**Python Variables**

In Python, variables are declared without any keywords. Simply create a name and assign it a value. Since Python is weakly typed, you also don’t need to declare the data type. Creating a variable and reassigning a value to an already existing variable look like the exact same thing

**name = "James" *#*** *string*

**num = 5 *#*** *int*

**boolean = True *#*** *Boolean*

**num = 555 *#*** *reassigning a value to num*

Variables in Python are **function scoped** (when they are defined, they are known throughout the function they are created).

Variables in Python are **not hoisted** (using the variable before it is created in the code will cause an error).

**print( age ) *#*** *ERROR age is accessed before being created*

**age = 41**

**if 2 > 0:**

**job = "developer"**

**print( age ) *#*** *Works*

**print( job ) *#*** *Works*

**print( age ) *#*** *Works*

**print( job ) *#*** *Works because variables are function*

***#*** *scoped and not block scoped*

**Data Types**

Python has some data types the aren’t present in JavaScript. Below in the table you’ll see “Sequence”, “Mapping” and “Set” types which behave similarly to arrays and objects in JavaScript. We’ll discuss these in more detail later.

|  |  |  |  |
| --- | --- | --- | --- |
| Text Type: | str |  |  |
| Numeric Types: | int, float |  |  |
| Sequence Types: | list, tuple, range |  |  |
| Mapping Type: | dict |  |  |
| Set Types: | set |  |  |
| Boolean Type: | bool |  |  |

**type()**

You can use the **type( )** function to check and see what the data type of a variable is.

**type( "string" ) #** *<class 'string'>*

**type( 10 ) #** *<class 'int'>*

**type( 3.14 ) #** *<class 'float'>*

**type( False ) #** *<class 'bool'>* **\***Note you must use capital letter at start

**Casting**

When you want to be explicit when defining a variable as a specific data type you can use casting. You can also use casting to convert a variable’s data type to another, for instance turning “3.8” into a float.

**myInt = int( 3 ) myInt = int( "3" ) myInt = int( 3.9 ) #** *3, 3, 3*

**myFloat = float( 3 ) myFloat = float( "3.9" ) myFloat = float( 3.9 ) #** *3.0, 3.9, 3.9*

**myStr = str( 3 ) myStr = str( 3.9 ) myStr = str( "hi" ) #** *“3”, “3.9”, “hi”*

**Strings**

\*Note that Python doesn’t allow you to concatenate non strings with strings unlike JavaScript. Python uses the **+** operator for both string concatenation and math addition. Therefore, you’ll need to use casting when putting calculations or number variables into a string message for example.

**num = 5**

**name = "James"**

**print( num + name ) #** *ERROR can’t concatenate an* ***int***

**print( str( num ) + name ) #** *This is OK “5James”*

Or you can use **f-Strings**, a format to output calculations or variables in a string. This is done by placing **f** before the **""** and then use **{}** inside the string to compute the variable/expression. This of it as being similar to back ticks in JavaScript **`My name is ${name}`**.

**print( f"The number {num} multiplied by 2 is {num \* 2}" )**

**#** *The number 5 multiplied by 2 in 10*

**String Methods** <https://www.w3schools.com/python/python_ref_string.asp>

Python has a lot of useful in-built methods that deal with strings. Note that all string methods return new values and do not modify the original string variable.

For the below examples we are using this variable **name = "james"**

|  |  |
| --- | --- |
| **Method** | **Output** |
| **name.capitalize()** | “James” |
| **name.find( "m" )** | 2 |
| **name.upper()** | “JAMES” |
| **name.strip( "jas" )** | “me” |
| **name.replace( "ame", "X" )** | “jXs” |

**Numbers int, float**

Python excels with numbers. This is the reason why it has become a popular language for data analysis. Python does not limit how big an integer can be and the limit set on floats is 2 x 10400 which far exceeds most computer’s capabilities and is larger than the number of atoms in the known universe.

**Number Methods**

Python has only a few useful in-built methods that deal with numbers.

|  |  |
| --- | --- |
| **Method** | **Output** |
| **pow( 2, 3 )** | 8 |
| **abs( -5 )** | 5 |
| **round( 2.56 ) round( 2.56, 1 )** | 3 2.6 |

In order to avail of more mathematical methods you will have to import the in-built Python module **math**.

<https://realpython.com/python-math-module/>

**import math #** *Must include the math module to use these methods*

|  |  |
| --- | --- |
| **Method** | **Output** |
| **math.floor( 2.6 )** | 2 |
| **math.ceil( 3.2 )** | 4 |
| **math.sqrt( 9 )** | 3.0 |
| **math.pi** | 3.141592653589793 |

**Bools**

One thing to note with bools is that when you are assigning them a value of True or False you need to have to start each of those words with a capital letter.

**isOverAge = True**

**hasPassport = False**

**Truthy and Falsy**

Conditional statements such as **if** and loops that use **while** check to see if an expression equates to either **True** or **False**. However you may have noticed that these expressions can run under less obvious circumstances. This behaviour isn’t just connected to Python as both examples below, even the JavaScript one, will run and print “TRUE”

**if 5: if ( 5 ) {**

**print( “TRUE” ) console.log( “TRUE” )**

**}**

This is because individual expressions and values can equate to true or false. Values that equate to False are called Falsy and values that equate to True are called Truthy. Below you will see how zero values or empty strings and collection variables are considered falsy when checked while any number that isn’t zero nor a non-empty string or collection equates to **True**.

**Falsy Values Truthy Values**

**0 == False 1 == True**

**[] == False [ 2 ] == True**

**() == False ( 3 ) == True**

**{} == False { a: "1" } == True**

**"" == False "A" == True**

**False == False True == True**