**Data Types and Operators**

Welcome to this lesson on Data Types and Operators! You'll learn about:

* Data Types: Integers, Floats, Booleans, Strings
* Operators: Arithmetic, Assignment, Comparison, Logical
* Built-In Functions, Type Conversion
* Whitespace and Style Guidelines

**Data Types and Operators**

Welcome to this lesson on Data Types and Operators! You'll learn about:

* Data Types: Integers, Floats, Booleans, Strings
* Operators: Arithmetic, Assignment, Comparison, Logical
* Built-In Functions, Type Conversion
* Whitespace and Style Guidelines

# Variables I

Variables are used all the time in Python! Below is the example you saw in the video where we performed the following:

mv\_population = 74728

Here mv\_population is a variable, which holds the value of 74728. This assigns the item on the right to the name on the left, which is actually a little different than mathematical equality, as 74728 does not hold the value of mv\_population.

In any case, whatever term is on the left side, is now a name for whatever value is on the right side. Once a value has been assigned to a variable name, you can access the value from the variable name.

# Variables II

In this video you saw that the following two are equivalent in terms of assignment:

x = 3

y = 4

z = 5

and

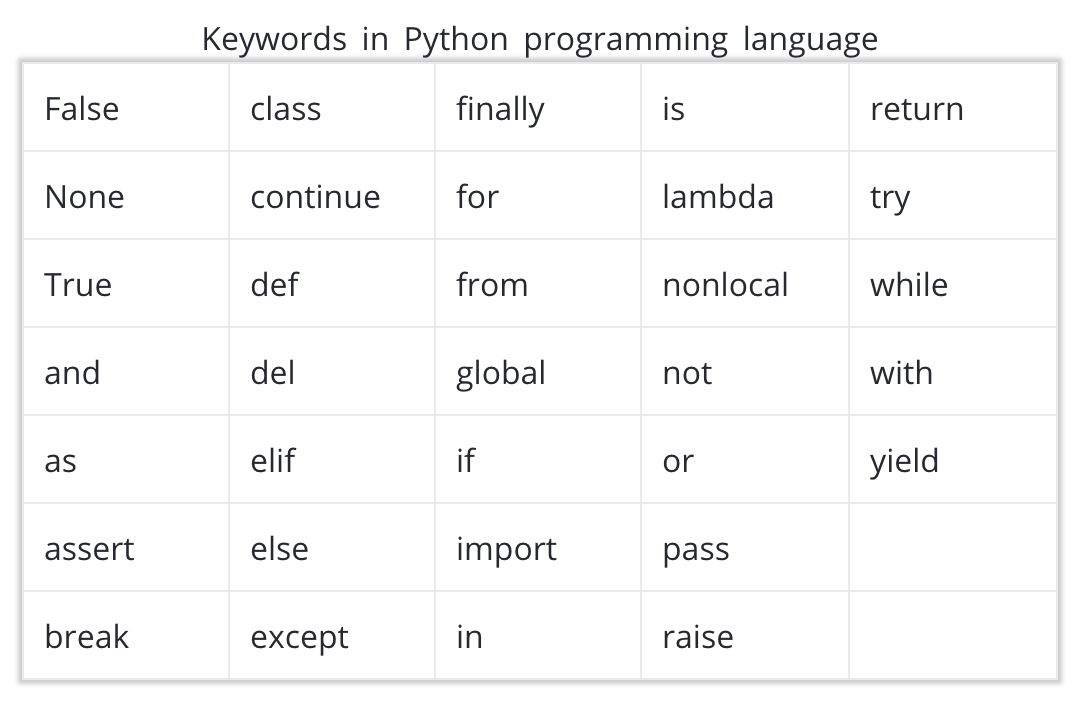
x, y, z = 3, 4, 5

However, the above isn't a great way to assign variables in most cases, because our variable names should be descriptive of the values they hold.

Besides writing variable names that are descriptive, there are a few things to watch out for when naming variables in Python.

1. Only use ordinary letters, numbers and underscores in your variable names. They can’t have spaces, and need to start with a letter or underscore.

2. **You can’t use reserved words or built-in identifiers** that have important purposes in Python, which you’ll learn about throughout this course. A list of python reserved words is described [here](https://pentangle.net/python/handbook/node52.html). Creating names that are descriptive of the values often will help you avoid using any of these words. A quick table of these words is also available below.

[[](https://classroom.udacity.com/courses/ud1110/lessons/23fc3d8b-a2a4-48ba-aada-652b2c216f2e/concepts/cf20e9f1-9b19-4bea-b2e1-d5c591e20877)](https://classroom.udacity.com/courses/ud1110/lessons/23fc3d8b-a2a4-48ba-aada-652b2c216f2e/concepts/cf20e9f1-9b19-4bea-b2e1-d5c591e20877)

3. The pythonic way to name variables is to use all lowercase letters and underscores to separate words.

**YES**

my\_height = 58

my\_lat = 40

my\_long = 105

**NO**

my height = 58

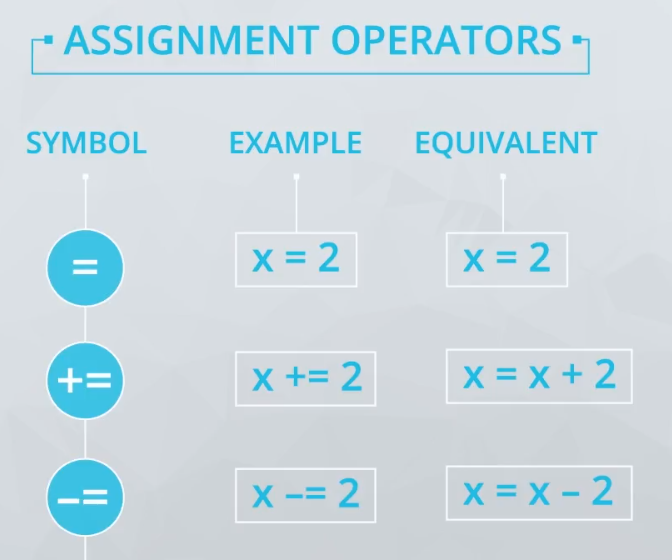
MYLONG = 40

MyLat = 105

Though the last two of these would work in python, they are not pythonic ways to name variables. The way we name variables is called snake case, because we tend to connect the words with underscores.

# Assignment Operators

Below are the assignment operators from the video. You can also use \*= in a similar way, but this is less common than the operations shown below. You can find some practice with much of what we have already covered [here](https://www.programiz.com/python-programming/operators).

[[](https://classroom.udacity.com/courses/ud1110/lessons/23fc3d8b-a2a4-48ba-aada-652b2c216f2e/concepts/cf20e9f1-9b19-4bea-b2e1-d5c591e20877)](https://classroom.udacity.com/courses/ud1110/lessons/23fc3d8b-a2a4-48ba-aada-652b2c216f2e/concepts/cf20e9f1-9b19-4bea-b2e1-d5c591e20877)

# Quiz: Changing Variable Values

How does changing the value of a variable affect another variable that was defined in terms of it? Let's look at an example.

We're intentionally not providing a place to execute the code here, because we want to help you practice the important skill of walking through lines of code by hand.

Each line of code executes in order, one at a time, with control going from one line to the next.

>>> carrots = 24

>>> rabbits = 8

>>> crs\_per\_rab = carrots/rabbits

**Type and Type Conversion**

You have seen four data types so far:

1. int
2. float
3. bool
4. string

You got a quick look at type() from an earlier video, and it can be used to check the data type of any variable you are working with.

>>> print(type(4))

int

>>> print(type(3.7))

float

>>> print(type('this'))

str

>>> print(type(True))

bool

You saw that you can change variable types to perform different operations. For example,

"0" + "5"

provides completely different output than

0 + 5

What do you think the below would provide?

"0" + 5

How about the code here:

0 + "5"

**String Methods**

In this video you were introduced to **methods**. **Methods** are like some of the **functions** you have already seen:

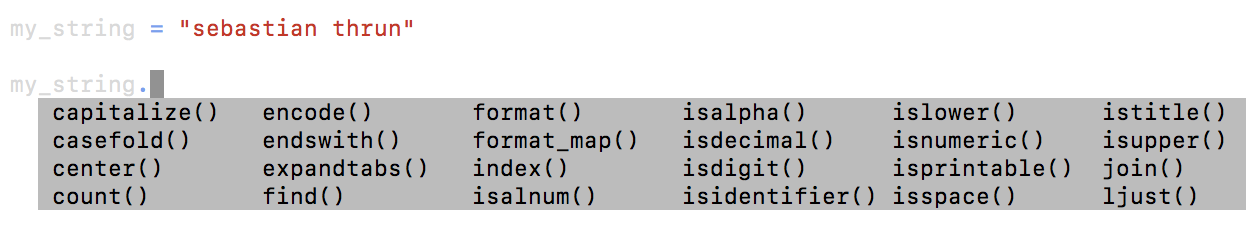
1. len("this")
2. type(12)
3. print("Hello world")

These three above are **functions** - notice they use parentheses, and accept one or more **arguments**. Functions will be studied in much more detail in a later lesson!

A **method** in Python behaves similarly to a function. Methods actually are functions that are called using dot notation. For example, lower() is a string method that can be used like this, on a string called "sample string": sample\_string.lower().

Methods are specific to the data type for a particular variable. So there are some built-in methods that are available for all strings, different methods that are available for all integers, etc.

Below is an image that shows some methods that are possible with any string.

[[](https://classroom.udacity.com/courses/ud1110/lessons/23fc3d8b-a2a4-48ba-aada-652b2c216f2e/concepts/e2ecaf07-5c9a-4ec9-be9e-e4faae40ac37)](https://classroom.udacity.com/courses/ud1110/lessons/23fc3d8b-a2a4-48ba-aada-652b2c216f2e/concepts/e2ecaf07-5c9a-4ec9-be9e-e4faae40ac37)

Each of these methods accepts the string itself as the first argument of the method. However, they also could receive additional arguments, that are passed inside the parentheses. Let's look at the output for a few examples.

>>> my\_string.islower()

True

>>> my\_string.count('a')

2

>>> my\_string.find('a')

3

You can see that the count and find methods both take another argument. However, the .islower() method does not accept another argument.

No professional has all the methods memorized, which is why understanding how to use documentation and find answers is so important. Gaining a strong grasp of the foundations of programming will allow you to use those foundations to use documentation to build so much more than someone who tries to memorize all the built-in methods in Python.

### One important string method: format()

We will be using the format() string method a good bit in our future work in Python, and you will find it very valuable in your coding, especially with your print statements.

We can best illustrate how to use format() by looking at some examples:

**Example 1**

print("Mohammed has {} balloons".format(27))

**Example 1 Output**

Mohammed has 27 balloons

**Example 2**

animal = "dog"

action = "bite"

print("Does your {} {}?".format(animal, action))

**Example 2 Output**

Does your dog bite?

**Example 3**

maria\_string = "Maria loves {} and {}"

print(maria\_string.format("math", "statistics"))

**Example 3 Output**

Maria loves math and statistics

Notice how in each example, the number of pairs of curly braces {} you use inside the string is the same as the number of replacements you want to make using the values inside format().

More advanced students can learn more about the formal syntax for using the format() string method [here](https://docs.python.org/3.6/library/string.html#format-string-syntax).

**Data Structures**

Welcome to this lesson on Data Structures! You'll learn about:

* Types of Data Structures: Lists, Tuples, Sets, Dictionaries, Compound Data Structures
* Operators: Membership, Identity
* Built-In Functions or Methods

# Lists!

**Data structures** are containers that organize and group data types together in different ways. A **list** is one of the most common and basic data structures in Python.

You saw here that you can create a list with square brackets. Lists can contain any mix and match of the data types you have seen so far.

list\_of\_random\_things = [1, 3.4, 'a string', **True**]

This is a list of 4 elements. All ordered containers (like lists) are indexed in python using a starting index of 0. Therefore, to pull the first value from the above list, we can write:

>>> list\_of\_random\_things[0]

1

It might seem like you can pull the last element with the following code, but this actually won't work:

>>> list\_of\_random\_things[len(list\_of\_random\_things)]

*---------------------------------------------------------------------------*

IndexError Traceback (most recent call **last**)

<ipython-input-34-f88b03e5c60e> **in** <module>()

*----> 1 lst[len(lst)]*

IndexError: **list** index **out of** range

However, you can retrieve the last element by reducing the index by 1. Therefore, you can do the following:

>>> list\_of\_random\_things[len(list\_of\_random\_things) - 1]

True

Alternatively, you can index from the end of a list by using negative values, where -1 is the last element, -2 is the second to last element and so on.

>>> list\_of\_random\_things[-1]

True

>>> list\_of\_random\_things[-2]

a string

# Slice and Dice with Lists

You saw that we can pull more than one value from a list at a time by using **slicing**. When using slicing, it is important to remember that the lower index is inclusive and the upper index is exclusive.

Therefore, this:

>>> list\_of\_random\_things = [1, 3.4, 'a string', True]

>>> list\_of\_random\_things[1:2]

[3.4]

will only return **3.4** in a list. Notice this is still different than just indexing a single element, because you get a list back with this indexing. The colon tells us to go from the starting value on the left of the colon up to, but not including, the element on the right.

If you know that you want to start at the beginning, of the list you can also leave out this value.

>>> list\_of\_random\_things[:2]

[1, 3.4]

or to return all of the elements to the end of the list, we can leave off a final element.

>>> list\_of\_random\_things[1:]

[3.4, 'a string', True]

This type of indexing works exactly the same on strings, where the returned value will be a string.

# Are you in OR not in?

You saw that we can also use in and not in to return a **bool** of whether an element exists within our list, or if one string is a substring of another.

>>> 'this' **in** 'this is a string'

**True**

>>> 'in' **in** 'this is a string'

**True**

>>> 'isa' **in** 'this is a string'

**False**

>>> 5 **not** **in** [1, 2, 3, 4, 6]

**True**

>>> 5 **in** [1, 2, 3, 4, 6]

**False**

**Mutability and Order**

**Mutability** is about whether or not we can change an object once it has been created. If an object (like a list or string) can be changed (like a list can), then it is called **mutable**. However, if an object cannot be changed with creating a completely new object (like strings), then the object is considered **immutable**.

>>> my\_lst = [1, 2, 3, 4, 5]

>>> my\_lst[0] = 'one'

>>> print(my\_lst)

['one', 2, 3, 4, 5]

As shown above, you are able to replace 1 with 'one' in the above list. This is because lists are **mutable**.

However, the following does not work:

>>> greeting = "Hello there"

>>> greeting[0] = 'M'

This is because strings are **immutable**. This means to change this string, you will need to create a completely new string.

There are two things to keep in mind for each of the data types you are using:

1. Are they **mutable**?
2. Are they **ordered**?

**Order** is about whether the position of an element in the object can be used to access the element. **Both strings and lists are ordered.** We can use the order to access parts of a list and string.X

However, you will see some data types in the next sections that will be unordered. For each of the upcoming data structures you see, it is useful to understand how you index, are they mutable, and are they ordered. Knowing this about the data structure is really useful!

Additionally, you will see how these each have different methods, so why you would use one data structure vs. another is largely dependent on these properties, and what you can easily do with it!