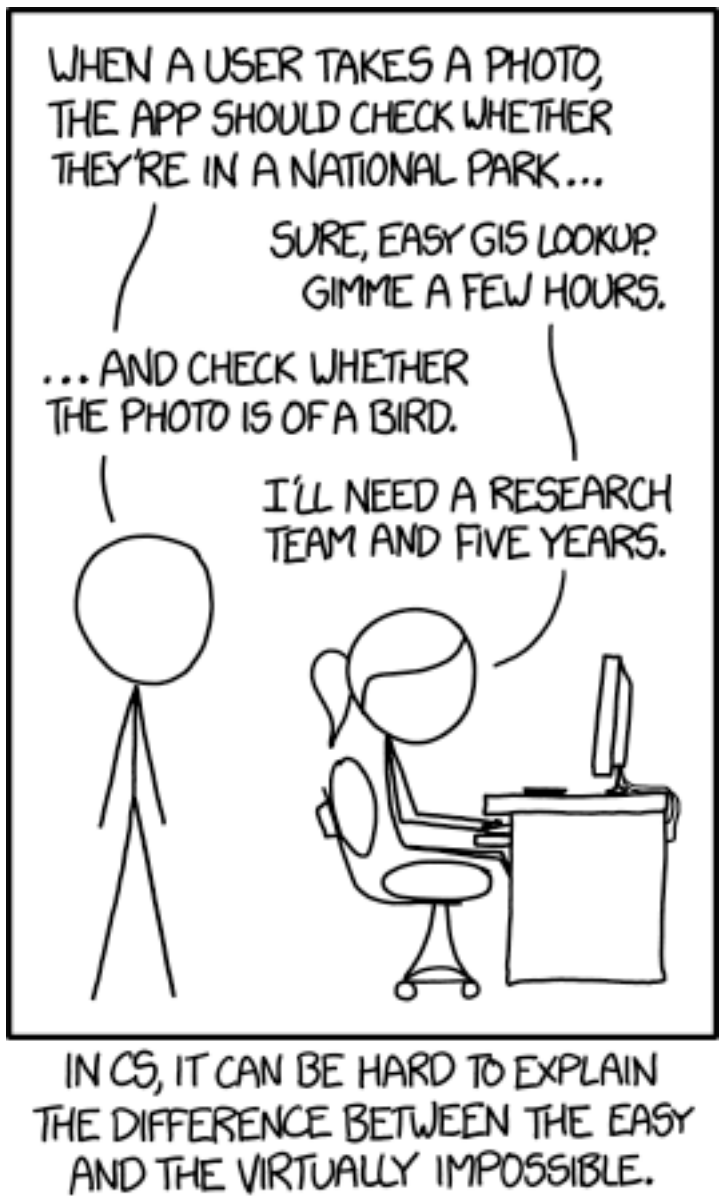
Why bother with pseudo-code?

Typically, humans **solve problems** in a different way than computers and therefore we **frame problems** differently from how a computer does.




```
my_list = [4, 2, 8, 1, 9, -1, 8]

smallest_number = 1000000
for number in my_list:
    if number > smallest_number:
        smallest_number = number
print("I found the smallest number: " + smallest_number)
```

Let's be honest - that is a lot of steps for a very simple task.

What is the smallest number in this list?

[4, 2, 8, 1, 9, -1, 8]

Could you repeat that 1400 times?
Or with a list of 5'000'000 elements?

Pseudo-code is solving problems like a computer would ---------- but without worrying about syntax

Working iteratively with pseudo-code is very powerful

[4, 2, 8, 1, 9, -1, 8]

"Compare all the elements and tell me which one is smallest"

"Go through all the elements and compare one-to-one to find the smallest so far"
"And save it in a variable of course"

"Take the first element of the list and save it in a variable smallest_so_far"
"Use a loop to go through the list and compare each element to smallest_so_far"
"Save new element in smallest_so_far if the new element is smaller"

"Save *my_list[0]* in *smallest_so_far*"
"*for* each *element in my_list* compare it to *smallest_so_far*"
"*if* the element is smaller than *smallest_so_far* then make *smallest_so_far* equal to that element"

In [1]:

```
my_list = [4, 2, 8, 1, 9, -1, 8]

smallest_so_far = my_list[0]
for element in my_list:
    if element < smallest_so_far:
        smallest_so_far = element
print("I found the smallest number: " + str(smallest_so_far))
```

```
I found the smallest number: -1
```

From code to pseudo-code

Some nice rules of thumb:

Every time you see a colon, `:` , read it as "do this" or "then"

`if element < smallest_so_far:`
translates to
"If variable_1 is less than variable_2, then" or
"If variable_1 is less than variable_2, do this"

Every time you see a `for` , read it as "for each"

`for element in my_list:`
translates to
"For each element in iterable, then" or
"For each element in iterable, do this"

[Nice to know]
And every time you see a `for` remember that the variable after `for` will contain the next iterable[i] every time the loop repeats

This

```
i = 0
while i < len(my_list):
    element = my_list[i]
    print(element)
    i = i +1
```

is the same as

```
for element in my_list:
    print(element)
```

In [4]:

```
1 i = 0
2 while i < len(my_list):
3     element = my_list[i]
4     print(element)
5     i = i +1
```

```
4
2
8
1
9
-1
8
```

In [5]:

```
for element in my_list:
    print(element)
```

```
4
2
8
1
9
-1
8
```

Last thing:
Every time you see an `and` , add a "both" after the **while** or **if** keyword
Every time you see an `or` , add an "either" after the **while** or **if** keyword

```
if (element < smallest_so_far) and (heart == True):
```
translates to
"If *both* variable_1 is less than variable_2 *and* variable_3 equals True, then do this"

```
while element < smallest_so_far or heart == True:
```
translates to
"While *either* variable_1 is less than variable_2 *or* variable_3 equals True, then do this"

Isn't this just creating outlines like in the 7th grade?

Algorithm 2 k-means$\|$ (k, ℓ) initialization.

1: $\mathcal{C} \leftarrow$ sample a point uniformly at random from X
2: $\psi \leftarrow \phi_X(\mathcal{C})$
3: **for** $O(\log \psi)$ times **do**
4: $\quad \mathcal{C}' \leftarrow$ sample each point $x \in X$ independently with probability $p_x = \frac{\ell \cdot d^2(x, \mathcal{C})}{\phi_X(\mathcal{C})}$
5: $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}'$
6: **end for**
7: For $x \in \mathcal{C}$, set w_x to be the number of points in X closer to x than any other point in $\mathcal{C}$
8: Recluster the weighted points in $\mathcal{C}$ into k clusters

Source: Bahmani, B., Moseley, B., Vattani, A., Kumar, R., & Vassilvitskii, S. (2012). Scalable k-means++. Proceedings of the VLDB Endowment, 5(7), 622-633

https://en.wikipedia.org/wiki/Computational_thinking (https://en.wikipedia.org/wiki/Computational_thinking)
https://en.wikipedia.org/wiki/Rubber_duck_debugging
(https://en.wikipedia.org/wiki/Rubber_duck_debugging)