

# 인공지능 기초 프로그래밍

Python 기초 (모듈/패키지)

# 파이썬 프로그램 기초-모듈

## ■ 모듈 만들기

### ■ 모듈

- 함수나 변수 또는 클래스를 모아 놓은 파일
- 다른 파이썬 프로그램에서 불러와 사용할 수 있게끔 만든 파이썬 파일

- add와 sub 함수만 있는 파일(모듈) mod1.py를 만들어 특정 디렉터리에 저장

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

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

# 파이썬 프로그램 기초-모듈

## ■ 모듈 불러오기

### ■ import

- 이미 만들어 놓은 파이썬 모듈을 사용할 수 있게 해주는 명령어
  - mod1.py에서 확장자 .py를 제거한 mod1이 모듈 이름

import 모듈 이름

파일 내에 정의된 모든 함수 사용 가능  
사용 시: 모듈이름.함수명

- mod1.add처럼 쓰지 않고 모듈 이름 없이 함수 이름만 쓰고 싶은 경우

from 모듈 이름 import 모듈 함수

사용 시: "함수명"으로 바로 사용 가능

```
>>> from mod1 import add
>>> add(3, 4)
7
```

# 파이썬 프로그램 기초-모듈

## ■ 모듈 불러오기

### ■ import

- 모듈의 함수를 여러 개 불러오고 싶은 경우

1) 콤마로 구분하여 필요한 함수 불러오기

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

2) \* 문자 사용하기

- \* 문자는 '모든 것'이라는 뜻으로, 모듈 내의 모든 함수를 불러오겠다는 의미

```
from mod1 import *
```

## 파이썬 프로그램 기초-모듈

- `if __name__ == "__main__":`

- `if __name__ == "__main__"` 사용하도록 `mod1.py` 수정하기

- `__name__` 변수

- 파이썬이 내부적으로 사용하는 특별한 변수

- 1) `python mod1.py`와 같이 이 파일을 실행한 경우

- `__name__` 변수에는 `__main__` 값이 저장되어 `__name__ == "__main__"`이 참이 됨

```
C:\doit>python mod1.py
```

```
5
```

```
2
```

```
# 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))
```

## 파이썬 프로그램 기초-모듈

- `if __name__ == "__main__":`

- `if __name__ == "__main__"` 사용하도록 `mod1.py` 수정하기

- `__name__` 변수

- 파이썬이 내부적으로 사용하는 특별한 변수

- 2) 대화형 인터프리터나 다른 파일에서 이 모듈을 불러서 사용할 경우

- `__name__` 변수에는 모듈 이름 `mod1`이 저장되어 `__name__ == "__main__"`이 거짓이 됨

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

```
# 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))
```

# 파이썬 프로그램 기초-모듈

## ▪ 다른 파일에서 모듈 불러오기

- mod2.py 파일 만들어서 저장하기
  - 원의 넓이를 계산하는 Math 클래스
  - 두 값을 더하는 add 함수
  - 원주율 값에 대항되는 PI 변수

```
# modtest.py
import mod2
result = mod2.add(3, 4)
print(result)
```

```
# mod2.py
PI = 3.141592

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

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

## Practice #1 ()

### modTmp.py

```
def add(a, b):  
    return a + b  
  
def mul(a, b):  
    return a * b  
  
if __name__ == "__main__":  
    print(add(3, 4))  
    print(mul(3, 4))
```

### Mod\_import.py

```
# import modTmp  
#  
# print(modTmp.add(1, 2))  
# print(modTmp.mul(2, 23))  
  
# from modTmp import add  
# print(add(1, 2))  
# # print(mul(1, 2))  
  
from modTmp import *  
print(add(1, 2))  
print(mul(1, 2))
```



# 인공지능 기초 프로그래밍

Python 기초 (예외처리/내장함수)

# 파이썬 프로그램 기초-예외처리

## ■ 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

# 파이썬 프로그램 기초-예외처리

## ■ 예외 처리 기법

### ■ try, except문

#### ■ except 구문 사용법

##### 1) try, except만 쓰는 방법

- 오류 종류에 상관없이  
오류가 발생하면 except 블록 수행

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

##### 2) 발생 오류만 포함하는 방법

- 오류가 발생했을 때 except문에  
미리 정해 놓은 오류 이름과 일치할 때만  
except 블록을 수행한다는 뜻

```
try:  
    ...  
except 발생 오류:  
    ...
```

# 파이썬 프로그램 기초-예외처리

## ■ 예외 처리 기법

### ■ try, except문

#### ■ except 구문 사용법

##### 3) 발생 오류와 오류 메시지 변수를 포함한 except문

- 오류가 발생했을 때 except문에 미리 정해 놓은 오류 이름과 일치할 때만 except 블록을 수행하고, 오류 메시지의 내용까지 알고 싶을 때 사용하는 방법

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

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

결괏값: division by zero

# 파이썬 프로그램 기초-예외처리

## ■ 예외 처리 기법

### ■ try ... finally

- finally절은 try문 수행 도중 예외 발생 여부에 상관없이 항상 수행됨
- 보통 finally절은 사용한 리소스를 close해야 할 때에 많이 사용함
  - 예) foo.txt 파일을 쓰기 모드로 열어 try문을 수행한 후  
예외 발생 여부와 상관없이 finally절에서 f.close()로 열린 파일을 닫을 수 있음

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

# 파이썬 프로그램 기초-예외처리

## ■ 예외 처리 기법

### ■ 여러 개의 오류 처리하기

- try문 안에서  
여러 개의 오류를 처리하기 위한 방법

```
try:  
    ...  
except 발생 오류 1:  
    ...  
except 발생 오류 2:  
    ...
```

먼저 발생한 오류 관련 except 블록만 수행

- 예) 0으로 나누는 오류와 인덱싱 오류 처리

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

# 파이썬 프로그램 기초-예외처리

## ■ 예외 처리 기법

### ■ 여러 개의 오류 처리하기

#### ■ 오류 메시지 가져오기

```
try:
    a = [1,2]
    print(a[3])
    4/0
except ZeroDivisionError as e:
    print(e)
except IndexError as e:
    print(e)
```

List의 인덱스가 정의된 리스트의 크기를 넘어가기 때문에 에러가 발생

➔ IndexError가 먼저 발생했기 때문에 에러 메시지는 하나만 발생

#### ■ 2개 이상의 오류를 동시에 처리하기 위해 괄호를 사용하여 함께 묶어 처리

```
try:
    a = [1,2]
    print(a[3])
    4/0
except (ZeroDivisionError, IndexError) as e:
    print(e)
```



2개 이상의 오류를  
동시 처리하기 위한 방법

# 파이썬 프로그램 기초-예외처리

## ■ 오류 회피하기

- 특정 오류가 발생할 경우 그냥 통과시키는 방법

```
try:  
    f = open("나없는파일", 'r')  
except FileNotFoundError: ← 파일이 없더라도 오류를 발생시키지 않고 통과한다.  
    pass
```

- try문 안에서 FileNotFoundError가 발생할 경우에 pass를 사용하여 오류를 그냥 회피하도록 함



# 파이썬 프로그램 기초-예외처리

## 오류 일부러 발생시키기

- raise 명령어를 사용해 오류를 강제로 발생시킬 수 있음
  - 예) Bird 클래스를 상속받는 자식 클래스가 반드시 fly라는 함수를 구현하도록 하고 싶은 경우
    - 파이썬 내장 오류 NotImplementedError와 raise문 활용
    - fly 함수를 구현하지 않은 상태로 fly 함수 호출 시 NotImplementedError 오류 발생

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

```
class Eagle(Bird): ← Eagle 클래스는 Bird 클래스를 상속 받음  
    pass
```

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



```
Traceback (most recent call last):  
  File "...", line 33, in <module>  
    eagle.fly()  
  File "...", line 26, in fly  
    raise NotImplementedError  
NotImplementedError
```

# 파이썬 프로그램 기초-예외처리

## ■ 예외 만들기

- 파이썬 내장 클래스인 Exception 클래스를 상속하여 생성 가능

```
class MyError(Exception):  
    pass
```

- 예) 별명을 출력해주는 함수에서 MyError 사용하기

```
def say_nick(nick):  
    if nick == '바보':  
        raise MyError()  
    print(nick)
```

```
say_nick("바보")
```



```
Traceback (most recent call last):  
  File "...", line 11, in <module>  
    say_nick("바보")  
  File "...", line 7, in say_nick  
    raise MyError()  
__main__.MyError
```

# 파이썬 프로그램 기초-예외처리

## ■ 예외 만들기

- 파이썬 내장 클래스인 Exception 클래스를 상속하여 생성 가능
  - 예) 예외처리 기법을 사용하여 MyError 발생 예외 처리

```
try:  
    say_nick("천사")  
    say_nick("바보")  
except MyError:  
    print("허용되지 않는 별명입니다.")
```



천사  
허용되지 않는 별명입니다.

# 파이썬 프로그램 기초-예외처리

## ■ 예외 만들기

- 파이썬 내장 클래스인 Exception 클래스를 상속하여 생성 가능
  - 예) MyError에서 \_\_str\_\_ 메서드 구현하여 오류 메시지 사용하기

```
class MyError(Exception):  
    def __str__(self):  
        return "허용되지 않는 별명입니다."
```

```
try:  
    say_nick("천사")  
    say_nick("바보")  
except MyError as e:  
    print(e)
```



천사  
허용되지 않는 별명입니다.

## Practice #1 (예외처리)

```
def process_error():  
  
    f = open("tmp.txt", "r")  
  
    try:  
        a = [1, 2]  
        print(a[3])  
        b = 4 / 0  
        f = open("foo.txt", "r")  
  
    # except ZeroDivisionError as e2:  
    #     print(e2)  
    # except IndexError as e1:  
    #     print(e1)  
    # except FileNotFoundError as e3:  
    #     print(e3)  
    except (ZeroDivisionError, IndexError, FileNotFoundError) as e:  
        print(e)  
    # except:  
    #     print("ALL Error")  
  
    finally:  
        f.close()
```

```
class Bird:  
    def fly(self):  
        raise NotImplementedError  
...  
class Eagle(Bird):  
    def fly(self):  
        print("Eagle is Birds")  
  
eagle = Eagle()  
eagle.fly()
```

```
class MyError(Exception):  
    def __str__(self):  
        return "List 접근X"  
  
def list_access(in_param_bool, in_param_list):  
    if in_param_bool:  
        print(in_param_list[:])  
    else:  
        raise MyError()  
  
try:  
    list_access(0, [1, 2, 3])  
    list_access(1, [1, 2, 3])  
except MyError as e:  
    print(e)  
...
```

# 파이썬 프로그램 기초-내장함수

## ■ 파이썬 내장 함수

- 외부 모듈과 달리 import 등 기타 설정 없이 바로 사용 가능

## ■ abs(x)

- x의 절댓값을 돌려주는 함수

```
>>> abs(3)
3
```

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

## ■ all(x) AND

- 반복 가능한(iterable) 자료형 x가 모두 참이면 True, 하나라도 거짓이면 False 반환

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

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

## ■ any(x) OR

- x가 모두 거짓이면 False, 하나라도 참이면 True 반환

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

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

# 파이썬 프로그램 기초-내장함수

## ▪ chr(x)

- 아스키(ASCII) 코드를 입력받아 코드에 해당하는 문자 반환

```
>>> chr(97)
'a' ← 아스키 코드 97은 소문자 a
>>> chr(48)
'0' ← 아스키 코드 48은 숫자 0
```

## ▪ dir(x) 자료형이 가지고 있는 내장함수 반환

- 객체가 자체적으로 가지고 있는 변수나 함수 반환

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

## ▪ divmod(a, b)

- a를 b로 나눈 몫과 나머지를 튜플 형태로 반환

```
>>> divmod(7, 3)
(2, 1) ← 7 나누기 3의 몫은 2, 나머지는 1
```

$a = 7 // 3$

$b = 7 \% 3$

# 파이썬 프로그램 기초-내장함수

## ▪ enumerate(x)

- '열거하다'라는 뜻
- 순서가 있는 자료형(리스트, 튜플, 문자열)을 입력으로 받아 인덱스 값을 포함하는 enumerate 객체 반환

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

For statement에서  
index 값을 같이 사용하고 싶을 때.

## ▪ eval(expression)

- 실행 가능한 문자열(expression)을 입력으로 받아 문자열을 실행한 결과값 반환
- 문자열로 파이썬 함수나 클래스를 동적으로 실행할 때 사용

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



# 파이썬 프로그램 기초-내장함수

## ▪ filter(f, iterable)

- Iterable 자료형의 요소가 함수 f에 입력되었을 때 반환 값이 참인 것만 묶어서 반환

```
#filter1.py
def positive(x):
    return x > 0

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

결과값: [1, 2, 6]

- lambda도 사용 가능

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

## ▪ hex(x)

- 정수 값을 입력받아 16진수로 변환

```
>>> hex(234)
'0xea'
>>> hex(3)
'0x3'
```

# 파이썬 프로그램 기초-내장함수

## ▪ id(object)

- object(객체)의 고유 주소 값 반환

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

— 3, a, b는 모두 같은 객체를 가리킴

## ▪ input([prompt])

- 사용자 입력을 받는 함수
- 문자열 인자 생략 가능
- 매개변수로 문자열을 주면 프롬프트 띄움

```
>>> a = input() ← 사용자가 입력한 정보를 변수 a에 저장
hi
>>> a
'hi'
>>> b = input("Enter: ") ← Enter: 프롬프트를 띄우고 사용자 입력을 받음
Enter: hi
```

# 파이썬 프로그램 기초-내장함수

## ▪ int(x)

- 문자열 형태의 숫자나 소수점이 있는 숫자 등을 정수 형태로 변환

```
>>> int('3') ← 문자열 형태 '3'  
3  
>>> int(3.4) ← 소수점이 있는 숫자 3.4  
3
```

## ▪ int(x, radix)

- radix 진수 문자열을 10진수로 변환

```
>>> int('1A', 16)  
26
```

## ▪ instance(object, class)

- object: 인스턴스 / class: 클래스 이름
- 인스턴스가 클래스의 인스턴스인지 판단하여 참이면 True, 거짓이면 False 반환

```
>>> 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
```

# 파이썬 프로그램 기초-내장함수

## ▪ len(x)

- 입력값의 길이(요소의 전체 개수) 반환

```
>>> len("python")  
6
```

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

## ▪ list(iterable)

- Iterable 자료형을 리스트로 변환

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

## ▪ map(f, iterable)

- Iterable 자료형의 각 요소를 함수 f가 수행한 결과를 묶어서 반환

```
>>> def two_times(x): return x*2  
...  
>>> list(map(two_times, [1, 2, 3, 4]))  
[2, 4, 6, 8]
```

- lambda 활용 가능

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

# 파이썬 프로그램 기초-내장함수

## ▪ max(iterable)

- Iterable 자료형의 최대값 반환

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

```
>>> max("python")  
'y'
```

## ▪ min(iterable)

- Iterable 자료형의 최소값 반환

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

```
>>> min("python")  
'h'
```

## ▪ oct(x)

- 정수 형태의 숫자를 8진수 문자열로 변환

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

# 파이썬 프로그램 기초-내장함수

## ▪ open(filename, [mode])

- filename: 파일 이름 / mode: 읽기 방법
- mode를 생략하면 기본값인 읽기 전용 모드(r)로 파일 객체 반환
- b는 w, r, a와 함께 사용

모드	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")
```

## ▪ ord(c)

- 문자의 아스키 코드 값 반환

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

```
>>> ord('\0')  
48
```

## ▪ pow(x, y)

- x의 y 제곱한 결과값 반환

```
>>> pow(2, 4)  
16 ← 2의 4 제곱
```

```
>>> pow(3, 3)  
27 ← 3의 3 제곱
```

# 파이썬 프로그램 기초-내장함수

## ▪ range([start,] stop [,step])

- 입력받은 숫자에 해당하는 범위 값을 iterable 객체로 반환

### 1) 인수가 하나일 경우

- 시작 숫자를 지정해 주지 않으면 range 함수는 0부터 시작

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

### 2) 인수가 2개일 경우

- 시작 숫자와 끝 숫자
- 끝 숫자는 해당 범위에 포함되지 않음

```
>>> list(range(5, 10))  
[5, 6, 7, 8, 9] ← 끝 숫자 10은 포함되지 않음
```

### 3) 인수가 3개일 경우

- 세 번째 인수는 숫자 사이의 거리

```
>>> list(range(1, 10, 2))  
[1, 3, 5, 7, 9] ← 1부터 9까지, 숫자 사이의 거리는 2  
>>> list(range(0, -10, -1))  
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9] ← 0부터 -9까지, 숫자 사이의 거리는 -1
```

## 파이썬 프로그램 기초-내장함수

### ▪ round(number[, ndigits])

- 숫자를 입력받아 반올림해 주는 함수

```
>>> round(4.6)
5
>>> round(4.2)
4
```

- ndigits는 반올림하여 표시하고 싶은 소수점의 자릿수

```
>>> round(5.678, 2)
5.68
```

### ▪ sorted(iterable)

- 입력값을 정렬한 후 그 결과를 리스트로 반환

```
>>> 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]
```



# 파이썬 프로그램 기초-내장함수

## ▪ str(object)

- 객체를 문자열 형태로 변환

```
>>> str(3)
'3'
```

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

## ▪ sum(iterable)

- 리스트나 튜플의 모든 요소의 합 반환

```
>>> sum([1,2,3])
6
```

```
>>> sum((4,5,6))
15
```

## ▪ tuple(iterable)

- Iterable 자료형을 튜플 형태로 변환

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

# 파이썬 프로그램 기초-내장함수

## ▪ type(object)

- 입력값의 자료형 반환

```
>>> type("abc")
<class 'str'> ← "abc"는 문자열 자료형
>>> type([])
<class 'list'> ← []는 리스트 자료형
>>> type(open("test", 'w'))
<class '_io.TextIOWrapper'> ← 파일 자료형
```

## ▪ zip(\*iterable)

- 동일한 개수로 이루어진 자료형을 묶어주는 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')]
```

## Practice #2 (내장함수)

```
# ABS
print(-3, abs(-3), abs(3))

# ALL / ANY
list_a = [1, 2, 3, 0]
list_b = [1, 2, 3]
print(all(list_a), all(list_b), any(list_a), any(list_b), any([0, "", []]))

# CHR (ASCII to Char)
print(f'{chr(72)}{chr(69)}{chr(76)}{chr(76)}{chr(79)}')
```

# dir (객체, 자료형이 가지고 있는 내장함수 리스트 출력)

```
list_var = [1, 2, 3]
str_var = "ABC"
dict_var = {'key': "value"}
print(dir(list_var))
print(dir(str_var))
print(dir(dict_var))
```

# 인공지능 기초 프로그래밍

Python 기초 (라이브러리): 247-261

# 파이썬 프로그램 기초-라이브러리

## ■ 파이썬 라이브러리

- 라이브러리(library) = 도서관 (원하는 정보를 찾아보는 곳)
- 파이썬 라이브러리는 전 세계의 파이썬 사용자들이 만든 유용한 프로그램을 모아 놓은 것
- 파이썬을 설치할 때 자동으로 컴퓨터에 설치됨
- 자주 사용되고 꼭 알아 두면 좋은 라이브러리 소개

# 파이썬 프로그램 기초-라이브러리

## ▪ sys

- 파이썬 인터프리터가 제공하는 변수와 함수를 직접 제어할 수 있게 해주는 모듈

### 1) **sys.argv** - 명령 행에서 인수 전달하기

```
C:/User/home>python test.py abc pey guido
```

- 명령 프롬프트 창에서 .py 뒤에 또 다른 값을 함께 넣어주면 sys.argv 리스트에 그 값이 추가됨

### 예) argv\_test.py

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

- 명령 프롬프트 창에서 argv\_test.py 실행

```
C:/doit/Mymod>python argv_test.py you need python
['argv_test.py', 'you', 'need', 'python']
```

- python 명령어 뒤의 입력 값이 공백을 기준으로 나뉘어 sys.argv 리스트의 요소가 됨

# 파이썬 프로그램 기초-라이브러리

## ▪ sys

### 2) **sys.exit** - 강제로 스크립트 종료하기

```
>>> sys.exit()
```

- 프로그램 파일 안에서 사용하면 프로그램을 중단시킴
- [Ctrl + Z]나 [Ctrl + D]를 눌러서 대화형 인터프리터를 종료하는 것과 같은 기능

# 파이썬 프로그램 기초-라이브러리

## ▪ sys

### 3) **sys.path** - 자신이 만든 모듈 불러와 사용하기

- 파이썬 모듈들이 저장되어 있는 위치를 나타냄
- 이 위치에 있는 파이썬 모듈은 경로에 상관없이 어디에서나 불러올 수 있음

```
>>> import sys
>>> sys.path
['', 'C:\\Windows\\SYSTEM32\\python37.zip', 'c:\\Python37\\DLLs',
'c:\\Python37\\lib', 'c:\\Python37', 'c:\\Python37\\lib\\site-packages']
```

- 경로 이름을 추가하면, 해당 경로에 있는 파이썬 모듈을 불러와서 사용 가능!

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



## ▪ pickle

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

### 1) pickle.dump

- 예) 딕셔너리 객체를 그대로 파일에 저장

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

### 2) pickle.load

- 예) pickle.dump로 저장한 파일을 원래 딕셔너리 객체 상태로 불러오기

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

# 파이썬 프로그램 기초-라이브러리

## ▪ OS

- OS 모듈은 환경 변수나 디렉터리, 파일 등의 OS 자원을 제어할 수 있게 해주는 모듈

### 1) os.environ

- 현재 시스템의 환경 변수에 대한 정보를 딕셔너리 객체로 반환

```
>>> import os
>>> os.environ
environ({'PROGRAMFILES': 'C:\\Program Files', 'APPDATA': ... 생략 ...})
```

```
>>> os.environ['PATH']
'C:\\ProgramData\\Oracle\\Java\\javapath; ... 생략 ...'
```

# 파이썬 프로그램 기초-라이브러리

## ▪ OS

### 2) **os.chdir**

- 디렉터리 위치 변경

```
>>> os.chdir("C:\WINDOWS")
```

### 3) **os.getcwd**

- 현재 디렉터리 위치 반환

```
>>> os.getcwd()  
'C:\WINDOWS' ← 현재 디렉터리 위치에 따라 결과가 다를 수 있음
```

# 파이썬 프로그램 기초-라이브러리

## ▪ OS

### 4) **os.system**

- 시스템 자체의 프로그램이나 기타 명령어를 파이썬에서 호출할 수 있음
- 예) 현재 디렉터리에서 시스템 명령어 dir 실행

```
>>> os.system("dir")
```

### 5) **os.popen**

- 시스템 명령어를 실행한 결과값을 읽기 모드 형태의 파일 객체로 반환

```
>>> f = os.popen("dir")
```

# 파이썬 프로그램 기초-라이브러리

## ▪ OS

### ▪ 기타 유용한 os 관련 함수

함수	설명
os.mkdir(디렉터리)	디렉터리를 생성한다.
os.rmdir(디렉터리)	디렉터리를 삭제한다. 단 디렉터리가 비어 있어야 삭제가 가능하다.
os.unlink(파일 이름)	파일을 지운다.
os.rename(src, dst)	src라는 이름의 파일을 dst라는 이름으로 바꾼다.

# 파이썬 프로그램 기초-라이브러리

## ▪ shutil

- 파일을 복사해주는 파이썬 모듈
- 예) src라는 이름의 파일을 dst로 복사
  - dst가 디렉터리 이름이라면 src라는 파일 이름으로 dst 디렉터리에 복사
  - 동일한 파일 이름이 있을 경우에는 덮어 씌움

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

# 파이썬 프로그램 기초-라이브러리

## ▪ glob

- 특정 디렉터리에 있는 파일들을 리스트로 반환
- \*, ? 등 메타 문자를 써서 원하는 파일만 읽어 들일 수 있음
- 예) C:\doit 디렉터리에 있는 파일 중 이름이 mark로 시작하는 파일 모두 찾기

```
>>> import glob
>>> glob.glob("c:/doit/mark*")
['c:/doit\\marks1.py', 'c:/doit\\marks2.py', 'c:/doit\\marks3.py']
>>>
```

# 파이썬 프로그램 기초-라이브러리

## ▪ tempfile

- 파일을 임시로 만들어서 사용

### 1) **tempfile.mkstemp**

- 중복되지 않는 임시 파일의 이름을 무작위로 생성하여 반환

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



# 파이썬 프로그램 기초-라이브러리

## ▪ tempfile

### 2) tempfile.TemporaryFile

- 임시 저장 공간으로 사용할 파일 객체 반환
- 기본적으로 바이너리 쓰기 모드(wb)
- f.close()가 호출되면 파일 객체는 자동으로 사라짐

```
>>> import tempfile
>>> f = tempfile.TemporaryFile()
>>> f.close() ← 생성한 임시 파일이 자동으로 삭제됨
```

# 파이썬 프로그램 기초-라이브러리

## ▪ time

1. 프로그램 시간 측정용도
2. 시간 출력 포맷

### 1) time.time

- UTC(Universal Time Coordinated, 협정 세계 표준시)를 사용하여 현재 시간을 실수 형태로 반환
  - 1970년 1월 1일 0시 0분 0초를 기준으로 지난 시간을 초 단위로 반환

```
>>> import time
>>> time.time()
988458015.73417199
```

### 2) time.localtime

- time.time( )이 돌려준 실수 값을 사용하여 연도, 월, 일, 시, 분, 초, ...의 형태로 반환

```
>>> time.localtime(time.time())
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)
```

# 파이썬 프로그램 기초-라이브러리

## ▪ time

### 3) time.asctime

- time.localtime에서 반환된 튜플 값을 인수로 받아서 날짜와 시간을 알아보기 쉬운 형태로 반환

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

### 4) time.ctime

- 현재 시간 반환

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

# 파이썬 프로그램 기초-라이브러리

## ▪ time

### 5) time.strftime

- 시간에 관계된 것을 세밀하게 표현하는 여러 가지 포맷 코드 제공

```
time.strftime('출력할 형식 포맷 코드', time.localtime(time.time()))
```

- 예)

```
>>> 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'
```

교재 254 페이지: 포맷 참고

# 파이썬 프로그램 기초-라이브러리

## ▪ time

### 6) time.sleep

- 주로 루프 안에서 많이 사용
- 일정한 시간 간격을 두고 루프를 실행할 수 있음
- 인수는 실수 형태 (예) 1이면 1초, 0.5면 0.5초
- 예) 1초 간격으로 0부터 9까지의 숫자 출력

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

디버깅....

# 파이썬 프로그램 기초-라이브러리

## ■ calendar

### 1) **calendar.calendar(연도)**

- 그해의 전체 달력 출력

```
>>> import calendar
>>> print(calendar.calendar(2015))
```

### 2) **calendar.prcal(연도)**

- calendar.calendar와 같은 결과

```
>>> calendar.prcal(2015)
```

### ■ 예) 2015년 12월의 달력 출력

```
>>> 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			

# 파이썬 프로그램 기초-라이브러리

## ▪ calendar

### 3) calendar.weekday(연도, 월, 일)

- weekday(연도, 월, 일) 함수는 그 날짜에 해당하는 요일 정보 반환
- 월 ~ 일은 0 ~ 6이라는 값으로 반환

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

### 4) calendar.monthrange(연도, 월)

- 입력받은 달의 1일이 무슨 요일인지와 그 달이 며칠까지 있는지 튜플 형태로 반환

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

## ▪ random

- 난수를 발생시키는 모듈

### 1) random.random

- 예) 0.0에서 1.0 사이의 실수 중 난수 생성

```
>>> import random
>>> random.random()
0.53840103305098674
```

### 2) random.randint

- 예) 1에서 10 사이의 정수 중 난수 생성

```
>>> random.randint(1, 10)
6
```

### 3) random.shuffle

- 리스트 항목 무작위로 섞는 함수

```
>>> import random
>>> data = [1, 2, 3, 4, 5]
>>> random.shuffle(data)
>>> data
[5, 1, 3, 4, 2]
>>>
```



# 파이썬 프로그램 기초-라이브러리

## ▪ webbrowser

- 자신의 시스템에서 사용하는 기본 웹 브라우저를 자동으로 실행하는 모듈
- 예) 웹 브라우저를 자동으로 실행하고 해당 URL인 google.com으로 이동

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

- open 함수는 웹 브라우저가 이미 실행된 상태라면 입력 주소로 이동하고, 실행되지 않은 상태라면 새로 웹 브라우저를 실행한 후 해당 주소로 이동
- open\_new 함수는 이미 웹 브라우저가 실행된 상태이더라도 새로운 창으로 해당 주소가 열림

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

## Practice #3 (라이브러리)

```
import pickle
import random

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

f = open("lib_example.txt", "rb")
data = pickle.load(f)
print(data)

print(random.random())
print(random.randint(1, 10))
data_rnd = [1, 2, 3, 4, 5]
random.shuffle(data_rnd)
print(data_rnd)
```

# Report

**실습하기(Report):** json 파일들을 읽어들이고 후,  
데이터 가공 후, 새로운 json 파일 생성

1. glob을 이용하여 폴더 내에 모든 json파일 Read

```
import glob
import os
```

2. open, json.load()를 통해 json 파일 load

```
f = open("CAM_FRONT/000000.json")
json_data = json.load(f)
```

3. class가 level이 0이 아니고, Dontcare인  
데이터는 삭제

4. Truck, Car는 Vehicle로 통일

5. 가공 데이터는 open, json.dump를 이용하여  
**modified 원본파일명으로 파일 저장**

```
json.dump(data_img, f, indent=2)
f.close()
```

indent: json 포맷 정렬 시 사용

```
os.path.basename(filename) - 파일명만 추출
os.path.dirname(filename) - 디렉토리 경로 추출
```

```
{
  "Image": {
    "size": "1920x1208",
    "scene": "Highway",
    "weather": "Overcast",
    "timeofday": "Daytime",
    "nation": "Vietnam"
  },
  "Object": [
    {
      "class": "Truck",
      "box2d": {
        "x1": 1136,
        "y1": 304,
        "x2": 1262,
        "y2": 455
      },
      "level": 0
    },
    {
      "class": "Cyclist",
      "box2d": {
        "x1": 1054,
        "y1": 398,
        "x2": 1075,
        "y2": 433
      },
      "level": 2
    },
    {
      "class": "Cyclist",
      "box2d": {
        "x1": 936,
        "y1": 395,
        "x2": 952,
        "y2": 425
      },
      "level": 0
    },
    {
      "class": "Dontcare",
      "box2d": {
        "x1": 936,
        "y1": 395,
        "x2": 952,
        "y2": 425
      },
      "level": 0
    }
  ]
}
```

Thank you

Q&A

[www.kopo.ac.kr](http://www.kopo.ac.kr)  
[jsshin7@kopo.ac.kr](mailto:jsshin7@kopo.ac.kr)