CPSC 250 - Review

Can I assume that everyone got a grade of 100% in CPSC 150/150L, and can instantly recall everything from both of those courses at the 100% level? That's fair, right?

So, no, probably not. Here is the issue. Some of you probably found 150/150L trivial. And others of you actually struggled, and had to work super hard. I have to aim more at the latter group than the former!

But, even if you did super well in 150/150L, I hope that along

the way through this course, you will still learn some things that were not really covered in 150/150L

1. Data Structures.

In my opinion, this is the most important concept in programming. Python, for better or for worse, obfuscates this, maybe for good reason. But, in CPSC 250/250L, we are going to look at this in detail.

Key concepts: Every bit of data is stored in a memory location. Understanding the data structure/data type is going to be

type is (...)
super important, in terms
of designing algorithms
to solve problems by
manipulating that data!!

Primitive Data Types in Python.

1. Integers 1, 0, 17, 42, -6, ...
2. Floats 1.2, -3.14159265, 3.28×10^{-600}, ...
3. Strings "a", "1.2", "-Bob", "\n"
4. Boolean true, false

Notes:

① The amount of space in memory is different for the different types

② The storage mechanism is different for the

different types.

Integers:

Really cool fact: Python's way of storing integers is amazing! Unlike other languages, one does not have to worry about overflow problems.

e.g. in C/C++, only a certain # of bytes in memory are allocated for the different integer types. This means there are limits on the largest/smallest #'s that can be stored.

In Python, there is a lot going on under the hood! It's a high-level language, after all. In fact, calling Python integers a

"primitive" type is a misnomer.

(Underneath, it is actually a C struct object, that acts more like a linked list of memory locations.).

Python example:

```
def factorial(n):
    if (n == 0 or n == 1):
        return 1
    return n * factorial(n-1)

print(factorial(231))
```

Output:

17922 000

with hundreds of digits!!

This probably requires a few 10's of bytes in memory to store! It's amazing that Python just handles it for us !!!

Note: Python requires at least 28 bytes to store even small-ish integers. There is a lot of overhead. In fact, you can think of Python integers as a class of objects that have integer properties, integer algebra, plus other built-in functions, methods, etc. ...

Floats in Python

Python, like most languages, stores floating point numbers as

base 2 (binary) fractions.

(a) what does this mean?

(b) what are the consequences?

Recall: Scientific Notation

3352.28 ⇨ 3.35228×10^3

Okay, that's simple enough.

Q1. How do we represent a "decimal" in binary?

A1.

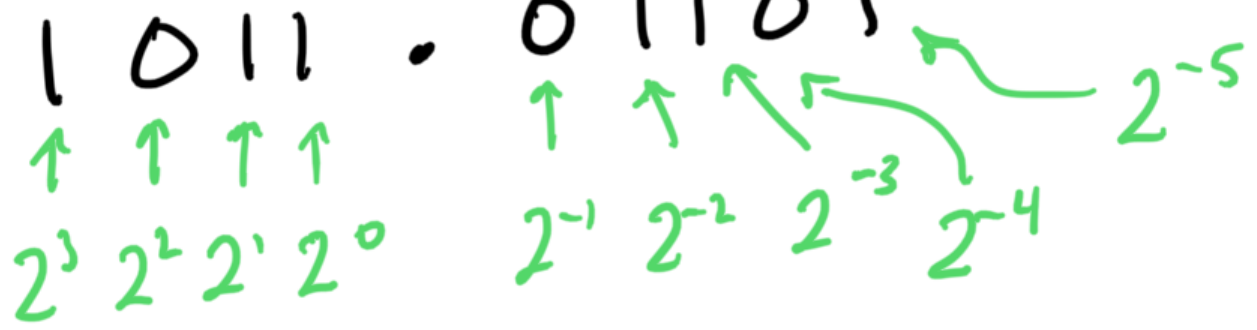
32.56

↑ ↑ ↑ ↑

10's 1's $\frac{1}{10}$'s $\frac{1}{100}$'s

$\therefore$ $32.56 = 3(100) + 2(1) + 5\left(\frac{1}{10}\right) + 6\left(\frac{1}{100}\right)$

So, now in binary:



$$= 1(8) + 0(4) + 1(2) + 1(1) + 0\left(\tfrac{1}{2}\right) + 1\left(\tfrac{1}{4}\right) + 1\left(\tfrac{1}{8}\right) + 0\left(\tfrac{1}{16}\right) + 1\left(\tfrac{1}{32}\right)$$

Binary Scientific Notation

Decimal: 3.285×10^{3} (Mantissa; Base; Exponent)

range = 1.0000 ... to 9.999999

= 1 up to just under the base.

Binary:

Binary Exponent

1011

$1.1011 \times 2^{\text{[illegible]}}$

Base

Binary Mantissa

Range = $1.00000000\ldots$ to $1.11111111111\ldots$

always ~ 1 !

Example : 3.25 (Decimal)

$11 \quad .01$

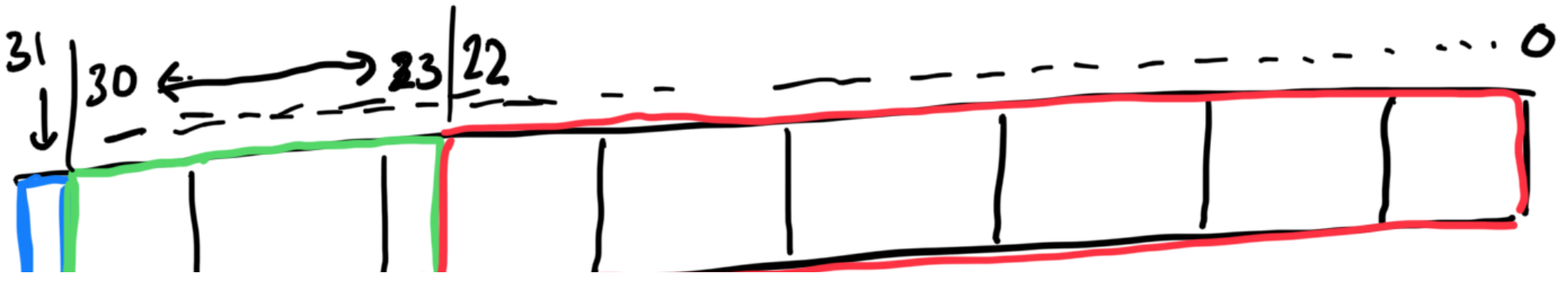
$= 11.01$

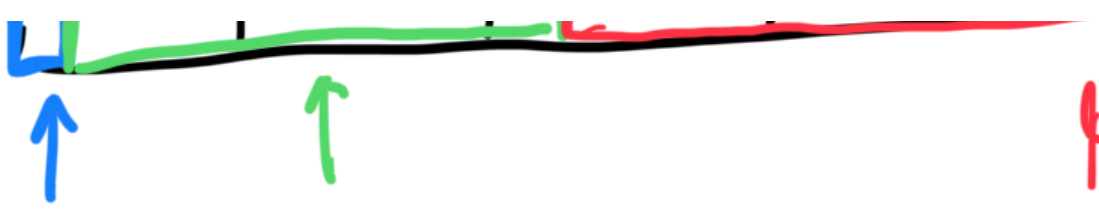
$= 1.101 \times 2^{1}$

just store this part !!

How to store this ?

32 Bit Floating Point





Sign (8 bits) Mantissa (23 bits)

Exponent: → $2^{(E-127)}$ (7 bits)

min 00000001 → $2^{(1-127)} = 2^{-126}$

max 11111110 → $2^{(254-127)} = 2^{127}$

Note: 00000000 and 11111111 are reserved for special #s