

CSC 141: Computer Science I

Programming Assignment 3

Loops, Accumulators, Nested Loops, and Strings

Dr. Si Chen · West Chester University · Fall 2026

Assigned	Tuesday, October 20, 2026
Due	Tuesday, November 3, 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 14–16: while and for loops, range() , the accumulator and counting patterns, nested loops, and strings (indexing, slicing, traversal)

Dates

D2L is authoritative for the due date. The D2L timestamp decides whether a submission is on time. Class 16 (Thursday, October 22) covers the string material used in Program 3; everything else is in Classes 14–15, so you can start Programs 1, 2, and 4 right away.

1 Overview and learning goals

This assignment uses everything from Classes 14–16. You will trace loops by hand, design test cases *before* writing code, build four loop-driven programs, and debug a program whose three bugs are the three classic loop mistakes. After finishing it you should be able to:

- trace a **while** loop with a table and state exactly what it prints and when it stops;
- predict the values a **range()** call produces, including a negative step;
- use the accumulator pattern (totals, best value, built-up strings) and the counting pattern;
- write a loop inside a loop and predict how many times the inner body runs;
- read one character of a string with an index, take a slice, and loop over its characters;
- recognize and fix a missing update, an accumulator reset inside the loop, and an off-by-one **range()**.

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 3 - <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 (for example `DAYS = 7`, `VOWELS = "aeiou"`).
3. Convert `input()` text with `int()` or `float()` before comparing it or doing math.
4. Every accumulator (`total`, `count`, a built-up string) starts *before* its loop, changes *inside* the loop, and is used *after* the loop.
5. Every **while** loop must have an update that eventually makes its condition **False**.
6. Match the output format shown in the sample runs. Spacing and rounding are part of the grade.
7. 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 loops (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 while loop

8 points

Trace this program with a table that has one row per pass: the value of `n` at the start of the pass, whether `n < 20` is **True** or **False**, and `total` after the pass. Then write exactly what the `print` line prints.

```
n = 1
total = 0
while n < 20:
    total += n
    n = n * 2
print(n, total)
```

A2. range() values**6 points**

Write the values each call produces, in order (write **nothing** if there are none). Then answer (d).

- (a) `range(3, 12, 3)`
- (b) `range(10, 0, -3)`
- (c) `range(2, 2)`
- (d) How many passes does `for i in range(5, 15, 2):` make? List the values of `i`.

A3. Index and slice a string**6 points**

Let `s = "computer"`. Write the value of each expression. For (g), write the exact name of the error Python raises.

- (a) `s[0]` (b) `s[-1]` (c) `s[2:5]` (d) `s[:3]` (e) `s[5:]` (f) `len(s)`
- (g) `s[8]`

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

Before writing `savings_plan.py` (Program 2 below), write your test plan in a plain-text file named `test_plan.txt`. List **at least five** test cases as `goal, start, deposit, rate -> expected months`. Your cases must include: a run where the starting balance already meets the goal (the loop body runs zero times); a run that takes exactly one month; a run where interest, not just the deposit, is what pushes the balance over the goal; and a run with a rate of 0. Add one sentence describing an input that would make the loop never end, and why the program does not need to handle it. 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: step_counter.py (for + range, accumulator, counting)**15 points**

Ask for a daily step goal, then ask for the step count on each of 7 days using a **for** loop over **range()** (the prompt must number the days 1 through 7). After the loop print the total, the average per day with one decimal place, the best (largest) single day, and how many days met or beat the goal. Use a constant **DAYS = 7**; the only number 7 in your code should be that line.

```
Daily step goal: 8000
Day 1 steps: 9000
Day 2 steps: 7500
Day 3 steps: 12000
Day 4 steps: 8000
Day 5 steps: 4000
Day 6 steps: 10000
Day 7 steps: 8000
Total steps: 58500
Average per day: 8357.1
Best day: 12000
Days that met the goal: 5 of 7
```

Hints: three accumulators live before the loop: a total, a best-so-far, and a count. “Met the goal” means $\geq$, so day 4 and day 7 both count. The best-so-far starts at 0 and is replaced whenever a larger value appears.

Program 2: savings_plan.py (while until a goal; include your Part B test plan) 20 points

Ask for a savings goal, a starting balance, a monthly deposit, and a monthly interest rate in percent (all may have decimals). Simulate one month at a time with a `while` loop: add the deposit, then add interest equal to the new balance times the rate divided by 100, and print the month number and the balance. Stop as soon as the balance reaches the goal. After the loop print how many months it took and the final balance. If the starting balance already meets the goal, the loop body must not run at all.

```
Savings goal: 1000
Starting balance: 200
Monthly deposit: 150
Monthly interest rate (%): 1
Month 1: $353.50
Month 2: $508.54
Month 3: $665.12
Month 4: $823.27
Month 5: $983.00
Month 6: $1144.33
Goal reached in 6 months.
Final balance: $1144.33
```

```
Savings goal: 500
Starting balance: 500
Monthly deposit: 50
Monthly interest rate (%): 1
Goal reached in 0 months.
Final balance: $500.00
```

Hints: month 1 is $(200 + 150) \times 1.01 = 353.50$. The condition is “while the balance is still below the goal”; the deposit is the update that moves it toward `False`. This is the `weeks_to_goal` pattern from Class 14 with a second line of arithmetic in the body.

Program 3: word_stats.py (strings: index, loop, build)**15 points**

Ask for one word (letters only, no spaces). Print its length, its first and last characters (use indexing, not a loop), the word reversed (build it with a loop, not a slice), the number of vowels (a e i o u, counted case-insensitively), and a “masked” copy in lower case with every vowel replaced by *. Use a constant `VOWELS = "aeiou"` and `in` to test each character.

```
Word: Programming
Length: 11
First: P Last: g
Reversed: gnimmargorp
Vowels: 3
Masked: pr*gr*mm*ng
```

```
Word: sky
Length: 3
First: s Last: y
Reversed: yks
Vowels: 0
Masked: sky
```

Hints: one `for ch in ...` loop can do all three jobs: put `ch` in front of the reversed string, test `ch` in `VOWELS`, and append either `*` or `ch` to the masked string. Lower-case the word once, before the loop. Note the two spaces between `First: P` and `Last: g`.

Program 4: times_grid.py (nested loops)**10 points**

Ask for a grid size `n` (a whole number from 1 to 9) and print the $n \times n$ multiplication grid with nested `for` loops. Every number is right-aligned in a column 4 characters wide (use an f-string width, for example `f"{value:4}"`), and each row ends with a newline produced by a bare `print()`.

```
Grid size: 4
 1  2  3  4
 2  4  6  8
 3  6  9 12
 4  8 12 16
```

Hints: the outer loop picks the row, the inner loop picks the column, and `print(..., end="")` keeps the numbers on one line. Test with `n = 1` (one number) and `n = 9` (the widest row).

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

Someone asked an AI assistant for a program that reads the points scored in a number of rounds, prints the round numbers, and prints the total. The AI produced the code below. It contains the **three classic loop bugs**: a `while` loop with **no update** (it never ends), an accumulator that is **reset inside the loop**, and an **off-by-one** `range()`.

Save this code as `debug_tally.py`. Trace it by hand first for `rounds = 3` and points 10, 15, 5 –

you should be able to find all three bugs without running it. At the top of the file, add a comment that lists each bug, the line it was on, and how you fixed it.

Before you run it

As written, the first loop never ends. In VS Code, stop it with **Ctrl+C**. Do *not* run the unfixed version in the browser playground: an endless loop freezes the page.

```
# Tallies points over a number of rounds and lists the round numbers.
rounds = int(input("Rounds: "))

count = 0
while count < rounds:
    total = 0
    points = int(input("Points: "))
    total += points

print("Round numbers:", end=" ")
for k in range(1, rounds):
    print(k, end=" ")
print()
print(f"Total points: {total}")
```

The corrected program must print exactly the following for `rounds = 3` and points 10, 15, 5:

```
Rounds: 3
Points: 10
Points: 15
Points: 5
Round numbers: 1 2 3
Total points: 30
```

7 Grading

Component	Points	What earns full credit
Part A: reading and tracing loops	20	Complete trace table; correct <code>range()</code> values; correct index/slice values and error name.
Part B: test plan	10	At least 5 cases including the four required kinds; the never-ending input explained; matches the final program.
Program 1: <code>step_counter.py</code>	15	<code>for</code> + <code>range()</code> with numbered prompts; total, average, best, and count all correct; <code>DAYS</code> constant.
Program 2: <code>savings_plan.py</code>	20	<code>while</code> with the right condition and update; month lines and totals match to the cent; zero-month case works.
Program 3: <code>word_stats.py</code>	15	Indexing for first/last; reversed built in a loop; vowel count and mask correct for mixed case.
Program 4: <code>times_grid.py</code>	10	Nested loops; right-aligned 4-wide columns; correct for 1 through 9.
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 (<code>x</code> , <code>temp</code> , <code>data2</code>), magic numbers instead of named constants, unconverted <code>input()</code> values, an accumulator initialized inside its loop, or a loop written by copying the body several times instead of looping will lose points even when the visible output happens to look correct.		

8 What to submit

Upload a single ZIP file named `Lastname_Firstname_PA3.zip` to the **Programming Assignment 3** 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>step_counter.py</code>	Program 1
<code>savings_plan.py</code>	Program 2
<code>word_stats.py</code>	Program 3
<code>times_grid.py</code>	Program 4
<code>debug_tally.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.
- ☐ Every `while` loop has an update, and every accumulator starts before its loop.
- ☐ `savings_plan.py` matches every row of your `test_plan.txt`, including the zero-month case.
- ☐ `word_stats.py` gives the same vowel count for `Programming` and `PROGRAMMING`.

- ☐ `debug_tally.py` contains your three-bug list comment and ends on its own.
- ☐ The ZIP is named correctly and contains only the seven files above.

The three loop bugs

`range(1, n)` stops at `n - 1`: use `range(1, n + 1)` when `n` itself belongs in the loop. `total = 0` inside the loop throws away every earlier pass. A `while` loop whose body never changes the condition variable runs forever. Part D contains all three on purpose; make sure Programs 1–4 contain none of them.

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. Loop practice with instant feedback: PokéLab Phase V (The Loop Trails) and the Class 14–16 examples in the Python Playground.