

AMSE318 HW1 WITH PYTHON

49004547 Seokjee SHIN

```
1 #HW1-1
```

2

f 3 #Add 0.00001 million times, showing the result every 100,000th step

4 n=0

5 step=0

6

```
7 while step<=1000000:
```

- 8 n+=0.00001
- 9 step += 1
- 10 if step%100000 == 0:
- 11 print(n)

Code evaluation 🗸

- 0.9999999999980838
- 2.00000000004635
- 3.000000000011186
- 4.000000000017737
- 4.9999999999979879
- 5.9999999999420215
- 6.999999999904164
- 7.999999999866306
- 8.999999999828448
- 9.99999999979059

,

1 #HW1-1

2

3 #Add 1, milion times, showing the result /100000 every 100,000th step

4 n=0

5 step=0

- 6 while step<1000000:
- 7 n+=1
- 8 step+=1
- 9 if step%100000 == 0:
- 10 print(n/100000)

Code evaluation 🗸

1.0			
2.0			
3.0			
4.0			
5.0			
6.0			
7.0			
8.0			
9.0			
10.0			

 \bigvee

 \mathbf{N}

CONCLUSION OF HW1-1

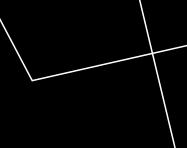
- Although, numerically both exercises should get 10 as an answer, the first exercise was not able to get the correct answer due to the calculation error. When the program calculates very small number, due to its limitation from transition to binary code (0 and 1), there's calculation error.
- Unlike the first exercise, the second exercise was able to get the correct answer as it divided the number by 100 000 every 100 000th times. This way, the results can be much accurate with integer number result.

4

```
#HW1-2a
import numpy as np
from math import *
double_pi = pi
single_pi = np.float32(pi)
print(double_pi)
print('Number of digits of pi:',len(str(double_pi)))
print(single_pi)
print('Number of digits of pi:',len(str(single_pi)))
```

```
error = abs((double_pi-single_pi)/double_pi)*100
print('Error%:',error)
```

3.141592653589793 Number of digits of pi: 17 3.1415927 Number of digits of pi: 9 Error%: 2.782753519183795e-06



<pre>#HW1-2 import numpy as np #32-bit n=1 count=0 for x in range(64): count+=1 n = np.float32((n+2)/2) print(count,',', n) if n == 2: print('mantissa = ',co break #gap print() #64-bit n=int(1) count=0 for x in range(64): count+=1 n = (n+2)/2 print(count,',', n) if n == 2: print('mantissa = ',co break</pre>	unt -1)	<pre>1 , 1.5 2 , 1.75 3 , 1.875 4 , 1.9375 5 , 1.96875 6 , 1.984375 7 , 1.9921875 8 , 1.9960938 9 , 1.9980469 10 , 1.9990234 11 , 1.9997559 13 , 1.9998779 14 , 1.999939 15 , 1.9999695 16 , 1.9999847 17 , 1.9999962 19 , 1.9999981 20 , 1.999999 21 , 1.999999 22 , 1.9999998 23 , 1.9999999 24 , 2.0 mantissa = 23</pre>	1 , 1.5 2 , 1.75 3 , 1.875 4 , 1.9375 5 , 1.96875 6 , 1.984375 7 , 1.9921875 8 , 1.99609375 9 , 1.998046875 10 , 1.9990234375 11 , 1.99951171875 12 , 1.999755859375 13 , 1.9998779296875 14 , 1.9999847412109375 17 , 1.99999482421875 16 , 1.9999847412109375 17 , 1.9999923706054688 18 , 1.999990463256836 21 , 1.9999990463256836 21 , 1.9999990463256836 21 , 1.9999990463256836 21 , 1.999999761581421 23 , 1.999999761581421 23 , 1.9999999701976776 26 , 1.99999998807907104 24 , 1.9999999850988388 27 , 1.99999999701976776 26 , 1.9999999952494194 28 , 1.9999999925494194 28 , 1.9999999952494194 28 , 1.999999995343387 32 , 1.9999999995343387 32 , 1.9999999997671694 33 , 1.99999999997671694 33 , 1.99999999997671694 33 , 1.9999999999708962 36 , 1.99999999999708962 36 , 1.999999999997724 38 , 1.9999999999998181	<pre>40 , 1.99999999999999005 41 , 1.9999999999995453 42 , 1.9999999999998863 44 , 1.9999999999999432 45 , 1.9999999999999716 46 , 1.99999999999993 48 , 1.99999999999993 48 , 1.99999999999999 50 , 1.9999999999999 51 , 1.9999999999999 52 , 1.9999999999999 53 , 2.0 mantissa = 52 </pre>
20XX	PRESENTATION TITLE	6	39, 1.9999999999998181	

CONCLUSION OF HW1-2

- In my computer, pi as an example, a single precision can produce 9digits accuracy whereas double precision can produce 17-digits accuracy.
- The number system of computer is:
 - 32-bit: 23 mantissa
 - 64-bit: 52 mantissa