مكتب

Quick Reference Notes

Week 12: Debugging Workshop

7th Grade Computer Science

1 Introduction

1.1 This Week's Big Question

How do programmers solve problems when their code doesn't work? When your program crashes or gives wrong results, how can you systematically track down the issue instead of randomly guessing? Just like a detective solving a mystery, we'll learn to gather clues, form hypotheses, and test our theories!

Prerequisites

This week brings together everything you've learned! You should be comfortable with:

- Variables and data types (Week 3)
- Input/output and type conversion (Week 4)
- Expressions and operators (Week 5)
- If/else statements and conditions (Weeks 7-8)
- For loops and nested loops (Weeks 9-10)
- Combining loops with conditions (Week 11)

1.2 What You Already Know

You've written lots of Python programs and encountered various error messages. You know the frustration of code that doesn't work as expected. You've probably fixed bugs by trial and error, changing things until they work.

1.3 What You'll Be Able to Do

By the end of this week, you'll:

- Identify and classify different types of errors (syntax, runtime, logic)
- Read error messages like a pro to quickly locate problems
- Use systematic debugging strategies instead of random guessing
- Test hypotheses about why code isn't working
- Debug complex programs with multiple errors
- Help others debug their code effectively

2 VIDEO 1: Understanding Error Types

2.1 The Three Families of Errors

Just like illnesses have different types (cold, flu, stomach bug), Python errors come in three main families. Understanding which type you're dealing with helps you choose the right debugging strategy.

2.2 Syntax Errors: Grammar Mistakes

Syntax errors are like spelling or grammar mistakes in English. Python can't even start running your program because it doesn't understand what you wrote.

```
# Example 1: Missing colon
if age > 12
    print("You're a teenager!")

# Error message:
# SyntaxError: invalid syntax

# Example 2: Mismatched parentheses
print("Hello World"

# Error message:
# SyntaxError: unexpected EOF while parsing

# Example 3: Wrong indentation
for i in range(5):
print(i)

# Error message:
# IndentationError: expected an indented block
```

2.3 Runtime Errors: Crashes During Execution

Runtime errors happen when Python understands your code but crashes when trying to run it. It's like following a recipe that says "add the eggs" when you forgot to buy eggs!

```
# Example 1: Division by zero
students = 25
groups = 0
per_group = students / groups

# Error message:
# ZeroDivisionError: division by zero

# Example 2: Using undefined variable
print(score)  # Never created 'score'

# Error message:
# NameError: name 'score' is not defined

# Example 3: Wrong type conversion
age = input("Enter age: ")  # User types "thirteen"
age_next_year = int(age) + 1

# Error message:
# ValueError: invalid literal for int() with base 10: 'thirteen'
```

2.4 Logic Errors: Wrong Results

Logic errors are the trickiest - your program runs without crashing but gives wrong results. It's like following directions perfectly but ending up at the wrong destination!

```
# Example: Calculate average (but it's wrong!)
score1 = 80
score2 = 90
score3 = 85

# Wrong calculation - forgot parentheses!
average = score1 + score2 + score3 / 3
print(f"Average: {average}")

# Output: 255.33333...  (Should be 85!)
# The division happens first: 85/3 = 28.33, then 80+90+28.33
```

Rule

Error Type Quick Guide:

- **Syntax Error** = Won't run at all (red underline in editor)
- **Runtime Error** = Crashes while running (error message with line number)
- **Logic Error** = Runs but wrong output (need to trace through)

Quick Check

What type of error is this?

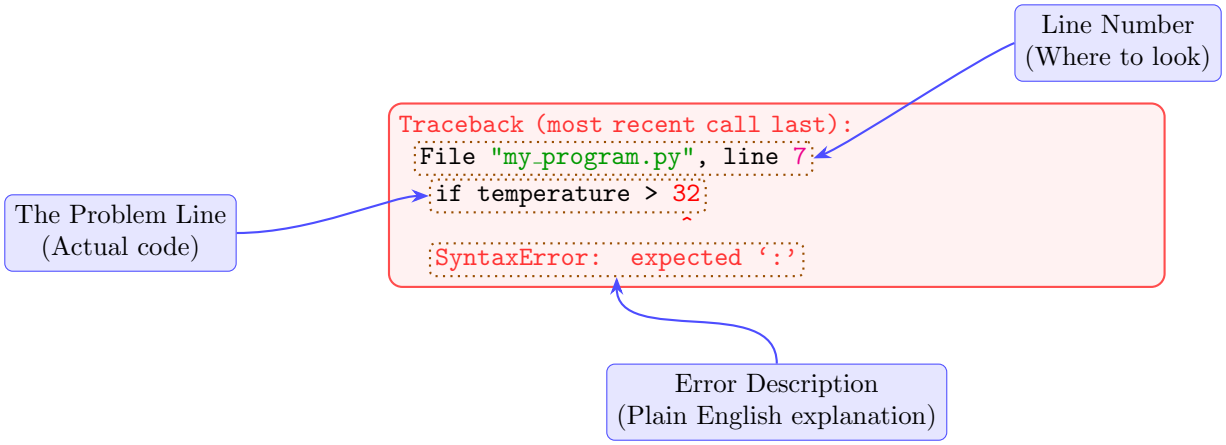
```
temperature = 100
if temperature > 32
    print("Above freezing!")
```

a) Syntax Error
b) Runtime Error
c) Logic Error

3 VIDEO 2: Reading Error Messages

3.1 Error Messages Are Your Friends!

Many students see error messages and panic. But error messages are actually helpful clues! They tell you exactly what went wrong and where to look.



3.2 Decoding Common Error Messages

Let's decode this error message:

```
name = input("Enter name: )

# Error message breakdown:
#   File "program.py", line 2
#     name = input("Enter name: )
#                  ^
# SyntaxError: unterminated string literal (detected at line 1)
```

Breaking it down:

- **File "program.py", line 2** → Look at line 2 of your file
- **The caret ^** → Points to where Python got confused
- **SyntaxError** → It's a grammar mistake

3.3 Common Error Messages Decoded

Error Message	Plain English	Common Cause
`SyntaxError: invalid syntax`	Grammar mistake	Missing : or ( ) etc
`IndentationError`	Spacing is wrong	Forgot to indent
`NameError: name 'x' is not defined`	Never created variable x	Typo or forgot to create
`TypeError: unsupported operand`	Can't do that operation	"5" + 5 (string + number)
`ValueError: invalid literal`	Can't convert	`int("hello")`
`ZeroDivisionError`	Divided by zero	Check your denominator
`SyntaxError: unexpected EOF...`	Couldn't find something	Missing ending " or)

Tip

Error Message Reading Strategy:

1. Look at the line number first
2. Read the error type (`SyntaxError`, `NameError`, etc.)
3. Check where the caret points
4. Read the description in simple terms
5. Look at the line above too *(error might start there)*

Quick Check

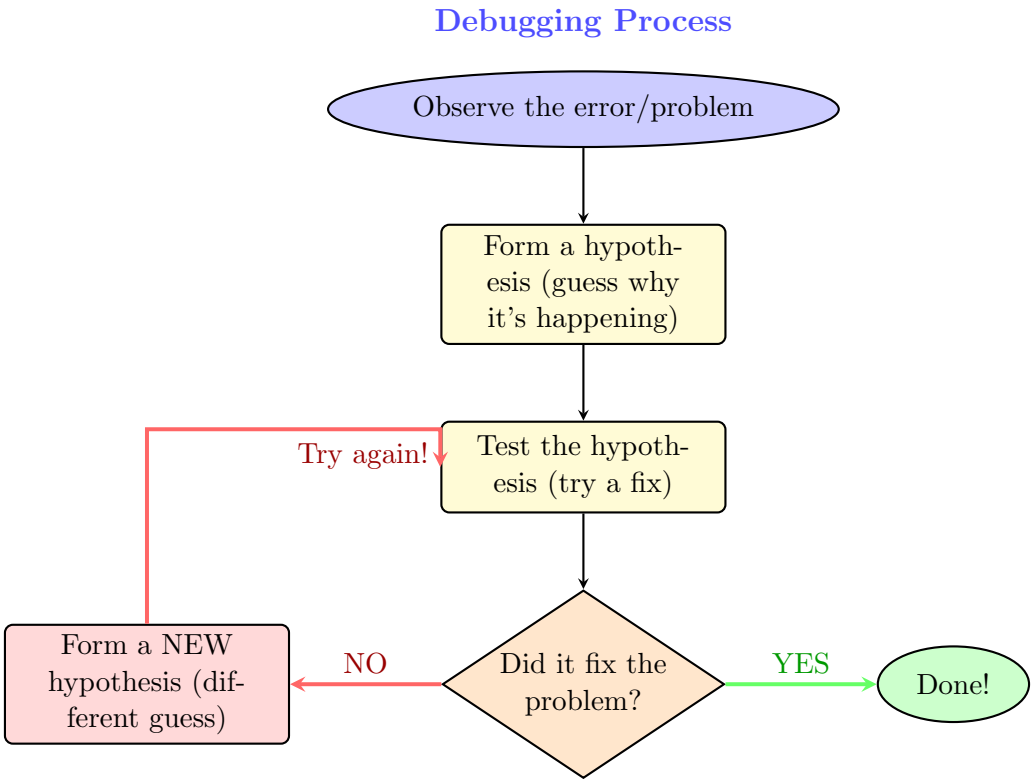
What's wrong based on this error?

```
File "test.py", line 3
    print("Hello"
          ^
SyntaxError: unexpected EOF while parsing
```

4 VIDEO 3: Systematic Debugging Strategies

4.1 The Scientific Method of Debugging

Debugging is like being a detective or scientist. Instead of randomly changing things, we follow a systematic process to find and fix problems.



4.2 Strategy 1: Print Statement Debugging

Add print statements to see what's happening inside your program - like leaving breadcrumbs to track your path.

```python
# Program to find largest of three numbers (has a bug!)
num1 = int(input("First number: "))
num2 = int(input("Second number: "))
num3 = int(input("Third number: "))

# Add debug prints to track values
print(f"DEBUG: num1={num1}, num2={num2}, num3={num3}")

largest = num1
if num2 > largest:
    largest = num2
    print(f"DEBUG: num2 is larger, largest now = {largest}")
if num3 > num2:  # BUG! Should compare with 'largest'
    largest = num3
    print(f"DEBUG: num3 > num2, largest now = {largest}")

print(f"The largest is: {largest}")
```

4.3 Strategy 2: Divide and Conquer

When your program has multiple parts and you're not sure where the bug is, isolate the problem by testing sections separately. It's like finding a broken bulb in a string of lights - test half at a time until you find it.

The Process:

1. Comment out approximately half your code
2. Run the program - does the error still happen?
3. If yes → bug is in the active code. If no → bug is in the commented code
4. Repeat with smaller sections until you isolate the problem

Example: Finding a Logic Bug
This discount calculator runs without errors but gives wrong results:

```
# Shopping discount calculator
print("=== Store Discount Calculator ===")

# Get purchase info
price = float(input("Item price: $"))
member = input("Are you a member? (yes/no): ")
day = input("What day is it? (weekday/weekend): ")

# Calculate base discount
discount = 0
if member == "yes":
    discount = 10  # Members get 10% off

# Weekend bonus discount
if day == "weekend":
    discount = discount + 5  # Extra 5% on weekends

if price > 50:
    discount = 15

# Calculate final price
discount_amount = price * discount / 100
final_price = price - discount_amount

print(f"\nOriginal price: ${price}")
print(f"Total discount: {discount}%")
print(f"You save: ${discount_amount}")
print(f"Final price: ${final_price}")
```

The Problem: A member buying a $75 item on weekend should get 30% off (10% + 5% + 15%) but only gets 15% off. Why?
Step 1: Comment out the bottom half to check if discounts are calculated correctly. Add debug prints to track the discount variable:

```
price = float(input("Item price: $"))
member = input("Are you a member? (yes/no): ")
day = input("What day is it? (weekday/weekend): ")

# Calculate base discount
discount = 0
if member == "yes":
    discount = 10
print(f"DEBUG: After member check: discount = {discount}")

# Weekend bonus discount
if day == "weekend":
    discount = discount + 5
print(f"DEBUG: After weekend check: discount = {discount}")

```

```
# # Special offer section - COMMENTED OUT
# if price > 50:
#     discount = 15
#
# # Final calculation - COMMENTED OUT
# discount_amount = price * discount / 100
# final_price = price - discount_amount
# print(f"\nOriginal price: ${price}")
# # ... rest of output
```

Result: Shows `discount = 15` after weekend check. Good so far!
Step 2: Uncomment the special offer section:

```
# Previous code...
print(f"After weekend check: discount = {discount}")

# Special offer
if price > 50:
    discount = 15  # AHA! This REPLACES the discount!
print(f"DEBUG: After special offer: discount = {discount}")  # Debug
```

Found it! The line `discount = 15` should be `discount = discount + 15`

Tip

Divide and Conquer Tips:

- Comment logical blocks (all input, all calculations, all output)
- Add `print("variable = ", variable)` after each section
- For logic bugs, trace how variables change through the program
- Save a backup before debugging: `program_backup.py`

4.4 Strategy 3: Rubber Duck Debugging

Explain your code line-by-line to a rubber duck (or a friend). Often, just explaining it helps you spot the problem!

Rule

Debugging Checklist:

1. Read the error message carefully
2. Check the line number mentioned
3. Add print statements before the error
4. Test with very small and simple input values
5. Comment out complex parts
6. Check one line above and below
7. Explain code out loud
8. Take a 5-minute break and return

Quick Check

Which debugging strategy would help find why this gives wrong output?

```
total = 0
for i in range(1, 4):
    total = i
print(total)
```

5 VIDEO 4: Common Bug Patterns and Fixes

5.1 The Usual Suspects

After debugging hundreds of programs, you'll notice the same bugs appear repeatedly. Let's learn to recognize these patterns!

5.2 Bug Pattern 1: Off-By-One Errors

These happen with loop ranges and list indices.

```
# Bug: Trying to print 1 to 10
for i in range(1, 10):  # Only goes to 9!
    print(i)

# Fix: Remember range stops BEFORE the end value
for i in range(1, 11):  # Now includes 10
    print(i)
```

5.3 Bug Pattern 2: Variable Name Typos

Python is case-sensitive and spelling matters!

```
# Bug: Inconsistent variable names
Score = 85
bonus = 10
total = score + bonus  # NameError: 'score' not defined

# Fix: Be consistent with capitalization
score = 85
bonus = 10
total = score + bonus  # Now it works!

# Bug: Similar names
count = 0
for i in range(10):
    count = count + 1
print(counts)  # NameError: 'counts' not defined
```

5.4 Bug Pattern 3: Type Confusion

Mixing strings and numbers causes many bugs.

```
# Bug: Forgot to convert input
age = input("Enter age: ")  # Returns string "13"
if age > 12:  # Can't compare string to number!
    print("Teenager!")

# Fix: Convert to int
```

```
age = int(input("Enter age: "))
if age > 12:
    print("Teenager!")

# Bug: Concatenating instead of adding
num1 = input("First number: ")  # "5"
num2 = input("Second number: ")  # "3"
total = num1 + num2  # Results in "53" not 8!
```

Common Error

Top 5 Beginner Bugs:

1. Forgetting colons after if/for/while
2. Wrong indentation (too much/too little)
3. Using = instead of == in comparisons
4. Forgetting to convert input() to int/float
5. Off-by-one errors in range()

Check for these first when debugging!

Quick Check

What type of bug is this?

```
password = input("Enter password: ")
if password = "secret123":  # Bug here!
    print("Access granted")
```

6 VIDEO 5: Summary

6.1 You're Now a Debugging Detective!

This week transformed you from someone who fears error messages into a systematic problem-solver. You've learned that debugging is a skill that improves with practice, and errors are learning opportunities, not failures.

6.2 Your Debugging Toolkit

Error Type	Detection Method	Fix Strategy
Syntax Errors	Won't run, red underlines	Check colons, quotes, parentheses
Runtime Errors	Crashes with error message	Read message, check that line
Logic Errors	Wrong output	Add prints, trace through logic

6.3 Debugging Process Summary

1. **Don't Panic** - Errors are normal and fixable
2. **Read Carefully** - Error messages contain valuable clues
3. **Locate the Problem** - Use line numbers and error types
4. **Form Hypothesis** - Make an educated guess about the cause

5. **Test Systematically** - Try one fix at a time
6. **Use Debug Prints** - See what's happening inside
7. **Learn the Pattern** - Remember for next time

Landmark Moment

You've mastered one of the most important skills in programming - systematic debugging! Every programmer, from beginners to experts at Google, spends time debugging. The difference is that experts debug systematically while beginners guess randomly. You now have the tools to approach any error with confidence. This skill will serve you in all future programming and problem-solving!

7 Quick Reference

Quick Reference

Week 12 Debugging Quick Reference:
Error Types:

- **Syntax**: Code won't run (grammar mistakes)
- **Runtime**: Code crashes while running
- **Logic**: Code runs but gives wrong results

Common Error Messages:

```
SyntaxError         # Missing :, quotes, or ()
IndentationError    # Wrong spacing after :
NameError           # Variable doesn't exist
TypeError           # Wrong type for operation
ValueError          # Can't convert (e.g., int("abc"))
ZeroDivisionError   # Divided by zero
```

Debug Strategies:

```
# 1. Print debugging
print(f"DEBUG: variable = {variable}")

# 2. Divide and conquer
# Comment out half the code

# 3. Test with simple input
# Use 1, 2, 3 instead of complex values

# 4. Check one line above/below error
```

Common Fixes:

- Add missing colons after if/for/while
- Fix indentation (4 spaces after :)
- Change = to == in conditions
- Add int() around input() for numbers
- Check range() end values (stops before end)
- Verify variable name spelling/capitalization

7.1 Reflection

Think about your debugging journey this week:

1. Error messages used to be scary. How do you feel about them now?
2. Which type of error (syntax, runtime, logic) do you find easiest to fix? Why?
3. Print debugging helps us see inside our programs. What other real-world problems could benefit from "making the invisible visible"?
4. Why is systematic debugging better than random guessing? Can you think of other areas where a systematic approach helps?

5. What was your most frustrating bug this semester? How would you debug it differently now?
6. When helping a friend debug, what's the first question you would ask them?
7. How is debugging like being a detective? What skills do both require?

Next week, we'll celebrate your programming journey with creative Python Turtle Graphics projects!