Project 1

Execute the following lines in a Python interpreter.

```
x = 9.4
y = x - 9
z = y - 0.4
print(z)
```

- What did you get for z? What should it be in exact arithmetic? Why is it not what it should be?
- See Sauer's Numerical Analysis Chapter 0.3.3 for a complete description of the rounding phenomenon.

In [1]:

```python
#Floating point arithmetic:
x = 9.4
print(x)
y = x - 9
print(y)
z = y - 0.4
print(z)

#hmm but in exact arithmetic this is 0. Let us investigate
```

```
9.4
0.40000000000000036
3.3306690738754696e-16
```

In [2]:

```python
#Lets try changing the order
# hm now this showing exactly 0, which is what we expect
a = 9.4
print(a)
b = a - 0.4
print(b)
c = b - 9
print(c)
```

```
9.4
9.0
0.0
```

- From Sauer's textbook and Paul's lecture 1.2 and previous classes, I know that computers have a finite precision memory, however they must store infinite precision numbers and hence must handle rounding in some way
- IEEE sets standards for "that establish specifications and procedures designed to maximize the reliability of the materials, products, methods, and/or services people use every day". This includes a majority of computers, storage devices, and software platforms, including python

- 9.4 represented by the IEE double precision floating point standard is

$$\boxed{1.0010110011001100110011001100110011001100110011001100}1100\ldots \times 2^3$$

thus the 53rd bit is a 1 so the IEE rounding to nearest says to round up, or add a 1 to bit 52

  - We are discarding the infinite tail which means we subtract

  $$.1100 \times 2^{-52}\text{\textbackslash time}2^3 = 0.4 \times 2^{-48}$$

  and then round up, adding

  $$2^{-52} \times 2^3 = 2^{-49}$$

  - Thus we have it that 9.4 is represented in the computer by

  $$fl(9.4) = 9.4 - 0.4 \times 2^{-48} + 2^{-49}$$
  $$= 9.4 + 2^{-49} - 0.4 \times 2^{-48}$$

- Which means that the rounding error when storing 9.4 is

$$= 0.2 \times 2^{-49}$$

In [3]:

```
roundingError= 0.2*2**-49 # this is the rounding error when we store 9.4
print(roundingError)
print(9.4-9-roundingError-0.4)
```

```
3.552713678800501e-16
0.0
```

- Likewise, 0.4 represented by the IEE double precision floating point standard is

$$\boxed{1.1001100110011001100110011001100110011001100110011001}1001\ldots \times 2^{-2}$$

thus the 53rd bit is a 1 so the IEE rounding to nearest says to round up, or add a 1 to bit 52

- So fl(0.4) becomes

$$\boxed{1.1001100110011001100110011001100110011001100110011010} \times 2^{-2}$$

  - Thus, we are discarding the infinite tail which means we subtract

  $$2^{-53} \times 2^{-2} + .0\bar{1}10 \times 2^{-54} \times 2^{-2} = 0.4 \times 2^{-48}$$

  and then round up, adding

  $$2^{-52} \times 2^{-}2 = 2^{-54}$$

  - Thus we have it that 0.4 is represented in the computer by

  $$fl(0.4) = 0.4 - 2^{-55} - 0.4 \times 2^{-56} + 2^{-54}$$
  $$= 0.4 + 0.1 \times 2^{-52}$$

- Which means that the rounding error when storing 0.4 is

$$= 0.1 \times 2^{-52}$$

[Aside: Note I ended up coping this directly from the textbook after I first (stubbornly) tried to do this by hand without referencing the textbook to "test myself". Then I made the even dumber naieve decision not to check if I was correct, and ended up doing the rounding for incorrectly and I carried the error way to far through and wasted time. Learning point: just use the resources early and often]

In [4]:
```
roundingError2= 0.1*2**-52# this is the rounding error when we store 9.4
print(roundingError2)
print(roundingError+roundingError2)

print(9.4-9-roundingError)
print(roundingError+9.4-9-.4-roundingError2)

0.00000000000000036
```

```
2.2204460492503132e-17
3.7747582837255326e-16
0.4
3.108624468950438e-16
3.6e-16
```
Out[4]:

- Thus it is easy to see how operations between floating points can quickly exasperate the error

In [5]:
```
#I wanted to see if this is the rounding errors for 9.4 and 0.4 is equal to 3.330669073
totalRoundingError=(roundingError+roundingError2)

print("Python reported 9.4-9-0.4 was:  {}". format(z))
print("The rounding error for 9.4 and 0.4 is:  {}". format(totalRoundingError))

#Clearly this is not the case, and I believe this is because there are rounding errors

# I really wanted to see if there was some way around this, and obviously there is not,
```

```
Python reported 9.4-9-0.4 was:  3.3306690738754696e-16
The rounding error for 9.4 and 0.4 is:  3.7747582837255326e-16
```

Misc experiments

- Explored a bunch of resources and experimented with ALOT of ways to represent 9.4, 9, and 0.4 because I really wanted to see if there were other ways to do `9.4-9-0.4` in that order and get 0 and obviously there is not, I cannot trick the computer to store infinite numbers, but regardless here are some cool ways to play with numbers

In [6]:
```
# From source: https://docs.python.org/3/tutorial/floatingpoint.html
# Python has some methods to find the exact value of a float.
# The float.as_integer_ratio() method expresses the value of a float as a fraction:
```

```
dec=9.4
rat= dec.as_integer_ratio()
print("The float 9.4 is printed as {} ".format(dec))
print("Which in ratio form is {}".format(rat))

#convert the ratio to decimal
rat_to_dec= 5291729562160333/562949953421312
print(rat_to_dec) #appears to be 9.4, but lets check if they are actually stored equiva
print(rat_to_dec== 9.4) ##wow they are
```

```
The float 9.4 is printed as 9.4
Which in ratio form is (5291729562160333, 562949953421312)
9.4
True
```

In [7]:

```
#Really interesting source: http://stupidpythonideas.blogspot.com/2015/01/ieee-floats-a

import bitstring

f = 9.4
b = bitstring.pack('>d', f)
sbit, wbits, pbits = b[:1], b[1:12], b[12:]
sign= sbit.bin
print("Sign bit is: " , sign) #Sign bit 0 means positive, 1 means negative

wbits.bin
exp= wbits.uint - (1<<10) + 1
print("The exponent is: ", exp)

pbits.bin
mantissa= 1 + pbits.uint / (1<<52)
print("The mantissa is: ",mantissa )

scinot= mantissa*2 **exp
print("The float {} or {} in Scientific Notation is : {} * 2**{}".format(f, scinot, man
```

```
Sign bit is:  0
The exponent is:  3
The mantissa is:  1.175
The float 9.4 or 9.4 in Scientific Notation is : 1.175 * 2**3
```

In [8]:

```
# https://www.delftstack.com/howto/python/python-epsilon/
#this didn't really end up leading to any conclusions, but a neat method in numpy to ge
import numpy
mach=numpy.finfo('float').eps
print("The value of epsilon is:",mach)
```

```
The value of epsilon is: 2.220446049250313e-16
```

Conclusions

- Ultimately concluded that I am not a magician so I cannot trick the computer into having infinite storage capacity
- What I did learn:

    - Floating point operations can be really dangerous depending on how they are used/what they are used for because of these storage and rounding errors
        - This has really got my mind thinking and I will now be on the lookout for it and I am interested to find ways to "avoid" this
    - I forgot how neat Computer Systems is and man I was quite rusty
    - Python has a lot of really neat libraries and methods and I learned how to add more packages to my jupyter kernel
    - Also:
        - I tried to set cut off timers for myself of 2 hours per day for max of 8 hours on this because I knew I would spend far too long investigating this considering I have many other assignments to do. However, like usual I was not successful in sticking to this, but will try again next time

References

- https://standards.ieee.org/develop/develop-standards/overview/
- http://stupidpythonideas.blogspot.com/2015/01/ieee-floats-and-python.html (this website has a whole rabbit hole of cool articles, I highly recommend)
- Sauer Textbook 2nd edition
- https://docs.python.org/
- https://www.delftstack.com/howto/python/python-epsilon/

Notes

- watched lectures 1.1 and 1.2 and read Sauer chapter 0.0-0.3 prior to starting
- began on Friday 8/26 at 11:30pm worked for 1 hour
- worked 4 hours on Tuesday 8/30, still not much to show for it
- 8/31: office hours and rereading Sauer Wednesday 4 hours
- Thursday 9/1: 3ish hours REALLY TRYING to get bitstring imported
- Friday 9/2: FINALLY figured out how to add packages to my jupyter notebook (probably definitely wasted more time on the bitstring attempt than I should have, especially considering it did not lead to any conclusions/discoveries like I was hoping). At least I learned how to do that. Cleaned up the rest of my notebook and actually reported my "findings". 4 more hours submitted 5:40.