

Lecture 3:

# INTRO TO PYTHON & PAIR PROGRAMMING

---

CSC111: Introduction to CS through Programming

R. Jordan Crouser

Assistant Professor of Computer Science

Smith College

# Announcements

- Note Taker: paid position available
  - **Requirements:** on work study, 1.5hr week
  - **What you'll do:** take clear, concise notes for this course and deliver them to the Office of Disability Services
  - **If interested:** email Lisa Roberge in ODS at [lroberge@smith.edu](mailto:lroberge@smith.edu) (subject: *Note Taker for CSC111-01 / R. Jordan Crouser*)

# Overview of the week

- ✓ Crash course in computers
  - ✓ A little history
  - ✓ 4 key components
  - ✓ Quick hardware demo
  - ✓ Boolean logic
- Introduction to Python
- Life skill #1: pair programming
- Lab: Getting Started with Python
- User Input

# Recap

What do you remember  
from Monday's class?



# Recap: the good news

- “High level” programming languages like Python mean we don’t have to write in “low level” binary
- Instead, we write statements like:

```
print("hello")
```



# What do you know?

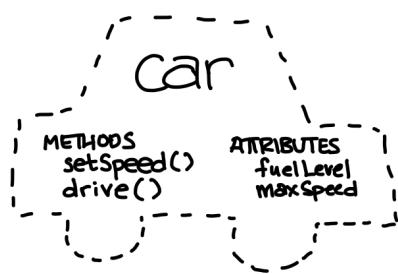




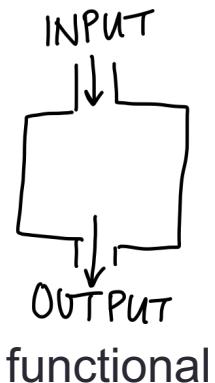
multi-paradigm  
interpreted language  
with dynamic typing  
and automatic memory management



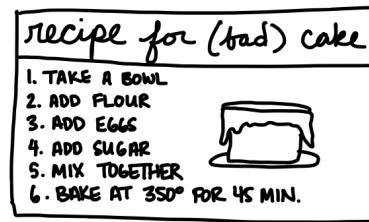
multi-paradigm  
interpreted language  
with dynamic typing  
and automatic memory management



object-oriented



functional



imperative



declarative



multi-paradigm  
interpreted language  
with dynamic typing  
and automatic memory management

# Quick digression

	COMPILER	INTERPRETER
What it takes in:		
What it returns:		
Relative speed:		
Memory usage:		
Work is done:		
Reports errors:		
Example language:	 Java	 python™

# Quick digression

	COMPILER	INTERPRETER
What it takes in:	an entire program	a single line of code (or a single instruction)
What it returns:	intermediate object code (e.g. classes)	<nothing>
Relative speed:	faster	slower
Memory usage:	uses more memory (↑ objects take space ↑)	uses less memory (no intermediate objects)
Work is done:	just once	every time the code is executed
Reports errors:	after checking the entire program	after each instruction is run



multi-paradigm  
interpreted language  
with dynamic typing  
and automatic memory management



more about this  
a bit later

# Core concept 1: variables

- In CS, a **variable** is a place to store a piece of data
- In Python, variables are:
  - declared by giving them a name
  - assigned using the equals sign
- Example:

The diagram shows the assignment statement `x = 3`. A blue arrow points from the text "declaring a variable x" to the variable `x`. Another blue arrow points from the text "assigning the value 3 to x" to the value `3`.

declaring  
a variable x      `x = 3`

assigning  
the value 3 to x

# Core concept 2: numeric values

- Two kinds of **numbers** in CS:
  - integers (“whole numbers”)
  - floats (“decimals” or “floating point numbers”)
- In Python, the kind of number is implied by whether or not the number contains a **decimal point**
- Example:

**x = 3**

**x = 3 . 0**

# Basic operations on numbers

- Addition: +
- Subtraction: -
- Multiplication: \*
- Division: /
- Exponentiation: \*\* (power)
- Modular arithmetic: % (modulo)

# Core concept 3: strings

- In CS, a sequence of characters that isn't a number is called a **string**
- In Python, a string is declared using **quotation marks**
- Strings can contain letters, numbers, spaces, and special characters
- Example:

```
x = "Jordan"
```

```
x = "Stoddard G2"
```

# Math on strings?!

- Let's say we have:

`x = "Jordan"`

- If we ask Python for:

`2 * x`

what will happen?

# `print()`: a very useful function

- A function is a procedure / routine that takes in some input and does something with it (just like in math)
- In Python, the `print()` function takes in a value and outputs the value to the console
- This seems silly now, but will come in handy in lab when you write/run your first program inside a `file`

# Quick check in

a = 1

b = 2.7

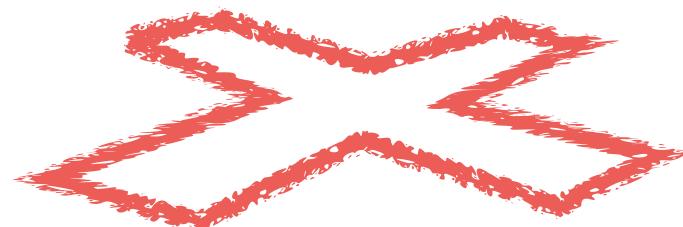
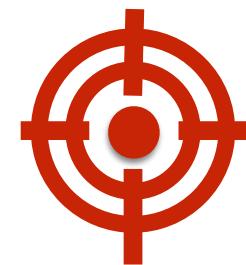
c = "Smithies"



# Overview of the week

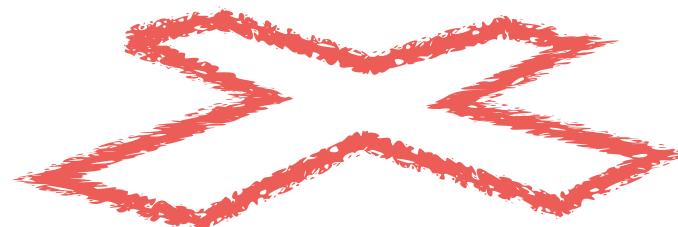
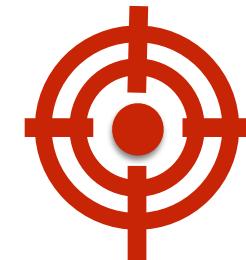
- ✓ Crash course in computers
  - ✓ A little history
  - ✓ 4 key components
  - ✓ Quick hardware demo
  - ✓ Boolean logic
- ✓ Introduction to Python
- Life skill #1: pair programming
- Lab: Getting Started with Python
- User Input

# The programming process



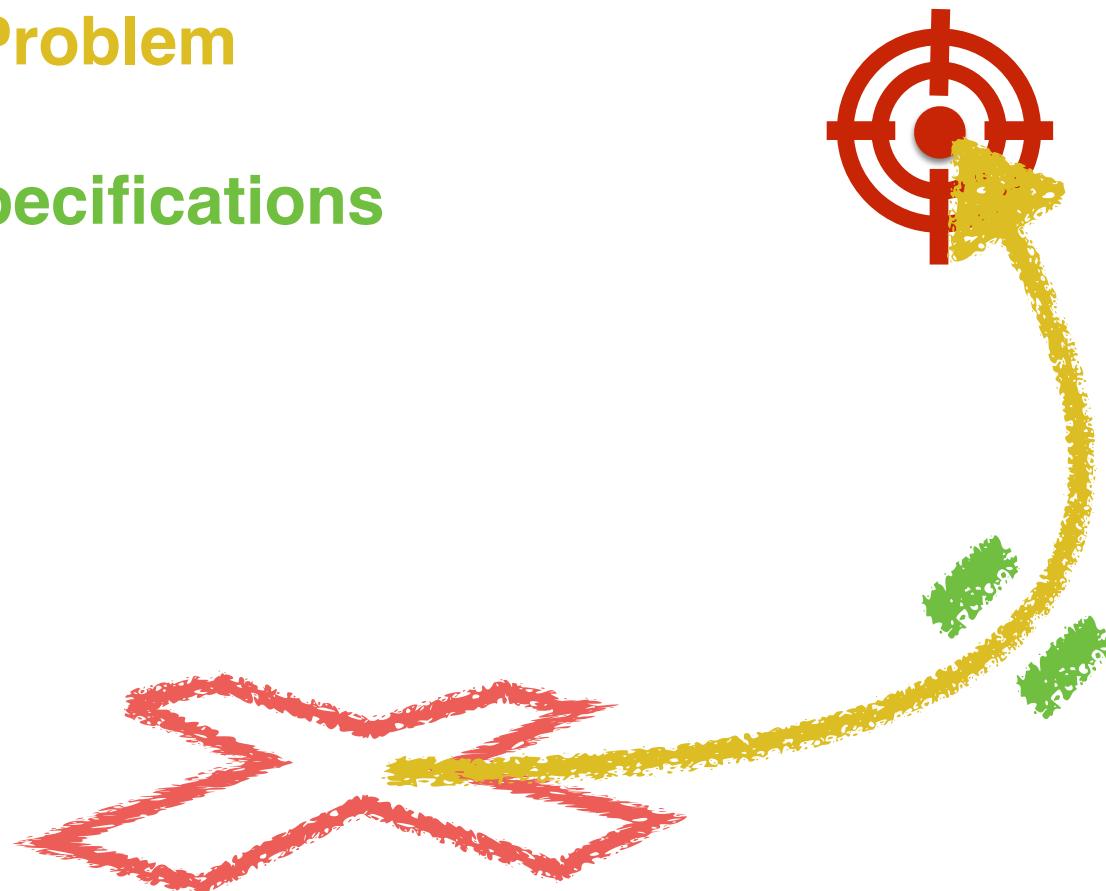
# The programming process (idealized)

- Analyze the **Problem**



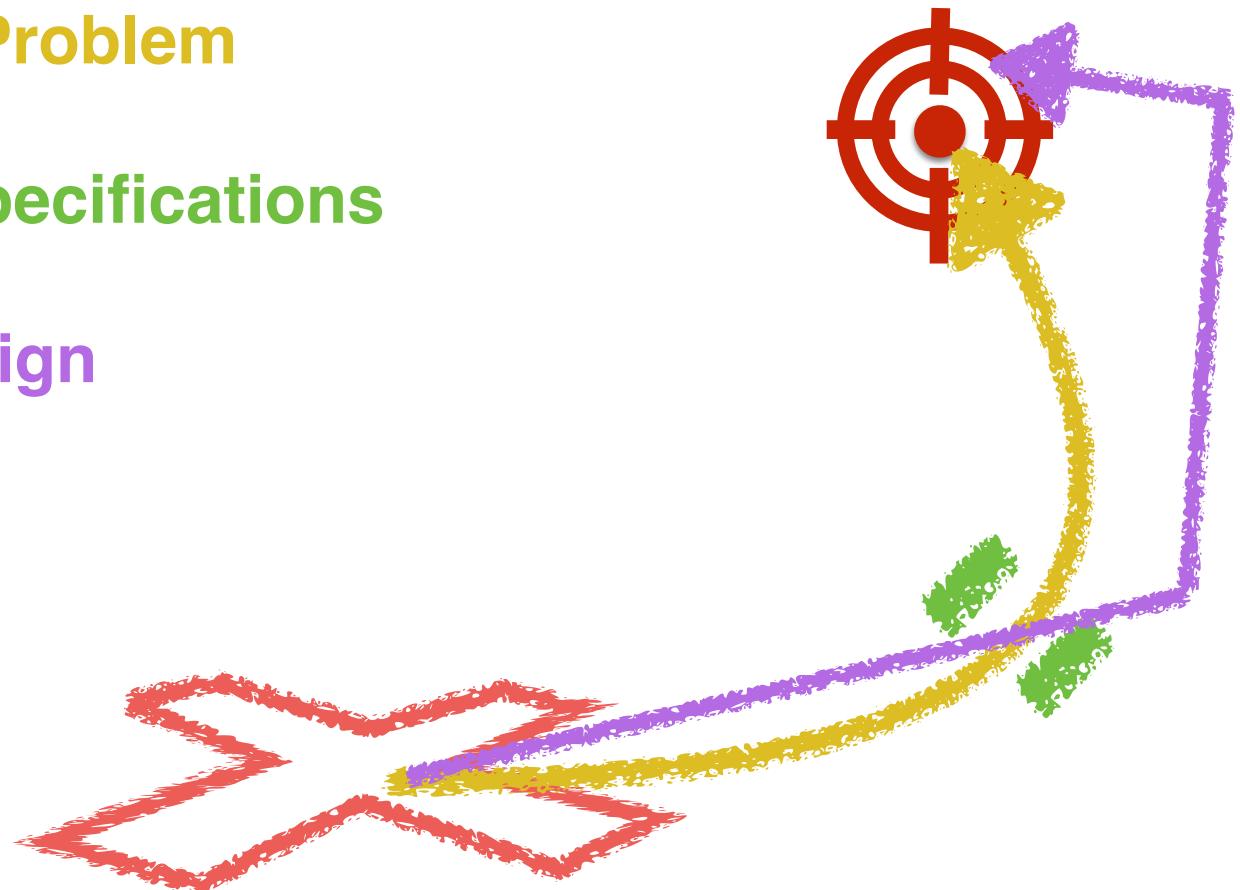
# The programming process (idealized)

- Analyze the **Problem**
- Determine **Specifications**



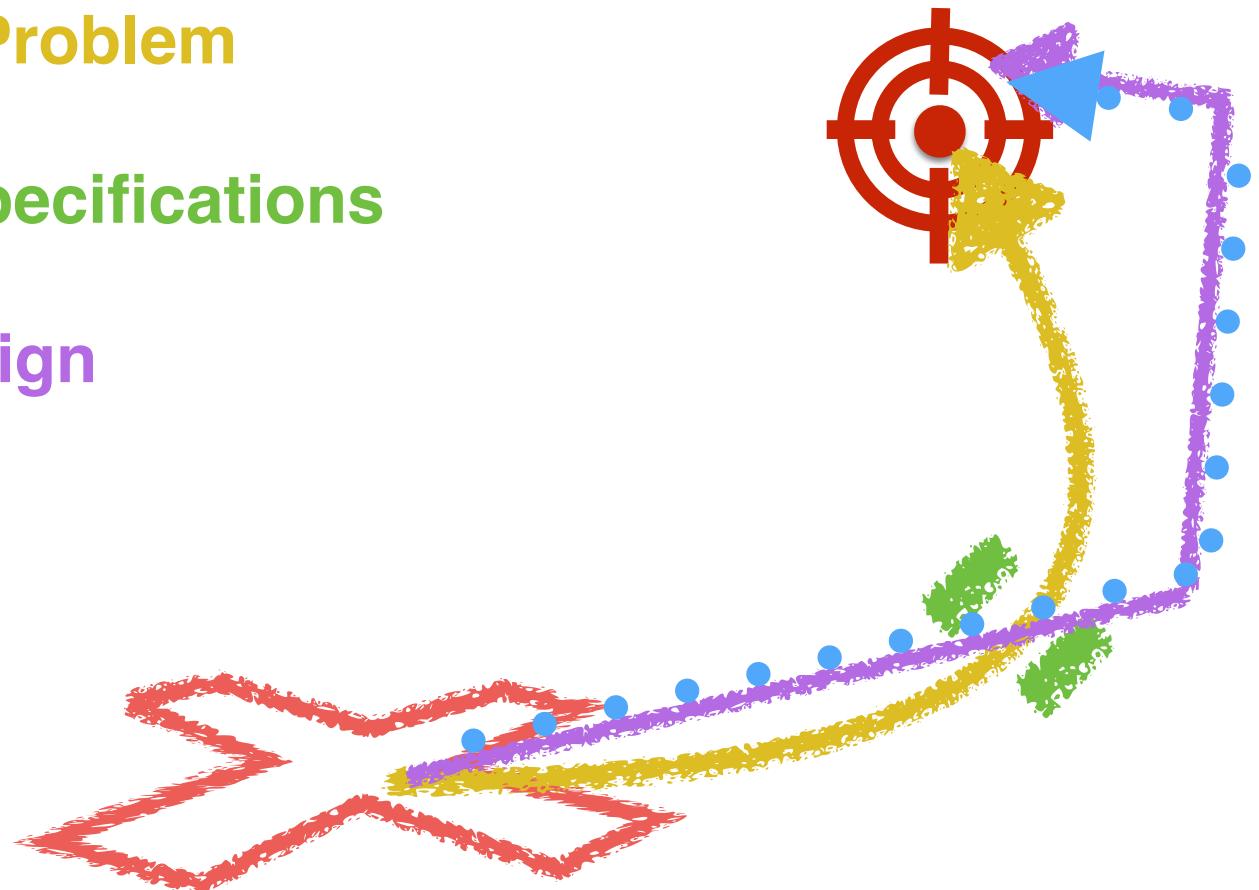
# The programming process (idealized)

- Analyze the **Problem**
- Determine **Specifications**
- Create a **Design**



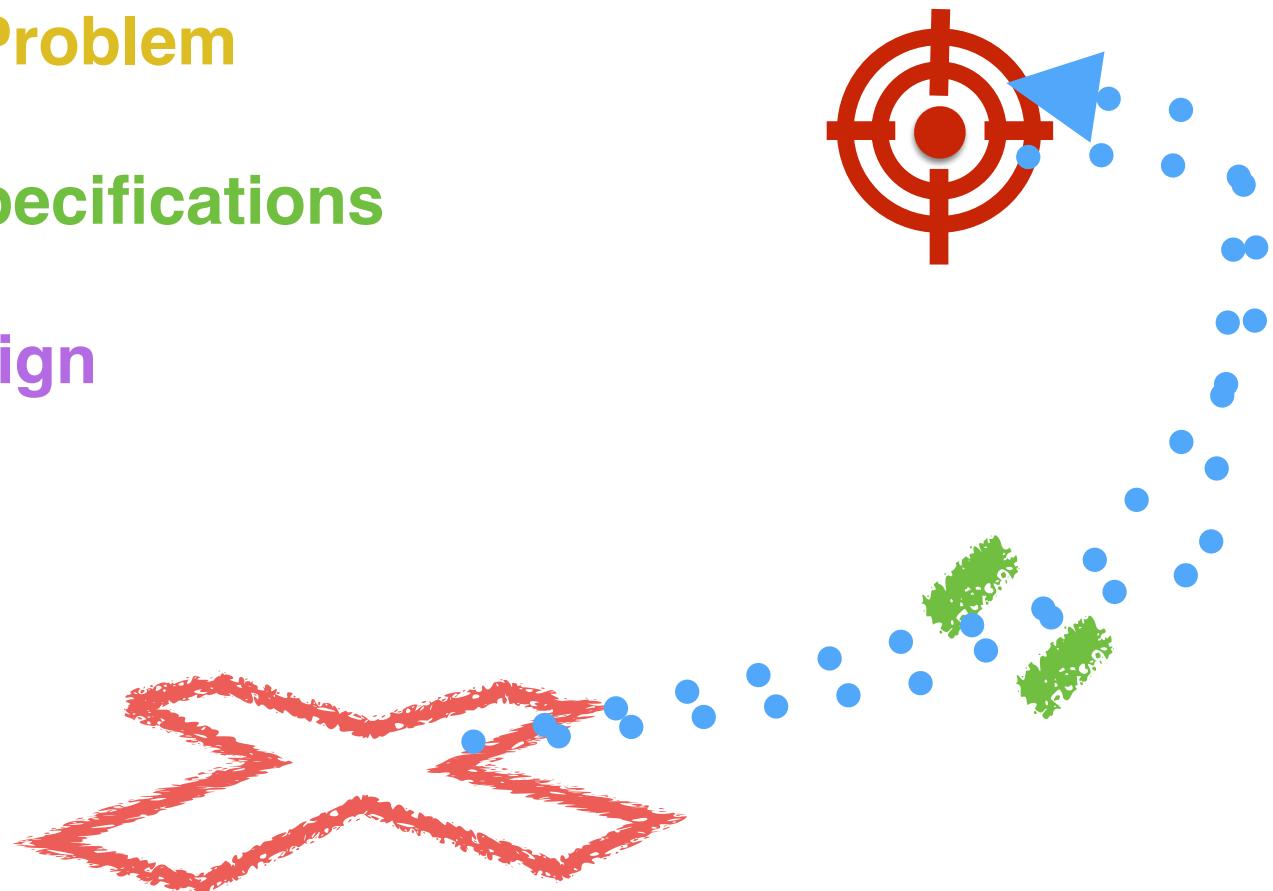
# The programming process (idealized)

- Analyze the **Problem**
- Determine **Specifications**
- Create a **Design**
- **Implement**



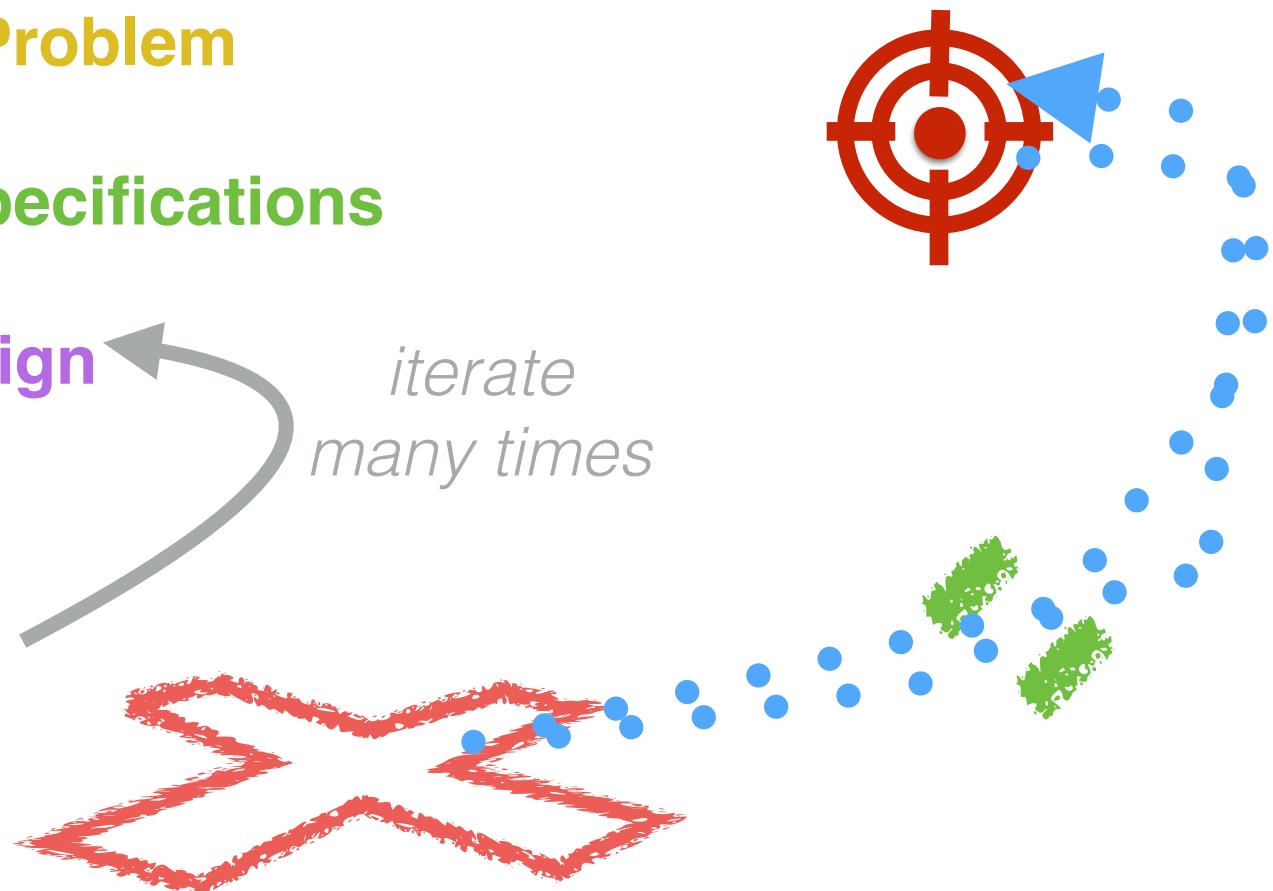
# The programming process (idealized)

- Analyze the **Problem**
- Determine **Specifications**
- Create a **Design**
- **Implement**
- Test & Debug

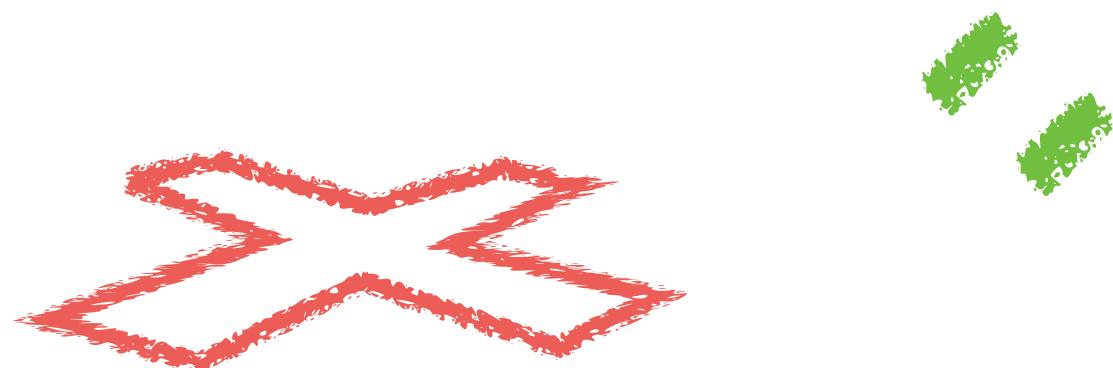


# The programming process (more realistic)

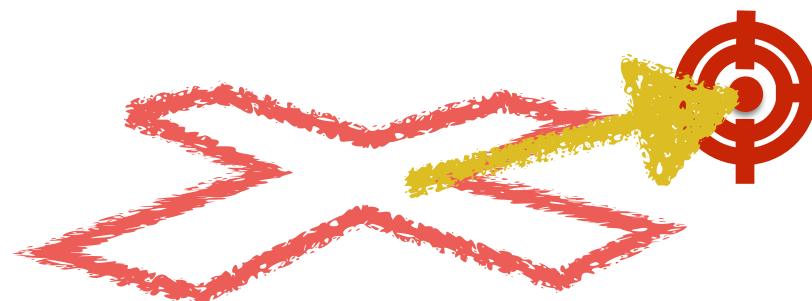
- Analyze the **Problem**
- Determine **Specifications**  
*Refine the*
- ~~Create a **Design**~~  
↓
- **Implement**
- **Test & Debug**



# Getting started



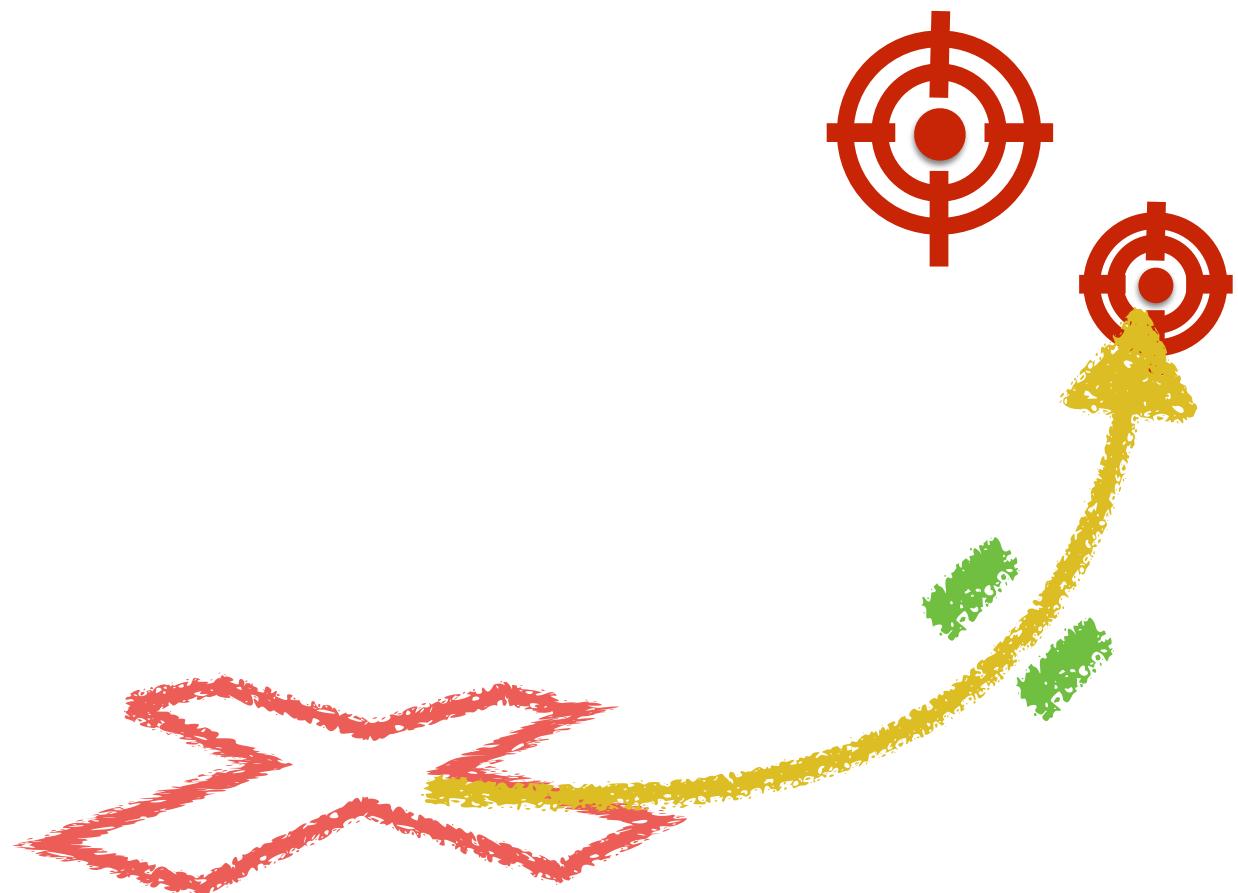
# “S<sup>4</sup>”: start small | slow | simple



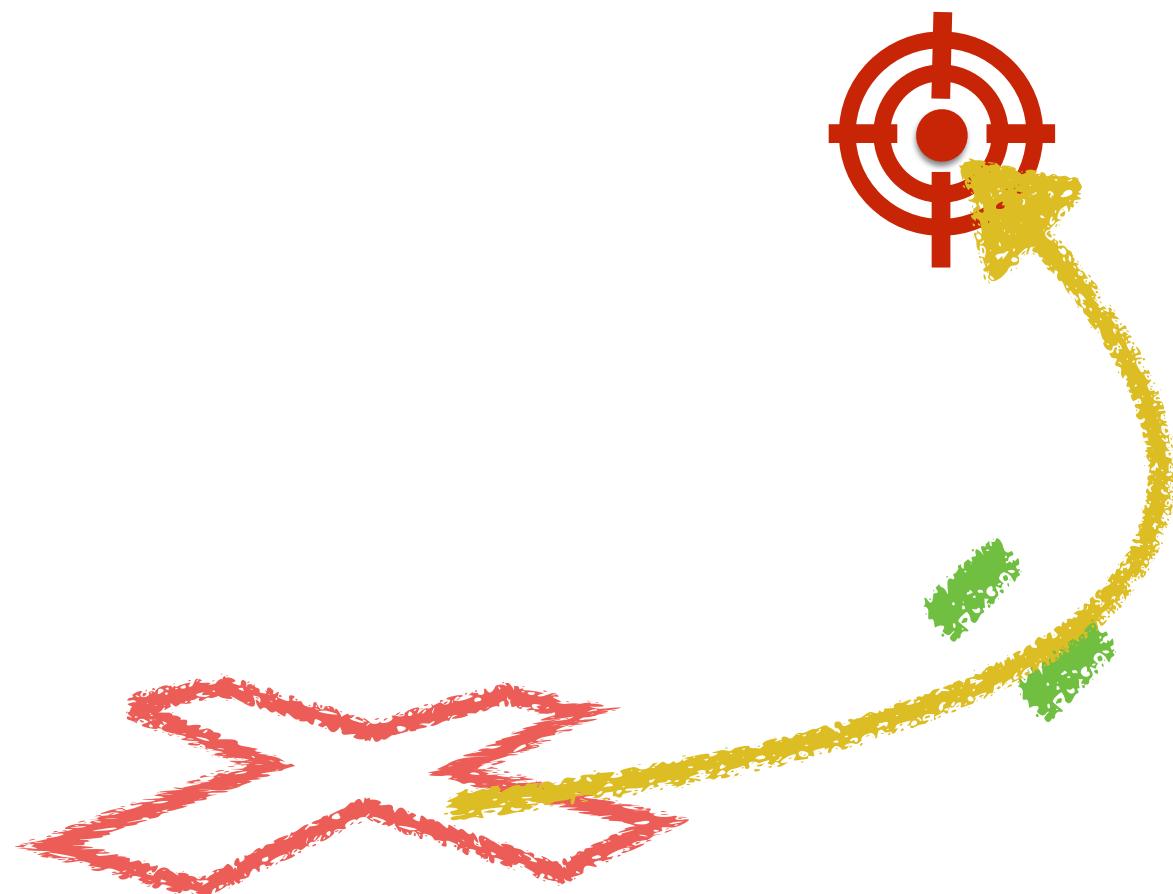
# Next: address the constraints



# Add additional features



# Finally: hit target



# Discussion

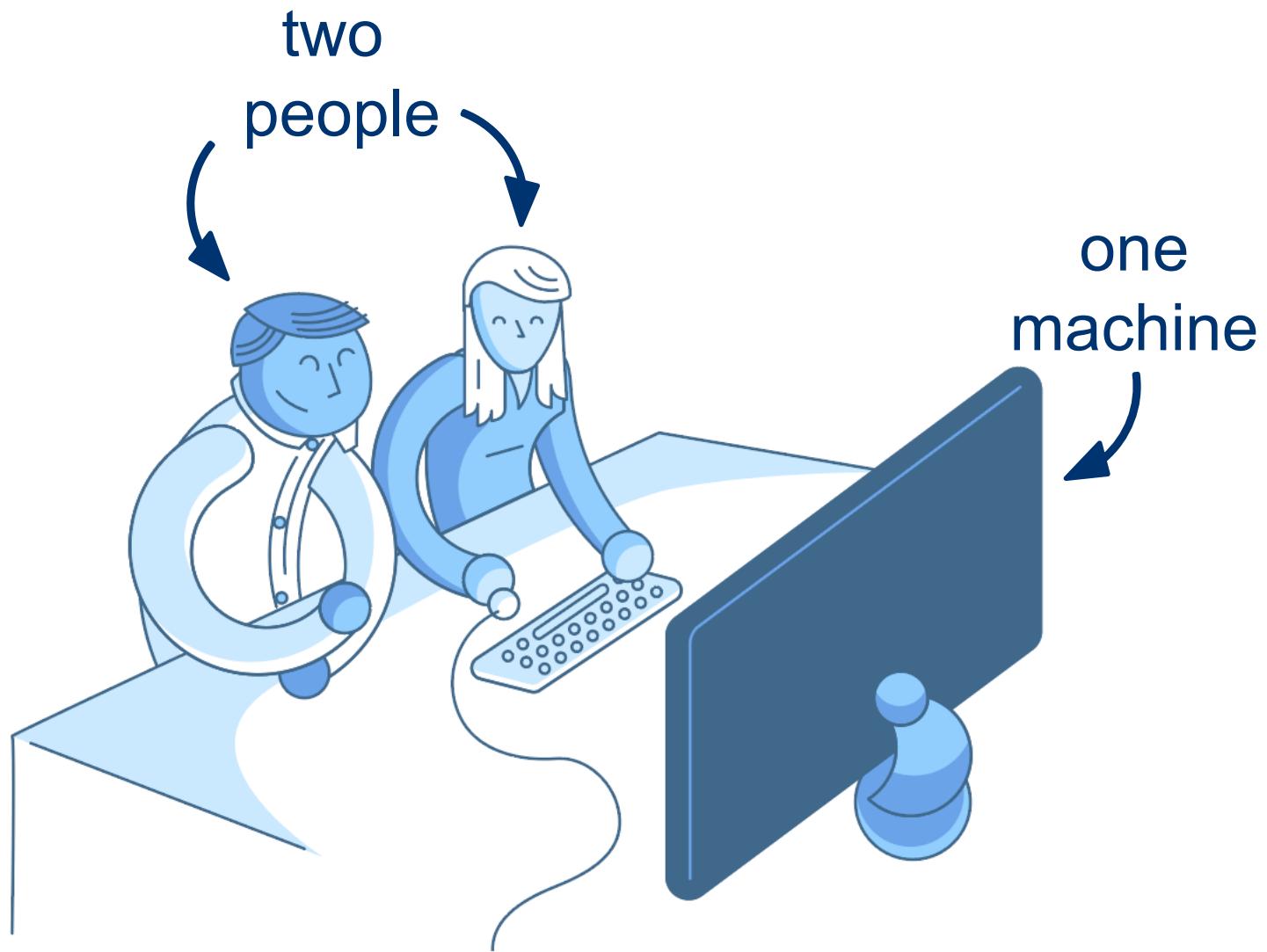
So what does this  
look like in practice?



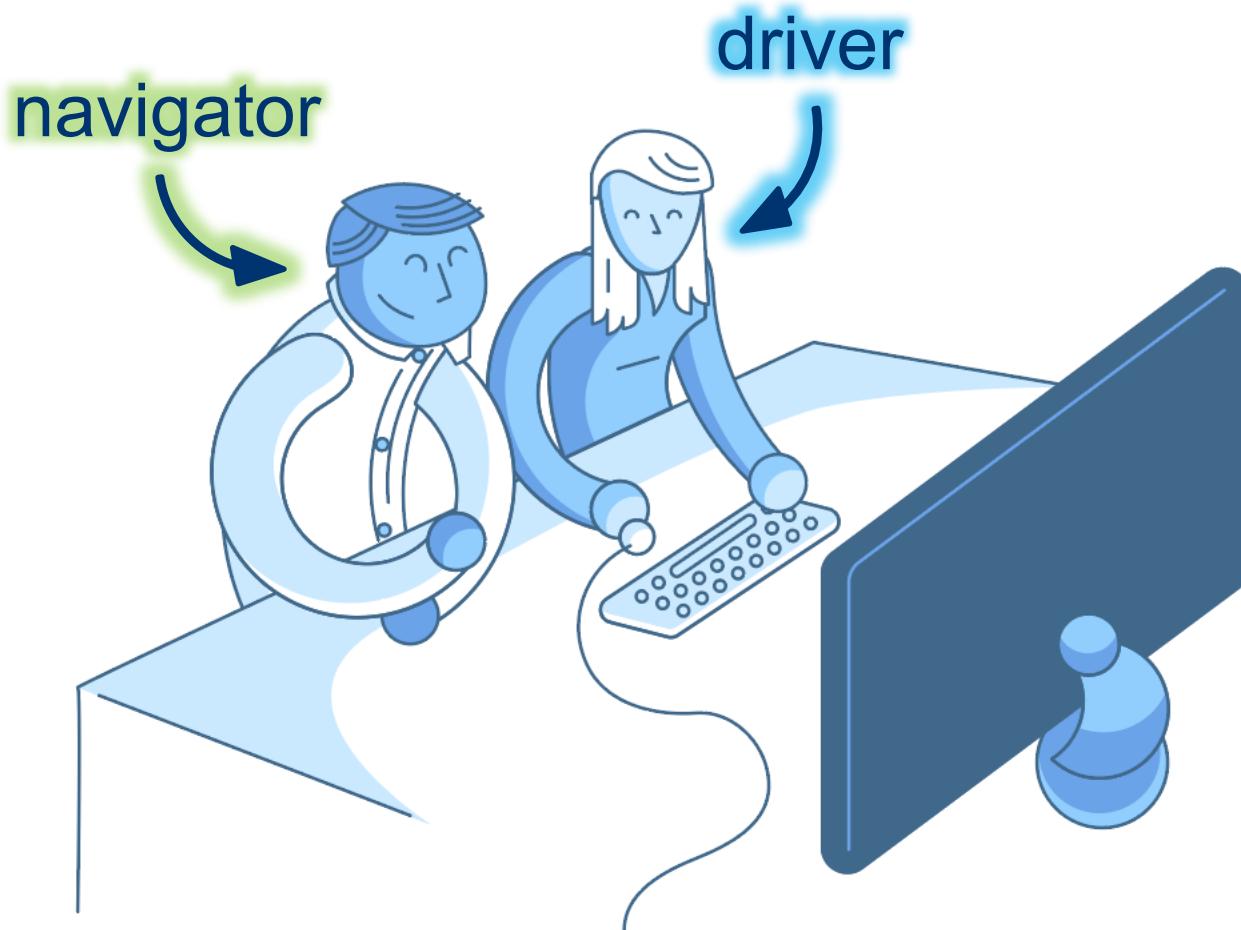
# A problematic (but common) model



# A better model: “pair programming”



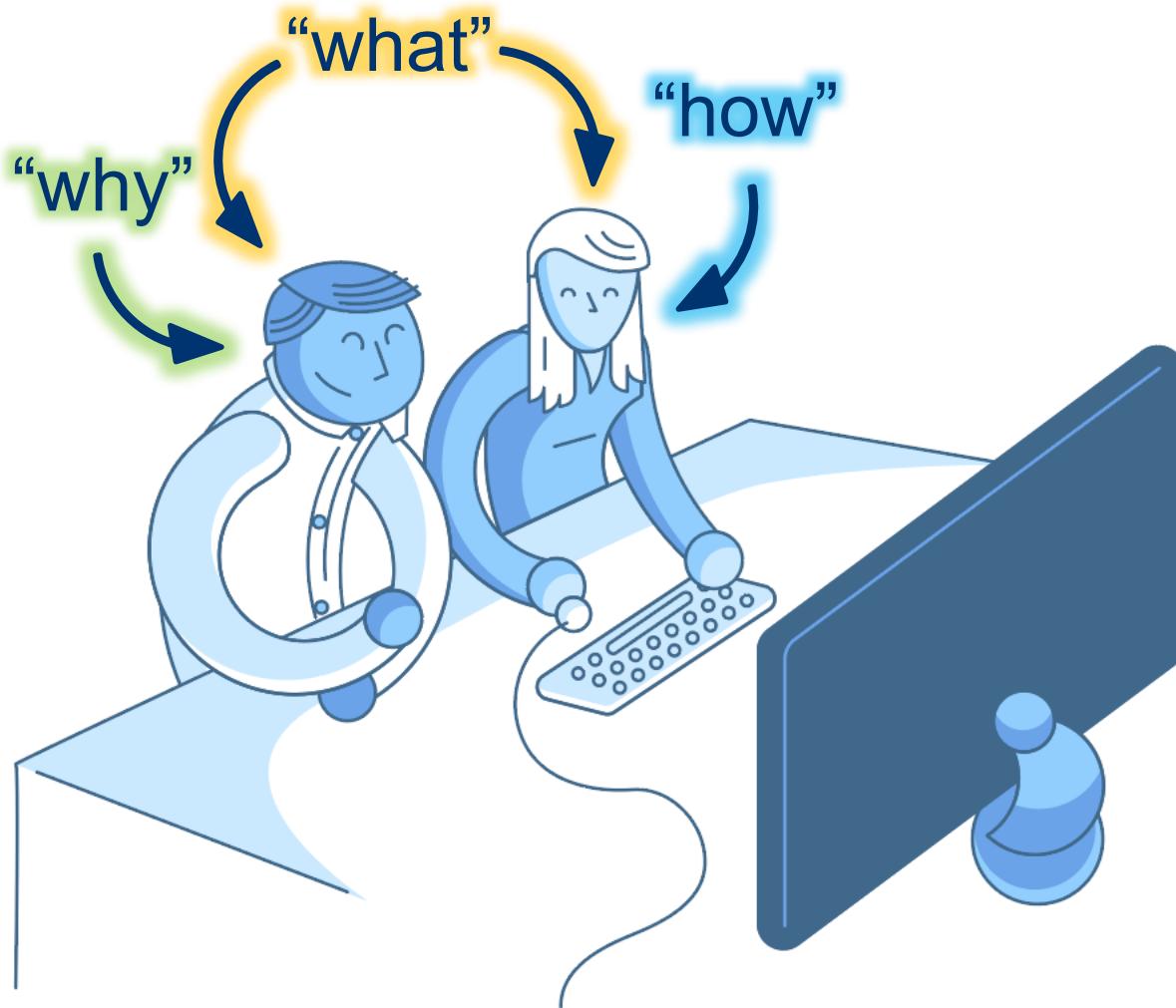
# Two complimentary roles



# A common analogy



# Navigator vs. driver: different focus



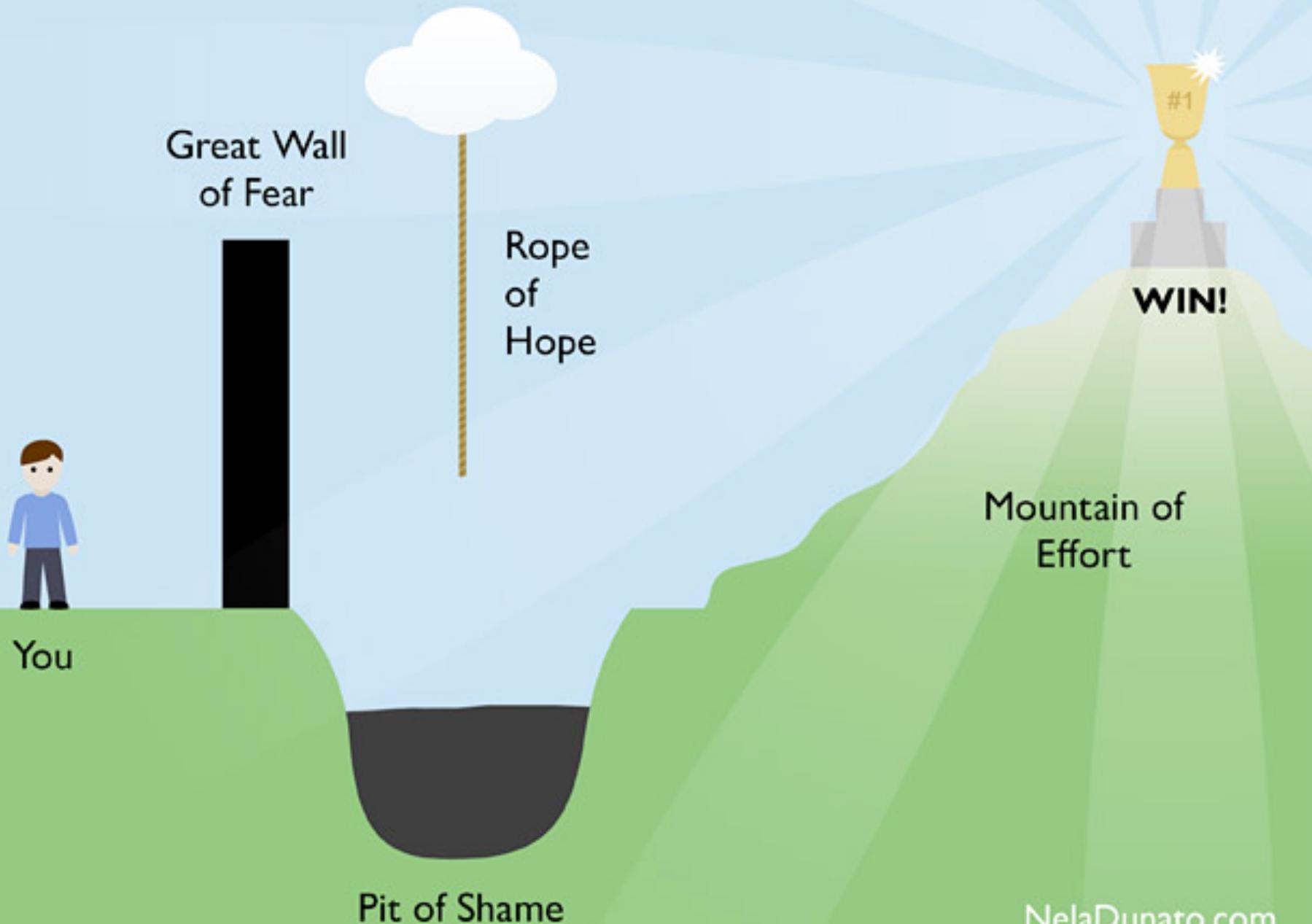
# A professional perspective (& some tips)



# Reality check

What makes this approach  
**so hard?**





# Overview of the week

- ✓ Crash course in computers
  - ✓ A little history
  - ✓ 4 key components
  - ✓ Quick hardware demo
  - ✓ Boolean logic
- ✓ Introduction to Python
- ✓ Life skill #1: pair programming
- Lab: Getting Started with Python
  - User Input

# Your first taste of pair programming!

A screenshot of a web browser window titled "Getting Started with Python". The URL in the address bar is <https://jcrouser.github.io/CSC111/labs/lab-1-intro.html>. The browser interface includes a toolbar with various icons and a status bar showing "Jordan" and the time "10:13". The main content area displays the "Getting Started with Python" page. On the left, a sidebar titled "Prerequisite: installing Python 3" lists several topics: Overview, Setting up, Playing with numbers, Numbers and Strings, Repeating Strings, Playing with Lists and Loops, Repeating several statements, Challenge of the Day, and Submission of Lab 1. The "Overview" item is highlighted with a blue background. The main content area features a large blue header "Getting Started with Python". Below it, a text block states: "This lab is just an introduction to having fun with `Python`. The original version was written by [Dominique Thiebaut](#) at Smith College, and it was adapted for this course in June 2018 by [R. Jordan Crouser](#)". Another text block explains the purpose of the lab: "The purpose of this lab is to give you a first taste of programming in `Python`, to give you space to explore the different systems that are available to you, and get a sense of how to program in an “intuitive” fashion. **Try not to worry too much about the details.** This is our first dive into programming, and we haven’t had time to fully understand what variables are, what strings are, or how the function `print()` works. Today is just a day for you to explore the tools, run some `Python` code in `Idle`, and submit a program using Moodle." A section titled "Prerequisite: installing Python 3" provides instructions for installing Python 3. The text reads: "If you haven’t done so already, you will need to install Python 3 on your laptop. Go to [www.python.org/downloads](http://www.python.org/downloads) and follow the instructions for downloading Python 3 to your laptop. If you get stuck, let the lab instructor or a TA know." The "Overview" section is also described in this text. At the bottom, a numbered list item 1 describes the console shell: "1. You can use the `console` (also known as the `shell`), where each line you type is interpreted by Python as soon as you hit `return`. The shell is what pops up when you first open IDLE, and it looks like this on a Mac: