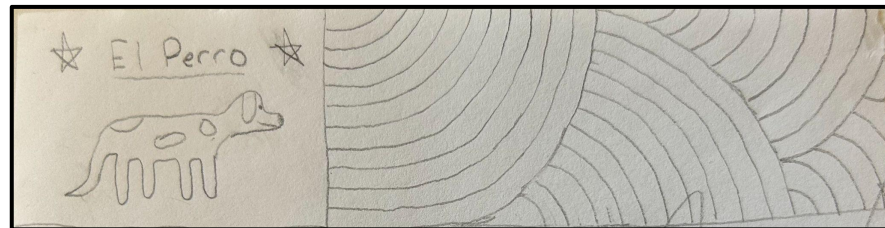


CS 1: Intro to CS



Wrapping up strings, introducing scope and user-defined functions

What's a ghost's favorite data type?



A "Boo"-lean!



Announcements

HW1 is due next Friday 11:30PM on CodePost (refer to Syllabus for submission/rework policies)

Python So Far

Writing and running Python programs

Our first `print("Hello world!")`

Expressions and arithmetic

Different Python types (ints, floats, strings, booleans)

Variables and assignment

Identifying functions "in the real world"

Built-in (`print`, `int`, `str`, `min`, `max`, etc.)

L2 Exit Ticket Questions Were Great!

I will use some of the student questions this term (keep 'em coming!) and will also sprinkle in answers to questions students asked me in lectures/Discord.

Today's review questions:

1. What are 4 types in Python you learned from lecture and this week's readings?
2. What is the correct variable type for the question "what is the chance of it being cloudy today?"

L2: What is One Thing You Learned?

I learned how to add numbers and assign the result to a variable in Python.

I learned what why the += operator works the way it does.

The tree visualization was helpful!

I learned what an interpreter is

I learned about how to do the expressions and also got to familiarize myself with a bunch of new syntax!

I got more comfortable with the terminal and the difference between being in bash (\$ or % prompt) where we can use ls (list), cd (change directory), and python3 (start Python interpreter, or execute a Python file in the current directory) vs. being in the >>> (wakka) Python interpreter

A Few L2 Questions for EL

How should the [homework template look like](#)?

How many libraries does Python have?

If we have $x = y$, does this mean that x and y are interchangeable? For instance, if $x = 5$, would $y = 5$?

What is the purpose of having a bunch of different programming languages instead of just one big one? How are these decisions made?

How do you go back to editing source code after using the terminal?

What is the syntax for writing our own functions?

- *We will learn this today!*

L2: What is One Thing You Learned?

I learned that in a case such "x = 45, y = 45, z = x + y," it will evaluate z = x + y from the right-hand side, replacing x + y with 90. Thus, z is now stored as 90, which means that it will not update to a new value if x and/or y are changed later on.

```
# 1. Assign a new variable x to be 120
```

```
x = 120
```

```
# Remember that = assignment evaluates the RHS and then updates the LHS
```

```
# RHS Evaluation:
```

```
# 1. Look up the variable x to get its value: x -> 120
```

```
# 2. Evaluate addition (+) on x + 10 -> 120 + 10 -> 130
```

```
# LHS Update:
```

```
# 3. Re-assign x to 130
```

```
x = x + 10
```

Check Your Understanding: Lecture Check 1

Q2: Remember that if a variable is assigned to another, when the second updates the first **does not**

```
1  a = 12
2  b = 10
3  c = b # 10
4
5  a = a + 1 # 13
6  b = b ** 2 # 100
7  c = c + a # 10 + 13 = 23
```


Today's Learning Objectives

Learn more about data types (including booleans) and functions

Use the `input(...)` function to prompt user input and save as variables

Learn how to write your own reusable functions in Python

Start to better distinguish between **functions** and **methods** with a preview of object-oriented programming

Learn how to use the VSCode debugger to debug your programs and observe program execution/variable scope

Review: Types

Data in programming languages is subdivided into different "types":

- integers: `0 -43 1001`
- floating-point numbers: `3.1415 2.718`
- boolean values: `True False`
- strings: `'foobar' 'hello, world!'`
- and many others

Today, we'll go deeper into strings and how to use them

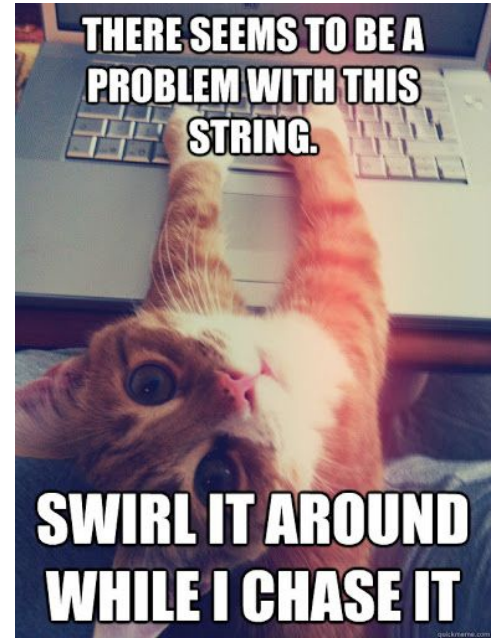
Strings

A sequence of characters

Enclosed in quotes (" " or ' ')

One of the most-used data types in programs, for example:

- DNA sequences: 'ACCTGGAACT'
- Documents in word processors
- Web pages and text from social media posts
- User interaction (string input/output)
- etc.



Sequence...

In Python, strings are just one example of *sequences*

Other kinds of sequences exist, which we'll cover later (e.g. lists, tuples)

Significance:

- The same functions and operators can be used with different sequences for the same kind of purpose
 - E.g., `len(seq)` returns the length of any sequence input

... of characters

In Python, there is no special data type for single characters (letters, digits, etc.)

- This is unlike some other programming languages such as Java

A character can only be represented as a string of length 1:

'a' # the character a (letter)

'1' # the character 1 (digit)

'_' # the underscore character

' ' # a space character

Quotation Marks

Like many other programming languages, Python allows you to use either single quotes (') or double quotes (") to represent strings.

```
'I am a string'
```

```
"So am I!"
```

However, you must be consistent:

```
"I am not a valid string'
```

It will happen...

If you leave out the quotation marks on a string, you'll get an error:

```
>>> 'foobar'
```

```
'Foobar'
```

```
>>> foobar
```

```
NameError: name 'foobar' is not defined
```

Python interprets foobar as a **variable**, not a string

String Concatenation

Python supports + for both ints (addition) and strings (concatenation), but you cannot mix operand types

We need to be careful when concatenating strings with other types such as numbers

```
>>> name = 'Bowie'
>>> age = 6
>>> print(name + ' is ' + age + ' years old.')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>>
```


String Concatenation

We can use `str()` to convert non-string types to strings for concatenation

```
[>>> name = 'Bowie'  
>>> age = 6  
>>> print(name + ' is ' + str(age) + ' years old.')  
Bowie is 6 years old.
```

We could also use the `format` method here to make things easier (coming up)

```
[>>> print('{} is {} years old.'.format(name, age))  
Bowie is 6 years old.
```

String Multiplication

We can also multiply strings with `*` to repeat them

```
>>> 'na' * 20 + '... batman!'
'nananananananananananananananananana... batman!'
>>> █
```

Check Your Understanding

Suppose we're trying to produce the following output:

```
Meet the 3 stooges!
```

```
Curly and Larry and Moe
```

What is wrong with the following code which attempts to produce this?

```
1 cat1 = Curly
2 cat2 = Larry
3 cat3 = Moe
4
5 print('Meet the 3 stooges!')
6 print(cat1 and cat2 and cat3)
```

VSCode Hints

VSCode will provide "hints" when you have a .py file open

These are very useful as you're learning programming and the rules of Python

Moving your cursor over the first yellow-underline gives a message that **Curly** is being referenced in the RHS of an assignment (expecting an expression) but there is no variable called **Curly**; don't forget to distinguish between variables (no quotes) and strings (quoted) in Python!

```
1  cat1 = Curly
2  cat2 = Larry
3  cat3 = Moe
4  print('Meet the 3 stooges!')
5  print(cat1 and cat2 and cat3)
```

```
1  cat1 = Curly
2  cat2 =
3  cat3 =
4  print('
5  print(c
6
```

Curly: Any
"Curly" is not defined Pylance([reportUn](#)
[View Problem](#) No quick fixes available

VSCode Hints

A caveat: not all hints will be useful, and not all bugs will be found for you in VSCode

In the example below, we've fixed the first lines, but there's still a bug!

```
precheck-1-q6.py U ×
lectures > lec03 > precheck-1-q6.py > ...
1  cat1 = 'Curly'
2  cat2 = 'Larry'
3  cat3 = 'Moe'
4  print('Meet the 3 stooges!')
5  print(cat1 and cat2 and cat3)
6
```

```
OUTPUT  TERMINAL  JUPYTER: VARIABLES  PROBLEMS
>  ▼  TERMINAL  Python Debug Co
mehovik@Els-MacBook-Pro:~/eipsum.github.io/cs1/lectures/lec03$ python3 precheck-1-q6.py
Meet the 3 stooges!
Moe
```

... ???

String Formatting (Reading 4)

Using comma-separated values with `print` is tedious and error-prone

Better approach is to use formatting with the string's built-in `format` method

Aside: Functions vs. Methods:

- We'll learn more about methods when we cover object-oriented programming, but the key difference is that methods work on objects whereas functions do not

The format method

```
[>>> name = 'Bowie'  
[>>> age = 6  
[>>> print(name + ' is ' + str(age) + ' years old.')  
Bowie is 6 years old.
```

```
[>>> print('{} is {} years old.'.format(name, age))  
Bowie is 6 years old.
```

The format method

You can use the `format` method (using `.` "dot" syntax) on a string:

- Use `{}` for placeholders in the string (one for each argument)
- Each argument in the `format` method call will replace the placeholder (in order)

```
>>> '{} is the instructor for {}!'.format('E1', 'CS1')
```

```
'E1 is the instructor for CS1!'
```

```
>>> salary = 18.5
```

```
>>> weekly_salary = salary * 20
```

```
>>> 'If your hourly salary is ${}, you earn ${} for working 20 hours a  
week.'.format(salary, weekly_salary)
```

```
'If your hourly salary is $30, you earn $370 for working 20 hours a week.'
```


More on Format Strings

Sometimes, we want to format numbers in a specific way. Here, `:d` formats integers, `:f` formats floats, and `:.2f` formats a float with exactly 2 digits following the decimal.

```
'an integer: {:d}'.format(42)
```

```
>>> 'an integer: 42'
```

```
'a float: {:f}'.format(42.123400)
```

```
>>> 'a float: 42.123400'
```

```
'a float: {:.2f}'.format(42.123400)
```

```
>>> 'a float: 42.12'
```

```
>>> 'If your hourly salary is ${:.2f}, you earn ${:.2f} for working 25 hours a  
week.'.format(salary, weekly_salary)
```

```
'If your hourly salary is $18.50, you earn $462.50 for working 25 hours a week.'
```

More on Format Strings

Using `.format()` in format strings can be tedious...

```
'x = {}, b = {}'.format(a, b)'
```

There is conveniently a shortcut:

```
f'a {a}, b = {b}'
```

The `f'...'` syntax indicates that it's a format string, and allows you to use variables in the format string without the `.format()` method call; practice using `f'...'` in HW 1!

You can also add modifiers (e.g. `{a:.2f}`) to substitute the numeric value of `a` to 2 decimal points

String Indexing

Can access parts of string sequences in various ways

```
[>>> name = 'Bowie'  
[>>> name[0]  
'B'  
[>>> name[-1]  
'e'  
[>>> name[len(name) - 1]  
'e'  
[>>> name[len(name)]  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
IndexError: string index out of range
```

Python uses “0-based” indexing, meaning the first character is at index 0, not 1

We’ll learn more about string-processing next week

Recall: Functions

A function takes some input data and transforms it into output data

Functions must be defined and then called with the appropriate arguments

A few functions are built-in to Python so we don't have to define them ourselves:

- `print(x)`
- `input(x)`
- `type(x)`
- `int(x)`, `float(x)`, `str(x)`
- `min(x, y, ...)`, `max(x, y, ...)`
- `help()`, `help(fn)`
- ...

The Built-in `input` Function

Recall that we introduced a few built-in **functions** last week

One common one is **`input`**, which:

- Takes an **argument** string to print a prompt to the user, pausing the program
- **Returns** the answer the user types after the prompt **as a string**

```
>>> name = input('What is your dog\'s name? ')
What is your dog's name? Lorem
>>> name
'Lorem'
>>> █
```

More about User Input

It's important to remember that the user's input is **always** returned as a string, even if the user provides a number

We may need to convert the string to the desired type (e.g. with `int(str)` for integers or `float(str)` for floats)

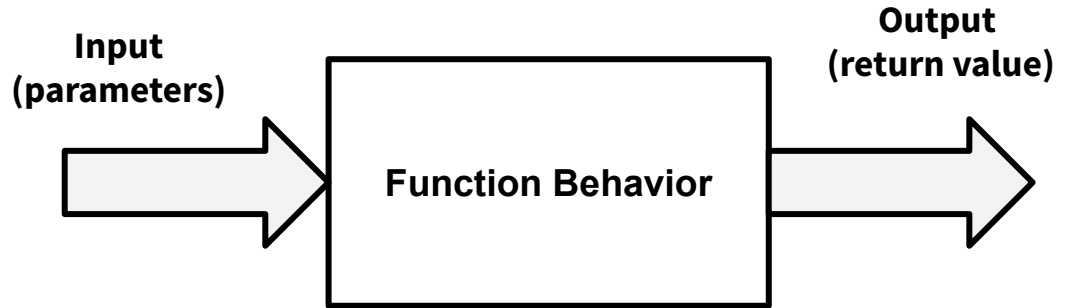
```
[>>> age = input('What is your dog\'s age? ')
What is your dog's age? 3
[>>> age_in_dog_years = age * 7
[>>> age_in_dog_years
'3333333'
[>>> age_in_dog_years = int(age) * 7
[>>> age_in_dog_years
21
>>> █
```

Anatomy of a Function

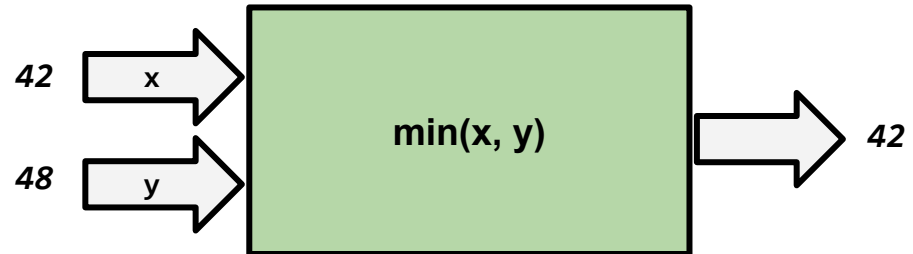
A function is like a machine to perform tasks and possibly return some result

Every function has:

- Behavior (body)
- Parameters (optional)
- Return value (optional)



Example with built-in `min` function:



Defining and Calling Functions

Functions may have parameters passed to help generalize functionality and may also specify a return value with the return keyword (**None** if no return specified)

Definition Syntax:

```
def name(<parameters>):  
    <body>  
    return <value> # optional
```

Definition Examples:

```
def say_hello(name):  
    print('Hello ' + name + '!')
```

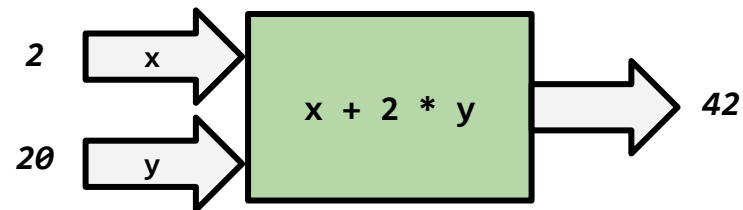
```
def f(x, y):  
    return x + 2 * y
```

Function Call Examples:

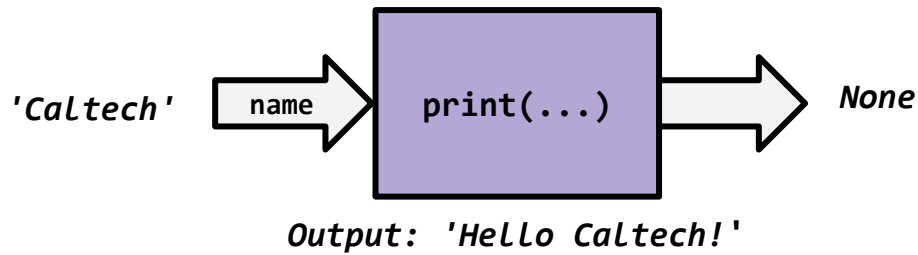
```
say_hello('world')    # Hello world!  
say_hello('Caltech')  # Hello Caltech!  
ans = f(2, 20)        # ans == 42
```


Functions as Machines

```
1 # Defining the function
2 def f(x, y):
3     return x + 2 * y
4
5 # Calling the function
6 ans = f(2, 20)
```



```
1 # Defining the function
2 def say_hello(name):
3     print('Hello', name, '!')
4
5 # Calling the function
6 say_hello('Caltech')
```



Scope is Important!

So far, our variables have been defined top-down - later assignments will **shadow** earlier ones.

Functions introduce their own **local** scope - **variables inside functions only exist in during the lifetime of a function call.**

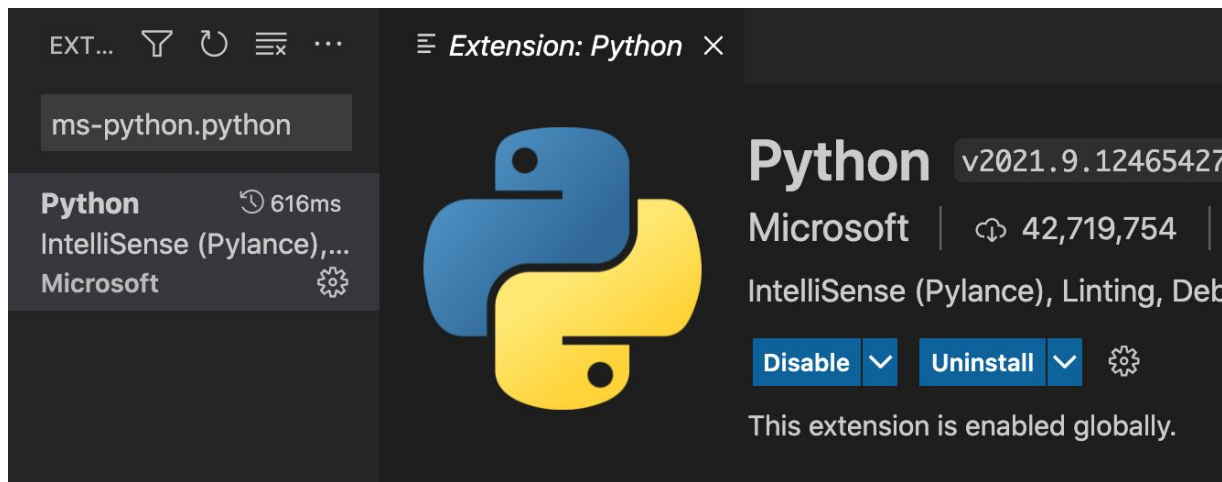
Local Variables

```
1 def f(x, y):  
2     z = 2 * y # z is a local variable  
3     return z + x  
4  
5 ans1 = f(2, 20) # 42  
6 ans2 = f(x, 20) # error! x is not in scope here
```

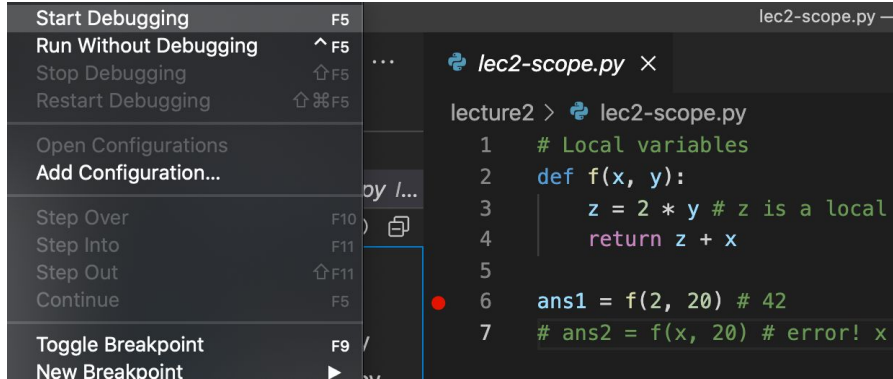
Note: ms-python VSCode Extension

In the VSCode setup guide, there is a step to install the ms-python extension:

- “Open the extensions view (Ctrl/Cmd+Shift+X), and install the [ms-python.python](https://marketplace.visualstudio.com/items?itemName=ms-python.python) extension.”

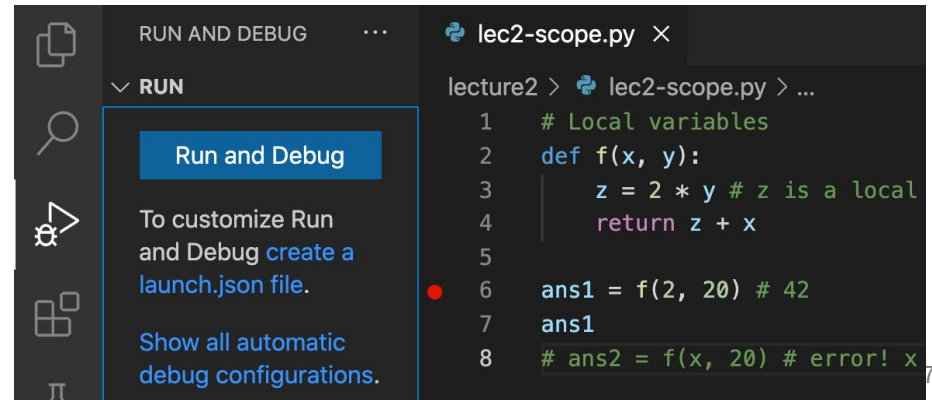


Starting a Debugging Session



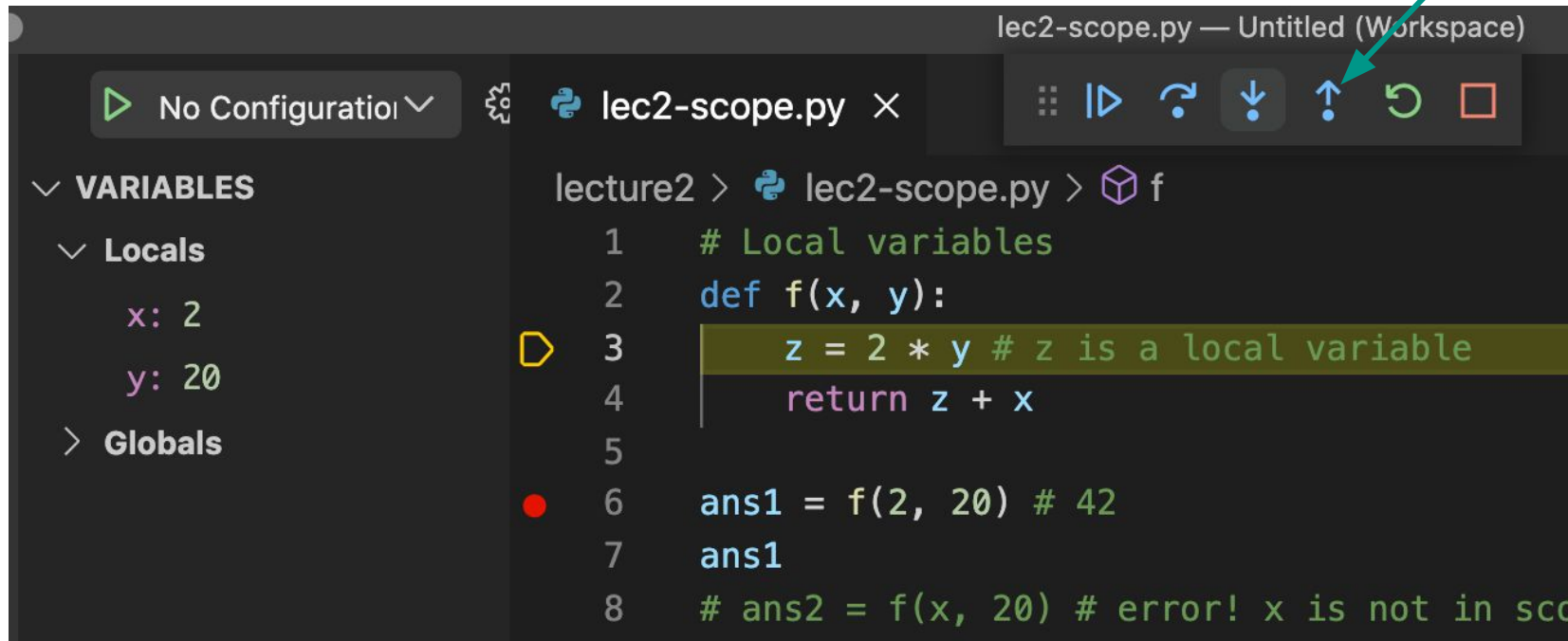
Option 1: Run > Start Debugging

Option 2: Click the Debug icon on the left pan of VSCode and click “Run and Debug”



Stepping into a Function

“Step Into”



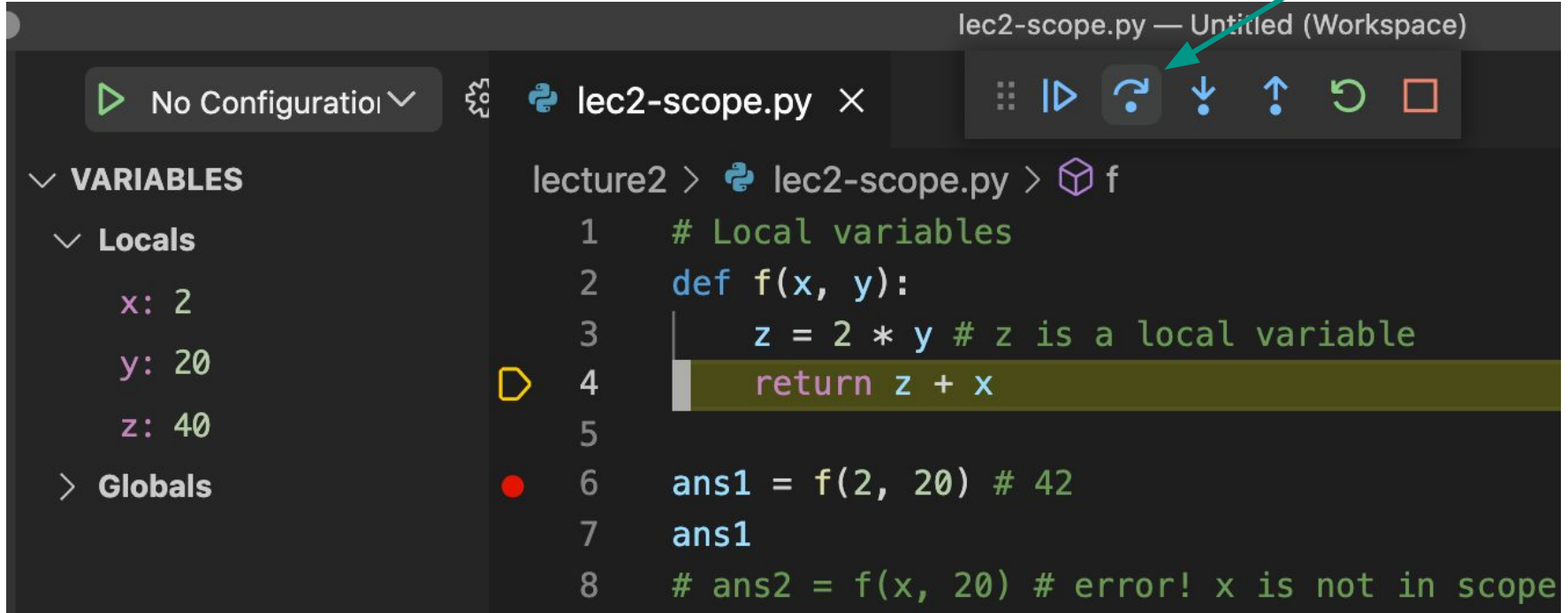
The screenshot shows a Python IDE interface. On the left, a sidebar displays the 'VARIABLES' panel with 'Locals' containing `x: 2` and `y: 20`, and 'Globals' expanded. The main editor shows a Python file named `lec2-scope.py` with the following code:

```
1 # Local variables
2 def f(x, y):
3     z = 2 * y # z is a local variable
4     return z + x
5
6 ans1 = f(2, 20) # 42
7 ans1
8 # ans2 = f(x, 20) # error! x is not in scope
```

A yellow arrow points to line 3, and a red dot is on line 6. The top toolbar includes a 'Step Into' icon (a blue arrow pointing up) which is highlighted by a green arrow from the text 'Step Into' above it.

Stepping Over to Next Statement

“Step Over”



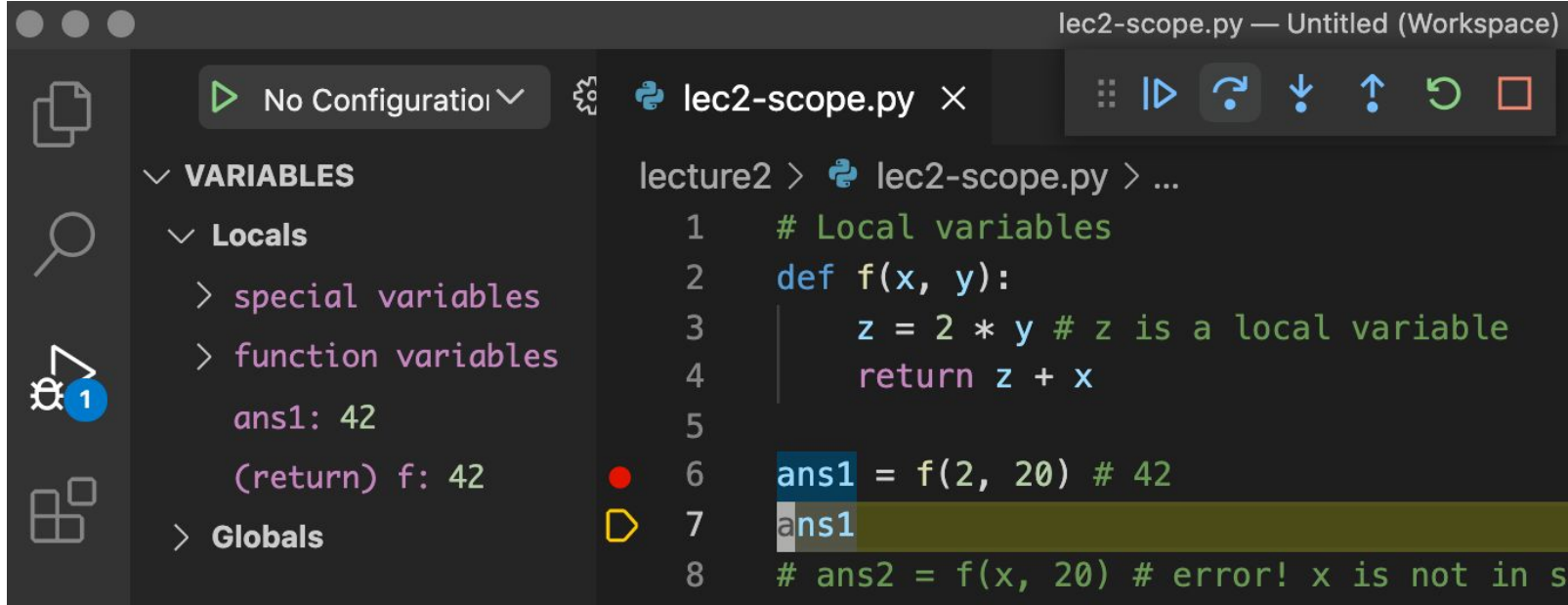
The screenshot shows a Python IDE with a workspace titled "lec2-scope.py — Untitled (Workspace)". The code editor displays the following Python code:

```
lecture2 > lec2-scope.py > f
1 # Local variables
2 def f(x, y):
3     z = 2 * y # z is a local variable
4     return z + x
5
6 ans1 = f(2, 20) # 42
7 ans1
8 # ans2 = f(x, 20) # error! x is not in scope
```

The IDE interface includes a toolbar with a "Step Over" button (a blue arrow pointing down) and a "Variables" panel on the left showing local variables:

- VARIABLES**
- Locals**
 - x: 2
 - y: 20
 - z: 40
- Globals**

Final Statement After Function return



The screenshot shows a Python IDE window titled "lec2-scope.py — Untitled (Workspace)". The interface includes a toolbar with a play button, a search icon, and a variable explorer icon. The variable explorer on the left shows a tree structure under "VARIABLES":

- Variables
- Locals
 - special variables
 - function variables
 - ans1: 42
 - (return) f: 42
- Globals

The main editor displays the following code:

```
lecture2 > lec2-scope.py > ...
1 # Local variables
2 def f(x, y):
3     z = 2 * y # z is a local variable
4     return z + x
5
6 ans1 = f(2, 20) # 42
7 ans1
8 # ans2 = f(x, 20) # error! x is not in s
```

The code is executed, and the output shows the state of the variable `ans1` in the global scope. The variable `ans1` is highlighted in blue, and the value `42` is shown next to it. The code on line 8 is commented out, indicating an error that would occur if it were executed.

Note: Line 7 was added to show the scope of `ans1` (and absence of `x`, `y`, and `z`) before the program exits

Continued Next Time

More on strings

- String "methods"
- Indexing
- Formatting strings

Deeper into program execution and function scope

(Preview is provided in the following slides)

Parameters vs. Arguments

Formal parameters are simply names for the argument values passed in a function call. The **position of arguments** will determine what formal parameter name they are assigned.

They have no relationship to other variable names in the program and will override other variables if there is a naming conflict.

```
1 def f(x, y):
2     z = 2 * y # here, z is a local variable!
3     return z + x
4
5 a = 2
6 b = 20
7 ans1 = f(a, b)
8 ans2 = f(b, a) # b and a are mapped to x and y in f, respectively
```

Practice: Variables and Scoping

What is the result of executing the following program?

```
1 x = 1
2 y = 2
3 z = 3
4
5 def square(x):
6     return x * x
7
8 def mystery(x, y, z):
9     print('x: ', x, 'y: ', y, 'z: ', z)
10
11 mystery(x, y, z)
12 mystery(x + y, x, square(y))
```

Practice: Variables and Scoping

What is the result of executing the following program?

```
1 x = 1
2 y = 2
3 z = 3
4
5 def square(x):
6     return x * x
7
8 def mystery(x, y, z):
9     print('x: ', x, 'y: ', y, 'z: ', z)
10
11 mystery(x, y, z)
12 mystery(x + y, x, square(y))
```

Output:

x: 1 y: 2 z: 3

x: 3 y: 1 z: 4

Practice: String Functions

Suppose Caltech usernames were automatically generated in the format of first initial followed by full last name (making an unrealistic assumption that everyone has a single first and last name and there are no duplicates). For example, “Lorem Hovik” would generate “lhovik@caltech.edu).

Write a Python function which takes a Caltech username as an argument and returns a valid email address in the format of <username>@caltech.edu.

Hint: You can lowercase a string with the `str.lower()` method (e.g. `'ABC'.lower()`).

Strings are Objects

Python is what's called an **object-oriented** language (we'll learn more about what this means in upcoming lectures)

Most data types are represented as "objects"

An "object" is some **data** with associated **methods** (similar to functions) that work on that data

Python strings are an example of an object

Functions vs. Methods

Functions that are associated with an object are called **methods**

Methods are called on an object using what's called "dot-syntax"

```
>>> 'hello world'.upper()  
'HELLO WORLD'  
>>>
```

Compare this with the **function print**, which takes values (including objects) as arguments:

```
>>> print('hello world')  
hello world  
>>>
```

Check Your Understanding

Assume the string variable `s` is defined. Which of the following are function calls?

`len(s)`

`s.upper()`

`s.lower()`

`help(str)`

`print(s)`

`input(s)`

`"hello {}".format(s)`

Check Your Understanding

Assume the string variable `s` is defined. Which of the following are function calls?

`len(s)`

`s.upper()` # method

`s.lower()` # method

`help(str)`

`print(s)`

`input(s)`

`"hello {}".format(s)` # method

Function and String Practice

The last exercise in HW1 is to write a function called `GC_content` which returns the percentage (between 0 and 1) of characters in a given string that are "G" or "C"

Let's practice a related function called `vowel_count` which:

- Takes a string as a single argument
- Returns the number of vowels ("a", "e", "i", "o", or "u") in that string
- What if we want to make it case-insensitive (i.e. "A" is treated as "a")?

More Practice (On Your Own)

[Lecture Check Q5:](#) (per syllabus, these are optional exercises to practice important concepts)

Recall the syntax for defining a function:

```
def <function_name>(<args>):  
    <body>  
    return <optional_return> # if no return, omit this line
```

For example, we can write a function to return the double of a number:

```
def double(x):  
    return x + x
```

Write a function `capitalize_all_words` that takes 3 strings as arguments and returns a space-separated string with all 3 strings capitalized (hint: use the string's `capitalize` method). For example, `capitalize_all_words("python", "is", "awesome")` should return "Python Is Awesome". You don't need to worry about capitalizing all words in a string if there is a space in a string argument. For example, `capitalize_all_words("python", "is clearly", "awesome")` should return "Python Is clearly Awesome".

Next Time

Modules

Documentation with docstrings

- How and why can we properly document *our own* programs and functions?

Our first data structure: **lists**!

- A sequence similar to strings