

CSC 141: Computer Science I

Programming Assignment 2

Decisions, Program Design, Testing, and Debugging

Dr. Si Chen · West Chester University · Fall 2026

Assigned	Friday, September 18, 2026
Due	Thursday, October 1, 2026, 11:59 PM (submit on D2L)
Points	100 points (one of five programming assignments; 30% of the course grade in total)
Work	Individual. See the collaboration and AI rules in Section 2.
Covers	Classes 4–7: Boolean expressions, <code>if/elif/else</code> , program design, nested decisions, tracing, boundary testing, and syntax/runtime/logic errors

Dates

D2L is authoritative for the due date. The D2L timestamp decides whether a submission is on time.

1 Overview and learning goals

This assignment builds on Programming Assignment 1 with everything from Classes 4–7. You will trace nested decisions by hand, design test cases *before* writing code, build four decision-driven programs, and debug a program with three separate, specific bugs. After finishing it you should be able to:

- trace an `if/elif/else` chain and a nested decision and state exactly what runs;
- explain why `elif` order matters and find a bug caused by the wrong order;
- choose test inputs that check a rule below, at, and above each boundary, before writing the program;
- build multi-branch and nested decisions that assign every output on every path;
- tell a syntax error, a runtime error, and a logic error apart, and fix each one.

2 Ground rules

- **Individual work.** You may discuss concepts and help a classmate spot an error, but you may not exchange solution code, write a solution together, or submit anyone else's work.
- **Generative AI.** You may ask an AI tool to explain a concept or an error message. You may *not* ask it to write any part of these programs and submit that output as your own. You must be able to explain and modify every line you submit. The instructor may ask you to.
- **Python 3 and VS Code.** Write and test your programs in VS Code (the browser playground at <https://wcu.ninja/csc141/python> is fine for experimenting, but submit `.py` files).
- **Partial credit.** A program that does not fully work can still earn credit if a comment at the top clearly says what works, what does not, and what you tried. A blank submission earns nothing.
- **Late work.** Late submissions receive no credit unless a valid excuse is communicated *before* the deadline.

Requirements for every .py file

1. Start with a header comment:

```
# CSC 141 - Programming Assignment 2 - <file name>
# Name: <your full name>
# Date: <date you finished>
# Description: <one or two sentences: what this program does>
```

2. Use descriptive `snake_case` names and `UPPER_CASE` constants for fixed values, as in Programming Assignment 1.
3. Convert `input()` text with `int()` or `float()` before comparing it or doing math.
4. Every path through your decisions must assign the values the sample runs expect; do not leave a variable unset on any branch.
5. Match the output format shown in the sample runs. Spacing and rounding are part of the grade.
6. Your program must not crash on valid input. You do *not* need to handle invalid input (for example, a user typing `abc` when a number is expected).

3 Part A: Reading and tracing decisions (20 points)

Answer Part A in a plain-text file named `partA.txt`. Do **not** run the code first. Trace it by hand, write your answer, and only then check yourself in Python. If you were wrong, keep your original answer and add one sentence explaining what you misunderstood. (Honest wrong answers with a correct explanation earn full credit.)

A1. Trace a nested decision

10 points

For each pair of starting values, write exactly what this program prints. Two points each.

```
lab_open = True
has_key = False

if lab_open:
    if has_key:
        print("Enter the lab")
    else:
        print("Wait for staff")
else:
    print("Lab closed")
```

- (a) `lab_open = True, has_key = True`
- (b) `lab_open = True, has_key = False`
- (c) `lab_open = False, has_key = True`
- (d) `lab_open = False, has_key = False`
- (e) Change only the last line of the outer `else` to also check `has_key` in a new nested decision that prints "Locker only" when the lab is closed but the student has a key, otherwise still prints "Lab closed". Write the two new lines you would add, and where.

A2. Find the elif-order bug**5 points**

A student wrote this to award a badge based on points. For `points = 50`, they expected "Gold" but the program prints "Silver".

```
points = 50
if points >= 10:
    print("Silver")
elif points >= 30:
    print("Gold")
else:
    print("Bronze")
```

- (a) Explain exactly why "Silver" prints instead of "Gold".
- (b) Rewrite the three conditions in the order that fixes the bug for every point value.

A3. Explain the runtime error**5 points**

A student wrote the program below. When they type 15 at the prompt, Python stops with an error.

```
minutes_late = input("Minutes late? ")
if minutes_late >= 15:
    print("Marked absent")
else:
    print("Marked late")
```

- (a) Copy the exact error message Python prints (the last line of the traceback).
- (b) In one or two sentences, explain *why* the error happens. Mention the type of `minutes_late`.
- (c) Show the corrected line of code.

4 Part B: Design your test cases first**(10 points)**

Before writing `library_fine.py` (Program 2 below), write your test plan in a plain-text file named `test_plan.txt`. For the fine rule described in Program 2, list **at least six** test cases as `days_late` -> `expected fine`, chosen to check every boundary in the rule (just below, exactly at, and just above each threshold), the same way Class 7 tested the free-delivery boundary. Write the plan *first*; then turn it into code and confirm your program matches every row.

5 Part C: Programs**(60 points)**

Each program reads its input with `input()` and prints its result with f-strings. The sample runs show what the user typed after each prompt; everything else is printed by your program.

Program 1: grade_bracket.py**15 points**

Ask for a numeric score from 0 to 100. Use `if/elif/else` to print the letter grade (90 or above is A; 80 or above is B; 70 or above is C; 60 or above is D; anything else is F) and one study tip that matches the grade, exactly as shown.

```
Score: 95
Grade: A
Tip: Keep up the strong work!
```

```
Score: 82
Grade: B
Tip: Review missed topics before the exam.
```

```
Score: 55
Grade: F
Tip: See the instructor immediately.
```

Tips (use exactly): A – *Keep up the strong work!* B – *Review missed topics before the exam.* C – *Meet with a tutor this week.* D – *See the instructor during office hours.* F – *See the instructor immediately.* *Hint:* test the highest threshold first, the same way Class 6 fixed the beacon-rank order bug.

Program 2: library_fine.py (include your Part B test plan)**20 points**

A fictional library charges no fine for the first 7 days a book is late. Days 8 through 14 cost \$0.25 per day *in that range*. Every day after day 14 costs \$0.50 per day, on top of the fine already owed for days 8–14. Ask for the number of days late (a whole number) and print the fine.

```
Days late: 5
Fine: $0.00
```

```
Days late: 8
Fine: $0.25
```

```
Days late: 14
Fine: $1.75
```

```
Days late: 20
Fine: $4.75
```

Hints: day 14 owes exactly \$1.75 (7 days in the \$0.25 range). A day-20 fine is the day-14 fine plus 6 more days at \$0.50. Do not recompute the full amount at the \$0.50 rate; only the days past 14 use that rate.

Program 3: movie_ticket.py (a nested decision)**15 points**

Ask for an age (a whole number), whether it is the weekend (**yes/no**), and whether it is a matinee showing (**yes/no**). Base price: under 13 is \$8.00; 65 or older is \$9.00; everyone else is \$14.00 on a weekend and \$11.00 on a weekday. **Then**, if it is a matinee, subtract \$3.00 from whatever base price you found – but the final price never drops below \$6.00.

```
Age: 10
Weekend? (yes/no): no
Matinee? (yes/no): no
Price: $8.00
```

```
Age: 30
Weekend? (yes/no): yes
Matinee? (yes/no): no
Price: $14.00
```

```
Age: 8
Weekend? (yes/no): yes
Matinee? (yes/no): yes
Price: $6.00
```

Hints: find the base price with one decision, then apply the matinee adjustment with a *second*, separate decision that checks the matinee answer and, inside it, checks whether subtracting \$3.00 would go below \$6.00. This is exactly the “nested decision inside a plan” pattern from the Class 6 workshop-ticket example.

Program 4: shipping_zone.py**10 points**

Ask for a shipping distance in miles (may include a decimal) and whether the order is rush (**yes/no**). The base fee is \$3.00 for 10 miles or fewer, \$6.00 for more than 10 and up to 50 miles, and \$12.00 for anything farther. A rush order adds a flat \$5.00 on top of the base fee.

```
Distance (miles): 10
Rush? (yes/no): no
Fee: $3.00
```

```
Distance (miles): 10.01
Rush? (yes/no): no
Fee: $6.00
```

```
Distance (miles): 51
Rush? (yes/no): no
Fee: $12.00
```

Hint: test 10 and 10.01 separately, the same way Class 7 tested \$29.99 against \$30.00. This fee does *not* accumulate across tiers the way Program 2’s fine does – only one tier’s fee applies.

6 Part D: Debug the AI-written program (10 points)

Someone asked an AI assistant for a program that gives a small attendance bonus and flags a student as late. The AI produced the code below. It looks reasonable, but it contains **three bugs**: one **syntax** error, one **runtime** error caused by a type, and one **logic** error caused by the wrong boundary comparison for a rule stated as “15 minutes or more is late.”

Save this code as `debug_attendance.py`, trace it by hand for `minutes_late = 15` and `attended = yes` before running it, then fix all three bugs. At the top of the file, add a comment that lists each bug, the line it was on, and how you fixed it.

```
# Gives a $2 bonus for attending, and flags anyone 15+ minutes late.
attended = input("Attended? (yes/no): ")
minutes_late = input("Minutes late: ")

if attended = "yes":
    bonus = 2.00
else:
    bonus = 0.00

if minutes_late > 15:
    status = "Late"
else:
    status = "On time"

print("Bonus: $" + bonus)
print("Status:", status)
```

The corrected program must print exactly the following for `attended = yes` and `minutes_late = 15`:

```
Attended? (yes/no): yes
Minutes late: 15
Bonus: $2.00
Status: Late
```

7 Grading

Component	Points	What earns full credit
Part A: reading and tracing decisions	20	Correct traces; correct elif-order fix; clear error explanation.
Part B: test plan	10	At least 6 cases; every boundary covered; matches the final program.
Program 1: <code>grade_bracket.py</code>	15	Correct bracket order; exact grade and tip text.
Program 2: <code>library_fine.py</code>	20	Correct tiered math; boundaries at 7 and 14 exactly right.
Program 3: <code>movie_ticket.py</code>	15	Correct base price; matinee adjustment applied and floored correctly.
Program 4: <code>shipping_zone.py</code>	10	Correct single-tier fee; boundary at 10 and 50 exactly right.
Part D: debugging	10	All three bugs found, explained, and fixed; output matches exactly.

Code quality (applied across all programs, up to –10): missing header comment, vague names (`x`, `temp`, `data2`), magic numbers instead of named constants, unconverted `input()` values, or a variable left unassigned on some path will lose points even when the visible output happens to look correct.

8 What to submit

Upload a single ZIP file named `Lastname_Firstname_PA2.zip` to the **Programming Assignment 2** dropbox on D2L. It must contain exactly these seven files:

<code>partA.txt</code>	Part A answers (plain text)
<code>test_plan.txt</code>	Part B test plan (plain text)
<code>grade_bracket.py</code>	Program 1
<code>library_fine.py</code>	Program 2
<code>movie_ticket.py</code>	Program 3
<code>shipping_zone.py</code>	Program 4
<code>debug_attendance.py</code>	Part D, fixed, with the bug list comment at the top

Before you click Submit

- ☐ Every `.py` file has the header comment with your name.
- ☐ Each program runs from start to finish in VS Code with the sample inputs and matches the sample output.
- ☐ You tried at least one boundary input of your own for each program and the answer makes sense.
- ☐ `library_fine.py` matches every row of your `test_plan.txt`.
- ☐ `debug_attendance.py` contains your three-bug list comment.
- ☐ The ZIP is named correctly and contains only the seven files above.

Boundaries are not optional

“8 through 14” includes both 8 and 14. “15 minutes or more” includes exactly 15. If your program uses `>` where the rule means `>=`, it will pass every test except the one at the exact boundary – which is exactly the test we will run.

Getting help

Office hours: Tue/Thu 12:30–2:00 PM (by appointment) and Wed 1:00–3:00 PM, 25 University Ave., Room 131. When you email (schen@wcupa.edu), say which program you are working on, paste the exact error message, and describe what you have already tried.