Variables

- So, \( x \) becomes associated with the value 10, which is stored in memory.
- You can think of the variable \( x \) as a label for the value 10.
- Variables don’t have to be called \( x \), you can name them as you like (\( cs151 = 10 \), \( myVariableName = 10 \), \( this_is_my_variable_name = 10 \) all work), but there are rules for valid variable naming.

Variables can change value

New variables can be assigned to existing variables:

```python
gemsCollected = 5
gemsCollected = 7
print(gemsCollected)
```

Can use variable **alone** in a statement or on the right-hand-side of assignment **ONLY if already defined**.

```python
gemsCollected = gemsCollected**2
```

Right-hand-side operations happen **BEFORE** assignment

```python
x = 3
y = 4
distanceTraveled = math.sqrt(x**2 + y**2)
# distanceTraveled is assigned 5
```

Variables can be assigned to variables

```python
a = 1
b = 2
a = b
print(a)
```

*For now*, it is the case that after equating two variables (\( a \) and \( b \)) and **one changes, the other does NOT change (it keeps its value)**.
a = 1
b = 2
a = b
b = 9
print(a)  # b's old value, 2
print(b)  # b's new value 9

Reason: Variables and their values are stored in the symbol table, not just function names. When equating the variables, their symbol table value get COPIED.

Go thru example with symbol table.

Variables assigned with the 4 data types we know work using hard values

Example: If myString = 'hello' then myString = myString + 'world concatenates the strings 'hello' and 'world' and assigns it to myString.

Like before, something like myString = myString / '3' won't work.

Step thru temperature program

1. Python reads the definition of convertTtoC and stores it for later use in the main symbol table. Now Python 'knows' what the function convertTtoC is. You can call it now.
2. Python executes line 12, which is an assignment statement. Python now associated the value 32 to the variable name tempInF1. So, Python 'knows' what the first temperature is.
3. Python executes line 13, which means it must execute lines 7, then 8. A message is printed to the console because of the command on line 8: 32 degrees in F is 0.0 degrees in C.

... 

There's a few things worth pointing out about this program:

- We can pass variables into function just like if they were hard-coded values, like we're used to (e.g. convertFtoC(tempInF1)).
- We've been already working with variables in functions. In convertFtoC, the parameter temp is actually a variable.
Temperature symbol table: functions and now variables

What happens to convertedTemp after the line convertFtoC(tempInF1) finishes executing in our main code? In other words, could we access the value of convertedTemp or even temp outside the function convertFtoC, in our main code, say in between lines 14 and 17? No, it turns out that we can’t!

What we know about symbol tables so far

1. One symbol table for main code of a program. Contains your functions.
2. Separate symbol tables for modules, code you import (e.g. turtle, math, etc.)

Today, we’re going to update our definition of symbol lookup tables

There’s two new big ideas:

1. Variable names, data types, and values get stored in each symbol table, not just function names.
2. Each function gets its own symbol table when it is called.