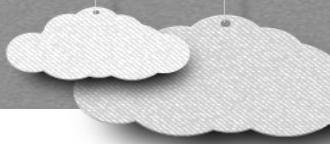




IO, re, Class

목차



chapter-01. 파일입출력	- 3
chapter-02. 정규식(re모듈)	- 34
chapter-03. class 기본	- 78
youtube 재생목록	

- ☞ io : https://www.youtube.com/playlist?list=PLnp1rUgG4UVb_pIQVxfPps6Unnf1dDXKN
- ☞ _____ <https://www.youtube.com/playlist?list=PLnp1rUgG4UVbsulGyRDQV91A7J7CaFCI->
- ☞ re : <https://www.youtube.com/playlist?list=PLnp1rUgG4UVbGVR1QPhR-tGGGne2xWAuB>
- ☞ class : <https://www.youtube.com/playlist?list=PLnp1rUgG4UVZc4ib8q2sRmnI8gKylwh7J>

강사 : 윤소영

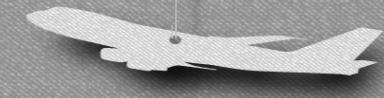
본 과정은 python 문법 학습을 종료한 학습자들을 대상으로 합니다.

본 과정의 모든 내용은 youtube를 통해 보기 하실 수 있습니다.

<https://www.youtube.com/c/EduAtoZPython>



Chapter - 01



파일 입출력



1. 파일, 디렉터리의 이해



- ▶ 프로그램이 종료된 후에도 데이터를 계속 유지하려면 “파일”로 저장해야 한다
- ▶ 파일은 경로와 파일이름으로 구성 됨

C:\MyPythonFiles\TextFiles\test.txt

경로 (Dir name)

파일 이름 (Base name)

\(₩)는 무엇일까요?

Windows에서 directory 와 directory, base name 구분에 사용함

/ 는 무엇일까요?

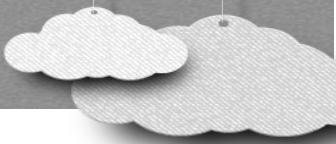
OS X, Linux에서 사용하는 경로 구분자

Python에서는요?

\(₩), / 모두 사용 가능! BUT → ₩는₩₩ 와 같이 사용 함

1. Windows 예 : 'C:\\PythonFiles\\letter.txt' 또는 r'C:\\PythonFiles\\letter.txt'
2. Linux 예 : '/content/drive/MyDrive/MyPythonFiles/letter.txt'

1-01. dirname, basename



- ▶ 프로그램이 종료된 후에도 데이터를 계속 유지하려면 “파일”로 저장해야 한다

dirname

- path에서 basename을 제외한 경로
- os.path.dirname(file)을 사용하여 dirname을 알아낼 수 있음

basename

- path에서 마지막 파일/디렉터리 이름
- os.path.basename(file)을 사용하여 basename을 알아낼 수 있음

- ▶ Windows 환경에서의 dirname, basename 예

```
file = "C:\\MyPythonFiles\\TextFiles\\test.txt"
```

```
os.path.dirname(file)
```

```
'C:\\MyPythonFiles\\TextFiles'
```

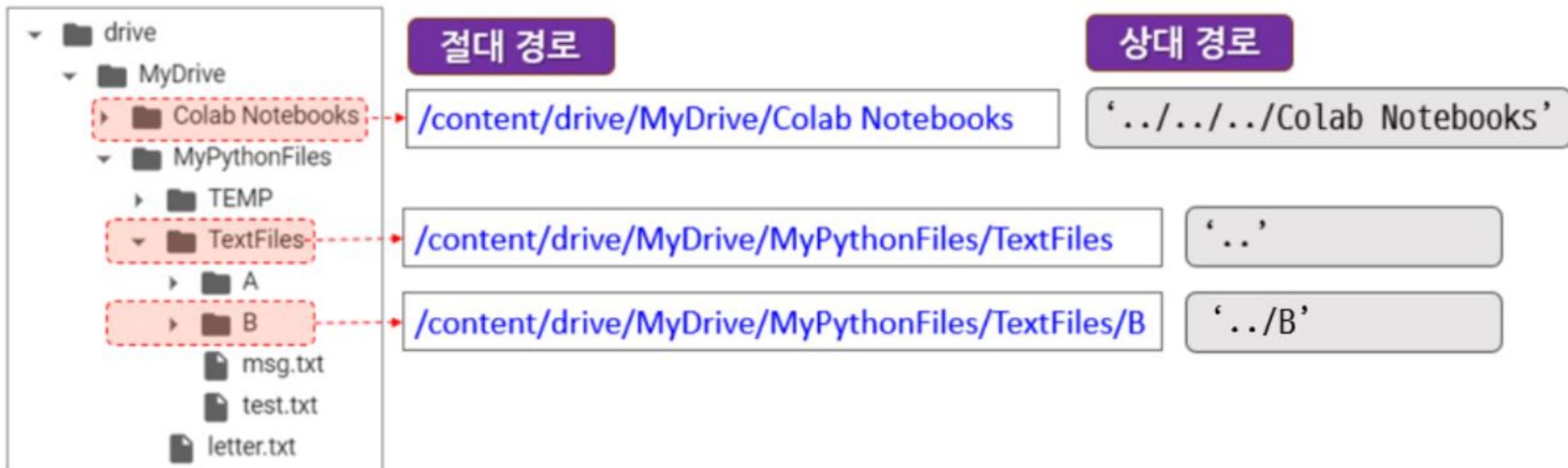
```
os.path.basename(file)
```

```
'text.txt'
```

1-02. 절대 경로, 상대 경로



- File System에서 경로(Path)는 File, Directory 등의 위치 정보를 의미함
- Path를 표기하는 방법에는 ‘절대 경로’와 ‘상대 경로’가 있음
 - 절대 경로 : 리눅스인 경우 '/' 부터, Windows인 경우 'C:/'와 같은 드라이브 이름부터 시작하는 경로
 - 상대 경로 : 현재 디렉터리를 기준으로 상위, 하위 디렉터리를 이동하며 표현되는 경로
 - .. : 상위 디렉터리 이름 : 하위 디렉터리
 - 현재 작업 디렉터리가 C:＼＼＼MyPythonFiles＼＼TextFiles＼＼A’ 인 경우



1-03. 디렉터리 변경/확인 및 파일 속성 관련 함수



▶ 현재 작업 디렉터리(Current Working Directory)란

☞ 사용자 프로그램은 동작 중 특정 디렉터리를 사용하게 되며, 현재 사용하는 디렉터리를 현재 작업 디렉터리라고 함

사용자 프로그램에서 현재 사용하는 디렉터리

`os.getcwd()`로 알 수 있음

디렉터리 변경(Change Directory)

`os.chdir(path)`로 지정함

디렉터리 생성(Make Directory)

`os.mkdir(path)`로 생성함

▶ 파일 속성 관련 함수

`os.path.getsize(path)`

path의 파일 크기를 바이트 단위로 반환

`os.path.getsize(path)`

path 파일이 존재하는지 확인, 존재하면 True

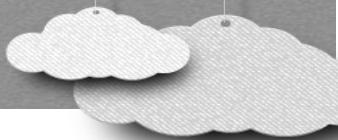
`os.path.getsize(path)`

path가 디렉터리인지 확인

`os.path.getsize(path)`

path가 파일인지 확인, 파일이면 True

1-03. 디렉터리 변경/확인 및 파일 속성 관련 함수



1-03) 실습 코드

```
import os  
① cwd = os.getcwd()  
print(cwd)
```

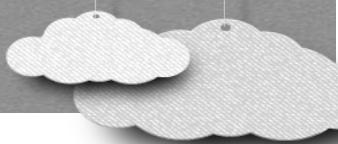
- cwd 에는 현재 작업 디렉터리가 저장되어 있음
- 디렉터리를 변경했다가, 복원할 때 정보로 사용하는 용도로 만듦

```
os.chdir('./MyPythonFiles/TextFiles/A')  
② print(os.getcwd())
```

```
if not os.path.exists('../C') :  
    ③ os.mkdir('../C')  
os.chdir('../C')  
print(os.getcwd())
```

```
os.chdir(cwd)  
os.chdir('./MyPythonFiles')  
print(os.getcwd())  
filename1 = 'letter.txt' # 일반 파일  
filename2 = 'TextFiles' # 디렉터리  
for filename in filename1, filename2 :  
    print(os.path.getsize(filename))  
    print(os.path.exists(filename))  
    print(os.path.isdir(filename))  
    print(os.path.isfile(filename))  
    print('-' * 7)
```

1-04. 경로 및 파일이름 관련 함수 -1



▣ 경로 및 파일 이름 관련 함수

`os.path.abspath(path)`

path에 대한 절대 경로 문자열 반환

`os.path.isabs(path)`

path에 대해 절대 경로인지 확인 절대 경로일 때 True

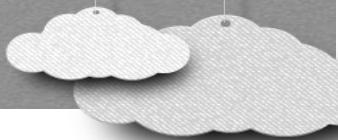
`os.path.relpath(path[, start])`

start 경로로 시작하는 path의 상대 경로 문자열 반환,
start가 제공되지 않으면 현재 작업 디렉터리가 사용됨

`filePath.split(os.path.sep)`

- `os.path.sep`에 맞추어 filePath를 분리함
- filePath는 경로 + 파일이름이며, '문자열'로 작성함

1-04. 경로 및 파일이름 관련 함수 -2

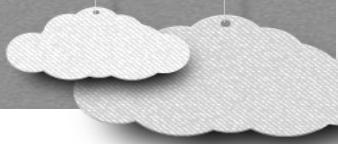


1-04) 실습 코드

```
# 영상에서의 file 지정과 방법만 다름  
# 사용환경에 따라 변경되도록 하기 위함  
# file='/content/drive/MyDrive/MyPythonFiles/TextFiles/A'  
import os  
file = cwd + '/MyPythonFiles/TextFiles/A'  
os.chdir(file)  
print(f'current working directory \n{os.getcwd()}\n')  
print(os.path.isabs('.'))  
print(os.path.isabs(file))  
print(os.path.abspath('.'))  
print(os.path.abspath('../B'))
```

```
# 영상에서는 cwd를 사용하지 않고 직접 /content/drive/MyDrive 를 사용  
# 코드를 수정하여 환경에 적응하는 형태로 file을 작성함  
print(f'current working directory \n{os.getcwd()}\n')  
file = cwd + '/MyPythonFiles/TextFiles/B'  
print(os.path.relpath(file))  
print(os.path.relpath(cwd + '/MyPythonFiles/TextFiles',  
                     cwd + '/MyPythonFiles/TextFiles/A'))
```

1-04. 경로 및 파일이름 관련 함수 -3

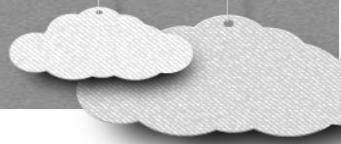


1-04) 실습 코드

```
# windows에서의 경로
file1 = 'C:\\MyPythonFiles\\TextFiles' # os.path.sep로 분리 안됨, colab = Linux
file2 = 'C:/MyPythonFiles/TextFiles'    # 잘못된 것!!
# Linux에서의 경로
file3 = '/content/drive/MyDrive'

print(f'os.path.sep = {os.path.sep}') # /
for file in file1, file2, file3 :
    print(os.path.sep, file.split(os.path.sep))
    print('-' * 50)
```

2. 파일 입출력



▣ 텍스트 파일 vs 이진 파일

텍스트 파일

- txt, py, c, h, cpp, sh 등의 확장자인 일반 텍스트 파일
- 텍스트 편집기(메모장)로 열어 내용을 볼 수 있음

이진 파일

- 워드, 엑셀, 파워포인트 문서, PDF, 이미지, 실행 파일 등
- 이진 파일은 형식마다 고유한 방식으로 처리됨
→ 일반 메모장 같은 편집기로 확인 할 수 없음

▣ 파이썬에서의 파일 읽기/쓰기의 3 단계는 다음과 같다

1

open() 함수를 호출하여 File 객체 얻기

2

File 객체의 read(), write() 메서드로 읽기 쓰기

3

File 객체의 close() 메서드로 파일 닫기

2-01. 파일 읽기/쓰기의 3단계



파일 읽기/쓰기의 3단계

1. open() 함수를 호출하여 File 객체 얻기

- File 객체 저장 이름 = `open(path, mode)`
- mode : 읽기(r), 쓰기(w), 추가(a)
- 'w' : 파일이 존재하면 파일의 모든 내용을 삭제하고 다시 쓰기 준비
- 'a' : 파일이 존재하면 내용 삭제 없이, 파일의 뒤에 내용을 추가함
- 'b' : 'r', 'w' 등과 같이 사용하며, 이진 파일에 사용함

2. File 객체의 `read()`/`readlines()`, `write()` 메서드로 읽기 쓰기

- `File 객체.read()` : 파일 전체를 하나의 문자열로 읽기
- `File 객체.readlines()` : 한 줄(`\n`)을 하나의 문자열로 읽어 리스트로 반환
- `File 객체.write(str)` : 문자열을 파일에 쓰기
- 자동줄바꿈 추가 없음 필요시 `\n`을 사용함

3. File 객체의 `close()` 메서드로 파일 닫기

- `File 객체.close()`

2-01. 파일 읽기/쓰기 실습 - 1



2-01 실습) 파일을 open하고 내용을 읽는 방법을 살펴보자

- 파일의 절대/상대 경로를 문자열로 전달하여 open() 함수를 호출한다

```
File 객체 저장 변수 = open('파일명', mode) # mode 생략 시 읽기 전용 'r'
```

- 파일로부터 읽기 작업은 File 객체의 read() 또는 readlines() 메서드를 사용한다

```
읽은 내용 저장 변수 = File객체.read() # File객체.readlines()
```

```
file = cwd + '/MyPythonFiles/letter.txt'  
myfile = open(file)  
mycont = myfile.read()  
print(type(mycont), mycont, sep='\n')  
myfile.close()
```

```
<class 'str'>  
Dear friend,  
How are you?
```

하나의 문자열로 읽기

```
file = cwd + '/MyPythonFiles/letter.txt'  
myfile = open(file)  
mycont = myfile.readlines()  
print(type(mycont), mycont, sep='\n')  
myfile.close()
```

```
<class 'list'>  
['Dear friend,\n', 'How are you?\n']
```

줄바꿈

하나의 \n을 하나의 문자열로 하는 list로 읽기

2-01. 파일 읽기/쓰기 실습 -3



2-01 실습) 파일을 쓰기 모드로 open하고 내용을 쓰는 방법을 살펴보자

1. letter2.txt 파일을 쓰기 모드를 이용해서 열기한다

File 객체 저장 변수 = open('파일명', mode) # 쓰기 모드 'w', 추가 모드 'a'

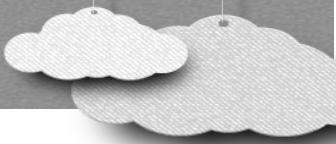
2. 파일에 쓰기 작업은 File 객체의 write() 메서드를 사용한다

File 객체 저장 변수.write('내용')

```
file = cwd + '/MyPythonFiles/letter2.txt'
myfile = open(file, 'w')
myfile.write('Take care! See u later\n') → print(): 행 끝에 자동 줄바꿈
myfile.close()                                write(): 자동 줄바꿈 추가 없음
```

```
file = cwd + '/MyPythonFiles/letter2.txt'
myfile = open(file, 'a')
myfile.write('Nice to meet you!\n')
myfile.close()
```

2-02. with statement -1



파일과 소켓 같이 '리소스'에 접근하는 경우 핸들(handle) 처리가 중요하다

リ소스를 제어를 위한 핸들을 얻고, 반환하는 작업이 필요하다

- 핸들 반환 전에 특정 조건에 의해 예외가 발생하여 리소스 정리를 못하고 종료 될 수 있다

with문은 파일, 소켓 같이 열고 닫는 자원 접근 시 사용 된다

- with 문을 통해 리소스 접근을 특정 블록 내의 동작을 제한한다
- with문 블록이 끝날 때 리소스의 자동으로 해제 처리 하는 것을 보장한다

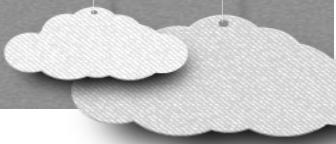
with문의 용법

- with문을 콤마를 이용해 나열해 사용하거나 중첩하여 사용할 수 있다

with expression1 [as target1[, expression2 as target2]]:
실행할 문장들

with expression1 [as target1] :
[with expression2 [as target2] :]
실행할 문장들

2-02. with statement -2



▶ 2-02 실습) 파일을 사용할 때의 오류 처리의 예를 살펴보고 분석하라

▶ try ~ except 와 with 문 함께 사용

```
try :  
    with open(file) as fp:  
        msg = fp.read()  
        print(msg)  
except OSError:  
    print("File doesn't exist or No read permission")
```

▶ 파일이 존재하고, 읽기 권한이 있는지 확인 후 사용

```
import os  
if os.access(file, os.F_OK) and os.access(file, os.R_OK):  
    with open(file) as fp:  
        msg = fp.read()  
        print(msg)  
else:  
    print("File doesn't exist or No read permission")
```

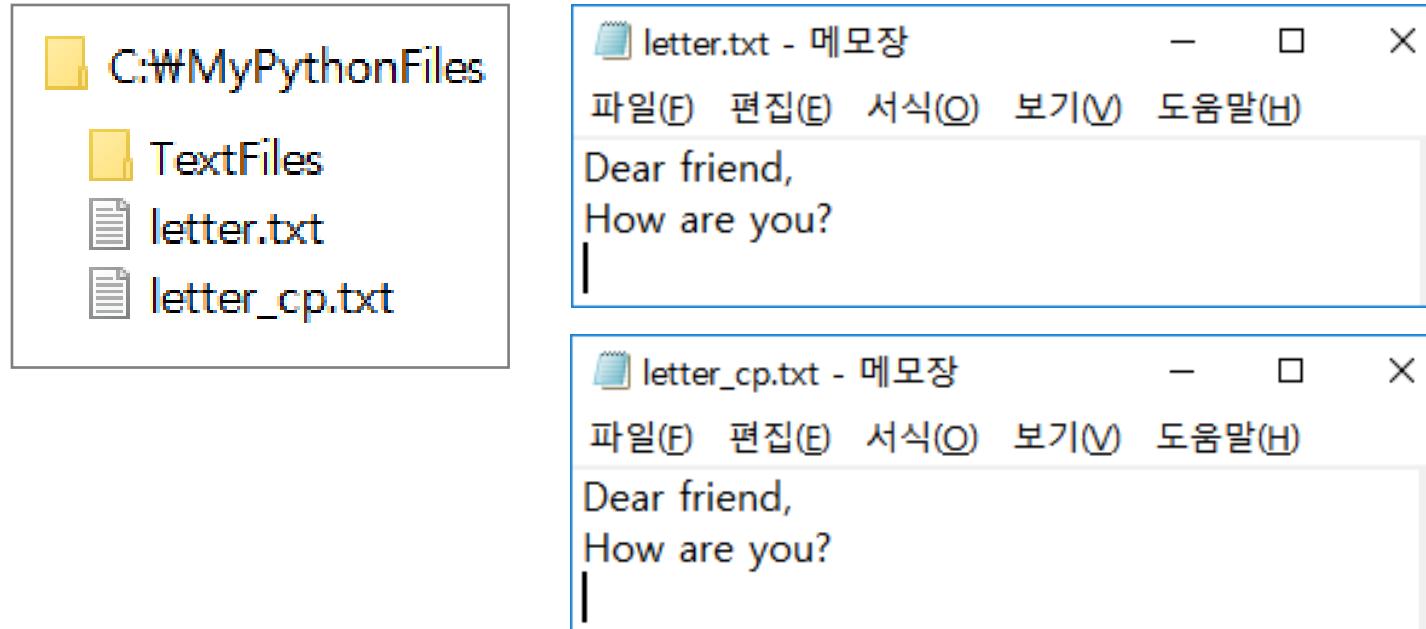
2-03. 파일 복사 프로그램 작성



▶ 2-03 실습) open, read, write를 사용해 파일을 복사하는 프로그램을 작성하라

☞ fileR을 읽어 fileW에 쓰기 작업하는 동작을 통해 파일을 복사하는 프로그램을 작성한다

- fileR이 존재하고 읽기 권한이 있는지 확인한다
- 존재하고 읽기 권한이 있는 경우 fileR의 내용을 읽어 fileW에 쓰기 동작하고, 복사 성공 메시지를 출력한다
- 존재하지 않거나, 읽기 권한이 없다면, 오류 메시지를 출력한다



2-04. 행 조작 프로그램 작성



▶ 2-04 실습) 파일을 열어 다음과 같이 조작하여 저장하는 프로그램을 작성하라

☞ [./MyPythonFiles/mydata.txt](#) 파일을 사용한다

☞ 파일 조작 내용은 다음과 같다

- 1~3번째 행을 삭제한다
- 파일의 각 행을 길이 별로 정렬한다 (길이에 행 번호는 포함하지 않음)
- 마지막 행에 “Good luck to you!₩n”를 추가한다
- 새롭게 1번부터 시작하는 행 번호를 붙여 mydataT.txt로 저장한다

```
1 It doesn't hurt to try.  
2 Rather be dead then cool.      ←----- mydataT.txt  
3 A friend is a second self.  
4 Make hay while the sun shines.  
5 Life isn't always what one like.  
6 He who laughs alst, laughs best.  
7 Every failure is a stepping stone to success.  
8 Learning to love yourself is the greatest love of all.  
9 Good luck to you!
```

2-05. 열 조작 프로그램 작성



▶ 2-05 실습) 파일을 열어 다음과 같이 조작하여 저장하는 프로그램을 작성하라

☞ [./MyPythonFiles/mydata2.txt](#) 파일을 사용한다

☞ 파일 조작 내용은 다음과 같다

- 1번, 6번 열을 교환하고, 3번, 4번 열을 교환한다
- 마지막 열을 삭제한다
- 교환을 위한 함수와 삭제를 위한 함수를 각각 작성하여 사용한다
- mydata2T.txt로 저장한다

mydata2.txt

```
1 283 748 328 57 454 7 148 289
2 240 996 7 578 826 254 65 272
3 754 674 781 925 681 656 963 445
4 700 304 598 584 469 40 808 954
5 94 27 374 587 373 912 218 447
6 216 487 800 320 434 324 449 756
7 889 742 259 636 71 194 598 391
8 308 623 9 993 426 567 243 629
9 131 668 137 654 321 758 301 742
10 51 331 211 408 817 529 513 161
```

mydata2T.txt

```
1 7 748 57 328 454 283 148
2 254 996 578 7 826 240 65
3 656 674 925 781 681 754 963
4 40 304 584 598 469 700 808
5 912 27 587 374 373 94 218
6 324 487 320 800 434 216 449
7 194 742 636 259 71 889 598
8 567 623 993 9 426 308 243
9 758 668 654 137 321 131 301
10 529 331 408 211 817 51 513
```

3. 파일 관련 다양한 모듈



↳ 파일 복사 관련

↳ [shutil](#)

↳ [distutils.dir_util](#)

↳ binary 파일 관련

↳ [pickle](#)

↳ [shelve](#)

↳ 파일 목록 가져오기

↳ [glob](#)

3-01. 파일, 디렉터리 복사



3-01 실습) 모듈을 사용하여 파일, 디렉터리를 복사하는 방법을 알아보자

```
import shutil  
shutil.copy('C:\\MyPythonFiles\\letter.txt',  
            '.\\letter_cp2.txt')  ----- 파일이 이미 존재하면 덮어쓰기 됨  
  
from distutils.dir_util import copy_tree  
copy_tree('C:\\MyPythonFiles\\TextFiles', '.\\TextFiles')
```

파일 복사 - shutil.copy

```
shutil.copy(‘원본 파일’, ‘복사본 파일’)
```

디렉터리 복사 - distutils.dir_util.copy_tree

```
distutils.dir_util.copy_tree(‘원본 디렉터리’, ‘복사본 디렉터리’)
```

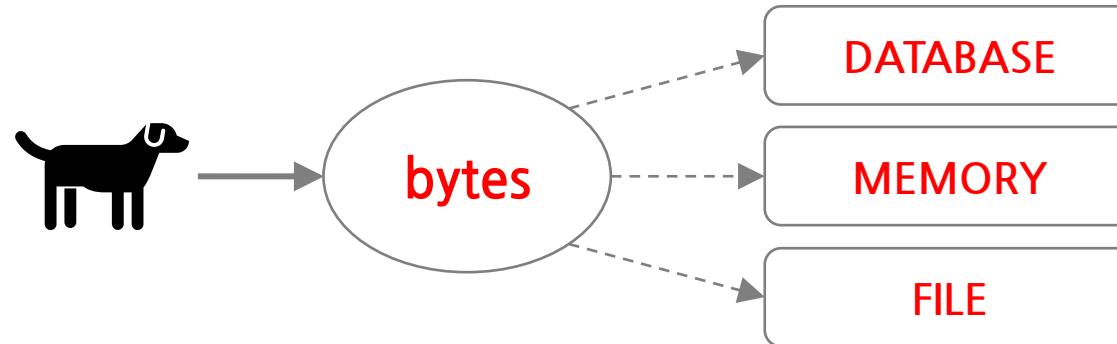
- ‘원본 디렉터리’ 내부의 모든 파일과 디렉터리를 모두 ‘복사본 디렉터리’에 복사한다
- ‘복사본 디렉터리’가 없으면 자동 생성된다

3-02. binary 파일 저장 pickle 모듈



3-02 실습) pickle 모듈

- 객체 serialization/deserialization을 수행한다
- 객체의 상태를 메모리나 영구 저장 장치에 저장이 가능한 정보로 바꾸는 것



- 다음을 실행하여 pickle로 쓰기와 읽기 수행을 알아보자

```
import pickle

D = {'A' : 65, 'B' : 66 }
with open('data.pkl', 'wb') as F:
    pickle.dump(D, F)
    pickle.dump([1, 2, 3, 4, 5], F)
```

```
with open('data.pkl', 'rb') as F:
    X = pickle.load(F)
    Y = pickle.load(F)

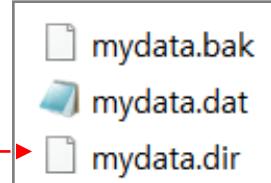
print(X, Y)
```

3-03. binary 파일 저장 - shelve 모듈 -1



3-03 실습1) shelve 모듈을 이용하여 이진 파일에 객체를 저장하고, 읽을 수 있다

```
import shelve  
  
mydata = shelve.open('mydata')  
fruits = ['orange', 'banana', 'apple']  
① mydata['fruits'] = fruits  
② print(mydata['fruits'])  
mydata.close()
```



3개의 파일이 생성됨
OS X → mydata.db

- key 를 이용해 객체를 저장하고 불러옴
 - key는 문자열을 사용함
- mode 부여 필요 없음
 - 읽고, 쓰기 모두 가능하게 열림
- 한 번 쓰기 한 내용은 계속 유지됨
- 같은 'key'로 여러 번 저장 시 최근 것으로 저장됨 (갱신)
- Windows에서는 X.bak, X.dat, X.dir 세 개의 파일이 생성되고, Linux에서는 X.db 파일이 생성됨

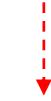
3-03. binary 파일 저장 - shelve 모듈 -2

3-03 실습1) shelve 파일에 다양한 객체를 저장하고 가져오기를 수행한다

```
import shelve
fruits = ['orange', 'apple']
myStr = 'hello!'
myValue = 100.2

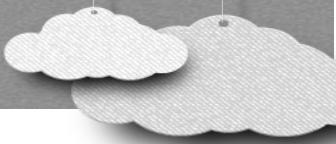
with shelve.open('myshelf') as mydata:
    mydata['fruits'] = fruits
    mydata['myStr'] = myStr
    mydata['myValue'] = myValue
    mydata['myValue'] = 200.54
    print(list(mydata.keys()))
    print(list(mydata.values()))
    print(mydata['fruits'])
    print(mydata['myStr'])
    print(mydata['myValue'])
```

```
['fruits', 'myStr', 'myValue']
[['orange', 'apple'], 'hello!', 200.54]
['orange', 'apple']
hello!
200.54
```



처음엔 100.2 저장, 200.54로 갱신됨

3-04. glob 모듈 -1



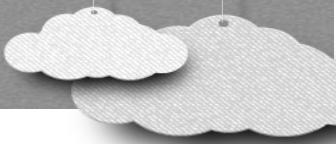
▶ glob 모듈은 윈도우의 dir, 리눅스의 ls 명령과 유사한 기능을 제공한다

▶ glob 모듈의 주요 메서드는 다음과 같다

메서드	기능
glob.glob(path)	<ul style="list-style-type: none">▪ path에 대응되는 모든 파일 및 디렉터리 리스트 반환▪ 와일드 카드 문자('*', '?') 및 [...] 사용 가능
glob.iglob(path)	<ul style="list-style-type: none">▪ glob()와 같은 기능을 수행하지만 결과를 generator로 반환▪ 한 번에 결과를 리스트에 담지 않음▪ 결과가 매우 많은 경우 유용하게 사용될 수 있음

glob 문자	기능	예시
?	1글자 문자를 대신함	a???.txt
*	0글자 이상의 여러 글자 문자를 대신함	a*.xlsx
[...]	[] 내부에 포함된 글자 중 1개 문자	a[bcd].csv

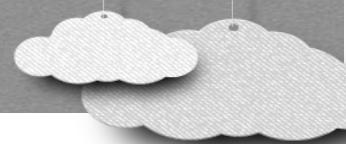
3-04. glob 모듈 -2



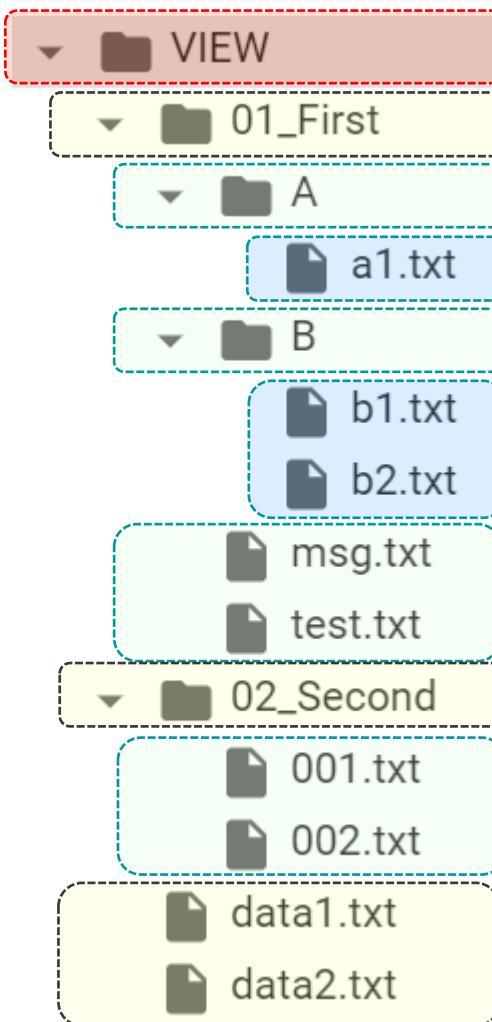
3-04 실습) 다음 glob 모듈의 사용을 살펴보자

```
import os, glob
file = cwd + '/MyPythonFiles'
os.chdir(file)
print(os.getcwd())
# 현재 디렉터리에서 모든 파일의 목록 가져오기
print(glob.glob('*'))
# 현재 디렉터리에서 모든 .txt로 끝나는 파일의 목록 가져오기
print(glob.glob('*.txt'))
# 하위 디렉터리 TEMP에서 이름이 두 글자이고 .txt로 끝나는 파일의 목록 가져오기
# 단, 이름의 첫 글자는 숫자이고 두 번째는 아무거나 한 글자이다
print(glob.glob('TEMP/[0-9]??.txt'))
# 하위 디렉터리 TEMP에서 이름이 한글자 이상이고 .txt로 끝나는 파일의 목록 가져오기
# 단, 이름의 첫 글자는 숫자이고 두 번째 글자부터는 아무거나 여러 글자로 0글자 이상이다.
print(glob.glob('TEMP/[0-9]*.txt'))
```

3-05. glob 모듈 - 하위 디렉터리까지 모두 검색



▣ 다음의 파일, 디렉터리 구조를 분석하여 보자



→ tree('~/VIEW', 0)
→ tree('~/VIEW/01_First', 1)
→ tree('~/VIEW/01_First/A', 2)
→ 파일 출력 depth 3
→ tree('~/VIEW/01_First/B', 2)
→ 파일 출력 depth 3
→ 파일 출력 depth 2
→ tree('~/VIEW/02_Second', 1)
→ 파일 출력 depth 2
→ 파일 출력 depth 1

```
|-- 01_First
|---/-- A
|---/---/-- a1.txt
|---/--- B
|---/---/-- b1.txt
|---/---/-- b2.txt
|---/-- msg.txt
|---/-- test.txt
|-- 02_Second
|---/-- 001.txt
|---/-- 002.txt
|-- data1.txt
|-- data2.txt
total 4 directories 9 files
```

3-05. glob 모듈 - 하위 디렉터리까지 모두 검색



3-05 실습) 하위 디렉터리 검색

```
dircnt = filecnt = 0
def tree(path, depth):
    global dircnt, filecnt
    for x in glob.glob(path + '/*'):
        prefix = '|--' if depth == 0 else '|' + '--' * depth + '|--'
        if os.path.isdir(x):
            dircnt += 1
            print(prefix + os.path.basename(x))
            tree(x, depth + 1)
        elif os.path.isfile(x):
            filecnt += 1
            print(prefix + os.path.basename(x))
        else:
            print(prefix + 'unknown : ', x)

os.chdir(mypath)
tree('.', 0)
print('total ', dircnt, 'directories', filecnt, 'files')
```

3-06. os 모듈의 유용한 함수 -1



os 모듈은 운영체제(OS)와 직접적으로 연결해주는 모듈이다

import os 를 하면 사용할 수 있음, <https://docs.python.org/ko/3.7/library/os.html> 참조

함수	기능
os.listdir([path])	<ul style="list-style-type: none">path 생략시 현재 작업 디렉터리의 파일 목록을 list로 반환path 지정시 해당하는 디렉터리의 파일 목록을 list로 반환
os.system(cmd)	<ul style="list-style-type: none">문자열 형식의 cmd를 os가 실행함
os.access(path, mode)	<ul style="list-style-type: none">path에 대해 mode 작업 가능 여부를 검사함가능할 때 True, mode는 ' '로 연결 가능os.F_OK(파일 존재), os.R_OK, os.W_OK, os.X_OK (실행)
os.makedirs(path [, mode=0o777, exist_ok=False])	<ul style="list-style-type: none">디렉터리를 재귀적으로 생성exist_ok가 False면 존재하는 경우 다시 생성하지 않음
os.remove(path), os.unlink(path)	<ul style="list-style-type: none">파일을 삭제함, 디렉터리면 OSError 발생
os.removedirs(path)	<ul style="list-style-type: none">디렉터리를 재귀적으로 삭제함path가 존재하지 않으면 FileNotFoundError 발생
os.rename(src, dst)	<ul style="list-style-type: none">src를 dst로 이름을 변경하거나 이동함, 파일, 디렉터리에 모두 적용 됨src가 없으면 FileNotFoundError 발생, dst가 존재하면 OSError 발생

3-06. os 모듈의 유용한 함수 -2

3-06 실습) 다음 os 모듈 함수를 이용한 동작을 분석하라

```
import os
file = cwd + '/MyPythonFiles'
#os.chdir(file)
print(os.getcwd())
print(os.listdir(file))
os.system('mkdir MYDIR')
print(os.listdir())
```

```
import os
os.chdir(cwd + '/MyPythonFiles')
print('1:', os.listdir())
os.makedirs('./dirs/a/b', exist_ok=True)
print('2:', os.listdir())
print('3:', os.listdir('./dirs'), os.listdir('./dirs/a'))
if os.access('MYDIR', os.F_OK) and not os.access('MYTEMP', os.F_OK):
    os.rename('MYDIR', 'MYTEMP')
else :
    print("MYDIR doesn't exist or MYTEMP exist")

os.removedirs('./dirs/a/b')
print('4:', os.listdir())
```

3-07. sys 모듈의 유용한 속성 -1

sys 모듈은 파이썬 인터프리터 관련 정보와 기능을 제공한다

sys 모듈의 주요 속성은 다음과 같다

속성	정보
sys.argv	파이썬 스크립트로 넘어온 argument 리스트
sys.modules	현재 로딩 되어 있는 모듈들을 사전 형태로 저장
sys.path	모듈을 찾을 때 참조하는 경로 리스트
sys.copyright, sys.version	설치된 파이썬의 저작권, 버전 정보
sys.stdin, sys.stdout, sys.stderr	표준 입력, 출력, 에러 스트림에 대응되는 파일 객체 <code>sys.stdin = open('a.txt', 'r')</code> : 표준 입력을 파일로 변경
sys.prefix, sys.exec_prefix	파이썬이 설치된 경로 정보
sys.executable	파이썬 인터프리터의 실행파일 경로 정보

3-07. sys 모듈의 유용한 속성 -2



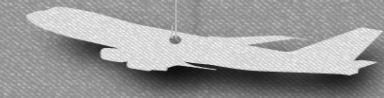
3-07 실습) 다음 sys 모듈 속성을 이용한 동작을 분석하라

```
import sys, os

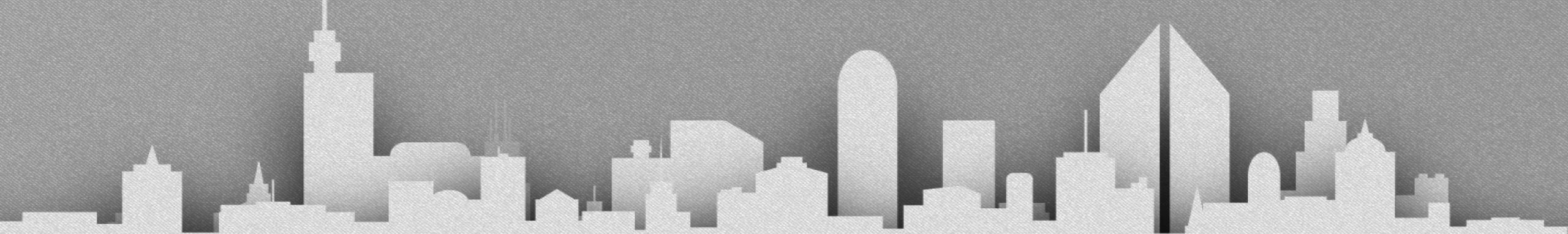
os.chdir(cwd + '/MyPythonFiles')
backup = sys.stdout
sys.stdout = open('data.dat', 'w')
print(sys.argv)
print(sys.modules)
print(sys.path)
print(sys.copyright, sys.version)
print(sys.prefix, sys.executable)
sys.stdout = backup
sys.stdin = open('in.dat', 'r')
data = input("input message from 'data.dat'")
print("\n" + data)
```



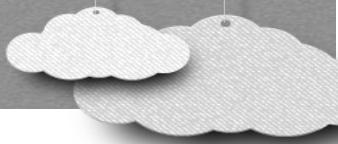
Chapter - 02



정규식(re 모듈)



정규식(Regular Expression)



▶ 정규식 필요성

- ◀ 문자열에서 특정 패턴(규칙)을 갖는 부분이 있는지 확인/추출
- ◀ 기호를 사용하여 간단하고 명료하게 패턴을 표기(높은 가독성)
- ◀ 다양한 확인/추출을 위한 함수와 세부 설정을 위한 옵션 존재
- ◀ import re 사용

▶ 참고 사이트

- ◀ <https://www.regular-expressions.info/tutorial.html>

1-01. 암호 유효성 검사



▶ 암호의 유효성을 검사하는 `isPassword1` 함수를 검토하자

▶ 암호의 유효성 검사에 사용되는 규칙은 다음과 같다

- 총 길이는 10글자 이상이어야 한다
- 1개 이상의 숫자, 영문 대문자, 영문 소문자가 포함되어 있어야 한다
- `!@#$%^&*`() 문자 중 1개 이상의 특수 문자를 포함해야 한다
- `verify()` 함수는 `isPassword`의 결과에 따른 출력을 하는 함수로 제공된다

▶ 입출력의 예시는 다음과 같다

```
print(check('asbdJKL12**'))  
print(check('123jkKL*'))  
print(check('ajkdk123kjKL'))  
print(check('ABC123*^5D'))
```

```
True  
False  
False  
False  
>>>
```

1-01. 암호 유효성 검사



▶ 1-01실습1) 풀이를 분석하고 간략하게 풀이할 수 있는 방법을 찾아보자

```
# 특정 특수 문자 인지 확인하는 함수
def isspecial(ch):
    return ch in '!@#$%^&*()'

# 암호 유효성 검사 함수
def isPassword_1(pw):
    if len(pw) < 10 : return False
    check = [False] * 4
    for ch in pw:
        if ch.islower() : check[0] = True
        if ch.isupper() : check[1] = True
        if ch.isdecimal(): check[2] = True
        if isspecial(ch) : check[3] = True
    if all(check) : return True
    return False
```

1-01. 암호 유효성 검사 - 정규식



▶ 1-01실습2) 정규식을 이용한 암호 유효성 검사는 다음과 같다

- ◀ 앞의 풀이보다 직관적으로 분석할 수 있다
- ◀ 아래의 풀이는 정규식 학습 후에 한 번 더 분석해 보도록 한다

```
import re

# 암호 유효성 검사 함수
def isPassword_2(pw):
    if len(pw) < 10 : return False
    rules = ( r'[0-9]+', r'[a-z]+', r'[A-Z]+', r'[^!@#$%^&*]+')
    for rule in rules:
        if not re.search(rule, pw) : return False
    return True
```

1-02. 휴대폰 번호 유효성 검사



▶ 휴대폰 번호 여부를 검사하는 `isPhoneNumber_T1` 함수를 검토하라

▶ 휴대폰 번호에 사용되는 규칙은 다음과 같다

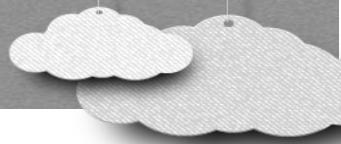
- 3개의 그룹의 숫자로 되어 있으며 그룹은 ‘-’로 구분된다
- 첫 번째 그룹은 3글자로 되어 있으며 다음 중 하나이다
010, 011, 012, 016, 017, 018, 019
- 두 번째 그룹은 3글자 또는 4글자로 되어 있다
- 세 번째 그룹은 4글자로 되어 있다
- `verify()`함수는 `isPhoneNumber`의 결과에 따른 출력을 하는 함수로 제공됨

▶ `check()`함수는 `isPhoneNumber`의 결과에 따른 출력을 하는 함수로 제공된다

```
check('010-456-7892')
check('010-2345-1234')
check('013-123-1231')
check('012 1234 1234')
check('016-12-1234')
check('TEL: 012-123-1234')
check('017-123-1234!')
```

```
010-456-7892 Correct number
010-2345-1234 Correct number
013-123-1231 Wrong number
012 1234 1234 Wrong number
016-12-1234 Wrong number
TEL: 012-123-1234 Wrong number
017-123-1234! Wrong number
```

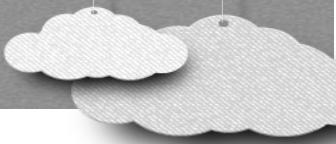
1-02. 휴대폰 번호 유효성 검사



▶ 1-02 실습1) 다음 풀이를 분석하고 간략하게 풀이할 수 있는 방법을 찾아보자

```
pno = [1, 1, 1, 0, 0, 0, 1, 1, 1, 1]
def isPhoneNumber_T1(num):
    if len(num)<12 or len(num)>13 : return False
    grp = num.split('-')
    if (len(grp) != 3) : return False
    if len(grp[0]) != 3 or grp[0].isdecimal() == False : return False
    if grp[0][0:2] != '01' : return False
    if not pno[int(grp[0][2])] : return False
    if len(grp[1]) < 3 or len(grp[1])>4 or grp[1].isdecimal() == False : return False
    if len(grp[2]) != 4 or not grp[2].isdecimal() : return False
    return True
```

1-02. 휴대폰 번호 유효성 검사 - 정규식



▶ 1-02 실습2) 정규식을 이용한 핸드폰 번호 유효성 검사는 다음과 같다

- ◀ 매우 직관적이며 짧은 표현으로 변경되었다
- ◀ 아래의 풀이는 정규식 학습 후에 한 번 더 분석해 보도록 한다

```
import re

def isPhoneNumber_T2(num):
    if re.search(r'^01[0-26-9]-[0-9]{3,4}-[0-9]{4}$', num) :
        return True
    return False
```

1-03. 문자열에서 휴대폰 번호 검색



▶ 문자열에서 휴대폰 번호를 검색 및 출력하는 코드를 검토하라

▶ 문자열은 문자와 숫자가 함께 사용되며, 여러 개의 휴대폰 번호를 포함 할 수 있다

- msg = 'julie 010-123-1234 peter 010-1234-1234 office 030-123-1234'

▶ 휴대폰 번호에 사용되는 규칙은 다음과 같다

- 3개의 그룹의 숫자로 되어 있으며 그룹은 ‘-’로 구분된다
- 첫 번째 그룹은 3글자로 되어 있으며 다음 중 하나이다
010, 011, 012, 016, 017, 018, 019
- 두 번째 그룹은 3글자 또는 4글자로 되어 있다
- 세 번째 그룹은 4글자로 되어 있다

▶ 출력은 다음과 같이 처리한다

```
Phone number : 010-123-1234
Phone number : 011-1234-1234
>>>
```

1-03. 문자열에서 휴대폰 번호 검색



▶ 1-03 실습1) 다음 풀이를 분석하고 간략하게 풀이할 수 있는 방법을 찾아보자

☞ [isPhoneNumber\(문자열\) 함수](#)

- 휴대폰 번호인 경우 True 아닌 경우 False를 반환하도록 설계된 함수 (1-02에서 작성된 함수)

```
idx = 0
for i in range(len(msg)-12):
    phone12 = msg[i:i+12]
    phone13 = msg[i:i+13]
    phone = None
    if isPhoneNumber(phone12) : phone = phone12
    elif isPhoneNumber(phone13) : phone = phone13
    if phone :
        idx += 1
        print(f'Phone number{idx} : {phone}')
```

1-03. 문자열에서 휴대폰 번호 검색 - 정규식

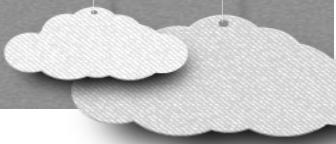


- 1-03 실습2) 정규식을 이용한 문자열에서 휴대폰 번호 찾기 풀이이다
- 매우 직관적이며 짧은 표현으로 변경되었다
- 아래의 풀이는 정규식 학습 후에 분석해 보도록 한다

```
# re를 사용한 프로그램
import re

msg = 'julie 010-123-1234 peter 011-1234-1234 office 031-123-1234'
r = re.finditer(r'01[0-26-9]-[0-9]{3,4}-[0-9]{4}', msg)
for idx, m in enumerate(r, start=1):
    print(f'Phone number{idx} : {m.group()}')
```

2-01. 정규식(Regular Expression)의 이해

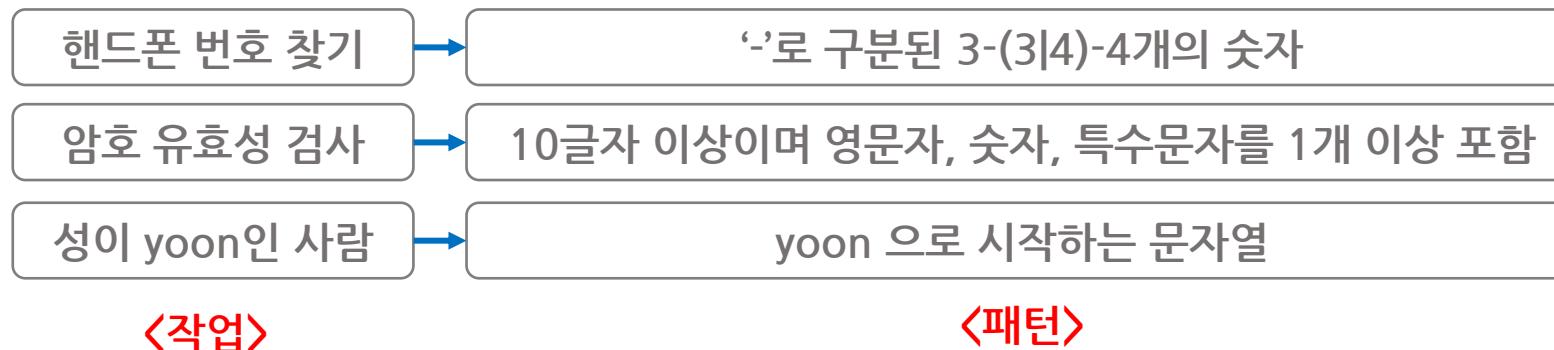


▶ 문자집합의 패턴이 일치하는지 검사하기 위해 사용하는 표현식을 의미한다

☞ Python에서는 re 모듈을 import 하여 정규식을 사용할 수 있다

- import re

☞ 문자집합으로 하고자 하는 작업과 패턴을 나타내 보면 다음과 같다



☞ 정규식 사용의 장점은 다음과 같다

- 하고자 하는 작업을 패턴(정규식)으로 표현하면 직관적인 표현이 된다
- 검색 대상이 큰 경우 보다 빠른 결과를 얻을 수 있다
- 코드의 작성길이가 짧아 진다

2-02. re 모듈 사용법

▣ [1] compile()을 사용하여 정규식을 패턴 객체로 컴파일

```
import re  
pattern = re.compile(패턴문자열)
```

import로 re 객체 사용을 알림
정규식 pattern 객체 반환 받기

▣ [2] 패턴 탐색 함수에 패턴객체 및 문자열을 전달하고 Match 객체 받기

- mc=re.search(pattern, 문자열)
- mc=pattern.search(문자열)

search()는 탐색 함수, 다양한 함수 존재
문자열은 패턴을 조사할 대상을 의미함

▣ 탐색 함수의 인수로 패턴 객체를 전달하지 않고 바로 문자열을 사용해도 된다

mc=re.search(pattern, 문자열)

=

mc=re.search(패턴문자열, 문자열)

▣ [3] 결과로 받은 Match 객체의 정보 출력

```
print("{}~{}에 {}존재".format(mc.start(), mc.end(), mc.group()))
```

2-02. 간단한 re 모듈 사용 예

▶ 2-02 실습) 다음 예제를 실행하여 re모듈을 이용하는 방법을 정리하라

```
import re  
  
pattern = re.compile(r'abc') # raw string 패턴문자열 -> Pattern 객체  
  
mc1 = re.search(pattern, '123abcd') # re.search -> Match객체  
mc2 = re.search(r'abc', '123abcd') ➔ 동일한 결과를 갖음  
mc3 = pattern.search('123abcd')  
  
for mc in mc1, mc2, mc3 :  
    print(f'{mc.start()}~{mc.end()}에 {mc.group()}존재')
```

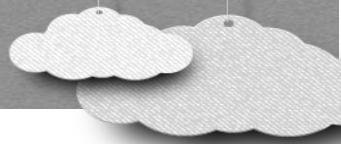
패턴은 'W'를 그대로 전달하기 위해
원시 문자열('문자열')을 사용

찾아진 패턴의 시작, 끝+1, 내용

- 앞으로 패턴에 사용될 문자학습에서는 위 예제의 형식을 사용한다
- Match 객체에서 찾아진 패턴의 **시작, 끝+1, 내용**을 표시한다

3~6에 abc존재
>>>

2-03. 정규식의 character class



- ▶ 대괄호 내부에 문자를 나열한 것을 character class라고 함

character class	의미
[abc]	abc 중 한 개 문자
[a-z]	a에서 z까지 범위의 문자 중 1개
[a-zA-Z]	알파벳 전부 (소문자 + 대문자)
[^0-9]	^를 사용하면 not의 의미임, 숫자를 제외한 전부

- ▶ 2-03 실습) 다음의 정규식을 해석하고 결과를 예측하라

```
import re

mc = re.search(r'', '123abcd')
if mc :
    print(f'{mc.start()}~{mc.end()}에 {mc.group()}존재')
else:
    print('문자열에 패턴이 존재하지 않음')
```

해석

2-04. 횟수 관련 메타 문자



2-04 실습) 패턴에 사용하는 반복 관련 메타 문자의 종류는 다음과 같다

메타 문자는 문자가 가진 뜻이 아닌 특별한 용도로 사용되는 문자를 의미한다

메타 문자	의미	예시	
x^*	x 를 0번 이상 반복	$r' [0-9]^*$ '	숫자가 0회 이상 사용된 패턴 찾기 (Greedy)
x^+	x 를 1번 이상 반복	$r' [a-z]^+$ '	알파벳 소문자가 1회 이상 사용된 패턴 찾기 (Greedy)
$x^?$	x 를 0 또는 1개	$r' [A-Z]?$ '	알파벳 대문자가 없거나 1회 사용된 패턴 찾기
$x^{\{n\}}$	x 를 n번 반복	$r' [0-9]^{\{3\}}'$	숫자가 3회 연속 사용된 패턴 찾기
$x^{\{n, k\}}$	x 를 n번 이상 k번 이하 반복	$r' [0-9]^{\{3,\}}$ '	숫자가 3회 이상 연속 사용된 패턴 찾기
		$r' [0-9]^{\{,,3\}}$ '	숫자가 3회 이하 연속 사용된 패턴 찾기 (0개 포함)

- $r' [0-9]^{\{3,4\}}$ - 숫자가 3이상 4이하 반복
- {0,}은 *, {1,}은 + {0, 1}은 ?와 동일한 의미

2-05. 여러 가지 메타 문자



▶ 2-05 실습) 패턴에 사용할 수 있는 메타 문자의 종류는 다음과 같다

메타 문자	의미	예시	
.	문자 한 개 (\n은 제외)	r'a..o'	a로 시작하고 o로 끝나는데 사이에 2개의 문자가 있는 패턴
^x	문자열의 시작이 x	r'^[a-z]+'	1개 이상의 알파벳 소문자로 시작되는 패턴에서 '알파벳 소문자' 검색
x\$	문자열의 끝이 x	r'[0-9]+\$'	1개 이상의 숫자로 끝나는 패턴에서 '숫자' 검색
x y	x or y 중 하나	r'abc 123'	abc 또는 123 패턴 찾기

- `\W`을 포함한 모든 문자를 표현하고 싶을 경우
→ `re.DOTALL` 또는 `re.S` 옵션으로 정규식을 `compile` 한다

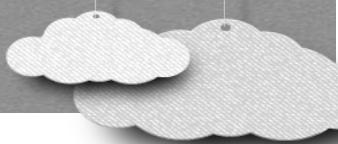
```
pattern = re.compile('.abc')
ret = pattern.search('\Wnabc')
```

결과: None

```
pattern = re.compile('.abc', re.S)
ret = pattern.search('\Wnabc')
```

결과: match='\Wnabc'

2-06. 그룹 관련 메타 문자



▶ 패턴에 사용할 수 있는 그룹관련 메타 문자의 종류는 다음과 같다

▶ 다른 메타 문자 종류는 C:\Python\Lib\re.py 의 docstring을 참조한다 → print(re.__doc__)

메타 문자	예시
(x)	() 안의 내용을 그룹화하고 내용이 캡처 됨, 그룹 번호 1번부터 부여
\1...\99	<ul style="list-style-type: none">▪ 캡쳐 된 그룹 1번 ~ 99번의 사용▪ 앞에 사용된 내용이 뒤에 동일하게 반복되어야 하는 경우 사용함
(?:x)	패턴은 검사하나, 캡쳐 하지 않으므로 그룹번호 부여되지 않음
(?P<name>...)	<ul style="list-style-type: none">▪ () 안의 내용을 그룹화하고 내용을 캡처 함, 그룹의 이름 지정▪ 사용시 “name”으로 사용 (예) match.group(“name”)▪ 그룹 번호로도 사용할 수 있음 (위치에 따른 번호는 동일하게 적용됨)

캡쳐: 저장되는 것을 의미하며, 후에 번호를 사용해 꺼내 사용할 수 있다

(세자리숫자) (: - 공백' 중 1글자) (세 자리 또는 네 자리 숫자) (네 자리 숫자)

(\d{3})([:]|\d{3,4})\d{2}(?:\d{4}) → 총 3개의 그룹이 생성됨

2-06. 그룹관련 메타문자 연습



2-06 실습) 다음 예제를 수행하여 결과를 분석하라

- # 1. 숫자 그룹 => 숫자 3글자
- # 2. 구분 기호 그룹 => : 또는 - 또는 공백 문자가 있을 수 있으며, 그룹 지정
- # 3. 그룹번호 부여하지 않는 숫자 그룹 => 숫자 3글자 또는 4글자
- # 4. 2번에서 사용된 구분 기호 => 첫 번째 구분 기호와 동일한 것을 사용함
- # 5. 이름을 pnum으로 갖는 숫자 그룹 => 숫자 4글자

```
import re
pattern = re.compile(r'([0-9]{3})([:|-| ])((?:[0-9]{3,4})\2(?P<pnum>[0-9]{4}))')
mc = re.search(pattern, '010-123-4567')
if mc :
    print('{}~{}에 {}존재'.format(mc.start(), mc.end(), mc.group()))
    for x in mc.groups():
        print(x)
    print(mc.group(3), mc.group('pnum'))
else:
    print('문자열에 패턴이 존재하지 않음')
```

match객체.groups()
→ '캡처 된 그룹'들을 문자열 튜플로 반환

0~12에 010-123-4567존재
010
-
4567
4567
3개의 캡쳐 된 그룹이 존재함

2-07. Special Sequences -1

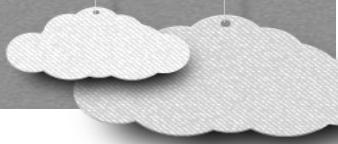
▶ 패턴에 사용할 수 있는 축약형 문자의 종류는 다음과 같다

축약형 문자	의미	character class 표현
\d	숫자 한 개	[0-9]
\D	\d가 아닌 문자	[^0-9]
\w	숫자 + 문자(알파벳, 한글 등) + _	[가-힣a-zA-Z0-9_]
\W	\w가 아닌 문자	[^가-힣a-zA-Z0-9_]
\s	화이트 스페이스 문자	공백 있음 → [\t\n\r\f\v]
\S	\s가 아닌 문자	[^ \t\n\r\f\v]
\^	메타 문자 ‘^’를 일반 문자로 취급	
\A	문자열의 시작부터 매치되는 문자열을 찾음	
\Z	문자열의 끝부터 매치되는 문자열을 찾음 - \A, \Z는 여러 줄 문자열도 1개의 문자열로 취급함	

^ \$: 여러 줄 문자열인 경우 옵션에 따라서 여러 줄을 각 줄로 나누어서 사용하거나 하나의 문자열로 사용

\A, \Z : 여러 줄 문자열인 경우 항상 하나의 문자열로 사용

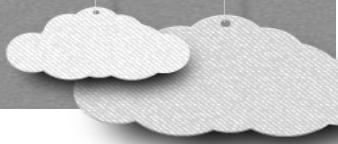
2-08. Special Sequences -2



(matches empty string)

메타 문자	설명
\b	<ul style="list-style-type: none">▪ boundary - \w와 \W의 사이를 의미함(단어의 경계)▪ 단어의 시작 또는 끝에서 매치되는 것을 찾음▪ 맨 앞에 사용할 경우 단어의 시작▪ 맨 뒤에 사용할 경우 단어의 끝
\B	<ul style="list-style-type: none">▪ boundary가 아닌 것▪ 단어의 시작 또는 끝이 아닌 곳에서 매치되는 것을 찾음▪ 맨 앞에 사용할 경우 단어의 시작이 아닌 것▪ 맨 뒤에 사용할 경우 단어의 끝이 아닌 것
특이사항	<ul style="list-style-type: none">▪ \b, \B는 패턴의 맨 앞, 뒤에 사용함▪ \b, \B는 \w(워드문자)가 아닌 문자를 구분문자로 취급함

2-08. Special Sequences -2



▶ 2-08 실습) 실습 예제 실행을 통해 이해하여 두시기 바랍니다.

패턴 문자열	해석
r'pi'	▪ pi가 포함된 모든 문자열
r'\bpi'	▪ 단어의 시작이 pi인 것
r'\bp\w+ed'	▪ p로 시작하는 단어이면서, ed로 끝나는 모든 것
r'\bp\w+ed\b'	▪ 경계 + p + 워드문자 1개 이상 + ed ▪ 경계 + p + 워드문자 1개 이상 + ed + 경계
r'\B\w+\B'	▪ 경계가 아닌 것 + 워드 문자 1개 이상 + 경계가 아닌 것
r'\B\w+'	▪ 경계가 아닌 것이 앞에 있고 ed가 있는
r'\Bed'	▪ 단어의 시작이 ed가 아닌, ed를 찾기
r'\w+\B'	▪ pic가 있고 경계가 아닌 것이 뒤에 있는
r'pic\B'	▪ pic로 끝나지 않는 pic 검색
r'\b\w+[!?.]+'	▪ word문자 1개 이상으로 시작하는 단어이면서 그 뒤로 [!?.] 중 하나 이상이 포함된
r'\b\w+[!?.]+\B'	▪ word문자 1개 이상으로 시작하는 단어이면서 [!?.] 중 하나 이상이 포함되었지만 끝나지 않는 ▪ [!?.] -> 단어문자가 아님

2-09. 패턴 작성 연습 1



▶ 2-09 실습) 실습 예제 실행을 통해 이해하여 두시기 바랍니다.

▶ 패턴은 'W'를 그대로 전달하기 위해 원시 문자열(r‘문자열’)을 사용하는 것이 좋다

패턴	해석
r'\b\d+-\d+-\d+\b'	<ul style="list-style-type: none">경계문자 + 숫자1개 이상 + '-' + 숫자1개 이상 + '-' + 숫자1개 이상 + 경계문자birthday:1990-12-12 : 9~19에 1990-12-12존재birthday1990-12-12 : 문자열에 \b\d+-\d+-\d+\b패턴이 존재하지 않음93-6-30 : 0~7에 93-6-30존재a-b-c : 문자열에 \b\d+-\d+-\d+\b패턴이 존재하지 않음
r'\b\d{3}-\d{3,4}-\d{4}\b'	<ul style="list-style-type: none">경계문자 + 숫자3개 + '-' + 숫자3~4개 + '-' + 숫자4개 + 경계문자A:123-123-1234 : 2~14에 123-123-1234존재123-1234-1234 : 0~13에 123-1234-1234존재C:12-34-5678 : 문자열에 \b\d{3}-\d{3,4}-\d{4}\b패턴이 존재하지 않음
r'\b01[0-26-9]-\d{3,4}-\d{4}\b'	<ul style="list-style-type: none">경계문자 + (010 011 012 016 017 018 019 중 1개) + '-' + 숫자3~4개 + '-' + 숫자4개 + 경계문자phone:010-123-1234 : 6~18에 010-123-1234존재phone010-123-1234 : 문자열에 왼쪽의 패턴이 존재하지 않음018-1234-1234 : 0~13에 018-1234-1234존재001-567-9876 : 문자열에 왼쪽의 패턴이 존재하지 않음

2-10. 패턴 작성 연습

2-10 실습) 다음의 패턴 및 해석을 살펴 보아 활용법을 익히도록 한다

패턴	해석
r'(so){3,5}'	<ul style="list-style-type: none">so가 3번에서 5번 반복, 최대 일치를 찾음 – greedy (큰 횟수)sososo, sosososo, sososososo 모두 매치<code>re.search(r'(so){3,5}', 'sososososo')</code> → sososososo
r'(so){3,5}?'	<ul style="list-style-type: none">so가 3번에서 5번 반복, 최소 일치를 찾음 – non-greedy (작은 횟수)sososo, sosososo, sososososo 모두 매치<code>re.search(r'(so){3,5}?', 'sososososo')</code> → sososo
r'(sa so){3,4}'	<ul style="list-style-type: none">sa 또는 so를 3회 ~ 4회 반복sasasa, sososo, sasosa, sasaso, sososa, sosaso ... 모두 매치<code>re.search(r'(sa so){3}', 'sososososo')</code> → sososo
r'^\d+\$'	<ul style="list-style-type: none">숫자 1개 이상으로 이루어진 문자열 (숫자만으로 이루어진 문자열)1, 12, 1234 모두 매치12ab34, 12 34 와 같이 중간에 숫자가 아닌 것이 있으면 안됨

3-01. re 모듈의 탐색 함수

☞ re 모듈에서 검색에 자주 사용되는 탐색 함수는 다음과 같다

함수	동작
re.findall(패턴,문자열)	매치하는 모든 패턴을 겹치지 않게 탐색하여 List로 반환
re.finditer(패턴,문자열)	findall()과 같은 동작이며, Match iterator 객체 로 반환
re.search(패턴,문자열)	<ul style="list-style-type: none">문자열의 임의지점에서 패턴과 매치하는 텍스트 탐색결과를 Match 객체로 반환
re.match(패턴,문자열)	<ul style="list-style-type: none">문자열의 시작지점에서 패턴과 매치하는 텍스트 탐색결과를 Match 객체로 반환
re.fullmatch(패턴,문자열)	<ul style="list-style-type: none">문자열에서 패턴과 전체 매치하는 텍스트 탐색결과를 Match 객체로 반환

- 매칭 되는 것이 없으면 **None**을 반환한다
- None**은 if문에서 **False**와 같이 취급 된다 if not None: print('True')

3-01. re 모듈의 탐색 함수 - findall() 함수



3-01 실습1) findall() 함수는 겹치지 않는 패턴 매치를 모두 찾는다

겹치는 것과 겹치지 않는 것은 다음과 같다



겹치는 것



겹치지 않는 것

- re.findall() 의 첫 번째 인수로 패턴을 두 번째 인수로 검색 문자열을 넘긴다
- re.findall() 은 찾은 패턴 매치에 대해서 문자열 리스트 (list)로 반환한다

```
# findall
ret = re.findall(r'wow', 'wowowowow')
print(type(ret))
for t in ret:
    print(f"Matched : {t} ")
```

```
Matched : wow
Matched : wow
>>>
```

문자열

결과가 문자열 List 이기 때문에 문자열로 변환하는 별도 작업이 불필요함

3-01. re 모듈의 탐색 함수 - finditer() 함수

- 3-01 실습2) finditer() 함수는 겹치지 않는 패턴 매치를 모두 찾는다
- 다음 예제를 실행하여 ret, t의 type을 살펴보도록 한다

```
# finditer
ret = re.finditer(r'wow', 'wowowowow')
print(type(ret))
for t in ret:
    print(f"Matched : {t}")
```

```
<class 'callable_iterator'>
Matched : <_sre.SRE_Match object; span=(0, 3), match='wow'>
Matched : <_sre.SRE_Match object; span=(4, 7), match='wow'>
```

- finditer()의 반환은 Match iterator 객체이며 일치하는 것이 없어도 객체 반환
- Match 객체에서 각 요소의 string 결과를 얻기 위해 group() 함수 사용

3-01. re 모듈의 탐색 함수 - search() 함수



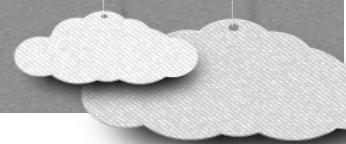
- 3-01 실습3) search()는 문자열 임의지점에서 매치하는 텍스트를 탐색 한다
- 알파벳 소문자 두 글자가 어떤 위치에 있어도 매치로 판단 한다

```
# search
pattern = re.compile(r'[a-z]{2}')
ret1 = pattern.search('123abc123')
ret2 = pattern.search('abcXde')
if ret1 : print(f'Matched ret1 : {ret1.group()}')
if ret2 : print(f'Matched ret2 : {ret2.group()}')
```

- 패턴: 알파벳 소문자 두 글자 연속
- 문자열 '123abc123' → ab 매치
- 문자열 'abcXde' → ab 매치

- search()의 반환은 Match 객체이며 일치하는 것이 없으면 None 반환
- Match 객체에서 string 결과를 얻기 위해 group() 함수 사용

3-01. re 모듈의 탐색 함수 - match() 함수



- 3-01 실습4) match() 는 문자열의 시작부분부터 매치를 탐색 한다
- match 함수는 시작부터 패턴이 일치되지 않으면 찾지 않는다 (중간부터 매칭X)

```
# match
pattern = re.compile(r'[a-z]{2}')
ret1 = pattern.match('123abc123')
ret2 = pattern.match('abcXde')
if ret1 : print(f'Matched ret1 : {ret1.group()}')
if ret2 : print(f'Matched ret2 : {ret2.group()}')
```

- 패턴: 알파벳 소문자 두 글자 연속
- 문자열 '123abc123' → None
- 문자열 'abcXde' → ab 매치

- match()의 반환은 Match 객체이며 일치하는 것이 없으면 None 반환
- Match 객체에서 string 결과를 얻기 위해 group() 함수 사용

3-01. re 모듈의 탐색 함수 - fullmatch() 함수



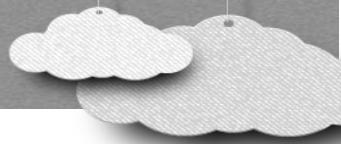
3-01 실습5) fullmatch() 는 문자열의 전체 매치하는 텍스트를 탐색 한다

알파벳 대문자 세 글자로 이루어진 문자열만 매치로 판단 한다

```
# fullmatch
pattern = re.compile(r'[a-z]{2}')
ret1 = pattern.fullmatch('abcd')
ret2 = pattern.fullmatch('ab')
if ret1 : print(f'Matched ret1 : {ret1.group()}')
if ret2 : print(f'Matched ret2 : {ret2.group()}')                                Matched ret2 : ab
>>>

# 숫자 1개 이상으로 이루어진 문자열 (숫자만으로 이루어진 문자열)
data = ('1', '12', '1234', '12ab34', '12 34')
for x in data :
    r = re.fullmatch(r'\d+', x)      <_sre.SRE_Match object; span=(0, 1), match='1'>
    print(r)                        <_sre.SRE_Match object; span=(0, 2), match='12'>
                                    <_sre.SRE_Match object; span=(0, 4), match='1234'>
                                    None
                                    None
>>>
```

3-02. re 모듈의 치환/분리 함수



3-02 실습) re 모듈에서 치환/분리에 사용되는 함수는 다음과 같다

함수	동작
re.sub(패턴, 치환문자열, 문자열)	문자열에서 패턴과 매치하는 텍스트를 치환함
re.split(패턴, 문자열)	<ul style="list-style-type: none">▪ 문자열을 패턴 기준으로 분리함▪ 문자열 list로 반환하며, 패턴에 맞는 것이 없으면 문자열을 분리하지 않고 1개 아이템으로 취급함

```
import re

ret = re.sub('[a-z]+', 'X', 'abc1de2fg3hij')
print(ret)
ret = re.split(r'\d', 'abc1de2fg3hij')
print(ret)
```

X1X2X3X
['abc', 'de', 'fg', 'hij']
=>

3-03. Match 객체의 함수



Match 객체에서 사용되는 함수는 다음과 같다

함수	동작
객체.group()	매치되는 텍스트를 string으로 변환하여 반환
객체.group(번호)	<ul style="list-style-type: none">그룹 번호에 해당하는 텍스트를 string으로 변환하여 반환그룹 번호는 1부터 시작하여 1씩 증가하는 정수
객체.group(번호1, 번호…)	여러 개의 그룹을 텍스트로 변환하여 튜플 반환
객체.groups()	매치되는 텍스트들을 string으로 변환하여 튜플 반환
객체.start()	매치된 텍스트의 시작 위치 반환
객체.end()	매치된 텍스트의 끝 위치 반환
객체.span()	매치된 텍스트의 (시작, 끝)에 해당되는 튜플 반환

3-03. Match 객체의 함수

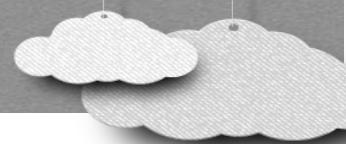


▶ 3-03 실습) 다음 실습을 통해 match 객체의 다양한 메서드의 동작을 이해한다

```
import re

pattern = re.compile(r'(?P<localcode>\(\d{3}\)) (\d{3})-(\d{4})')
ret = pattern.search('(031) 234-2345')
print(ret.group(1))                                (031)
print(ret.group('localcode'))                      (031)
print(ret.group(2))                                234
print(ret.group(3))                                2345
print(ret.groups())                               ('(031)', '234', '2345')
print(ret.group(2, 3))                            ('234', '2345')
print(ret.group(), ret.start(), ret.end(), ret.span())
                                                (031) 234-2345 0 14 (0, 14)
                                                >>>
```

3-04. compile()의 옵션



Pattern 객체를 반환하는 compile()의 옵션의 종류를 살펴보자

☞ re.compile(패턴문자열, 옵션) 과 같이 사용한다

- 여러 개의 옵션을 동시 적용하기 위해서는 or 연산을 하는 '|' 문자를 사용한다

옵션	의미
re.DOTALL, re.S	. 메타문자에 \n을 포함한 글자 1개를 표현할 수 있도록 함
re.IGNORECASE, re.I	대/소문자를 구분하지 않고 매치 수행
re.MULTILINE, re.M	^ 및 \$과 연관 있는 옵션으로 문자열이 '여러 줄'인 것을 허용
re.VERBOSE, re.X	이해 어려운 정규식에 주석 또는 라인단위로 구분하여 표시할 수 있도록 처리

☞ re.search()와 같은 탐색 함수에도 옵션을 넣어 사용할 수도 있다

- re.탐색함수(패턴, 문자열, 옵션) 과 같이 사용한다

3-04. compile()의 옵션 - re.S



- 3-04 실습1) 다음 코드의 패턴을 해석하고, 결과를 예측하라
- 각 옵션을 사용할 때와 사용하지 않을 때의 결과를 비교하라
- compile() 또는 탐색함수에 옵션으로 사용하는 방법을 모두 연습하라

```
import re

pt1 = re.compile(r'.*')
pt2 = re.compile(r'.*', re.DOTALL)
pt3 = re.compile(r'.*', re.S)
for pattern in (pt1, pt2, pt3) :
    ret = re.search(pattern, 'Good\nBetter')
    print(ret.group())
    print('-' * 10)

Good
-----
Good
Better
-----
Good
Better
-----
>>>
```

3-04. compile()의 옵션 - re.I



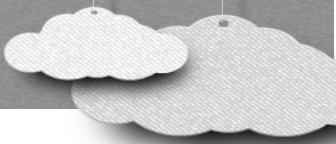
- 3-04 실습2) 다음 코드의 패턴을 해석하고, 결과를 예측하라
- 각 옵션을 사용할 때와 사용하지 않을 때의 결과를 비교하라
- compile() 또는 탐색함수에 옵션으로 사용하는 방법을 모두 연습하라

```
import re

pt1 = re.compile(r'abc')
pt2 = re.compile(r'abc', re.IGNORECASE)
pt3 = re.compile(r'abc', re.I)
for pt in pt1, pt2, pt3 :
    ret = re.findall(pt, 'abcABCAbC')
    print(ret)
```

	['abc'] ['abc', 'ABC', 'AbC'] ['abc', 'ABC', 'AbC'] =>>>
--	---

3-04. compile()의 옵션 - re.MULTILINE



- 3-04 실습3) 다음 코드의 패턴을 해석하고, 결과를 예측하라
- re.MULTILINE 옵션을 사용할 때와 사용하지 않을 때의 결과를 비교하라

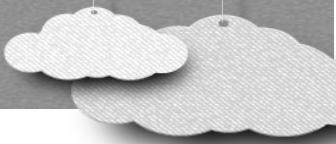
```
import re

msg = 'Good Morning!\nGood Afternoon!\nGood Evening!'
pt1 = re.compile(r'^G(.*)ing!$')
pt2 = re.compile(r'^G(.*)ing!', re.MULTILINE)
pt3 = re.compile(r'^G(.*)ing!', re.S)

ret = re.findall(pt2, msg)
print(*ret, sep='\n')  # argument unpack
print("DONE!")
```

```
<_sre.SRE_Match object; span=(0, 13), match='Good Morning!'>
<_sre.SRE_Match object; span=(30, 43), match='Good Evening!'>
DONE!
>>>
```

3-04. compile()의 옵션 - re.VERBOSE



3-04 실습4) 다음 코드의 패턴을 해석하고, 결과를 예측하라

```
import re
phone1 = re.compile(r'\\(\d{3})\\s?\\d{3}-\\d{4}')      (031) 234-2345
phone2 = re.compile(r'''(
    \\(\d{3})      # area code
    \\s?          # white space (0 or 1)
    \\d{3}        # first 3 digits
    -
    \\d{4}        # last 4 digits
)''', re.VERBOSE)
ret = re.search(phone2, 'Phone: (031) 234-2345')
print(ret.group())
```

4-01. 패턴 작성 연습 - 양수 검색



- 4-01실습) 0보다 큰 양수를 검색하는 패턴을 작성하라

- 1, 2, 3, ...
 - 단, 0, 01, 1a 등은 검색 대상으로 삼지 않는다
 - 0부터 시작된 숫자 또는 숫자 이외의 문자와 섞여 있으면 검색 대상이 아님
 - 주어진 dataList를 패턴 확인용 데이터로 사용한다
 - 실습 코드에 주어진 printMatch 함수를 사용하여 패턴의 동작을 확인한다

```
dataList = ('1', '12', '321', '100', '0', '01', '001', '1a')  
          양수           양수 아님
```

4-01. 패턴 작성 연습 - 양수 검색



- 4-01실습) 0보다 큰 양수를 검색하는 패턴을 작성하라

- 1, 2, 3, ...
 - 단, 0, 01, 1a 등은 검색 대상으로 삼지 않는다
 - 0부터 시작된 숫자 또는 숫자 이외의 문자와 섞여 있으면 검색 대상이 아님
 - 주어진 dataList를 패턴 확인용 데이터로 사용한다
 - 실습 코드에 주어진 printMatch 함수를 사용하여 패턴의 동작을 확인한다

```
dataList = (['1', '12', '321', '100'], ['0', '01', '001', '1a'])
```

4-02. 패턴 작성 연습 - 날짜 형식 찾기



4-02실습) 날짜 형식을 검색하는 패턴을 작성하라

- ☞ 년도는 숫자 4글자이면서 19 또는 20으로 시작함
- ☞ 월을 01 ~ 12로 표시되고, 일은 01~31로 표시됨
- ☞ 윤달 및 28, 29, 30으로 끝나는 달을 별도 확인하지 않음
- ☞ '-'를 사용하여 년, 월, 일을 구분함
 - 예) 1990-09-12, 2021-12-02
- ☞ 실습 코드에 주어진 printMatch 함수를 사용하여 패턴의 동작을 확인한다

```
data = ('1990-09-12', '2020-12-31', '2099-10-32', '89-01-08')
```

날짜 형식

날짜 형식 아님

4-03. 패턴 작성 연습 - 메일 주소 찾기

4-03실습) 메일 주소 형식을 검색하는 패턴을 작성하라

- ☞ @ 기호 앞은 '.'을 포함한 워드문자로 구성된 문자열이 올 수 있다 ('.'이 있을 수도 있고 없을 수도 있음)
- ☞ @ 기호
- ☞ @ 기호 뒤는 워드 문자가 1개 이상 온다
- ☞ 위의 1개 이상의 워드 문자 뒤로 '.'으로 시작하고 워드문자 1개 이상으로 구성된 문자열이 1회 또는 2회 반복되어 사용된다 ('.'이 1, 2회 사용되어야 함)
- ☞ 실습 코드에 주어진 printMatch 함수를 사용하여 패턴의 동작을 확인한다

```
data = ('abc@naver.com', 'abcdef@abc')
```

메일 주소 형식

메일 주소 형식 아님

4-04. 한글, 콤마(,) 및 소수 부분 제거



- 금액에 함께 표기된 한글, 콤마(,) 및 소수 부분을 제거한다
 - re.sub() 를 활용하여 removeThings(data): 함수를 작성한다
 - 다음의 코드와 출력 예시를 참조한다

```
import re

def removeThings(data):
    pass

datalist = ('1,234,567원', '200,000.00달러', '1,234.15엔')
for x in datalist:
    r = removeThings(x)
    print(f'{x} => {r}')
```

```
1,234,567원 => 1234567
200,000.00달러 => 200000
1,234.15엔 => 1234
>>>
```

4-05. 주민등록번호 처리



4-05 실습) 주민등록번호에 대한 다음의 처리를 수행하는 코드를 작성하라

有效性 검사를 수행한다 (정밀한 검증이 아닌 형태에 대한 검증)

- 숫자6자-숫자7자로 구성 된 경우 주민등록번호로 인정한다

유효한 주민등록번호인 경우 뒤의 7자를 *로 치환하여 출력하도록 한다

유효하지 않은 주민등록번호인 경우 'XX is NOT an id'를 출력 한다

다음의 코드와 출력 예시를 참조한다

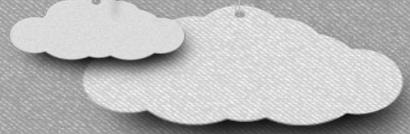
```
import re

def idcheck(data):
    pass

datalist = ('881103-1231233', '891312-1234512', '1990-12-28')
for x in datalist:
    r = idcheck(x)
    print(f'{x} => {r}')

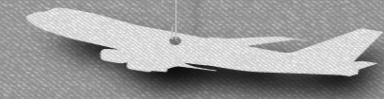
```

881103-1231233 => 881103-*****
891312-1234512 => 891312-1234512 is not an id
1990-12-28 => 1990-12-28 is not an id
>>>

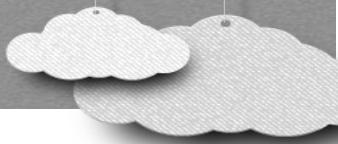


Chapter - 03

class 기본



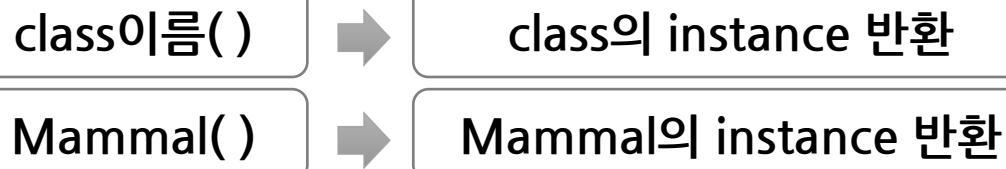
1-01. 클래스(class) 생성



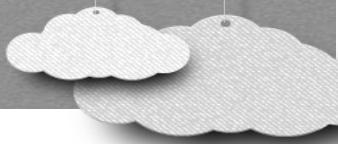
▶ class 및 instance 생성

```
class Mammal: -----> 클래스이름 앞에 class 키워드 사용  
    pass -----> 하나 이상의 statement (속성, 메서드 추가 위치)  
  
m = Mammal() -----> instance 생성을 위한 생성자(constructor) 호출  
  
print(id(m), m) -----> <__main__.Mammal object at 0x0000023AE55C2A88>  
print(id(Mammal), Mammal) -----> <class '__main__.Mammal'>
```

▶ 생성자 호출



1-02. 클래스(class) 생성



속성(attribute)을 갖는 instance를 생성하는 클래스

↗ `__init__` 메서드

↗ 생성자를 호출하면 `__init__` 메서드가 실행되고 instance를 반환한다

Mammal(argument ...) → `__init__(self, parameter ...)` → class의 instance 반환

```
class Mammal:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age
```

name, age 두 개의 속성(attribute)을 갖는 instance의 초기화 메서드

self

instance 를 reference하는 이름

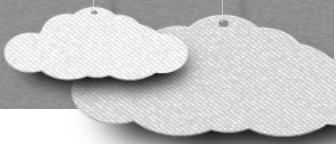
```
m1 = Mammal('Happy', age=3)  
m2 = Mammal('Tom', age=2)
```

→ 두 개의 instance 생성

```
print(m1.name, m1.age)  
print(m2.name, m2.age)
```

} instance의 attribute 사용

1-03. class / instance namespace



☞ class와 instance는 각각 namespace를 갖는다

```
class Mammal:
```

```
    def __init__(self, name, age):  
        self.name = name  
        self.age = age
```

```
m1 = Mammal('Happy', age=3)  
m2 = Mammal('Tom', age=2)
```

instance namespace에 name, age 등록

class namespace

Mammal.__dict__

m1' instance namespace

m1.__dict__

m2' instance namespace

m2.__dict__

class namespace

→ class 생성 과정에서 namespace를 할당 받고 내용이 채워짐

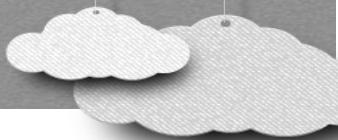
instance namespace

→ instantiation 과정에서 namespace를 할당 받고 내용이 채워짐

__init__ 메서드에서 instance의 attribute 추가 동작을 함

instance namespace에 존재하지 않는 이름은 class namespace에서 검색함

1-04. instantiation 과정 이해



- 생성자를 호출하면 `__new__`, `__init__` 메서드가 차례로 실행되고 instance를 반환한다

`__new__`

argument 확인 및 instance 생성

`__init__`

instance namespace 초기화

```
class Mammal:
```

```
    def __new__(cls, *args, **kwargs):  
        print('__new__', cls.__name__)  
        print(args, kwargs)  
        return super().__new__(cls)
```

```
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
        print('__init__', self.__class__)
```

```
m = Mammal('Happy', age=3)
```

```
__new__ Mammal  
('Happy',)  
{'age': 3}  
__init__ <class '__main__.Mammal'>
```

cls

class 를 reference하는 이름

1-05. 클래스(class) 생성



사용자 정의 method를 갖는 클래스

```
class Mammal:  
    def __init__(self, age):  
        self.age = age  
    print("init Mammal")
```

```
def printAge(self):  
    print(f'my age is {self.age}')
```

```
m = Mammal(20)  
m.printAge()
```

age를 출력하는 method

```
class Mammal(object):  
    def __init__(self, age):  
        self.age = age  
    print("init Mammal")
```

```
def __add__(self, other):  
    return self.age + other.age
```

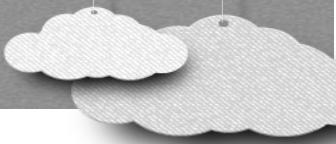
```
m1 = Mammal(20)  
m2 = Mammal(30)  
print(m1 + m2)
```

+ 연산에 대한
special method

```
print(m1 + 10)  AttributeError: 'int' object has no attribute 'age'
```

```
def __add__(self, other):  
    return self.age + other if type(other) == int else self.age + other.age
```

2-01. 상속(inheritance)

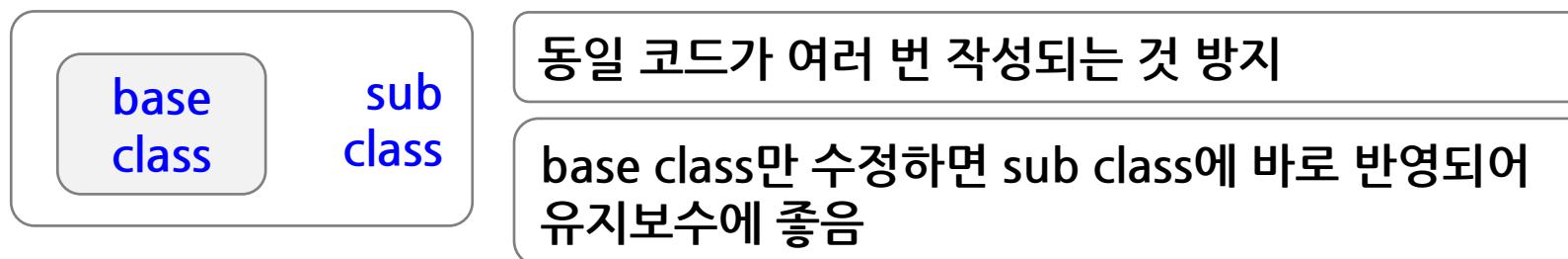


☞ sub class, base class의 개념



☞ 상속의 장점

- ☞ sub class는 base class의 attributes와 method 를 자신의 것 처럼 사용할 수 있다



2-02. 클래스(class) 생성

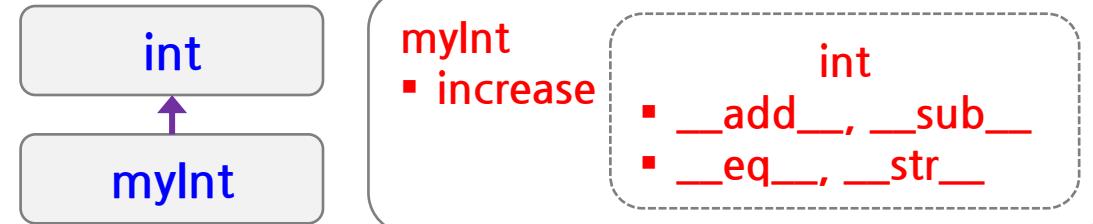


▶ base class를 지정하는 클래스

```
class myInt(int):  
  
    def increase(self, step):  
        if type(step) != int :  
            raise TypeError  
        return self.real + step
```

```
a = myInt(5)  
b = myInt(10)  
print(a+b, a-b, a*b, a/b)  
  
c = a.increase(3.5)  
print(c)
```

class ClassName(BaseClassName) : suite



base class

sub class

여러 sub class에 사용될 공통된 속성 정의

자신만의 속성 추가, 변경(이름이 같은 경우)

base 클래스에서 필요로 하는 정보 제공

2-03. MRO(Method Resolution Order)



▶ MRO : class의 namespace에서 이름을 찾는 순서

instance namespace → class namespace (MRO를 따른)

```
>>> class A : pass
```

```
>>> class B(A) : pass
```

```
>>> class C(B) : pass
```

```
>>> C.__mro__
```

→ MRO 확인을 위한 속성(attribute)
(<class '__main__.C'>, <class '__main__.B'>, <class '__main__.A'>, <class 'object'>)

2-04. 상속 관계의 이해 - 1/3



▶ 간단한 상속을 구현하여 상속 관계를 이해하여 보자

```
class Mammal:  
    def __init__(self):  
        print("init 1:", self.__class__.__name__)  
  
    def printAge(self):  
        print(f"my age is {self.age}")  
  
class Dog(Mammal):--> base 클래스 지정  
    def __init__(self, age):  
        print("init 2:", self.__class__.__name__)  
        self.age = age  
  
d = Dog(100)      init 2: Dog  
d.printAge()      my age is 100
```

base class



여러 sub class에 사용될 공통된 속성 정의

sub class



자신만의 속성 추가, 변경(이름이 같은 경우)

base 클래스에서 필요로 하는 정보 제공

2-04. 상속 관계의 이해 - 2/3

▶ Dog의 base 클래스의 `__init__()` 호출방법을 살펴보자

```
class Mammal:  
    def __init__(self):  
        print("init 1:", self.__class__.__name__)  
  
    def printAge(self):  
        print(f"my age is {self.age}")  
  
class Dog(Mammal):  
    def __init__(self, age):  
        print("init 2:", self.__class__.__name__)  
        self.age = age  
        super().__init__()  
        #Mammal.__init__(self)  
  
d = Dog(100)      Mammal 클래스의 __init__() 호출  
d.printAge()      자동으로 전달되는 argument 없음
```

base 클래스 지정

MRO에서 Dog 다음 클래스의 `__init__()` 호출

첫 번째 argument로 `self`가 자동으로 전달됨

추가 argument 전달 가능

2-05. 상속 관계의 이해 - 3/3



Dog의 base 클래스의 __init__() 호출방법을 살펴보자

```
class Mammal:  
    def __init__(self, age): → parameter로 age 추가 (이름은 같을 필요 없음)  
        print("init 1:", self.__class__.__name__)  
        self.age = age  
  
    def printAge(self): instance namespace에 age 등록  
        print(f"my age is {self.age}")  
  
class Dog(Mammal):  
    def __init__(self, name, age):  
        print("init 2:", self.__class__.__name__)  
        self.name = name  
        super().__init__(age) → self, age를 Mammal에 전달  
        #Mammal.__init__(self, age)  
  
    def printName(self): → Dog에서 추가 한 메서드  
        print(f"my name is {self.name}")
```