', 'need', 'python']
```



## 05-6 외장함수

---

### pickle

➔ 객체의 형태를 그대로 유지하면서 파일에 저장하고 불러올 수 있게 하는 모듈

```
>>> import pickle
>>> f = open("test.txt", 'wb')
>>> data = {1: 'python', 2: 'you need'}
>>> pickle.dump(data, f)
>>> f.close()
```

```
>>> import pickle
>>> f = open("test.txt", 'rb')
>>> data = pickle.load(f)
>>> print(data)
{2: 'you need', 1: 'python'}
```



## 05-6 외장함수

### OS

➔ 환경 변수나 디렉토리, 파일 등의 OS자원을 제어할 수 있게 해 주는 모듈

```
>>> import os
>>> os.environ          #내 시스템의 환경 변수를 알고 싶을 때
environ({'PROGRAMFILES': 'C:\\\\Program Files', 'APPDATA': ...
생략 ...})
>>> os.chdir("C:\\WINDOWS")
>>> os.getcwd()
'C:\\WINDOWS'
>>> os.system("dir")
>>> f = os.popen("dir")
>>> print(f.read())
```



## 05-6 외장함수

---

shutil → 파일을 복사해 주는 파이썬 모듈

```
>>> import shutil  
>>> shutil.copy("src.txt", "dst.txt")
```



## 05-6 외장함수

---

**glob** → 특정 디렉터리에 있는 파일 이름 모두를 알아야 할 때가 있다. 이럴 때 사용하는 모듈

```
>>> import glob
>>> glob.glob("C:/Python/q*")
['C:\\Python\\quiz.py', 'C:\\Python\\quiz.py.bak']
>>>
```



## 05-6 외장함수

---

tempfile

```
>>> import tempfile  
>>> filename = tempfile.mktemp()  
>>> filename  
'C:\WINDOWS\TEMP\~-275151-0'
```

```
>>> import tempfile  
>>> f = tempfile.TemporaryFile()  
>>> f.close()
```



## 05-6 외장함수

---

### time 1

```
>>> import time
```

```
>>> time.time() →현재 시간을 실수 형태로 리턴하는 함수
```

```
988458015.73417199
```

```
>>> time.localtime(time.time()) →time.time()에 의해서 반환된 실수값을  
이용해서 연도, 월, 일, 시, 분, 초,.. 의 형태로 바꾸어 주는 함수
```

```
time.struct_time(tm_year=2013, tm_mon=5, tm_mday=21,  
tm_hour=16,
```

```
tm_min=48, tm_sec=42, tm_wday=1, tm_yday=141, tm_isdst=0)
```





## 05-6 외장함수

---

time 2

```
>>> time.asctime(time.localtime(time.time()))  
'Sat Apr 28 20:50:20 2001'
```

```
>>> time.ctime()  
'Sat Apr 28 20:56:31 2001'
```

```
>>> import time  
>>> time.strftime('%x', time.localtime(time.time()))  
'05/01/01'  
>>> time.strftime('%c', time.localtime(time.time()))  
'05/01/01 17:22:21'
```



## 05-6 외장함수

---

time.sleep

```
import time
for i in range(10):
    print(i)
    time.sleep(1)
```

1초 간격으로 0부터 9까지의 숫자를 출력한다



## 05-6 외장함수

---

calendar

```
>>> calendar.weekday(2015, 12, 31)
3
```

```
>>> calendar.monthrange(2015,12)
(1, 31)
```

```
>>> calendar.prcal(2015)    ➔ (연도)를 사용해도 같은 결과
```



## 05-6 외장함수

---

calendar

```
>>> calendar.prmonth(2015, 12)
```

December 2015

Mo	Tu	We	Th	Fr	Sa	Su
----	----	----	----	----	----	----

						1
--	--	--	--	--	--	---

2	3	4	5	6	7	8
---	---	---	---	---	---	---

9	10	11	12	13	14	15
---	----	----	----	----	----	----

16	17	18	19	20	21	22
----	----	----	----	----	----	----

23	24	25	26	27	28	29
----	----	----	----	----	----	----

30	31					
----	----	--	--	--	--	--



## 05-6 외장함수

---

calendar

```
>>> calendar.weekday(2015, 12, 31)
```

```
3
```

→ 목요일

```
>>> calendar.monthrange(2015,12)
```

```
(1, 31)
```

→ 2015년 12월 1일은 화요일, 이 달은 31일까지 있다.



## 05-6 외장함수

---

random

```
>>> import random
>>> random.random() → 0.0~1.0사이의 실수 중에서 난수 값을 리턴하는 예
0.53840103305098674
```

```
>>> random.randint(1, 10) → 1~10 사이의 정수 중에서 난수 값을 리턴
6
```

```
>>> data = [1, 2, 3, 4, 5]
>>> random.shuffle(data) → 리스트의 항목을 무작위로 섞고 싶을 때
>>> data
[5, 1, 3, 4, 2]
```



## 05-6 외장함수

---

random

```
>>> import random  
>>> lotto = sorted(random.sample(range(1,46), 6))  
>>> print(lotto)
```



## 05-6 외장함수

---

webbrowser → 기본 웹 브라우저가 자동으로 실행되게 하는 모듈

```
>>> import webbrowser  
>>> webbrowser.open("http://google.com")
```

```
>>> webbrowser.open_new("http://google.com")
```





## 05-6 외장함수

---

### threading

```
import threading
import time

def say(msg):
    while True:
        time.sleep(1)
        print(msg)

for msg in ['you', 'need', 'python']:
    t = threading.Thread(target=say, args=(msg,))
    t.daemon = True
    t.start()

for i in range(100):
    time.sleep(0.1)
    print(i)
```