CSC 141: Computer Science I

Programming Assignment 1

Variables, Types, Expressions, and Formatted Output

Dr. Si Chen · West Chester University · Fall 2026

Assigned	Thursday, September 3, 2026
Due	Thursday, September 17, 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 1–3: computational thinking, `print()`, escape sequences, values and types, variables and assignment, arithmetic operators, `input()` and conversion, f-strings

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 puts together everything from the first three classes. You will read and trace short programs, plan a program in pseudocode before typing it, write four small Python programs, and fix a program that "looks right" but is not. After finishing it you should be able to:

- predict the value *and the type* of an expression before running it;
- read text with `input()` and convert it with `int()` / `float()` on purpose;
- use `/`, `//`, `%`, and `**` correctly;
- produce clean output with f-strings, `:.2f`, and alignment;
- choose descriptive variable names and document your code;
- find and explain type bugs in code someone (or something) else wrote.

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 1 - <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 (`total_seconds`, not `x`). Write fixed values that never change as UPPER_CASE constants (`TAX_RATE = 0.06`).
3. Convert `input()` text with `int()` or `float()` before doing any math.
4. Match the output format shown in the sample runs. Spacing and rounding are part of the grade.
5. 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 code (20 points)

Answer Part A in a plain-text file named `partA.txt`. Do **not** run the code first. Work it out 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. Predict the value — **10 points**

For each expression, write the value Python prints *and* its type (`int`, `float`, `str`, or `bool`). One point each.

```
(a) 17 // 5
(b) 17 / 5
(c) 17 % 5
(d) 2 ** 5
(e) 5 + 5.0
(f) "5" + "5"
(g) "ab" * 3
(h) len("CSC 141")
(i) int(3.99)
(j) round(3.14159, 3)
```

A2. Trace the program — 5 points

Write exactly what this program prints, one line per `print()`.

```
a = 4
b = 10
a = a + b          # line 3
b -= 3             # line 4
a, b = b, a        # line 5
print(a, b)
print(a * 2, b // 3, sep="-")
print("a is", a, end="!")
print("done")
```

A3. Explain the error — 5 points

A student wrote the program below. When they type `19` at the prompt, Python stops with an error.

```
age = input("How old are you? ")
print("Next year you will be", age + 1)
```

(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 `age`.
(c) Show the corrected line(s) of code.

4 Part B: Plan before you type (10 points)

Before writing `bill_split.py` (Program 2 below), write its plan in **pseudocode** as a comment block right under the header comment in that file. Pseudocode has no syntax rules. Plain English steps in order are exactly what we want, as in the class example:

```
# PLAN (pseudocode):
#   1. ask for the bill amount (a number with decimals)
#   2. ask for the tip percent
#   3. ask for the number of people (a whole number)
#   4. tip = bill x percent / 100
#   ...
```

You are graded on whether the plan is complete (every input, every computation, every output) and in a sensible order, not on how it is worded. Write the plan *first*; then turn each step into code.

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: `time_convert.py` — 15 points

Ask the user for a number of seconds (a whole number). Convert it to hours, minutes, and seconds and print the breakdown. Use `//` and `%`; do not use `if` statements or loops (we have not covered them yet).

```
Enter a number of seconds: 3725
3725 seconds = 1 hour(s), 2 minute(s), 5 second(s)
That is about 62.08 minutes in total.
```

```
Enter a number of seconds: 100
100 seconds = 0 hour(s), 1 minute(s), 40 second(s)
That is about 1.67 minutes in total.
```

Hints: there are 3600 seconds in an hour. After removing the hours, how many seconds are left? The last line uses true division and `:.2f`.

Program 2: `bill_split.py` (include your Part B pseudocode) — 20 points

Ask for the bill amount, the tip percentage, and the number of people sharing the bill. Compute the tip, the total, and each person's share, then print a receipt. Money is always printed with exactly two decimal places. Right-align the amounts so the decimal points line up (use a width such as `{total:>10.2f}`).

```
Bill amount ($): 86.40
Tip percent: 18
Number of people: 4

======== RECEIPT ========
Bill:        $     86.40
Tip (18.0%): $     15.55
Total:       $    101.95
------------------------
Per person:  $     25.49
========================
```

Requirements: the tip percent must appear inside the "Tip" label using an f-string; the lines of `=` and `-` must be built with string repetition (`"=" * 25`), not typed out by hand.

Program 3: `name_tag.py` **15 points**

Ask for the user's first name, last name, and birth year. Build the full name, print it inside a box of asterisks whose width matches the name, and print a few facts. Define a constant `CURRENT_YEAR = 2026` and compute the age from it. Assume the birthday has already happened this year.

```
First name: Ada
Last name: Lovelace
Birth year: 2007

****************
* Ada Lovelace *
****************
Name length:    12 characters
Age this year:  19
Days lived:     about 6935 days
Name repeated:  Ada Ada Ada
```

Requirements: the box width must come from `len(full_name)`, so it adapts to any name. "Days lived" is age × 365. The labels are aligned with `\t` or with an f-string width; either is fine, as long as the columns line up for the sample input.

Program 4: `free_fall.py` (a scientific computation) **10 points**

An object dropped from rest falls a height h (in meters) in time $t = \sqrt{2h/g}$ seconds, and hits the ground at speed $v = g\,t$ meters per second, where $g = 9.81$ m/s^2. Ask the user for the height, then print the fall time and the impact speed in both m/s and km/h (1 m/s = 3.6 km/h), each rounded to two decimals.

```
Drop height in meters: 45
Fall time:     3.03 s
Impact speed:  29.71 m/s  (106.97 km/h)
```

Hints: Python has no square-root operator, but `x ** 0.5` computes $\sqrt{x}$. Store `GRAVITY = 9.81` as a constant. Use parentheses; `2 * h / g ** 0.5` is *not* the same thing.

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

Someone asked an AI assistant for a program that computes the cost of buying several copies of an item, including 6% sales tax. The AI produced the code below. It looks reasonable, but it contains **three bugs**: two cause errors or wrong output because of *types*, and one is a *logic* mistake that produces a wildly wrong total without any error message.

Save this code as `debug_me.py`, trace it by hand for input `10` and `2` 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.

```
# Computes the cost of qty copies of an item, with 6% sales tax.
price = input("Unit price? ")
qty = int(input("How many? "))
subtotal = price * qty
tax = subtotal * 6
total = subtotal + tax
print("Total: " + total)
```

The corrected program must print exactly the following for a price of `10` and a quantity of `2`:

```
Unit price? 10
How many? 2
Total: $21.20
```

7 Grading

Component	Points	What earns full credit
Part A: reading and tracing	20	Correct values and types; clear error explanation; correct fix.
Part B: pseudocode plan	10	Every input, computation, and output listed, in a workable order.
Program 1: `time_convert.py`	15	Correct math with `//` and `%`; output matches the samples.
Program 2: `bill_split.py`	20	Correct amounts; two decimals; aligned columns; rules built with `*`.
Program 3: `name_tag.py`	15	Box width from `len()`; constant used; facts correct and aligned.
Program 4: `free_fall.py`	10	Correct formula and units; constant used; two-decimal output.
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, or unconverted `input()` values will lose points even when the output is correct.

8 What to submit

Upload a single ZIP file named `Lastname_Firstname_PA1.zip` to the **Programming Assignment 1** dropbox on D2L. It must contain exactly these six files:

`partA.txt`	Part A answers (plain text)
`time_convert.py`	Program 1
`bill_split.py`	Program 2, with the Part B pseudocode comment at the top
`name_tag.py`	Program 3
`free_fall.py`	Program 4
`debug_me.py`	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 input of your own for each program and the answer makes sense.
- ☐ `bill_split.py` contains your pseudocode; `debug_me.py` contains your bug list.
- ☐ The ZIP is named correctly and contains only the six files above.

The week-1 bug, one more time

`input()` *always* returns a `str`. `"10" * 2` is `"1010"`, and `"Total: " + 21.2` is a `TypeError`. If your output looks strange, check the types first: `print(type(price))` is a perfectly good debugging tool.

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.