Due date : 27th October Point: 26 points CEC 470

Name: Jeremiah Webb

ID: 2545328

HW 4: Floating Point Numbers

[6 points] Task 1:

Write the binary IEEE 32 bit representation for the following numbers:

a) +0
0 00000000 00000000000000000000000
b) -0
1 00000000 00000000000000000000000

c) Inf
0 11111111 00000000000000000000000

d) -inf
1 11111111 00000000000000000000000

e) Denormal

0 00000000 11000000000000000000000
f) NaN

0 11111111 00000000000000000000000

[10 points] Task 2:

1. Convert 0x7237176E from hexadecimal to IEEE 32-bit Format. (0x means the number is given in hex)
Convert to binary
0111 0010 0011 0111 0001 0111 0110 1110
7 2 3 7 1 7 6 E
Normalize:
1.1100100011011100010111011101110 * 2^ 30
Exponent: 30+127 = 157 = 10011101
Final Answer: 0 10011101 11001000110111000101111

2. Convert 8.625 from Decimal to IEEE 754 32- bit Floating Point format
   Convert to binary:
   1000.101
   Normalize: 1.000101 * 2^3
   Find exponent: 3+127 = 130 = 10000010
   Final Answer: 0 10000010 00010100000000000000000

[5 points] Task 3: Precision

As you should know, floating points are approximation of real numbers. Not every real number can be represented. Define three numbers in double precision in python:

A = 5555555555555555555555252352352352555 5.1305197613051976;

B = -5555555555555555555552523525235255555 55.1305197613051976;

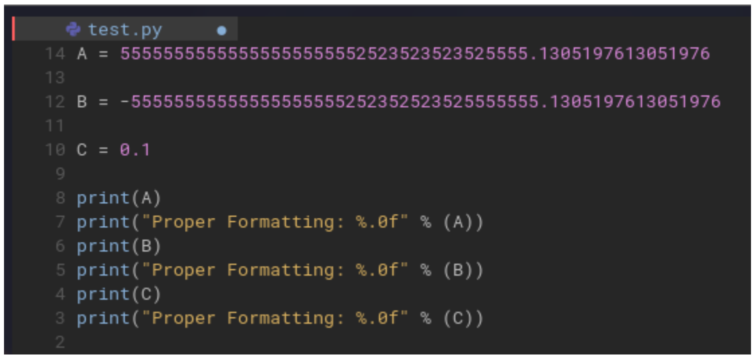
C=0.1;

And now print them on the screen. Are the numbers you have printed the one you have defined? If not why? Submit the screenshot aswell. Refer the supplemental materials below.

The numbers are not the same, and this is due to floating point error. Python actually uses the dtoa algorithm to accurately calculate and mitigate floating point error. Additionally Python's print statement does not accurately present a float number without specific formatting, as demonstrated below. Since the floating point "Double" type number has to be converted to an ASCII string for stdout, there is some data lost.

Link for CPython code showing dtoa algorithm: link

(Python is dynamically typed, didn't need to define data types).



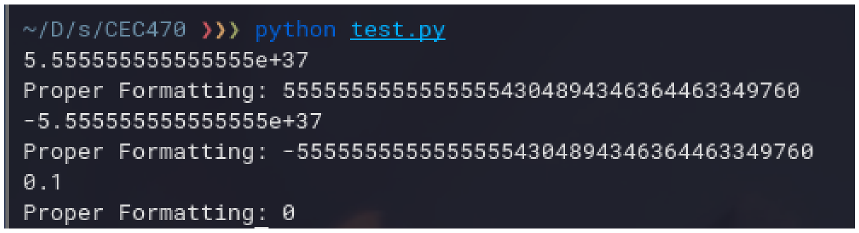
```
test.py
A = 55555555555555555555552523523523525555.1305197613051976

B = -555555555555555555555525235252352555555.1305197613051976

C = 0.1

print(A)
print("Proper Formatting: %.0f" % (A))
print(B)
print("Proper Formatting: %.0f" % (B))
print(C)
print("Proper Formatting: %.0f" % (C))

```

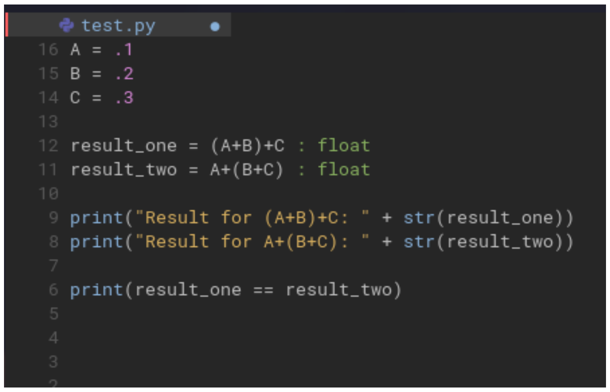


```
~/D/s/CEC470 ❯❯❯ python test.py
5.555555555555555e+37
Proper Formatting: 55555555555555554304894346364463349760
-5.555555555555555e+37
Proper Formatting: -55555555555555554304894346364463349760
0.1
Proper Formatting: 0
```

[5 points] Task 4: Associative property

It should be pretty clear now that floating point numbers and real numbers are two different sets of numbers. You have studied that addition is associative for real numbers. This means that if A,B and C are real number, than (A+B)+C = A+(B+C). Well, this is NOT true for floating point.

Write a short python program to prove that addition is not associative for floating point.



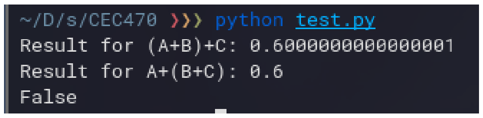
```
test.py
A = .1
B = .2
C = .3

result_one = (A+B)+C : float
result_two = A+(B+C) : float

print("Result for (A+B)+C: " + str(result_one))
print("Result for A+(B+C): " + str(result_two))

print(result_one == result_two)
```

This is neovim btw, I highly suggest you check it out.



```
~/D/s/CEC470 ❯❯❯ python test.py
Result for (A+B)+C: 0.6000000000000001
Result for A+(B+C): 0.6
False
```

[Bonus 2 points] Task 5: Watch the video https://www.youtube.com/watch?v=PZRI1IfStY0 and summarize the main points covered.

Notes:

- Please put all code into one file along with all the screenshots

Supplemental materials:

1. https://www.h-schmidt.net/FloatConverter/IEEE754.html
2. https://dwayneneed.github.io/.net/2010/05/06/fun-with-floating-point.html
3. https://www.youtube.com/watch?v=PZRI1IfStY0
4. https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html