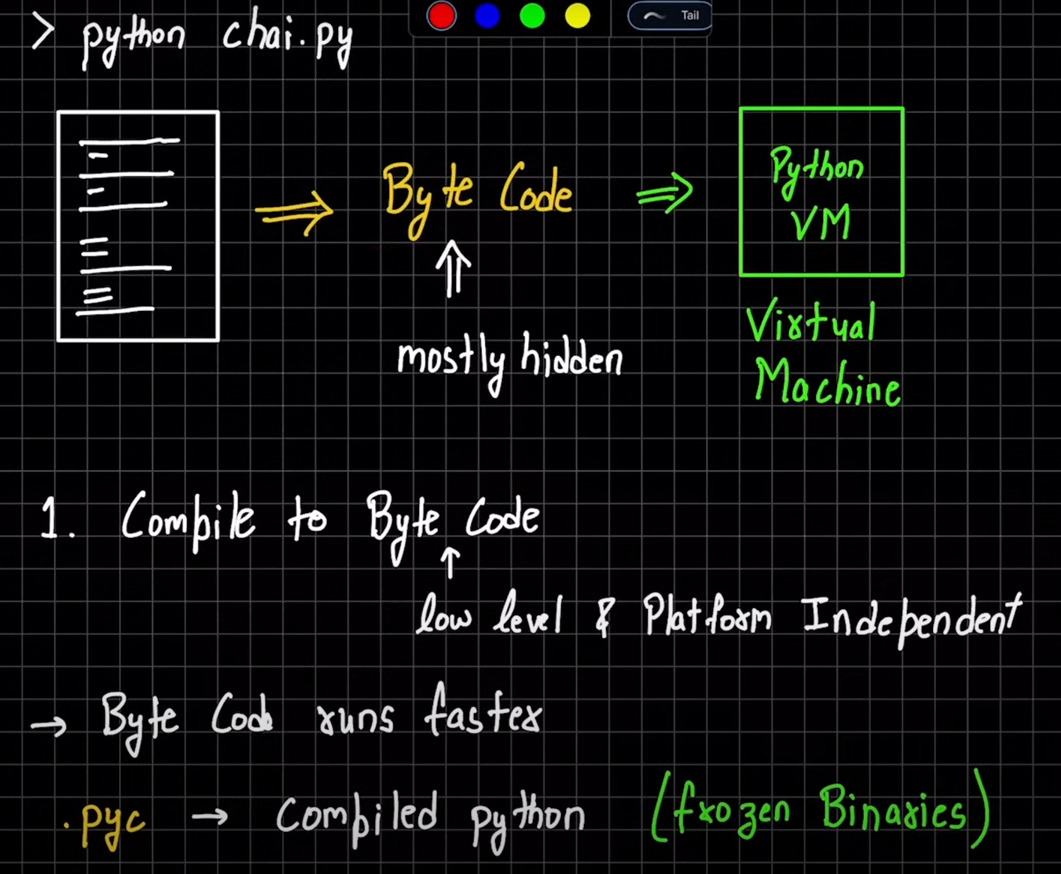
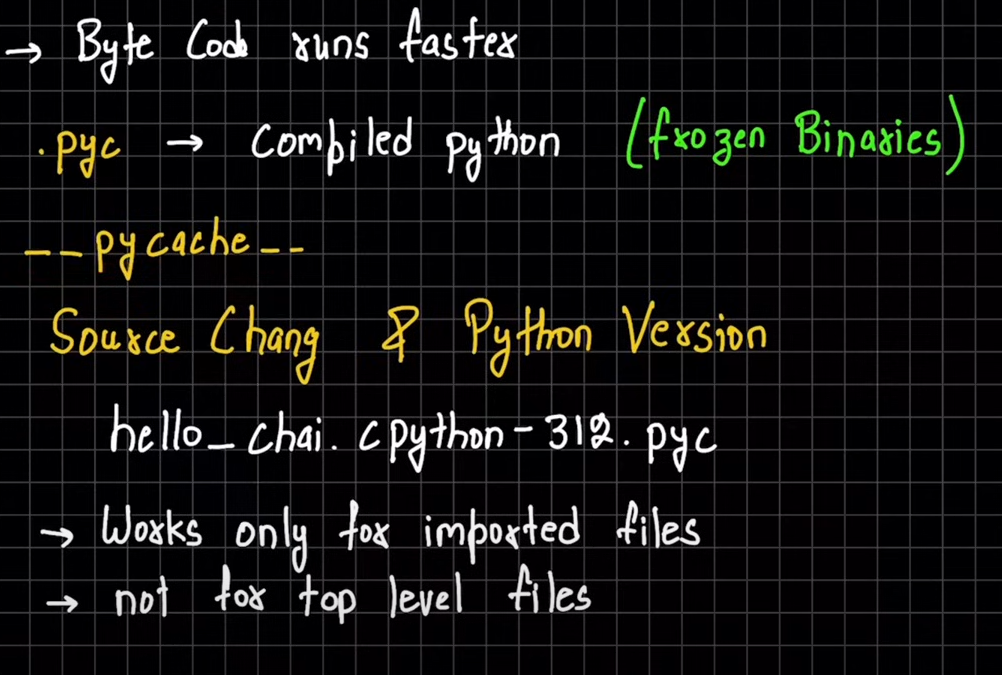
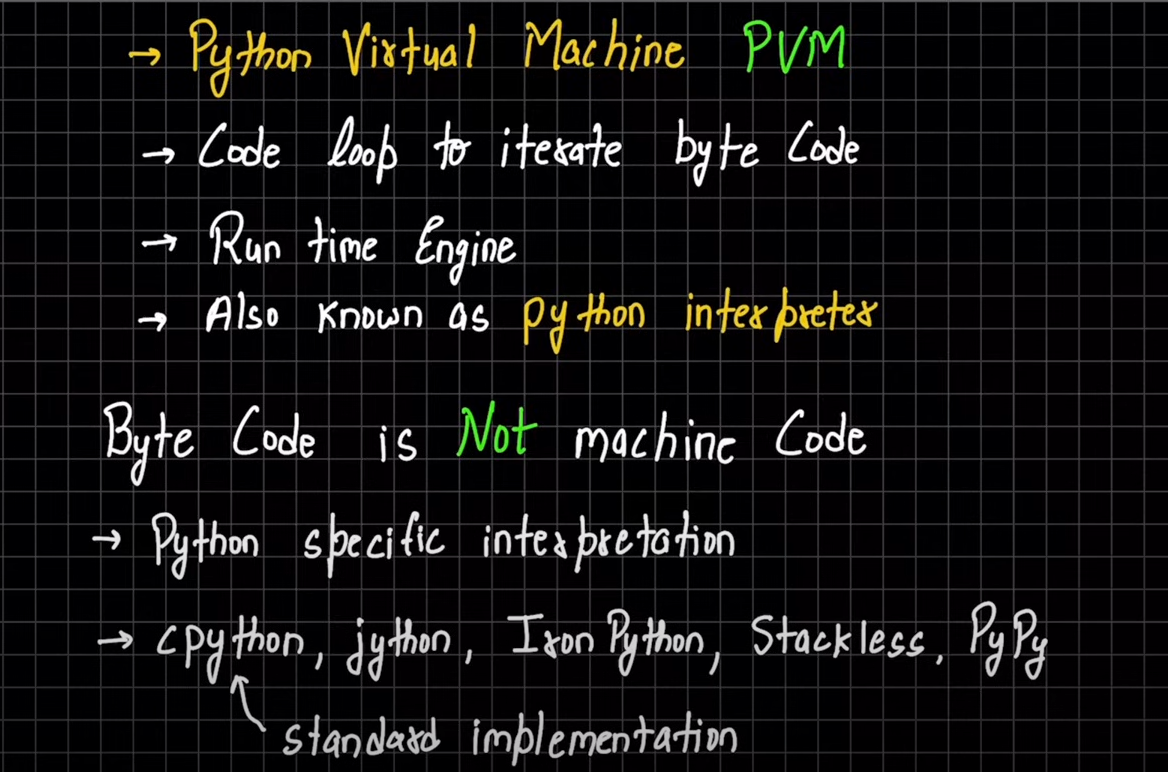
**Python a new start:**



****

****

Machine code== direct instruction to the python

**Mutable and Immutable**

In Python, everything is an object.

Immutable== The reference which is made inside the memory can never be changed.

Eg.

>>> x = 10

>>> y = x

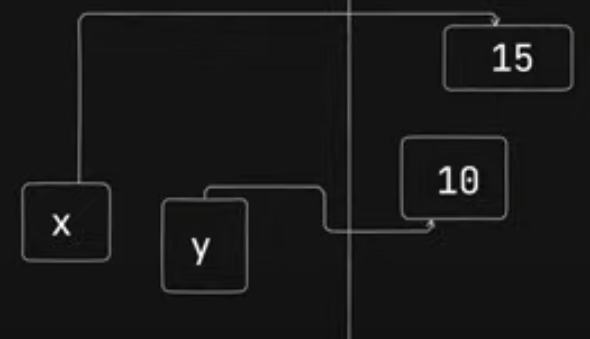
Memory diagram:  


Now change in the code.

>>> x = 15

What will be y =?

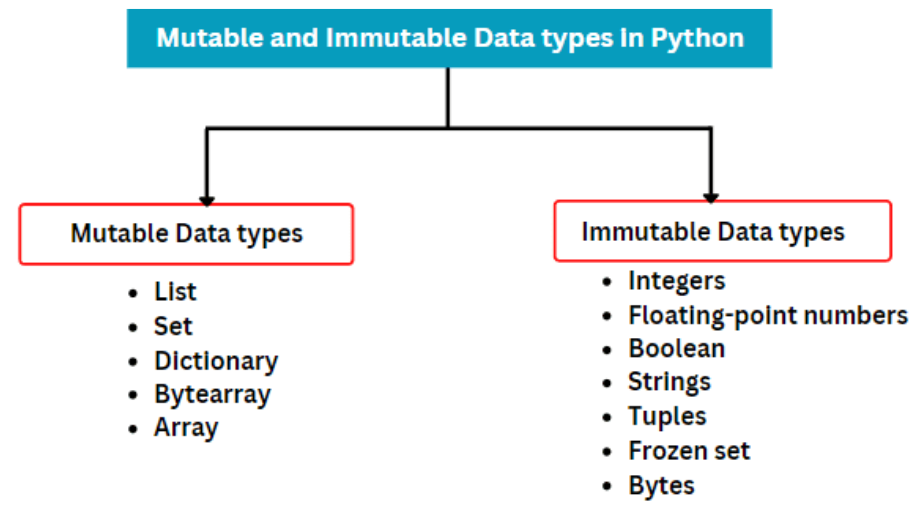
Let’s see in the memory diagram:



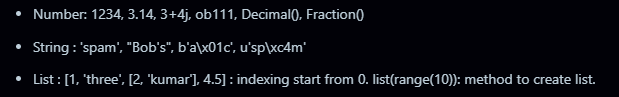
Here we can clearly see, the reference of x is change to 15 but y is still referencing to 10.

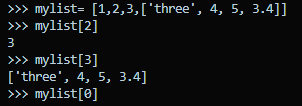
So, y=10;

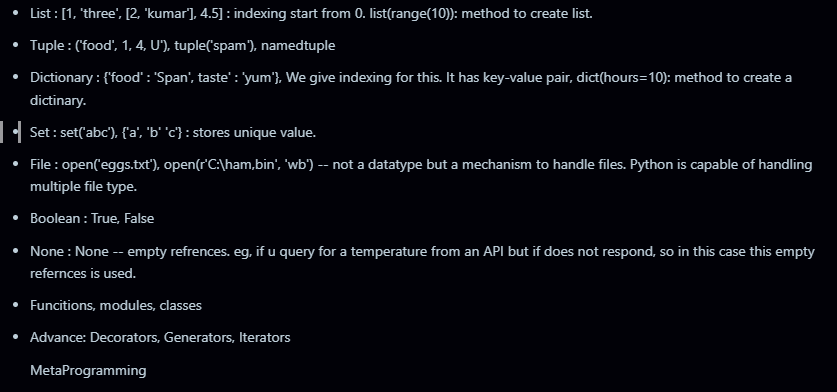
And hence, it is immutable.



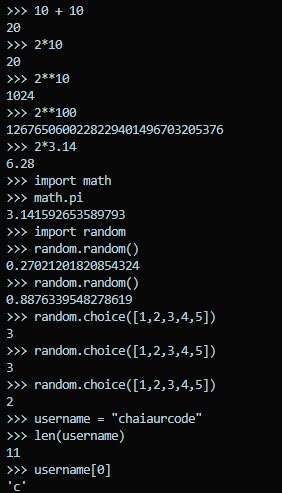
**DATA TYPES / OBJECT TYPES:**

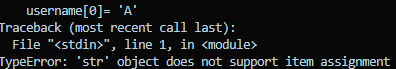
****

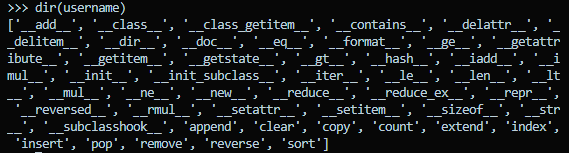
****

****

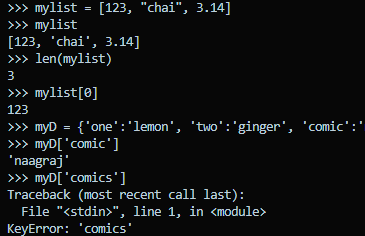
**Basic overview of each:**

****

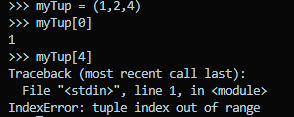
****

**When we tried to replace zeroth index of username with new value, error is given because “String” is an immutable object. **

**dir (username): list out all the methods available in python for the object/ datatype username.**

****

* **mylist is same as array in other language, it has same properties as arary.**
* **In dictionary, when key is not available, it gives key error.**

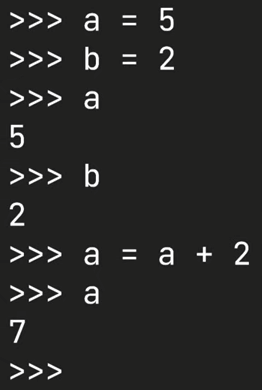
****

**INTERNAL WORKING OF PYTHON:**

\*\*We never assign any datatype to variable in python, it is referenced only in memory.

a = 3

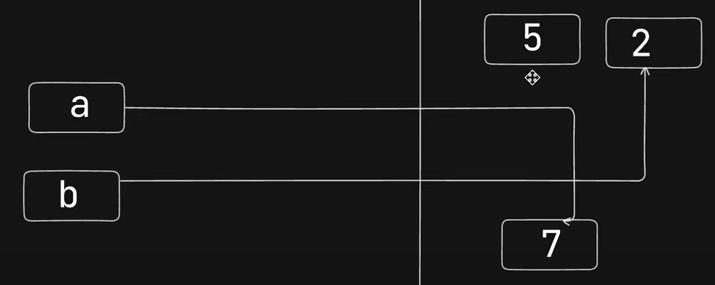
here a is not of datatype “Number”. 3 is of ‘Number’ type which is stored in a variable a.



Let’s understand it through memory referencing.

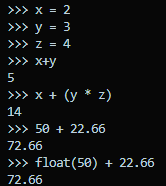
initially, this happens.



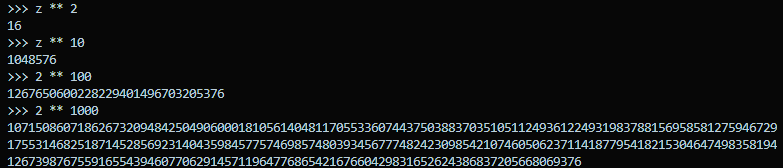
as, a = a + 2 #comes in pictures. a becomes 7. Now, reference of a is changed to 7.  
 

**NUMBERS IN PYTHON**number carry everything in python – int, float, even sets are internally equal to number.

boolean is also number internally.

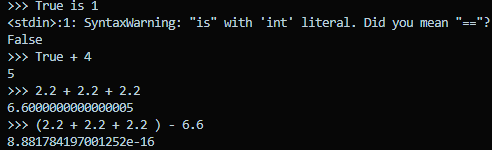


try using same datatypes explicitly for better code readability.



Python has highest no handling capacity and high precision.

\*\*Python has no precision as infinite but it lacks in decimal values.



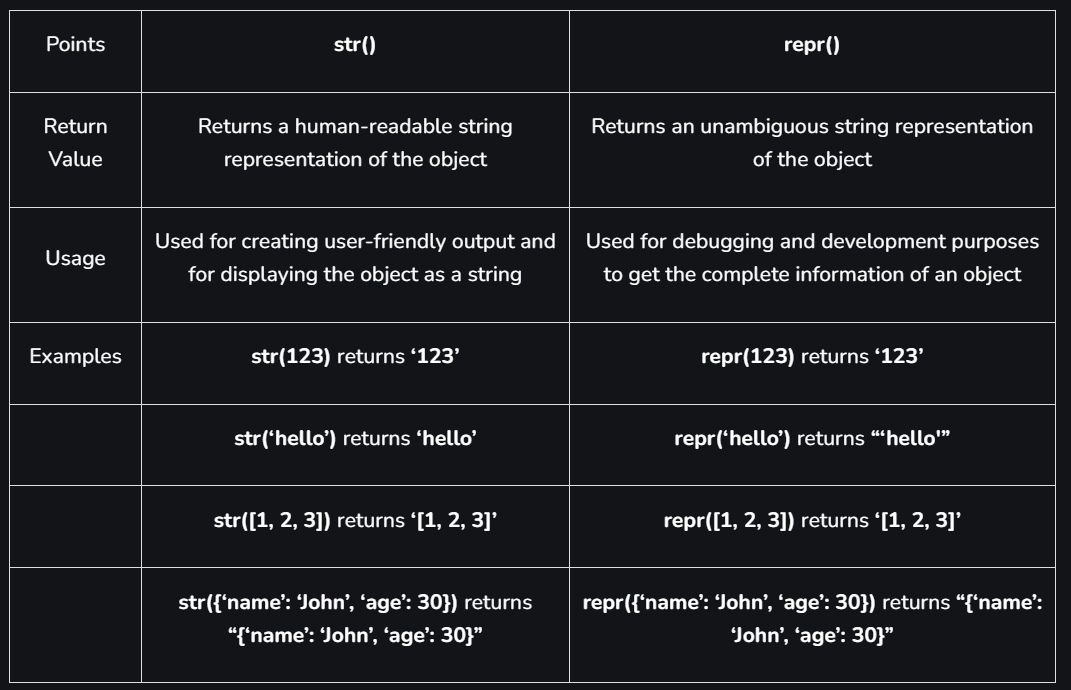
\*\* for handling decimal error in python, we need to import decimal library.

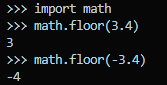
****

Syntactically true: 

But production level, we must write it as more readable: 

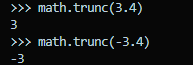
Difference between \_\_repr\_\_ and \_\_str\_\_:





import math:

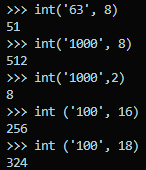
floor: it gives closest no below the value.

math.trunc(): this goes towards zero in no line: 

0o: represent octal literal : 

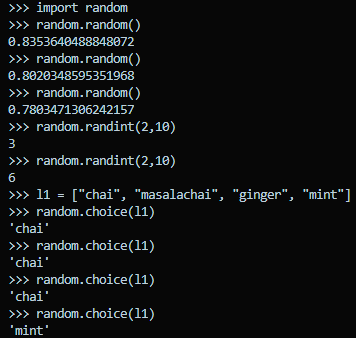
0x: represent hexadecimal literal: 

0b: represent binary literal : 

\*We can convert any no system into int with the help of integer method.   
here is how:

import random: this random library helps in generating random number between 0 to 1.

but it can used to perform various task:

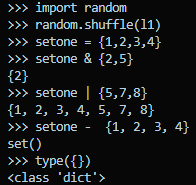


**SETS:**

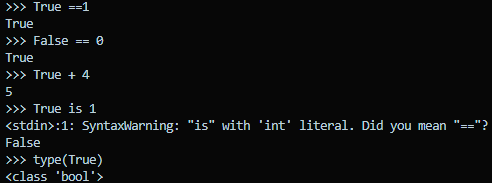
**&** represents: intersection

**|** represents: union

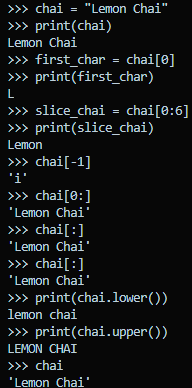
empty set is denoted as = set() because {}: it is dictionary in python.

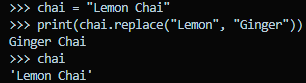


**Boolean**

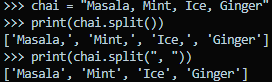
****

**Strings in Python:**





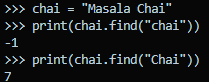
* **change a string into a list[]:**



split(): by default, it splits on the basis of spaces.

split(“, ”): this splits by removing “, ” from the string.

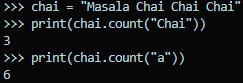
* **Find something in string.**

****

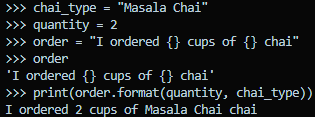
If nothing is found in the string. It returns -1.

If found, it gives the starting index of the character, string.

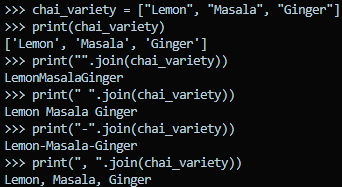
* **Finding no of character or string**

****

* **Importance of placeholder ({}).**

****

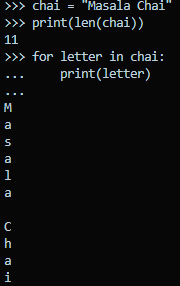
* **Changing lists into string.**

****

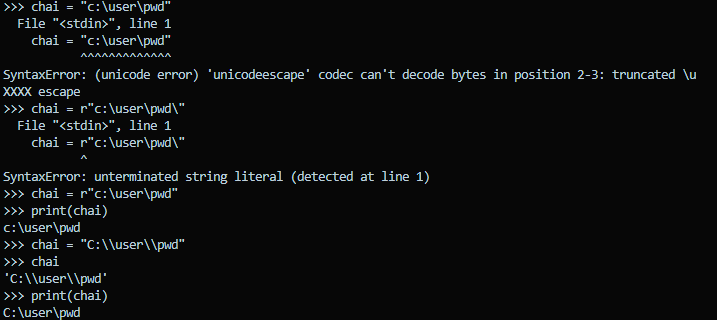
* **Length of string.**

****

* **Printing all the letters of a string.**

****

* **Use of raw string in python : basically, used to print the addresses[path of directories] of windows.**

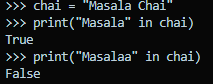
****

**Notes:**

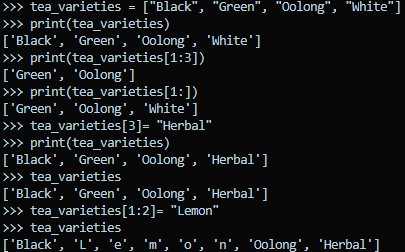
regarding errors: ‘\’ is a Unicode, so it can be included in the string as normal string. To include this, we either use “raw string” or “one extra ‘\’ ”.

also, in raw string there should not be any ‘\’ in the end of string.

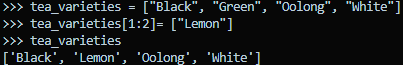
* **We can ask some question like this:**

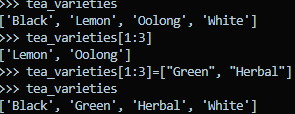


**LIST [ARRAY] IN PYTHON**

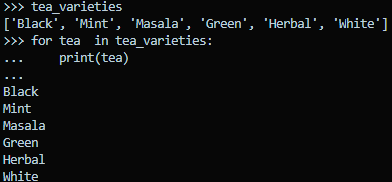


when we are trying to insert or replace an element in list using slicing dicing then one-one character of element is inserted because each character is treated as an array. To solve this, we will pass element as list or array.

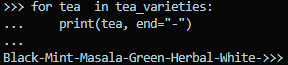




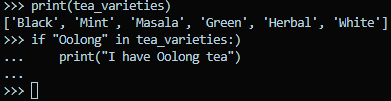




* “\n” is the default after each iteration, we can define our own like this:

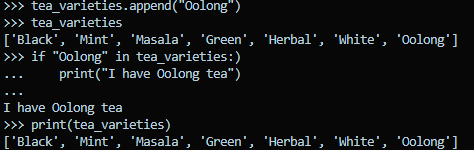


* we can ask question in this way:

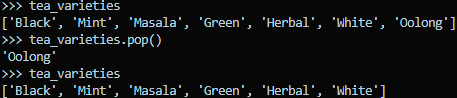


nothing is printed as there was no “Oolong” tea.

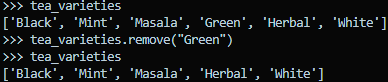
* append() is used to add an element at the end of the list.



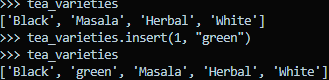
* to remove last element, we use: .pop()



* to remove any specific element, we use .remove(“element”)



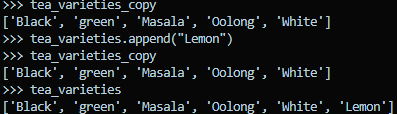
* to insert any element at a position



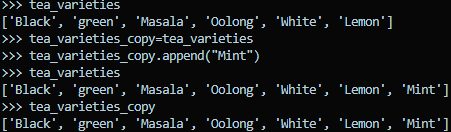
* when we copy our list to another variable, our new variable get the same reference as previous one, but if we want new reference, we can use .copy() method:



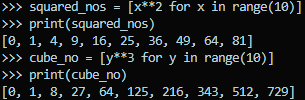
see the difference clearly:



if references would have been seen, there would have been change in both as well, see:

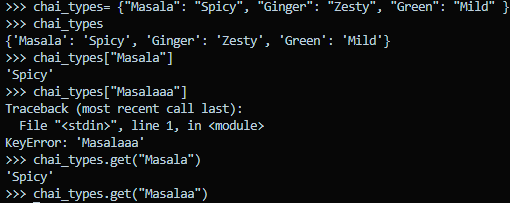


* list comprehension in list.



**DICTIONARY IN PYTHON**

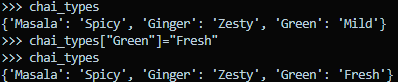
It stores elements in key-value pair. It is useful in various places, eg, if we want to get DOB out of all the elements. We can access DOB by DOB key.



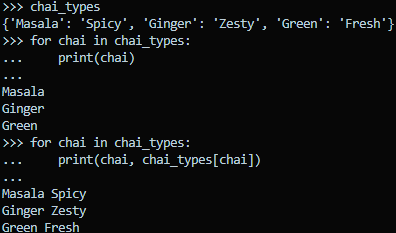
dict is a special datatype so it comes with special method.

we can access the value of key by get(“key”), however this doesn’t throw any error if key is not available. But when we access any key normally, the error comes when key is not available.

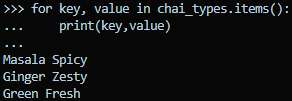
* Change the value of any key



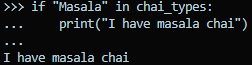
* print key-values using loop.



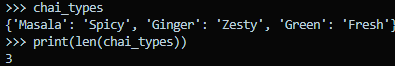
* accessing key, value with loop and method: .items()



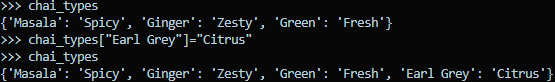
* asking a question



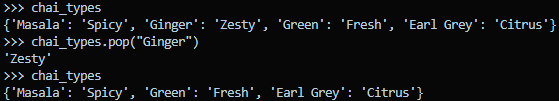
* len() method works well with dictionary.



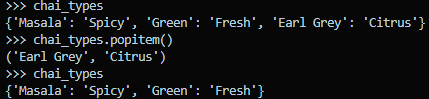
* adding a key-value



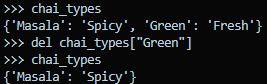
* pop() also works well, but we need to provide key here.



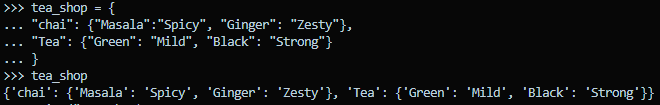
* by using .popitem(): we can we remove the last added key-value in the dictionary. This does not require any key:

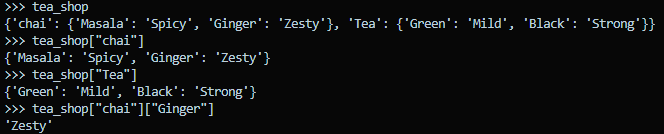


* del method: it actually deletes the reference from the memory.

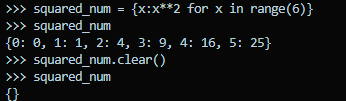


* dict inside a dict

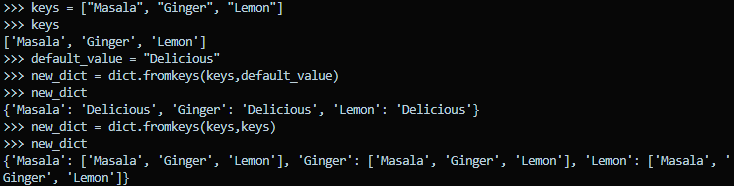



* some more stuff



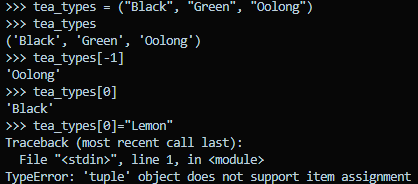
* create a dict using list and string

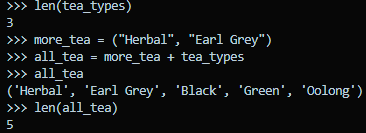


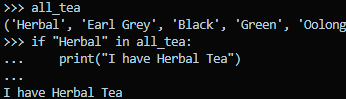
**TUPLES IN PYTHON:**

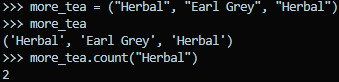
Why do we need tuples, when we have list which does the same work?

Ans: List is mutable, Tuples is immutable.

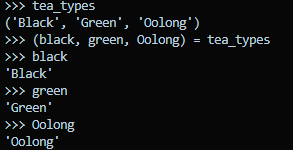




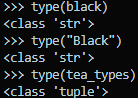




* tuples are used to unwrap tuples



* type() find the type of the datatypes.



* **How to take an input?**

You can take input from terminal.

Or using input(“statement”)

The input taken from the input statement is always a str.

* **If- else conditional:**

If condition:

print statement.

or

if condition:

print statement

elif condition:

print statement

else:

print statement.

* **Writing single line if statement.**

price = 12 if age>=18 else 8. => this means price is equal to 12 when age is greater or equal to 18, else 8.

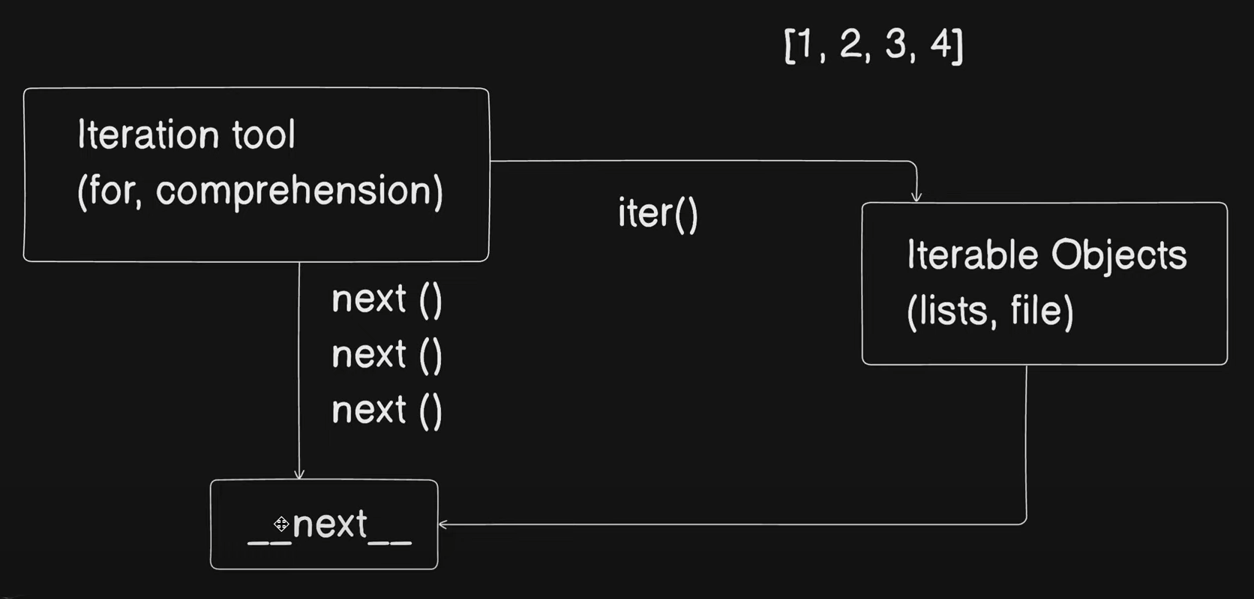
**Loops:**

range(starting, ending):ending is always exclusive.

continue: it skips the current iteration.

break: it exits all other existing operation. (say in a loop).

* Behind the scene of loop or iteration tool.

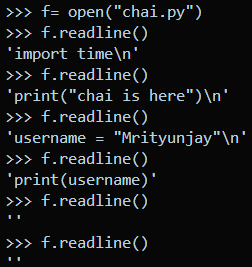


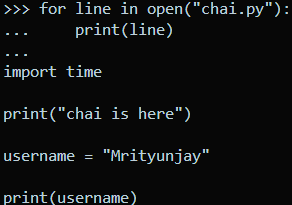
import time

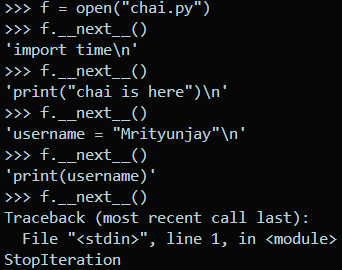
print("chai is here")

username = "Mrityunjay"

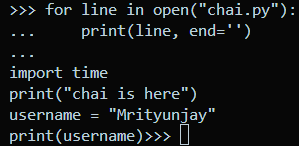
print(username)

when all the lines are ended in the file, then there is a ‘’ (empty string) which is a kind of notation to say = file ended.

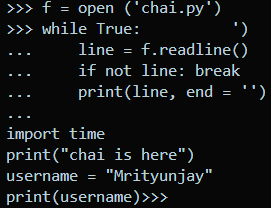
we can also use \_\_next\_\_() to read the files line by line, but in this case, when line will end, it will give error to stop iteration. we can also use loops to read the lines.



* to remove the space between the line, while printing through loop, we can modify our print(line, end= ‘’)



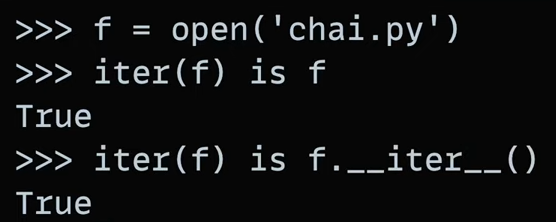
* printing the line using while loop



* when we store a file inside a variable, that variable is also a iterable object.

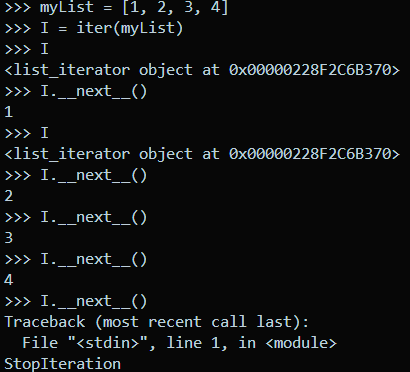
iter(f) and f is same only for file.

**Note:** iter object is by default in files.

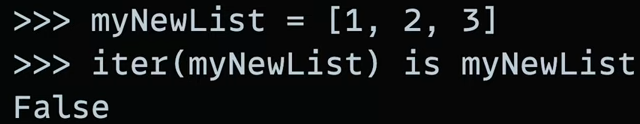


* when we hold the reference of iter() of any list.

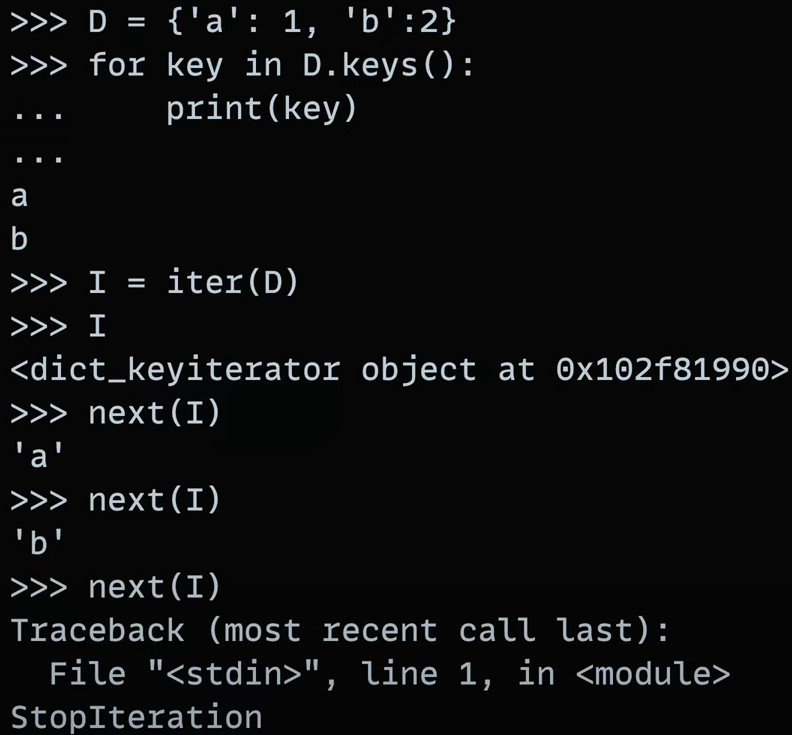
Note: memory reference of lift iterator always points at starting.



* In case of list, only reference of variable is passed. Hence when a list is stored in a variable, it is not iterable.



* Dictionary is also an iterable object.



* **Functions**

def square(parameter):

print() or return

* **polymorphism in function**

polymorphism == ek naam, anek kaam.

* Funtction with default parameter:

**def** greet(**name** **=** "Mrityunjay")**:**

**return** "Hello, " **+** name **+** "!"

print(greet())

print(greet("Hitesh"))

* assign a default value to the variable in function parameter, if no argument is passed, then variable value will be treated as default parameter.

**Output:**

* **Function with \*args**

**args can take n no of parameter.**

**def** sum\_all(**\*args**)**:**

**return** sum(args)

print(sum\_all(1**,**11))

* **\*\*kwargs 🡺 it helps return output in key value pair.**

**def** print\_kwargs(**\*\*kwargs**)**:**

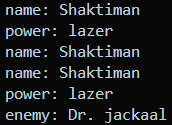
**for** key**,** value **in** kwargs**.**items()**:**

        print(f"{key}: {value}")

print\_kwargs(**name=** "Shaktiman"**,** **power** **=** "lazer")

print\_kwargs(**name=** "Shaktiman")

print\_kwargs(**name=** "Shaktiman"**,** **power** **=** "lazer"**,enemy** **=** "Dr. jackaal")

**output:**

* yield returns the value, but keep update of what is happening in the memory, also it manages the state.

**def** even\_generator (**limit**)**:**

**for** i **in** range(2**,** limit**+**1**,**2)**:**

**yield**  i

**for** num **in** even\_generator(10)**:**

    print(num)

output:

* **scope (namespace)**

in python, indentation is scope.

* **Closure (factory function)**

**def chaicoder(num):**

**def actual (x):**

**return x \*\* num**

**return actual**

**f = chaicoder(2)**

**g = chaicoder(3)**

**print (f) # only reference of f is passed here.**

**print(g) # only reference of g is passed here.**

output: 

let’s execute the function here.

**def** chaicoder(**num**)**:**

**def** actual (**x**)**:**

**return** x **\*\*** num

**return** actual

f **=** chaicoder(2)

g **=** chaicoder((3))

print (f(3))

print(g(3))

output:

* **OBJECT ORIENTED PROGRAMMING.**
* Basic syntax:

class **Car:**

**def** \_\_init\_\_(*self***,brand,model**)**:**

*self***.**brand **=** brand

*self***.**model **=** model

my\_car **=** Car("Toyota"**,** "Corolla")

print(my\_car**.**brand**,** my\_car**.**model)

output:

* self: sets the context. In lay man language, self helps to talk from Car and also this helps to refer itself or say (parameter or say attributes)
* when you add \_\_brand: then this attribute becomes private. This mean, class can access that attribute but object cannot.
* static method 🡺 it is a method that can be accessed only by class, can not be accessed by instance (object) @staticmethod = also know as decorator
* property decorator 🡺 it makes sure that, we can not modify any attribute.

@property

**def** model(*self*)**:**

**return** *self***.**\_\_model

also to call this method, we need only reference of the method model, not the execution

e.g.,

print(my\_car**.**model)

* pass statement is a null operation in python.

class **ElectricCarTwo**(*Battery***,***Engine***,***Car*)**:**

**pass**

here pass does nothing, hence null operation.

* **Decorators**

decorators are like toll booth in real life.

* a function is passed into a pipe (decorator)