**Variables And Data Types**

What Is A Variable?

-- Variable is like a container that holds data.

-- Very similar to the containers that are in our kitchen which holds our sugar, salt etc.

-- Creating a variable is like creating a placeholder in memory and assigning it some value.

-- In Python, variables need not be declared or defined in advance, as is the case in many other programming languages. To create a variable, you just assign it a value and then start using it.

-- In many programming languages, variables are statically typed. That means a variable is initially declared to have a specific data type, and any value assigned to it during its lifetime must always have that type. For example, in C#, *int a = 100;*

-- Variables in Python are not subjected to this restriction. In Python, a variable may be assigned a value of one type and then later re-assigned a value of a different type.

Object References in Variable

-- What is happening when you make a variable assignment? This is an important question in Python, because the answer differs somewhat from what you would find in many other programming languages.

-- Python is a highly object-oriented language. In fact, virtually every item of data in a Python program is **an object** of a specific type or class.

-- Consider this code:

>>> print(300)

300

-- When presented with the statement print(300), the interpreter does the following:

* Creates an integer object
* Gives it the value 300
* Displays it to the console

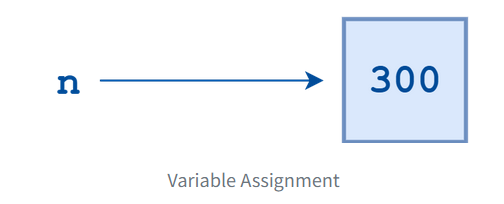
-- A Python variable is a symbolic name that is a reference or pointer to an object.

-- Once an object is assigned to a variable, you can refer to the object by that name. But the data itself is still contained within the object.

-- For example:

>>> n = 300

-- This assignment creates an integer object with the value 300 and assigns the variable n to point to that object.



-- The following code verifies that n points to an integer object:

>>> print(n)

300

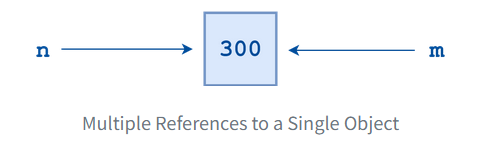
>>> type(n)

<class ‘int’>

-- Now consider the following statement:

>>> m = n

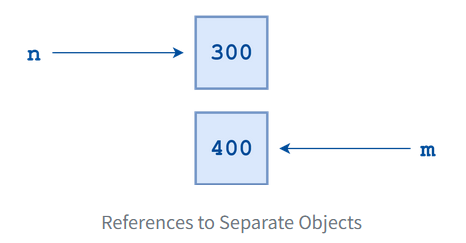
-- What happens when it is executed? Python does not create another object. It simply creates a new symbolic name or reference, m, which points to the same object that n points to.



-- Next, suppose you do this:

>>> m = 400

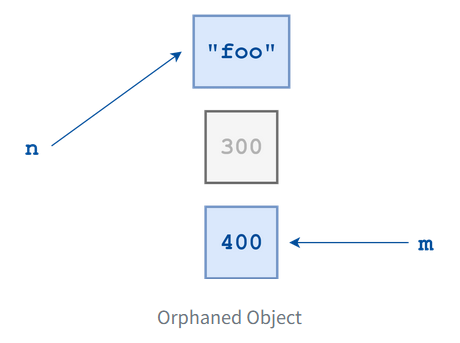
-- Now Python creates a new integer object with the value 400, and m becomes a reference to it.



-- Lastly, suppose this statement is executed next:

>> n = “Shobha”

-- Now Python creates a string object with the value " Shobha " and makes n reference that.



-- There is no longer any reference to the integer object 300. It is orphaned, and there is no way to access it.

-- When the number of references to an object drops to zero, it is no longer accessible.

-- At that point, its lifetime is over. Python will eventually notice that it is inaccessible and reclaim the allocated memory so it can be used for something else.

-- In computer lingo, this process is referred to as garbage collection.

Object Identity

-- In Python, every object that is created is given a number that uniquely identifies it.

-- It is guaranteed that no two objects will have the same identifier during any period in which their lifetimes overlap.

-- Once an object’s reference count drops to zero and it is garbage collected, as happened to the 300 object above, then its identifying number becomes available and may be used again.

-- The built-in Python function id() returns an object’s integer identifier.

-- Using the id() function, you can verify that two variables indeed point to the same object.

>>> n = 300

>>> m = n

>>> id(n)

60127840

>>> id(m)

60127840

>>> m = 400

>>> id(m)

60127872

-- After the assignment m = n, m and n both point to the same object, confirmed by the fact that id(m) and id(n) return the same number. Once m is reassigned to 400, m and n point to different objects with different identities.

Datatypes

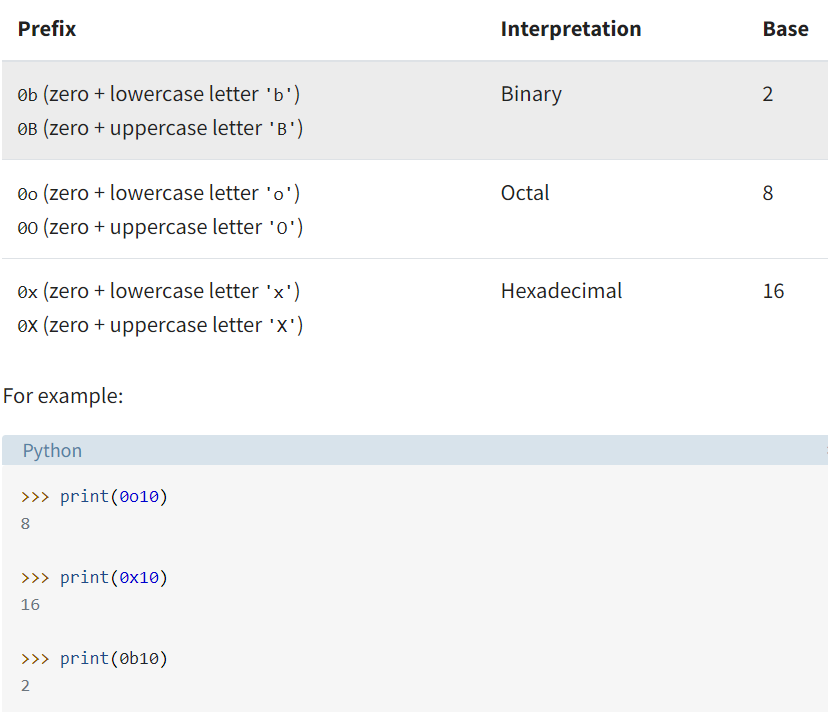
**1. Integer:**

-- Python interprets a sequence of decimal digits without any prefix to be a decimal number.

>>> print(10)

10

-- The following strings can be prepended to an integer value to indicate a base other than 10.



**2. Floating-Point Numbers:**

-- The float type in Python designates a floating-point number. float values are specified with a decimal point.

>>> 4.2

4.2

>>> type(4.2)

<class 'float'>

>>> 4.

4.0

>>> .2

0.2

**3. Strings:**

-- Strings are sequences of character data. The string type in Python is called str.

>>> print("I am a string.")

I am a string.

>>> type("I am a string.")

<class 'str'>

>>> print('I am too.')

I am too.

>>> type('I am too.')

<class 'str'>