CS106A: Wrap-up

CS 106A

Stanford University

Chris Gregg

PDF of this presentation

CS106A: Wrap-up

Where have you been?

Where are you going?

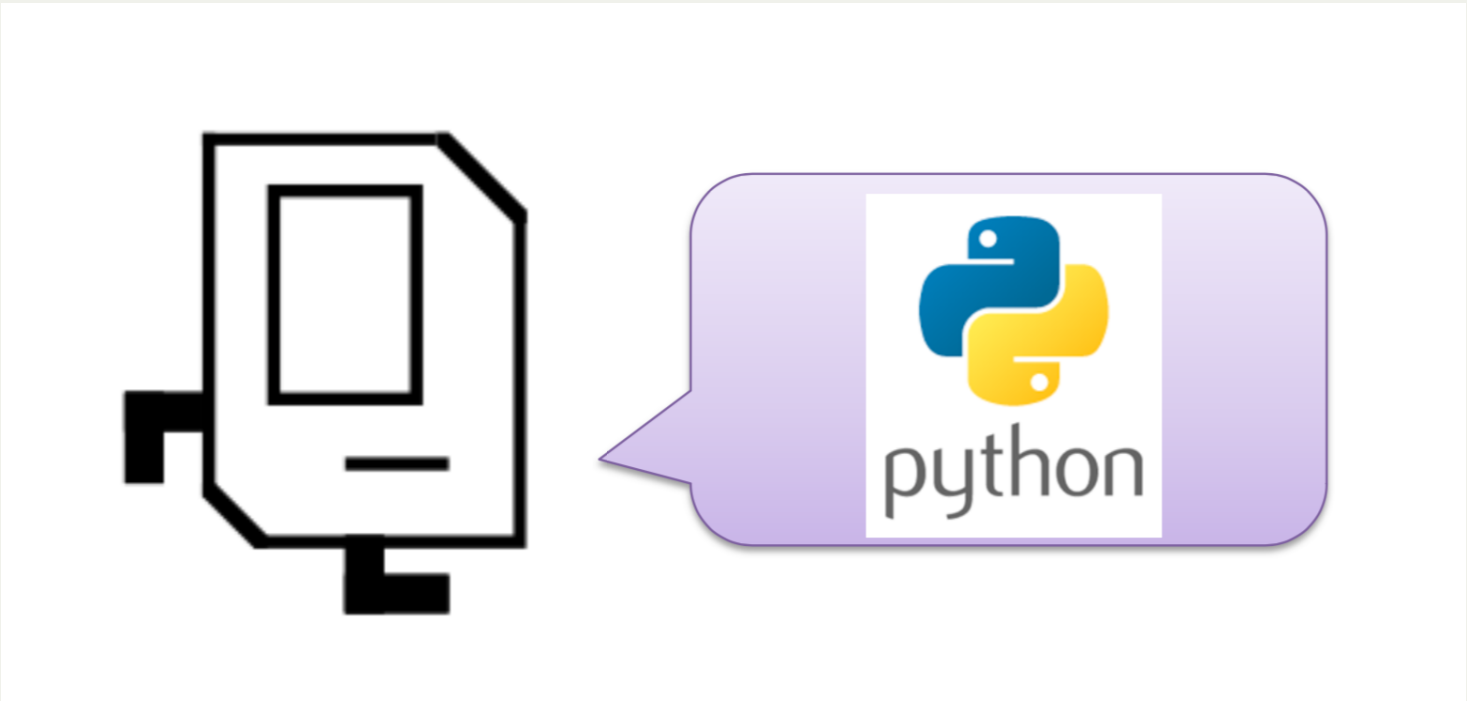
- CS106B Preview

PDF of this presentation

CS106A: Wrap-up: Where have you been?

When you started this course, you may never have programmed a line of code in your life!

We stared simple: Karel!



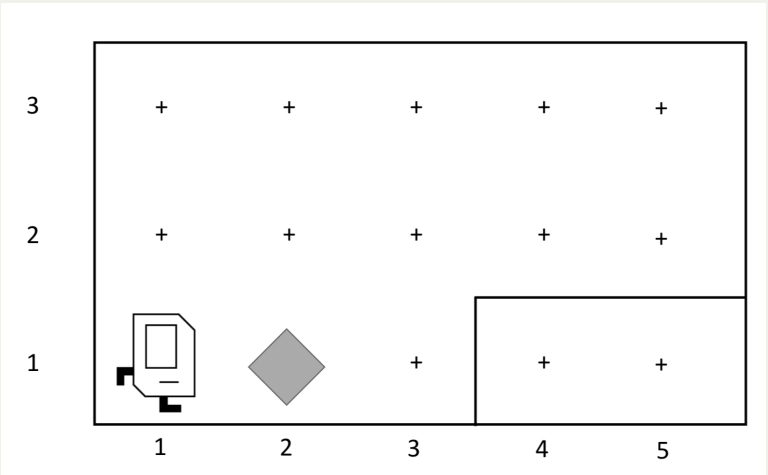
CS106A: Wrap-up: Where have you been?

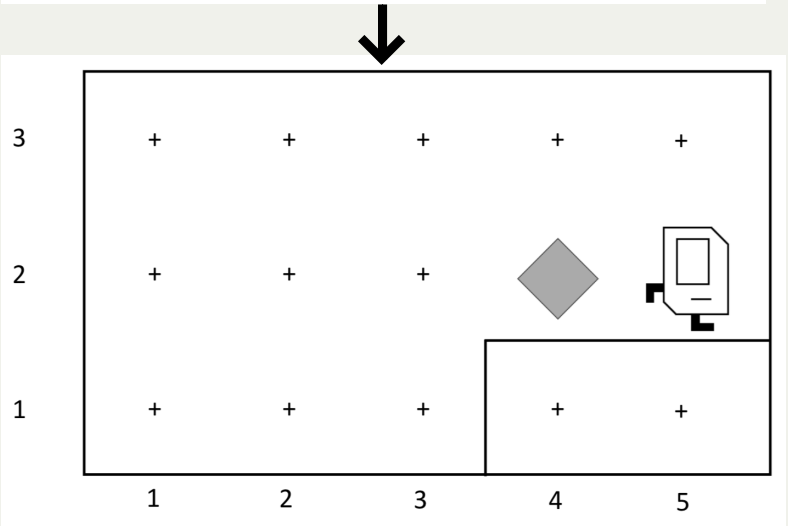
Karel is a great robot...but doesn't know much!

```
move()

turn_left()

put_beeper()

pick_beeper()
```

CS106A: Wrap-up: Where have you been?

But with functions, Karel learns quickly!






```
# Our first program!
def main():
    move()
    pick_beeper()
    turn_left()
    move()
    turn_right()
    move()
    move()
    put_beeper()
    move()

def turn_right():
    turn_left()
    turn_left()
    turn_left()
```

CS106A: Wrap-up: Where have you been?

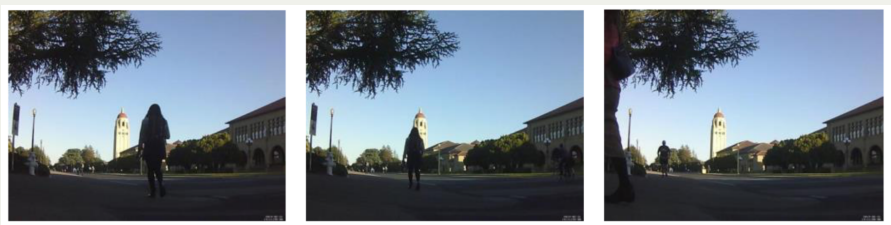
Your first non-Karel assignment!

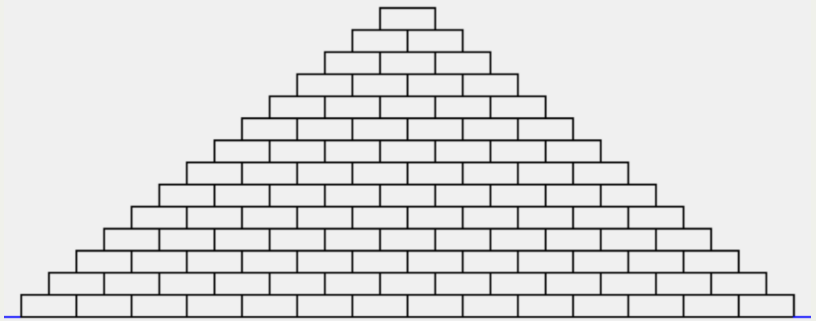
- while loops, random numbers, integers...

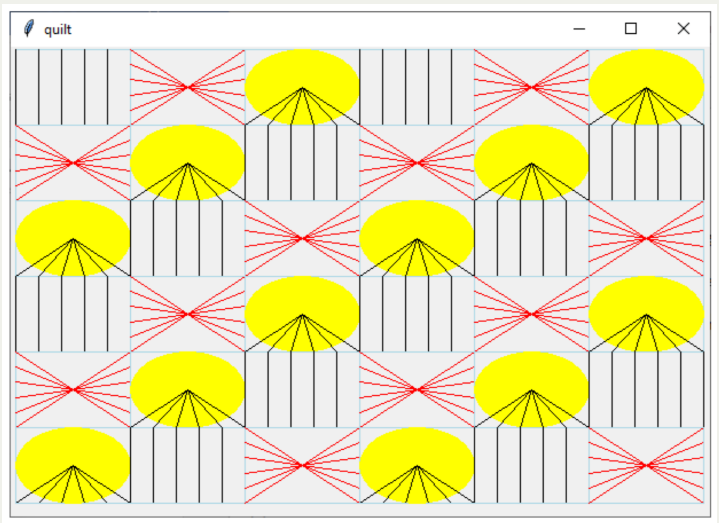
```
What is 51 + 79?
Your answer: 120
Incorrect.  The expected answer is 130
What is 33 + 19?
Your answer: 42
Incorrect.  The expected answer is 52
What is 55 + 11?
Your answer: 66
Correct!  You've gotten 1 correct in a row.
What is 84 + 25?
Your answer: 109
Correct!  You've gotten 2 correct in a row.
What is 26 + 58?
Your answer: 74
Incorrect.  The expected answer is 84
What is 98 + 85?
Your answer: 183
Correct!  You've gotten 1 correct in a row.
What is 79 + 66?
Your answer: 145
Correct!  You've gotten 2 correct in a row.
What is 97 + 20?
Your answer: 117
Correct!  You've gotten 3 correct in a row.
Congratulations!  You mastered addition.
```

CS106A: Wrap-up: Where have you been?

Assignment 3: Ghost and Quilt

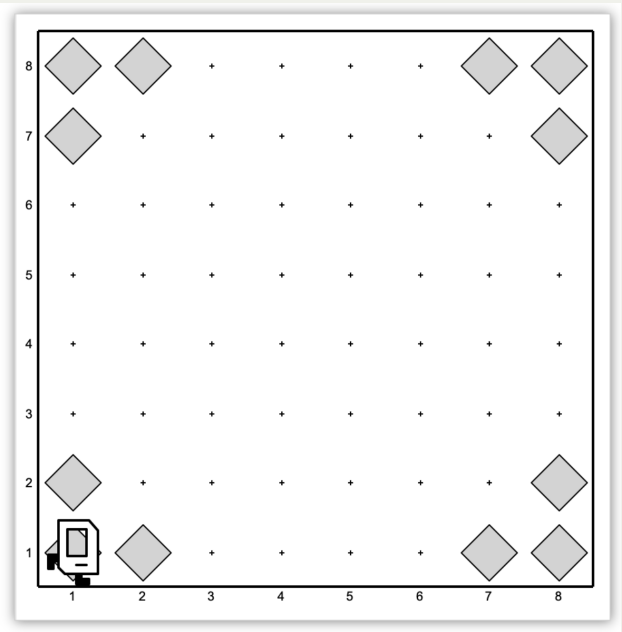






CS106A: Wrap-up: Where have you been?

Week 3 diagnostic!




```
Please enter a starting number: 45
The first square root is 6.708203932499369
The next square root is 2.5900200641113513
The next square root is 1.6093539275471234
It took 3 square roots to get below 2.0
```

CS106A: Wrap-up: Where have you been?

Assignment 4: Sand!

CS106A: Wrap-up: Where have you been?

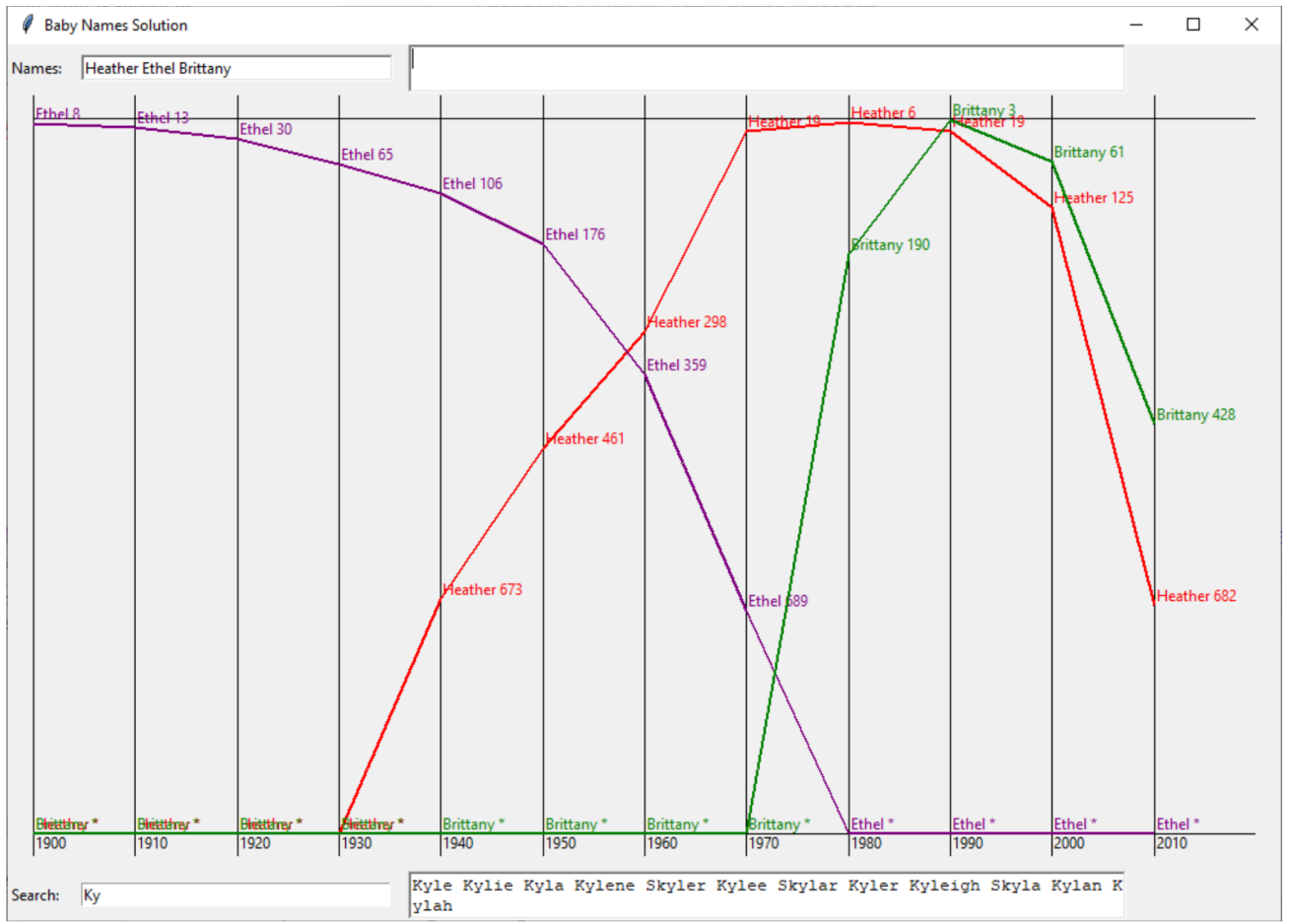
Assignment 5: Strings and Word Guessing

```
Shake Shack: $16
Grocery Hut: $293
Ace Hardware: $14
Joan's Fabric: $18
Nom Nom Nom: $12
```

```
The word now looks like this: -----
You have 8 guesses left
Type a single letter here, then press enter: a
That guess is correct.
The word now looks like this: -A---
You have 8 guesses left
Type a single letter here, then press enter: q
There are no Q's in the word
The word now looks like this: -A---
You have 7 guesses left
Type a single letter here, then press enter: p
That guess is correct.
The word now looks like this: -APP-
You have 7 guesses left
Type a single letter here, then press enter: C
There are no C's in the word
The word now looks like this: -APP-
You have 6 guesses left
```

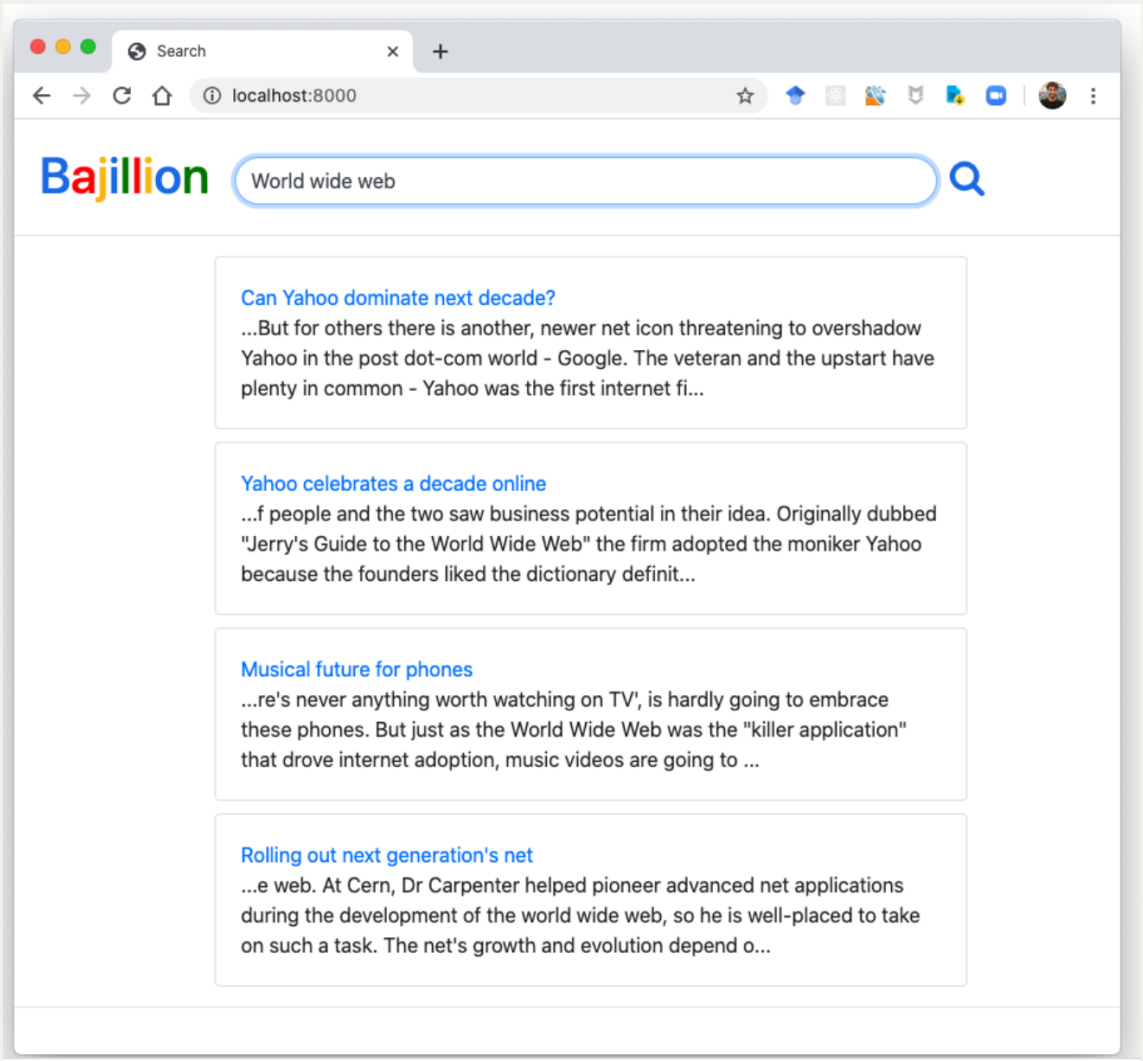
CS106A: Wrap-up: Where have you been?

Assignment 6: Baby Names



CS106A: Wrap-up: Where have you been?

Assignment 7: Bajillion!



CS106A: Wrap-up: Where are you going?

CS106A is just the beginning!

- You can take many other official courses
  - CS106B (or equivalent) -- this is a traditional "data structures" class where you will learn about many different data structures, and how to build them from more rudimentary parts.

CS106A: Wrap-up: CS106B Preview

Data Structures courses (like CS106B) teach you more programming, but they also teach you about efficiency. We never really cared if your programs were particularly efficient in CS106A, but if they aren't efficient in CS106B, your programs may never finish!

For example, does it matter if we remove elements from the front or the back of a list?

It depends on what you care about -- we often care about how long our programs take to run. So, let's check this question in Python. Take a look at this function:

```
def remove_from_list(big_list, front):
    if front:
        for i in range(len(big_list)):
            big_list.pop(0)  # pop from the front
    else:
        for i in range(len(big_list)):
            big_list.pop()  # pop from the back
```

We either remove from the front, or from the back, depending on the argument.

CS106A: Wrap-up: CS106B Preview

In Python, we can use the timeit library to time how long a function takes to run. This function times our function from the previous slide for larger and larger numbers, so we can create a graph:

```
import timeit

def multiple_tests():
    print('num_elements,pop_front(s),pop_back(s)')
    num_elements = 100
    while num_elements < 1000000:
        big_list = [x for x in range(num_elements)]
        runtime_front = timeit.timeit(lambda: remove_from_list(big_list, True),
                                      number=1)

        big_list = [x for x in range(num_elements)]
        runtime_back = timeit.timeit(lambda: remove_from_list(big_list, False),
                                     number=1)
        print(f"{num_elements},{runtime_front},{runtime_back}")
        num_elements *= 2
```

(the program has a fancy *lambda* function, which is necessary to time the function using our parameters)

CS106A: Wrap-up: CS106B Preview

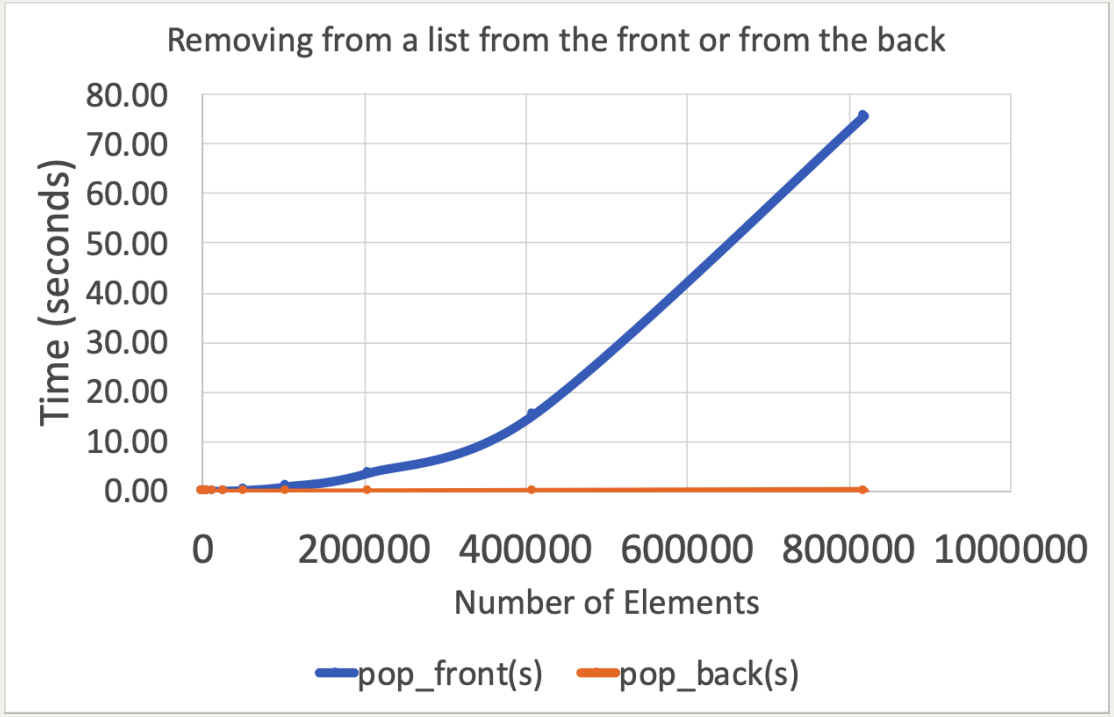
Here is the output when we run the test:

```
num_elements,pop_front(s),pop_back(s)
100,1.2403999999993642e-05,7.804999999999618e-06
200,1.997599999999794e-05,1.2365000000000292e-05
400,4.2077000000001474e-05,2.5162000000002183e-05
800,9.651799999999683e-05,4.8698000000006736e-05
1600,0.00023973700000000375,9.68699999999989e-05
3200,0.0006860179999999966,0.0002778829999999996
6400,0.0026121519999999995,0.0003684139999999747
12800,0.009626186000000002,0.0006882139999999995
25600,0.043316131,0.001345416000000002
51200,0.189158414,0.0026960799999999896
102400,0.8873830819999999,0.005571295000000198
204800,3.6363927990000002,0.01142525800000005
409600,15.318506804,0.02394107599999984
819200,75.538091239,0.051048672999996825
```

Let's graph the data in Excel

CS106A: Wrap-up: CS106B Preview

Here is the graph of the test:



If you remove from the front, the time is much slower! Why? Every time you remove from the *front* of a list, Python has to move all the other elements backwards! When you remove from the *back* of a list, no movement is necessary.

CS106A: Wrap-up: CS106B Preview

So, CS106B involves data structures *and* efficiency

What else does CS106B teach you?

- More data structures: sets, stacks, queues, priority queues, heaps, linked lists, trees, graphs, and others (and you learn more about lists (arrays, or vectors) and dictionaries (maps))
- Recursion: when a function *calls itself*. Example:

```
def count_down(n):
  print(n)
  if n == 0:
    return
  else:
    count_down(n - 1)

def main():
  count_down(10)


if __name__ == "__main__":
  main()
```

Output:

```
10
9
8
7
6
5
4
3
2
1
0
```

CS106A: Wrap-up: CS106B Preview

So, CS106B involves data structures *and* efficiency

What else does CS106B teach you?

- Sorting and Searching (i.e., how to do both algorithmically)
- Low-level memory management, using *pointers*
- *Asymptotic Analysis*, otherwise known as "Big Oh" notation -- this is used to analyze the data we saw in the list timing example earlier.

CS106B is more challenging than CS106A, but of course you already know more! What is more challenging about it?

- At Stanford, it is in a different language (C++) instead of Python. C++ is a different language, which you will have to learn (but you can do it!)
- The assignments are more involved, and expect a better ability to program
- The concepts are generally more advanced.

CS106A: Wrap-up: CS106B Preview

How can you prepare for CS106B?

You *are* prepared for it! CS106A is a *great* preparation for CS106B!

However, if you do want to start learning something now, I would suggest taking a look at a C++ tutorial (e.g., here), and learning a bit about C++. Here is an example Python program, and then the same program in C++:

```python
def main():
    for i in range(5):
        print(f"Python {i}")

if __name__ == "__main__":
    main()
```

```cpp
#include<iostream>
using namespace std;

int main() {
    for (int i = 0; i < 5; i++) {
        cout << "C++ " << i << endl;
    }
    return 0;
}
```

Output:

```
Python 0
Python 1
Python 2
Python 3
Python 4
```

```
C++ 0
C++ 1
C++ 2
C++ 3
C++ 4
```

CS106A: Wrap-up: CS106B Preview

Here is another example:

```
// Assumes we are using the Stanford Library
#include<iostream>
#include<string>
#include "vector.h" // Stanford vector, similar to a list in Python
#include "simpio.h"
using namespace std;

int main() {
    // create a vector
    Vector<string> vec;
    while (true) {
        string s = getLine("Please enter a name (blank line to end): ");
        if (s == "") {
            break;
        }
        vec.add(s);
    }
    cout << "You entered:" << endl;
    for (string s : vec) {
        cout << s << endl;
    }
    return 0;
}
```

C++ does do things differently! (See the next slide for the equivalent Python program)

CS106A: Wrap-up: CS106B Preview

Equivalent Python program:

```
def main():
    my_list = []
    while True:
        s = input("Please enter a name (blank line to end): ")
        if s == '':
            break
        my_list.append(s)
    print("You entered:")
    for s in my_list:
        print(s)

if __name__ == "__main__":
    main()
```

Python programs tend to be shorter than C++ programs. But, there are some similarities:

- There are **while** loops and **for** loops in both.
- Lists and Vectors are similar
- Both have **break** statements
- Both use dot notation

CS106A: Wrap-up: CS106B Preview

Some things are different in C++

- All variables must be declared before they are used
- The **for** loop has a different structure
- You can only have a single type inside of a collection (e.g., **int**s, or **string**s)
- Indentation is not required, though it is good style. Blocks are differentiated by curly braces, **{}**, instead of indentation.
- **`if`** statements and **`while`** statements require the boolean value to be inside of parentheses, e.g., **`if (x == 4) {`**
- There is no list or string slicing. :(

In the end: C++ is a new language and will take some time to learn, but it is a powerful and fast language, and a good one to have in your programming toolbox.

CS106A: Wrap-up: Other Programming Ideas

You don't have to take other official courses to learn more computer science!

Things to learn on your own:

- A new programming language. Good candidates:
  - Javascript: the de facto standard on the World Wide Web
  - Haskell – a great functional programming language. Functional programming is very different than procedural programming (which we have been learning), and it is eye-opening to code in a functional language. The best online resource: Learn You a Haskell for Great Good
  - Swift – if you want to program for a Mac or iPhone
  - Java – if you want o learn to program an Android phone (also, Kotlin, a newer language for Android phones)
  - C++ – already discussed
  - C -- a great low-level language for embedded systems, and what C++ is based on

CS106A: Wrap-up: Other Programming Ideas

You don't have to take other official courses to learn more computer science!

Things to learn on your own:

iOS / Android Programming: learn to program your phone!

- Best iOS resource: https://www.raywenderlich.com
- Good tutorials link: http://equallysimple.com/best-android-development-video-tutorials/
- Want to code for all phones (and the web, and the desktop?) Check out React Native: https://facebook.github.io/react-native/

Hardware: Raspberry Pi, Arduino, FPGA: Hardware is awesome!

- Raspberry Pi resources
- Arduino Resources
- FPGA resources

CS106A: Wrap-up: Final Thoughts

It is a *great* time to be in computer science!

- Virtually everything you do in the world today has some level of computing to support it.
- Even if you never program another line of code in your life, you have learned a ton of problem solving skills, and you also know enough about computing to understand it better than you did before (and to appreciate how challenging programming is!)
- If you do end up in a computing field, you will almost certainly be very employable for the rest of your life.
- As we saw in Monday's lecture: there is still a lot of work to do to get the computing field to be more diverse, but you can be a driving force to accomplish that!
- If you think back on how much you knew about programming when you started this course, and how much you know now, you deserve a huge congratulations -- you've learned *so* much this quarter, and you still have a lot of great learning ahead of you. CONGRATULATIONS!