THE CONCEPT OF IMMUTABILITY IN PYTHON:

Immutability means tendency not to change.

The concept of Immutability is a very simple yet a very powerful concept in Python. Python is a fully object-oriented programming language that is everything in Python is an object.

There are two type of objects that are being created by python:

1)Mutable (list, set, dict, bytearray)

2)Immutable (int, float, str, bool, complex, tuple, frozenset)

The concept of immutability says that once an object is created its value can never be changed if it is an immutable object, whenever we try to change the content of an immutable object a new object will automatically be created.

Example

```
# an int object with value 10 is created
a=10
id(a)
# we try to change the value of ' a '
a+=1
id(a)
```

a → 10

a → 11 10

Output:

```
1623315648
1623315664
```

Clearly both the memory locations are different as shown in the figure and this is the case with every immutable type. The object earlier created with value 10 is now available to garbage collector since it is not referenced by any variable.

HOW DOES IMMUTABILITY HELP?

Immutability helps in reusing of objects. Reusing objects helps in better performance and better memory management, thus immutability help in optimizing speed and memory usage.

Example:

```
x=10
y=10
# here only one object is created instead of two
print (x is y)
# checks if id(x)==id(y)
```

x → 10

y →

Output:

```
True
```

Now assume that there was no immutability then changing the value of Y, would have automatically changed the value of X, which was not at all desirable.

This is where immutability is required, to ensure that the speed and memory management is not compromised along with data being secured.

Some mutable data types are also there which helps in data passing and modification among different scopes.

Reusability is applied in python to fundamental data types only and that too on limited ranges which are listed below:

1) Int: 0-256
2) Float: No reusability of float type objects
3) Str: Strings objects containing Unicode alphabets and digits are reusable
4) Bool: Both True and False objects are reusable
5) Complex: No reusability of complex type objects

Example:

```
#string containing only alphabet and digit
a=”str1”
b=”str1”
print (a is b)

#string containing anything other than alphabet and digit
a1=”str str”
a2=”str str”
print (a1 is a2)

#floating value
f1=3.6
f2=3.6
print (f1 is f2)
```

Output:

```
True
False
False
```