Variables

Harry Smith

Learning Objectives

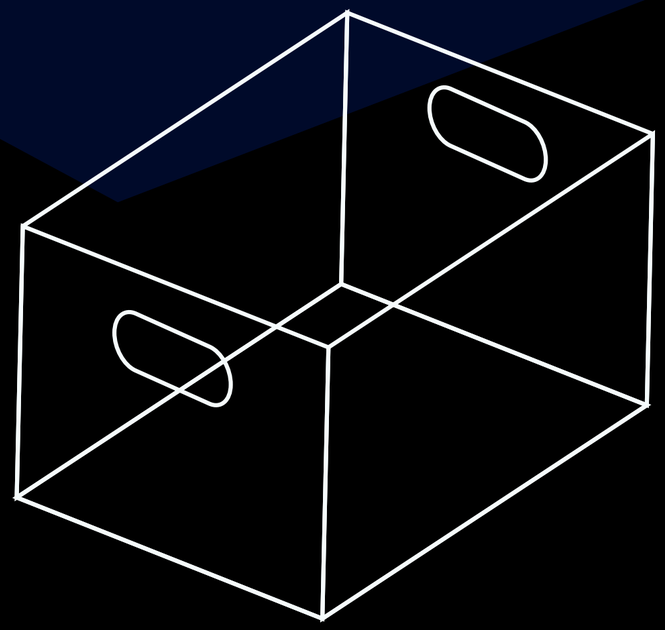
1. Declaring variables
2. Using good variable naming conventions
3. Printing using variables
4. Reassigning values to variables after they have been declared

Declaring Variables

What is a Variable?

A **variable** is a named portion of computer memory used to store a data value.

- Like a box with a name
- Can store any kind of data, but only stores one piece at a time
- Contents can *vary* throughout a program



Declaring Variables

Variable Declaration is the process of creating a variable by giving it:

1. a name
2. an initial value.

Generally:

```
new_variable_name = <expression>
```

Examples: Declaring Variables

Variable Declaration is the process of creating a variable by giving it:

1. a name
2. an initial value.

```
year = 2024
first_name = "Harry"
pi = 3.141592653
```

The Assignment Operator

- The `=` operator is the assignment operator in Python
  - Creates the variable the first time it's used
  - Assigns a new value to the variable
- Assigning a value to a variable is a *proclamation*, not a mere *question*.
  - Different way of thinking about `=` compared to, say, algebra

Naming Conventions

- Use `snake_case`: lowercase letters, separate words with underscores (`_`)
  - `first_name` is good, `firstName` or `first-name` are not.
- Variable names should be descriptive
  - Prefer `average_height` to `avg_ht` or `ah` or `x`
- Variable names can include digits but cannot start with digits
  - `color_2` is OK, but `2_color` is not.

Expressions & Using Variables

Expressions

- **Expressions** are portions of a program that have a value.
- Basic expressions are composed of **literals**, **variables**, and **operators**

Term	Definition	Example
Literal	A part of an expression that has a value which can be interpreted *literally*	`4.0` or `"python"`
Variable	A named portion of memory that stores some value	`year` or `x` or `last_name`
Operator	A symbol defining an operation or transformation	`+` or `*`

Expressions

The definitions are very formal, but expressions are usually pretty friendly.

```
5 * 3                    # has a value of 15
(9 - 3 + 10.5) * 2 / 4   # 8.25
4 == 7 - 3               # True
"yes" == "no"            # False
```

Evaluation Order of Expressions

Evaluation order in Python is generally similar to mathematics

- Use "PEMDAS" to do parentheses, then multiplication/division, then addition/subtraction.
- Hard to remember the specifics, easy to just use parentheses

Example of evaluation order:

```
(9 - 3 + 10.5) * 2 / 4 # original expression
(6 + 10.5) * 2 / 4     # equivalent expressions...
16.5 * 2 / 4
33 / 4
8.25
```

Expressions with Variables

Valid expressions may include variables!

- The value of the expression will depend on the values of the variables inside it
- You can always print the value of an expression

```
three = 3
print(three + 4) # 🖨️➡️`7`
```

Check: What does this print?

```
three = 5
print(three + 4)
```

Expressions with Variables

A variable's value can be examined by printing it out.

```
mystery = 4
print(mystery) # 🖨️ ➡️ 4
```

Expressions might consist only of variables and operators.

```
x = 17
y = 4
print(x - y) # 🖨️ ➡️ 13
```

Expressions with Variables

Variables store values.

➡️ Expressions have values.

➡️ Expressions can be written in terms of variables.

➡️ Variables can be defined in terms of other variables!

```
a = 10
b = 20
c = a + b
print(c) # 🖨️ ➡️ 30
a = 5
print(c) # 🖨️ ➡️ 30
```

The value stored is the result of evaluating the right-hand side expression at the moment the assignment is done.

Updating Variables

Updating Variables: Assigning Again

Variables can *vary*, and we make this happen by assigning a new value to them.

- Old value is totally forgotten
- Assignment works the same way: `my_var = <expression>`

```
coin = "heads"
print(coin) # 🖨️ ➡️ heads
coin = "tails"
print(coin) # 🖨️ ➡️ tails
```

Counting & Adding

Keeping score in board games and card games often relies on adding up points earned in different rounds.

- e.g. each time you play a word in Scrabble/Words with Friends
- We can use a `score` variable to represent a running total that we update with new values as they come in

```
score = 0            # starting score is 0
score = score + 8    # points for round 1
score = score + 15   # points for round 2
score = score + 22   # points for round 3
print(score)         # 🖨️ ➡️ 45
```

Keep Your Types Consistent!

Try to preserve the type of a variable over time! This code is very confusing.

```
my_name = "Harry Smith"
print("My name is:")
print(my_name)
my_name = 27
print("In three years, I will be:")
print(my_name + 3)
```

More Powerful Printing

Always Be Printing 🖨️

If you need to know the value of a variable or expression, don't guess! **Just** `print()`.

```
mystery = "hooooo egassem terces"[::-1]
print(mystery)
```

🖨️👇

```
secret message oooooh
```

Printing Multiple Things

If you want to print out multiple pieces of information on a single line, each separated with a space, you can do so by interleaving commas between the things you want to print.

```
num_bottles = 99
print(num_bottles, "bottles of beer on the wall,", num_bottles, "bottles of beer...")
```

🖨️👇

```
99 bottles of beer on the wall, 99 bottles of beer...
```

Printing Multiple Things

Each time we write `print()`, the information inside of that print statement all goes on its own line.

```
num_bottles = 99
print(num_bottles, "bottles of beer on the wall,", num_bottles, "bottles of beer...")
print("Take one down, pass it around!")
num_bottles = num_bottles - 1 # decrease the value stored in num_bottles by one
print(num_bottles, "bottles of beer on the wall.")
```

🖨️👇

```
99 bottles of beer on the wall, 99 bottles of beer...
Take one down, pass it around!
98 bottles of beer on the wall.
```

f-strings; or, Enough with the Commas!

An f-string is a slight variation of a typical string that is denoted by placing an `f` right before the start of the string:

```
msg = f"this is a simple f-string. You can tell by the f."
print(msg)
```

🖨️👇

```
this is a simple f-string. You can tell by the f.
```

f-strings; or, Enough with the Commas!

We can leave slots inside of the f-string to be filled with the result of an expression.

- Slots are denoted with curly braces (`{}`)
- Slots can be filled with any expression that you want to write.

```
age = 27
birthday = "August 29"
print(f"I'm {age}, and after {birthday}, I'll turn {age + 1}.")
```

🖨️👇

```
I'm 27, and after August 29, I'll turn 28.
```