4.1 Python Variables

Variables are containers for storing data values. Python has no command for declaring a variable. But the variables are created the moment you first assign a value to it. Example:

```
x = 5
y = "ByteUp Academy"
print(x)
print(y)
```

Also, variables do not need to be declared with any particular *type*, and can even change type after they have been set.

```
x = 4       # x is of type int
x = "ByteUp Academy"          # x is now of type str
print(x)
```

4.2 Python Variable Naming Convention

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for Python variables:

- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the below listed keywords:

Keyword	Description
and	A logical operator
as	To create an alias
assert	For debugging
async	Define an asynchronous function

await	Wait for and get a result from an awaitable
break	To break out of a loop
case	Pattern in a match statement
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs
FALSE	Boolean value, result of comparison operations
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module
in	To check if a value is present in a list, tuple, etc.
is	To test if two variables are equal
lambda	To create an anonymous function
match	Start a match statement (compare a value against cases)
None	Represents a null value
nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator
pass	A null statement, a statement that will do nothing
raise	To raise an exception
return	To exit a function and return a value
TRUE	Boolean value, result of comparison operations
try	To make a try...except statement
while	To create a while loop
with	Used to simplify exception handling
yield	To return a list of values from a generator

Python allows us to assign values to multiple variables in one line like below:

```
x, y, z = "Byte", "Up", "Academy"
```

Also, you can assign the *same* value to multiple variables in one line like below:

```
x = y = z = "Academy"
```

If you have a collection of values in a list, tuple (we will learn them in the upcoming sessions) etc, Python allows you to extract the values into variables. This is called **unpacking**.

```
center = ["Byte", "Up", "Academy"]
x, y, z = center
```

In the *print()* function, Python allows to output multiple variables, separated by a comma:

```
print(x, y, z)
```

Also you can also use the + operator to output multiple variables:

```
print(x + y + z)
```

- For numbers, the + character works as a mathematical operator:

So, what will happen if we combine a number & a string?

```
x = 100
y = "ByteUp Academy"
print(x + y)
```

- This throws error. So the best way is to separate them with commas which supports even with different data types like below:

```
x = 100
y = "ByteUp Academy"
print(x, y)
```

Normally, when you create a variable inside a function (we will learn about functions in the upcoming sessions), that variable is local, and can only be used inside that function. To create a global variable inside a function, you can use the **global** keyword.

```
global x
 x = "ByteUp Academy"
```