Class

3

# CSC 141 Computer Science I

## Variables, Types & Formatted Output

**Dr. Si Chen (schen@wcupa.edu)**

Tue / Thu · 11:00 AM – 12:15 PM · Mitchell Hall 102
Fall 2026 · Week 2 · Thursday, September 3, 2026
Programming Assignment 1 is out · wcu.ninja/csc141

# Recap from last class

- Every value has a **type**: `int`, `float`, `str`, `bool`; `type(x)` tells you which.
- `=` **assigns** a value to a name; expressions use `+ - * / // % **`.
- `input()` **always returns str** — convert with `int()` / `float()` before math.
- Reading & checking code is a skill: `"10" * 2` is `"1010"`, not `20`.

PART 1

# Variables & assignment

A variable is a named box in memory — and you can change what is in it

# Variables: name, assign, reassign

```
value = 5              # create a
variable and assign 5
print("The value is", value)   # The
value is 5

value = 12             # reassign — the
box now holds 12
value = value + 3      # read old value,
compute, store back
print(value)           # 15

# Python has NO type declarations —
the value sets the type
x = 7         # x is an int
x = "seven"   # now x is a str —
allowed in Python
```

- A **variable** is a **named storage location** in memory.
- `=` is **assignment**, right-to-left: compute the right side, store it in the name.
- You can **reassign** freely — and even change the **type** of what a name holds.
- Unlike Java (`int value;`), Python needs **no declaration** — the value decides the type.

*"value = value + 3" reads the old value first, then stores the new one.*

# Augmented assignment & swapping

```
score = 10
score += 5     # same as: score =
score + 5   -> 15
score -= 3     # 12
score *= 2     # 24
count = 0
count += 1     # the everyday "add
one" (we will loop with this)

# Python can swap two variables in one
line:
a, b = 1, 2
a, b = b, a    # now a is 2, b is 1
```

- `+=`, `-=`, `*=`, `/=` update a variable **in place** — very common.
- `count += 1` is the counting pattern behind loops (coming next week).
- Python's `a, b = b, a` **swaps** without a temporary variable — a nice touch.

# Identifiers: the rules & good style

## The rules

- A name may use **letters, digits, and _** —
- ...but **cannot start with a digit** and **cannot be a keyword** (`if`, `for`, `True`, `print`…).
- Names are **case-sensitive**: `rate`, `Rate`, `RATE` differ.
- No spaces: use `sales_tax_rate`, not `sales tax rate`.

## Naming well

- **Style (Python convention):**
- variables & functions → `snake_case`
- constants → `TAX_RATE = 0.06`

- **Descriptive names document your code:**
- `tr = 0.0725` ✗ vague
- `sales_tax_rate = 0.0725` ✓ clear

PART 2

# Types in depth

What Python's types can do — and one big difference from Java

# Python types vs. Java types

## Java

- Java has **8 fixed-size** primitive types:
- `byte`(1) `short`(2) `int`(4) `long`(8),
- `float` `double`, `boolean`, `char`.
- Each has **hard limits** (e.g. int maxes at ~2.1 billion).
- You must **declare** the type up front.

## Python

- Python keeps it simpler:
- `int` — whole numbers, **no size limit** (`2 ** 200` just works!).
- `float` — decimals; `str` — text; `bool` — `True`/`False`.
- No declarations; the **value** sets the type.
- Convert on purpose: `int()`, `float()`, `str()`.

# Numbers, division, and conversion

```
7 / 2      # 3.5    true division — ALWAYS
a float
7 // 2     # 3      floor division — drops
the remainder
7 % 2      # 1      modulo — the remainder
round(3.14159, 2)   # 3.14   round to 2
decimals

int("42")    # 42     text -> int
float("3.5") # 3.5    text -> float
str(42)      # "42"   number -> text
int(3.9)     # 3      float -> int
(truncates, no rounding!)
```

- `/` is float division; `//` floors; `%` gives the remainder (even/odd, last digit).
- `round(x, n)` rounds; `int(3.9)` **truncates to 3** — it does not round.
- `int()`, `float()`, `str()` convert **on purpose** — you control the type.
- Money tip: floats are approximate; `0.1 + 0.2` is `0.30000000000000004`.

# The + and * operators on strings

```
"Hello, " + "Ada"     # "Hello, Ada"   +
joins strings (concatenation)
"ab" * 3              # "ababab"        *
repeats a string
len("Ada")            # 3
number of characters

# + needs matching types:
"Score: " + 95        # ERROR: str + int
"Score: " + str(95)   # "Score: 95"
convert first
# ...or use an f-string (next slide) —
much cleaner
```

- `+` **concatenates** strings; on numbers it **adds** — same symbol, two jobs (from the historical "+ operator" slide).
- `*` **repeats** a string; `len(s)` counts its characters.
- You cannot `+` a `str` and an `int` — convert with `str()`, or use an f-string.

PART 3

# Formatted output with f-strings

The clean, modern way to build text — you will use this everywhere

# f-strings: put values right inside the text

```
name = "Ada"
age = 19
gpa = 3.756

# start the string with f, put variables
in { }
print(f"Hi {name}, you are {age}.")     #
Hi Ada, you are 19.
print(f"Next year: {age + 1}")          #
expressions work too: 20
print(f"GPA: {gpa:.2f}")                #
3.76  (:.2f = 2 decimals)
print(f"{name:>10}")                    #
right-align in 10 spaces
```

- Prefix a string with `f` and drop variables (or expressions) inside `{ }`.
- No more `str()` conversions or `+` juggling — much easier to read.
- `{value:.2f}` formats to **2 decimal places**; `{x:>10}` aligns in columns.
- This replaces the old escape-sequence tab tricks for lining things up.

*f-strings are the modern replacement for clumsy + concatenation and \t alignment.*

PART 4

# Your turn

Variables, types, and f-strings together — in the browser playground

# Practice: a temperature converter

HANDS-ON ⏱ 8–10 min

```
f = float(input("Temp in F? "))

celsius = (f - 32) * 5 / 9

# f-string with 1 decimal place:
print(f"{f}F = {celsius:.1f}C")

# your turn: add Kelvin
```

- Open **wcu.ninja/csc141/python**. Plan it first, then code it.
  - Ask the user for a temperature in **Fahrenheit** (a number).
  - Convert to **Celsius**: C = (F − 32) × 5 / 9.
  - Print the result with an **f-string, rounded to 1 decimal**.
  - Use a **well-named variable** for each value.
- **Stretch:** also print Kelvin (K = C + 273.15).

# Debrief — what you practiced

- You used **well-named variables** to hold each value (readable, maintainable).
- You mixed **int/float** and converted `input()` text on purpose.
- You produced **clean, formatted output** with an f-string and `:.1f`.
  - This is exactly the toolkit **Programming Assignment 1** needs — start it early.

# Programming Assignment 1 + before Tuesday

- **Programming Assignment 1 is assigned today** (on D2L / the course site) — it uses variables, types, I/O, and f-strings.
- Redo the temperature converter **from scratch** in VS Code, not just the playground.
- Read the Week 3 preview: **Boolean expressions and if / elif / else** (making decisions).
- Course site: **wcu.ninja/csc141**. Bring your laptop Tuesday.