



Middlebury

CSCI 146: Intensive Introduction to Computing

Fall 2025

Lecture 2: Expressions, variables, functions

Goals for today

- Evaluate arithmetic expressions over integers and floats.
- Describe the concept of "type" and *operations* between types.
- Describe the purpose of a variable and perform variable assignment.
- Given a function definition, identify the name, function parameters and body.
- Distinguish between global variables, function parameters and local variables.
- Determine the return value of a function.
- Use built-in functions such as `print`, `str`, `int`.
- Implement simple functions.

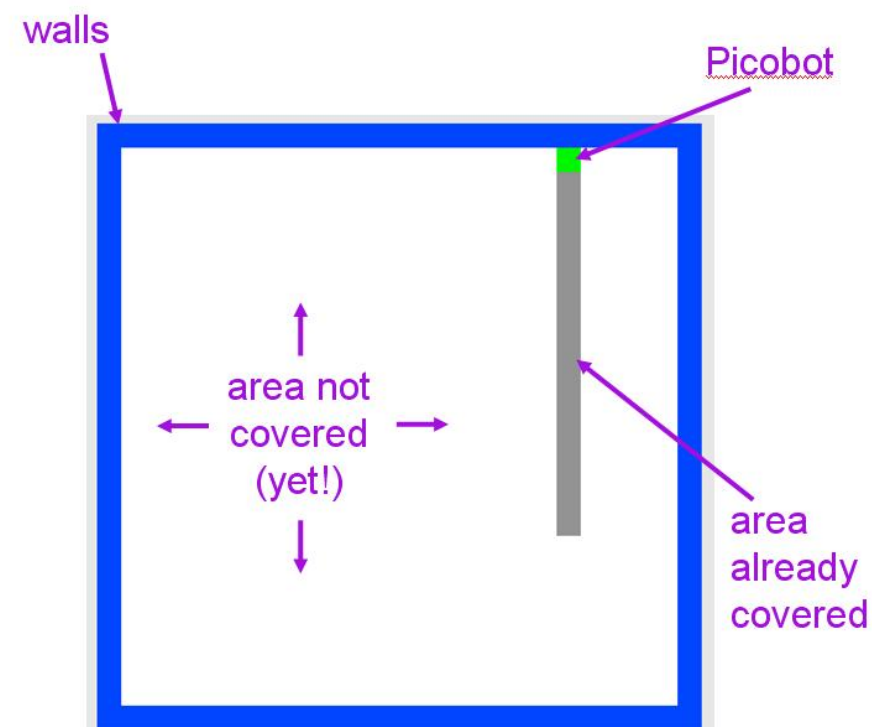
How did installing `Python` and VS Code go?

 or  on our **Reactions** page.

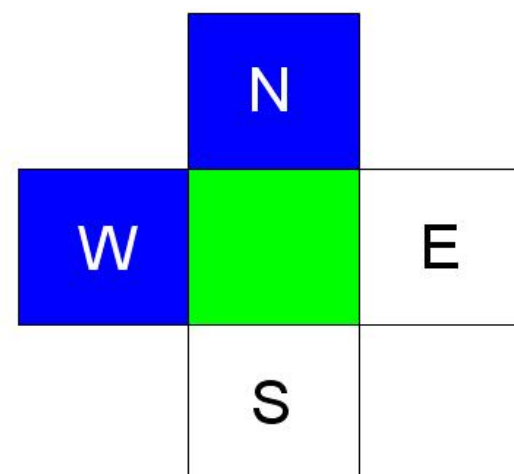
Recall our **Picobot** problem: input, outputs (goals), algorithm and implementation.

STATE SURROUNDINGS \rightarrow MoveDirection NewState

```
1 # "If Picobot is in state 0 and senses xxxS (only a wall to the south)
2 # it should move north and stay in state 0"
3 0   xxxS   ->   N   0
4
5 # "If Picobot is in state 0 and senses nothing to the north and
6 # anything (* wildcard) to the east, west, and south (wall or not)
7 # it should move north and stay in state 0"
8 0   x***   ->   N   0
```



Goal: whole-environment coverage
with only *local sensing*...




Here, picobot's surroundings are

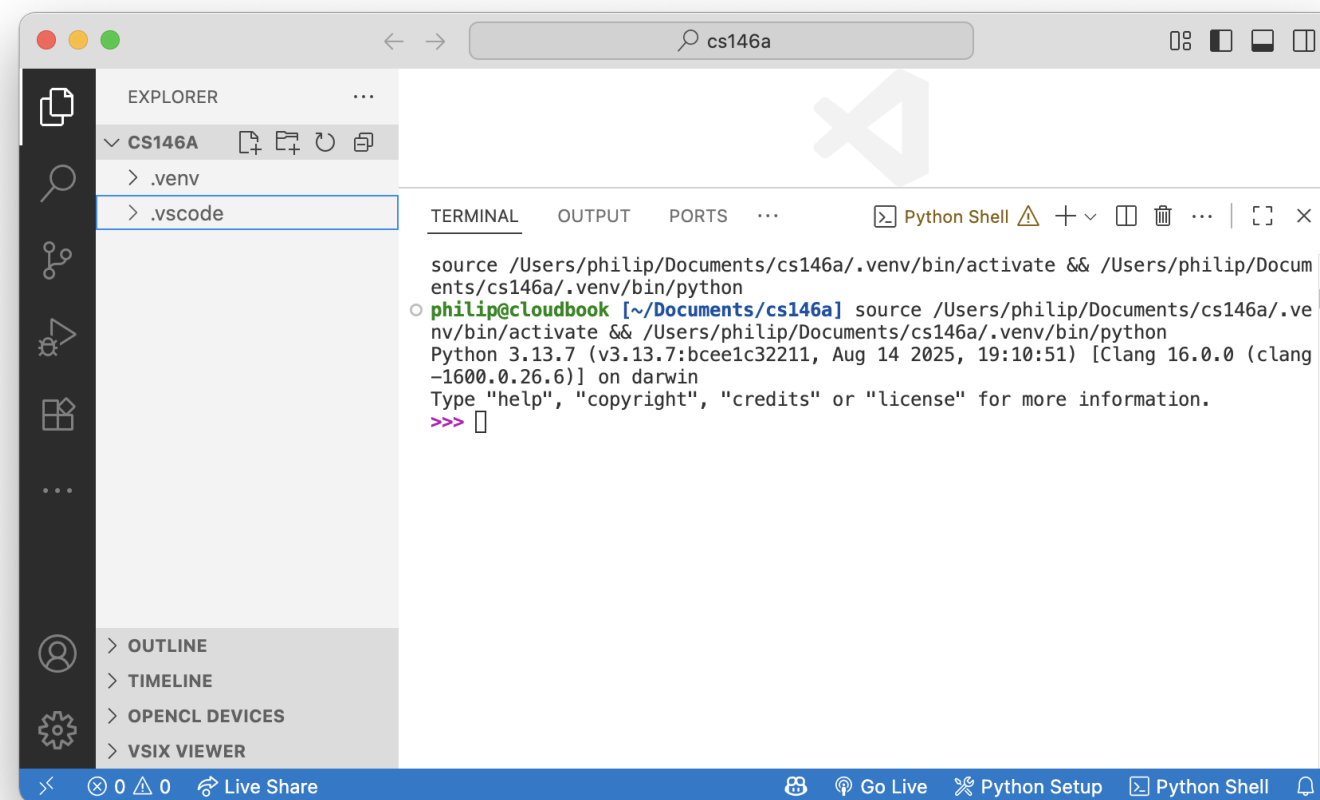
NxWx

↑ ↑ ↑ ↑
N E W S

Surroundings are
always in NEWS order.

We'll be using **Python** to solve problems this semester.

- Assuming you already downloaded the **cs146** folder, and also downloaded + installed **Python** (3.13) and **VS Code** (if not, please follow the links in the notes from last class).
- Open **VS Code** and then **File -> Open Folder** and open your **cs146** folder.
- Install the **middpy** extension (if not done already) or update the extension (I made changes after Monday, sorry! Make sure you update to version 1.0.4).
- Click **Setup Python** (if not done already).
- To open the shell:
 - If you have a file open: click the button at the top-right that looks like a terminal .
 - If you don't have a file open: click **Python Shell** in the bottom-right.
- A terminal should open and you should see **>>>** which is "prompting" us for input.



Question 1: Which of the following gives a result of **16** in **Python**?

A. `8 // 2 * 4`

B. `8 / (2 * 2 + 2)`

C. `8 // (2 * 4)`

D. None of the above.

Variables allow us to save intermediate results and use them again later.

Syntax of assignment: `<variable> = <expression>`

```
1 >>> x = 7 + 2
```

1. First, right-hand-side (RHS) is evaluated.
2. The resulting value is then stored in memory.
3. The variable name on the left-hand-side (LHS) points to the memory location with the result.
4. But the assignment *statement* `x = 7 + 2` does not itself evaluate to anything which is why nothing is printed (we would need to `print(x)` to see what value `x` stores).

Variable names can start with any LETTER (`a-z`, `A-Z`), or UNDERSCORE (`_`), and can *contain* digits (`0-9`) but cannot start with a digit.

Question 2: At the end of this code, what value is assigned to **y**?

```
1 y = 3 + 2 + 1 6
2 y = y + 2 8
```

A. 2

B. 6

→ C. 8

D. This code will cause an error.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Question 3: At the end of this code, what will the memory look like?

```
1 y = 3
2 x = 7
3 y = 3 + 2 + 1
```

A.

y: 6

B.

y: 3

x: 7

y: 6

C.

y: 6

x: 7

D.

y: 3

x: 7

How much money in...

- 1 quarter, 2 dimes?
- 5 quarters, 10 dimes?
- 7 quarters, 3 dimes, 2 nickels?
- 9 nickels, 1 quarter, 1 toonie?



This is kind of annoying to do repeatedly. There must be a better way.

Functions!

First, let's talk about built-in functions.

Things to consider:

- Arguments (expressions within parentheses) are evaluated from left to right.
- Expression results are "bound" to what the function will use as function *parameters*.
- Expressions and statements in the body of the function are then executed.
- If necessary, a value may be *returned*.

```
1 >>> print("hello cs 146!")
2 "hello cs 146!"
3
4 >>> y = int("3")
5
6 >>> z = str(4)
```


A critical component to programming and abstraction is writing our own functions.

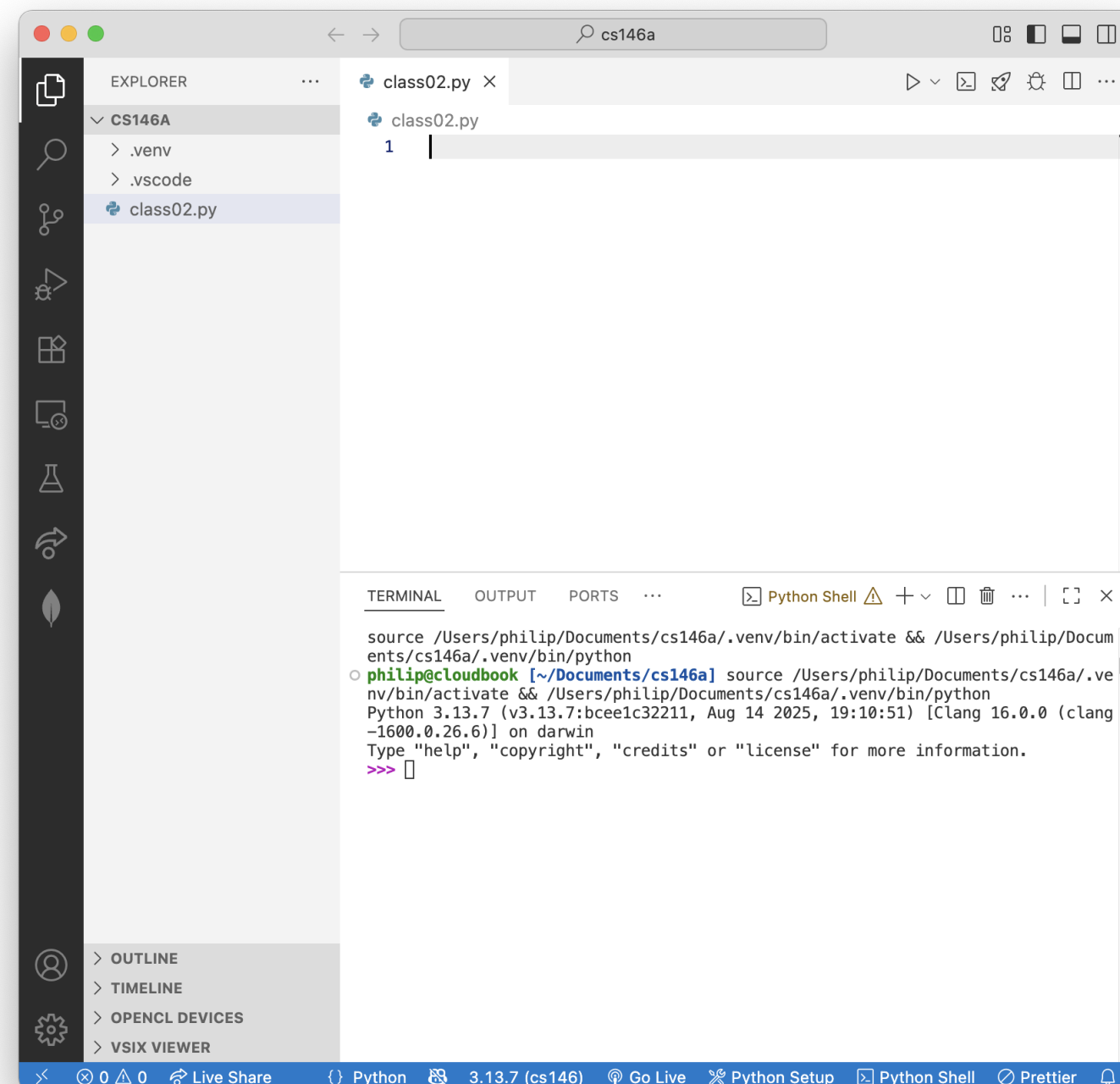
Question 4: What would this code print?

```
1 x = 6
2 print(4x)
```

- A. 6
- B. 24
- C. 4x
- D. Nothing, this code produces a syntax error

Working in the shell is good for short expressions or to try things out. Really we want to write programs in *scripts*.

- Click **File** -> **New File**, select **Python File**, then save it (**File** -> **Save**), name it **class02.py**
- Be careful that this is saved directly in **cs146** and not in **cs146/.venv** or **cs146/.vscode**.
- When a file is open, you should see some buttons at the top-right (terminal, rocket and bug).
- Let's investigate! **Note:** Use the  button to "run" your script.



Anatomy of a function through an example: computing $f(x) = x^2$.

```
1 def f(x):  
2     # note we can also use x ** 2, using the ** (exponentiation) operator  
3     return x * x
```

- `def` is a keyword and has a special meaning in Python: you can't name a variable `def`.
- Everything in the parentheses are the parameters, e.g. `x`. There can be zero or more parameters and are only defined within the function body/scope.
- The body of the function is *indented* with respect to the definition.
- The `return` keyword will terminate the function at that execution point and provide the result to the "caller". If there is no `return`, then the function returns `None`.

Execute your `class02.py` script with the  button and let's *call* our function.

Question 5: Which of the following does not contain a function call?

A. `type(4.5)`

B. `area(2, 9)`

C.

```
1 def add_one(x):  
2     return x + 1
```

D. `print("hello")`

E. None of the above, all contain function calls.

Question 6: What does the call to **calculate** return?

```
1 w = 0
2
3 def calculate(w, x, y):
4     a = x
5     b = w + 1
6     return a + b + 3
7 calculate(3, 2, 0)
```

Handwritten annotations: Above line 3, the values 3, 2, and 0 are written above the parameters w, x, and y respectively. Above line 4, the value 2 is written above x. Above line 5, the value 4 is written above the expression w + 1. Above line 6, the value 9 is written above the expression a + b + 3.

- A. 3
- B. 5
- C. 6
- D. 9

Example: writing a function to calculate the value of an investment.

$$f(P, r, t) = P(1 + r)^t$$

- P is the principal, r is the rate and t is time in years.
- Advice for writing functions:
 - Figure out what you want this function to do.
 - Define a *contract* for what the function expects and what will be returned (if any).
 - Write out the header `def compound_interest(principal, rate, years):`
 - Implement the body of the function.

```
1 # Calculates investment value with interest compounded yearly
2 def compound_interest(principal, rate, years):
3     amount = principal * ((1 + rate) ** years)
4     return amount
5
6 value = compound_interest(100, 0.03, 2)
7 print("Value = " + str(value))
```


Your turn! Work with someone to write a function to calculate the cost of an In-N-Out burger from the "special menu."

Goal: calculate the price of a 100x100 and a 100x50 In-N-Out burger.

Assume a single hamburger (bun + patty) costs \$1.50, and a cheeseburger (bun + patty + cheese) costs \$1.75 and a double-double costs \$2.65.

```
1 def innout_price(hamburger, cheeseburger, doubledouble, patties, cheeses):
2     """Compute the price of an In-n-Out burger with arbitrary numbers of patties and slices of cheeses
3
4     Args:
5         hamburger: Price of a hamburger
6         cheeseburger: Price of a cheeseburger
7         doubledouble: Price of a Double-Double
8         patties: Number of patties (must be at least 1)
9         cheeses: Number of cheese slices (can be zero or more)
10
11     Returns:
12         Price of burger
13     """
14     patty_and_cheese = doubledouble - cheeseburger
15     cheese = cheeseburger - hamburger
16     patty = patty_and_cheese - cheese
17     return hamburger + (patties - 1) * patty + cheeses * cheese
```

Summary

- The shell is good to test a few small expressions, *scripting* is better for complete programs.
- Be careful with types: we saw `int`, `float` and `str` types.
- Functions are useful for writing code blocks we will re-use with different input values.
- Be careful with the *scope* of the variables in a function (local versus global variables).
- First quiz on Friday (about 15 minutes at the beginning of class); notes (cheat sheet) linked on calendar.
 - I would recommend attempting the PrairieLearn practice problems.
- Please complete the Introduction Survey linked on the calendar.
- Come to office hours! Monday 10 - 11am (my office, room 219) and Thursdays 2:30 - 4:30pm (room 224).

Come to the *What I did this summer* CS seminar on Friday at 12:30pm (with pizza)!