# *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.

Computer Scientists use 0-index counting, whereas the world and R (screw you R) and EXCEL (hurray for EXCEL) uses 1-indexing.

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.

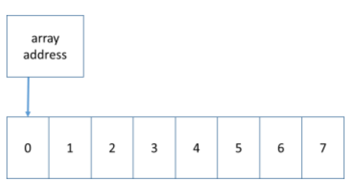
If you aren't confused enough, 0-index counting typically uses range of the form [a,b); the list starts and includes a, but the list ends and does not include b.

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? They are bigger weirdos.

- 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 = ____; ________________; i++){}
```

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

```
start = 1;
stop = 10;
for (int i = ____; ________________; i++){}
```

Why is “<” (less than) the hallmark sign of zero indexing?

**Formulas**

*Length of Range*

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

```
length = ________________________
```

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

```
length = ________________________
```

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

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

```
start = 0
end = 10
for (int i = ____; ________________; i++){}
```

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

```
start = 1
stop = 10
for (int i = ____; ________________; 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
- you can use negative numbers; -1 is last, so that -1*length is first member

*mystring is 30 letters*

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]

___________________________________

*What if we used 1-indexing? (NOT PYTHON)*

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]

___________________________________

# 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 <u>out of order</u>

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

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) In-place (hard)
(2) Sort Once (we consider this in context of difficulty; I don't sort multiple times. Once per program.)
(3) Unstable (easy)

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

Heap Sort

**Heap Sort**

(1) Store data on un-sorted heap. Write: O(1)
(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. 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 $O(\sum n_i) = O(n)$ time.