**Python Syntax: DataTypes**

**Data Types:**

1. **Integers (int)**: Whole numbers, e.g., **5**, **-10**, **100**.
2. **Floating-Point Numbers (float)**: Numbers with a decimal point, e.g., **3.14**, **-0.5**, **2.0**.
3. **Strings (str)**: Text or sequences of characters, e.g., **"Hello"**, **'Python'**, **"123"**.
4. **Boolean (bool)**: Represents binary values **True** or **Fals**
5. **Complex numbers:** Complex numbers represent numbers as a pair of double-precision numbers.

**Intergers**

Python has three distinctive numeric types; this means that for every number that you use in your programs there is a suitable data type that you will use to ensure that your programs are truly performance efficient. As you know by now, integers are numbers. Numbers can have negative and positive values. Monetary numbers can have decimal values.

We say that Python is a static language because only values of a certain type can be assigned to particular variables, according to their data types assigned.

Integers are always whole numbers. Integers include negative and positive numbers. The only factor that determines the range of an integer variable is the amount of memory a machine has available.

**Booleans**

Boolean data type has corresponding integer values. There are only two possible values that a Boolean variable can have, **True** (1) or **False** (0). When returning Booleans as strings they are seen as “True” and “False”, and never as “1” and “0”. True and False are case-sensitive in Python. Boolean tests whether conditions are valid or not. The three logical operators used to test conditions between two arguments are:

* The **and-operator**
* The **or** operator
* The **not** operator

The next section will refresh your memory, and you will soon be able to write programs that execute operations automatically based on decisions (such as the if statement explained in week 2). The following variable values are considered **False**:

* False
* None
* Zero for any numeric data type, **0, 0.0, 0j**
* An empty sequence or mapping. Like a list or tuple, **' '**, **( )**, **[ ]**, **{ }**
* Instances of user-defined classes, where a class that defines a**\_\_bool\_\_()** method returns **zero** or **False**.

All values returned otherwise are always considered true. This means that many objects will always return true.

Operators and built-in functions that have a Boolean result always return (False or 0) or (True or 1). The Boolean **or** and **and** operations always return only one of the options, either **True** or **False**.

**Floating Point Numbers**

Floating point numbers are better known as floats. Float is the data type that manages numbers with decimal places with very accurate precision. The float data type can be called as a function with zero or 1 argument of any data type. If no argument is given, then float returns 0.0. If an argument is given, an attempt will then be made to convert the value to a float data type, but this does not mean it is always possible.

**Complex numbers**

Complex numbers are two numbers contained in a single variable. The first part of a complex number is the real part (float), and the second part is the imaginary part (float), assigned in this manner: complex(real, image). Imaginary numbers are real multiples of the imaginary unit, written with a suffix of **j (J)**. The imaginary part is **the square root of -1**. Python has built-in support for complex numbers. The latter notation is written as follows: 4+8j.

Complex numbers are used in Python to combine two numbers into one manageable number. The following example shows how complex numbers are used in a simple manner.

**Strings**

 Strings are represented by the immutable (unchangeable) str data type. Strings are a sequence of Unicode characters which form a single manageable string. The str data type can be called to create a string; when there is no argument supplied, it returns an empty string. s = str("") is the same as s = str(), when an argument is passed to the string method that is not a string value, it is passed as a string representation of the type supplied: s = str(17.2354), is the same as s = str("17.2354"). The string function is often used to convert other data types to strings.

**Lamda Expressions**

Small anonymous functions can be created with the lambda keyword. This function returns the sum of its two arguments: lambda a, b: a+b. Lambda functions can be used wherever function objects are required.

They are syntactically restricted to a single expression. Semantically, they are just syntactic sugar for a normal function definition.

**Function Annotations**

Function annotations are completely optional metadata information about the types used by user-defined functions.

Annotations are stored in the \_\_annotations\_\_ attribute of the function as a dictionary and do not affect any other part of the function. Parameter annotations are defined by a colon after the parameter name, followed by an expression evaluating the value of the annotation. Return annotations are defined by a literal ->, followed by an expression, between the parameter list and the colon denoting the end of the def statement.