Class

2

CSC 141 Computer Science I

Computational Thinking & Expressions

Dr. Si Chen (schen@wcupa.edu)

Tue / Thu · 11:00 AM – 12:15 PM · Mitchell Hall 102
Fall 2026 · Week 1 · Thursday, August 27, 2026
Bring a laptop · wcu.ninja/csc141

Recap from Tuesday

- A program is **precise, ordered instructions**; Python runs them **line by line**, top to bottom.
- You wrote a program with `print()`, a **string** (`"..."`), and a **comment** (`#`).
- The core loop of programming: **edit** → **run** → **read the output/error** → **fix**.
- Run everything today with no setup: **wcu.ninja/csc141/python**.

West Chester University

PART 1

How to think like a programmer

Before syntax: the four moves that turn a hard problem into code

West Chester University

Computational thinking: four moves

Decomposition

Break a big problem into small, solvable pieces.

"Compute a grade" → read scores → average → letter.

Pattern finding

Spot repetition and similarity.

The same step for every student → a loop.

Abstraction

Keep what matters, ignore the rest.

A name is just text; an age is just a number.

Algorithm

A clear step-by-step recipe, in order.

Write it in plain words (pseudocode) before code.

From a problem to a plan (pseudocode first)

Pseudocode

- **Problem:** tell a user how many seconds are in their age.

- **Plan (pseudocode):**
- ask for age in years
- seconds = years × 365 × 24 × 60 × 60
- display the result

The point

- **Why write the plan first?**
- You separate *thinking* from *typing*.
- You can check the logic before fighting syntax.
- Same habit that lets you **direct and check AI** later — you own the plan.
- Pseudocode has no rules; clarity is the only goal.

PART 2

Talking to the console

print() in depth, and the escape sequences every programmer uses

West Chester University

print() — more than one way

```
print("Hi", "Ada", "!")      # Hi Ada !
(spaces between args)
print("A", "B", sep="-")     # A-B
(change the separator)
print("no newline", end=" ") # stays on
the same line
print("next")                # ... next

print()                      # a blank
line
print("2 + 3 =", 2 + 3)      # 2 + 3 = 5
 (text and a number)
```

- `print()` separates multiple arguments with a **space** by default.
- `sep=` changes what goes **between** arguments.
- `end=` changes what goes **at the end** (default is a newline).
- You can print **strings and numbers together** — Python inserts the spaces.

In Java this was System.out.print / println. In Python it is one function: print().

West Chester University

Escape sequences: printing the unprintable

The common ones

- `\n` — **newline** (go to the next line)
- `\t` — **tab** (align columns)
- `\\` — a literal **backslash**
- `\"` — a literal **double quote** inside a string
- `\'` — a literal **single quote**

What they do

```
print("Line one\nLine two")
  → Line one
  → Line two

print("Name:\tAda")
  → Name:    Ada

print("She said \"hi\"")
  → She said "hi"
```

PART 3

The raw materials: values, types, variables

What Python works with — and the mistakes everyone makes first

Values have types

```
age = 19          # int   — whole
numbers
gpa = 3.75        # float — numbers
with a decimal point
name = "Ada"      # str   — text, in
quotes
is_student = True # bool  — True or
False

print(type(age), type(gpa),
type(name))
# <class 'int'> <class 'float'> <class
'str'>
```

- Every value has a **type**: `int`, `float`, `str`, `bool`.
- A **variable** is a name that points to a value; `=` means **assign** (not "equals").
- Ask Python the type with `type(x)`.
- Types decide what operations make sense — text and numbers behave differently.

Unlike Java, you do not declare the type — Python figures it out from the value.

West Chester University

Variables & identifiers: naming rules

The rules

- A **variable name** (identifier) may use letters, digits, and `_`.
- It **cannot start with a digit**, and **cannot be a keyword** (`if`, `for`, `print`…).
- Names are **case-sensitive**: `age`, `Age`, and `AGE` are three different names.
- Convention: **lowercase_with_underscores** (snake_case).

Naming well

- **Good names are documentation:**
- `student_count` — obvious
- `price`, `total_seconds` — clear

- **Poor names hide meaning:**
- `x1`, `temp`, `data2` — avoid
- Future-you will thank present-you.

Expressions & operators

```
10 + 3      # 13    addition
10 - 3      # 7     subtraction
10 * 3      # 30    multiplication
10 / 3      # 3.333...  true division
(always a float)
10 // 3     # 3     floor division (drops
the remainder)
10 % 3      # 1     modulo (the remainder)
2 ** 10     # 1024  exponent

2 + 3 * 4       # 14, not 20  (* before +)
(2 + 3) * 4     # 20          (parentheses
win)
```

- An **expression** is anything that computes a value.
- `/` always gives a **float**; `//` floors; `%` gives the **remainder**.
- `%` is everywhere: even/odd (`n % 2`), last digit, wrapping.
- Order of operations follows math — **use parentheses** when unsure.

print() and input(), precisely — the #1 week-1 bug

```
# input() ALWAYS returns text (a str)
age = input("Age? ")        # age is "19",
NOT 19
print(age + 1)              # ERROR: can
only concatenate str to str

# convert the text to a number first:
age = int(input("Age? "))   # now age is
the integer 19
print(age + 1)              # 20

price = float(input("Price? "))   # for
decimals, use float()
```

- `input(prompt)` shows the prompt and returns whatever the user types — **always a `str`**.
- `"19" + 1` is a **TypeError** — you cannot add text and a number.
- Convert with `int(...)` or `float(...)` **before** doing math.
- This single trap causes most week-1 bugs. **Expect it, and read the error.**

TypeError: can only concatenate str (not "int") to str — learn to recognize this message.

West Chester University

PART 4

Your turn

Put the four moves and today's Python together — in the browser playground

Practice: a tiny receipt program

HANDS-ON ⏱ 8–10 min

```
item = input("Item? ")
price = float(input("Unit price? "))
qty = int(input("Quantity? "))

subtotal = price * qty
tax = subtotal * 0.06
total = subtotal + tax

print("Item:\t", item)
print("Total:\t", total)
# add subtotal and tax lines too
```

- Open **wcu.ninja/csc141/python**. Plan it first (pseudocode!), then code it.
  - Ask for an **item name** and a **unit price** (a number).
  - Ask for the **quantity** (a whole number).
  - Compute **subtotal = price × quantity**, then add **6% tax**.
  - Print the item, subtotal, tax, and total — use `\t` to line things up.
- **Watch out:** `input()` gives text — convert with `int()` / `float()` first.

Debrief — what you practiced

- **Decomposition:** you split "make a receipt" into read → compute → display.
- **Types matter:** price is `float`, quantity is `int`; `input()` gave you `str` to convert.
- **Expressions & escapes:** you combined `*`, `+`, a percentage, and `\t` formatting.
- You read at least one **error message** and fixed it — the real skill.
  - An AI could write these lines instantly — but now *you* can read, change, and trust them.

Reading code is the new superpower

```
# An AI wrote this. Will it print
"Total: 20"?
# Trace it BY HAND before you run it.
price = "10"        # note the quotes
quantity = 2
total = price * quantity
print("Total:", total)
```

- In the AI era, code is cheap to **generate** — the value is being able to **read and check** it.
- Trace it: what **type** is `price`? What does `str * int` do in Python?
- It prints **Total: 1010** — `"10" * 2` **repeats the string**, it does not multiply!
- The bug is a **type** mistake — exactly today's lesson. You caught what merely *looked* right.

The skill AI cannot replace: verifying that plausible-looking code is actually correct.

West Chester University

Before next class (Thu, Sep 3)

- Redo the receipt program **from scratch** — first in the playground, then in **VS Code**.
- Confirm **Python 3 + VS Code + the Python extension** work on your own laptop.
  - Our next class (**Thu, Sep 3**) covers variables, numeric & string types, and **f-strings**.
- Course site: **wcu.ninja/csc141**. **Programming Assignment 1** is assigned next class.

West Chester University