CSC 141 / FALL 2026

Testing and debugging decisions

CSC 141 · Class 7
Thursday, September 17, 2026
Dr. Si Chen

Today’s learning goals

Choose tests for each branch and each boundary.
Distinguish syntax, runtime, and logic errors.
Use a trace to locate the first wrong value.
Repair one issue and rerun the relevant tests.

Warm-up: predict before running

```
temperature = 18
if temperature < 18:
    print("Jacket")
else:
    print("Light layer")
```

What prints at 18?

What changes at 17?

What changes at 19?

A test needs an expected result

A test case contains:
An input.
The expected output from the rule.
The actual output after running.
A comparison of expected and actual.

“Python did not crash” is only one observation.

Boundary tests

```
Rule: free delivery for totals of $30 or more.

Input        Expected delivery charge
29.99        $4.00
30.00        $0.00
30.01        $0.00

The exact threshold distinguishes > from >=.
```

Branch coverage

For an if/else, test a True path and a False path.
For an elif chain, choose a case for every output.
For nested decisions, test the outer False path and each reachable inner path.

Passing these cases builds evidence; it does not prove every possible input is correct.

Valid input assumptions

This week's examples assume the stated input types.
A ticket count is a whole number.
A price may have a decimal part.
A yes/no prompt receives yes or no.

Invalid-input recovery will need additional techniques later.

Three kinds of errors

Syntax error: Python cannot read the program.
Runtime error: an operation fails while the program runs.
Logic error: the program runs but does not follow the rule.

Syntax error: assignment in a condition

```
score = 10

if score = 10:
    print("Ten")
```

The condition needs == for equality.

Read the reported line and inspect nearby punctuation.

Runtime error: text mixed with a number

```
count = input("Tickets: ")
next_count = count + 1
```

input returns str.

Text + int raises TypeError.

Convert the input before addition.

Logic error: the wrong boundary

```
total = 30.0

if total > 30:
    delivery = 0.0
else:
    delivery = 4.0
```

Rule: free delivery at $30 or more.

At exactly $30, this code charges $4.

The condition needs >=.

A short debugging routine

Reproduce the problem with one input.
Write the expected result.
Read the last error line, if one appears.
Trace until the first unexpected value or branch.
Change one cause, save, and run again.
Rerun the earlier tests after the fix.

Trace table: a delivery calculator

subtotal	subtotal >= 30	delivery	total
25.00	False	4.00	29.00
30.00	True	0.00	30.00
42.00	True	0.00	42.00

Temporary prints reveal intermediate values

```
subtotal = float(input("Subtotal: "))
print("DEBUG subtotal:", subtotal)
print("DEBUG qualifies:", subtotal >= 30)

if subtotal >= 30:
    delivery = 0.0
else:
    delivery = 4.0
```

Inspect both the value and the condition.

Remove temporary prints when the final output is correct.

Guided repair: free delivery

```
subtotal = input("Subtotal: ")
if subtotal > 30:
    delivery = 0.0
else:
    delivery = 4.0
total = subtotal + delivery
print("Total: " + total)
```

Rule: totals of $30 or more receive free delivery.

Find the input-type problem, the boundary problem, and the output-type problem.

Delivery solution after your attempt

```
FREE_DELIVERY_MIN = 30.0
DELIVERY_FEE = 4.0
subtotal = float(input("Subtotal: "))

if subtotal >= FREE_DELIVERY_MIN:
    delivery = 0.0
else:
    delivery = DELIVERY_FEE

total = subtotal + delivery
print(f"Total: ${total:.2f}")
```

Test 29.99, 30.00, and 30.01.

Expected totals: $33.99, $30.00, $30.01.

Independent practice: a study-room session

Fictional rules:
Under 30 minutes: $0.00.
30–60 minutes, inclusive: $2.00.
More than 60 minutes: $4.00.

Read nonnegative whole minutes. Print one fee.
Use if / elif / else.

Test plan before implementation

Minutes	Expected fee
0	$0.00
29	$0.00
30	$2.00
60	$2.00
61	$4.00

Explain what each boundary case checks.

A small checkpoint for each step

First run: the input converts to an int.
Second run: the program chooses the fee.
Third run: the money has two decimal places.
Final check: all planned tests agree.

If stuck, show the first checkpoint that fails.

Explain your repair

With a partner:
Name one input you tested.
State the expected and actual output.
Identify the condition or value that went wrong.
Explain the change you made.
Run the boundary tests again.

Exit ticket and PA1 reminder

What is the difference between a runtime error and a logic error?
Which three nearby inputs test a threshold of 50?
Why should you rerun old tests after a change?

PA1 is due tonight at 11:59 PM in D2L.