MAKE SCHOOL

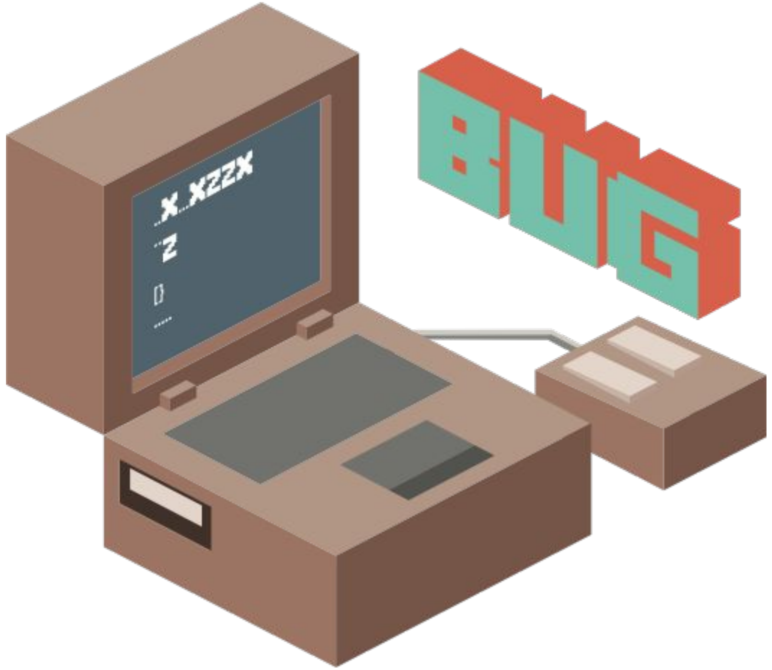
# Debugging 1: Steps to Debugging

A Guide to **Assuming *Your* Code is the Problem**

## Agenda

MAKE SCHOOL

- Learning Outcomes
- **Activity 1**: What is Debugging?
- Teacher Talk
- **Interactive Debugging Demo**
- Break
- **Activity 2**: Stepping Through



“The bug is not moving around in your code,
trying to evade you. It is just sitting in one place, **doing the wrong thing in the same way every time.**”

**- Nick Parlante, Debugging Zen**

## Learning Outcomes

By the end of today, you should be able to…

1. Compare and contrast the **pros and cons** of **different debugging techniques**.
2. **Use breakpoints** to debug code.
3. Apply debugging techniques in a project of their own.

## Activity 1: **What is Debugging?** (10 Minutes)

1. In a notebook, **write down** the following incomplete sentence:

**Debugging is ________________.**

**2. Fill in the blank**. What does debugging mean to *you*?

3. In a breakout group of 3, **discuss your answers**!

4. After **5 minutes**, we'll **share** our answers **with the class**.

MAKE SCHOOL

## Debugging is…

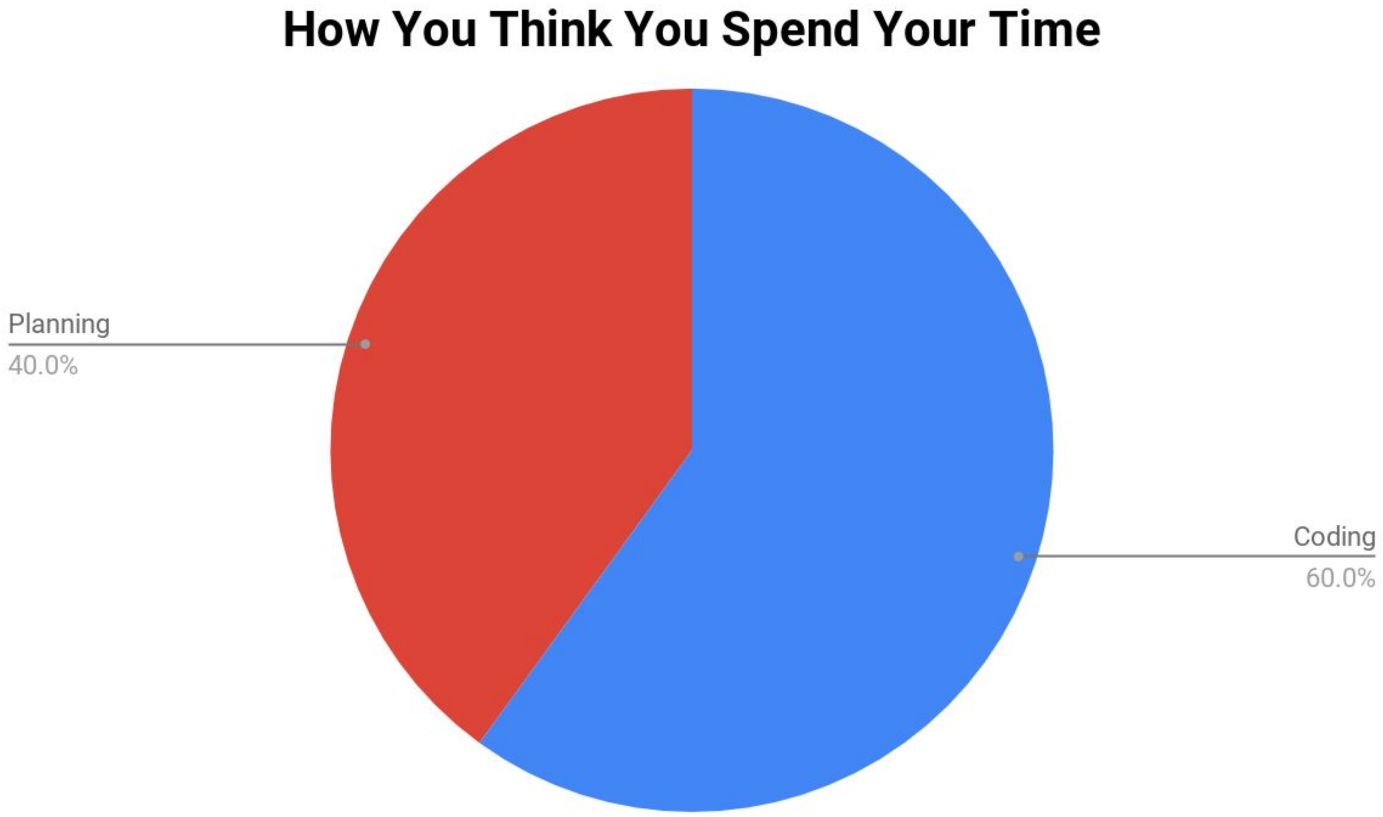
## Debugging is…

IMPORTANT!

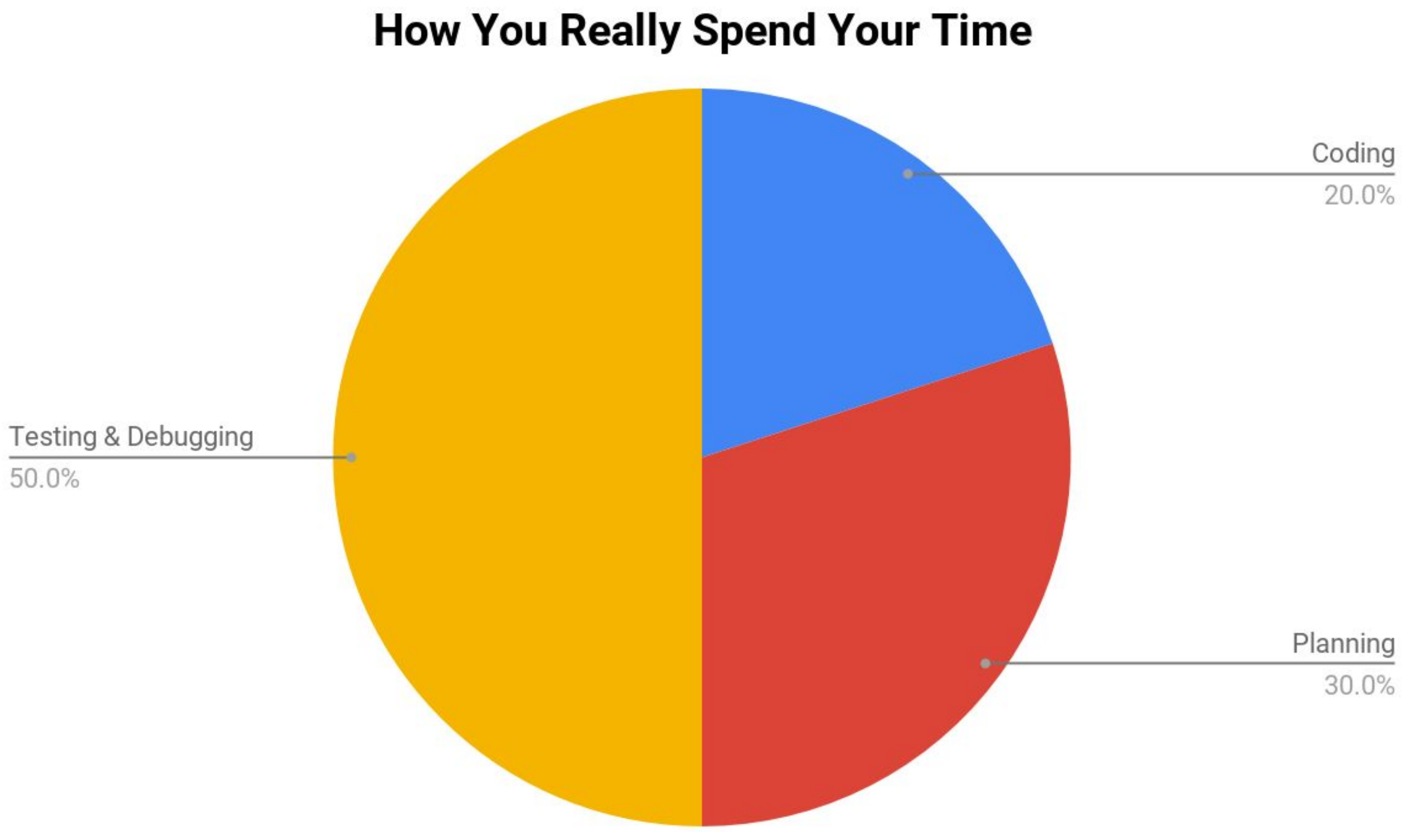
# **Debugging is…**

The **single most important skill** in programming.

# Debugging is… **a major time sink!**



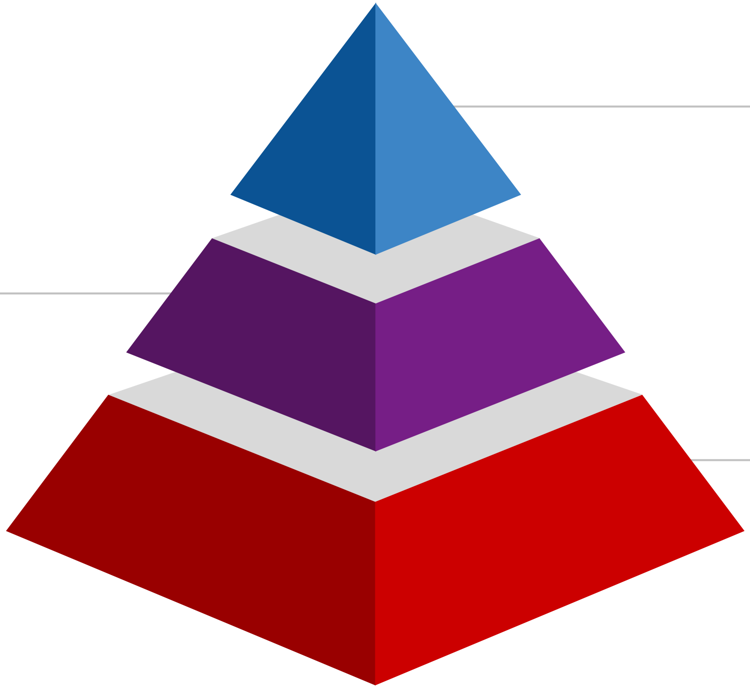
## Debugging is… **a major time sink!**



## Formal Definition

Debugging is the **process** of **finding and resolving** defects that **prevent correct operation** of computer software or a system.

# Forming Your Debugging Process



3 **Intuition**

Develop your intuition

2 **Experience**

Gain experience with code and tools

1 **Process**

The foundation of effective debugging

## How to Not Gain Debugging Skills

In order to gain experience, **first debug it yourself**.

- Don’t Jump to Google Search or Stackoverflow fo find the answer.
- Don’t ask your colleagues before debugging it yourself.

After you’ve exhausted all other ways, start getting help.

## Bugs Are Logical...

...so take a **systematic approach to solving bugs**!

1. Gather **Information**
2. **Replicate** the Issue
3. **Identify** the Culprit
4. Make a Change to **Fix It & Test Again**
5. **Mitigate** Future Occurrences

## Step 1: Gather Information

- **Expected vs Actual Behavior**
- Error Messages
- Stack Traces
- Screenshots
- **Environment Information**
  - Operating System
  - Browser
- Date & Time
- Logs

## Gather Information

*10 minutes*

Download today's starter code. For each problem, run the code and gather information about what went wrong:

- What is the **expected output** vs. **the actual output**?
- Is there an **error message** or **stack trace**? If so, **what information can we get from it**?
- How is this program **supposed to work**?

No need to actually solve the bugs yet! We'll do that in the next activity.

## Step 2: Replicate the Issue

Figure out how to

**reproduce the issue**

**with certainty**!

## Step 3: Identify the Culprit

- **Be methodical**
- **State all of your assumptions** and verify that they are true
- **Understand** the bug!

## Checking your Assumptions

When our code doesn't behave as expected, one of our **assumptions** about our code must be wrong.

Most of the time, bugs are caused by the assumptions **we didn't realize we were making**.

So, how do we find them?

## Checking your Assumptions

**What is the bug** in the following code? What **assumption** did the code's author make? Take a few minutes to think about your answer.

```
def find_largest_number(list_of_nums):
 largest_num = list_of_nums[0]
 for i in list_of_nums:
   if list_of_nums[i] > largest_num:
     largest_num = list_of_nums[i]
 return largest_num


answer = find_largest_number([3, 2, 1, 5, 4])
print(answer) # should print 5
```

## Checking your Assumptions

```
def find_largest_number(list_of_nums):
 largest_num = list_of_nums[0]
 for i in list_of_nums:
   if list_of_nums[i] > largest_num:
     largest_num = list_of_nums[i]
 return largest_num


answer = find_largest_number([3, 2, 1, 5, 4])
print(answer) # should print 5
```

The author made the **assumption** that the i variable contained the current list *index*, not the current list *value*.

## Checking your Assumptions

Sometimes, to verify your assumptions, you need to look outside your own code and **read the documentation** for the library code you're using.

- Does it work the way you think?
- What **inputs** does the library code take?
- What **outputs** does it return?

## Checking your Assumptions

**What is the bug** in the following code? What **assumption** did the code's author make? Take a few minutes to think about your answer.

```
def get_largest_num(list_of_nums):
  list_of_nums = list_of_nums.sort()
  last_index = len(list_of_nums) - 1
  return list_of_nums[last_index]


print(get_largest_num([5, 2, 17, 8, 3]))
```

# Checking your Assumptions

```
def get_largest_num(list_of_nums):
   list_of_nums = list_of_nums.sort()
   last_index = len(list_of_nums) - 1
   return list_of_nums[last_index]


print(get_largest_num([5, 2, 17, 8, 3]))
```

**TypeError: object of type 'NoneType' has no len()**

The **sort()** function sorts a list **in-place**, and returns **None**. The code's author assumed that **sort()** would return the newly sorted list.

MAKE SCHOOL

## Break - 10 min

## State your Assumptions

*20 minutes*

For each problem in the starter code, read through the code line-by-line and **state your assumptions** for the result of each line.

Then, **use print statements to verify whether your assumptions are correct**.

## Step 4: Fix it Right!

- **Attempt to Replicate Again**
  - Make sure the steps are written down!

- **No Temporary Workarounds**
  - Add **technical debt**
  - Tend to **introduce other issues**
  - **Rarely get replaced** later on with true solutions

## Step 5: Mitigate Future Occurrences

- Add an **automated test**
- **Share** your new knowledge
    - Project documentation
    - Blog post
    - StackOverflow (Ask a question and then answer it yourself)
- **Submit a patch**
    - Or merge your code upstream to master!

## Long Term Benefits

- Gain **experience**
- **Learn** how the system works
- Boost **confidence**

## Credits & Additional Resources

- Deck adapted from **Debugging Effectively** (2019)