Everything is an object in Python

In Python, everything is an object. This means that variables, functions, and even classes are all objects. Objects have properties and methods, which can be used to access and modify their data.

The "everything is an object" concept is important because it allows us to think about data in a more abstract way. In other languages, data is often stored in primitive types, such as integers, strings, and floats. These types are limited in what they can do. For example, an integer can only store a whole number. A string can only store a sequence of characters.

The built-in function ID in Python enables us to check the unique identifier of an object.

assign a variable to an object

**>>> a = 2**

**>> id(a)**

**....there will be a code of unique numbers/identifiers (serial)**

**>>> b = 3**

**>>> id(b)**

**>>> id(a)==id(b)**

**.....the return will be false**

**>>> d = 2**

**>>> id(a)==id(d)**

**true .....having the same value will point to the same object having the same id #assigning two objects 1.'Abble' 2.'abble' will have a different id() because of a small and capital letter**

Type another Python function

check the type of any valuable type()

**>>> a = 32**

**>>> type(a)**

**>>> int(integer)**

A mutability object is an object whose data can be changed means you can add changes, or remove without creating new objects. Examples of types are

mutable lists, dictionaries, and sets.

**>>> mutab\_obj = ['a','b']**

**>>> mutab\_obj[0] = 'z'**

**>>> mutab\_obj**

**,,,,'z','b'**

Immutability can't be changed means once the element is assigned to an index position you can't grab the element and reassign it to something else e.g. string, integers, float, and tuples

**>>> immutab\_obj = (1,2,3)**

**>>> immutab\_obj[0] = 'one'**

**........error**

**eg tuple >>> y = (1,2,4) can even be a string**

**>>> y = ('one', 'three')**

**>>> my list = ['new', 'old', 'updated']**

**#try to reassign the first element of my list to be one**

**mylist[0] = 'one'**

**>>> Try to run my list it will be displayed with a replacement on one**

**t[0] = 2**

**type error will occur**

Python treats immutable and mutable objects differently in a few ways.

For immutability passing around objects in your program and making sure they're accidentally changed and are immutable helps easily to find out the current states of an object and guarantee the value not be changed.

The difference between immutable and mutable objects is an important concept to understand in Python. By understanding this difference, we can write more concise and reusable code. We can also avoid unexpected behaviour when passing objects to functions. Happy Coding….