COMPUTER SCIENCE

LESSON 16+17, TUESDAY NOV 7TH + THURSDAY NOV 9TH

WWW.CITA.UTORONTO.CA/~WOODFORD

WOODFORD@CITA.UTORONTO.CA

THIS WEEK IN CS AND STEM

- Going beyond fibre optics – "twisting" light to better information transfer
  - https://futurism.com/twisted-light-could-create-an-ultra-fast-internet-and-make-fiber-optics-obsolete/
  - Get the article for free! http://advances.sciencemag.org/content/3/10/e1700552
- 1st interstellar object has been discovered entering (and leaving) our solar system
  - https://futurism.com/first-interstellar-object/

LIMITATIONS OF FINITE DATA REPRESENTATION

- You may have noticed when converting floats to ints that there was rounding involved, and it wasn't necessarily intuitive.
- There are also issues that arise due to fine data representation in coding as well, and it's not limited to python but an issue with computers in general – they're not infinitely accurate!

LIMITATIONS OF FINITE DATA REPRESENTATION

- We've learned that computers store information as a sequence of binary numbers (ie. base 2). All information is stored in this way, including integers and floating point numbers. But how are they represented in binary, and what's the limit for how many bits a number can occupy?

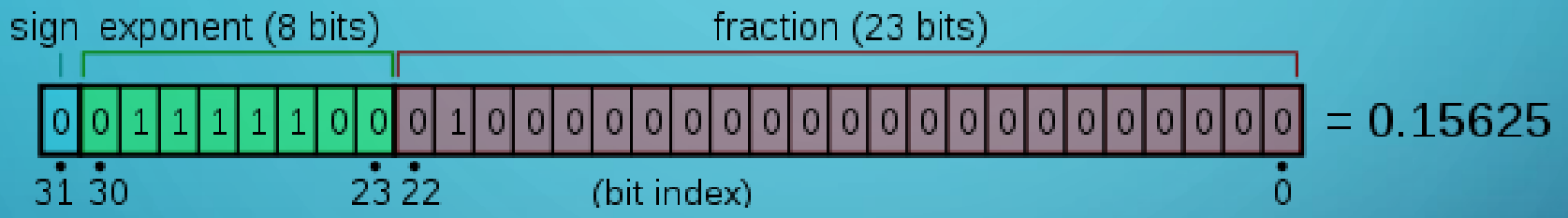
LIMITATIONS OF FINITE DATA REPRESENTATION

- We've learned that computers store information as a sequence of binary numbers (ie. base 2). All information is stored in this way, including integers and floating point numbers. But how are they represented in binary, and what's the limit for how many bits a number can occupy?
  - Single vs double precision (32 vs 64 bit numbers)
  - Typically numbers are stored as single precision*
  - Largest integer: $2^{31} - 1 = 2{,}147{,}483{,}647$ (ie. 32 bits that all have value 1)
  - Largest float: $(2 - 2^{-23}) \times 2^{127} \approx 3.402823 \times 10^{38}$, but how?

*We'll learn later that there are some Python functions that automatically use double precision floats!

LIMITATIONS OF FINITE DATA REPRESENTATION

- There are 3 parts to a float: the sign, the exponent, and the fraction.



- The sign indicates the sign of the float, and the exponent is the power of 2 that the entire float is multiplied by. The fraction is a little harder, where each bit is now $\frac{1}{2}^{\text{bitvalue}}$. Other ways to write this include:
  - $(-1)^{b_{31}} \times (1.b_{22}b_{21}\ldots b_0)_2 \times 2^{(b_{30}b_{29}\ldots b_{23})_2 - 127}$
  - $(-1)^{sign} \times (1 + \sum_{i=1}^{23} b_{23-i}2^{-i} \times 2^{e-127})$

LIMITATIONS OF FINITE DATA REPRESENTATION

- What are some issues with this representation for floating point numbers?

LIMITATIONS OF FINITE DATA REPRESENTATION

- What are some issues with this representation for floating point numbers?
  - The primary is that many floats cannot be represented as a combination of finite binary fractions. This means that many of the fraction (and hence the whole float) is often approximated in machine memory – which leads to loss of accuracy.
  - This also means that some floats that are close together may also be represented by the same binary fraction and exponent
  - Python typically will display a rounded value, and usually only displays the first 12 decimal places if necessary.
    - Note that repr() and str() will round differently. Repr() will keep all of the decimal places you specify (so 17 usually), while str() will reduce the rounding to 12 decimal places!
  - Let's take a look at this using the decimal package.

TESTING PLANS: UNIT TESTING

- Unit testing is something that we've already been doing to a point, and I bet that you all did it with Assignment 4/ Project 1!
- For this kind of testing, you need to make sure your code is well-modularized, and that you can pull parts of it out to test independently given an appropriate set of inputs and standard outputs to test against.
- Let's try this out with Lecture16-17_Testing.py (Open with Spyder)
  - Download and open, but SAVE AS to a new file name! (maybe Lecture16-17_Testing_Unit.py) You'll need the original one later.
  - Let's play a game: Team A goes through the code line by line to fix it. Team B does unit testing.

TESTING PLANS: INTEGRATION TESTING

- This one we haven't focused on too much, but it's the next logical step after unit testing – testing the interaction between two or more "units" explicitly.
- This can look like a check for passing a function, calculations shared between functions, etc. As the name suggests, integration testing is basically a unit test for the framework.
  - Does it make sense to use integration testing for Lecture16-17_Testing.py? Let's try it out!
  - Play the game again – see if it's faster/easier to use unit testing or integration testing. Was one more intuitive? Was one faster? Why?

TESTING PLANS: REGRESSION TESTING

- Regression testing is different than unit or integration testing. It is only applicable to code that did work at one point in the past, and has since been upgraded or modified in some way. One would use regression testing on such code to ensure that it's still working in the intended way.
- One would typically build regression tests OUT OF unit and integration tests, focusing them on areas that are modified often or typically cause issues or have defects, and especially for areas in the code that are critical to the calculation.
  - Does it make sense to do regression testing for Lecture16-17_Testing.py? Why or why not?

ASSIGNMENT #5

- Code/logic testing
- Follow the instructions in Assignment5.pdf. You can put your answers in a text, word, pdf, or pages document. Due Nov 12th by 11:59pm via email submission to woodford@cita.utoronto.ca

REFERENCES

- https://docs.python.org/3.4/tutorial/floatingpoint.html
- https://docs.python.org/3/library/decimal.html
- http://www.lahey.com/float.htm
- https://code.tutsplus.com/articles/the-beginners-guide-to-unit-testing-what-is-unit-testing--wp-25728
- https://docs.pylonsproject.org/projects/pyramid/en/latest/narr/testing.html
- https://www.tutorialspoint.com/software_testing_dictionary/regression_testing.htm