*Count What?*

As a teacher, I can tell you that

* Not being able to count ruins arithmetic students.
* Not being able to do arithmetic ruins Algebra students.
* Not being able to do Algebra ruins Calculus students.
* But Calculus students have finally mastered counting.

**Why do I need this?** Computer Scientists generally use 0-index counting, whereas the world and R (screw you R) and EXCEL (hurray for EXCEL) uses 1-indexing. Python uses 0-indexing. You will use R and python. So you need to know the differences.

In 0-index counting, your first number is zero.

Your second number is one.

In 1-index counting your first number is one.

Your second number is two.

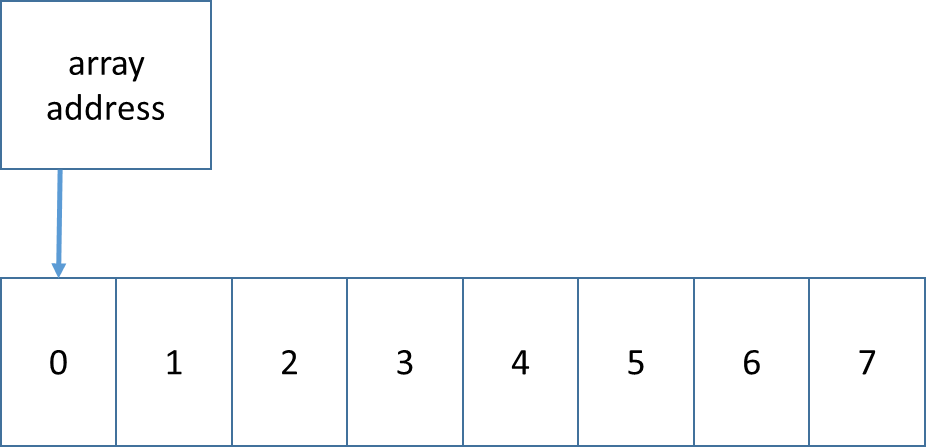
If you aren’t confused enough, 0-index counting typically uses range of the form [a,b); the list starts and includes a, but the list ends and does not include b.

Whereas you normal counters (in the real world) count 1 to 10 and include 10 or [a,b]. Weirdos!

--------------------------------------------------------------------------------------

Why do CS people torture themselves by counting wrong?

* History: Convention that starts with memory manipulation

 (64-bit is 8 bytes! Word.)

Q1 What is address of first byte? Q2 The last byte? Q3 What is the size?

* It can make the size of the counting clearer.
* Reduces calculation waste. (i.e. just good coding practice.)

Meet the C++/Java for loop:

for (int foo = 0; foo + you != 42; foo++){ array[foo] = them(i); }

there is a start condition (foo = 0), a condition to continue (foo + you != 42), and counter (foo++). The logical inverse of continue is your stop. So this loop stops when (foo + you == 42).

# ACTIVITY!

Count to 10!

**O-index [a,b)**

start = 0;

end = 10;

for (int i = start; i < end; i++){}

**1-index [a,b]**

start = 1;

stop = 10;

for (int i = start; i <= stop; i++){}

Why is “<” (less than) the hallmark sign of zero indexing? See only difference above.

**Formulas**

*Length of Range*

**O-index [a,b)**

length = b - a

**1-index [a,b]**

length = b - a + 1 // the + 1 is CS anathema: wasteful

*You want to read the first ten letters out of a string.*

**O-index [a,b)**

start = 0

end = 10

for (int i = start; i < end; i++){}

**1-index [a,b]**

start = 1

stop = 10

for (int i = start; i <= end; i++){}

**Python String slicing (uses 0-indexing)**

Slicing in python is a powerful way to take a substring out of a string

or array of smaller size out of a bigger one.

* Slicing uses 0-indexing [a,b)
* format: array[from:to]
* from field or to field may be blank, meaning beginning or end, respectively
* Python special feature: you can use negative numbers; -1 is last, and -1\*length is first member

***Python language*** */ 0-indexing*

mystring is an array:  
a string length 30

mystring[0:2] is two letters long

mystring[14:30] 15th letter thru end

mystring[10:20] is how many letters long?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

why does the following give an error?

mystring[14:31]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***R language*** */ 1-indexing?*

mystring is an array :  
a vector of characters, length 30

mystring[\_\_\_:\_\_\_] first two letters

mystring[\_\_\_:\_\_\_] 15th letter thru end

mystring[10:20] is how many letters long?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

why does the following give an error?

mystring[14:31]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Python special**

mystring[0:-1]

from 0 to “first position in reverse” or last position

which is same as mystring[0:30]

mystring[0:-29] is mystring [0:2]

position -29 depends on the length!

***Reasons to be a hater***

Pick a side: 0-indexing or 1-indexing

Against 0 indexing :: 14 is the 15th number (confusing)

Against 1 indexing :: length of (1, 30) is not 30 – 1 (confusing)

# Algorithms by tricksy CS people

Computer science is all about cheating. We don’t steal code, we borrow it. (We copy it and never give it back.)

“Mr Dean, everything in a computer is copy. Therefore, your policy for me not to plagiarize, i.e. copy a paper, cannot be enforced when we type our papers on computer. Therefore, ethically, you cannot enforce a plagiarism rule without charging everyone with plagiarism.”

# Case Study

## My database was too slow writing records out of order

1. Need to write to the database in-order
2. So I need to sort too many records to fit in memory. (Data structures training to the rescue.)

So I have to write the data in chunks. But still really big chunks.

Needs:

(1) efficient memory usage

(2) n is large, so a fast sort .. or else we wait forever

(3) read all equal numbers as a group

Definition: In-place – when a sort uses only the original list’s memory (without copying) -- “in place of the original memory”

Sorting needs:

1. Data is in random order. (ASIDE: Data is normally partially sorted --> enter quick sort).
2. In-place (hard)
3. Sort Once (we consider this in context of difficulty; I don’t sort multiple times. Once per program. This means I don’t need a super-efficient sort. But never use a wasteful one!)
4. Unstable (easy)

------------------

Heap Sort

**Heap Sort**

1. Store data on un-sorted heap. Write: O(1) Implemented as an array
   1. Save memory using only 1 pointer
2. In-place sort O(n log n).
3. My read is O( 1 ).

To use memory efficiently, I had to read memory in chunks. Thus, there are multiple databases, one per chunk. Note that here, I am using x pieces, because I chunk my input by memory limit.   
 x \* O( n/x \* log(n/x) ) < O (n log n), *i.e. faster*, despite the same upper bound.

Combining the sorted databases to one sorted database is done in time.