TERMS:

|  |  |  |
| --- | --- | --- |
| ITERABLE: | Iterable is **an object which can be looped over or iterated over with the help of a for loop**. Objects like lists, tuples, sets, dictionaries, st rings, etc. are called iterables. |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

\*The numbers with e in it means the scientific numbers are treated as float numbers(ex: 2e10,5.E10)

\*The complex numbers will have j as imaginary part(x+2j)

\*Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

import random

print(random.randrange(1, 10))

\*NOTE: In cast typing string to integer the value of string must be a whole number cannot be a floating value or letters and butin case of converting string to float the string might even be a integer value

Ex: a=int(“2.0”)🡺error

But a=float(“4”)🡺gives output

* Multiline strings: You can assign a multiline string to a variable by using three quotes:

|  |
| --- |
| a = """Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.""" print(a)  or  a = '''Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.''' print(a)  note: in the result, the line breaks are inserted at the same position as in the code. |

Strings are arrays:

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

U can check whether certain phrase or character is present in a string, in keyword is used to check.

txt = "The best things in life are free!"

print("free" in txt)

output: True

or use if

txt = "The best things in life are free!"

if "free" in txt:

  print("Yes, 'free' is present.")

if u want to check not present just use not before in keyword

print("free" not in txt) #output:False

NOTE:

if the there is space in a string like “hansi me” here space will have index 5 as indexing starts from 0

Negative Indexing:

H A N S I

-5 -4 -3 -2 -2

For loop:

In case of strings

Like in c u need not to increment or decrement the for loop in python it does on its own. lets say for example

for i in "hansi":

    print(i)

hansi is iterable

Iterable is **an object which can be looped over or iterated over with the help of a for loop**. Objects like lists, tuples, sets, dictionaries, strings, etc. are called iterables

here first I represents to h then a the n then s then I at last

in case of lists:

hansi=["1","2","3"]

for i in hansi:

    print(i)

hansi is iterable

Iterable is **an object which can be looped over or iterated over with the help of a for loop**. Objects like lists, tuples, sets, dictionaries, strings, etc. are called iterables

see her I first represents to 1 then to 2 then to 3 .

format()

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

Input:

age = 36

txt = "My name is John, and I am {}"

print(txt.format(age))

output:

My name is John, and I am 36

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

quantity = 3

itemno = 567

price = 49.95

myorder = "I want {} pieces of item {} for {} dollars."

print(myorder.format(quantity, itemno, price)

### **Example**

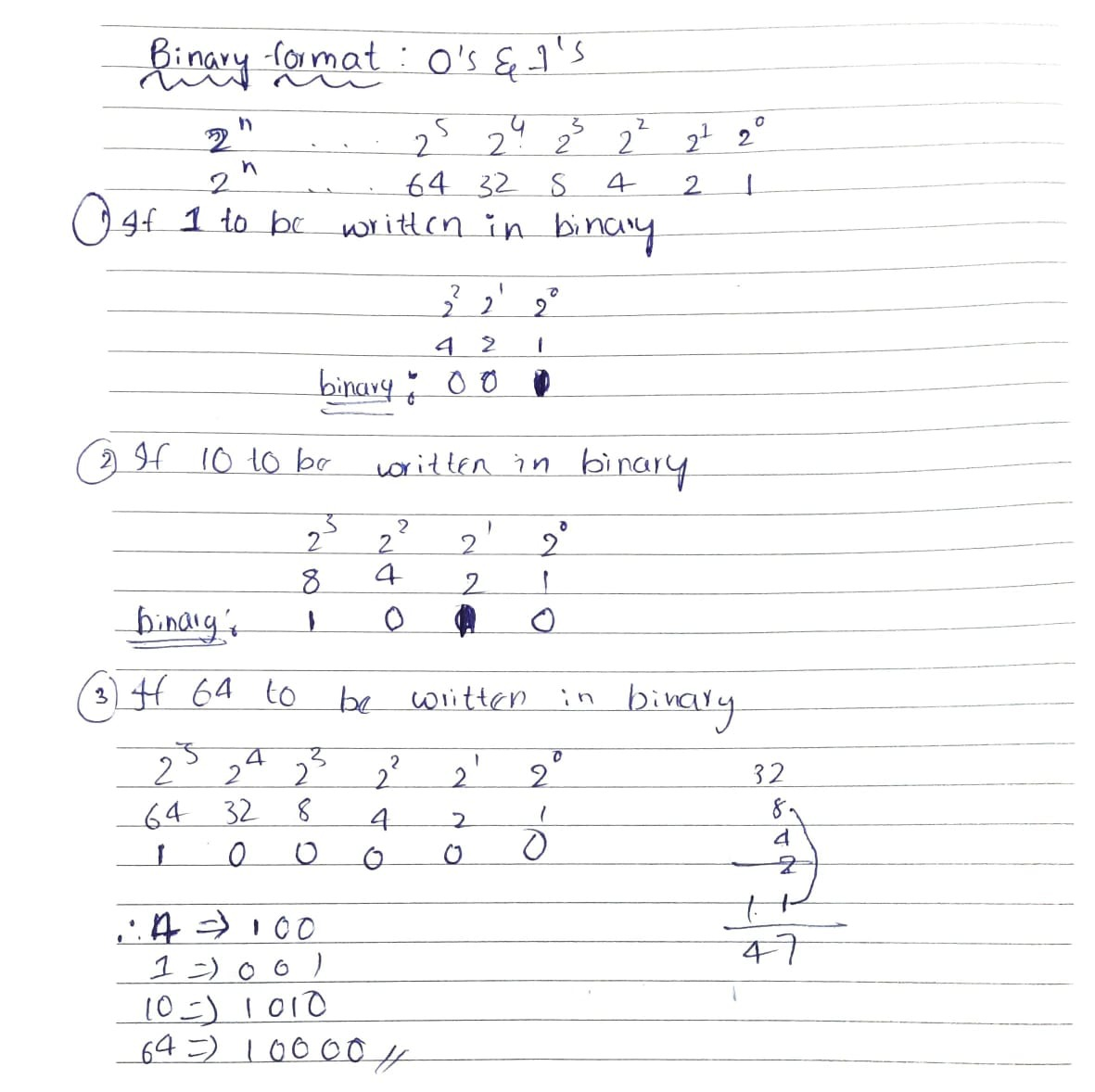
|  |
| --- |
| quantity = 3 itemno = 567 price = 49.95 myorder = "I want to pay {2} dollars for {0} pieces of item {1}." print(myorder.format(quantity, itemno, price)) |

Output:

I want to pay 49.95 dollars for 3 pieces of item 567

Dout: &=, |=,link: <https://www.w3schools.com/python/python_operators.asp> bitwise operators

BITWISE OPERATORS



&(AND OPERATOR):

**Bitwise AND operator:** Returns 1 if both the bits are 1 else 0.

EX1:

|  |
| --- |
| x=4(4 in binary format is 100)  y=10(10 in binary format is 1010)  0100  1010  By &:0000(bcz in any place both bits are not 1)  So answers is 0 |

EX:2

|  |
| --- |
| X= 2(0010)  Y= 6(0110)  By &(0010) (bcz in 2^1 place both bits are 1)  So the answer is 2 |

Coding ex 1:

x = 4

y = 10

print("x&y is ", x & y)

output:

x & y is 0

coding example 2:

x = 2

y = 6

print("x&y is ", x & y)

output:

x & y is 2

| (or operator):

**Bitwise or operator:** Returns 1 if either of the bit is 1 else 0.  
**Example:**

|  |
| --- |
| a = 10 = 1010 (Binary)  b = 4 = 0100 (Binary)  a | b = 1010  |  0100  = 1110  = 14 (Decimal) |

~ (NOT OPERATOR)

**Bitwise not operator:** Returns one’s complement of the number.

1)a = 10 = 1010 (Binary)

~a = ~1010

= -(1010 + 1)

= -(10+1)

= -11 (Decimal)

2)b = 1 = 0001 (binary)

~b = ~0001

= -(0001 + 1)

= -(1+1)

= -2 (Decimal)

Coding:

Input:

|  |
| --- |
| x = 1  print("~1 is ", ~x)  y = 10  print("~10 is ", ~y) |

Output:

|  |
| --- |
| ~1 is  -2  ~10 is  -11 |

**^(** **xor operator)**

**Bitwise xor operator:** Returns 1 if one of the bits is 1 and the other is 0 else returns false.  
**Example:**

|  |
| --- |
| a = 10 = 1010 (Binary)  b = 4 = 0100 (Binary)  a ^ b = 1010  ^  0100  = 1110  = 14 (Decimal) |

Coding:

Input

|  |
| --- |
| k = 10  l = 4  print("10 ^ 4 is ", k ^ l) |

Output:

|  |
| --- |
| 10 ^ 4 is 14 |

Table

Description automatically generated

A picture containing text, whiteboard

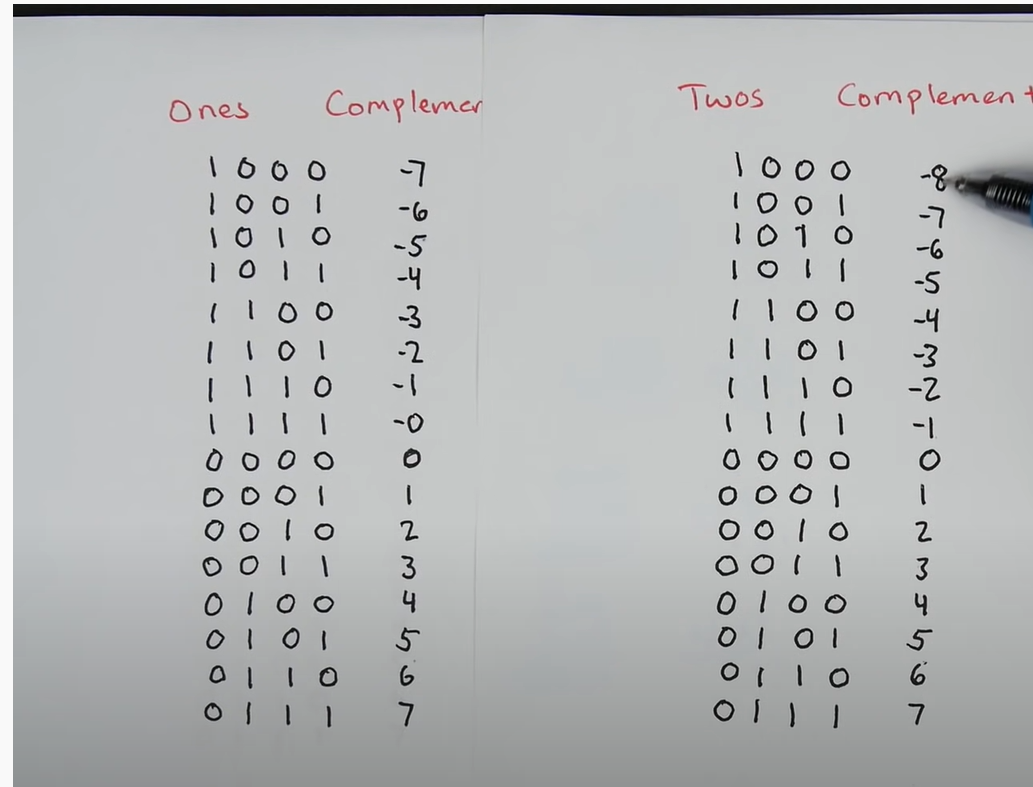
Description automatically generated

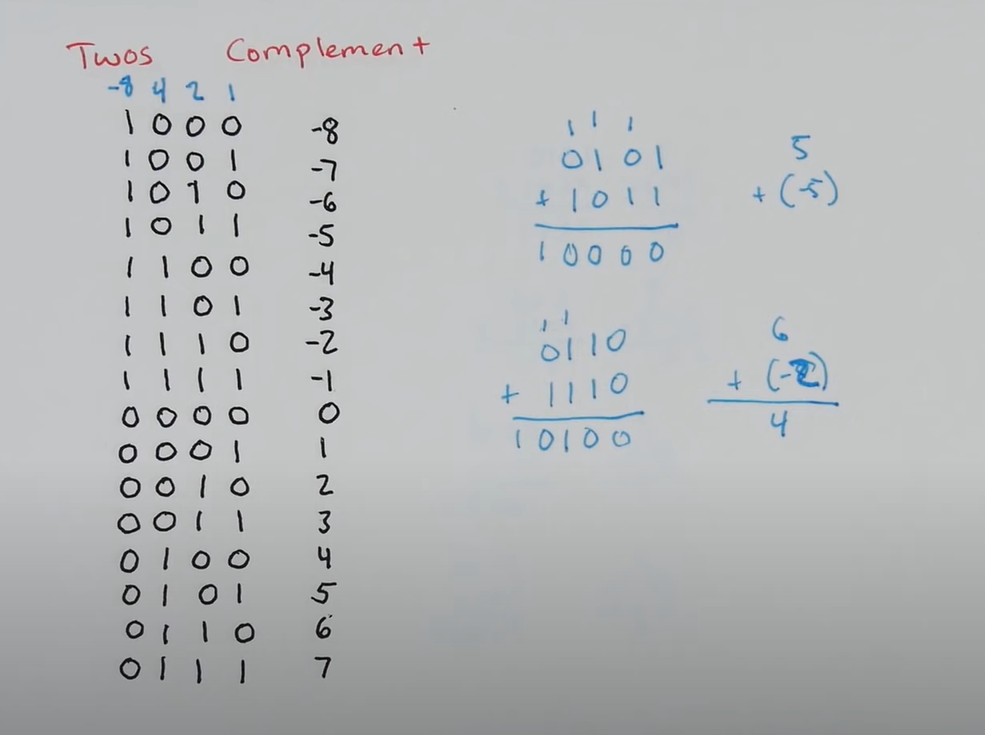
1)See here in above pic in 1st blue one addition of binary positive 5 and negative 5 number gave answer as 1111 which is -0 if we see here

2)so if we observe below(i.,e. -0+1 level down which is 0) -0 it is 0 which is exact answer

3)similarly if we see in 2n ,3rd,4th addition we could see exact answer of we go one level up see in 3rd binary addition it is 1 but exact answer(5-3 is 2) which we can get by going one level up in positive numbers

For explanation: <https://www.youtube.com/watch?v=4qH4unVtJkE&ab_channel=BenEater>





In twos complement mathematics works out.dont consider 16s place just consider (0000 is 1st case and 0100 in 2nd case)

NOTE: HERE the two complement will be the binary number of a negative number for ex

**Bitwise right shift:** Shifts the bits of the number to the right and fills 0 on voids left( fills 1 in the case of a negative number) as a result. Similar effect as of dividing the number with some power of two.

Example 1:

a = -10 = 1111 0110 (Binary) (binary of 9 is 1001 opp 0110 whisch is twos complement of -10)

a >> 1 = 1111 1011 = -5

**Bitwise left shift:** Shifts the bits of the number to the left and fills 0 on voids right as a result. Similar effect as of multiplying the number with some power of two.  
**Example:**

Example 1:

a = 5 = 0000 0101 (Binary)

a << 1 = 0000 1010 = 10

a << 2 = 0001 0100 = 20

Example 2:

b = -10 = 1111 0110 (Binary)

b << 1 = 1110 1100 = -20

b << 2 = 1101 1000 = -40

BINARY ADDITION:

Text, letter

Description automatically generated

FINDING NEGATIVE NUMBER BY ITS BINARY NUMBER:

A piece of paper with writing

Description automatically generated with medium confidence

NOTE:((VIP))

\*Set *items* are unchangeable, but you can remove and/or add items whenever you like.

\*\*As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

SOME BASICS:

X:Y

PRINT(A[X:Y])🡺 HERE STARTS FROM X INCLUDES X AND ENDS BEFORE Y EXCLUDES Y (Y IS NOT INCLUDED)

X:

STARTS AT X AND ENDS AT LAST ELEMENT LAST ELEMENT IS ALSO INCLUDED

:Y

STARTS FROM STARTING(0 IF +VE INDEXING -1 IF -VE INDEXING)AND ENDS BEFORE Y(Y IS NOT INCLUDED)

[ : ]

STARTS FROM STARTING(0 IF +VE INDEXING -1 IF -VE INDEXING) AND ENDS AT LAST ELEMENT(LAST ELEMENT IS INCLUDED)

+=

The **addition assignment** operator (+=) adds the value of the right operand to a variable and assigns the result to the variable.

LIST METHODS

ADDING:

INSERT 🡺 INSERTS ELE AT SPECIFIC INDEX BY MOVING ELE FROM THAT INDEX TO +1 I INDEX WITHOUT REPLACING ANY ELEMENT.

## APPEND🡺 To add an item to the end of the list, use the append() method:

EXTEND 🡺 To append elements from another list to the current list, use the extend() method. The extend() method does not have to append lists, you can add any iterable object (tuples, sets, dictionaries etc.).

REMOVING:

REMOVE🡺 The remove() method removes the specified item(VALUE)

EX: thislist.remove("banana").

POP 🡺 The pop() method removes the specified index .EX:  thislist.pop(1) #ele at 1 is del

. pop() with no index value deletes whole list

DEL 🡺 The del keyword also removes the specified index EX: del thislist[0]

. ele at index 0 is deleted.

. as in pop del with no index like del thislist deletes whole list

CLEAR 🡺 The clear()method empties the list.The list still remains,but it has no content.

Ex: thislist.clear()

## list.append(elmnt)

## list.clear()

## list.copy()

## list.count(value) The value to search for.

## list.extend(iterable) Any iterable (list, set, tuple, etc.)

## list.index(elmnt) The element to search for

## list.insert(pos, elmnt) pos:A number specifying in which position to insert the value

## elmnt: An element of any type (string, number, object etc.)

## list.pop(pos) A number specifying the position of the element you want to remove, default value is -1 , which returns the last item as in stack top of c the top element and also we used to initialize pop=-1 in c

## list.remove(elmnt)

## list.reverse()

## list.sort(reverse=True|False, key=myFunc)

## 

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| reverse | Optional. reverse=True will sort the list descending. Default is reverse=False |
| key | Optional. A function to specify the sorting criteria(s) |

## 

## List Comprehension

Links:1) https://www.w3schools.com/python/python\_lists\_comprehension.asp

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Syntax: newlist = [expression for item in iterable if condition == True]

* In list comprehension what we does is in [] brackets we writes the code like what to be added first here in below ex element in num to be added so as we used x as iteration variable so x then loops and then condition the thing is as it is without enter space in sequence how we write normal format other than list comprehension.

Ex:

num=[1,2,3,4]

newlist=[x for x in num]

print(newlist)

>>[1,2,3,4]

Example:

A short hand for loop that will print all items in a list:

Input:

list1 = ["apple", "banana", "cherry"]

[print(x) for x in list1]

Output:

apple

banana

cherry

note:

The expression can also contain conditions, not like a filter, but as a way to manipulate the outcome:

|  |
| --- |
| fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  newlist = [x if x != "banana" else "orange" for x in fruits]  print(newlist)  >> ['apple', 'orange', 'cherry', 'kiwi', 'mango'] |

But if u see below:

|  |
| --- |
| fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  newlist = [x for x in fruits if x != "apple"]  print(newlist) |

['banana', 'cherry', 'kiwi', 'mango'

See here in list comprehension if u used to write like

newlist = [x if x != "apple" for x in fruits ]

it shows error as the if condition act as filter and it is not followed by else where we declares what to be printed if condition is false .

## Customize Sort Function

You can also customize your own function by using the keyword argument key = function.

The function will return a number that will be used to sort the list (the lowest number first):

def myfunc(*i*):

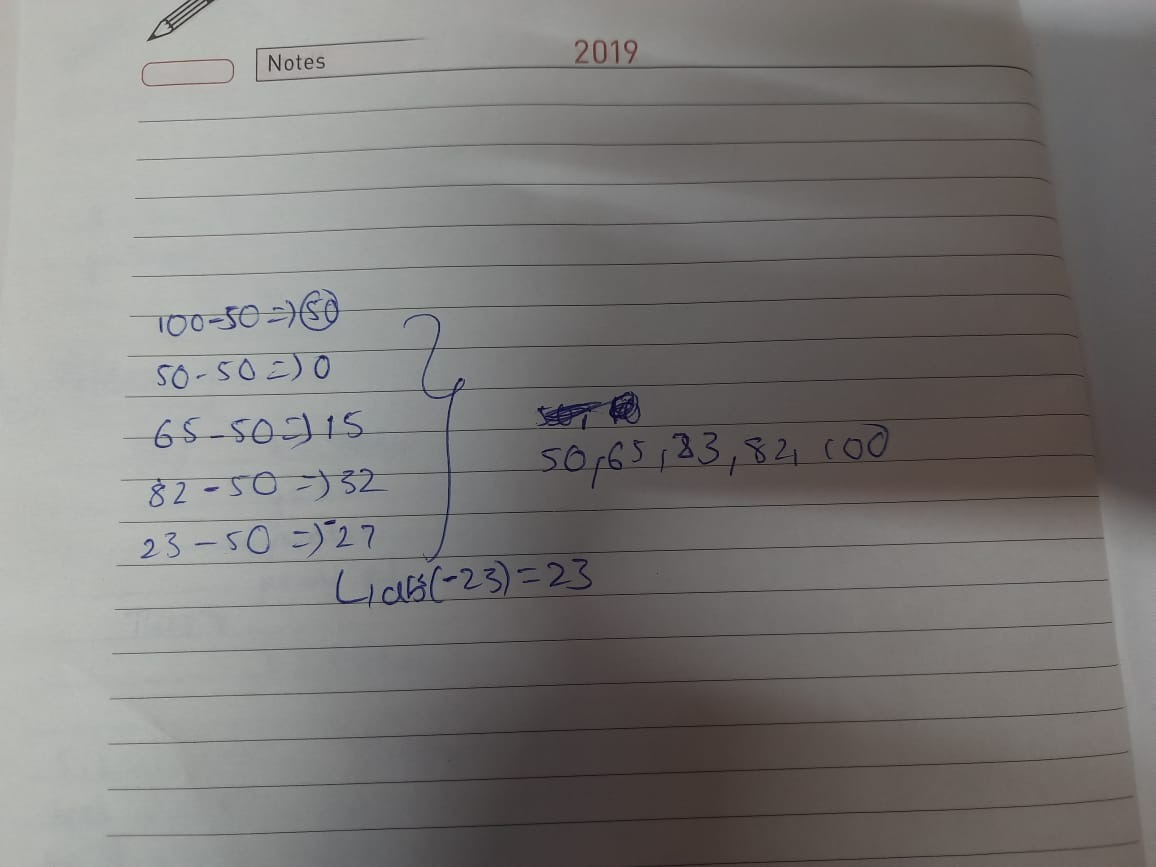
    return abs(*i* - 50)

thislist = [100, 50, 65, 82, 23]

thislist.sort(*key*=myfunc)

print(thislist)

>> [50, 65, 23, 82, 100]



TUPLE:

|  |
| --- |
| thistuple = ("apple",)  print(type(thistuple))  #NOT a tuple  thistuple = ("apple")  print(type(thistuple)) |

OUTPUT:

|  |
| --- |
| <class 'tuple'> <class 'str'> |

SETS:

In sets if u use

update() :u can just use the syntax which is like variable1.update(variable2) here the variabe1 will be updated with contents of var1 and var2

functionotherthanupdate: If u use u need to assign that function to some variable and call it whenever u want

DICTIONARIES:

Looping through the dictionary

Input:

|  |
| --- |
| thisdict = {  "brand": "Ford",  "model": "Mustang",  "year": 1964  }  for x, y in thisdict.items():  print(x, y) |

Output:

|  |
| --- |
| ​  brand Ford model Mustang year 1964 |

If-else:

Tenery operators (or) conditional expressions:

Ternary operators are also known as conditional expressions are **operators that evaluate something based on a condition being true or false**. It was added to Python in version 2.5. It simply allows testing a condition in a single line replacing the multiline if-else making the code compact

**Syntax :**

[on\_true] if [expression] else [on\_false]

Ex:

|  |
| --- |
| a=33  a=23  print(a,”a is elder”) if a>b else print(b,”is elder”) |

While loop:

|  |
| --- |
| i = 1 while i < 6:   print(i)   i += 1 |

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.(i=1)

Continue statement is while loop:

With the continue statement we can stop the current iteration, and continue with the next:

Input:

|  |
| --- |
| i = 0 while i < 6:   i += 1   if i == 3:     continue   print(i) |

Output:

|  |
| --- |
| 1 2 4 5 6 |

Note:

The else block will NOT be executed if the loop is stopped by a break statement.