# 15-110 recitation 02

**Recap:**

- -converting between decimal and binary
- -using binary data to represent numbers, pixels, letters & code

**Reminders!**

HW-1 due Monday @ Noon!

# Problems

## (review) algorithm trace

**Learning Objectives**

-Be comfortable tracking variables

**Problem**

Algorithm:

```
        Start off with a variable *result*, which is an empty string ""
        Based on an integer input *n*, do the following:
    1.  if  n is negative (n< 0)  do the following
            a.  result = result + 'A'
    2.  if the remainder of n/10 is greater than or equal to 5 (n % 10 >= 5)
            a.  result = result + 'B'
    3.  if n is greater than 5 (n > 5)
            a.  result = result + 'C'
    4.  if none of the above
            a.  result = result + 'D'
    5.  print(result)
```

What is the result when n = -5?

What is the result when n = 105?

What is the result when n = 0?

## fast facts

**Learning Objectives**

-commit to memory the fast facts about binary and binary abstractions
-reason about binary numbers

**Problem**

What is the smallest and largest integer that can be represented with 4 unsigned bits?

What is the smallest and largest integer that can be represented with 6 signed bits?

How many bits in a byte?

How can we store 18 in 4 digit binary?

What is ASCII?

What is Unicode? Why was it created?

## bytecode review

**Learning Objectives**

-students do NOT have to write bytecode

Explain how it works from tokenizing, parsing, then creating the bytecode that the machine runs.

Python interpreter first takes the Python file which is just a text file and checks it for **syntax** (rules that python code has to follow) by **tokenizing** (finding which characters go together) and **parsing** (figuring out what those tokens mean - are they variables, functions, constants?). Then the Interpreter converts the code to bytecode which is machine code that the computer can understand. It does so using strict rules based on the parse.
When it runs, there could still be errors - **runtime errors** like NameError.
There may not be any syntax or runtime errors, but it could still be wrong. Those are **logical errors**. Examples: Use / instead of %. Use < instead of >. Add up to 5 when it should add up to 6. etc.
Review this example from lecture:

-
  ```
  x = 5
  y = 7
  z = x+y
  ```
  becomes:
  ```
  0 LOAD_CONST        1 (5)
  3 STORE_FAST        0 (x)
  6 LOAD_CONST        2 (7)
  9 STORE_FAST        1 (y)
  12 LOAD_FAST        0 (x)
  15 LOAD_FAST        1 (y)
  18 BINARY_ADD
  19 STORE_FAST       2 (z)
  ```

## decimal -> binary

| | |
|---|---|
| **Learning Objectives** | -convert decimal to binary |
| **Problem** | Convert 38 to binary using 8 bits.<br>Convert  101 to binary using 7 bits. |

## binary -> decimal

| | |
|---|---|
| **Learning Objectives** | -convert binary to decimal |
| **Problem** | What is 10?<br>What is 1010100?<br>What is 11+1? (binary) |

## two’s complement

**Learning Objectives**

Recognize two’s  complement algorithm (don't have to write it from scratch)

**Problem**

Go through the algorithm for finding the negative binary representation:
Algorithm for computing negative #’s:
1) If positive, whole number in binary
2) If negative:
    a) Flip all the bits
    b) Compute the whole number
    c) multiply by -1 and subtract 1

Run this algorithm on a couple inputs to ensure that they can trace algorithms and understand why 2’s complement works. No need to memorize algorithm.

Convert (sign-magnitude) 1010100 to decimal.
Make the largest leading bit a 1 and then add back the difference. (Explain this more verbally).
How do you store -52 in 8 bits?

## abstractions of binary

**Learning Objectives**

Recognize the abstraction from lower-level bytecode to python and from binary numbers to integers, pixels, ASCII/unicode, and computer code.

**Problem**

Use what you know about binary and abstracting binary representation to devise a system to represent class numbers at CMU in binary. Consider how they are currently represented. How many bits will you need? How many bytes will you need?