파이썬을 이용한 파일 입출력 다루기

2021.02.10

RESONE
Seo.Jeong Beom

Review: 객체지향언어와 절차지향언어

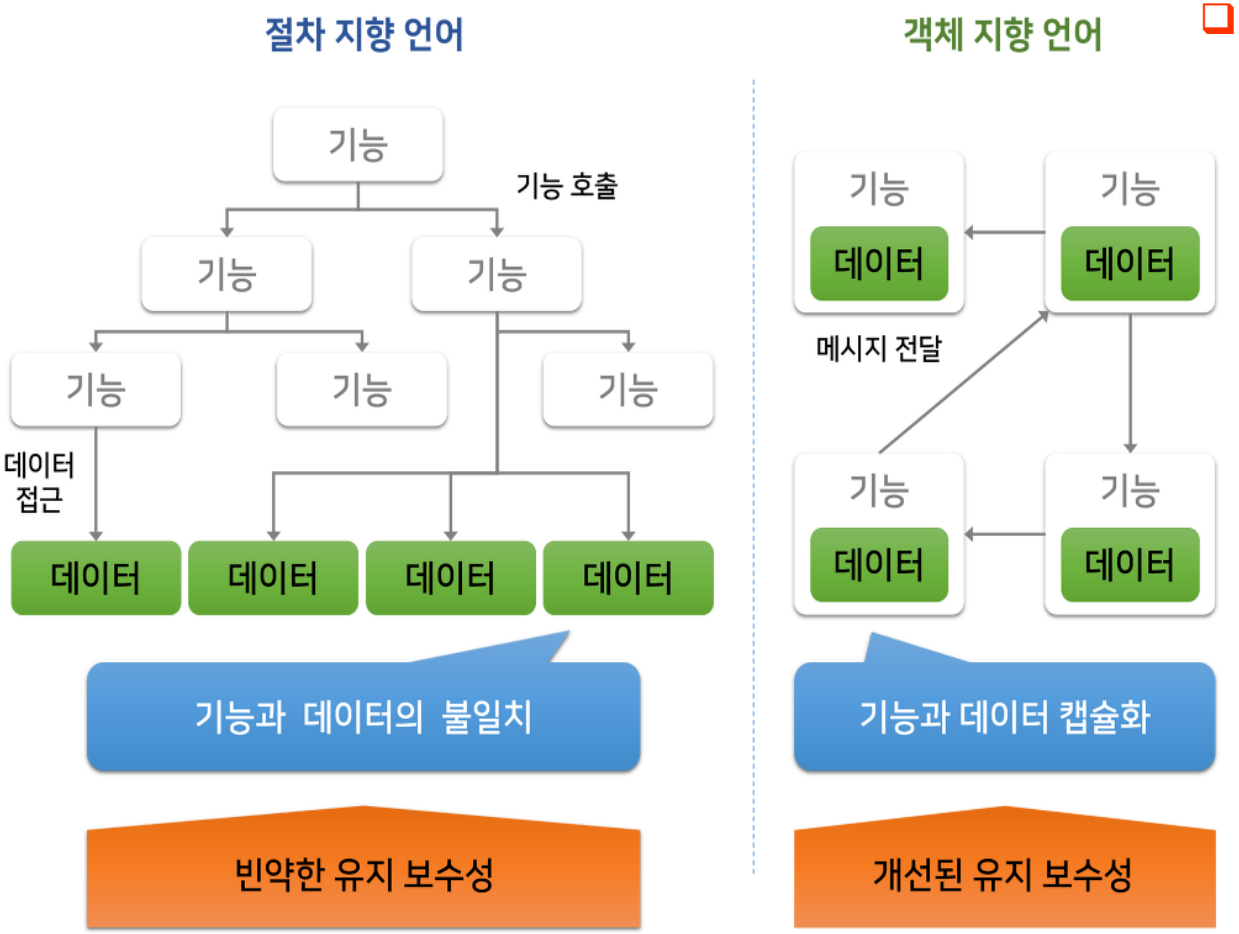


Fig 1. 언어의 특성에 따른 구조

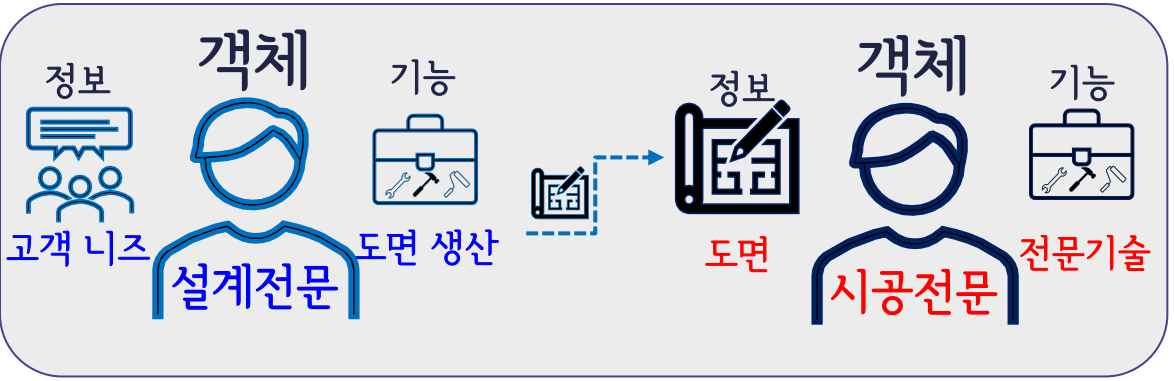
❑ 언어의 특성에 따른 구별

- 기능을 수행 하며, 필요한 서브루틴을 호출 (절차)
  - 서브루틴들은 또다른 서브루틴을 호출하며 서로서로 얽힌 관계를 가지게 됨.
  - 계산을 수행하는 데이터가 특정 기능에 귀속되지 않고, 독자적으로 존재하여 그 의미를 체계적으로 다루기 힘들다.
  - 기능과 데이터가 순서대로 연결되어 있어서, 유지보수시 그 복잡성이 도미노와 같이 커질 수 있다.
- 전문성 단위로 코드를 설계 (객체)
  - 특정한 기능 (method)을 수행하기 위한 데이터를 속성(데이터, attribute)으로 가지어서, 데이터별 의미를 체계적으로 관리 가능
  - 전문적인 기능을 수행하기 위한 속성을 가지는 **가상의 행동 단위를 객체라 한다**.
  - 객체들은 서로 독립적이며, 필요한 정보를 주고받으며 필요한 일을 수행하게 된다. 이는 유지보수가 편리한 장점이 있음.

Review: 객체지향언어와 절차지향언어

❑ 객체지향언어의 비유적 이해

- 객체의 의미
  - **객체** 란 특정한 일을 수행 할 수 있는 전문가이며, 필요한 **정보**(attribute)를 가지고 전문적인 **기능**(method)을 수행 할 수 있다.
  - 예를 들어, *건축시공 전문가*는 필요한 **정보** (도면)을 지참하며, 이를 이용하여 전문적인 **기능**(지반 다지기, 뼈대 세우기, 페인트 칠하기 등…)을 수행 할 수 있다.
  - 또한 *설계전문가*는 소비자의 니즈를 **정보**(니즈)로 하여 도면을 생산하는 **기술**(도면화)을 수행할 수 있다.
  - 즉 객체 지향적인 설계 란, 다루고자 하는 **큰 문제를 여러 전문적인 문제로 세분화**(설계, 시공)시키고, 각각을 수행하는 **객체를 설계하여**, 서로 상호 작용하며(**설계자가 도면을 생산하고, 시공전문가가 시공**) 전체 문제를 체계적으로 해결 하도록 코드를 설계하는 것




```
1  김아무개 = 설계전문(고객니즈) # 설계전문가 객체 생성
2  print(김아무개.니즈) # 입력 받은 고객의 니즈를 출력함
3
4  결과_도면 = 김아무개.도면화하기() # 도면화 메소드를 사용 하여 도면 만들기
5
6  이아무개 = 시공전문(결과_도면) # 시공전문가 객체 생성
7  print(이아무개.도면) # 전달 받은 도면을 출력함
8
9  이아무개.시공하기() # 시공하기 메소드를 이용하여 건물 시공
```

Review: 기본문법

❑ 객체지향 방식에서의 데이터 타입과 기본형 이해하기

- 파이썬은 실수와 정수 자료형의 형변환이 매우 자유롭다.
  - 파이썬은 자료형을 미리 선언하지 않아도 입력된 값에 따라서 자동으로 자료형을 선택하게 된다.
  - 또한 두 번째 사진에서 볼 수 있는 것 처럼 정수와 실수사이의 연산이 자연스럽게 수행되어 직관적이다.
  - 또한 파이썬은 우측과 같은 지수형 표현을 지원한다.
- 산술을 위한 연산자를 사용시 미리 정의된 방식으로 연산 된다.
  - 산술 연산자에는 사칙연산 (+,−,*,/) 등이 있으며, 제곱연산자 (**), 나머지 연산자 (%), 마지막으로 버림 연산자 (//) 등이 있다.
  - 이는 미리 정의된 방법으로 작동함

```
>>> a = -3.45
>>> b = 2.5
>>> a+b
-0.9500000000000002
```
(A)

```
>>> a =-3.45
>>> b = 6
>>> a+b
2.55
```
(B)

```
>>> 5/2
2.5
>>> 5//2
2
>>> 5*2
10
```
(C)

```
>>> 5**2
25
>>> 5-3
2
>>> 5%3
2
```
(D)

```
>>> a = 4.24E10
>>> b = -2.1E10
>>> c = a+b
>>> c
21400000000.0
```
(E)

```
>>> pprint(dir(a))
['__abs__',
 '__add__',
 '__bool__',
 '__class__',
 '__delattr__',
 '__dir__',
 '__divmod__',
 '__doc__',
 '__eq__',
 '__float__',
 '__floordiv__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__sizeof__',
 '__str__',
 '__sub__',
 '__subclasshook__',
 '__truediv__',
 '__trunc__',
 'as_integer_ratio',
 'conjugate',
 'fromhex',
 'hex',
 'imag',
 'is_integer',
 'real']
 '__getformat__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__int__',
 '__le__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__neg__',
 '__new__',
```

---(A,B,E)
---(D-3)
---(C-2)
---(C-1)
---(C-3)

Review: 기본문법

문자열 다루기

- 문자열은 문자의 나열을 의미하며 유용하고 다양한 메소드를 가지고 있다
  - 문자열을 전문으로 다루는 객체를 생성하고 다양하게 활용할 수 있다. (string.py)
  - 문자열 합치기 (Concatenate)
  - 문자열에서 추출하기 (indexing and slicing)
  - 문자열 쪼개기와 문자열 합치기 (split and join)
  - 대소문자 바꾸기 (upper and lower)
  - 문자열 교체하기 (replace)

```
# 문자열 concatenate
A = '\nHello, World!'
B = ' Welcome to Python!!\n'
# A+B = '\nHello, World! Welcome to Python!!\n'
print(A+B)
```

```
# 문자열 슬라이싱
C = 'abcdefg'
# --'0123456'
print(C[2],'==>c')
print(C[2:4],'==>cd')
print(C[2:6],'==>cdef')
print('\n')
```

Review: 기본문법

❑ 문자열 다루기

- 문자열은 문자의 나열을 의미하며 유용하고 다양한 메소드를 가지고 있다 *(2회 차 참조)*
  - 문자열을 전문으로 다루는 객체를 생성하고 다양하게 활용할 수 있다. (string.py)
  - 문자열 합치기 (Concatenate)
  - 문자열에서 추출하기 (indexing and slicing)
  - 문자열 쪼개기와 문자열 합치기 (split and join)
  - 대소문자 바꾸기 (upper and lower)
  - 문자열 교체하기 (replace)
  - 문자열 포메팅으로 원하는 규격의 문자열 출력하기

```
#문자열 split ; 특정기준으로 문자열 나누기
D = '0.125, 0.524, 0.166, 0.234, 0.747'
F = D.split(',')
print(F)
```

```
# 특정 문자를 기준으로 문자열 합하기
G = ','.join('abcde')#'a,b,c,d,e'
H = '\t'.join('abcde')#'a,b,c,d,e'
print(G)
print(H)
```

Review: 기본문법

❑ 리스트 자료형 다루기 *(list.py)*

- 동적 배열 리스트
  - 리스트는 값을 추가하거나 삭제할 수 있다.
  - 리스트는 요소에 자료형 제한이 없다.
  - 배열 내부의 값을 자유롭게 선택 (indexing) 하여 사용할 수 있으며, 일정 범위의 값들을 추출 (slice) 하여 사용
  - 인덱스는 첫 요소가 0 번 째 이며, 마지막 요소는 -1 번째로 접근할 수 있다.
  - 슬라이싱 *[시작 인덱스 : 끝 인덱스: 스텝 인덱스]*

```
A = [ 1, 2, 3, 4, 5, 6 ]
Index →  0, 1, 2, 3, 4, 5
```

```
A.append(8)
```

```
A = [ 1, 2,'a'3, 4, 5, 6 8]
```

```
A.insert(2,'a')
```

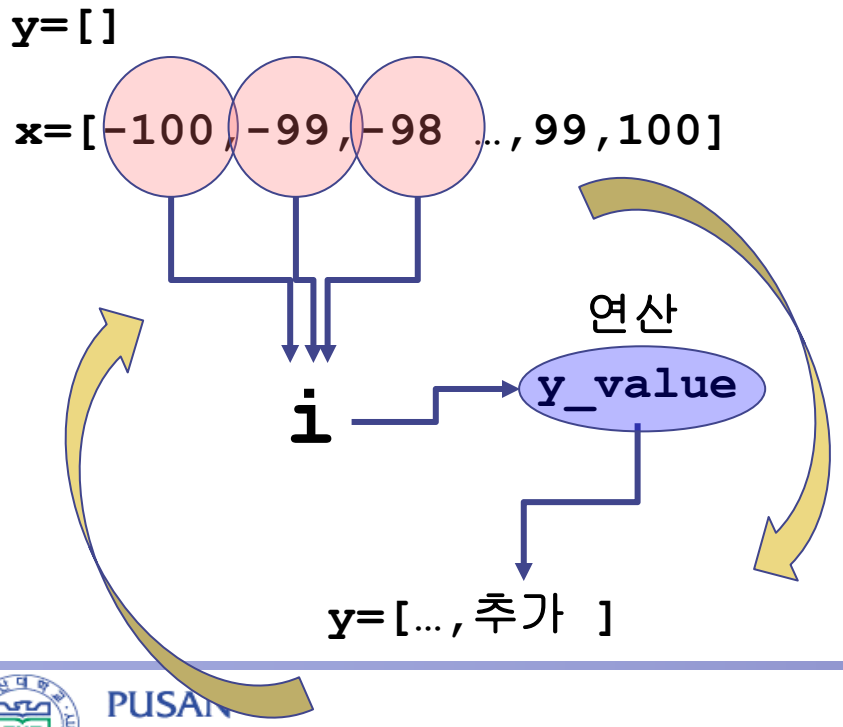
```
# 리스트 객체는 [] 속에 , 로 구별하여 생성
list = [1,2,3,4,5,6,7,8,9]
#------[0,1,2,3,4,5,6,7,8] 각 요소의 인덱스
list0= ['this','is','my',0,1,6,[1,2,3,4]]
#------[  0,    1,   2, 3,4,5,    6] 요소의 인덱스
# 리스트의 요소는 숫자가 아니여도 무방하다.
# 리스트 또한 리스트의 요소가 될 수 있다.
print(list0[2],'==> my')
print(list0[6],'==> [1,2,3,4]')
print(list0[6][2],'==> 3') #이중 인덱싱도 가능
```

```
# 리스트 슬라이싱
list1 = list[2:6] #== [3,4,5,6] 1 칸씩
list2 = list[2:8:3] #== [3,6] 3 칸씩
list3 = list[2::2] #== [3,5,7,9] 2 칸씩
list4 = list[-1::-1] #== [3,5,7,9] 끝부터 -1 칸씩
v = list[-1] #--> 9
v1 = list[-2]#--> 8
print(list1,list2,list3,list4)
```

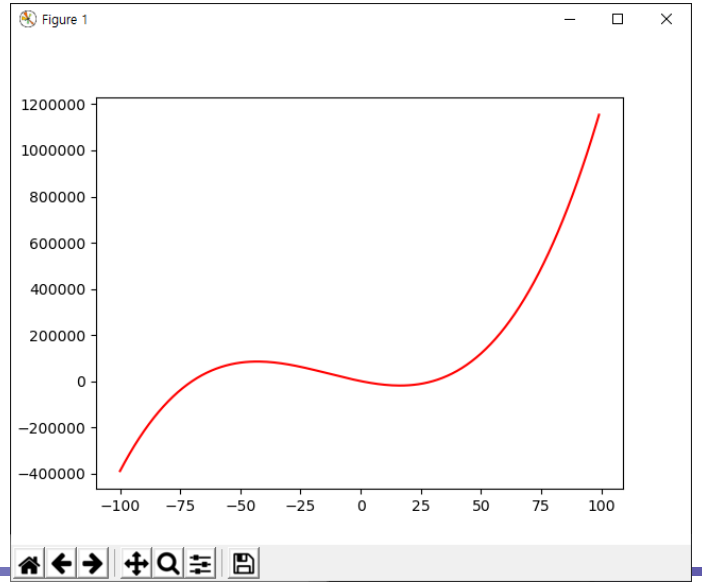
Review: 기본문법

❑ 반복문을 이용하여 매끈한 3차 함수 그리기 (polynomial.py)

- 파이썬 내장함수와 리스트 객체를 이용하여 다항함수를 그릴 수 있다.
  - 파이썬 range 함수는 일정한 간격을 가지는 배열을 생성할 수 있는 파이썬 내장함수이다.
    - **X = range(-100,101,1)** 의 구문은 -100 부터 100 까지의 구간에서 일정하게(1) 증가하는 값을 가지는 배열을 돌려준다.
    - **For i in X:** 구문은 배열의 값을 순서대로 하나씩 가져와 **i** 에 대입하여 사용한다.
    - 빈 배열 **y = []** 을 생성하고 **append**를 이용하여 배열의 끝에 다항식의 y 값을 하나씩 추가해 준다.




```
import matplotlib.pyplot as plt
x = range(-100,101,1)
y = []

for i in x:
    y_value= (i-30)*i*(i+70)
    y.append(y_value)

plt.plot(x,y,'r-')
plt.show()
```

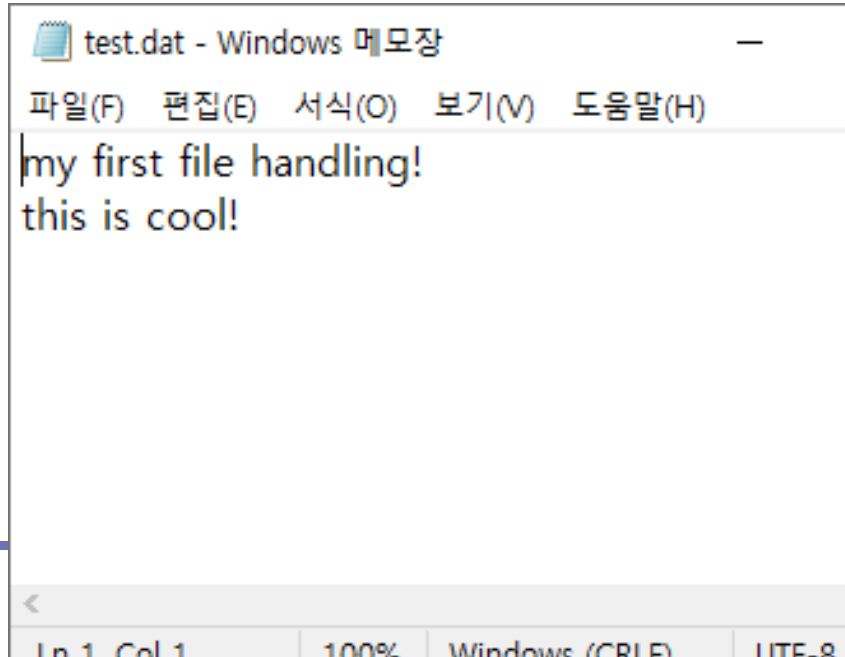


파이썬 기본 내장함수를 이용한 파일 다루기

파이썬 내장함수를 이용한 파일 다루기 [1] --- write.py

- 파이썬은 파일을 다루기 위한 기본 내장 모듈을 제공한다.
  - 프로그램에서 다루어진 여러가지 계산 결과 등은 프로그램이 종료될 때 함께 사라지는 휘발성 메모리 (RAM)에 저장되어 있으므로 지속적인 활용을 위해서는 비휘발성 메모리 (HARD)에 저장할 필요가 있다.
  - open(file-directory)는 파일을 전문적으로 다루기 위한 객체를 생성하고 다룬다.
    - 즉 파일을 읽고, 쓰고, 그리고 내용을 추가하는 등의 기능을 수행할 수 있는 객체이다.
    - (1) 은 현재 디렉토리에 test.dat 파일을 'w'모드로 다루어 해당 파일에(없으면 생성) 내용을 쓰기위한 객체를 생성한것.
      *→ (r,w,a ← 각각 **read, write, agument** 으로 단순 읽기인지, 혹은 내용을 지우고 값을 쓸것인지, 있는 내용을 이어 쓸것인지를 선택 할 수 있다.)*
    - ***Write** 함수는*
  - 파일 다루기가 모두 끝나면 close() (2) 를 수행하여 파일에 변화를 반영한다.

```
fileobj = open('./test.dat','w') --- (1)

fileobj.write('my first file handling!\n')

fileobj.write('this is cool!\n')

fileobj.close() ----------------------(2)
```

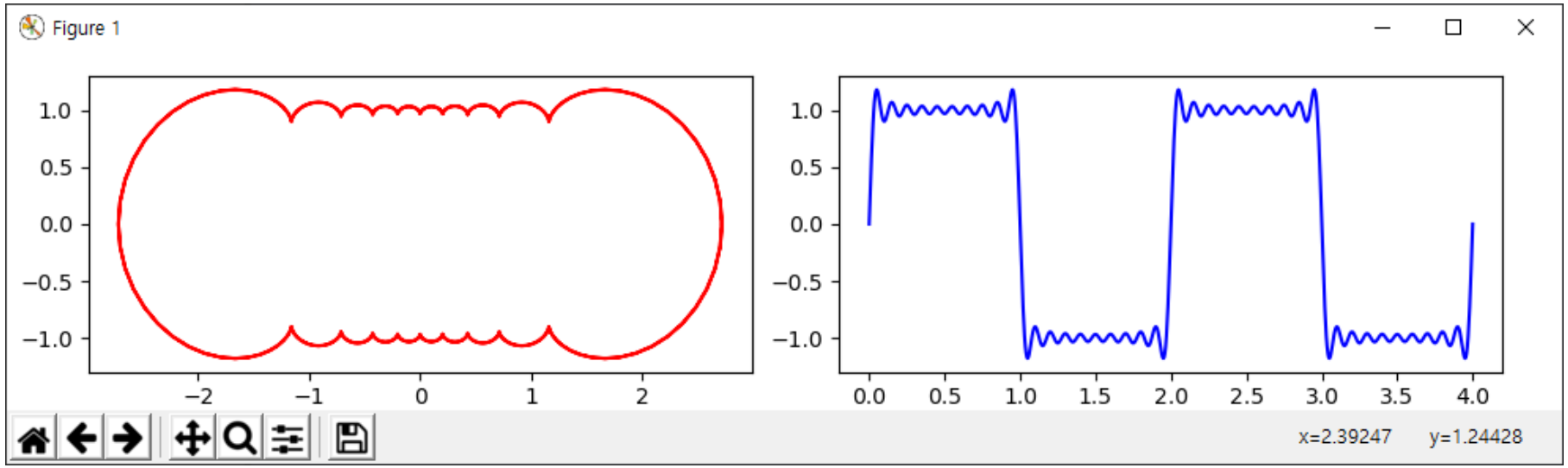


파이썬 기본 내장함수를 이용한 파일 다루기

파이썬 내장함수를 이용한 파일 다루기 (반복식과 함께 활용하기) write_array.py

- 간단한 삼각함수 계산결과를 파일로 출력하기.
  - 만약 특정한 값들을 일정하게 출력할 경우를 가정하여 반복식으로 배열 값을 출력하는 예제를 다룬다.
  - 사각파에 대한 푸리에 급수 결과를 예시로 사용하기 위해 아래의 코드와 같이 작성 (참조)
  - 만약 시간에 따른 급수의 값 (X,Y) 를 3개의 열 (T,X,Y)의 형태로 파일 작성한다면 ?

```
import math
import matplotlib.pyplot as plt
t = 0
X= []
Y= []
T =[]
n = range(1,21,2)
while t<=4:
    x = 0
    y = 0
    for N in [math.pi*i for i in n]:
        x += 4/N*math.cos(N*t)
        y += 4/N*math.sin(N*t)
    X.append(x)
    Y.append(y)
    T.append(t)
    t += 0.005

fig, axs = plt.subplots(1,2)
axs[0].plot(X,Y,'r-')
axs[1].plot(T,Y,'b-')
plt.show()
```



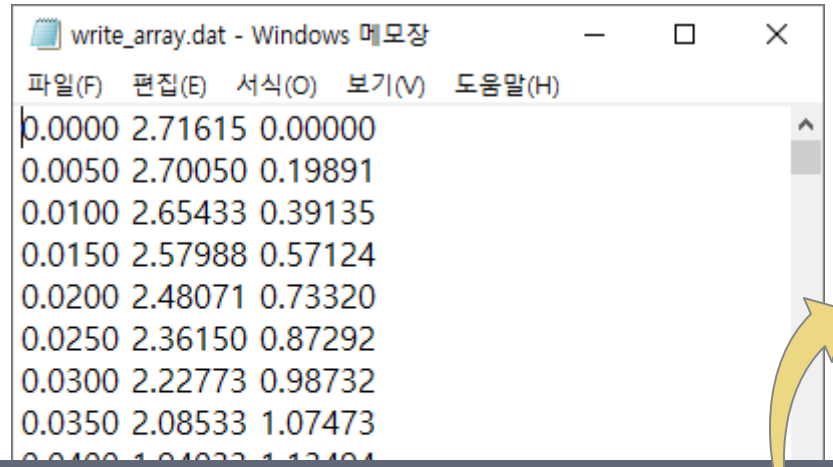
$$x_N(t) = \sum_{n=-N}^{N} c_n e^{jn\omega_0 t} = \sum_{\substack{n=-N \\ n \text{ odd}}}^{N} \frac{2}{jn\pi} e^{jn\omega_0 t}$$

$$= \sum_{\substack{n=1 \\ n \text{ odd}}}^{N} \frac{4}{n\pi} \sin(n\pi t)$$

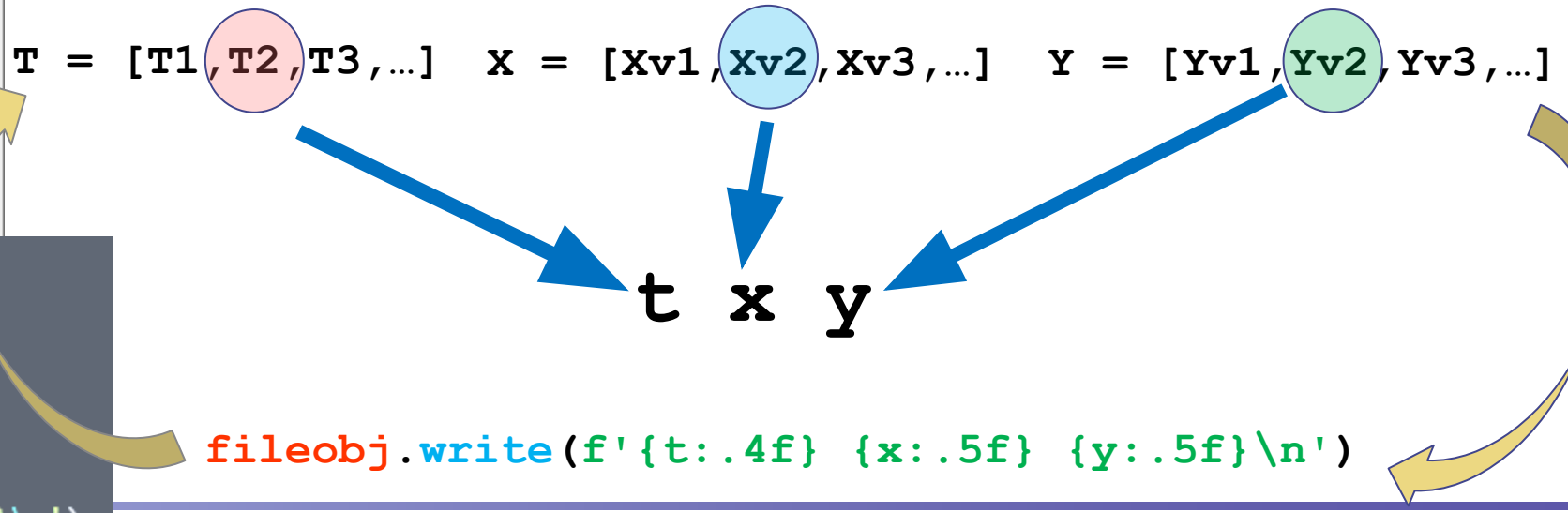
파이썬 기본 내장함수를 이용한 파일 다루기

❑ 파이썬 내장함수를 이용한 파일 다루기 (반복식과 함께 활용하기) write_array.py

- 간단한 삼각함수 계산결과를 파일로 출력하기.
  - 파이썬 내장함수 zip 을 이용하면 여러 개의 길이가 같은 배열을 묶어서 한 값 씩 순회 참조 할 수 있다.
  - 값들을 이용하여 원하는 유효숫자를 가지는 문자열을 만든다 (2장 포메팅 참조)
  - 이후 파일 다루기 객체의 write 함수를 이용하면 한 줄 씩 값을 파일에 써넣을 수 있다.
    - 현재 directory 에 write_array.dat 라는 파일을 write 모드로 열어서, write 함수를 이용하여 반복식내에서 한줄한줄 파일에 작성




```
# T = [T1, T2, .....]
# X = [Xv1, Xv2 , .....]
# Y = [Yv1, Yv2 , .....]
fileobj = open('./write_array.dat','w')
for t,x,y in zip(T,X,Y):
    fileobj.write(f'{t:.4f} {x:.5f} {y:.5f}\n')
```

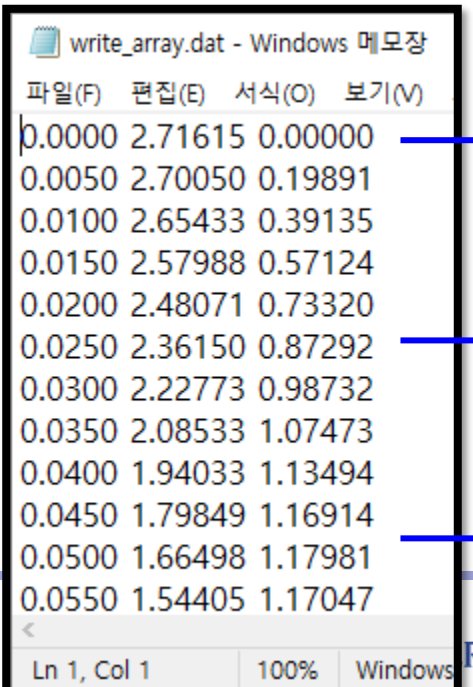


파이썬 기본 내장함수를 이용한 파일 다루기

파이썬 내장함수를 이용한 파일 다루기 (반복식과 함께 활용하기) read_array.py

- 파일 속 정보를 읽어서 파이썬 객체로 만들기 .
  - open 함수로 파일 객체를 생성하면 write 뿐만 아니라 읽기도 수행할 수 있다.
    - 현재 directory 에 write_array.dat 라는 파일을 read 모드로 열어서, readlines 함수를 이용하면 한 줄 당 한 개의 문자열 요소가 되도록 하는 리스트 객체를 반환한다.
  - 반복식을 이용하여 한 줄 씩 순회하며 문자를 쪼개고 (split) 문자에서 숫자로 형식 변환(형변환)해준다.
    - Split (2강 참조) , float함수는 숫자로 이루어진 문자를 실수형으로 변환하여 객체를 생성한다.
    - 각각의 실수 객체는 빈 리스트에 차곡차곡 append 하여 메모리에 저장한다.

readlines()




```
['0.0000 2.71615 0.00000',
 '0.0050 2.70050 0.19891',
 '0.0100 2.65433 0.39135',
 '0.0150 2.57988 0.57124',
 '0.0200 2.48071 0.73320',
 '0.0250 2.36150 0.87292',
 '0.0300 2.22773 0.98732',
 '0.0350 2.08533 1.07473',
 '0.0400 1.94033 1.13494',
 '0.0450 1.79849 1.16914',
 '0.0500 1.66498 1.17981',
 '0.0550 1.54405 1.17047']
```

split()

```
splited = one_line.split()
=['0.0250','2.36150','0.87292']
```

```
t =[t1,t2,…]
x =[x1,x2,…]   y =[y1,y2,…]
t.append(splited[0])
x.append(splited[1])
y.append(splited[2])
```

파이썬 기본 내장함수를 이용한 파일 다루기

❑ 파이썬 내장함수를 이용한 파일 다루기 (반복식과 함께 활용하기) read_array.py

- 파일 속 정보를 읽어서 파이썬 객체로 만들기 .
  - open 함수로 파일 객체를 생성하면 write 뿐만 아니라 읽기도 수행할 수 있다.
    - 현재 directory 에 write_array.dat 라는 파일을 read 모드로 열어서, readlines 함수를 이용하면 한 줄 당 한 개의 문자열 요소가 되도록 하는 리스트 객체를 반환한다.
  - 반복식을 이용하여 한 줄 씩 순회하며 문자를 쪼개고 (split) 문자에서 숫자로 형식 변환(형변환)해준다.
    - Split (2강 참조) , float함수는 숫자로 이루어진 문자를 실수형으로 변환하여 객체를 생성한다.
    - 각각의 실수 객체는 빈 리스트에 차곡차곡 append 하여 메모리에 저장한다.

```
f = open('./write_array.dat','r')
result = f.readlines()
f.close()
t = []
x = []
y = []
for line in result:
    splited = line.split()
    c1 = float(splited[0])
    c2 = float(splited[1])
    c3 = float(splited[2])
    t.append(c1)
    x.append(c2)
    y.append(c3)
```

```
['0.0000 2.71615 0.00000',
 '0.0050 2.70050 0.19891',
 '0.0100 2.65433 0.39135',
 '0.0150 2.57988 0.57124',
 '0.0200 2.48071 0.73320',
 '0.0250 2.36150 0.87292',
 '0.0300 2.22773 0.98732',
 '0.0350 2.08533 1.07473',
 '0.0400 1.94033 1.13494',
 '0.0450 1.79849 1.16914',
 '0.0500 1.66498 1.17981',
 '0.0550 1.54405 1.17047']
```

split()

```
splited = one_line.split()
=['0.0250','2.36150','0.87292']
```

```
t =[t1,t2,…]
x =[x1,x2,…]   y =[y1,y2,…]
t.append(splited[0])
x.append(splited[1])
y.append(splited[2])
```

파이썬 기본 내장함수를 이용한 파일 다루기

Pandas 를 이용하여 간편하게 파일 다루기 [2]

- Pandas는 배열, 행렬을 편리하게 다루기 위한 라이브러리이다.
  - 다차원 행렬 (m×n, m×n×l)에 대한 다양한 연산이나, 파일(text 기반 파일, excel 바이너리 파일)을 다루기 위한 다양한 함수와 모듈을 제공하는 패키지
  - 앞선 예제의 텍스트 파일을 읽어서 활용가능한 숫자 배열 객체로 만드는 예제를 통해 활용가치를 알아보자
- Pandas 파일 핸들링의 장점
  - Pandas 는 파일을 읽어 그 형태 그대로 행렬의 형태로 구성하여 효율적이다. (이를 pandas 에서는 dataframe 이라 한다.)
  - 행렬의 형태로 데이터가 정리되면 각각의 열과 행을 자유롭게 이용하는 함수를 활용할 수 있다.
  - 데이터를 출력할 때 정리된 표의 형태로 출력하기 때문에 그 결과값을 체크할 때 유용하다.

```
import pandas as pd

A = [{'x':0 , 'y':1},{'x':1 , 'y':6},{'x':6 , 'y':3}]
dataframe = pd.DataFrame(A)
print(dataframe)
```

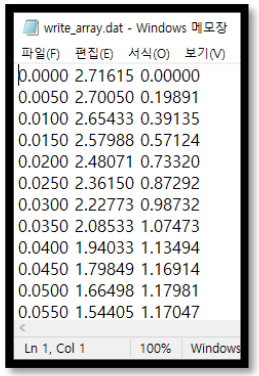
```
C:₩Users₩Jeongbeom Seo₩AppData₩Local₩Programs₩Python₩Python37₩python.exe
   x  y
0  0  1
1  1  6
2  6  3

Process returned 0 (0x0)        execution time : 0.984 s

계속하려면 아무 키나 누르십시오 . . .
```

Pandas 는 본래 구조를 지닌 데이터들을 효율적으로 다루기 위한 패키지 이기 때문에 데이터 입출력에서도 표의 형태로 데이터를 관리한다.

파이썬 기본 내장함수를 이용한 파일 다루기

Pandas 를 이용하여 간편하게 파일 다루기 [2]

- read_csv 함수를 이용한 파일 입력 (IO)
  - read_csv 함수는 특정 문자로 구별되어 있는 텍스트 기반 파일을 쉽게 읽고, 표의 형태로 접근 가능한 dataframe 객체를 생성한다.
    - 파일 디렉토리, 각각의 값을 구별하는 문자 (현재 ' ' 스페이스), header 유무 등 다양한 옵션을 지님
    - 기존에 비하여 아주 단순하고 편하게 데이터를 읽을 수 있음



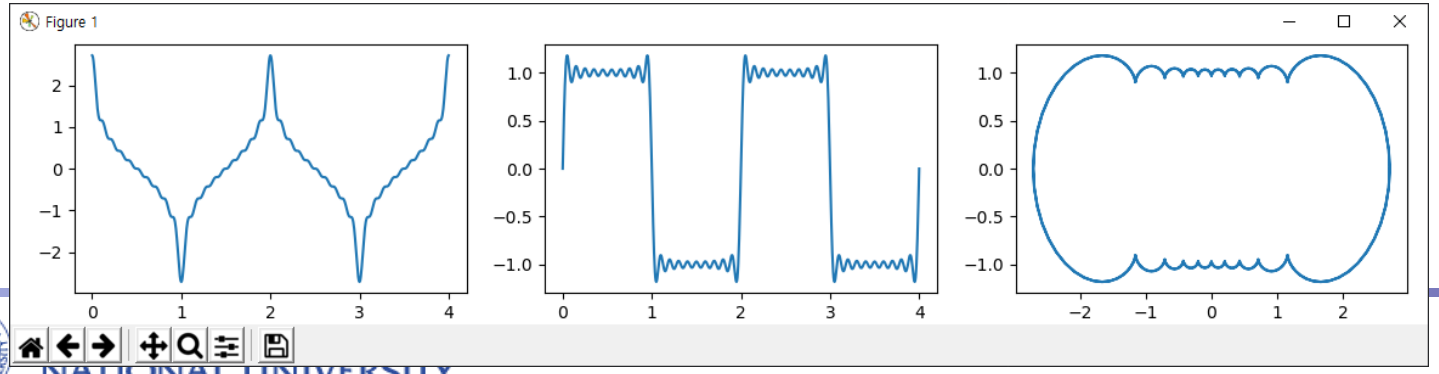

```
import pandas as pd
import matplotlib.pyplot as plt
data = pd.read_csv('./write_array.dat',sep=' ',header=None)
print(data)
```

data

	0	1	2
0	0.000	2.71615	0.00000
1	0.005	2.70050	0.19891
2	0.010	2.65433	0.39135
3	0.015	2.57988	0.57124
4	0.020	2.48071	0.73320
..	...	...	...
796	3.980	2.48071	-0.73320
797	3.985	2.57988	-0.57124
798	3.990	2.65433	-0.39135
799	3.995	2.70050	-0.19891
800	4.000	2.71615	-0.00000

[801 rows x 3 columns]

data[1]



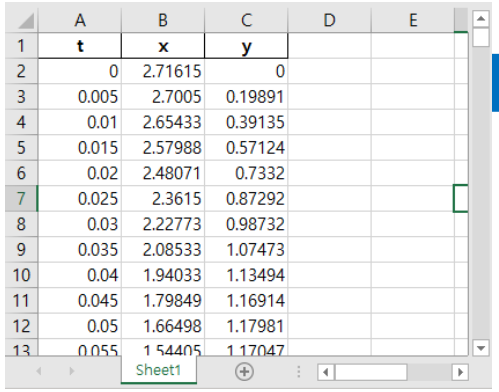

```
fig,axe = plt.subplots(1,3)
axe[0].plot(data[0],data[1])
axe[1].plot(data[0],data[2])
axe[2].plot(data[1],data[2])
plt.show()
```

파이썬 기본 내장함수를 이용한 파일 다루기

❑ Pandas 를 이용하여 간편하게 파일 다루기 [2]

- read_excel 함수를 이용한 파일 입력 (IO)
  - ◆ read_csv 함수는 특정 문자로 구별되어 있는 텍스트 기반 파일을 쉽게 읽고, 표의 형태로 접근 가능한 dataframe 객체를 생성한다.
    - 파일 디렉토리, 각각의 값을 구별하는 문자 (현재 ' ' 스페이스), header 유무 등 다양한 옵션을 지님
    - 기존에 비하여 아주 단순하고 편하게 데이터를 읽을 수 있음

write_array.xlsx



	A	B	C
1	t	x	y
2	0	2.71615	0
3	0.005	2.7005	0.19891
4	0.01	2.65433	0.39135
5	0.015	2.57988	0.57124
6	0.02	2.48071	0.7332
7	0.025	2.3615	0.87292
8	0.03	2.22773	0.98732
9	0.035	2.08533	1.07473
10	0.04	1.94033	1.13494
11	0.045	1.79849	1.16914
12	0.05	1.66498	1.17981
13	0.055	1.54405	1.17047

Sheet1

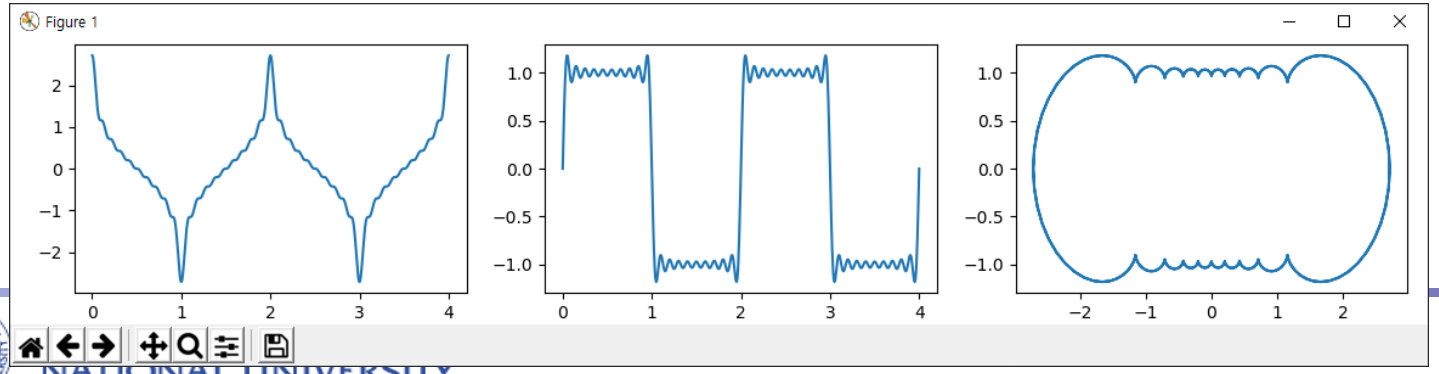
```
data = pd.read_excel('./write_array.xlsx')
```

data

	0	1	2
0	0.000	2.71615	0.00000
1	0.005	2.70050	0.19891
2	0.010	2.65433	0.39135
3	0.015	2.57988	0.57124
4	0.020	2.48071	0.73320
..	...	...	...
796	3.980	2.48071	-0.73320
797	3.985	2.57988	-0.57124
798	3.990	2.65433	-0.39135
799	3.995	2.70050	-0.19891
800	4.000	2.71615	-0.00000

[801 rows x 3 columns]

data[1]




```
fig,axe = plt.subplots(1,3)
axe[0].plot(data.t,data.x)
axe[1].plot(data.t,data.y)
axe[2].plot(data.x,data.y)
plt.show()
```

파이썬 기본 내장함수를 이용한 파일 다루기

Pandas 를 이용하여 간편하게 파일 다루기 [2]

- to_csv 함수를 이용한 파일 출력 (IO)
  - to_csv 함수는 dataframe 객체를 원하는 형식에 맞추어 텍스트 기반의 파일로 쉽게 출력 할 수 있다.
  - 기존의 x 와 y 를 각각 곱한값 (xy)을 하나의 열로 추가하여 데이터 구조를 변경하고 파일로 저장하는 예제를 살펴보자.
    - 파일 디렉토리, 각각의 값을 구별하는 문자 (현재 `` 스페이스), header 유무 등 다양한 옵션을 지님
    - 기존에 비하여 아주 단순하고 편하게 데이터를 파일로 출력 할 수 있음

data

	0	1	2
0	0.000	2.71615	0.00000
1	0.005	2.70050	0.19891
2	0.010	2.65433	0.39135
3	0.015	2.57988	0.57124
4	0.020	2.48071	0.73320
..	...	...	...
796	3.980	2.48071	-0.73320
797	3.985	2.57988	-0.57124
798	3.990	2.65433	-0.39135
799	3.995	2.70050	-0.19891
800	4.000	2.71615	-0.00000

[801 rows x 3 columns]

data[1]

Col_vector

DataFrame객체의 열 벡터 사이의 곱 연산자는 **자동으로 요소별 연산 (.*) 을 수행**하여 새로운 열 벡터를 생성하며,

data 객체 (행렬)에 열을 추가하는 과정은 매우 간단하다.

write_array_wPandas.dat - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
t x y xy
0.00000 2.71615 0.00000 0.00000
0.00500 2.70050 0.19891 0.53716
0.01000 2.65433 0.39135 1.03877
0.01500 2.57988 0.57124 1.47373
0.02000 2.48071 0.73320 1.81886
0.02500 2.36150 0.87292 2.06140
0.03000 2.22773 0.98732 2.19948
```

```
xy_col = data[1]*data[2]
data['xy'] = xy_col

data.to_csv('./write_array_wPandas.dat',\
index=False,sep=' ',float_format='%.5f',header=['t','x','y','xy'])
```

Pandas 를 이용하여 간편하게 파일 다루기 [2]

- to_excel 함수를 이용한 파일 출력 (IO)
  - to_excel 함수는 dataframe 객체를 원하는 형식에 맞추어 엑셀 파일로 쉽게 출력 할 수 있다.
  - Sep 옵션등은 엑셀 출력시에는 필요하지 않으며, 필요한 옵션은 판다스의 도큐멘트에서 쉽게 찾아볼 수 있다.

```
data.to_excel('./write_array_wPandas.xlsx',\
index=False,float_format='%.5f',header=['t','x','y','xy'])
```

	A	B	C	D	E	F	G	H	I	J
1	t	x	y	xy						
2	0	2.71615	0	0						
3	0.005	2.7005	0.19891	0.53716						
4	0.01	2.65433	0.39135	1.03877						
5	0.015	2.57988	0.57124	1.47373						
6	0.02	2.48071	0.7332	1.81886						
7	0.025	2.3615	0.87292	2.0614						
8	0.03	2.22773	0.98732	2.19948						
9	0.035	2.08533	1.07473	2.24117						
10	0.04	1.94033	1.13494	2.20216						
11	0.045	1.79849	1.16914	2.10269						

Sheet1

참고문헌

[1] Introducing Python（처음 시작하는 파이썬 ,빌 루바노빅)
[2] https://pandas.pydata.org/docs/user_guide/io.html

Thank you

감사합니다