CMU 15-112: Fundamentals of Programming and Computer Science
Class Notes: Debugging

1. **Avoiding Bugs**
   Bugs are a natural part of the programming process. However, you can reduce the number of bugs you encounter by following a few tips:
   - Write code with good style.
   - Write tests before writing functions, and test as you go.
   - Make sure each function only has one task.
   - Avoid copying and pasting code at all costs (this leads to bug propagation).

2. **Debugging Syntax Errors**
   1. When debugging a syntax error, the most important thing you can do is **read the error message**!
   2. Start with the bottom line. It will verify that this is a syntax error.
   3. Look for the line number. It will appear at the bottom of the stack trace.
   4. Then look carefully at where in the line the arrow is pointing. That is usually where the syntax error is.
   5. Common syntax errors include:
      - Forgetting to close a string or parenthesis (may result in an EOF error)
      - Forgetting a colon
      - Using = instead of ==, or vice versa
      - Mismatched indentation (automatically convert tabs to spaces to avoid this!)
      - And many more...

3. **Debugging Runtime Errors**
   1. When debugging a runtime error, the most important thing you can do is, again, **read the error message**!
   2. Start with the bottom line. The type of the error will probably tell you what's going wrong.
   3. Look for the line number. It will appear at the bottom of the stack trace.
   4. Go to that line in the code, and identify what part of that line might be associated with the runtime error. If there are multiple possibilities, split them up into separate lines and run the code again.
   5. If it seems like the data you're inputting shouldn't result in that runtime error, try tracing the code of your program with the input. You can use print statements in your code to identify where your expectations diverge from what the code is doing. It's especially helpful to print the data on the line before the error occurs, to see what the state of the program is.
   6. Finally, determine how your algorithm needs to change to handle the new program state.
   7. Common runtime errors include:
      - String/list index errors

- Having a typo in a variable name
- Infinitely-recursing programs
- Trying to use an operation on an inappropriate type
- Dividing by zero
- Trying to read a file that doesn't exist
- And many more...

4. **Debugging Logical Errors**
    1. When debugging a logical error (aka, trying to figure out why you're failing a test case), the most important thing you can do is **identify the input, expected output, and actual output of the failing case.**
    2. Does the expected output make sense to you? If it does not, re-read the problem prompt until you understand it.
    3. Start tracing your code to determine when it starts behaving differently from what you expected. Usually this can be done by adding **print statements** at important conjunctures of the code, to display the current program state.
        - It helps to pair the values you're printing with meaningful labels. Instead of print(n), try print("n before for:", n).
        - It also helps to put some debugging statements into conditionals, so they're only printed when a certain (bad) condition is met.
        - When printing strings that include whitespace, use repr(s) to display the escape sequences in a more readable format.
        - If your program is very large, try using binary-search-print-debugging! Print the program state in the middle to see if the problem occurs before or after that point, move to the affected area, and repeat until the problem is found.
    4. Once you've found the location of the error, use problem-solving approaches to determine how your algorithm needs to be changed.