201800294\_midterm

April 24, 2023

Copyright (C) 2023 201800294\_DongilKim All rights reserved *(https://KimTein.github.io)*

# Midterm

[ ]: **from IPython.core.interactiveshell import** InteractiveShell InteractiveShell.ast\_node\_interactivity = 'all'

# Problem. 1

[ ]:

*# initalizing*

func1

=

"

x

"

*# define function 1*

func2

=

"

2\*x\*\*2

"

*# define function 2*

func3

=

"

3\*x\*\*3

"

*# define function 3*

start

=

round

(

-

10

,

2

)

*# starting point*

end

=

round

(

10

,

2

)

*# ending point*

x

=

start

*# x intializing*

step

=

round

(

0.01

,

2

)

*# x step*

count

=

0

*# iteration counting num*

*# touch .txt file with file1, file2, file3, file4*

**with**

open

(

'

y=x.txt

'

,

'

w

'

)

**as**

file1, \

open

(

'

y=2x\*\*2.txt

'

,

'

w

'

)

**as**

file2, \

open

(

'

y=3x\*\*3.txt

'

,

'

w

'

)

**as**

file3, \

open

(

'

x data.txt

'

,

'

w

'

)

**as**

file4:

**while**

x

<

=

end:

*# while iteration*

file1

.

write(

str

(

round

(

eval

(

func1),

2

))

+

'

'

)

*# writing f1 file with*

␣

↪

*func1*

file2

.

write(

str

(

round

(

eval

(

func2),

2

))

+

'

'

)

*# wirting f2 file with*

␣

↪

*func2*

file3

.

write(

str

(

round

(

eval

(

func3),

2

))

+

'

'

)

*# writing f3 file with*

␣

↪

*func3*

file4

.

write(

str

(

round

(

x,

2

))

+

'

'

)

*# f4 : input x data*

count

+

=

1

x

+

=

step

*# x + 0.01*

*# new line*

**if**

count

%

5

==

0

:

file1

.

write(

'

**\n**

'

)

file2

.

write(

'

**\n**

'

)

file3

.

write(

'

**\n**

'

)

file4

.

write(

'

**\n**

'

)

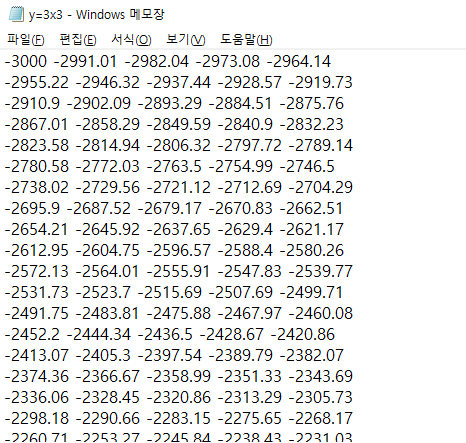
.

[결과창]









[알고리즘] :

1. 각각의 함수를 정의하고 시작 구간과 끝 구간을 정의한다. (이때 소수점 2자리로 끊었다.)
2. 반복횟수에 따른 줄바꿈을 넣기 위해 count변수 초기화한다.
3. x값는 step을 더하며 갱신되며 각 x에 해당되는 함수의 결과를 eval() 사용한다.
4. 반복문을 시행하며 매 시행 시 count값이 갱신되고 5의 배수가 될 때 마다 줄바꿈을 정의한다.

[예상 결과 및 실제 결과]:

결과창과 동일한 형태를 기대하였으며 주의할 점으로는 기계적인 오차로 인한 차이가 발생할 수 있다. 따라서 정확한 수치적인 값을 얻기 위해서는 원하는 조건이 필요로 하며 이를 위한 보완을 요구한다.

# 레포트 작성을 위해 파일을 옮기는 과정으로 실행 결과의 파일명을 간단히 작성함. 소스코드로 생성된 파일은 지장 없음.

# Problem.2

[ ]:

*# import data from txt file*

**with**

open

(

'

x data.txt

'

,

'

r

'

)

**as**

x\_data, \

open

(

'

y=x.txt

'

,

'

r

'

)

**as**

y\_data1, \

open

(

'

y=2x\*\*2.txt

'

,

'

r

'

)

**as**

y\_data2, \

open

(

'

y=3x\*\*3.txt

'

,

'

r

'

)

**as**

y\_data3, \

open

(

'

y=x + 2x\*\*2 + 3x\*\*3.txt

'

,

'

w

'

)

**as**

y\_data:

*# while iteration*

**while**

**True**

:

*# imported data to list*

x\_data\_box

=

x\_data

.

readline()

.

strip()

.

split()

*# data\_x to list type*

y\_list

=

[

1

]

\*

len

(

x\_data\_box

)

*# y\_list intializiing size with x\_data*

y\_data1\_list

=

list

(

map

(

float

, y\_data1

.

readline()

.

strip()

.

split()))

*#*

␣

↪

*y\_data1 to list*

y\_data2\_list

=

list

(

map

(

float

, y\_data2

.

readline()

.

strip()

.

split()))

*#*

␣

↪

*y\_data2 to list*

y\_data3\_list

=

list

(

map

(

float

, y\_data3

.

readline()

.

strip()

.

split()))

*#*

␣

↪

*y\_data3 to list*

*# while break*

**if**

**not**

x\_data\_box:

**break**

*# change y\_list to y\_data sum*

**for**

j

**in**

range

(

len

(

y\_list

)):

y\_list[j]

=

round

((

y\_data1\_list[j

]

+

y\_data2\_list[j]

+

␣

↪

y\_data3\_list[j]),

2

)

*# summation y\_data1, y\_data2, y\_data3*

y\_data

.

write(x\_data\_box[j]

+

'

'

+

str

(

y\_list[j

])

+

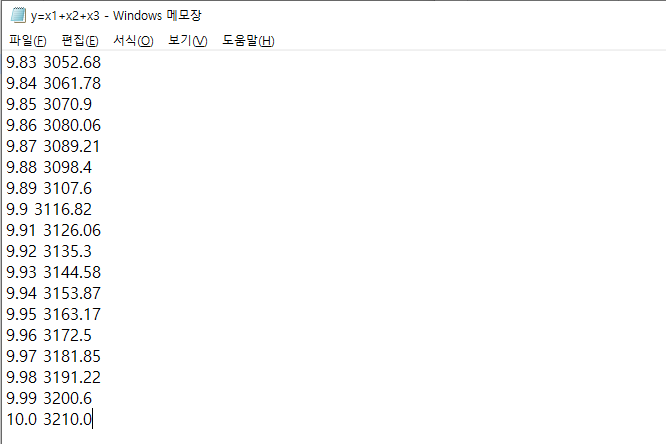
'

**\n**

'

)

[결과창]



[알고리즘]:

1. 앞서 각각의 x값과 해당되는 함수값을 출력한 파일을 읽어온다.
2. 최종 결과값을 위한 y\_data 파일을 출력하기로 한다.
3. 불러온 데이터들의 값만을 가져오기 위해 반복문을 통하여 한줄씩 읽어와 전처리를 한다. (readline(),strip(), split() 을 사용)
4. 한번의 반복문 시행 시 불러온 데이터들을 list형태로 변환하며 append해준다. 이때 우리가 원하는 y\_data의 리스트 크기는 입력값의 크기와 동일하게 맞춰준다.
5. 각 함수의 결과값들을 리스트 형태로 불러왔으므로 이를 y\_data 리스트에 summation하며 하나씩 append하여준다.
6. 이후, y\_list 의 각 인덱스 별로 입력값과 출력값을 txt파일로 export한다.

[예상 결과 및 실제 결과]:

결과창과 동일한 형태를 기대하였으며 다만, 소수점 자리를 맞추는 과정에 수치적인 오차가 쌓일 수 있다.

# Problem.3

[ ]:

*# define Max, Min*

**def**

getMax

(

numbers

):

result

=

-

100000000

**for**

number

**in**

numbers:

**if**

result

<

number:

result

=

number

**return**

result

**def**

getMin

(

numbers

):

result

=

100000000

**for**

number

**in**

numbers:

**if**

result

>

number:

result

=

number

**return**

result

*# make data*

f

=

"

-40\*x\*\*2+x\*\*4

"

*# define func*

start

=

round

(

-

10

,

2

)

*# starting point*

end

=

round

(

10

,

2

)

*# ending point*

x

=

start

*# initailizing x*

step

=

0.01

*# x step*

*# make data box*

y\_list

=

[]

x\_list

=

[]

val\_box

=

[]

*# fill x\_list, y\_list, val\_box*

**while**

x

<

=

(

end

):

x\_list

.

append(

round

(

x,

2

))

y\_list

.

append(

round

(

eval

(

f),

2

))

x

+

=

step

**for**

i

**in**

range

(

len

(

x\_list

)):

val\_box

.

append([x\_list[i], y\_list[i]])

*# decomposion value by interval*

*# interval (-1,1)*

x\_list\_1

=

x\_list[

901

:

1100

]

x\_list\_2

=

x\_list[

501

:

1500

]

x\_list\_3

=

x\_list[

1

:

-

1

]

*# interval (-5,5)*

y\_list\_1

=

y\_list[

901

:

1100

]

y\_list\_2

=

y\_list[

501

:

1500

]

y\_list\_3

=

y\_list[

1

:

-

1

]

*# interval (-10,10)*

val\_box\_1

=

val\_box[

901

:

1100

]

val\_box\_2

=

val\_box[

501

:

1500

]

val\_box\_3

=

val\_box[

1

:

-

1

]

*# get Max, Min value*

*# interval (-1,1)*

Max\_val\_1

=

getMax(y\_list\_1)

Min\_val\_1

=

getMin(y\_list\_1)

*# interval (-5,5)*

Max\_val\_2

=

getMax(y\_list\_2)

Min\_val\_2

=

getMin(y\_list\_2)

*# interval (-10,10)*

Max\_val\_3

=

getMax(y\_list\_3)

Min\_val\_3

=

getMin(y\_list\_3)

*# Make result list*

*# interval (-1,1)*

Max\_result\_1

=

[]

Min\_result\_1

=

[]

*# interval (-5,5)*

Max\_result\_2

=

[]

Min\_result\_2

=

[]

*# interval (-10,10)*

Max\_result\_3

=

[]

Min\_result\_3

=

[]

*# interval (-1,1)*

**for**

i

**in**

range

(

len

(

y\_list\_

1)):

**if**

y\_list\_1[i]

==

Max\_val\_1:

Max\_result\_1

.

append(val\_box\_1[i])

**elif**

y\_list\_1[i]

==

Min\_val\_1:

Min\_result\_1

.

append(val\_box\_1[i])

*# interval (-5,5)*

**for**

i

**in**

range

(

len

(

y\_list\_

2)):

**if**

y\_list\_2[i]

==

Max\_val\_2:

Max\_result\_2

.

append(val\_box\_2[i])

**elif**

y\_list\_2[i]

==

Min\_val\_2:

Min\_result\_2

.

append(val\_box\_2[i])

*# interval (-10,10)*

**for**

i

**in**

range

(

len

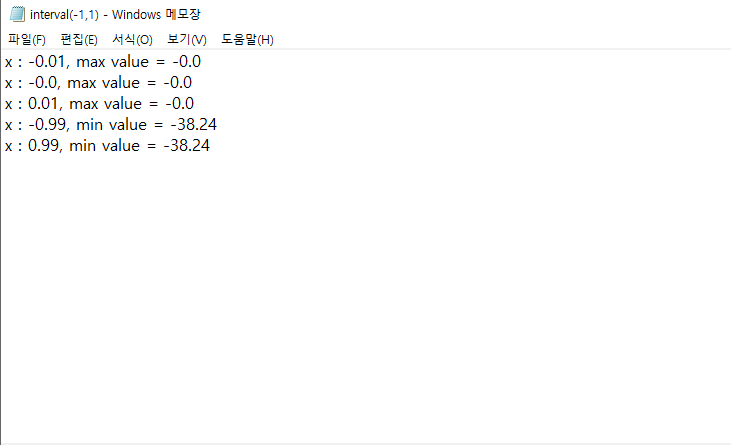
(

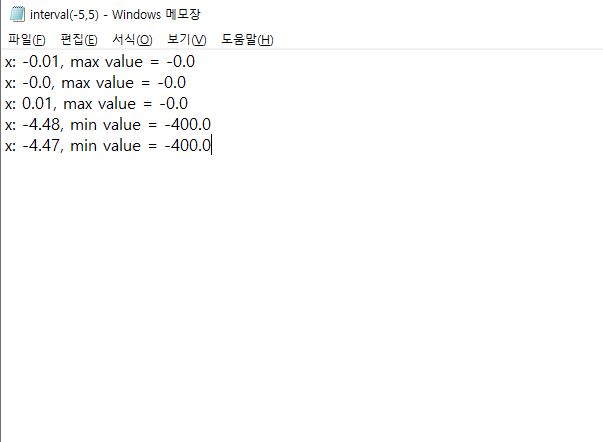
y\_list\_

3)):



[결과창]



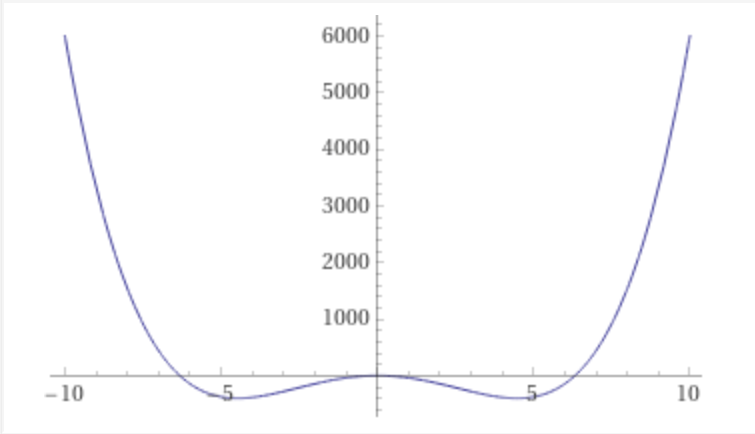




[알고리즘]:

1. Built-in 함수를 대신하기 위해 max, min 함수를 새롭게 정의한다.
2. 구간을 최대 범위인 -10 부터 10 까지 정의하며 각 변수를 초기화 한다.
3. 위 구간에 해당하는 함수 값들을 val\_box 안에 저장한다.
4. 문제에 제시된 구간별로 값들을 slicing하여 새롭게 정의한다.
5. 이후, 위에서 정의한 max, min함수를 이용하여 각 구간의 최대값과 최소값을 구한다.
6. 각 구간별로 위에서 구한 최대값과 최소값에 해당하는 인덱스를 추출하여 Result 리스트 안에 정의한다.
7. 이후, 제시된 형태로 txt파일로 출력한다.

[예상 결과 및 실제 결과]



문제에 제시된 함수를 plot하면 위와 같은 곡선이 그려진다.

각 최대값과 최소값에 해당하는 x값은 대략적으로 우리는 알 수 있다.

허나, 실제 결과에서는 -4.48, -4.47 과 같이 x값이 하나로 정의 되지 않고 여러 개가 나오는 것을 알 수 있다. 이는, 파이썬에서 내장된 기본 값들을 사용할 때 기계적인 오차가 발생하였기 때문이다. 결과창에서 값을 표기하기 위해 변수를 지정할 때 round()함수를 사용하였지만 수치적인 계산을 하는 과정 중에선 그렇지 않다. 따라서, 이를 보완하기 위해선 함수를 각 항별로 쪼갠 후 작은 값의 계산을 피하는 과정을 요한다.

Reference \* Title: Physics Programming Lecture Note (INU) \* Author: Jeongwoo Kim, Ph.D. \*

Availability: https://sites.google.com/view/jeongwookim

Copyright (C) 2023 201800294\_DongilKim All rights reserved *(https://KimTein.github.io)*