

# Code in Place

# 2024

Stanford CS106A

Section - Week 1

Welcome to Section!

David Tsai, Code in Place

# Today's Agenda



## Introductions

Icebreakers



## Norms

Guidelines



## Concept Review

Karel, Control Flow



## Practice Problems

"Hospital Karel"

# Please Turn On Your Camera



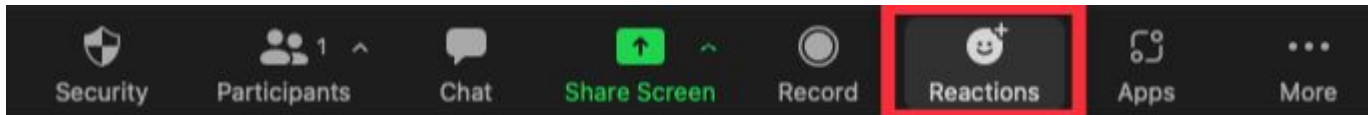
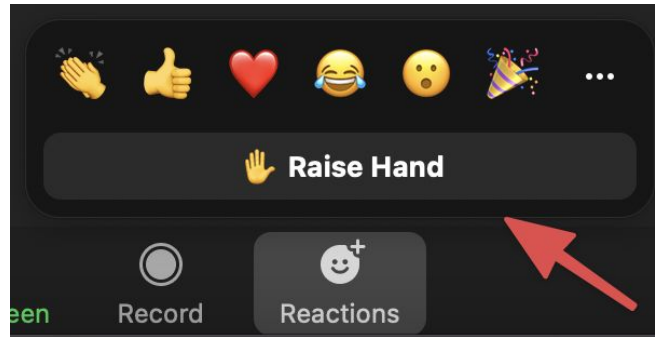
If you're able, please **turn on your camera!**  
.... It can really make the section come to life!



(Image source: <https://as.virginia.edu/eight-ways-get-more-out-zoom>)

# Zoom Reactions

- 👍 **Thumbs Up:** If you understand.
- 🙋 **Raise Hand:** If you have a question (or just speak in the mic).



# Introductions

Hi, I'm David, your  
Section Leader!

- Section Leader for Code in Place 2023. Student for CIP 2021.
- Massachusetts Institute of Technology (MIT) - Computer Science
- Worked as a Software Engineer and Product Manager
- Love photography, video games, movies
- Cool superpower I'd like to have:
  - Felix Felicis ("Liquid Luck" potion) from Harry Potter

How about you?

- Where are you from?
- Your coding experience
- Cool superpower you'd like to have

# Course Weekly Timeline

## Lectures

### 2 Lessons

Both released every Monday.  
~1 hour per lecture.

## Section

### Live Zoom Session

Practice lecture concepts.  
Prepare you for assignment.

## Assignment

### Coding Challenges

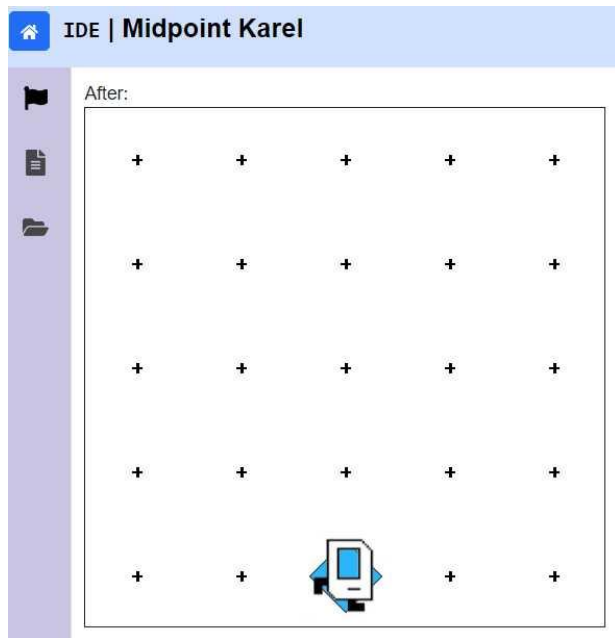
Solidify your understanding.  
Optional: "Extension" projects

# Brief Tour of Course's Web Platform

The screenshot displays the Code in Place web platform interface. The browser address bar shows the URL `codeinplace.stanford.edu/cip3/join/ide/a/piles`. The interface is divided into several sections:

- IDE | Piles**: The main header with a **Run** button and a **Share** button.
- Instructions**: A sidebar on the left containing a **Restart** button and a task description: "Your task: Write a program in the editor, that makes Karel pick up all the beepers on the first row of this world." Below this, it states: "After your program finishes, Karel's world should have no beepers, like so:" followed by a 6x6 grid of beepers. A **Text descriptions** toggle is visible.
- main.py**: The central code editor showing a Python program for Karel. The code includes imports, file naming, a main function, and a `pick_beeper` function. Comments at the bottom state: "# don't edit these next two lines", "# they tell python to run your main function", and "# if \_\_name\_\_ == '\_\_main\_\_':".
- World**: A sidebar on the right showing a 6x6 grid of beepers. Three beepers on the bottom row are highlighted with blue diamonds and the number 10. A **Text descriptions** toggle and a slider are also present.
- Terminal**: A black bar at the bottom of the interface.

# Some Basic Python Functionality is TURNED OFF....for now!



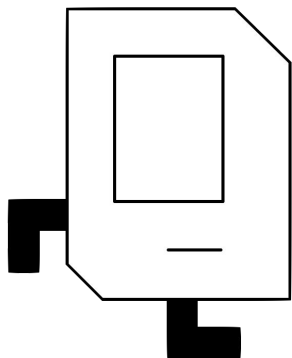
For the duration of Karel unit (Weeks 1 & 2):

- Basic Python functionality is turned off for simplicity
  - Focus on building blocks of Karel commands
- Examples of disabled functionality:
  - print statements
  - conditional keywords (**NOT**, etc.)
  - variables, lists, dictionaries
  - input arguments to functions
  - function return statements



# Concepts Review (Lightning Round!)

# Karel is like a LEGO set with only 4 types of blocks....but you can do a lot!



Karel only knows four commands at the beginning:

`move()`

`turn_left()`

`put_beeper()`

`pick_beeper()`

Helpful website to visualize Karel (by Sophia Westwood):

[www.karelhelper.com](http://www.karelhelper.com)

# Control Flow



for **Loops**



while **Loops**



if/else

# for Loop

```
for i in range(count):  
    statements                # note indenting
```

```
# Example:  
def turn_right():  
    for i in range(3):  
        turn_left()           # note indenting
```

# while Loop

```
while condition:  
    statements                # note indenting
```

```
# Example:  
def move_to_wall():  
    while front_is_clear():  
        move()                # note indenting
```



# Conditions Karel Can Check For

<i>Test</i>	<i>Opposite</i>	<i>What it checks</i>
<code>front_is_clear()</code>	<code>front_is_blocked()</code>	Is there a wall in front of Karel?
<code>left_is_clear()</code>	<code>left_is_blocked()</code>	Is there a wall to Karel's left?
<code>right_is_clear()</code>	<code>right_is_blocked()</code>	Is there a wall to Karel's right?
<code>beepers_present()</code>	<code>no_beepers_present()</code>	Are there beepers on this corner?
<code>beepers_in_bag()</code>	<code>no_beepers_in_bag()</code>	Any there beepers in Karel's bag?
<code>facing_north()</code>	<code>not_facing_north()</code>	Is Karel facing north?
<code>facing_east()</code>	<code>not_facing_east()</code>	Is Karel facing east?
<code>facing_south()</code>	<code>not_facing_south()</code>	Is Karel facing south?
<code>facing_west()</code>	<code>not_facing_west()</code>	Is Karel facing west?

Check out "Karel Reader", Chapter 10 for full reference

# Which Type of Loop Should You Use?

## for Loop

You know exactly how many times to repeat (definite loop)

## while Loop

You don't know exactly how many times to repeat (indefinite loop)

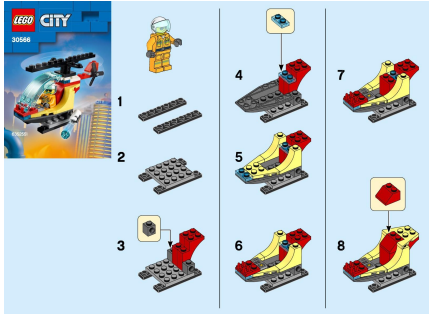
# if-else Statement

```
if condition:  
    statements                # note indenting  
else:  
    statements
```

```
# Example:  
def invert_beepers():  
    if beepers_present():  
        pick_beeper()        # note indenting  
    else:  
        put_beeper()         # note indenting
```

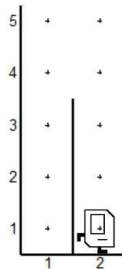


# Decomposition



Break down a problem into more manageable sub-problems

- “Divide and conquer” approach
- Which sorts of tasks should be decomposed?
- Rule of Thumb: Each function should execute a single purpose.
- Code becomes easier to read! For both you and everyone else.



```
turn_left()
while right_is_blocked():
    move()
turn_right()
move()
turn_right()
move_to_wall()
turn_left()
```



```
ascend_hurdle()
move()
descend_hurdle()
```

# Section Exercise: "Hospital Karel"

