Unit 10: Debugging

Title: Stimulate Motivation

Suggested Time: 2 mins

Materials / Handouts:
Whiteboard, computer, projector. Provide class slides for students to take notes on and unit glossary.

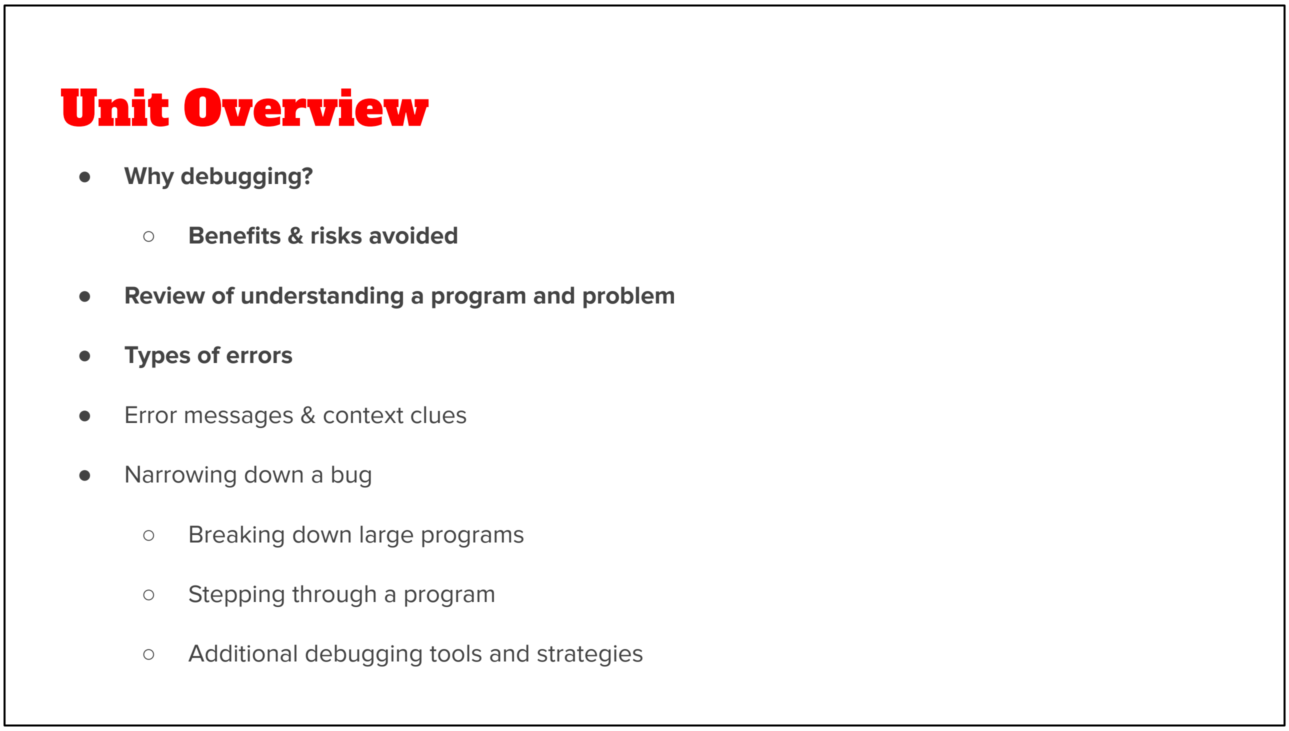
Description:
Class discussion of errors and bugs the students have encountered earlier in the course.

Instructor Actions:
Ask learners what errors they have faced.

Learner Actions:
Recall errors that they encountered in previous lessons.

Suggested Speaking Notes:
This is the 10th unit in this course. By now you've written many programs and have likely encountered many errors. What are some of the errors you have encountered?

Monday



Unit Overview

- **Why debugging?**
  - **Benefits & risks avoided**
- **Review of understanding a program and problem**
- **Types of errors**
- Error messages & context clues
- Narrowing down a bug
  - Breaking down large programs
  - Stepping through a program
  - Additional debugging tools and strategies

Title: Unit Overview

Suggested Time: 1 min

Materials / Handouts:

Description:
Discussion of what will be covered in the lesson.

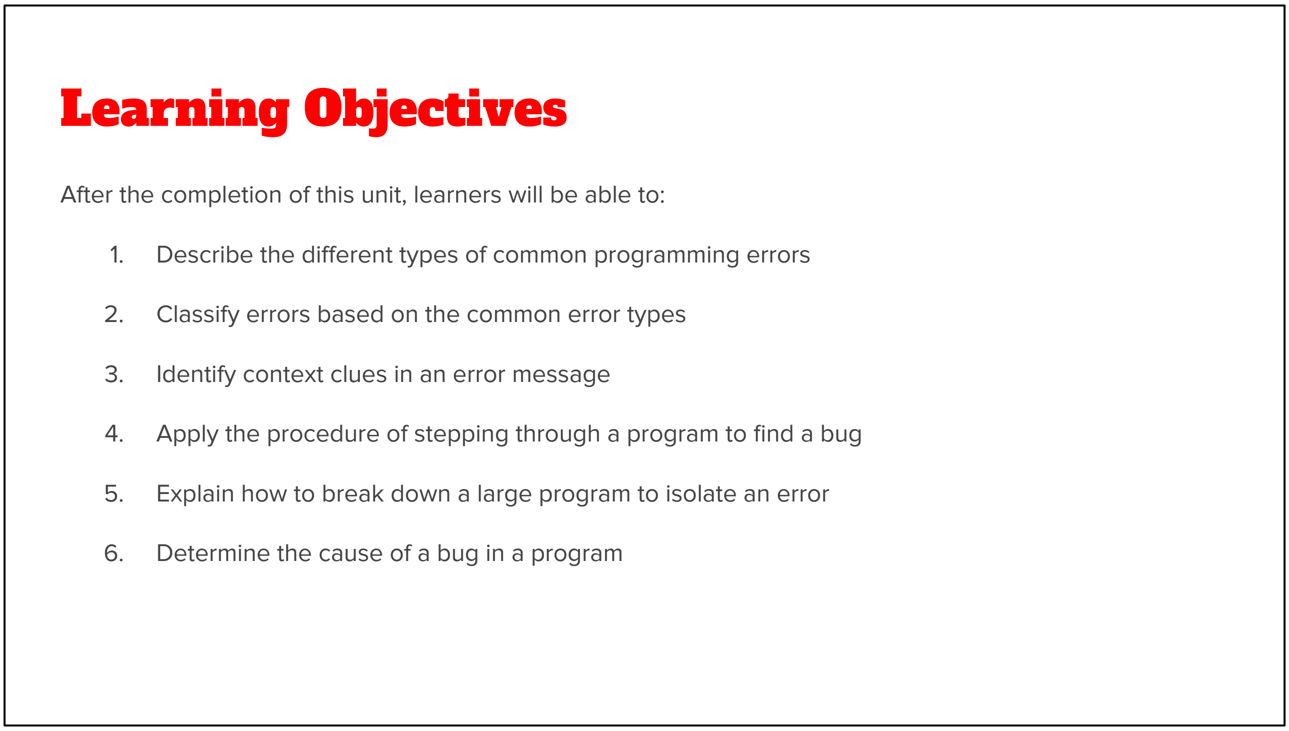
Instructor Actions:
Explain what will be covered in the unit and that the unit will take place over the course of a week.

Learner Actions:
Actively listen and ask clarifying questions as needed.

Suggested Speaking Notes:
"Here's what we will cover during this unit. Today (Monday), we'll start by talking about why debugging is important and do a little bit of review. Then we'll look at what different types of errors there are and how to identify them.

On Wednesday, we'll look more indepth at error messages and how to find and use context clues in the error messages.

Finally on Friday, we'll talk about how to break down problems and step through them. Then we'll finish off by briefly looking at additional tools and strategies that can be used for debugging."



Learning Objectives

After the completion of this unit, learners will be able to:

1. Describe the different types of common programming errors
2. Classify errors based on the common error types
3. Identify context clues in an error message
4. Apply the procedure of stepping through a program to find a bug
5. Explain how to break down a large program to isolate an error
6. Determine the cause of a bug in a program

Title: Learning Objectives

Suggested Time: 1 min

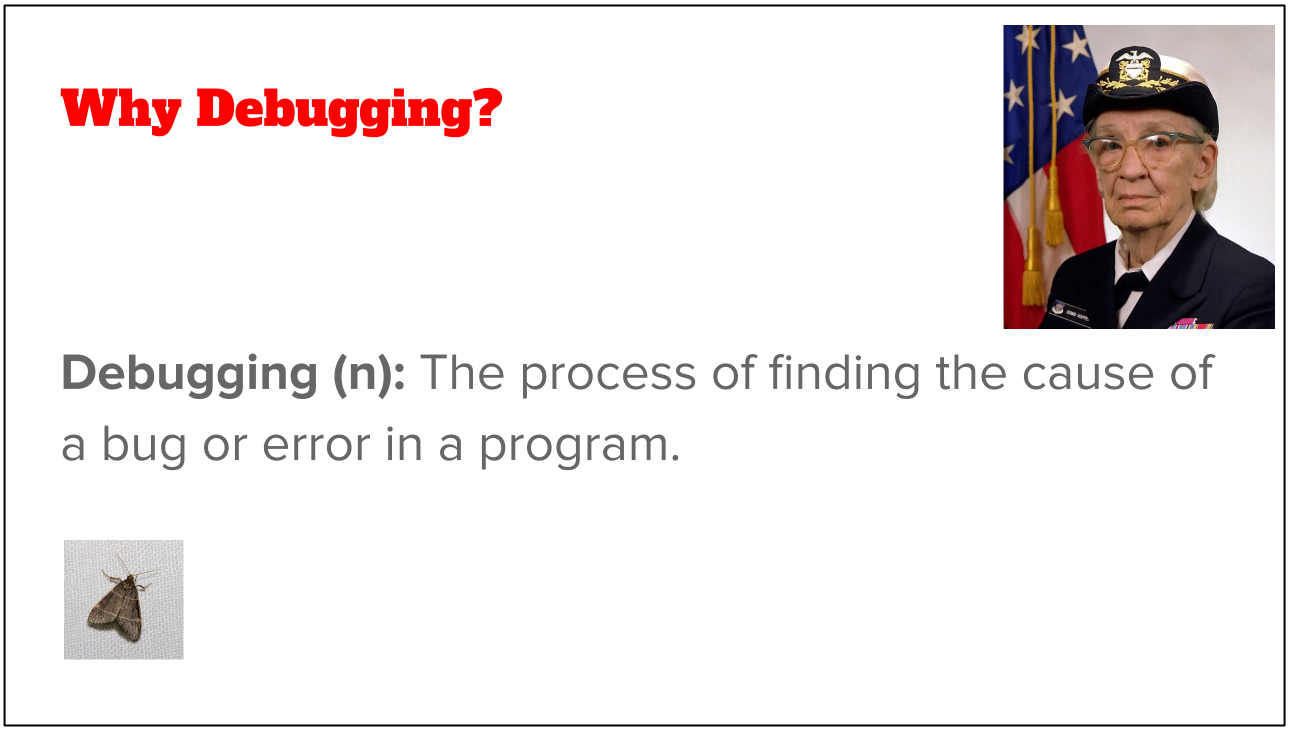
Materials / Handouts:

Description:
Present the terminal learning objectives for the unit.

Instructor Actions:
Share the terminal learning objective that students will be able to find and determine the cause of a bug in a program.

Learner Actions:
Ask clarifying questions about the terminal learning objective if needed.

Suggested Speaking Notes:
Here are the learning objectives for this unit. I'll give you a moment to read them. Let me know if you have any questions.



Why Debugging?

Debugging (n): The process of finding the cause of a bug or error in a program.

Title: Reasons for Learning & Benefits / Risks Avoided

Suggested Time: 2 mins

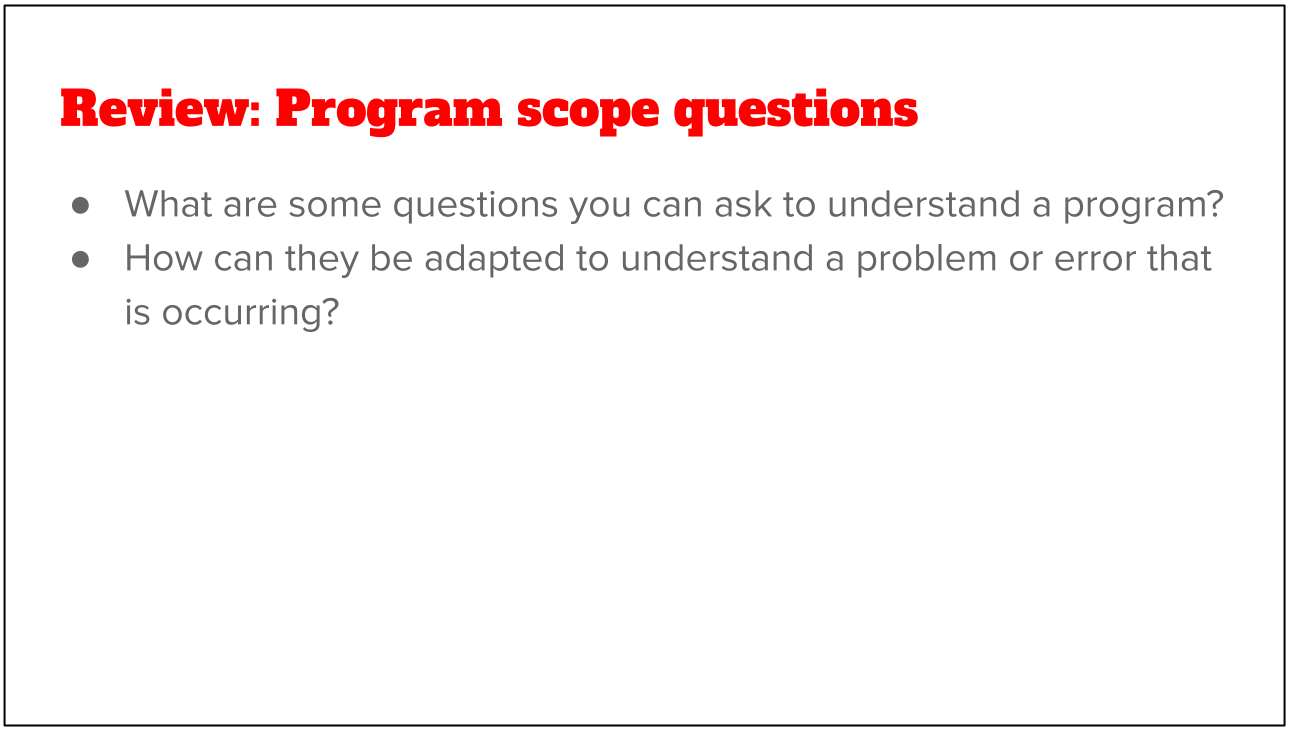
Materials / Handouts:
Whiteboard

Description:
Create a list of benefits of learning debugging strategies and a list of risks that are avoided by using debugging strategies effectively.

Instructor Actions:
Present the definition of debugging. Explain that the term comes from Admiral Grace Hopper who upon running into issues with a computer opened it up and found a bug inside. Ask the students about benefits and risks. Write the lists out on a whiteboard. If needed prompt students with some examples like saving time.

Learner Actions:
Share ideas of benefits and risks. Suggest revisions to the benefits list if ideas come up while creating the risks avoided list.

Suggested Speaking Notes:
Debugging is the process of finding a bug or error in a program. The term "debug" comes from Admiral Grace Hopper who upon running into issues with a computer opened it up and found a bug inside. Thinking about this definition, what are some benefits of learning how to debug? What are some risks that can be avoided by learning how to debug?



Review: Program scope questions

- What are some questions you can ask to understand a program?
- How can they be adapted to understand a problem or error that is occurring?

Title: Review / Recall prior knowledge

Suggested Time: 2 mins

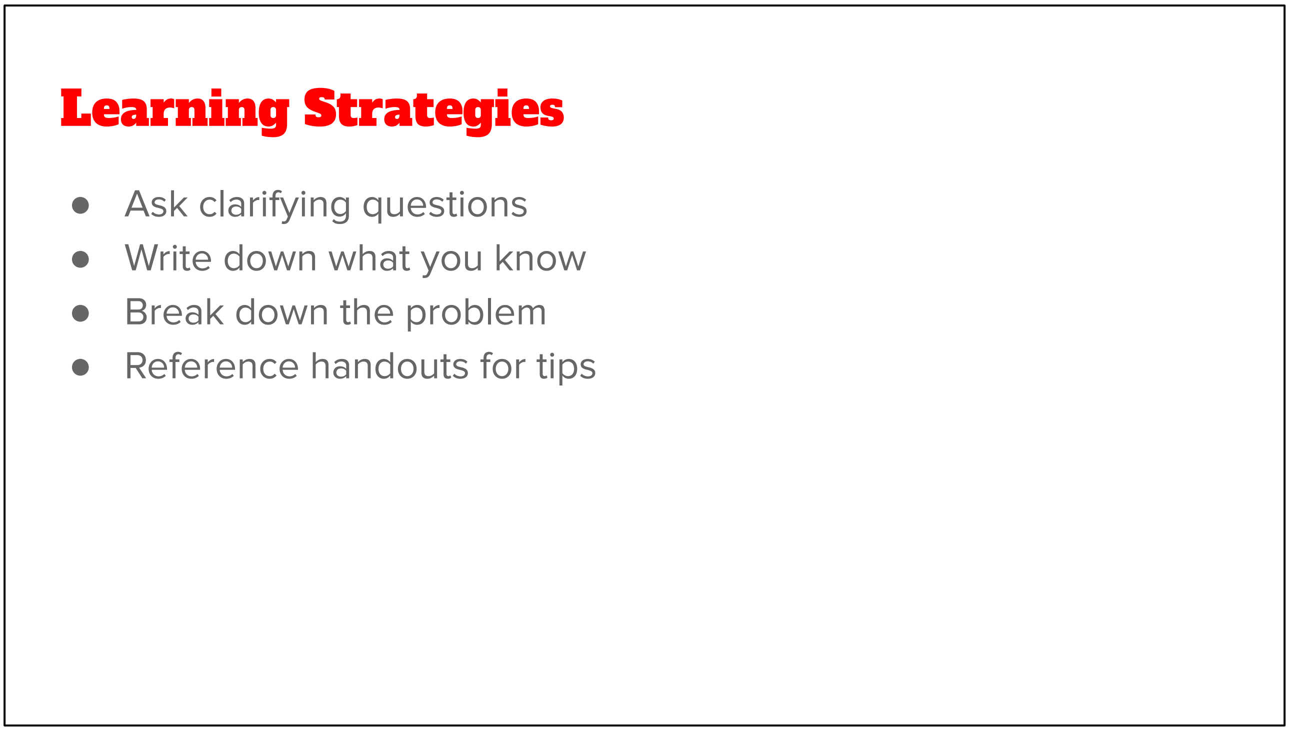
Materials / Handouts:
Whiteboard

Description:
Discussion of what was covered in previous lessons that this one will build off of.

Instructor Actions:
Ask students to think back to the design units and the questions they asked to understand the problem they were trying to solve. Ask how these questions might be modified to ask about bugs or errors. Create a list of questions that can be asked about an error.

Learner Actions:
Recall questions that can be asked when designing a program and share them with the class. Suggest ways the questions could be modified to ask about bugs or errors.

Suggested Speaking Notes:
That's a great list of reasons to debug but you're probably wondering now how to get started. Well, before you can find and solve an error you need to understand what the error looks like. So, we're going to think back on Unit 1 and the questions we had for understanding a problem. What are some of these questions and how might we modify them to ask questions about an error?



Learning Strategies

- Ask clarifying questions
- Write down what you know
- Break down the problem
- Reference handouts for tips

Title: Describe learning strategies

Suggested Time: 2 mins

Materials / Handouts:

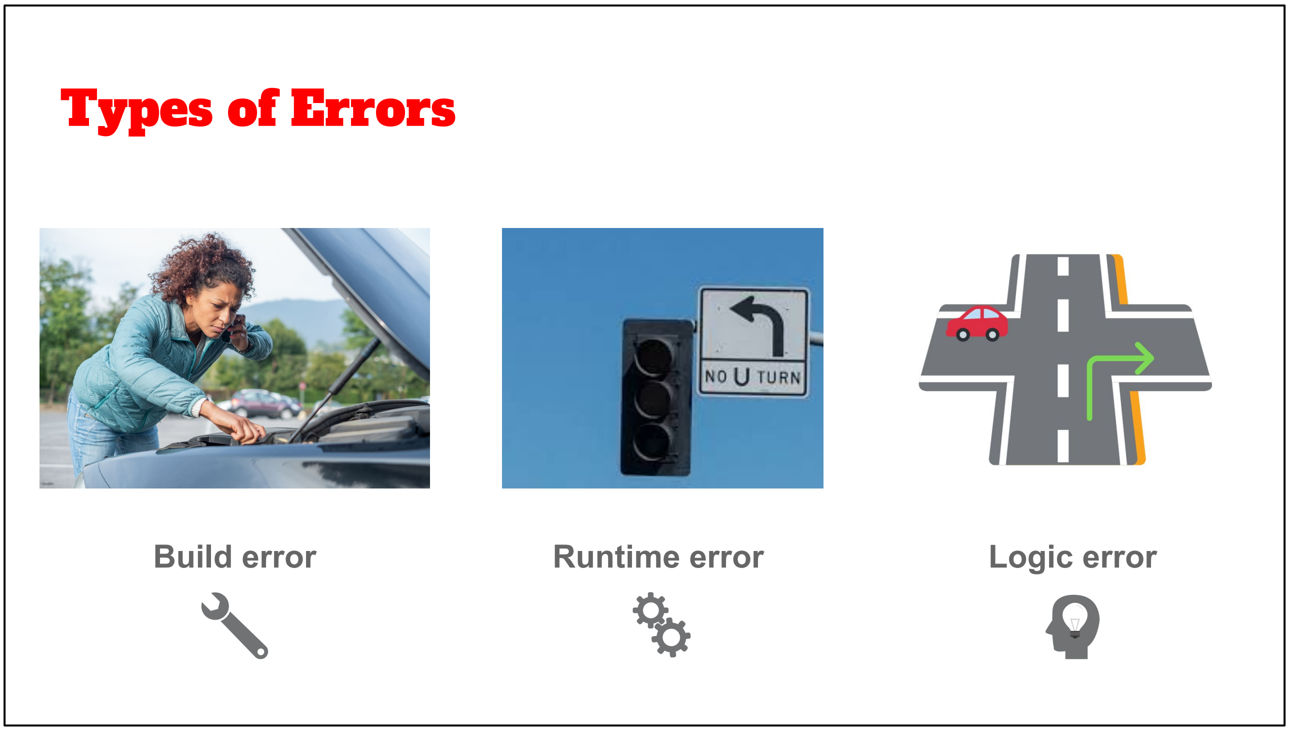
Description:
Discussion of strategies that can be used throughout the lesson, including asking clarifying questions, writing down what they know, and breaking things down into a smaller problem.

Instructor Actions:
Present different learning strategies and provide examples.

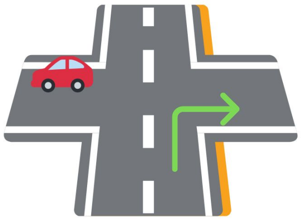
Learner Actions:
Listen and think about what learning strategies they already use and which ones they could try.

Suggested Speaking Notes:
Here are some learning strategies you can use throughout this lesson. They should look familiar from previous lessons. Are there any other learning strategies you already use that you could add to this list? What about

strategies you haven't used but might want to try?



Types of Errors



Build error

Runtime error

Logic error

Title: Describe what is new (to be learned)

Suggested Time: 1 min

Materials / Handouts:

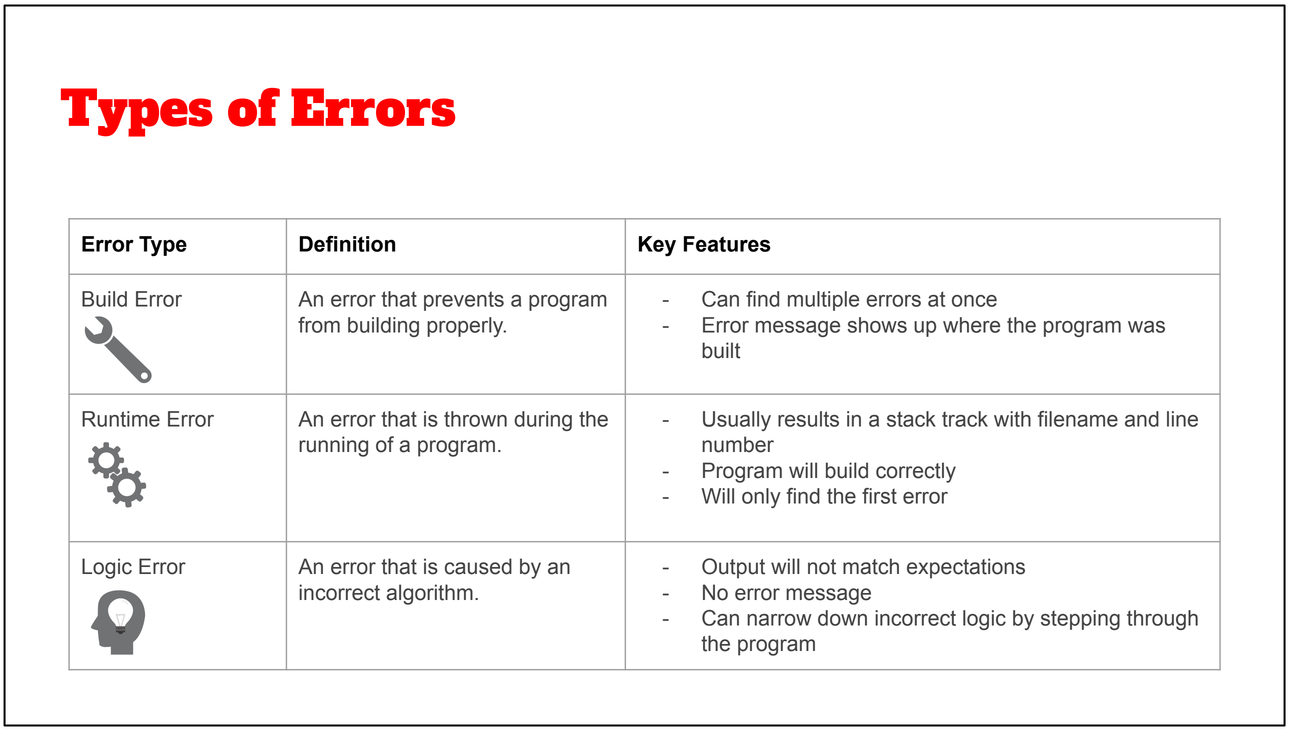
Description:
Discussion of what new material will be covered including the different types of errors that exist and strategies that can be used generally and for specific types of errors.

Instructor Actions:
Present an analogy that a program is like a car and there are different types of things that can go wrong when driving. For example, a build error is like your car not starting while a runtime error is like making a wrong turn because the directions are wrong.

Learner Actions:
Actively listen and ask clarifying questions as needed.

Suggested Speaking Notes:
For today's class we'll be talking about different types of errors and how to

recognize them. Knowing how to classify an error helps you know where to find the error. Let’s imagine a program you’re driving somewhere. There’s many different things that could go wrong. If your car won’t start? Or a traffic light isn’t working? Or imagine you turn left at a street but you were really supposed to turn right? Each of these may prevent you from getting to the correct destination, but they’re very different problems. These are like build, runtime, and logic errors.



Types of Errors

Error Type	Definition	Key Features
Build Error	An error that prevents a program from building properly.	- Can find multiple errors at once - Error message shows up where the program was built
Runtime Error	An error that is thrown during the running of a program.	- Usually results in a stack track with filename and line number - Program will build correctly - Will only find the first error
Logic Error	An error that is caused by an incorrect algorithm.	- Output will not match expectations - No error message - Can narrow down incorrect logic by stepping through the program

Title: Teach declarative knowledge

Suggested Time: 3 mins

Materials / Handouts:
Types of errors handout

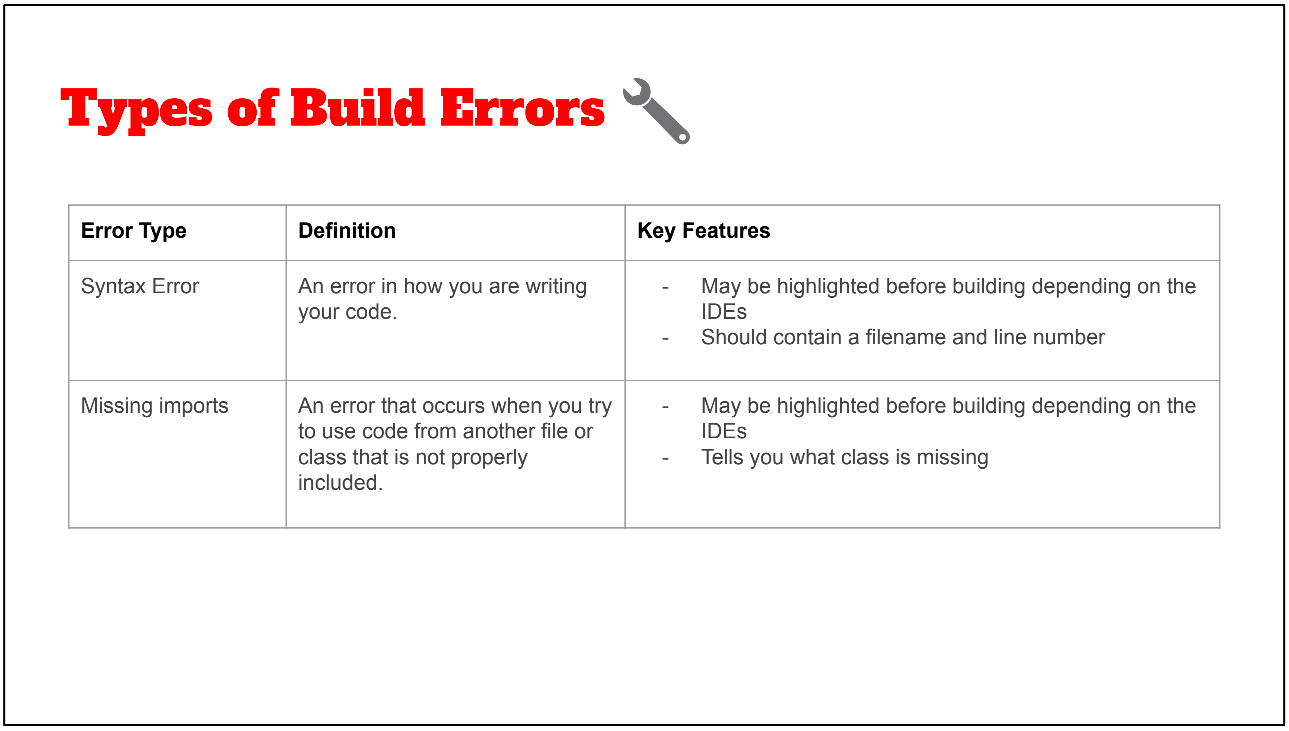
Description:
Discussion of types of errors and the key attributes to identify them.

Instructor Actions:
Present definitions and key attributes for each type of error. Present a simple example for each type of error. Answer any questions students might have.

Learner Actions:
Reference learning aid on the different types or errors, take notes on additional information, and ask clarifying questions as needed.

Suggested Speaking Notes:
There are many different types of errors that can show up in a program. You can find more details about all the errors we're going to talk about on the error types handout, but these generally fall into three categories of errors, build

errors, runtime errors, and logic errors. The main difference between these three types of errors is when they show up. Build errors show up when you try to build the program and prevent you from even running the program. Runtime errors show up after you have built the program and prevent a run of the program from completing. Logic errors may cause runtime errors but are caused by an incorrect assumption in program algorithm. This may mean the code doesn’t match the designed algorithm or that the designed algorithm does not correctly solve the problem it’s designed to solve.



Types of Build Errors

Error Type	Definition	Key Features
Syntax Error	An error in how you are writing your code.	- May be highlighted before building depending on the IDEs - Should contain a filename and line number
Missing imports	An error that occurs when you try to use code from another file or class that is not properly included.	- May be highlighted before building depending on the IDEs - Tells you what class is missing

Title: Teach declarative knowledge

Suggested Time: 2 mins

Materials / Handouts:
Types of errors handout

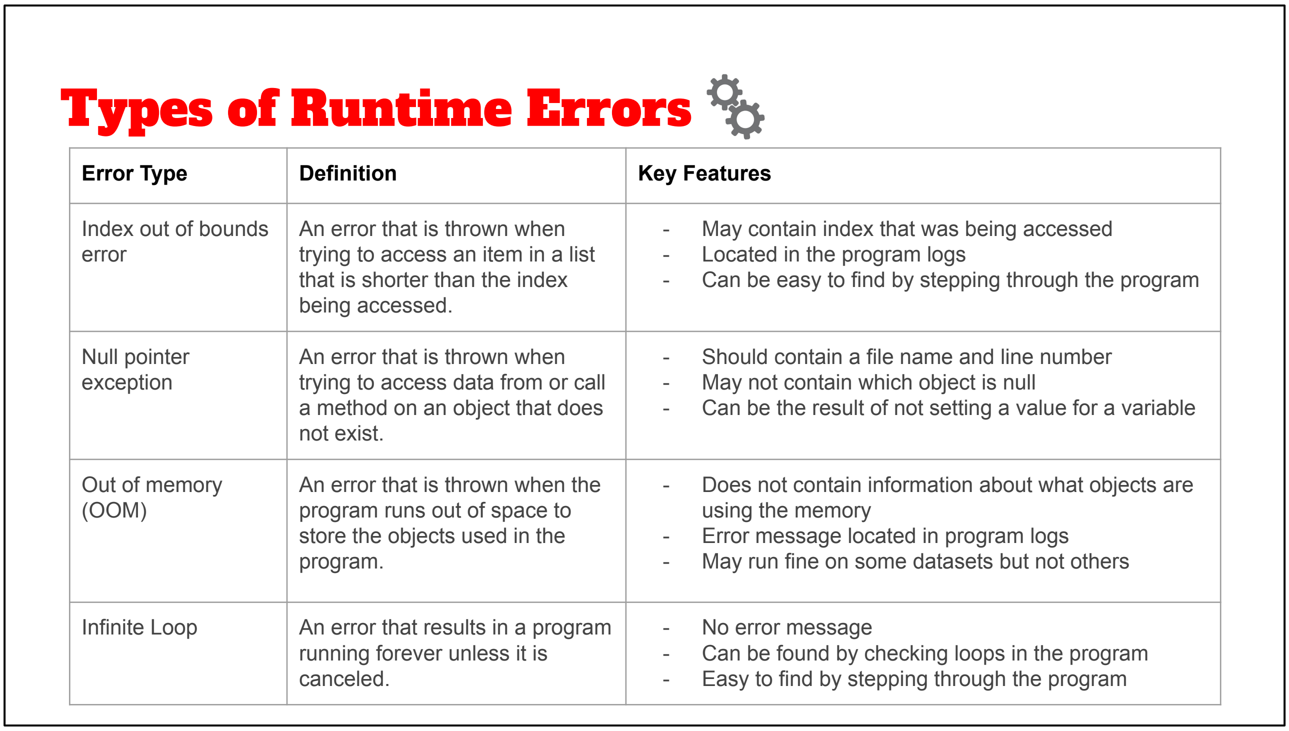
Description:
Discussion of types of errors and the key attributes to identify them.

Instructor Actions:
Present definitions and key attributes for each type of error. Present a simple example for each type of error. Answer any questions students might have.

Learner Actions:
Reference learning aid on the different types or errors, take notes on additional information, and ask clarifying questions as needed.

Suggested Speaking Notes:
Here are a couple of examples of build errors. A syntax error is an error in how you are writing a code. This can be something as simple as a typo or something loop that is missing an end condition. Another common example is

a missing import. This shows up when you try to use a class that is defined in another file but haven’t “imported” or included in the code.



Types of Runtime Errors

Error Type	Definition	Key Features
Index out of bounds error	An error that is thrown when trying to access an item in a list that is shorter than the index being accessed.	- May contain index that was being accessed - Located in the program logs - Can be easy to find by stepping through the program
Null pointer exception	An error that is thrown when trying to access data from or call a method on an object that does not exist.	- Should contain a file name and line number - May not contain which object is null - Can be the result of not setting a value for a variable
Out of memory (OOM)	An error that is thrown when the program runs out of space to store the objects used in the program.	- Does not contain information about what objects are using the memory - Error message located in program logs - May run fine on some datasets but not others
Infinite Loop	An error that results in a program running forever unless it is canceled.	- No error message - Can be found by checking loops in the program - Easy to find by stepping through the program

Title: Teach declarative knowledge

Suggested Time: 2 mins

Materials / Handouts:
Types of errors handout

Description:
Discussion of types of errors and the key attributes to identify them.

Instructor Actions:
Present definitions and key attributes for each type of error. Present a simple example for each type of error. Answer any questions students might have.

Learner Actions:
Reference learning aid on the different types or errors, take notes on additional information, and ask clarifying questions as needed.

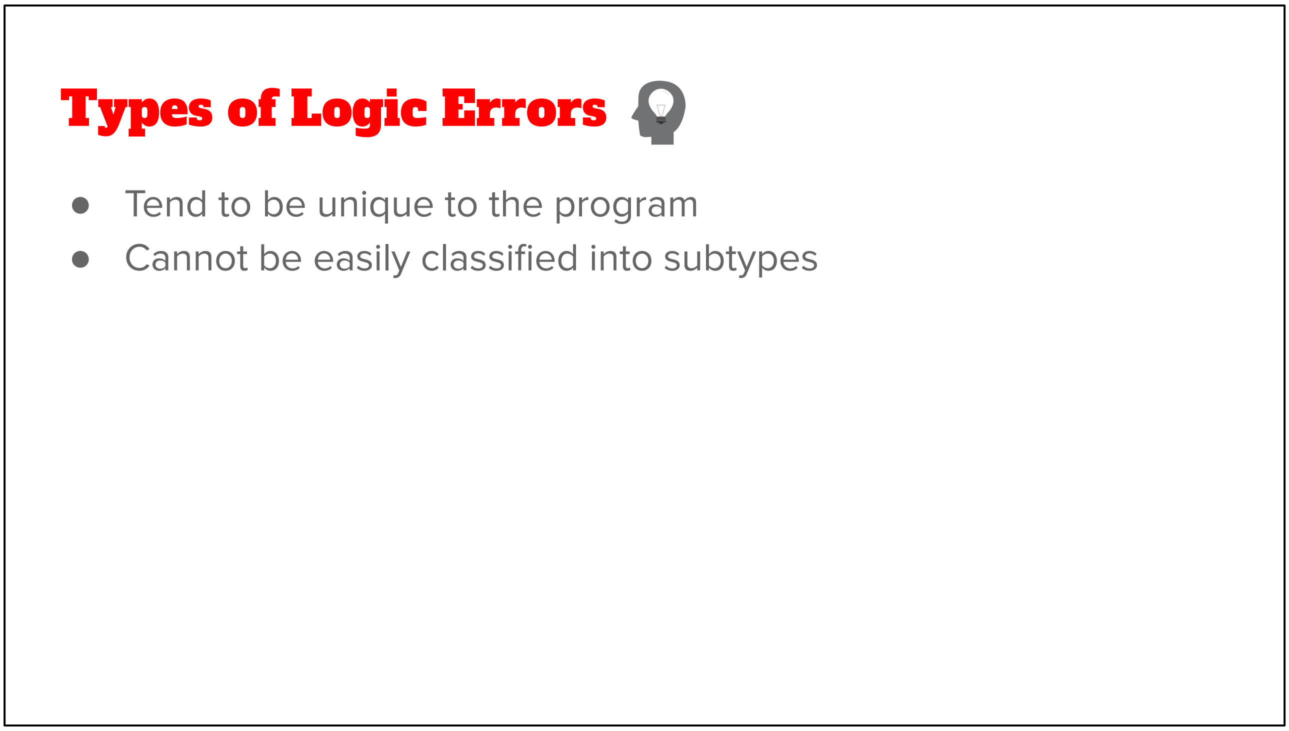
Suggested Speaking Notes:
Here are a couple of examples of runtime errors. An index out of bounds error means you're trying to get an item from a list but the list doesn't have that many items. For example, you might be trying to get the 4th item in a list that

only has 3 items.

A null pointer exception means you’re trying to perform an action on an object that doesn’t exist. Imagine trying to get the pay for something with a gift card but money as never added to the gift card.

An out of memory error means that a program has run out of space. This usually means you’re creating a lot of objects in your program and might not be deleting them.

An infinite loop is an error that shows up when you have a loop that never stops. This means your program will just keep running until you cancel it. Imagine wanting to count a group of things but never moving to the next object.



Types of Logic Errors

- Tend to be unique to the program
- Cannot be easily classified into subtypes

Title: Teach declarative knowledge

Suggested Time: 1 min

Materials / Handouts:
Types of errors handout

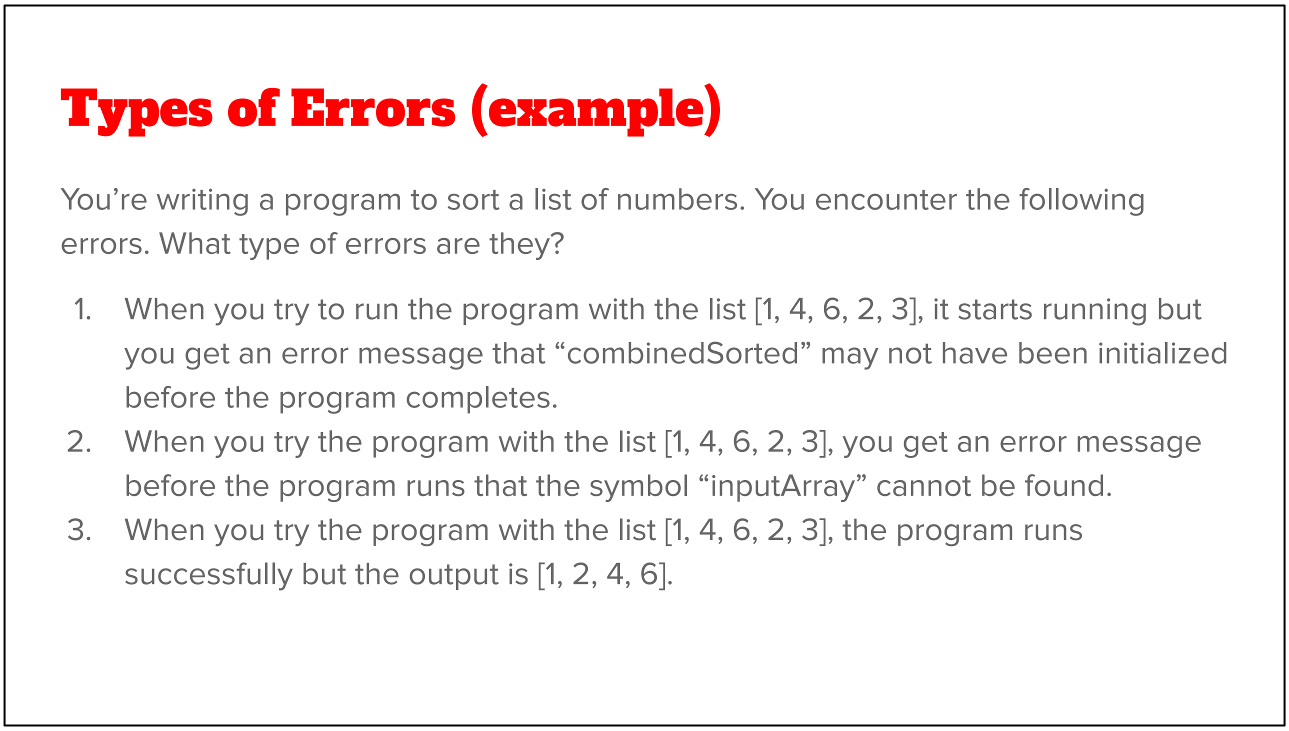
Description:
Discussion of types of errors and the key attributes to identify them.

Instructor Actions:
Explain that logic errors tend to be unique to the program and problem that is being solved. That they are not easily classified into subtypes and so there is no table for this slide. Answer any questions students might have.

Learner Actions:
Reference learning aid on the different types or errors, take notes on additional information, and ask clarifying questions as needed.

Suggested Speaking Notes:
The third type of error, logic errors, tends to be unique to the problem you're trying to solve and cannot easily be split into subtypes of errors. So, we don't

have a nice table here. The best way to describe these errors is that they come from thinking about the problem wrong. Maybe there’s some edge case you didn’t think of or there is some assumption that you’re making about the problem that isn’t always true.



Types of Errors (example)

You're writing a program to sort a list of numbers. You encounter the following errors. What type of errors are they?

1. When you try to run the program with the list [1, 4, 6, 2, 3], it starts running but you get an error message that "combinedSorted" may not have been initialized before the program completes.
2. When you try the program with the list [1, 4, 6, 2, 3], you get an error message before the program runs that the symbol "inputArray" cannot be found.
3. When you try the program with the list [1, 4, 6, 2, 3], the program runs successfully but the output is [1, 2, 4, 6].

Title: Demonstrate procedures ("how to")

Suggested Time: 5 mins

Materials / Handouts:
types of errors handout

Description:
Instructor presents example descriptions of errors and works with the group to classify each example.

Instructor Actions:
Read each example and make connections to key features of error types to classify examples.

Learner Actions:
Follow along with the instructor using the types of errors handout.

Suggested Speaking Notes:
Now let's walk through a couple of examples and see how they would be classified.
(runtime, build, logic)

Small Group Activity

Using the descriptions of the errors your group has received and your error types handout, discuss what type of errors you think they are and why. Be prepared to report back to the class.

Title: Provide practice and feedback

Suggested Time: 10 mins

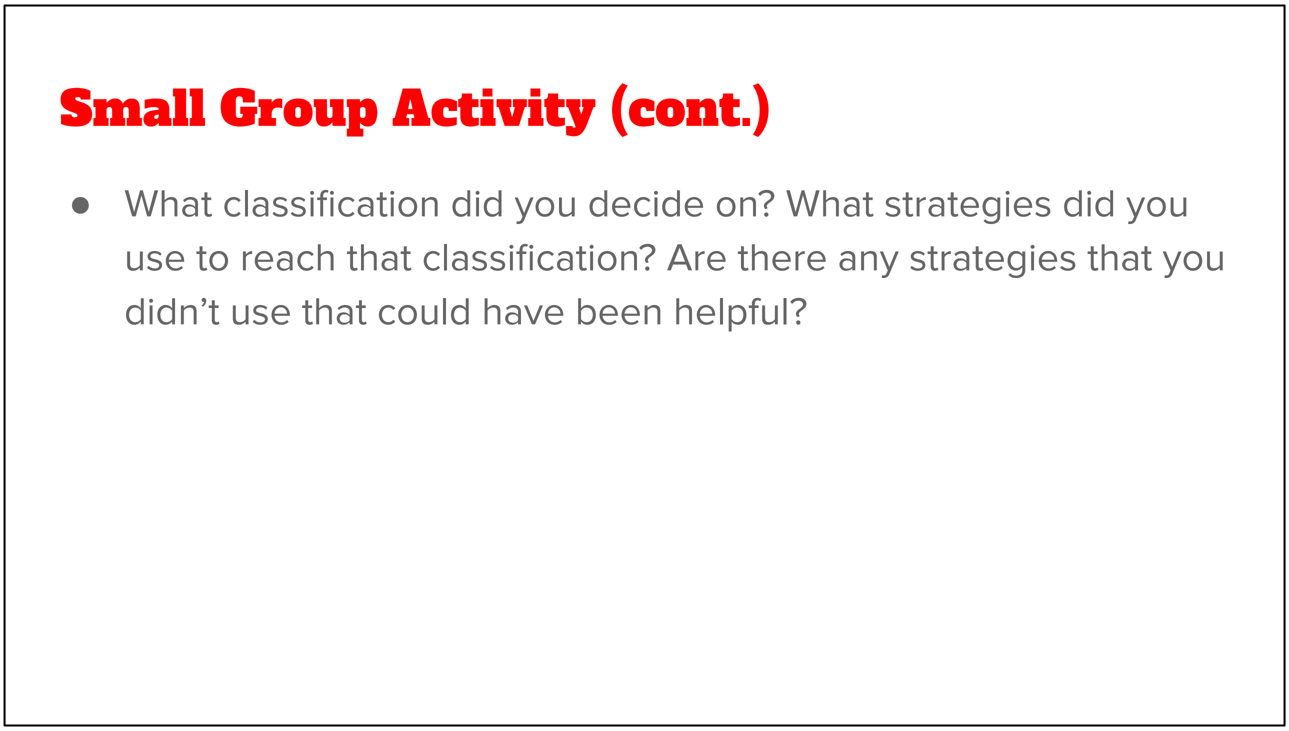
Materials / Handouts:
Example problems, types of errors handout

Description:
After definitions and key attributes are presented students will break up into small groups and be given example problems to classify.

Instructor Actions:
Provide sample problems for small group work. Observe students working to solve small group activities and provide feedback on what learning strategies they are using well and which ones to try if the learners get stuck.

Learner Actions:
During small group work, discuss what type of error they think the example problem is and why. Recall questions from previous lessons to help understand and classify the error. Report back to the class what problem they had, what type of error they classified it as, and how they made the decision.

Suggested Speaking Notes:
Now we’re going to break up into small groups and try this on our own.



Small Group Activity (cont.)

- What classification did you decide on? What strategies did you use to reach that classification? Are there any strategies that you didn't use that could have been helpful?

Title: Retention and Transfer

Suggested Time: 3 mins

Materials / Handouts:
Example problems, types of errors handout

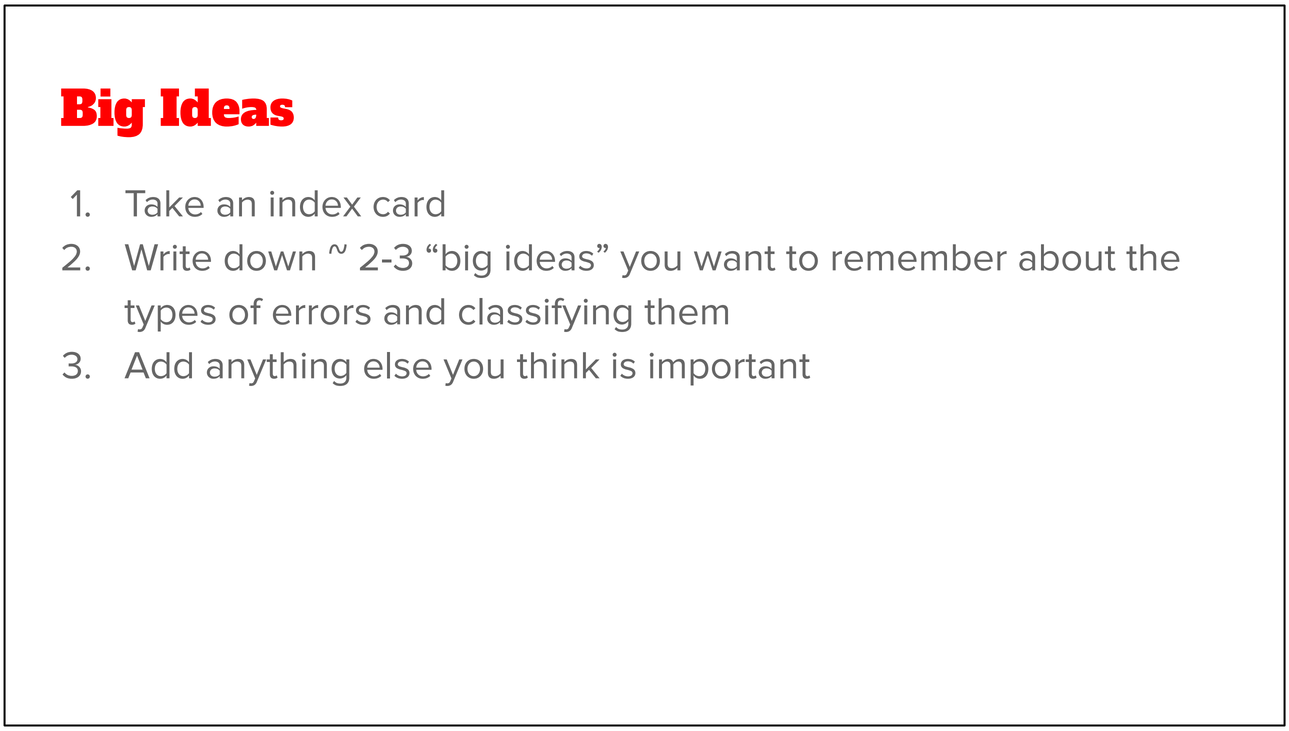
Description:
After each small group activity, students will share what strategies they used and the cues that helped them decide to use these strategies. This will help identify features of novel problems the strategies can transfer to.

Instructor Actions:
Ask students to share what cues lead to their use of different strategies and provide feedback on additional cues or strategies.

Learner Actions:
Recall what strategies they used during small group work and the cues that lead to the use of the strategies. Listen to the cues and strategies used by their classmates and consider which ones they could use in the future.

Suggested Speaking Notes:

Okay, now we’re going to report back to the group. Who wants to share what their group came up with and how you reached that classification?



Big Ideas

1. Take an index card
2. Write down ~ 2-3 “big ideas” you want to remember about the types of errors and classifying them
3. Add anything else you think is important

Title: Big Ideas

Suggested Time: 3 mins

Materials / Handouts:
Index cards

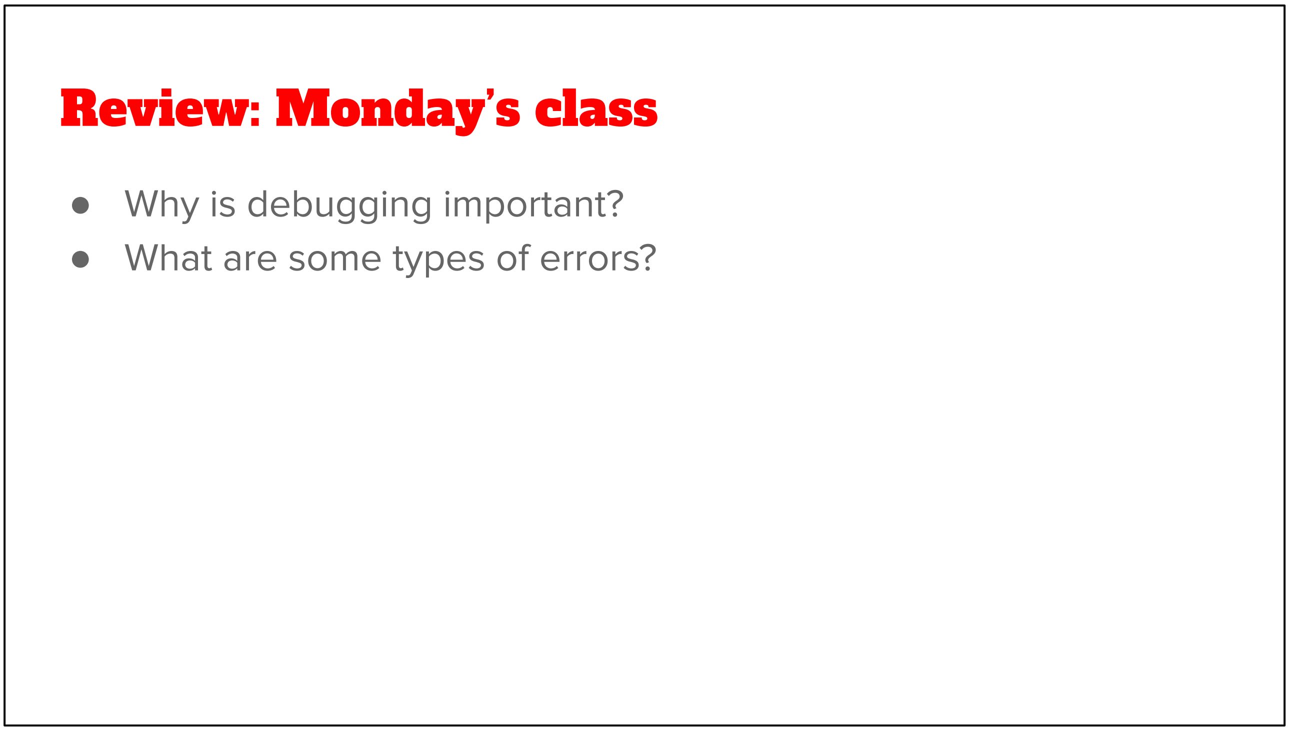
Description:
Students create reference guides of big ideas to remember when debugging problems in the future.

Instructor Actions:
Provide students with index cards or pieces of paper to create reference guides. Ask learners to think about what big ideas they would want to remember next time they have an error with one of their programs and need to debug it.

Learner Actions:
Recall key concepts they want to remember and write them on their reference guide. Share ideas with the class and incorporate new ideas they like into their reference guide.

Suggested Speaking Notes:
You’re going to need to be able to use what we’ve covered today in the future so think about what big ideas you want to remember and write them down on your index card.

Wednesday



Review: Monday's class

- Why is debugging important?
- What are some types of errors?

Title: Stimulate Motivation / Benefits & Risks Avoided / Recall Prior Knowledge

Suggested Time: 6 mins

Materials / Handouts:
Slides/notes from last class

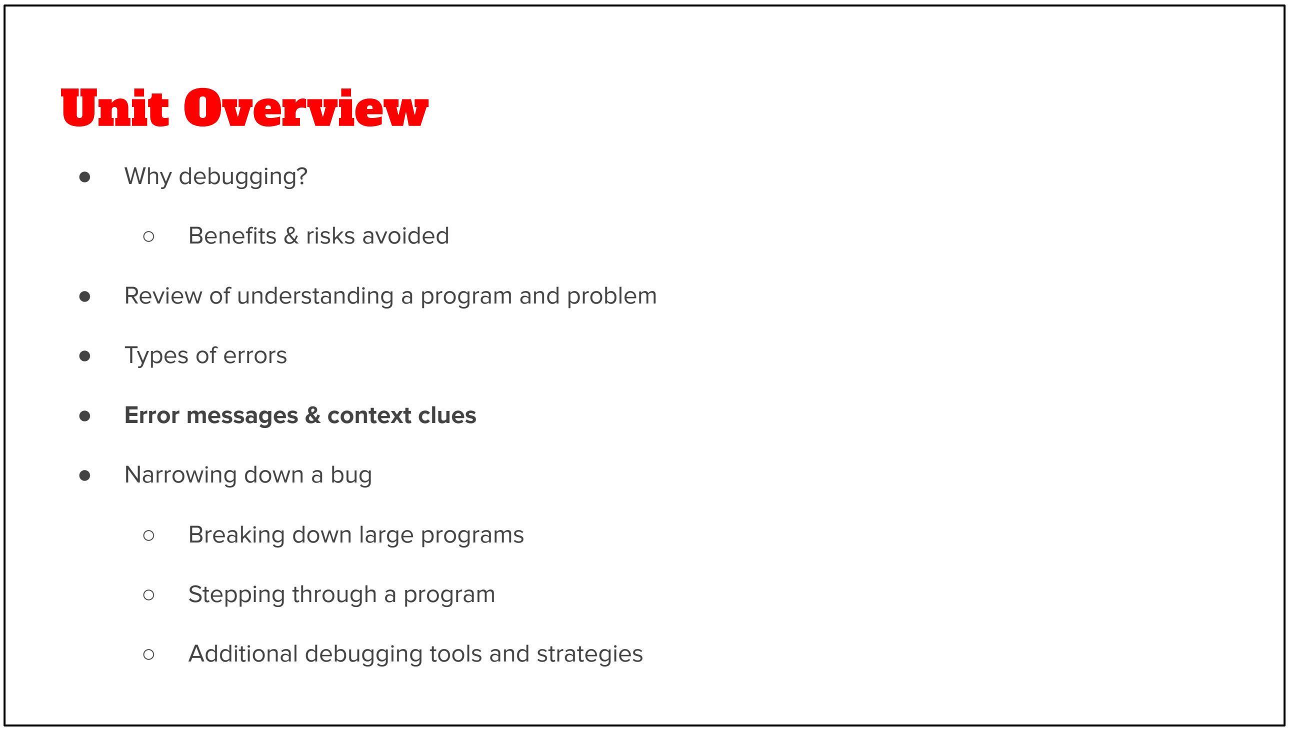
Description:
Review the lists created on Monday and add any items as needed.

Instructor Actions:
Present the lists created on Monday.

Learner Actions:
Actively listen and suggest any potential additions.

Suggested Speaking Notes:
Last class we talked about why debugging is important and some types of errors. Let's quickly review the lists we made on Monday.

What are some types of errors you remember from last class?



Unit Overview

- Why debugging?
  - Benefits & risks avoided
- Review of understanding a program and problem
- Types of errors
- **Error messages & context clues**
- Narrowing down a bug
  - Breaking down large programs
  - Stepping through a program
  - Additional debugging tools and strategies

Title: Describe what is new (to be learned)

Suggested Time: 1 min

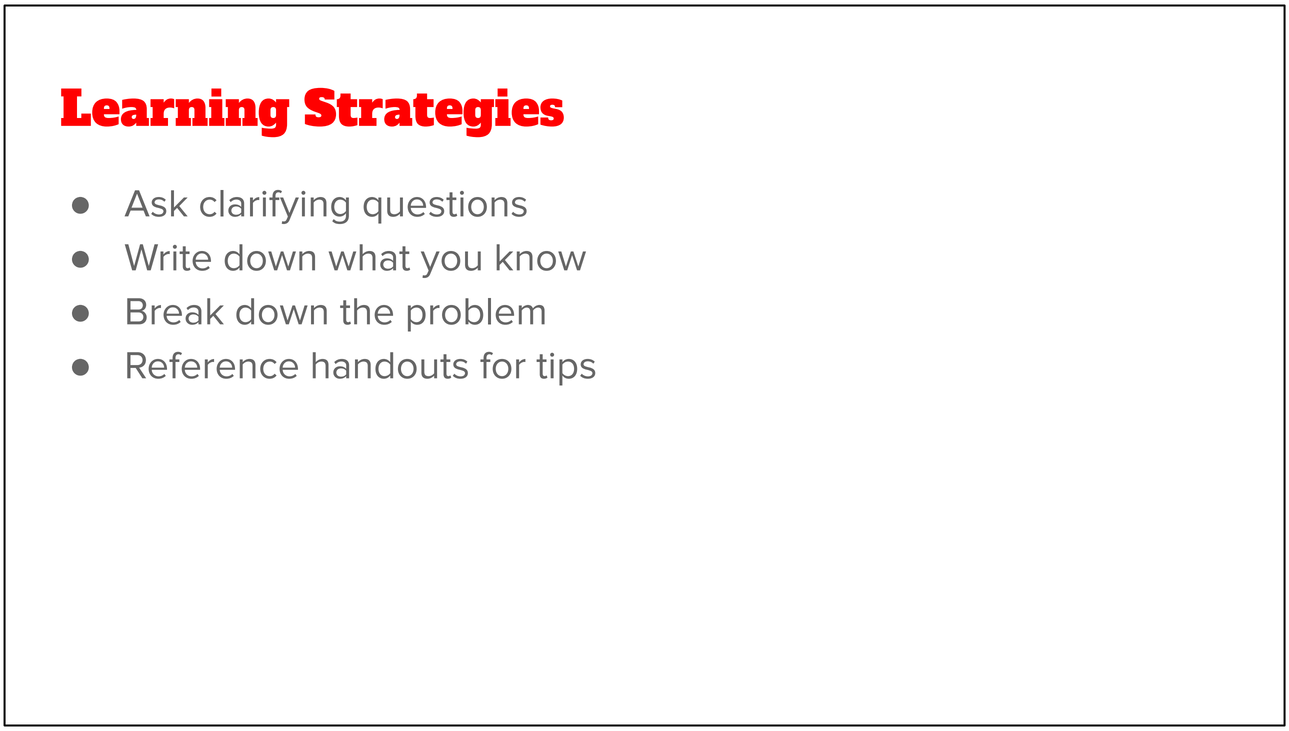
Materials / Handouts:

Description:
Discussion of what will be covered in the lesson.

Instructor Actions:
Explain what will be covered in the unit and that the unit will take place over the course of a week.

Learner Actions:
Actively listen and ask clarifying questions as needed.

Suggested Speaking Notes:
Here's the unit overview we looked at on Monday. Today, we'll look more indepth at error messages and how to find and use context clues in the error messages.



Learning Strategies

- Ask clarifying questions
- Write down what you know
- Break down the problem
- Reference handouts for tips

Title: Describe learning strategies

Suggested Time: 1 min

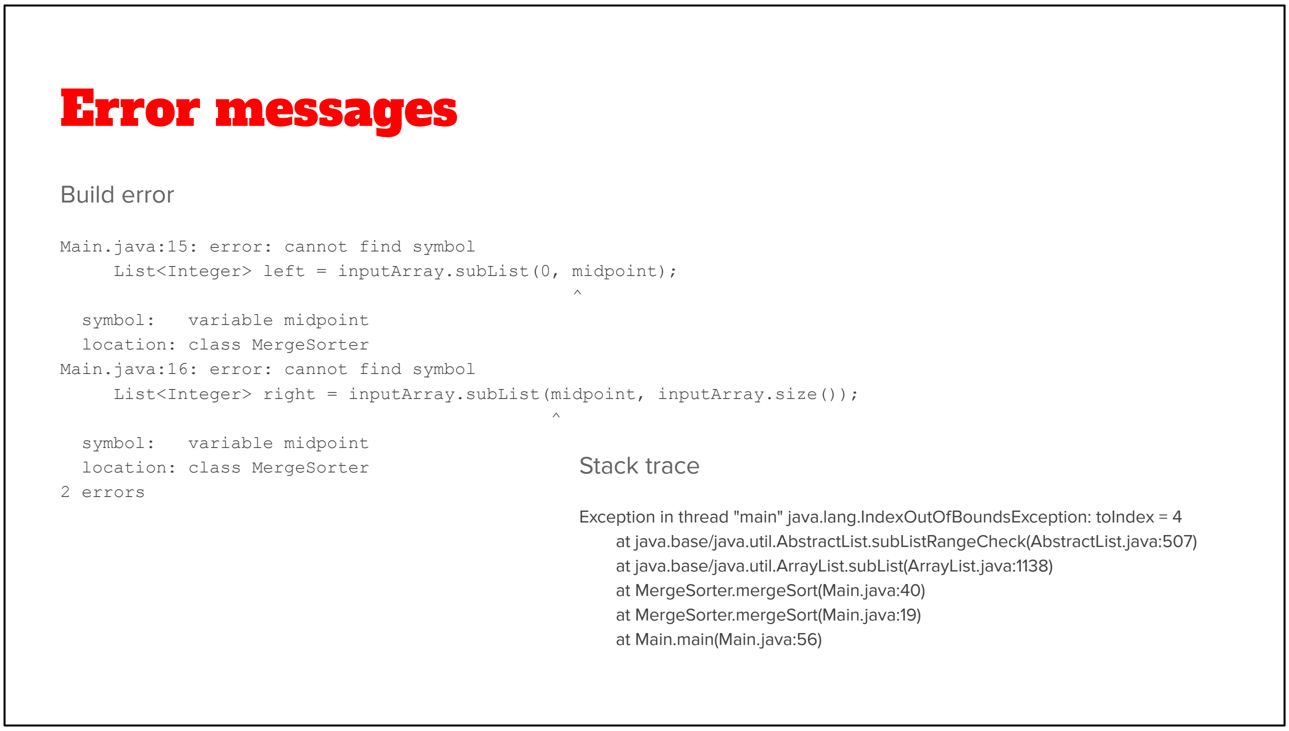
Materials / Handouts:

Description:
Discussion of strategies that can be used throughout the lesson, including asking clarifying questions, writing down what they know, and breaking things down into a smaller problem.

Instructor Actions:
Remind learners of the different learning strategies that we discussed on Monday.

Learner Actions:
Listen and think about what learning strategies they already use and which ones they could try.

Suggested Speaking Notes:
As a reminder, here are the learning strategies we discussed on Monday. Think about situations where you can use them during today's lesson.



Error messages

Build error

```
Main.java:15: error: cannot find symbol
     List<Integer> left = inputArray.subList(0, midpoint);
                                                ^
  symbol:   variable midpoint
  location: class MergeSorter
Main.java:16: error: cannot find symbol
     List<Integer> right = inputArray.subList(midpoint, inputArray.size());
                                              ^
  symbol:   variable midpoint
  location: class MergeSorter
2 errors
```

Stack trace

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: toIndex = 4
	at java.base/java.util.AbstractList.subListRangeCheck(AbstractList.java:507)
	at java.base/java.util.ArrayList.subList(ArrayList.java:1138)
	at MergeSorter.mergeSort(Main.java:40)
	at MergeSorter.mergeSort(Main.java:19)
	at Main.main(Main.java:56)
```

Title: Teach declarative knowledge

Suggested Time: 3 mins

Materials / Handouts:
context clues handout (text only)

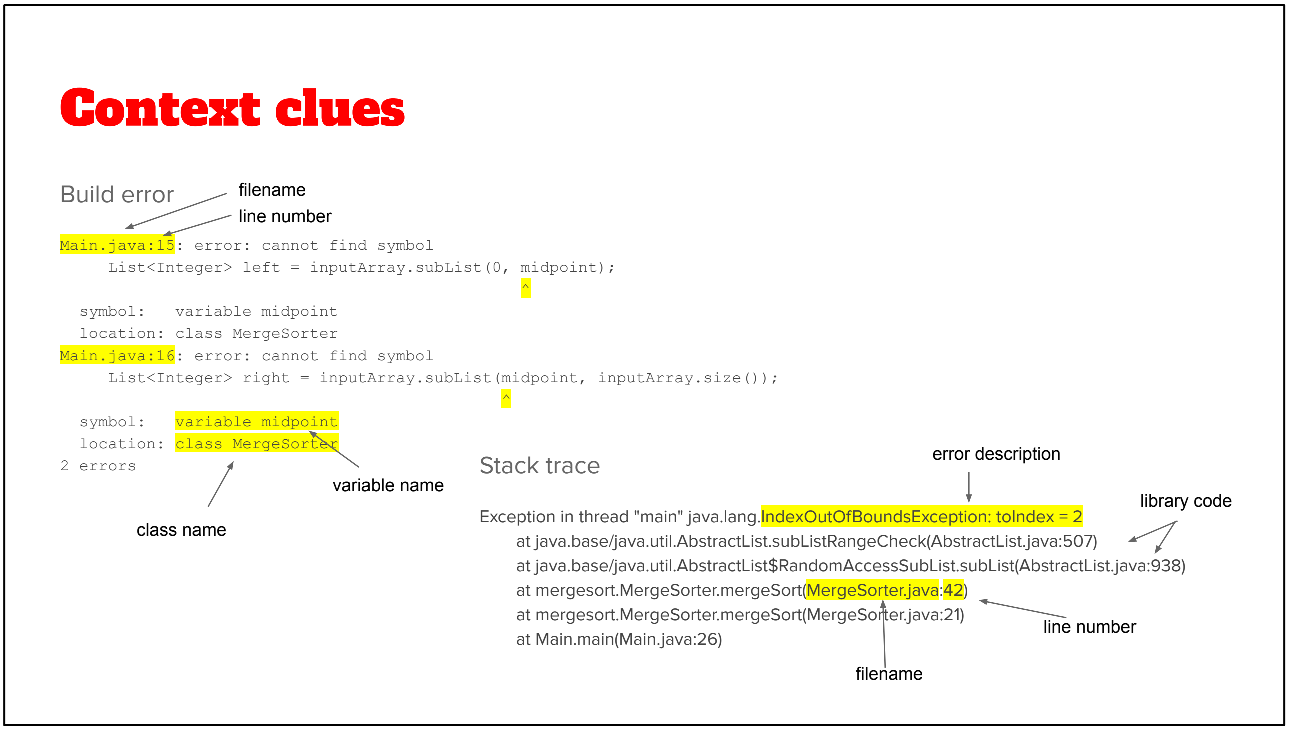
Description:
Instructor presents example error messages (e.g. stack traces and build errors) and highlights context clues.

Instructor Actions:
Present example error messages and highlight context clues including file names, line numbers, and variable types. Describe how the different context clues relate to the program code. *Note: Text on these slides is a bit small, encourage students to follow along on the print out of the slides.*

Learner Actions:
Listen to the instructor presenting examples of context clues and ask clarifying questions as needed.

Suggested Speaking Notes:

When a program fails we often get an error message. What this error message looks like depends on the language we're using and the type of error. Here's a couple of examples of what an error message might look like. There's an example of a build error and a stack trace you might get from a runtime error. Error messages can be particularly helpful because they often contain clues as to the context of the error. We'll call these context clues moving forward. If we look more closely at the build error, you'll notice that this is actually 2 errors. The error messages for build errors will show all the build errors in a particular file at once. They will also have a line number for each. Stack traces on the other hand will only show the first error that occurs. The also show a list of the functions that were called leading up to the error. So in this case, the error is being thrown from subListRangeCheck which is called by subList which was called from mergeSort.



Context clues

Build error

```
Main.java:15: error: cannot find symbol
     List<Integer> left = inputArray.subList(0, midpoint);
                                                ^

  symbol:   variable midpoint
  location: class MergeSorter
Main.java:16: error: cannot find symbol
     List<Integer> right = inputArray.subList(midpoint, inputArray.size());
                                              ^

  symbol:   variable midpoint
  location: class MergeSorter
2 errors
```

Stack trace

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: toIndex = 2
	at java.base/java.util.AbstractList.subListRangeCheck(AbstractList.java:507)
	at java.base/java.util.AbstractList$RandomAccessSubList.subList(AbstractList.java:938)
	at mergesort.MergeSorter.mergeSort(MergeSorter.java:42)
	at mergesort.MergeSorter.mergeSort(MergeSorter.java:21)
	at Main.main(Main.java:26)
```

Title: Teach declarative knowledge

Suggested Time: 3 mins

Materials / Handouts:
context clues handout (text only)

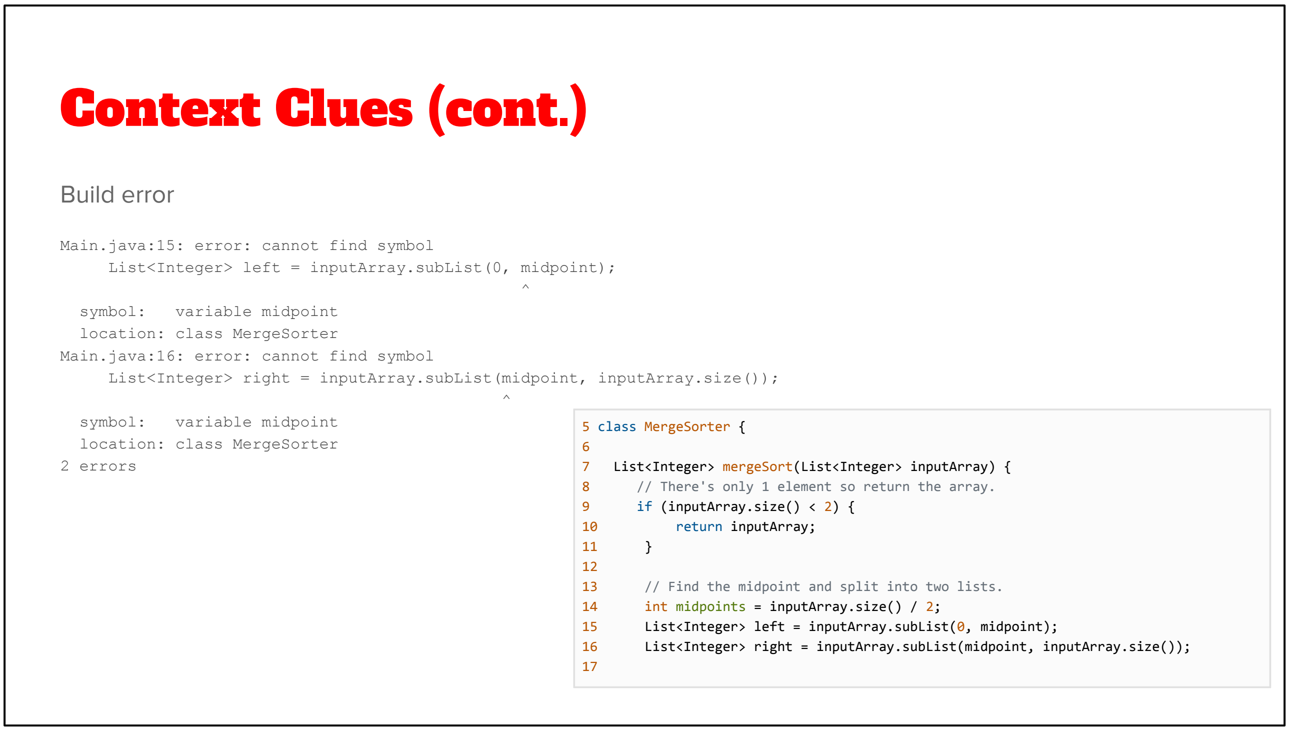
Description:
Instructor presents example error messages (e.g. stack traces and build errors) and highlights context clues.

Instructor Actions:
Present example error messages and highlight context clues including file names, line numbers, and variable types. Describe how the different context clues relate to the program code.

Learner Actions:
Listen to the instructor presenting examples of context clues and ask clarifying questions as needed.

Suggested Speaking Notes:
Here's some examples of the types of context clues we can find in these error

messages. There might be a filename or line number which we can point directly to the line of code with the error. Or there might be less specific context clues we need like the variable name or a class name. With a stack trace, you may not always recognize the first filename that is listed. When this happens it’s because the error is being thrown in library code. Since you did not write the library code, you can generally skip these files and move down the list until you find a filename for code you wrote.



Context Clues (cont.)

Build error

```
Main.java:15: error: cannot find symbol
     List<Integer> left = inputArray.subList(0, midpoint);
                                                ^
  symbol:   variable midpoint
  location: class MergeSorter
Main.java:16: error: cannot find symbol
     List<Integer> right = inputArray.subList(midpoint, inputArray.size());
                                              ^
  symbol:   variable midpoint
  location: class MergeSorter
2 errors
```

```
5 class MergeSorter {
6
7   List<Integer> mergeSort(List<Integer> inputArray) {
8      // There's only 1 element so return the array.
9      if (inputArray.size() < 2) {
10          return inputArray;
11      }
12
13      // Find the midpoint and split into two lists.
14      int midpoints = inputArray.size() / 2;
15      List<Integer> left = inputArray.subList(0, midpoint);
16      List<Integer> right = inputArray.subList(midpoint, inputArray.size());
17
```

Title: Demonstrate procedures ("how to")

Suggested Time: 3 mins

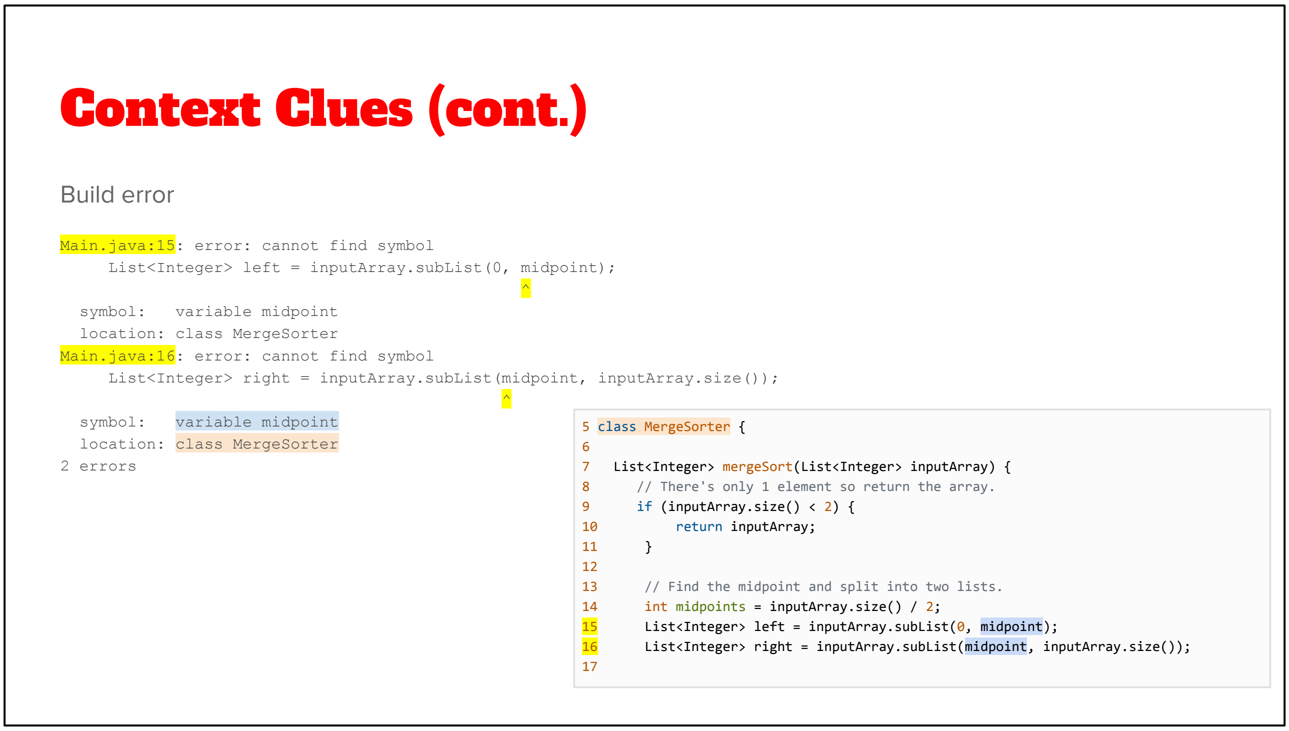
Materials / Handouts:
context clues handout (text only)

Description:
Instructor presents a practice problem for students to identify context clues as a class.

Instructor Actions:
Present a sample error message without context clues highlighted and ask students to name context clues.

Learner Actions:
Find and share context clues during the whole class example.

Suggested Speaking Notes:
Looking at this build error, what are some context we could pick out and how do they relate to the code that is shown here?



Context Clues (cont.)

Build error

```
Main.java:15: error: cannot find symbol
    List<Integer> left = inputArray.subList(0, midpoint);
                                               ^
  symbol:   variable midpoint
  location: class MergeSorter
Main.java:16: error: cannot find symbol
    List<Integer> right = inputArray.subList(midpoint, inputArray.size());
                                             ^
  symbol:   variable midpoint
  location: class MergeSorter
2 errors
```

```
5  class MergeSorter {
6
7    List<Integer> mergeSort(List<Integer> inputArray) {
8       // There's only 1 element so return the array.
9       if (inputArray.size() < 2) {
10          return inputArray;
11       }
12
13       // Find the midpoint and split into two lists.
14       int midpoints = inputArray.size() / 2;
15       List<Integer> left = inputArray.subList(0, midpoint);
16       List<Integer> right = inputArray.subList(midpoint, inputArray.size());
17
```

Title: Demonstrate procedures ("how to")

Suggested Time: 1 min

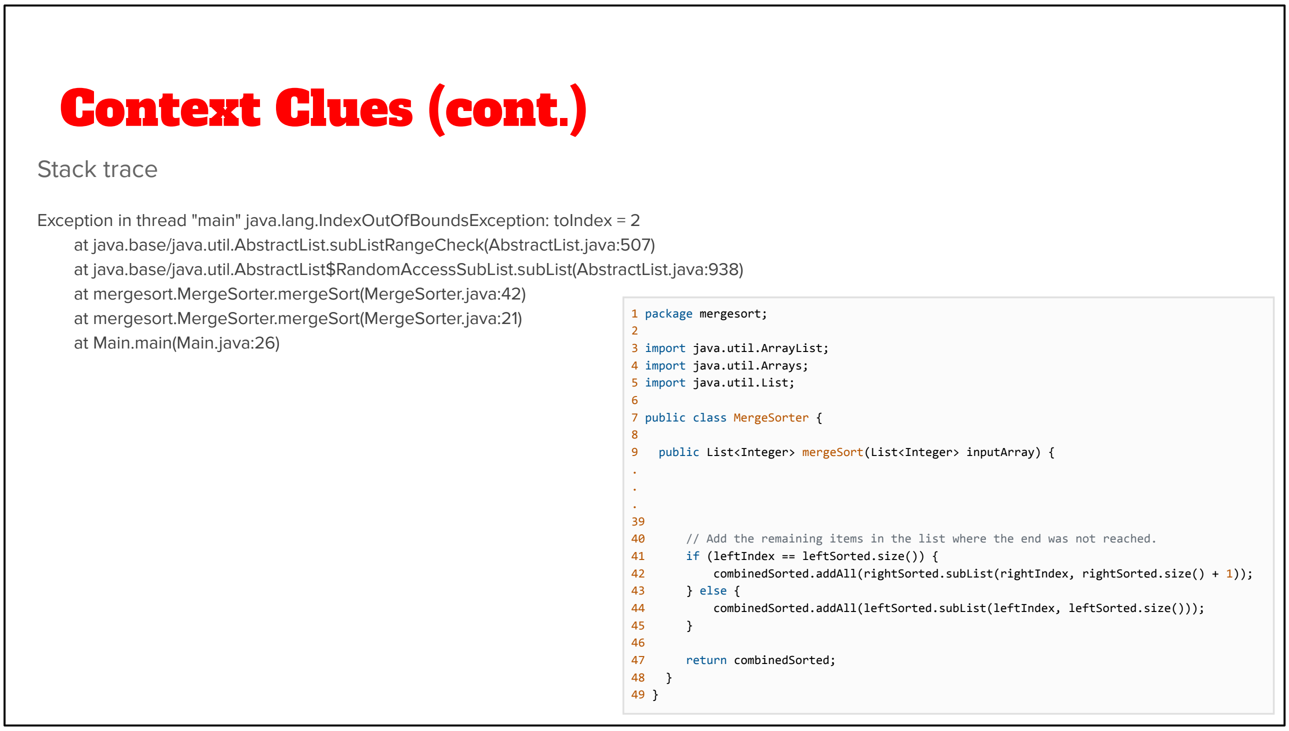
Materials / Handouts:
context clues handout (text only)

Description:
Instructor presents a practice problem for students to identify context clues as a class.

Instructor Actions:
Present a sample error message without context clues highlighted and ask students to name context clues.

Learner Actions:
Find and share context clues during the whole class example.

Suggested Speaking Notes:
Here we see those context clues and how they matched up.



Context Clues (cont.)

Stack trace

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: toIndex = 2
	at java.base/java.util.AbstractList.subListRangeCheck(AbstractList.java:507)
	at java.base/java.util.AbstractList$RandomAccessSubList.subList(AbstractList.java:938)
	at mergesort.MergeSorter.mergeSort(MergeSorter.java:42)
	at mergesort.MergeSorter.mergeSort(MergeSorter.java:21)
	at Main.main(Main.java:26)
```

```java
1 package mergesort;
2
3 import java.util.ArrayList;
4 import java.util.Arrays;
5 import java.util.List;
6
7 public class MergeSorter {
8
9   public List<Integer> mergeSort(List<Integer> inputArray) {
.
.
.
39
40      // Add the remaining items in the list where the end was not reached.
41      if (leftIndex == leftSorted.size()) {
42          combinedSorted.addAll(rightSorted.subList(rightIndex, rightSorted.size() + 1));
43      } else {
44          combinedSorted.addAll(leftSorted.subList(leftIndex, leftSorted.size()));
45      }
46
47      return combinedSorted;
48   }
49 }
```

Note: may want to use a different runtime error here like a null pointer exception

Title: Demonstrate procedures ("how to")

Suggested Time: 3 mins

Materials / Handouts:
context clues handout (text only)

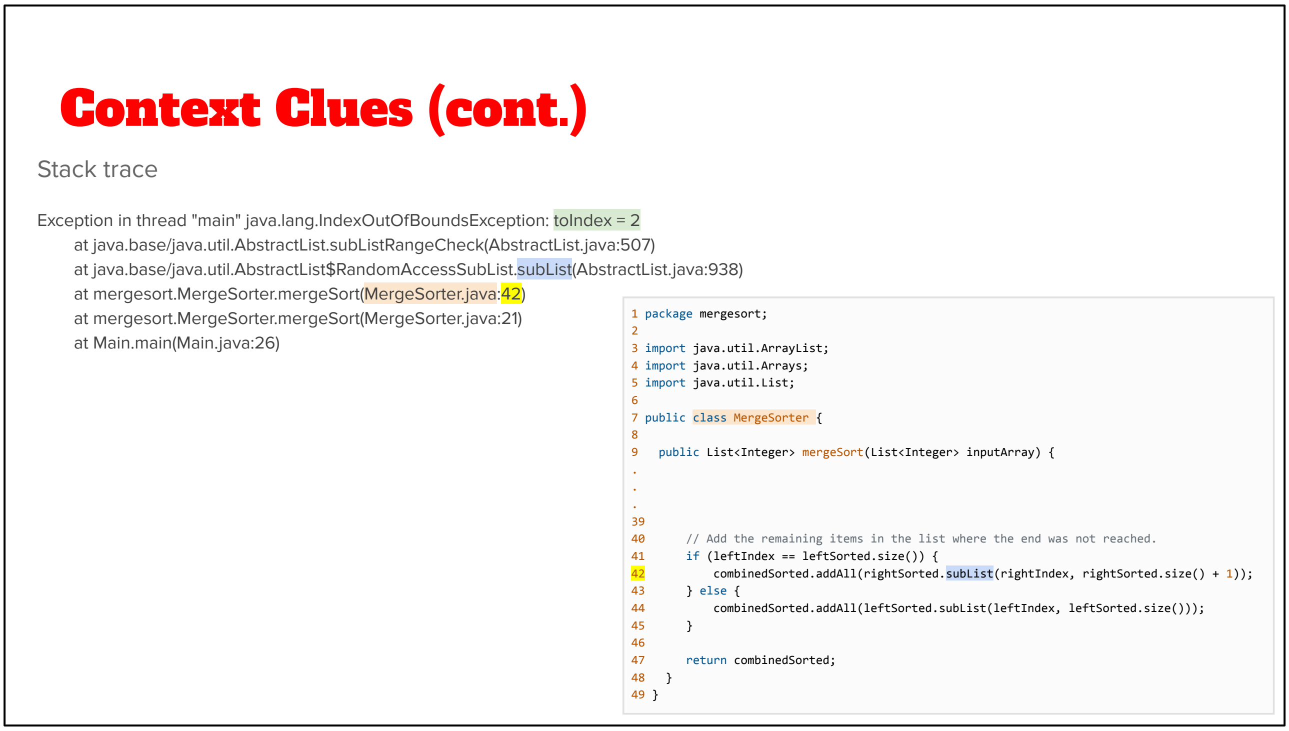
Description:
Instructor presents a practice problem for students to identify context clues as a class.

Instructor Actions:
Present a sample error message without context clues highlighted and ask students to name context clues.

Learner Actions:
Find and share context clues during the whole class example.

Suggested Speaking Notes:

Looking at this stack trace, what are some context we could pick out and how do they relate to the code that is shown here?



Context Clues (cont.)

Stack trace

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: toIndex = 2
    at java.base/java.util.AbstractList.subListRangeCheck(AbstractList.java:507)
    at java.base/java.util.AbstractList$RandomAccessSubList.subList(AbstractList.java:938)
    at mergesort.MergeSorter.mergeSort(MergeSorter.java:42)
    at mergesort.MergeSorter.mergeSort(MergeSorter.java:21)
    at Main.main(Main.java:26)
```

```
1 package mergesort;
2
3 import java.util.ArrayList;
4 import java.util.Arrays;
5 import java.util.List;
6
7 public class MergeSorter {
8
9   public List<Integer> mergeSort(List<Integer> inputArray) {
.
.
.
39
40      // Add the remaining items in the list where the end was not reached.
41      if (leftIndex == leftSorted.size()) {
42          combinedSorted.addAll(rightSorted.subList(rightIndex, rightSorted.size() + 1));
43      } else {
44          combinedSorted.addAll(leftSorted.subList(leftIndex, leftSorted.size()));
45      }
46
47      return combinedSorted;
48   }
49 }
```

Title: Demonstrate procedures ("how to")

Suggested Time: 1 min

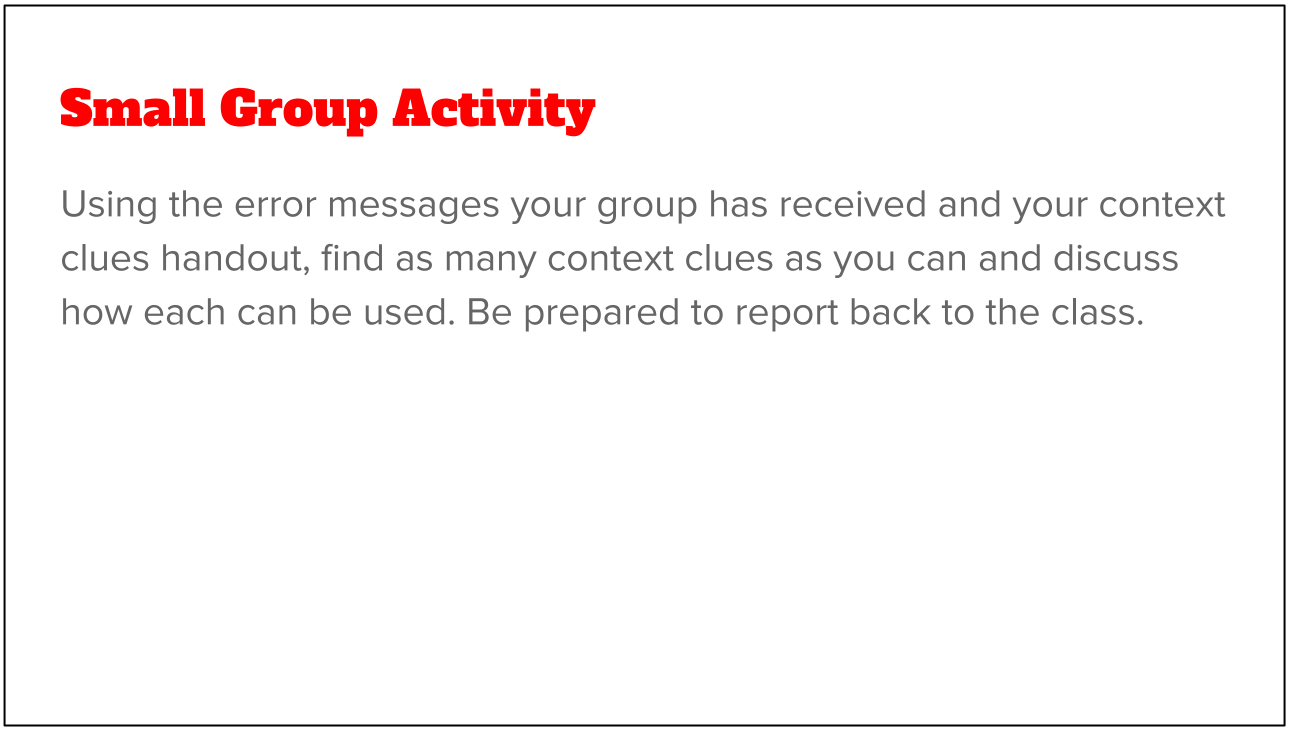
Materials / Handouts:
context clues handout (text only)

Description:
Instructor presents a practice problem for students to identify context clues as a class.

Instructor Actions:
Present a sample error message without context clues highlighted and ask students to name context clues.

Learner Actions:
Find and share context clues during the whole class example.

Suggested Speaking Notes:
Looking at this stack trace, what are some context we could pick out and how do they relate to the code that is shown here?



Small Group Activity

Using the error messages your group has received and your context clues handout, find as many context clues as you can and discuss how each can be used. Be prepared to report back to the class.

Title: Provide practice and feedback

Suggested Time: 10 mins

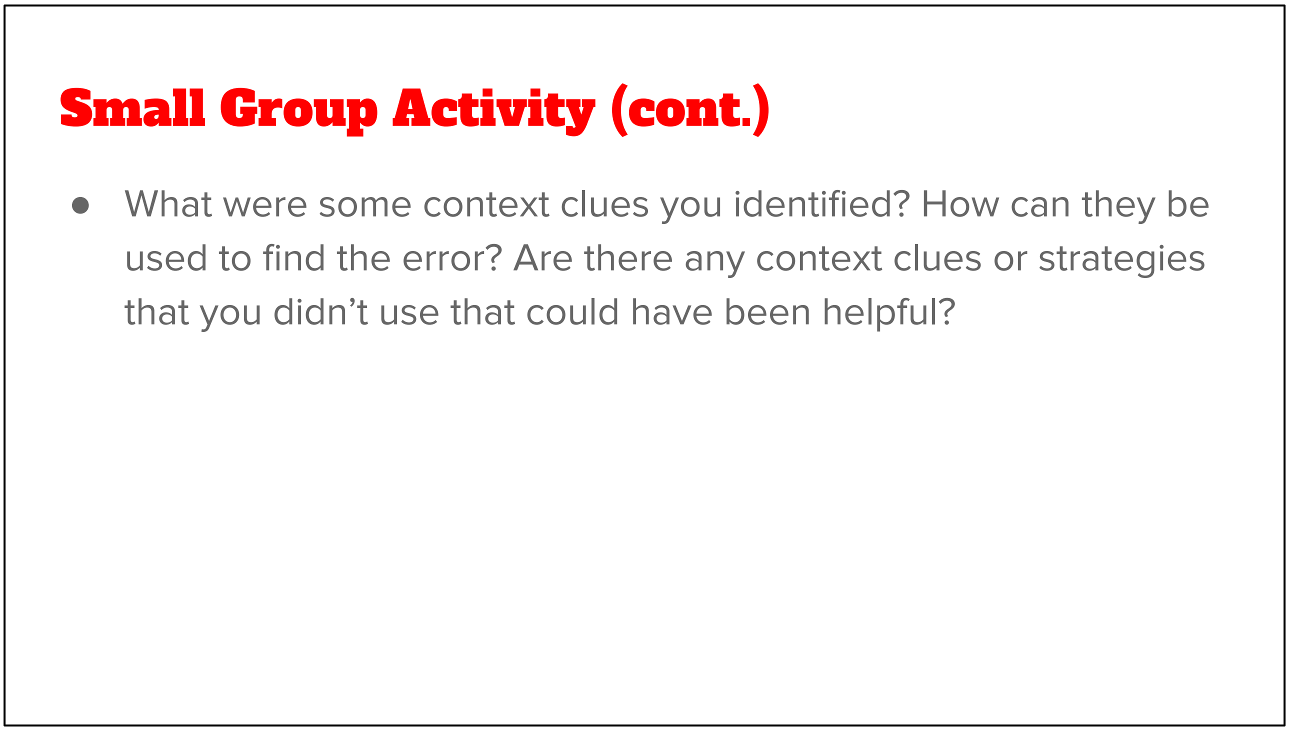
Materials / Handouts:
Example problems, types of errors handout, context clues handout (text only)

Description:
Students break up into small groups and work to identify context clues of sample problems.

Instructor Actions:
Provide sample problems for small group work. Observe students working to solve small group activities and provide feedback on what learning strategies they are using well and which ones to try if the learners get stuck.

Learner Actions:
Identify context clues and discuss with their small group based on the sample problem they were provided.

Suggested Speaking Notes:
Now we're going to break up into small groups and try this on our own.



Small Group Activity (cont.)

- What were some context clues you identified? How can they be used to find the error? Are there any context clues or strategies that you didn't use that could have been helpful?

Title: Retention and Transfer

Suggested Time: 3 mins

Materials / Handouts:
Example problems, types of errors handout, context clues handout (text only)

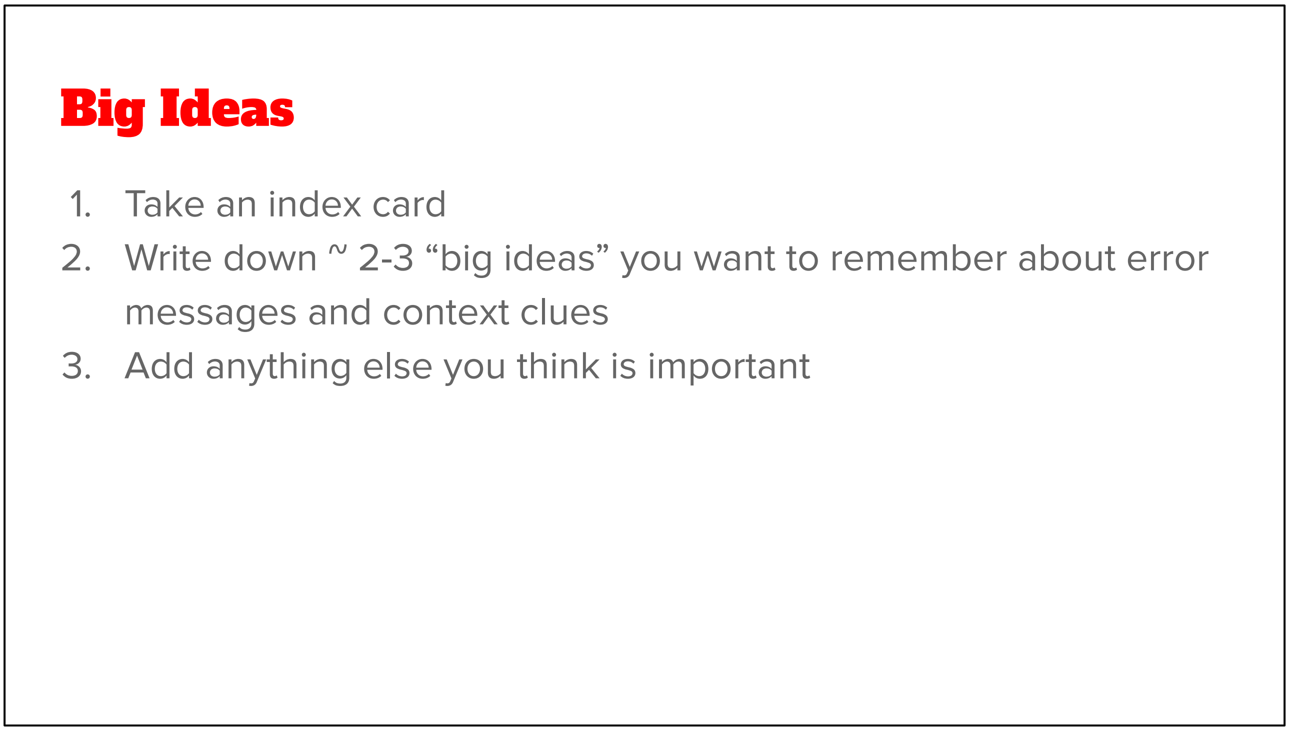
Description:
After each small group activity, students will share what strategies they used and the cues that helped them decide to use these strategies. This will help identify features of novel problems the strategies can transfer to.

Instructor Actions:
Ask students to share what cues lead to their use of different strategies and provide feedback on additional cues or strategies.

Learner Actions:
Recall what strategies they used during small group work and the cues that lead to the use of the strategies. Listen to the cues and strategies used by their classmates and consider which ones they could use in the future.

Suggested Speaking Notes:

Okay, now we’re going to report back to the group. Who wants to share what their group came up with and what strategies you used?



Big Ideas

1. Take an index card
2. Write down ~ 2-3 “big ideas” you want to remember about error messages and context clues
3. Add anything else you think is important

Title: Big Ideas

Suggested Time: 3 mins

Materials / Handouts:
Index cards

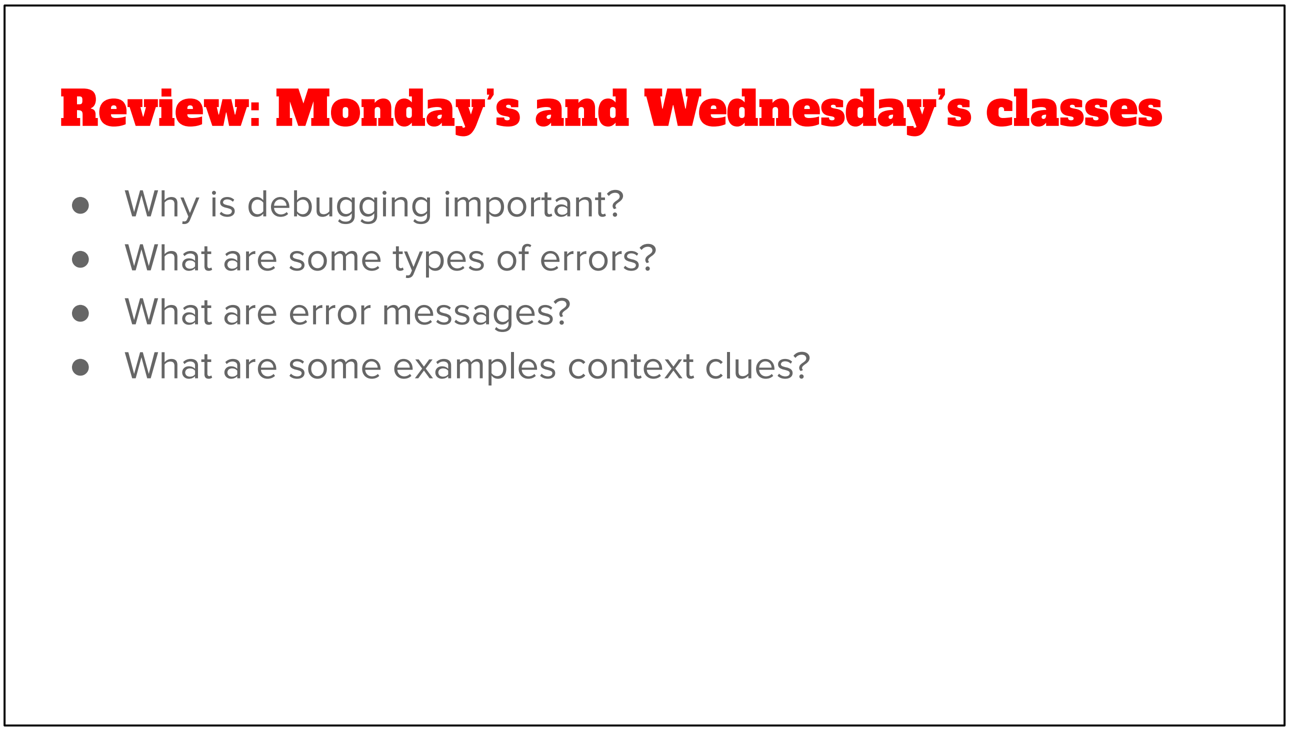
Description:
Students create reference guides of big ideas to remember when debugging problems in the future.

Instructor Actions:
Provide students with index cards or pieces of paper to create reference guides. Ask learners to think about what big ideas they would want to remember next time they have an error with one of their programs and need to debug it.

Learner Actions:
Recall key concepts they want to remember and write them on their reference guide. Share ideas with the class and incorporate new ideas they like into their reference guide.

Suggested Speaking Notes:
You’re going to need to be able to use what we’ve covered today in the future so think about what big ideas you want to remember and write them down on your index card.

Friday



Review: Monday's and Wednesday's classes

- Why is debugging important?
- What are some types of errors?
- What are error messages?
- What are some examples context clues?

Title: Stimulate Motivation / Benefits & Risks Avoided / Recall Prior Knowledge

Suggested Time: 6 mins

Materials / Handouts:
Slides/notes from last class

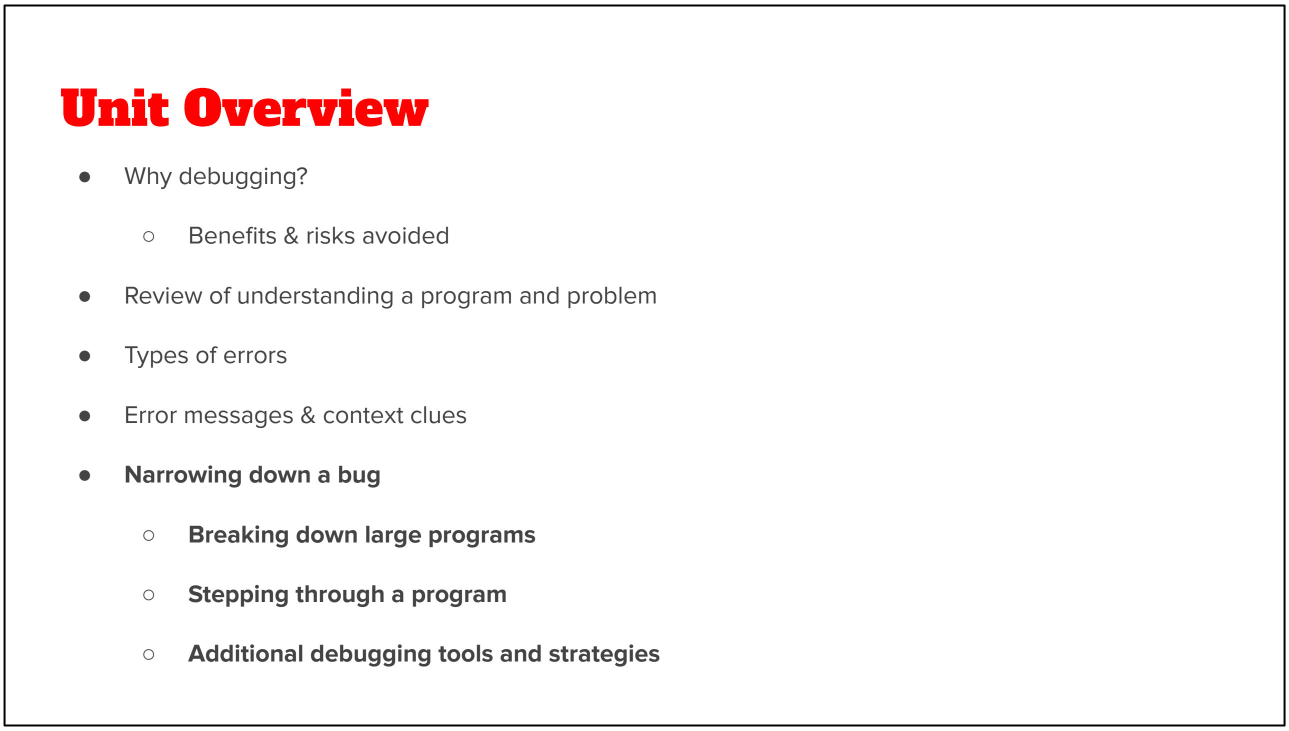
Description:
Review the lists created on Monday and add any items as needed.

Instructor Actions:
Present the lists created on Monday.

Learner Actions:
Actively listen and suggest any potential additions. Recall what error messages are and example context clues from Wednesday's class.

Suggested Speaking Notes:
Quickly reviewing what we already covered in this unit, here are our lists about why debugging is important. Is there anything you want to add?

What are some types of errors we've learned about? Can anyone describe what an error message is and give an example of a context clue you might find in an error message?



Unit Overview

- Why debugging?
  - Benefits & risks avoided
- Review of understanding a program and problem
- Types of errors
- Error messages & context clues
- **Narrowing down a bug**
  - **Breaking down large programs**
  - **Stepping through a program**
  - **Additional debugging tools and strategies**

Title: Describe what is new (to be learned)

Suggested Time: 1 min

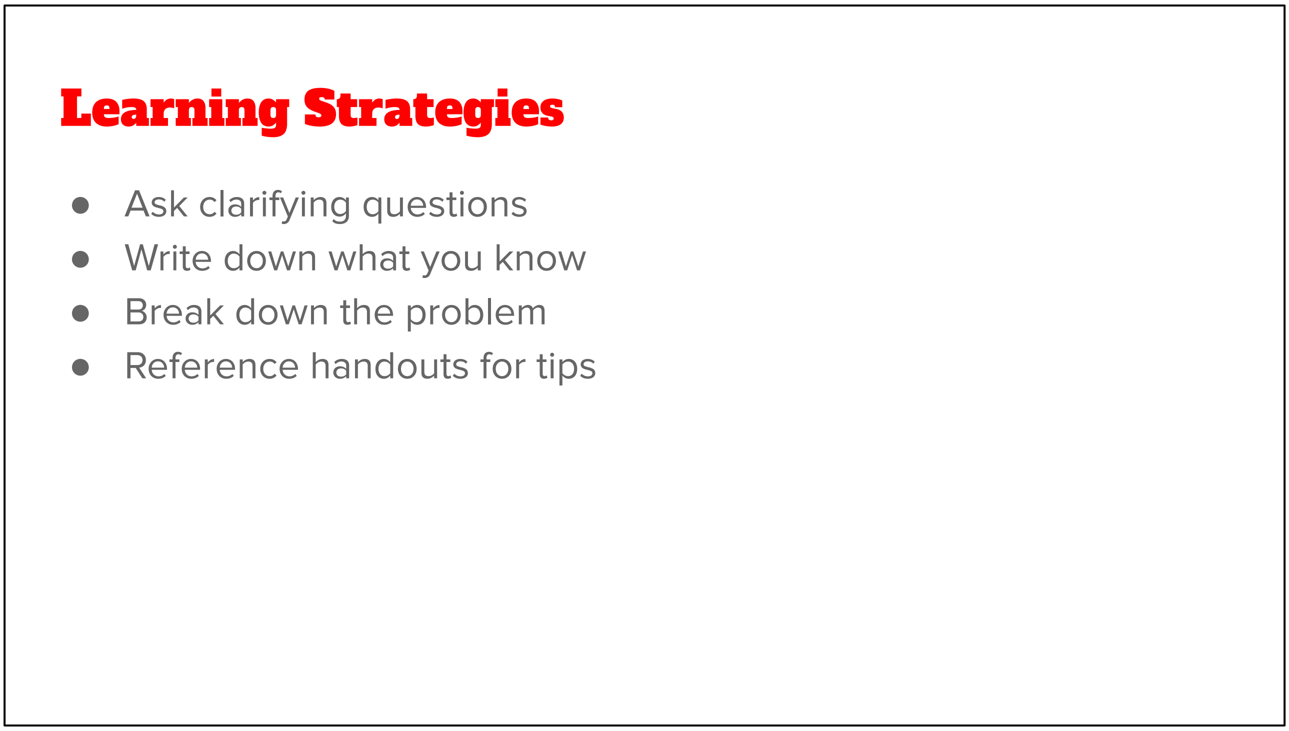
Materials / Handouts:

Description:
Discussion of what will be covered in the lesson.

Instructor Actions:
Explain what will be covered in the unit and that the unit will take place over the course of a week.

Learner Actions:
Actively listen and ask clarifying questions as needed.

Suggested Speaking Notes:
Here's the unit overview we looked at previously. Today, we'll talk about ways to narrow down a bug in a large program, what it means to step through a program, and quickly discuss other tools and strategies that can be used while debugging.



Learning Strategies

- Ask clarifying questions
- Write down what you know
- Break down the problem
- Reference handouts for tips

Title: Describe learning strategies

Suggested Time: 1 min

Materials / Handouts:

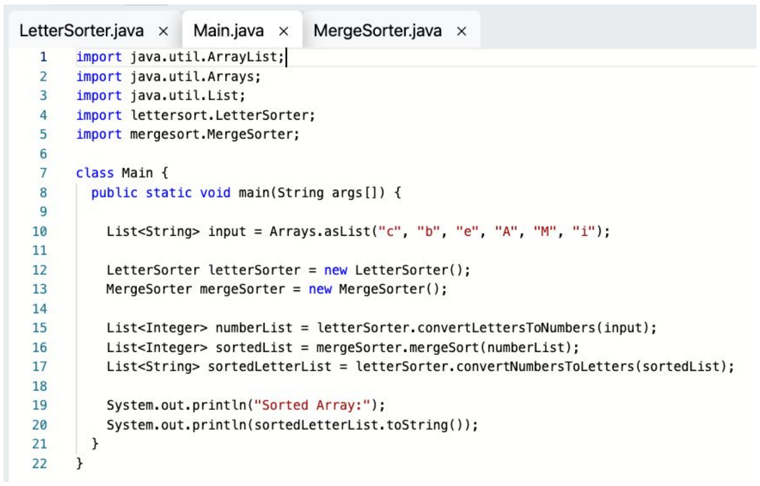
Description:
Discussion of strategies that can be used throughout the lesson, including asking clarifying questions, writing down what they know, and breaking things down into a smaller problem.

Instructor Actions:
Remind learners of the different learning strategies that we discussed on Monday.

Learner Actions:
Listen and think about what learning strategies they already use and which ones they could try.

Suggested Speaking Notes:
As a reminder, here are the learning strategies we discussed on Monday. Think about situations where you can use them during today's lesson.

Splitting Large Problems


LetterSorter.java × Main.java × MergeSorter.java ×

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import lettersort.LetterSorter;
import mergesort.MergeSorter;

class Main {
  public static void main(String args[]) {

    List<String> input = Arrays.asList("c", "b", "e", "A", "M", "i");

    LetterSorter letterSorter = new LetterSorter();
    MergeSorter mergeSorter = new MergeSorter();

    List<Integer> numberList = letterSorter.convertLettersToNumbers(input);
    List<Integer> sortedList = mergeSorter.mergeSort(numberList);
    List<String> sortedLetterList = letterSorter.convertNumbersToLetters(sortedList);

    System.out.println("Sorted Array:");
    System.out.println(sortedLetterList.toString());
  }
}
```

- Functions
- Loops
- Conditional statements
- Files
- Classes

Title: Teach declarative knowledge

Suggested Time: 2 mins

Materials / Handouts:
LetterSorter code handouts

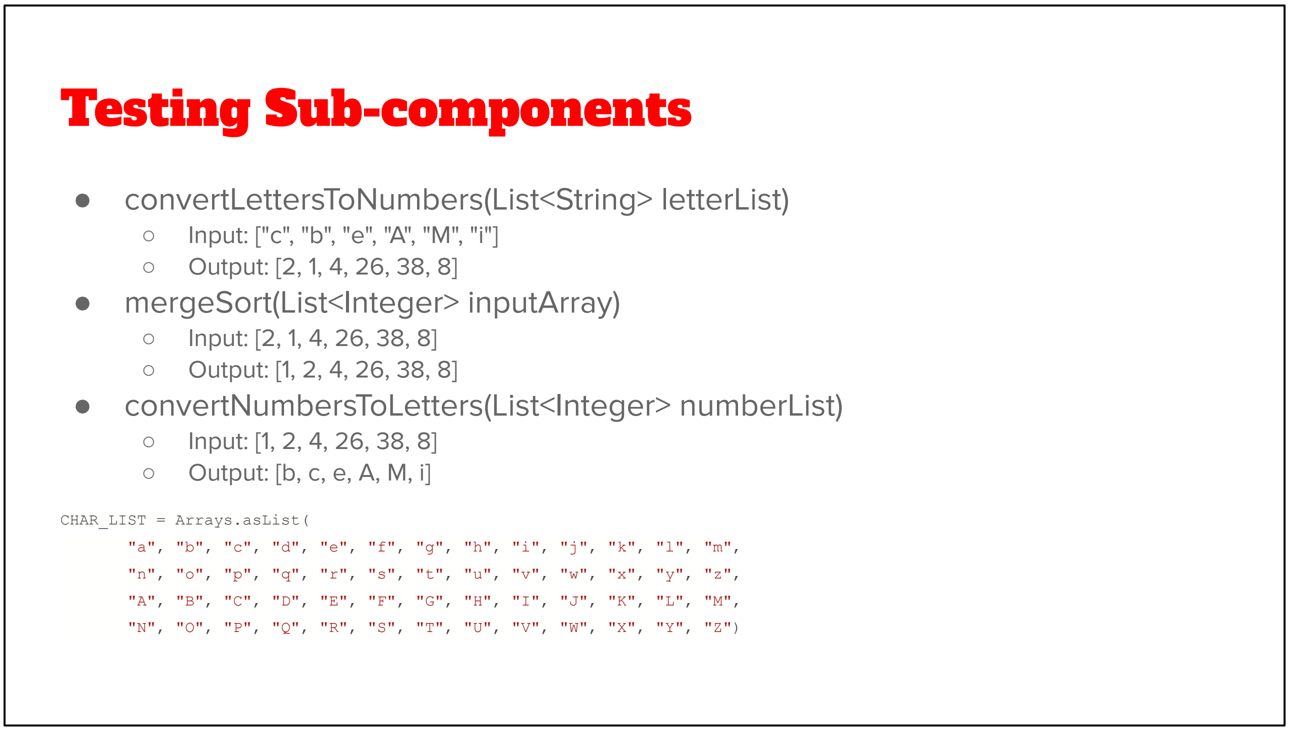
Description:
Discussion of larger programs and how it can be helpful to break problems down into smaller problems.

Instructor Actions:
Present an example of a large program and discuss the strategy of breaking a problem down to stepping through a subpiece.

Learner Actions:
Listen to how the strategies for narrowing down where the error is occuring relate to different size problems. Ask clarifying questions as needed.

Suggested Speaking Notes:
We've talked about how an error message can be used to find a line number where an error is but sometimes you won't have a line number to tell you

exactly where the bug is. This can often happen in large programs. So, what are some strategies you can use to narrow down where the error is in large or complex programs? Let’s take this large problem as an example, what are some “units” we could break it up into like we did when we talked about testing code? (pause for examples before showing the list)



Testing Sub-components

- convertLettersToNumbers(List<String> letterList)
  - Input: ["c", "b", "e", "A", "M", "i"]
  - Output: [2, 1, 4, 26, 38, 8]
- mergeSort(List<Integer> inputArray)
  - Input: [2, 1, 4, 26, 38, 8]
  - Output: [1, 2, 4, 26, 38, 8]
- convertNumbersToLetters(List<Integer> numberList)
  - Input: [1, 2, 4, 26, 38, 8]
  - Output: [b, c, e, A, M, i]

```
CHAR_LIST = Arrays.asList(
        "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m",
        "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z",
        "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M",
        "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z")
```

Title: Teach declarative knowledge

Suggested Time: 1 min

Materials / Handouts:

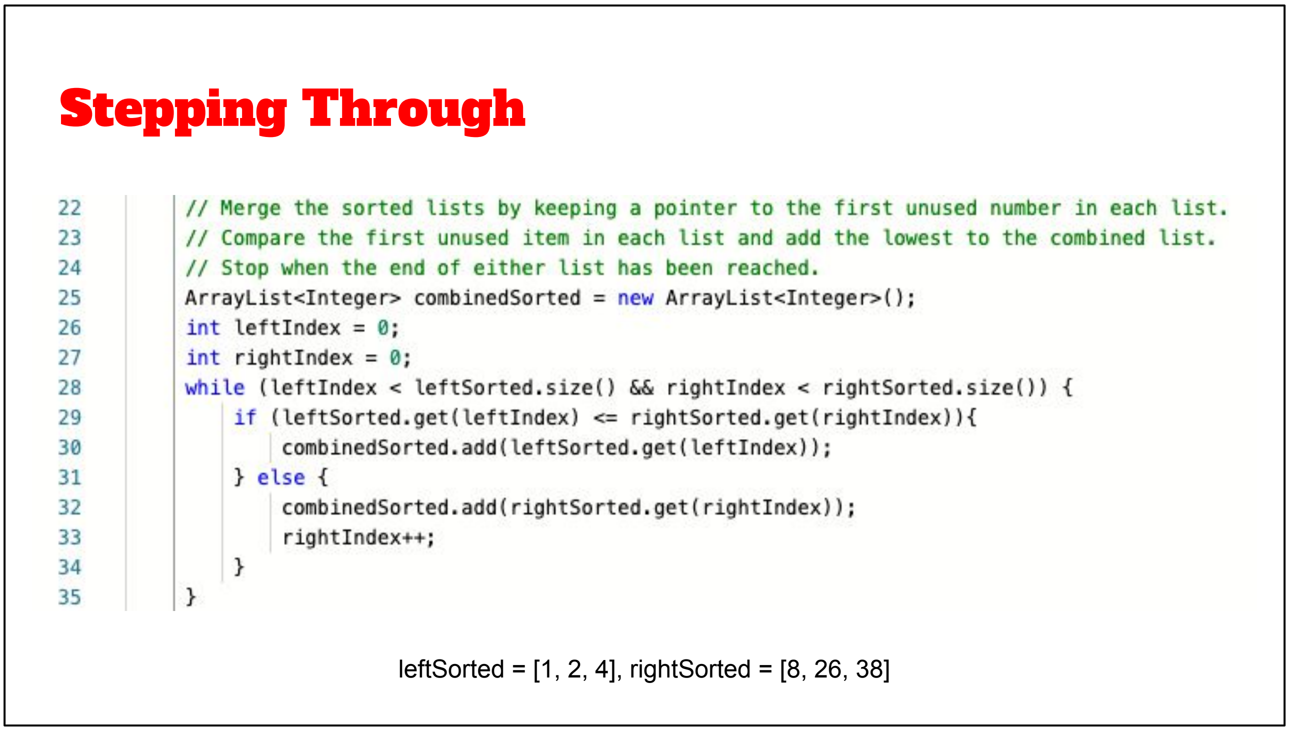
Description:
Explanation of how testing can be used to narrow down where in a large program an error is occuring. Discussion of how this concept can also be applied at the micro level by stepping through a program line by line.

Instructor Actions:
Present an example of a large program and discuss the strategy of breaking a problem down to stepping through a subpiece.

Learner Actions:
Listen to how the strategies for narrowing down where the error is occuring relate to different size problems. Ask clarifying questions as needed.

Suggested Speaking Notes:
Once we've broken down the program into these units, we can run just a sub-component to see if it's the part that produces the unexpected output. This

can be done with unit tests like we saw in Unit 3 or it can be done by isolating the code. For example, if you had a program that converted letters to numbers, sorted the numbers, and then converted the numbers back to letters you could break this into 3 parts. Then you might test just the sorting to see if it’s the part causing the error.



Stepping Through

```java
// Merge the sorted lists by keeping a pointer to the first unused number in each list.
// Compare the first unused item in each list and add the lowest to the combined list.
// Stop when the end of either list has been reached.
ArrayList<Integer> combinedSorted = new ArrayList<Integer>();
int leftIndex = 0;
int rightIndex = 0;
while (leftIndex < leftSorted.size() && rightIndex < rightSorted.size()) {
    if (leftSorted.get(leftIndex) <= rightSorted.get(rightIndex)){
        combinedSorted.add(leftSorted.get(leftIndex));
    } else {
        combinedSorted.add(rightSorted.get(rightIndex));
        rightIndex++;
    }
}
```

leftSorted = [1, 2, 4], rightSorted = [8, 26, 38]

Title: Demonstrate procedures (“how to”)

Suggested Time: 5 mins

Materials / Handouts:
whiteboard, personal whiteboards

Description:
Demonstration of how to step through a program and track intermediate values to determine where the error is.

Instructor Actions:
Demonstrate how to step through a program and track intermediate values to determine where the error is.

Learner Actions:
Observe how the instructor steps through the sample program. Follow along with their own whiteboards.

Suggested Speaking Notes:
Once we have a vague idea of where the bug is, we can use a process called “stepping through the program” to further narrow it down. Let’s take a look at

this example where we’ve narrowed down the error to this section of code. We can start at the first line and work our way through one line at a time. When we get to a line with a new variable we’ll add it to a list on our whiteboards and write down it’s value. Then any time the variable changes value we’ll update it on our whiteboard. If we reach a point where the value doesn’t match what we expect than we’ve found an error.

Small Group Activity

Using the error description, error message, and program code your group has received find the error in the program. Practice breaking the problem down and stepping through the code. Be prepared to report back to the class.

Title: Provide practice and feedback

Suggested Time: 10 mins

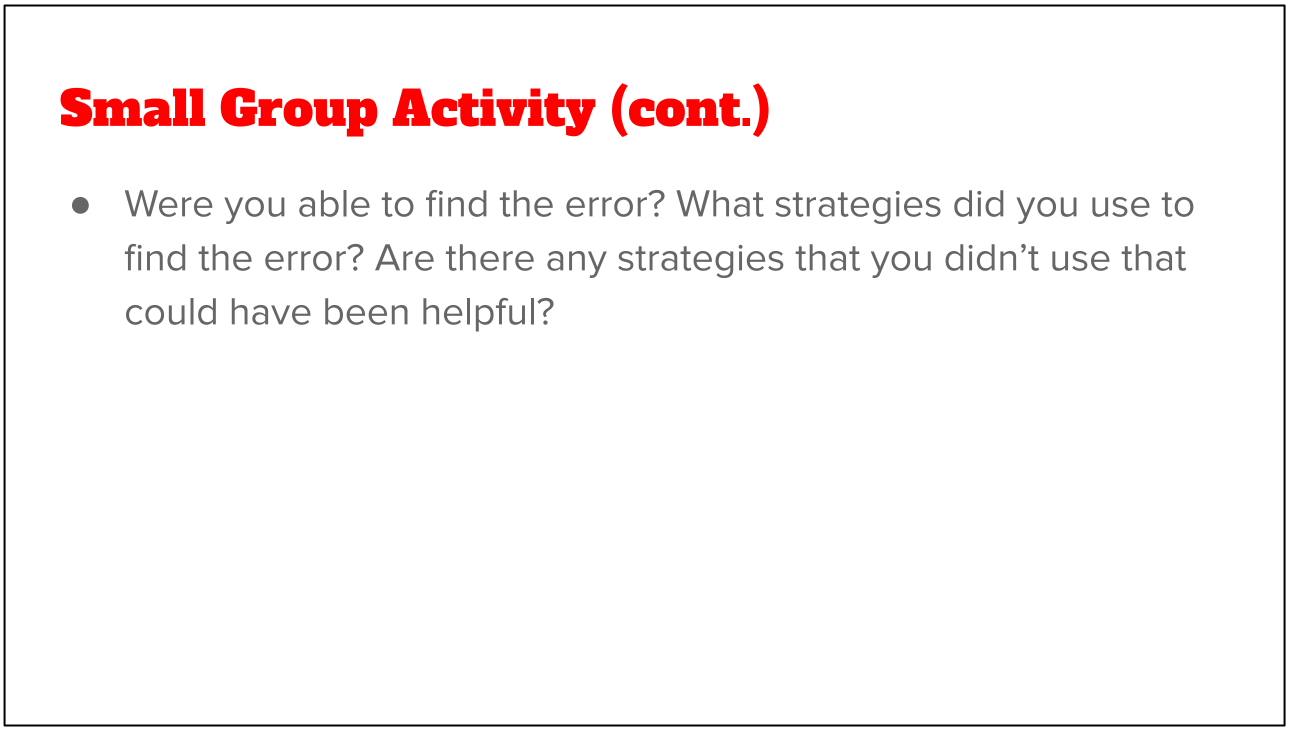
Materials / Handouts:
Example problems

Description:
In small groups, students are given a large program and asked to break the problem into sub pieces and use the strategies previously discussed to narrow down which piece has the error.

Instructor Actions:
Provide sample problems for small group work. Observe students working to solve small group activities and provide feedback on what learning strategies they are using well and which ones to try if the learners get stuck.

Learner Actions:
Use strategies covered during the previous discussion to narrow down where the error is occuring and make notes about the different pieces of the program and what they know about each. Use personal whiteboards to write down the values of variables as they step through the example program.

Suggested Speaking Notes:
Now we’re going to break up into small groups and try this on our own.



Small Group Activity (cont.)

- Were you able to find the error? What strategies did you use to find the error? Are there any strategies that you didn't use that could have been helpful?

Title: Retention and Transfer

Suggested Time: 3 mins

Materials / Handouts:
Example problems, types of errors handout

Description:
After each small group activity, students will share what strategies they used and the cues that helped them decide to use these strategies. This will help identify features of novel problems the strategies can transfer to.

Instructor Actions:
Ask students to share what cues lead to their use of different strategies and provide feedback on additional cues or strategies.

Learner Actions:
Recall what strategies they used during small group work and the cues that lead to the use of the strategies. Listen to the cues and strategies used by their classmates and consider which ones they could use in the future.

Suggested Speaking Notes:

Okay, now we’re going to report back to the group. Who wants to share what their group came up with and what strategies you used?

Debugging Tools & Other Strategies

- Debuggers
- Diffs

Title: Teach declarative knowledge / Retention and Transfer

Suggested Time: 1 min

Materials / Handouts:

Description:
Describe additional debugging tools and strategies that can be used if available.

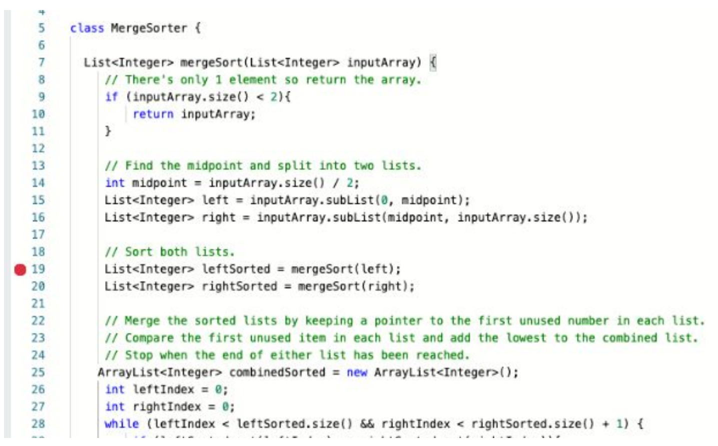
Instructor Actions:
Describe additional debugging tools and strategies that can be used if available.

Learner Actions:
Actively listen and ask clarifying questions if needed.

Suggested Speaking Notes:
In addition to what we've already covered, there are some other tools and strategies that can be helpful when debugging errors. They generally build off the same processes we've covered but can be helpful when working with large complex programs by saving time. Two important ones we'll cover are

debuggers and file diffs.

Debuggers



```
class MergeSorter {

  List<Integer> mergeSort(List<Integer> inputArray) {
    // There's only 1 element so return the array.
    if (inputArray.size() < 2){
        return inputArray;
    }

    // Find the midpoint and split into two lists.
    int midpoint = inputArray.size() / 2;
    List<Integer> left = inputArray.subList(0, midpoint);
    List<Integer> right = inputArray.subList(midpoint, inputArray.size());

    // Sort both lists.
    List<Integer> leftSorted = mergeSort(left);
    List<Integer> rightSorted = mergeSort(right);

    // Merge the sorted lists by keeping a pointer to the first unused number in each list.
    // Compare the first unused item in each list and add the lowest to the combined list.
    // Stop when the end of either list has been reached.
   ArrayList<Integer> combinedSorted = new ArrayList<Integer>();
    int leftIndex = 0;
    int rightIndex = 0;
    while (leftIndex < leftSorted.size() && rightIndex < rightSorted.size() + 1) {
```

.
.
.

inputArray = [6, 5, 12, 10, 9, 1, 7]
midpoint = 3
left = [6, 5, 12]
right = [10, 9, 1, 7]

.
.
.

inputArray = [6, 5, 12]
midpoint = 1
left = [6]
right = [5, 12]

Title: Teach declarative knowledge / Retention and Transfer

Suggested Time: 3 mins

Materials / Handouts:

Description:
Describe additional debugging tools and strategies that can be used if available.

Instructor Actions:
Describe additional debugging tools and strategies that can be used if available.

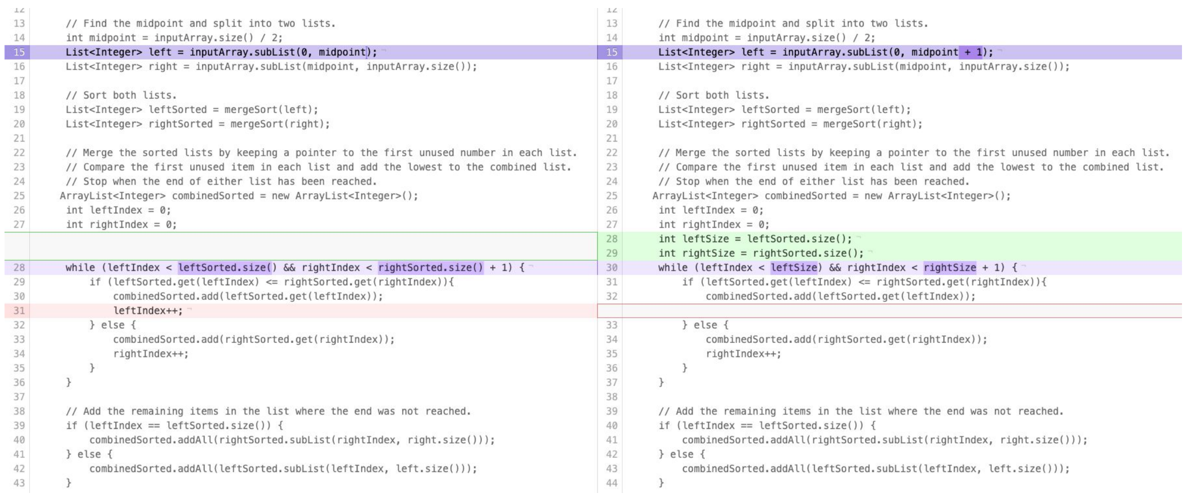
Learner Actions:
Actively listen and ask clarifying questions if needed.

Suggested Speaking Notes:
A debugger is a piece of software that allows you to step through your code in a much more efficient way. If you have an idea of where the bug might be but don't have the time to step through every single line a debugger can be particularly useful. The most common form of a debugger let's you set what's

called “breakpoints”. A breakpoint basically is stop side for the program at a particular line of code. For example, we might be interested in this particular line so we set a breakpoint, signalled by the red dot and then when we run the program every time it reaches that line of code it stops and waits for us. When it stops, we are also given a look at every variable in the scope of that line. So we might see the values on the top right the first time through and the values on the bottom right the second time through.

Now, if you don’t have the software to use a debugger, this can often be mimicked by adding print statements to the code. So if you’re interested in what left might look like at this line we can add a print statement to print the value of left after this line.

Diff



```
    // Find the midpoint and split into two lists.
    int midpoint = inputArray.size() / 2;
    List<Integer> left = inputArray.subList(0, midpoint);
    List<Integer> right = inputArray.subList(midpoint, inputArray.size());

    // Sort both lists.
    List<Integer> leftSorted = mergeSort(left);
    List<Integer> rightSorted = mergeSort(right);

    // Merge the sorted lists by keeping a pointer to the first unused number in each list.
    // Compare the first unused item in each list and add the lowest to the combined list.
    // Stop when the end of either list has been reached.
   ArrayList<Integer> combinedSorted = new ArrayList<Integer>();
    int leftIndex = 0;
    int rightIndex = 0;

    while (leftIndex < leftSorted.size() && rightIndex < rightSorted.size() + 1) {
        if (leftSorted.get(leftIndex) <= rightSorted.get(rightIndex)){
            combinedSorted.add(leftSorted.get(leftIndex));
            leftIndex++;
        } else {
            combinedSorted.add(rightSorted.get(rightIndex));
            rightIndex++;
        }
    }

    // Add the remaining items in the list where the end was not reached.
    if (leftIndex == leftSorted.size()) {
        combinedSorted.addAll(rightSorted.subList(rightIndex, right.size()));
    } else {
        combinedSorted.addAll(leftSorted.subList(leftIndex, left.size()));
    }
```

```
    // Find the midpoint and split into two lists.
    int midpoint = inputArray.size() / 2;
    List<Integer> left = inputArray.subList(0, midpoint + 1);
    List<Integer> right = inputArray.subList(midpoint, inputArray.size());

    // Sort both lists.
    List<Integer> leftSorted = mergeSort(left);
    List<Integer> rightSorted = mergeSort(right);

    // Merge the sorted lists by keeping a pointer to the first unused number in each list.
    // Compare the first unused item in each list and add the lowest to the combined list.
    // Stop when the end of either list has been reached.
   ArrayList<Integer> combinedSorted = new ArrayList<Integer>();
    int leftIndex = 0;
    int rightIndex = 0;
    int leftSize = leftSorted.size();
    int rightSize = rightSorted.size();
    while (leftIndex < leftSize) && rightIndex < rightSize + 1) {
        if (leftSorted.get(leftIndex) <= rightSorted.get(rightIndex)){
            combinedSorted.add(leftSorted.get(leftIndex));

        } else {
            combinedSorted.add(rightSorted.get(rightIndex));
            rightIndex++;
        }
    }

    // Add the remaining items in the list where the end was not reached.
    if (leftIndex == leftSorted.size()) {
        combinedSorted.addAll(rightSorted.subList(rightIndex, right.size()));
    } else {
        combinedSorted.addAll(leftSorted.subList(leftIndex, left.size()));
    }
```

Title: Teach declarative knowledge / Retention and Transfer

Suggested Time: 2 mins

Materials / Handouts:

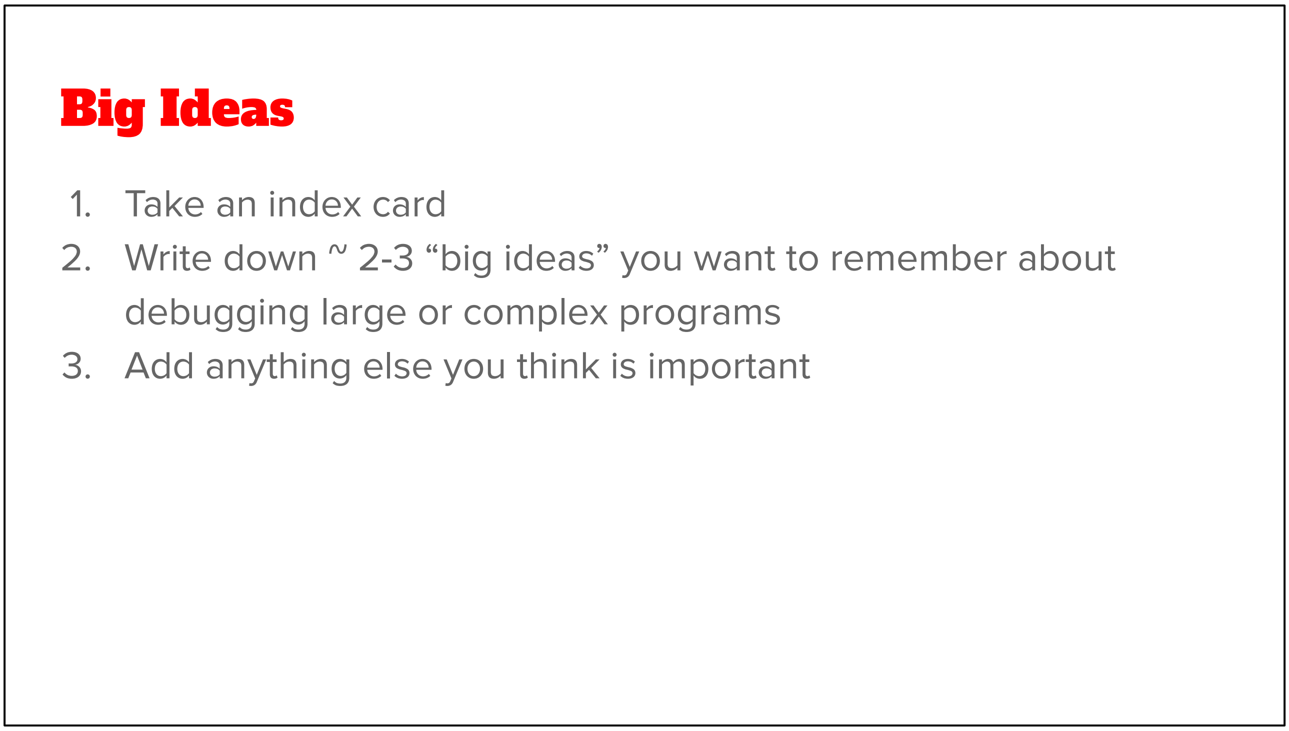
Description:
Describe additional debugging tools and strategies that can be used if available.

Instructor Actions:
Describe additional debugging tools and strategies that can be used if available.

Learner Actions:
Actively listen and ask clarifying questions if needed.

Suggested Speaking Notes:
Another useful tool is a diff. A diff is simply a way of visualizing what has changed between two things. In this case, we want to look at a diff of the program code. You'll see here that we have two versions of the same file. The diff highlights lines that have been added, removed, or modified. We can then

look at the lines that are highlighted and see if there are any unexpected changes or anything that might be related to the error.



Big Ideas

1. Take an index card
2. Write down ~ 2-3 "big ideas" you want to remember about debugging large or complex programs
3. Add anything else you think is important

Title: Big Ideas

Suggested Time: 3 mins

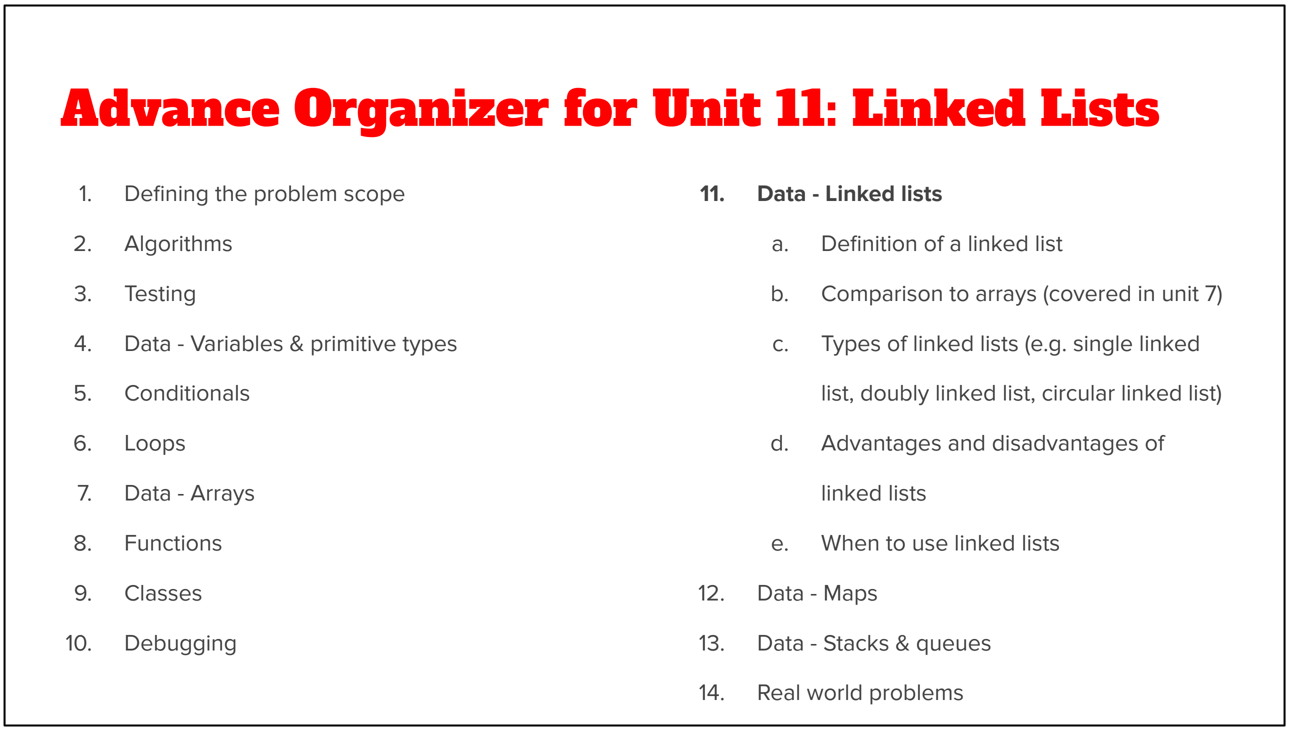
Materials / Handouts:
Index cards

Description:
Students create reference guides of big ideas to remember when debugging problems in the future.

Instructor Actions:
Provide students with index cards or pieces of paper to create reference guides. Ask learners to think about what big ideas they would want to remember next time they have an error with one of their programs and need to debug it.

Learner Actions:
Recall key concepts they want to remember and write them on their reference guide. Share ideas with the class and incorporate new ideas they like into their reference guide.

Suggested Speaking Notes:
You’re going to need to be able to use what we’ve covered today in the future so think about what big ideas you want to remember and write them down on your index card.



Advance Organizer for Unit 11: Linked Lists

1. Defining the problem scope
2. Algorithms
3. Testing
4. Data - Variables & primitive types
5. Conditionals
6. Loops
7. Data - Arrays
8. Functions
9. Classes
10. Debugging
11. **Data - Linked lists**
    a. Definition of a linked list
    b. Comparison to arrays (covered in unit 7)
    c. Types of linked lists (e.g. single linked list, doubly linked list, circular linked list)
    d. Advantages and disadvantages of linked lists
    e. When to use linked lists
12. Data - Maps
13. Data - Stacks & queues
14. Real world problems

Title: Advance Organizer for the Next Unit

Suggested Time: 1 min

Materials / Handouts:

Description:
Presentation of advance organizer explaining that the next several units will return to discussing different types of data structures with the next unit covering linked lists.

Instructor Actions:
Presents a quick overview of the units to come and answers any questions students have about what is coming up next.

Learner Actions:
Learners listen to what is coming up and ask any questions they might have.

Suggested Speaking Notes:
This wraps up our unit on debugging and here's a preview of what we'll cover in the next unit. We'll be looking at a new type of data structure, linked lists.