파일 입출력

파일 다루기 : os 모듈 활용

os 모듈 활용:

- 문자열로 path 만들기를 할때 **r**로 시작하면 raw를 의미한다:

예) 'c:□□spam□□eggs.png'

r'c:□spam□eggs.png'

파일 다루기 : os 모듈 활용

os 모듈 활용:

- 운영체계 마다 path를 만드는 방법이 다르다. 그러므로 다음과 같이 문자열을 연결한다:

```
import os
os.path.join(str1, str2, str3, …)
```

```
print(os.path.sep)                    # 현 운영체계의 path separator.
```

파일 다루기 : os 모듈 활용

os 모듈 활용:

- 다음과 같은 방식으로 현 작업 폴더의 위치를 알아볼 수 있고 변경할 수 있다:

```
# 현 작업 폴더.
print(os.getcwd())
# 새로운 작업 폴더로 이동.
os.chdir(str_path)
```

파일 다루기 : os 모듈 활용

os 모듈 활용:

- 절대 경로(path) 만들기:

```
# 현 작업 폴더 바탕으로 절대 path 만들기.
str_path_abs = os.path.abspath("토너주문.txt")
print(str_path_abs)
```

```
print(os.path.isabs(str_path_abs))                # 절대 path 확인.
```

```
# 파일 이름만 가져옴.
print(os.path.basename(str_path_abs))
# 폴더 부분만 가져옴.
print(os.path.dirname(str_path_abs))
```

파일 다루기 : os 모듈 활용

os 모듈 활용:

- 경로(path)가 폴더인지 파일인지 인식하는 방법:

```
#폴더인지 확인.
print(os.path.isdir(str_path))
print(os.path.isdir(str_path_abs))
```

```
#파일인지 확인.
print(os.path.isfile(str_path))
print(os.path.isfile(str_path_abs))
```

파일 다루기 : os 모듈 활용

os 모듈 활용:

- 폴더 내용 (파일) 보기:

```
# 현 작업 폴더의 파일 리스팅.
list_dir = os.listdir()
list_dir.sort()

# 'c' 또는 'C' 로 시작하는 파일만 보여줌.
for x in list_dir:
    if x.lower()[0] == 'c':
        print(x)
```

파일 다루기 : os 모듈 활용

os 모듈 활용:

- 폴더의 트리 구조 보기:

```
# 현 폴더 이하 트리 구조 출력.
for folderName, subFolder, fileName in os.walk(str_path):
    print(folderName)
    print(subFolder)
    print(fileName)
```

파일 다루기 : pickle 모듈 활용

pickle 모듈 활용:

- 복합 객체 저장 했다가 읽어오기:

```
import pickle
x = [1,2,3, {'이름':'임꺽정', '나이':30, '신장':180}]        # 복합 객체.
pickle.dump(x, open('my_pickle.txt','wb'))                  # '절여둔다'.
del x                                                       # 일단 지움.
new_x = pickle.load(open('my_pickle.txt','rb'))             # 절여둔 것 가져온다.
print(new_x)
```

파일 다루기 : shelve 모듈 활용

shelve 모듈 활용:

- 딕셔너리 형태로 저장 했다가 읽어오기:

```
import shelve
# 쓰기.
x = shelve.open('MyDict')          # 3개의 바이너리 파일 생성: .bak, .dat, .dir
x['이름'] = '홍길동'                # key-value 짝.
x['나이'] = 30
x['신장'] = 180
x.close()
```

파일 다루기 : shelve 모듈 활용

shelve 모듈 활용:

- 딕셔너리 형태로 저장 했다가 읽어오기:

```
# 읽어오기.
x = shelve.open('MyDict')
print(list(x.keys()))
print(list(x.values()))
print(list(x.items()))
x.close()
```

Excel 컨트롤

openpyxl 모듈 활용:

- Workbook > Sheet > Cell:

```
import openpyxl
wb = openpyxl.load_workbook('my_excel.xlsx')      # Workbook을 객체로 가져옴.
wb.sheetnames                                     # Sheet이름을 list로 가져옴.
```

```
sh = wb['Sheet1']                                 # Sheet1을 객체로 가져옴.
cl = sh['A1']                                     # A1 셀 객체.
print(cl.value)
print(sh['A1'].value)                             # A1셀의 값을 가져오기.
print(sh.cell(1,1).value)                         # 행과 열의 위치로 가져오기.
```

Excel 컨트롤

openpyxl 모듈 활용:

- Workbook > Sheet > Cell:

```
# 여러 셀에서 값을 가져와서 출력해 본다.
for i in range(1,11):
    print(sh.cell(i,1).value )
```

Excel 컨트롤

openpyxl 모듈 활용:

- Workbook > Sheet > Cell:

```
# 새롭게 워크북을 생성한다.
my_wb = openpyxl.Workbook()          # 메모리 안에서만 존재.
print(my_wb.sheetnames)              # 신규 워크북에는 'Sheet'만 있음.
```

```
my_sh = my_wb['Sheet']
my_sh['A1'].value = 999              # 새로운 값 대입.
my_sh['A2'] = 666                    # OK.
my_sh.title = 'MySheet1'             # 시트이름 바꿈.
my_sh2 = my_wb.create_sheet(index = 0, title = 'MySheet2')
my_wb.save('my_new_excel.xlsx')      # 메모리 안의 워크북을 저장한다.
```

Word 컨트롤

docx 모듈 활용:

- Document > Paragraph > Run:

```
import docx
# Word 문서 오픈!
my_doc = docx.Document('디자인 씽킹이란.docx')
```

```
n = len(my_doc.paragraphs)                  # 단락의 개수
print(n)
print(my_doc.paragraphs[0].text)            # 특정 단락의 내용.
print(my_doc.paragraphs[11].text)           # 특정 단락의 내용.
print(my_doc.paragraphs[33].text)           # 특정 단락의 내용.
```

Word 컨트롤

docx 모듈 활용:

- Document > Paragraph > Run:

```
# 스타일이 바뀌면 새로운 run 시작.
m = len(my_doc.paragraphs[33].runs)        # 특정 단락의 run의 개수.
print(m)
```

```
# run에는 text, bold, italic, underline 등의 속성이 있다.
print(my_doc.paragraphs[33].runs[10].text)  # 특정 run의 text 내용.
```

Word 컨트롤

docx 모듈 활용:

- Document > Paragraph > Run:

```
# 문서 내용을 다 출력해 주는 함수를 정의한다.
def getFullText(filename):
    a_doc = docx.Document(filename)
    fullText = []
    for a_para in a_doc.paragraphs:
        fullText.append(a_para.text)
    return '□n'.join(fullText)
```

```
# 위에서 정의한 함수를 사용하여 문서를 출력해 본다.
print(getFullText('디자인 씽킹이란.docx'))
```

Word 컨트롤

docx 모듈 활용:

- Document > Paragraph > Run:

```
# 새로운 Word 문서 객체.
my_new_doc = docx.Document()
# Paragraph 추가.
my_new_doc.add_paragraph("My first paragraph!")
my_new_doc.add_paragraph("My second paragraph!")
my_new_doc.add_paragraph("My third paragraph!")
# File로 저장.
my_new_doc.save("my_new_doc.docx")
```

PDF 컨트롤

PyPDF2 모듈 활용:

- PDF 문서를 읽어온다:

```
import PyPDF2
# Read as Binary로 오픈!
my_doc1 = open('Galaxy S6 manual.pdf', 'rb')
my_doc2 = open('MSE - Jacob.pdf', 'rb')
```

```
# 읽기 전용 PDF 객체.
my_reader1 = PyPDF2.PdfFileReader(my_doc1)
my_reader2 = PyPDF2.PdfFileReader(my_doc2)
n1 = my_reader1.numPages                    # 페이지 수.
n2 = my_reader2.numPages                    # 페이지 수.
```

PDF 컨트롤

PyPDF2 모듈 활용:

- PDF 문서를 읽어온다:

```
my_page = my_reader1.getPage(17)          # 17 쪽.
print(my_page.extractText())              # 내용 추출 (어려울 수 있음).
```

PDF 컨트롤

PyPDF2 모듈 활용:

- 두개의 PDF 문서를 이어 붙이기:

```
# 쓰기 전용 PDF 객체.
my_writer = PyPDF2.PdfFileWriter()
# 두개의 PDF를 이어 붙인다.
for i in range(n1):
    a_page = my_reader1.getPage(i)
    my_writer.addPage(a_page)
for i in range(n2):
    a_page = my_reader2.getPage(i)
    my_writer.addPage(a_page)
```

PDF 컨트롤

PyPDF2 모듈 활용:

- 두개의 PDF 문서를 이어 붙이기:

```
# 외부 파일로 출력.
f = open('my_merged.pdf','wb')
my_writer.write(f)
f.close()
```

이메일

이메일 보내기:

- smtplib 모듈 활용:

```
# 필요한 패키지를 불러온다.
import smtplib                                  # Simple Mail Transfer Protocol.
from email.mime.text import MIMEText
```

```
# 접속.
my_conn = smtplib.SMTP('smtp-mail.outlook.com', port=587)    # Hotmail.
my_conn.ehlo()                                  # 접속.
my_conn.starttls()                              # TLS encryption 시작.
```

이메일

이메일 보내기:

- smtplib 모듈 활용:

```
my_msg = MIMEText("This is a test")                # 이메일 내용.
my_msg['Subject'] = "A test message"               # 이메일 제목.
```

```
my_conn.login("로그인 이메일", "패스워드")                # 이메일 주소로 ID 대신.
my_conn.sendmail("보내는 주소", "받는 주소", my_msg.as_string())
my_conn.quit()
```

이메일

이메일 읽기:

- 바이너리 데이터로 읽어온 이메일을 parsing 한다:

```
# 필요한 패키지를 불러온다.
import imapclient
import email
```

```
my_conn = imapclient.IMAPClient('imap-mail.outlook.com', ssl=True)   # Hotmail.
my_conn.login("로그인 이메일", "패스워드")
my_conn.select_folder('INBOX', readonly=True)
my_conn.list_folders()                                    # 현존 folder 리스트.
```

이메일

이메일 읽기:

- 바이너리 데이터로 읽어온 이메일을 parsing 한다:

```
UIDs = my_conn.search(['UNSEEN'])                    # UID = Unique ID.
rawMessage = my_conn.fetch([9318], ['BODY[]', 'FLAGS'])    # No parsing yet.
```

```
# 보낸사람과 제목 출력.
for a_UID, message_data in my_conn.fetch(UIDs, ['RFC822','BODY[]',
'FLAGS']).items():
    email_message = email.message_from_bytes(message_data[b'RFC822'])
    print(a_UID, email_message.get('From'), email_message.get('Subject'))
```

GUI : tkinter 모듈

tkinter 모듈의 특징:

- 파이썬의 표준 GUI 패키지.
- 캔버스 (canvas)를 제공하며 도형을 그릴 수 있다.
- 마우스 클릭, 키보드 누름 등의 행동에 따른 event-driven programming을 할 수 있다.

GUI : tkinter 모듈

레이블 객체를 통해서 Hello World 출력:

```
from tkinter import *
from tkinter import ttk
tk = Tk()
my_label = ttk.Label(tk, text = "Hello World!")
my_label.pack()
tk.mainloop()
```

GUI : tkinter 모듈

클릭할 수 있는 버튼을 보여줌:

```
from tkinter import *
from tkinter import ttk
tk = Tk()
my_button = ttk.Button(tk, text = "Click me")
my_button.pack()
tk.mainloop()
```

GUI : tkinter 모듈

버튼을 클릭할 때 마다 인사를 “프린트”해 준다:

```
from tkinter import *
from tkinter import ttk


def hello():
    print("Hello there!")


tk = Tk()
my_button = ttk.Button(tk, text = "Click me", command = hello)
my_button.pack()
tk.mainloop()
```

GUI : tkinter 모듈

버튼을 클릭할 때 마다 클릭횟수를 보여줌:

```
from tkinter import *
from tkinter import ttk
def Clicked():
    global click_count
    click_count += 1
    my_label.config(text="You clicked "+str(click_count)+" times!")
tk = Tk()
my_label = ttk.Label(tk, text = "This is a label!")
my_label.pack()
click_count = 0
my_button = ttk.Button(tk, text = "Click me!", command = Clicked)
my_button.pack()
tk.mainloop()
```

GUI : tkinter 모듈

캔버스를 만들어서 보여줌:

```
from tkinter import *


tk = Tk()
my_canvas = Canvas(tk, width = 500, height=500)
my_canvas.pack()
tk.mainloop()
```

GUI : tkinter 모듈

캔버스에 선을 그려줌:

```
from tkinter import *


tk = Tk()
my_canvas = Canvas(tk, width = 500, height=500)
my_canvas.pack()
my_canvas.create_line(0,0,500,500)
tk.mainloop()
```

GUI : tkinter 모듈

캔버스에 직사각형을 그려줌:

```
from tkinter import *


tk = Tk()
my_canvas = Canvas(tk, width = 500, height=500)
my_canvas.pack()
my_canvas.create_rectangle(10,10,50,50)
my_canvas.create_rectangle(50,50,130,130)
my_canvas.create_rectangle(130,130,290,290)
tk.mainloop()
```

GUI : tkinter 모듈

캔버스에 랜덤으로 직사각형을 그려줌:

```
from tkinter import *
import random
def random_rectangle(width, height, color):
    x1 = random.randrange(width)
    y1 = random.randrange(height)
    x2 = x1 + random.randrange(width)
    y2 = y1 + random.randrange(height)
    my_canvas.create_rectangle(x1, y1, x2, y2, fill=color)
tk = Tk()
my_canvas = Canvas(tk, width = 500, height=500)
my_canvas.pack()
for color in ['red', 'green', 'blue', 'orange', 'yellow', 'pink', 'purple', 'violet','magenta']:
    random_rectangle(200,200, color)
tk.mainloop()
```

GUI : tkinter 모듈

캔버스에 다각형을 그려줌:

```
from tkinter import *


tk = Tk()
my_canvas = Canvas(tk, width = 500, height=500)
my_canvas.pack()
my_canvas.create_polygon(10, 10, 100, 10, 100, 110, fill='red')
tk.mainloop()
```

GUI : tkinter 모듈

캔버스에 텍스트를 출력해 본다:

```
from tkinter import *
tk = Tk()
my_canvas = Canvas(tk, width = 500, height=500)
my_canvas.pack()
my_canvas.create_text(120, 50, text = 'Hello There!', font=('Courier',10), fill='red')
my_canvas.create_text(120, 100, text = 'Hello There!', font=('Times',20), fill='green')
my_canvas.create_text(150, 150, text = 'Hello There!', font=('Helvetica',30), fill='purple')
my_canvas.create_text(220, 220, text = 'Hello There!', font=('Courier',40), fill='orange')
tk.mainloop()
```

GUI : tkinter 모듈

캔버스에 기하학 객체를 그리고 움직여 본다:

```
from tkinter import *
import time
tk = Tk()
my_canvas = Canvas(tk, width = 500, height=500)
my_canvas.pack()
my_polygon = my_canvas.create_polygon(10, 110, 10, 160, 50, 135, fill ='green', outline = 'green')
my_circle = my_canvas.create_oval(460, 210, 490, 240, fill = 'red', outline='red')
for x in range(80):
    my_canvas.move(my_polygon, 5, 0)
    my_canvas.move(my_circle, -5, 0)
    tk.update()
    time.sleep(0.05)
tk.mainloop()
```

GUI : tkinter 모듈

캔버스에 공을 그리고 바운스:

```
from tkinter import *
import random
import time
tk = Tk()
my_canvas = Canvas(tk, width = 500, height=500)
my_canvas.pack()
x = 75
y = 100
size = 20
x_min = 10
x_max = 500 - size - 10
y_min = 10
y_max = 500 - size - 10
```

캔버스에 공을 그리고 바운스 (계속):

```
vx = +3
vy = +3
my_ball = my_canvas.create_oval(x, y, x + size, y + size, fill = 'orange',outline='red')
for i in range(2000):
    if ( (x < x_min) or ( x > x_max) ):
        vx = -vx + random.randint(-1,1)
    if ( (y < y_min) or (y > y_max) ):
        vy = -vy + random.randint(-1,1)
    x += vx
    y += vy
    my_canvas.move(my_ball, vx, vy)
    tk.update()
    time.sleep(0.02)
tk.mainloop()
```

SymPy 모듈 : 심볼 객체와 연산

일반 연산과 심볼 연산의 비교:

```
In[1] : x = 1                          # 변수 정의.
In[2] : x + x + 1                      # 수식 정의.
Out[2]: 3                              # 일반 연산의 결과.
```

```
In[1] : from sympy import Symbol       # sympy 모듈에서 Symbol 클래스를 가져옴.
In[2] : x = Symbol('x')                # 객체 생성.
In[3] : x + x + 1                      # 심볼 수식 정의.
Out[3]: 2*x + 1                        # 심볼 연산의 결과.
```

```
In[1] : from sympy import Symbol       # sympy 모듈에서 Symbol 클래스를 가져옴.
In[2] : a = Symbol('x')                # 객체 생성. 객체의 이름과 심볼 이름 다름.
In[3] : a + a + 1                      # 심볼 수식 정의.
Out[3]: 2*x + 1                        # 객체 이름과 심볼 이름 일치시키는 것 선호!
```

SymPy 모듈 : 심볼 객체와 연산

심볼 연산의 예:

```
In[1] : from sympy import symbols                    # sympy 모듈에서 symbols 클래스를 가져옴.
In[2] : x, y, z = symbols('x, y, z')                 # 여러 객체 동시에 생성.
In[3] : s = x*y + x*y + z                            # 심볼 수식 정의.
In[4] : s
Out[4]: 2*x*y + z                                    # 수식의 단순화.
```

```
In[5] : p = x*(x + x)                                # 간단한 수식 정의.
In[6] : p
Out[6]: 2*x**2                                       # 심볼 연산 실행.
```

```
In[7] : q = (x + 2)*(x + 3)                          # 수식 정의.
In[8] : q
Out[8]: (x + 2)*(x + 3)                              # 더 이상 단순화 할 수 없으므로 그대로 있음.
```

SymPy 모듈 : 심볼 객체와 연산

다항식 인수 분해와 전개:

```
In[1] : from sympy import symbols                    # sympy 모듈에서 symbols 클래스를 가져옴.
In[2] : from sympy import factor, expand             # factor 함수와 expand 함수를 가져옴.
In[3] : x, y = symbols('x, y')                       # 여러 객체 동시에 생성.
In[4] : expr = x**2 – y**2                           # 심볼 수식 정의.
In[5] : factor(expr)
Out[5]: (x – y)*(x + y)                              # 다항식 인수 분해.
```

```
In[6] : expr = x**3 +3*x**2*y + 3*x*y**2 + y**3      # 또 다른 심볼 수식 정의.
In[7] : factor(expr)
Out[7]: (x + y)**3                                   # 다항식 인수 분해.
```

```
In[6] : expr = (x + y)**2                            # 또 다른 심볼 수식 정의.
In[7] : expand(expr)
Out[7]: x**2 + 2*x*y + y**2                          # 다항식 전개.
```

SymPy 모듈 : 심볼 객체와 연산

프리티 프린트 (pretty print):

```
In[1] : from sympy import symbols                  # sympy 모듈에서 symbols 클래스를 가져옴.
In[2] : from sympy import pprint                   # pprint 함수를 가져옴.
In[3] : x, y = symbols('x, y')                     # 여러 객체 동시에 생성.
In[4] : expr = x**2 +2*x*y + y**2                  # 심볼 수식 정의.
In[5] : pprint(expr)
Out[5]: x**2 + 2·x·y + y**2                        # pretty print.
```

SymPy 모듈 : 심볼 객체와 연산

심볼 수식에 수치값 또는 다름 수식 대입:

```
In[1] : from sympy import symbols                  # sympy 모듈에서 symbols 클래스를 가져옴.
In[2] : x, y = symbols('x, y')                     # 여러 객체 동시에 생성.
In[3] : expr = x**2 + 2*x*y + y**2                 # 심볼 수식 정의.
In[4] : expr.subs({x:2, y:2})                      # 딕셔너리의 형태로 수치 값을 대입.
Out[4]: 16                                         # 결과.
```

```
In[5] : expr = x**2 - 2*x*y + y**2                 # 심볼 수식 정의.
In[6] : expr.subs({x:2 + y})                       # 딕셔너리의 형태로 다른 수식을 대입.
Out[6]: ( 2 + y)**2 - 2*(2 + y)*y + y**2           # 결과.
```

```
In[7] : from sympy import simplify                 # simplify 함수를 가져옴.
In[8] : expr2 = expr.subs({x:2 + y})               # 딕셔너리의 형태로 다른 수식을 대입.
In[9] : simplify(expr2)                            # 수식을 단순화 함.
Out[9]: 4                                          # 결과.
```

SymPy 모듈 : 심볼 객체와 연산

미지수 풀이:

```
In[1] : from sympy import Symbol, solve, factor.
In[2] : x = Symbol('x')
In[3] : expr = x – 5 - 7
In[4] : solve(expr)
Out[4]: [12]                                    # 결과는 list의 형식으로 출력.
```

```
In[5] : expr = x**2 – 4*x + 3
In[6] : solve(expr)
Out[6]: [1, 3]                                  # 결과는 list의 형식으로 출력.
```

```
In[7] : factor(expr)                            # 인수분해 해 봄.
Out[7]: (x – 3)*(x – 1)                         # 미지수 값을 쉽게 알아볼 수 있음.
```

SymPy 모듈 : 심볼 객체와 연산

심볼 수식 풀이:

```
In[1] : from sympy import symbols, solve, pprint
In[2] : from sympy import simplify
In[3] : x, a, b, c = symbols('x, a, b, c')
In[4] : expr = a*x*x + b*x + c
In[5] : solve(expr, x)
Out[5]: [(-b + sqrt(-4*a*c + b**2))/(2*a), -(b + sqrt(-4*a*c + b**2))/(2*a)]
```

```
In[6] : my_sols = solve(expr, x)
In[7] : simplify(expr.subs({x: my_sols[0]}))
Out[7]: 0
In[8] : simplify(expr.subs({x: my_sols[1]}))
Out[8]: 0
```

SymPy 모듈 : 심볼 객체와 연산

심볼 수식 풀이:

```
In[1] : from sympy import symbols, solve, pprint
In[2] : from sympy import simplify
In[3] : s, u, t, g = symbols('s, u, t, g')
In[4] : expr = u*t + (1/2)*g*t*t - s
In[5] : my_sols = solve(expr, t)
In[6] : my_sols
Out[6]: [(-u + sqrt(2.0*g*s + u**2))/g, -(u + sqrt(2.0*g*s + u**2))/g]
```

```
In[6] : pprint(my_sols[0])
In[7] : pprint(my_sols[1])
In[8] : simplify(expr.subs({t:my_sols[0]}))
Out[8]: 0
In[9] : simplify(expr.subs({t:my_sols[1]}))
Out[9]: 0
```

SymPy 모듈 : 심볼 객체와 연산

연립 선형 방정식 풀기:

```
In[1] : from sympy import symbols, pprint
In[2] : from sympy import simplify
In[3] : x, y = symbols('x, y')
In[4] : expr1 = 2*x + 3*y - 6
In[5] : expr2 = 3*x + 2*y – 12
In[6] : my_sols = solve((expr1, expr2))
In[7] : my_sols
Out[7]: {x: 24/5, y: -6/5}
```

```
In[8] : expr1.subs(my_sols)
Out[8]: 0
In[9] : expr2.subs(my_sols)
Out[9]: 0
```

SymPy 모듈 : 시각화

시각화:

```
In[1] : from sympy import Symbol, symbols, solve, factor, expand, simplify
In[2] : from sympy import init_printing
In[3] : from sympy.plotting import plot
In[4] : %matplotlib inline
In[5] : init_printing()                    # 항상 수식을 최적화된 방식으로 출력해 줌.
In[6] : y = 2*x -3
In[7] : plot(y, (x,-3,3), title='A line graph', xlabel='x', ylabel='y')
```