KEYWORDS

* Special words which are reserved and have special meaning.
* In python, Keywords are case Sensitive.
* As of python 3.10 there are 35 Keywords
* we cannot assign any value to Keywords
* To get all the Keywords 1. help ("Keywords")
  2. import Keyword.
     Keyword.kwlist
* TO check wheather a string is Keyword or not. Keyword.iskeyword("string")
  ↳ Special Keywords -> are used values to variables.
  false, true, none

Identifiers example:

```
>>> "a".isidentifier()
>>> help("keywords") -> only works in IDLE shell
>>> import keywords
>>> Keyword.iskeyword("True")
True
```

Variables.

* A Variable represents an entity whose value can change as and when required.
* It is the name given to a memory location which holds the actual value.
* A Variable can hold objects of different types
* All Variable Should be in lower case as there are more than one word in the variable.
* Then user seperates with Underscore and this is Python Convection

while importing compiler will be print till execution

IDENTIFIER → name given to program

constants
class name
from names (method name)

→ User defined names to represent a variable function from class, module [file name] (or) any other object i.e if you assign some name to programable entity in python then it is nothing but technically called as identifier.

* To check whether a python identifier is valid or not "String".isidentifier().

Rules to create Identifier

* Identifiers name can start with alphabets or underscore(_)
* It can contain alpha-numerals but should not start with numbers
* Special Symbols [@ # $, % . ! &, * , white space] are not allowed except underscore(_)
* Keywords cannot be used as variables.
* Python Doc says that you can have an identifiers with unlimited length [max 79 characters]
* PEP8 [Python Enhancement proposal]: Document that provides guidelines and best practice on how to write python code
  → follow the convention while writing a code
  hints → if you dont follow it doesn't throw error.

Checking identifiers

→ "a".isidentifier()
→ True.

"ab".isidentifier()
→ false

Brief History of Python

* Python was developed by Guido Van Rossum
* It was implemented in 1989

Python Overview

* Python is very simple programming language
* It is very powerful object oriented programming language
* It is widely used general-purpose, high level programming language
* It is an interpreter language
* It Supports Dynamically memory allocation

Features of Python

* Readable Syntax
* Easy to learn
* Free Open Source
* Cross platform
* Large standard library
  - Becoz all variables are inbuilt libraries

[Translators :- Compilation
↓
interprets → line by line]

→ It initialisation in the next line whenever defines a variable of the datatype binding implicitly.

a=10 → binds to integer D-T
a='Hi' → it works in python
↓ binds to string Datatype

variable acts as pointer memory location changes.

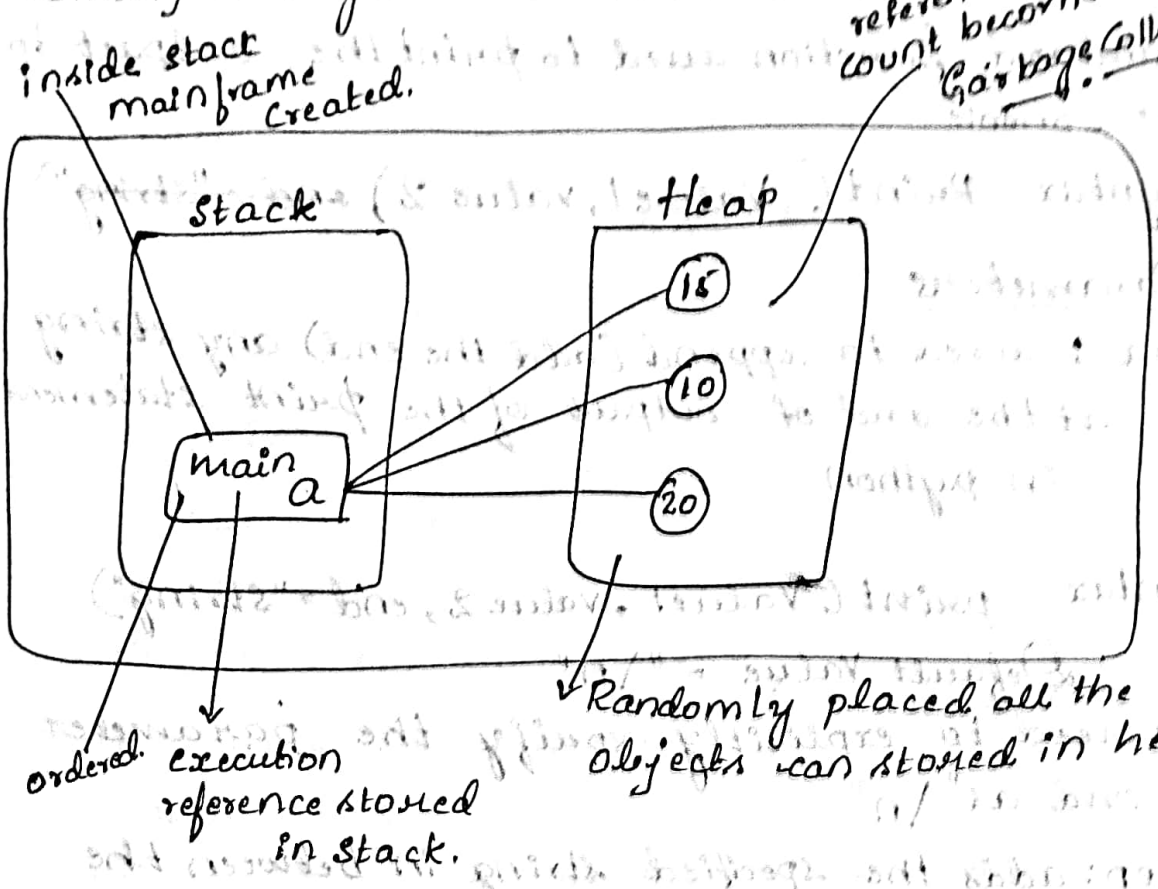
COMPILATION

Sourcecode → Compile → ByteCode .py
.py Cpython

in Cpython

ByteCode → Interpreter → Req
.Pyc PVM O/P

Memory management in variable.



Refernce Count :- no. of reference pointing to a object.

Garbage Collection :- when a reference count become zero Garbage collection comes to picture and

IN IDLE we just call Variable int enter a it will display '10'
but in other edition we use print &tack.

Ex) name = "John"
age = 32
Print ("My name is", name)
Print ("My age is", age)

To print seperate but in single line.
we use parameter we use (end = '\n')
we can override '\n' parameter to anything
They will default Seperate by using space
'Sep' - 'space'
we can overrid space

Rules to create a Variable.

* These are some rules need to follow where giving a name for a python variable

Rule 1 :- You should start variable name with an alphabet (or) Underscore (_)

Rule 2 :- A variable name can only contain A-Z a-z , 0-9 and underscore (_)

Rule 3 :- You cannot start the variable name with the number.

Rule 4 :- You cannot use Special characters with variables names such as $ # , * @ , $, 1

Rule 5 :- Variable names are case Sensitive for example abc & ABC are two different variables.

Rule 6 :- Do not use reserved Keyword as a variable name for example Keywords like class , for def , del , is else , try , from etc.

Creating a Variable.

a = 10 id → inbuilt function address to Call address of 'a' a=10 (or) Variable

id (a) → o/p → 2174080909870

b = 10
id (b)

whenever i have 2 or more Variable for same Value It will print the same location

Syntax.

Variable name = Value.

eg :- a = 10
x = a → # id(x) == id(a)

a = 15
x = a

id() : returns the memory location of an object

Print("I am", b, "years old")

o/p

a = "Ram"

b = 10

Print ("my name is", a, sep = "$$", end = '\n')

Print ("I am", b, "years old", sep = '$$')

* Print ("True", is identifies())

"True". isidentifier()

* *a = 10

Print(a) => o/p Syntax error.

* python we follow convention if we want to do as. Constant make it uppercase.

* note :- To Create a constant the variable name has to be taken (defined) assign all uppercase letters. This is just a convention which suggests that the created identifies is a Constant should not be changed at any cost through the program

Creating as Constant

Name = "Ram"

DATA TYPES

* Datatype represents the ~~data~~ types of data that is stored in a memory location.
* Variables can hold the values, and every value has a datatype
* Python is a dynamic type language, hence we do not need to define the type of the variable while declaring it [Dynamin memory allocation]
* The interpretor implicitly binds the value with its type
* type() : returns the type of the variable passed..

PRINT()

* standard function used to print the output to the console

Syntax Print(Value1, value 2) ~~end = "string"~~)

* Parametens

end :- used to append (add the end) any string at the end of output of the print statement in python

Syntax print (Value1, value 2, end = "string")

Default Value = "\n"

no need to explicitly specify the parameter end as "\n"

> Sep: adds the specified string in between the values to be printed.

Syntax: print ('values', sep = 'string')

eg print (1, 2, sep = '#') O/P 1#2

Builtin function:- we need not import we dont use import

IMPORT

INPUT()

* Takes the users input at the runtime.
* The input () always needs the input as a string even if it Comprises of digits.
* Syntax input() (or) input ("string")

eg:

```
name = input("Enter the name")
age = input("Enter the age")
Print ("my name is", name, end = '\n')
Print ("My name is, name, end = '\n')
```

round (3.625653, 2) → 3.63

```
>>> a = 50.7
>>> import math
>>> math.trunc(a)
    = 50
```

INBUILT FUNCTION.

* abs(integer) gives only the positive integer as the output even the expression results to negative number eg :- abs(-5) → 5
* round (num, [Precision]) - Round off the number to the ~~number~~ nearest value based on the next digit (if the digit is >=5), the precision no will be incremented by 1 or else it will be remain the same.
* Precision → Specifies how many decimal points need to be rounded off
* math.trunc() → truncates removes the decimal point from a floating point number.

COMPLEX Numbers : $j = \sqrt{-1}$

* written in the form of a+bj, where 'a' is the real part and 'b' is the imaginary part.
* It can be +ve and -ve.
* To convert integer into floating point the Syntax is

Variable name = Complex(Variable/value)

```
a = 2+3j
b = Complex(3+2j)  → 3+2j
c = Complex(5,3)   → 5+3j
```

whatever present in class it is called as attributes

dir() → will give all attributes of class. of any object.

a ---- abs_()
↳ tve to tve.
→ -ve to tve.

b = -4

b. -abs ---() o/p ④

Note:- Default value of integer is zero or empty
Constructor i,e int()

* Floor div : 3//2 | Add | division
* Exponent : 3**2 | Sub |
* modulus : 3 % 2 | multi |
* divmod(x,y) : Returns(x//2, x%y)
* abs() : converts negative numbers into positive

FLOAT

→ It specifies the data stored inside the memory location is of type Decimal

→ It can be tve (or) -ve

ex: x = 8.7
x = float(3.2) x = 6.3

Default value of float

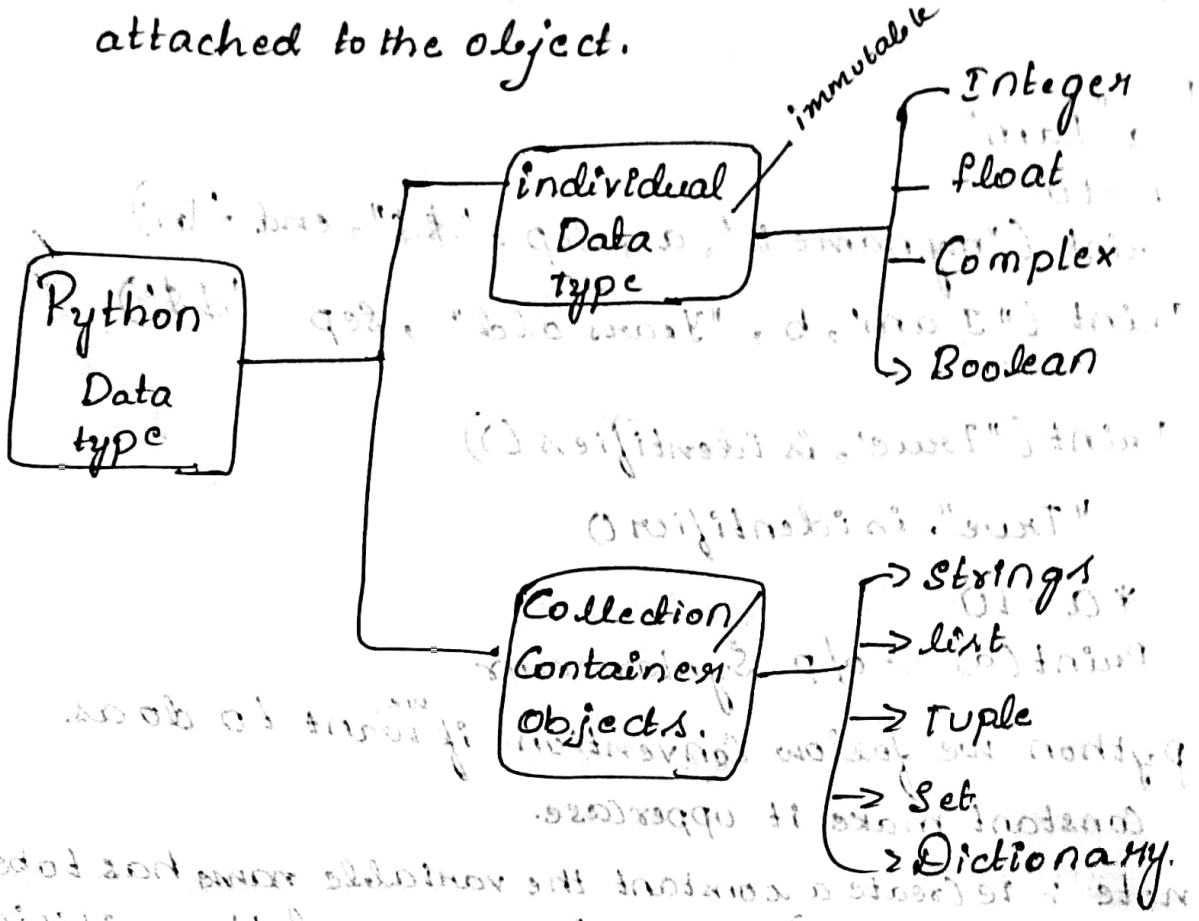
x = 0.0
x = float() → Empty.

Round of nearest value.

round()
a = 2.75
round(a) = 3%.

round(3.14) → 3
round(3.6) → 4

→ dir() : returns the list attributes that are attached to the object.



INTEGER

* It Specifies the data stored inside the memory location to be an integer.
* It can be positive (or) negative

Eg a=10

```
>>> a=int(10)
>>> a=15
```

whenever we are creating Data binding it will be implicitly.

To check what datatype we use type d word.

```
x=20
type(x)
<class 'int'>
a=int(84)
```

Creating a object classname, value, variable.

STRING FORMATING

* Infusing data dynamically into a string during the time of execution/Runtime is called formating

There are 3 ways of string formatting.

* Formatting with placeholders.
* Formatting with .format() Stringmethod
* Formatting with string literals, called f-string

→ FORMATING WITH PLACEHOLDERS:

* oldest method of string formatting
* Here we use the module '%' operator.
* The modulo % is also known as "string-formatting operators"
  - '%s' is used to inject string
  - %d is for integer
  - '%f' is for floating point values.

Eg

1. message = "hi my name is %s" % "John"
   >>> "hi my name is john"
2. message = "hi my name is %s and I am %d years old" % ("John", 25)
   >>> "hi my name is john and I am 25 years old"
3. Pi = "The value of Pi is %.4f" % (3.141592)
   >>> "The value of Pi is 3.1416

SLICING

* Process of extracting multiple characters at a time / simultaneously.

Note Elements at the end index will not be added

S = "hello"

-5	-4	-3	-2	-1
h	e	l	l	o
0	1	2	3	4

var [Startindex : Endindex+1 = Steprvalue]

default value ①

Positive → s[0:3] ⇒ hel

negative → s[-5:-2] → hel

Combination → s[0:-2] ⇒ hel

s[-5:3] ⇒ hel

a = "hello"

eg:- extract character present at even indices.

a[::2] / a[0:5:2] / a[0:len(a):2]

a=[-5:len(a):2] / a[-5::2]

S = "hello world"

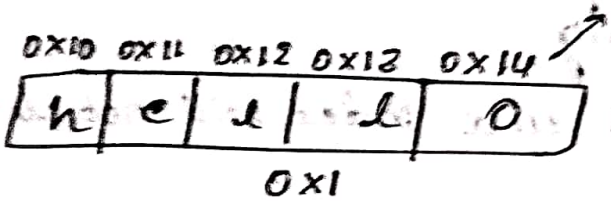
Reverse a string

S = "Hello world"

S = = S1[::-1]

Memory allocation in String

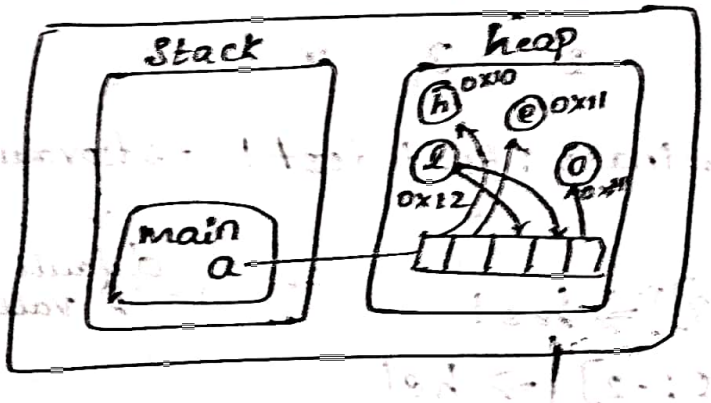
Eg :- a = 'hello'



→ It will be hexadecimal.

Del :- Keyword used for memory de-allocation.

RAM



INDEXING A STRING

* Process of extracting single character at a time.
* Indexing Can be positive (start with 0) or negative (start with -1)
* Syntax

var_name[index]

S = "hello"

0	1	2	3	4	length=5
h	e	l	l	o	
-5	-4	-3	-2	-1	

First = S[0] / S[-len(s)]

Last = S[len(s)-1] / S[-1]

RAW STRING

Whenever we are using slash / Backslash inside the string we will get different meaning.

Eg \t is taken as tab

\n is taken as nextline

\\ → Esc Sequence.

* Python raw string are the string literals prefixed with "r" or "R"
* Raw string donot treat backslashes as a part of an escape sequence. It will be printed normally string.
* Raw string Cannot take odd number of backslashes there must be a literal Succeding the backslash

Regular String V/S Raw String.

Regular string : considers backslash as a Special Sequence

eg. S = "hello \task from python world"

O/P : hello ask from python world.

Rawstring : considers entire string as character Set irrespective of any special Sequence

Eg : S = r"hello \task from python \n world"

O/P = "Hello \task from python \n world"

BOOLEAN: a = true

b = False

int(a) → 1

float(a) → 1.0

int(b) → 0

float(b) → 0.0

Complex(b) → (0j)

Complex(a) → (1+0j)

Default Value

integer → 0, int()

float → 0.0, float()

Complex → 0+0j, 0j, Complex

Boolean → False.

COLLECTION OF DATA TYPES.

STRINGS:- collection of characters.

Quotes (' ')

Creating a string

S = 'helloworld'

S1 = " hello world"

S2 = "' hello world"'

S3 = """ hello world""""

default values

S = ' ', " ", ''' ''', """ """

S1 = str() → Constructor

* Set of collection of character
* Boundry ' ' (or) " "
* It is an immutable data type.

* length of a string = len("string") or len(var_name)

SYNTAX Var_name = "String"

Zero length String (or) Empty string

word : '' (or) word = str()

TYPECASTING

converting from a variable or object of one data type into another datatype

INTEGER

a = 10

float(a) = 10.0

complex(a) = (10+0j)

bool(a) = True

float

a = 12.75 →

int(a) → 12

complex(a) → 12.75 + 0j)

bool(a) → true.

real part (or) imaginary part if you segregate this will be data

COMPLEX

a = 1 + 2j.

int(a) → type error

float(a) → type error

bool(a) → true.

in complex no both real and imaginary are equally importance

whenever we try to convert to integer data loss will occur we will remove whole we can not convert complete to integer.

when we mismatch of type of data (or) Signature (or) Callable.

To check expectation of any data

if it is default is false

if it has no default → True

When it will return false

bool(0j) → imaginary part is taken for complex no.

BOOLEAN

=> Useful in Condition expression

-> Boolean Variable are defined by the true and and false keywords.

>>> To Convert any value to boolean type the Syntax is

Variable name = bool (Values)

Eg: >> a=10

x= bool(a) → True

b=0.0

x= bool(b) → x= True

a= 87, b= 34.4

type(a) == type(b) → false

isinstance(a, float) → false

isinstance(1+2j, (int, float, complex))

true.

IS INSTANCE()

returns true if the specified object is of the specified type, otherwise false.

=> Syntax: isinstance(value, datatype)

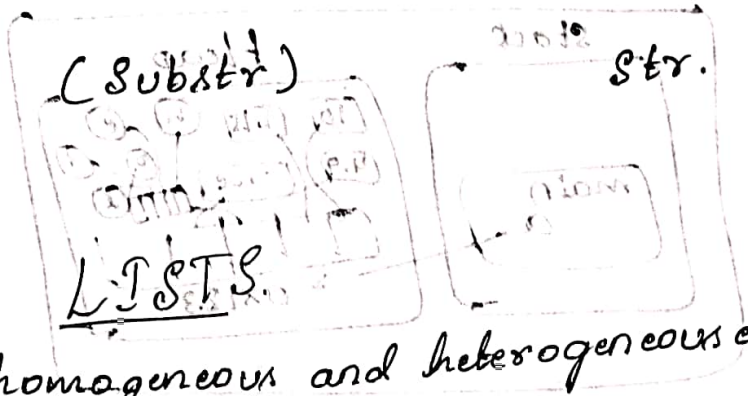
Eg: isinstance(5, int) → True

isinstance(10+8j, complex) → True

isinstance(10+y8), complex) → Error.

isinstance(10, (int, float, complex)) → True

METHOD	ARGUMENT	RETURN TYPE
upper()	–	str
lower()	–	str
swapCase()	–	str
count()	(substr, [SI, EI])	int
index(), rindex()	(substr, [SI, EI])	int
find, rfind	(substr, [SI, EI])	int
replace	(original, new, [count])	str
starts with	(substr, [SI, EI])	bool
ends with		
split / rsplit	([seperator, [max-split]])	list
join	(iterable)	str
strip / lstrip / rstrip.	(substr)	str.



* collection of homogeneous and heterogeneous elements.
* seperated by Comma.
* Elements in the lists are ordered.
* list can hold duplicate elements.
* It is a mutable type

- Boundary : [.....]
- Syntax : var_name : [ele1, ele2, ele3...]
- Length of a list : len(var_name)

lstrip():

* Remove any leading characters.
* Syntax : string.lstrip(characters)

eg

my_string = '*** *** Hello world ======'

print(my_string.rstrip('='))>>> *** *** hello world

print(my_string.lstrip('*'))>>> hellow world======

print(my_string.strip('*='))>>> hello world

isalnum(): Returns true if all characters in the string are alphanumeric

Isalpha(): Return true if all characters in the string are in the alphabet

isdigit(): Returns true if all characters are digits.

islower(): Returns true if all alphabets in the string are lowercase.

Isupper(): Returns true if all alphabets in the string are uppercase

isspace(): Returns true if all characters in the string are spaces.

Note : All string method returns new values they do not change the original string.

To convert list into string we join()

Split = String to list

join = list to string

JOIN

* join the elements of an iterable using the string specified.
* Syntax : String.join(iterable)
* Converting other data type into string.

Eg

data = "hai"

data.join(["hello", "world"])

>>> "hellohaiworld"

message = "hello"

"_".join(message) >>> "h_e_l_l_o"

",".join(message) >>> "h,e,l,l,o"

Strip(), lstrip(), rstrip()

Strip(): * remove any leading (space at the begining) and trialing (space at the end) characters (Space is default leading character to remove)

* Syntax : String.strip(characters)

rstrip():

* removes any trailing characters.
* Syntax : string.rstrip(characters)

Endswith()

>Returns true if the string ends with the specified value.

* Syntax : string.endswith(value, [start], [end])

eg : "howare you".endswith("You") >>> True

SPLIT()

* Splits the string at the specified seperator and returns a list.
* Syntax : String.split([seperator], [max split])
* rsplit() method splits a string into a list, starting from the right
* if no "max" is specified, this method will return the same as the split()
* Converts a string into a list.

Eg:

'This is my string'.splits('s'))

>>> ['Thi', ' i', 'my', 'tring']

"This is my string".split()

>>> ['This', 'is', 'my', 'string']

"This is my string".split(" ", 2)

>>> ['This', 'is', 'my string']

"This is my string".rsplit(" ", 2)

>>> ['This is', 'my', 'string']

INDEX() & rindex()

* Searches the string for a specified value and returns the position of where it was found.
* finds the first occurance of the specified value.
* Returns value error if the value is not found.
* Syntax: string.index(value, [start].[end])
* rindex(): Searches the string for a specified value and returns the last position of where it was found.

Eg my_message = "hello world"

1. print(my_message.index('d')) >>> 2 first occurance
2. print(my_message.rindex('l')) >>> 9 last occurance
3. print(my_message.index('universe')) >>> Value error

REPLACE()

* Replace a specified phrase with another specified phrases.
* Syntax: string.replace(old values, new value, [counts])

eg:-
* "malayalam".replace("a","q") >> mqlqyqlqm
* "Hello world".replace("world","universe")
>>> "Hello Universe"
* "malayalam".replace("a","q",1)
>>> "mqlayalam"

Startswith() and Endswith()

* Startswith() Return true if the string starts with
* the specified value.
* Syntax: string.startswith(values,[start],[end])
* eg:- "how are you".started with("are") >>> false

Print(my_message.count('l')) # it prints the occurances of the letter 'l'.

>>> 5

Print(my_message.count("hello", 0, 10))

prints the no of occurances of the word

>>> 1

Error: Index will throw value Error if the character is not present find (or) index perform same job.

both will provide position.

find will return '-1' if the character is not present

index (Reverse Index) → from last occurance)

S.rindex("l")

r.find() → last Occurance.

FIND(), r find(). → -1

* Searches the string for a specified value returns the position of where it was found.
* finds the first occurance of the specified value
* returns -1 if the value is not found.
* Syntax: String.find(value, [start], [end])
* r find(): Searches the string for a specified value and returns the last position of where it was found.

eg:- my_message = "hello world"

1. print(my_message.find('l'))
2. print(my_message.r find('l'))
3. Print("my_message.find('universe'))
4. Print("today is beautiful day".r find("day"))
5. print("today is beautiful day".find("day"))

Merging string:

st = "hai"
st1 = "hello"
>>> st + st1 ⇒ "haihello"

methods

LOWER(), UPPER(), SWAPCASE()

1) LOWER():
* converts a string into lowercase
* Syntax: string.lower()
* eg: "Hai WORLD".lower() >>> "hai world"

2) LOWER() UPPER()
* converts a string into upper case
* Syntax: string.upper()
* eg: "hai world".upper() >>> "HAI WORLD"

3) SWAPCASE()
* Converts a lower case char into upper case and vice versa
* Syntax: String.SwapCase()
* Eg: "HaiWORLD".SwapCase() >>> "hAi iworld"

COUNT
* Returns the number of occurrences of a substring in original string.
* Syntax: string.count("substring", st, en)

Eg: my_village = "hellowoorld, hello universe"

2. Formating with the .format() string method

* formating is done by calling .format() method.
* Here placeholders ({}) are used to infuse the data at the runtime.

Eg

1. message = "hi my name is {} and I am {} years old"
   .format("John", 20)
   >>> "hi my name is john and I am 20 years old")
2. message = "hi my name is {1} and I am {0} years old".format
   >> "hi my name is John and I am 25 years old." ("John", 25)
3. menage = "hi my name is {name} and I am {age}
   years old.".format(age=25, name="John")
   >>> "hi my name is john and I am 25 years old"

3. FORMATING with String literals, Called f-string

* Here we can use outside variables directly into the string instead of passing them as arguments
* Here placeholders ({Variable}) are used to infuse the data at the runtime.

Example

1. a = 24
   b = "day"
   msg = f "There are {a} hours in a {b}"
   >>> "Thereare 24 hours in a day"
   String = f "one {b} has {a*60} minutes"
   >>> One day has 1440 minutes'

INDEX ()

Returns the index of the first element with the Specified value.

Syntax : list.index(element)

Eg. names = ['apple', 'google', 'Yahoo', 'amazon']

names.index("facebook") >>> valueerror

names.index("Yahoo") >>> 2

COUNT ()

Returns the number of occurances of the specified value.

Syntax : list.count(element)

Eg name=['apple', 'google', 'yahoo', 'amazon', 'facebook']

name.count("flipkart") >>> 0

names.count("apple") >>> 1

converting a list into string and vice versa

1. S = "hello"

list(S) >>> ['h','e','l','l','o']

S.split() >>> ["hello"]

"".join(["hello"]) >>> "hello"

"".join(['h','e','l','l','o']) >>> "hello"

2. S = "hello world"

list(S) >>> ['h','e','l','l',' ','w','o','r','l','d']

S.split() >>> ["hello" "world"]

" ".join["hello", "world"]) >>> "hello world"

" ".join(['h','e','l','o',' ','w','o','r','l','d']) >>> "hello world"

* deep copy() is a method is copy module, so it needs to be imported.

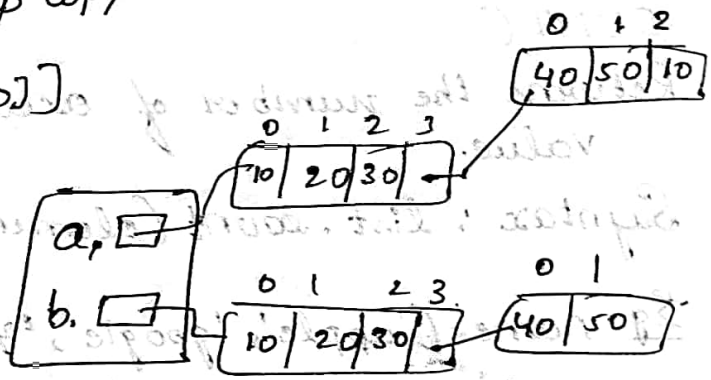
* Syntax

```
from copy import deepcopy()
a = [1, 2, 3, [4, 5]]
b = deepcopy [a]
```

Here both a and b will be having different memory locations even for the nested list.

Ex.

```
from copy imports deep copy
a1 = [10, 20, 30, [40, 50]]
b1 = deep copy [a1]
a1[-1].append(10)
```



SORTING LISTS

SORT():

* Sort the list ()
* In order to sort, a list should be homogeneous.
* Modifies the original list itself -> returns None
* Syntax : list.sort([key = function].[reverse = True])

Eg. names = ['apple', 'google', 'update', 'amazon', 'facebook', 'instagram', 'microsoft']

* names.sort() - it sorts the list in alphabetical order it sort method modifies the list inplace.
* names.sort(revers = true) it sort the list in descending order.
* names.sort(key = len) it sort the list based on length of the elements.

Eg

```
a = [1, 2, 3, 4]
b = a.copy()
    # or.
b = a[:]
```

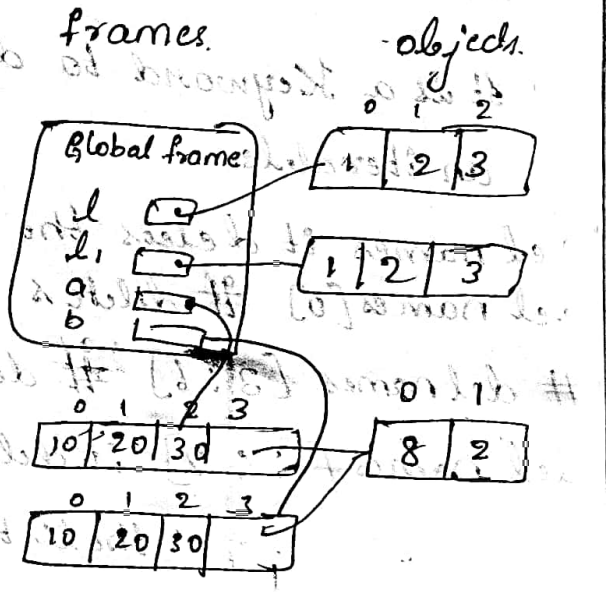
SHALLOW COPY:

IN This method, the elements of the main list will be copied to the new list as it is and the memory location will be different unless the main list has a nested list.

* If the main list has a method list then the same nested list will be shared along both the list
* i,e the modification done on the nested list will appear in both the original as well as copied list.

ex:-

```
l = [1, 2, 3]
l1 = l[:]
a = [10, 20, 30 [3, 2]]
b = a[:]
# b = a.copy()
a[-1] = .append(3)
```



DEEP COPY ©

DeepCopy In this method the elements of the main list will be copied to the new list

* i,e the modification done on the nested list will not get reflected in the other list

POP() :- Remove the first element at the specified position

* Syntax : list.pop[(POS)]
* By default POP() removes and returns the last element in the list.
* Returns removed value.
* If the index specified is not present -> IndexError

Clear() :

It is used to clear the entire list with out deleting the list.

* It returns an empty list when we check.
* Syntax : list..clear()

De-allocation @@ Memory Allocation of a list.

Del :- It is a Keyword to deallocate the memory in an iterable.

* del names, it delets the list from the memory.
* del names[0] # delete s 0th item in the list.
* # del names [3:6] # deletes 3rd, 4th, 5th item in the list.
* del names [::2] ; it deletes alternate items in the list.

COPY() - SHALLOW COPY.

COPY()

* Returns a copy of the list
* Syntax = list_var.copy()

METHODS USED IN LISTS

ADDING ELEMENTS TO A LIST

1) Append ():

* Add an element at the end of the position list (both individual and Collection)
* Syntax : list.append (element)
* Element can be of any datatype
* eg: [1,2], append ([3,4])
  o/p : [1, 2[3,4]]

EXTEND():

* extends the existing list with the items of the given sequence
* Syntax : list.extend (iterable)
* eg : [1, 2], extend([3,4]) o/p : [1, 2, 3, 4]

INSERT():

* Adds an element at the specified position (both individual + collection)

Syntax : list.insert (position, element)

Removing elements from list - remove(), POPO, clear()

1) remove(): Removes the first occurance of the element specified.
* Syntax: list.remove (element)
* If the value is not present → Value error

SLICING A LIST

* process of extracting a multiple elements at a time / simultaneously
* Syntax : var_name [SI : EI : SV]

elements at the end index will not be added in the slice.

ex $l = [10, 1.25, "hello", "Python", 49]$

extract 1st two elements.

$l[0:2]$ or $l[:2]$ or $l[0:-3]$ or $l[:-3]$ or $l[-5:-3]$

Merging two lists.

$l_1 = [1, 2, 3]$
$l_2 = [4, 5, 6]$

```
>>> l1+l2
>>> [*l1, *l2]
```

MUTABILITY and IMMUTABILITY

* Mutable Data type:
* Allows modification on the original objects.

eg. lists, sets, dictionary.

* Immutable data types.
* Doesn't allow modification on the original object
* even if the modification is done, there will be a new object created keeping the original object intact.

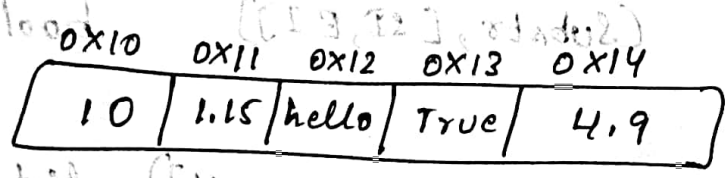
Eg: tuple, string, integers, float, Complex, boolean.

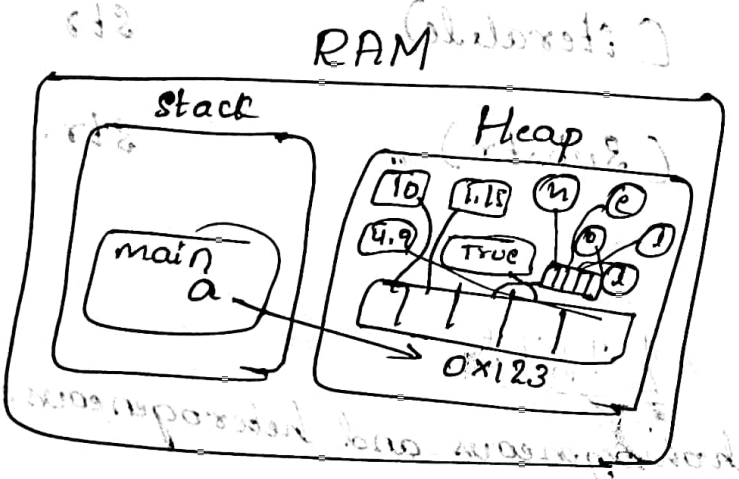
Different ways to Construct a list object.

* my_list = []
- my_list = [1, 2, 3, 4]
- my_list = [list() # using list_constructor
- my_list = list([1, 2, 3])

Memory allocation in lists.

eg a = [10, 1.15, "hello", True, 4-9]





INDEXING A LIST

* Process of extracting single element at a time.
* Indexing can be start with 0 (positive indexing)
  or starts with -1 (negative indexing)

Syntax : var_name[index]

Ex:- a = [1, 1+2j, 3.0, "String", [1, 2, "hello"], True]

names=["apple", "google", "yahoo", "amazon", "facebook",
"instagram", "microsoft"]

1) what is the output of.

a. names.append (['netflix', 'vishalmart', 'Kroger'])

names=["apple", "google", "yahoo", amazon", "facebook", "instagram",
"microsoft", ['netflix', vishalmart

b. names.extend ['netflix', 'vishalmart', 'Kroger'])

names = ["apple", "google", "yahoo", "amazon", "fb", "insta", "ms"
"netflix", vishalmart", "Kroger"]

* What is the difference b/w pop and remove.

POP is used to remove last element of list or by index remove is used to remove by using value/elem

3) Sort the above list in descending order

names.sort (reverse = true)

4) Sort above list in ascending order according their length. name.sort (Key = len)

5) Sort above list in descending order according their length

name.sort (Key = len, reverse = true)

6) String = "hai welcome to python"

a. Create a list with each character of the string as the elements.

list ( ) / list (string)

b. Create a list with each word of the String as the element

String.split()

TUPLE METHODS

COUNT() Return the no. of times a specified value occurs in a tuple.

* Syntax : tuple.count(value)
* eg. a = (12, 4, 7, 30, 9, 1, 4, 7)
  a.count(4) >>> 2//.

Index() return the first occurance of the element specified.

* raises value error if the value is not found
* Syntax : a = (12, 4, 7, 30, 9, 1, 4, 7)
  a.index(4) >>> 1//.

Method.	arguments	Return type
append()	(ind/coll)	None
extend()	(coll)	None
insert()	(POS, ind/coll)	None
POP()	([POS])	removed element
remove()	(element)	None
clear()		None
copy()	None	list
Sort()	([key, reverse])	None
index()	(element)	int
count()	(element)	int

TUPLES

* Collection of homogeneous or heterogeneous element.
* Seperated by Comma
* boundary : (.....)
* Immutable type
* Syntax : var_name = (ele1, ele2, ele3)
* Indexing and slicing is similar to that of string and list.
* length of tuple : len(var_name)

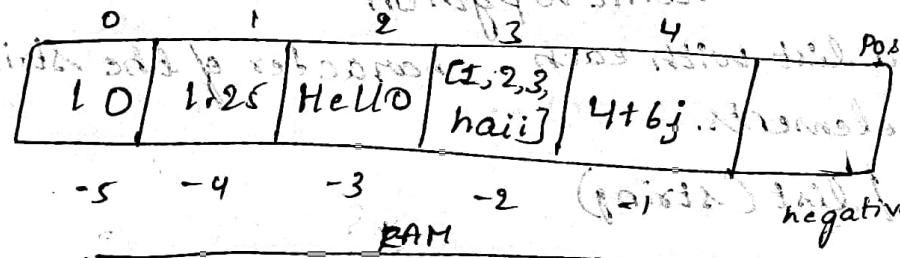
Ways to create tuple

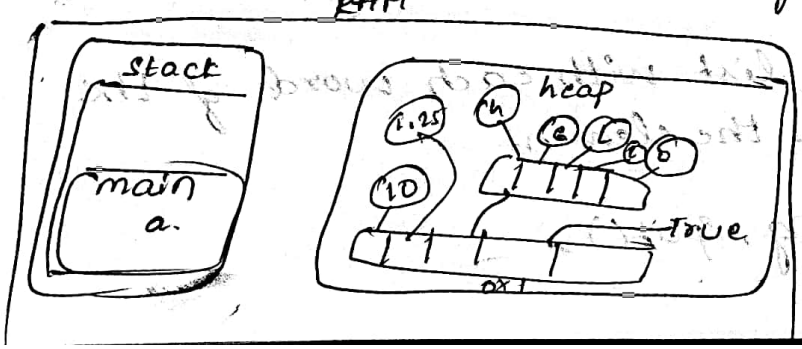
* t = () it creates an empty tuple
* t = tuple() using tuple constructor.

t = (1,) single item tuple

t = (1, 2, 3, 4, 5) tuple of integer objects.

eg. a = (10, 1.25, "hello", [1, 2, 3, "hai"], 4+6j)





2. a = {1, 2, 3, 4}

b = "hello"

a.update(b) >>> None

a >>> {1, 2, 3, 4, "h", "e", "l", "o"}

REMOVING ELEMENTS FROM A SET

* Remove(): removes the specified element from the set.

> Raises Keyerror if element is not present.

* Discard(): Remove the specified item

> Does not raise any error if the specified element is not present.

* POP(): Removes a random items from the set

> Returns the removed item.

> if the set is empty, it raises Key error.

* clear(); Removes all the elements from the set.

Eg. a = {'apple', 'Yahoo', 'google'}

* a.remove("apple") >>> None

a >>> {'Yahoo', 'google'}

Element must exist in the set

otherwise python throws key error

* a.discard('apple') >>> none

if the element does not exist, Key error is not thrown.

* a.pop() # throws Key Error if the set is empty

2. a = {1, 2, 3, 30, 300}
   b = {10, 20, 30, 40}

a. symmetric_difference(b) >>> {1, 2, 3, 10, 20, 40, 300}

* Symmetric_difference_update(): updates the base set with the items that are not common among all the sets.

Eg a = {1, 2, 3, 4}
   b = {3, 4, 5, 6}

a. symmetric_difference_update(b) >>> None.

a >>> {1, 2, 5, 6}

* ADDING ELEMENTS TO SETS

* add()
> adds an element (individual and Collection immutable) to the existing set.
> if the element already exists, the add() method does not add the element.

Eg. a = {'apple', 'Yahoo', 'google'}
   a.add('facebook') >>> none.

a >>> {'apple', 'Yahoo', 'google', 'facebook'}

UPDATE()

Update(): updates the current set by adding items from any iterable.

1. a = {1, 2, 3, 4}
   b = {3, 4, 5, 6}
   a. update(b) >>> none

   a >>> {1, 2, 3, 4, 5, 6}

Eg 1. a = {1, 2, 3, 4}
b = {3, 4, 5, 6}

a.difference(b) >>> {1, 2}

a - b >>> {1, 2}

Returns a new set with the elements in set a which are not in b.

2. a = {1, 2, 3, 30, 300}
b = {10, 20, 30, 40}
c = {100, 200, 300, 400}

a.difference(b, c) >>> {1, 2, 3}

Returns a new set with the elements in set a which are not in b, c

* Difference_update(): Updates the base set with elements that are present in base set but not in other set

Eg. a = {1, 2, 3}
b = {2}
a.difference_update(b) >>> None
a >>> {1, 3}

SYMMETRIC_difference(), Symmetric_difference_update()

Symmetric_difference(): returns a set that contains all items that are not common among all the sets.

1. a = {1, 2, 3, 4}
b = {3, 4, 5, 6}
a.Symmetric_difference(b) >>> {1, 2, 5, 6}

UNION()

* UNION(): returns a set that contains all items from the original set and all items from the specified sets.

a = {1, 2, 3, 4}
b = {3, 4, 5, 6}.

a.union(b) >>> {1, 2, 3, 4, 5, 6}

c = {6, 7, 8, 9}

a.union(b, c) >>> {1, 2, 3, 4, 5, 6, 7, 8 9}

intersection(), intersection_update()

* Intersection(): returns a set that contains the similarity between two or more sets.

a = {1, 2, 3, 4}
b = {3, 4, 5, 6}

a.intersection(b) >>> Returns a new set {3, 4}

* intersection_update(): modifies a set by retaining only elements found in both sets.

a = {1, 2, 3}
b = {2, 3, 4}

a.intersection_update(b) >> none.

a >>> {2, 3}

Difference(), difference_update()

* Difference():
  - > Returns a set containing the difference between two or more sets.
  - > The returned set contains items that exist only in the base set, but not in other set.

SETS

Sets are pythons built in datatype which has the following characteristics.

- Sets are unordered.
- Elements inside the Sets are unique.
- Sets are mutable, but elements inside the Set must be hashable.
- Sets cannot be indexed or sliced.
- Python Sets can only include hashable objects.

> Boundary : {---}
> To create empty set : var-name = set()
> Syntax : var-name = {1, 2.5 (1, 2, 3), "hai"}
> using set Constructor : var-name = set (var1)
> Length of set : len (var-name)

HASHABLE OBJECTS

> hashable objects are the objects which implements __hash__ magic method and hash() method can be called.
> only immutable objects will have hash values, hence hash() can be used to check for mutability and immutability.
> all immutable objects are hashable but all hashable objects are not immutable

1. How to create single value tuple.
2. Output of : C = C * t1, * t2)
3. Output of : t = (1, 2, 3, 4, ["hai", "hello", 23], "Python)

   t[4][0][-1] = "y" >>> t = ?
4. t[1:3] = ["hai", 10] >>> t = ?
5. what is the Output of the following.

```
a = [1, 2, 3]      b = [4, 5, 6]
Print ([a, b])
Print ((a, b))
```

* Git add foldername

you can remove whole folder by giving foldername.

* git rm -r --- Cached folder_name/ (or)

* Git Commit -m "message" (this command records the file permanently in the Version history.

* Git log (used to list the Version history for the current branch)

* Git log --online (i online)

* open github goto new repository and Create by giving a newname.

* Copy the url of that new repository.

* Git remote add release-name "Paste of URL"

refname:- generally we use origin as ref name.

Variable path or URL:- paste that copied path of repository.

* Git Push origin Branch_name

Branch name:- Here master is the branchname if you are not main person to this github is if you are not master to that you can create a new branchname and you can give that name as Branch name.

* you can Create any number of branches and you can add to that.

* we can see the file in any branch which we updated a master branch.

Git Pull [Repository link]

- This command fetches and merges changes on the remote server to your working directory.

GIT CLONE URL

This command is used to make a copy of a repository from an existing URL.

if you want a local copy of any repository from Github, This command allows Creating a local Copy of that repository on your local directory from the repository URL.

Git merge Branchname

This command is used to merge the specified branches history into the current branch.

STEPS FOR GIT

* create a file inside a folder with Py exe
* open folder and press on path and open command prompt.
* use below step1 and Commands:
* * Git init (initialize empty repository to git)
* Git Config --global user.name var-name (Configure the names and mail
* Git Config --global user.email
* Git status (It will show status of your directory)
* Git add filename (it will adds file)
* Git status (it will shows the files which are added)
* Git rm --cached filename
* Git status (to check removed or not)
* Git add * (it adds all files Creates a folder inside the folder add folder by using Command.

Git log --- online

This command is used to list the version history for the current branch in one line.

Git remode add [variable name] ["Remote Server link]

This Command is used to Connect your local repository to the remote Server.

Git push [variable name] master.

This Command Sends the Commited changes of master branch to your remote repository.

Git Push [variable name] branch

This command sends the branch commits to your remote repository.

Git Push all [variable name]

This command pushes all branches to your remote repository.

Git branch

This Command lists all the local branches in the Current repository

Git branch - d [branchname]

This Command delete the feature branch.

Git checkout [Branch name]

This command is used to Switch from One branch to another.

Git checkout -b [branch name]

This command Creates a new branch and also Switches to it.

GIT COMMANDS

Git Config - This command is used to sets the author name and email address respectively to be used with your commits.

git config - global user.name "[name]"
git config - global user.email "[email address]"

Git init -> This command is used to start a new repository.

Usage : git init [repository name]

Git Clone -> This command is used to obtain a repository from an existing URL.

Usage git clone [URL]

git add filename :- This command adds a file to the Stagging area.

git add * This command adds one or more to the staging area.

Git Commit - m "Commit millage"

This command is used to records or snapshots the file prermanently in the version history.

Git Status : This comman lists all the files that have to be committed.

Git rm --cached filename

This command deletes the file from your working directory and stores the deletion.

Git log :- Git log command is used to list the version history for the current branch.

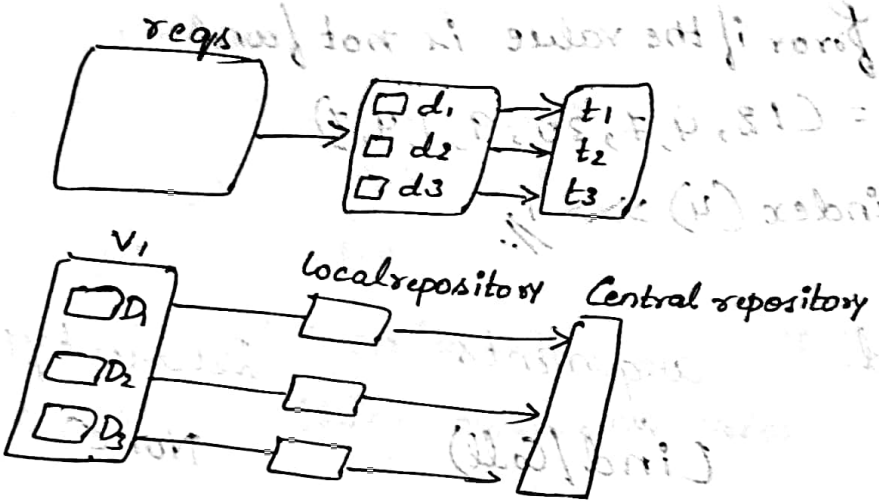
Version Control System

Local VCS
Centralized VCS
Distributed VCS
} types.

Version Control System are a category of software tools that helps in recording changes made to files by keeping a track of modification done to the Code.

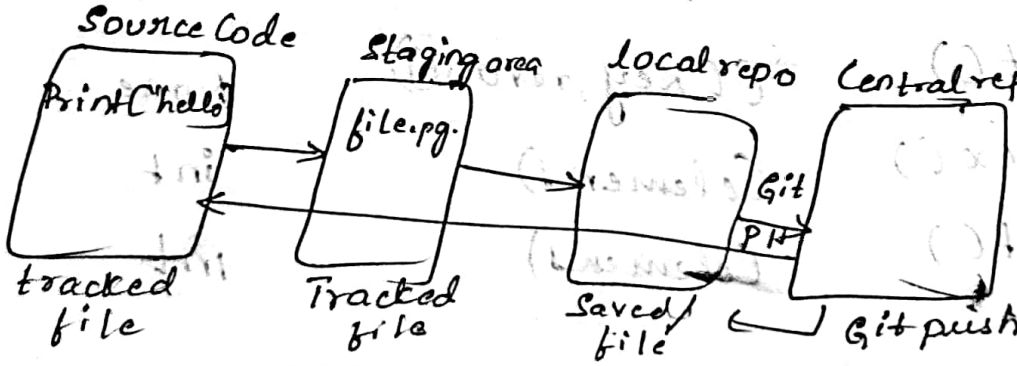
Types of version Control System are listed above

1.



GIT INTRODUCTION

* Git is an open source distributed version Control System
* It is designed to handle minor to major projects with high speed and efficiency.
* It is developed to Co-ordinate the work among the developers.
* The Version Control allows us to track and work together with an team members at the same workspace.



Characteristics of Dictionaries.

* Key cannot be duplicated.
* Keys will be a single element.
* Values can be of any datatype.
* Values can be accessed through keys only.
* Keys must be of immutable datatype or hashable.

COMOSITE KEYS.

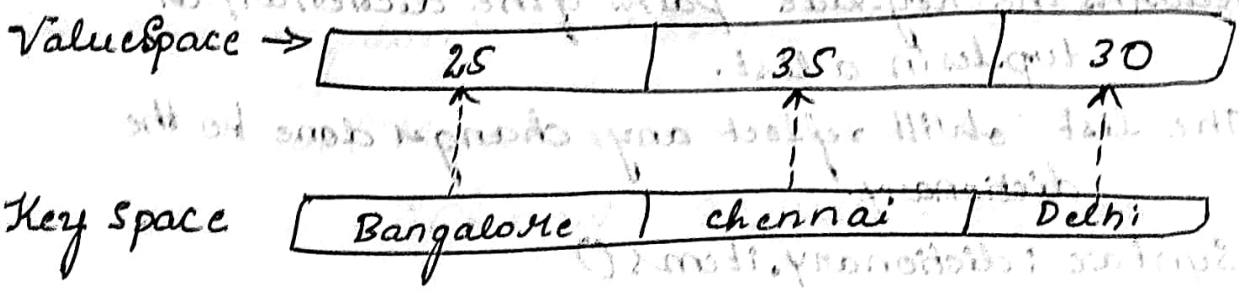
* Dictionaries can have composite keys i.e tuples as keys.

eg.

```
holidays = {
            (26,1) : 'Republic Day',
            (15,8) : "Independence Day",
            (21,6) : "Yoga day"
```

Memory allocation / deallocation in dictionary.

Eg d = {'Bangalore': 25, "Chennai" : 35, "Delhi" : 30};



d ⟶ 0x122.

Accessing values from a dictionary.

get() : Returns the value of the item with the specified key.

→ If keyname and value is not present, then it will give the second parameter [value] as value

PYTHON DICTIONARIES

```
d = {}
type(d)
<class 'dict'>
d = dict()
type(d)
<class 'dict'>
```

```
d = {"a":1, "b":2}
d = dict({"a":1, "b":2})
d = dict(a=1, b=2)
d1 = dist([["a",1),("b",2),("c",3)])
```

* only 2 values.
* only immutable should be first.

-> count the no. of keys
= length of Dist
= len(D1)

* collection of Key-value pairs.
* Each element is associated with unique key.
* Seperated by Comma operator.

> Boundary : { ... }
> Syntax : var_name = {key1 : value1, key2 : value2}
> length : returns the number of keys present in the dictionary. len(dist_variable)
> it does not support indexing slicing

Different ways of Construction a dictionary

```
>d = {}          # empty dictionary
>d = dict()
```

Using dictionary Constructor.

```
>d = dict(Bangalore = 25, chennai = 35, Delhi = 30)
>d = dict([("Bangalore", 25), ("chennai", 35), ("Delhi", 30)])
>d = dict({'Bangalore': 25, "chennai": 35, "Delhi": 30})
```

Method	args	returntype
1) union	iterables	Set
2) intersection	iterables	Set
3) Difference	iterables	Set
4) Symmetric_difference	iterable	Set
5) Intersection_update	iterables	None
6) difference_update	iterables	None
7) Symmetric_difference_update	iterables	None.
8) add.	all immutable datatype, indiridual and Collection.	None
9) update	iterables.	None
10) Remove (throws error)	Element.	None
11) Discard (Any argument will work)	Element	None
12) POP	-	Removed element
12) Clear	-	None
14) Is disjoint	-	Boolean Value.
Is Superset	-	
Is Subset.	-	

ISDISJOINT(), IS SUPERSET(), IS SUBSET()

* issubset(); Returns true if all items in the base Set exists in the reference set, otherwise returns false

Eg. x = {1, 2, 3}
a = {1, 2, 3, 30, 200}

x.issubset(a) >>> true.

y = {2, 3, 5}

y.issubset(a) >>> false.

* Issuperset(): Returns true if all items in the specified Set exists if the original set, otherwise it returns false.

Eg: x = {1, 2, 3}
a = {1, 2, 3, 30, 200}

a.issuperset(x) >>> true
x.issuperset(a) >>> false

* ISDISJOINT(); return true if none of the items are present in both sets, otherwise it returns false

Eg: X1 = {1, 3, 5}
X2 = {2, 4, 6}
X3 = {2, 3, 4}

X1.isdisjoint(X2) >>> true
X1.isdisjoint(X3) >>> False.

S, t, ID -> add

Eg. names = ['apple', 'googe', 'Yahoo', 'gmail', 'googb', 'apple']

* Count = dist.fromkeys (names, 0)
* Count >>> {'apple': 0, 'google': 0, 'Yahoo': 0, 'gmail': 0}

Deleting the Key and Value

POP() :- Removes the Specified item from the dictionary.
> Returns the removed value as output.
> Syntax: dictionary.pop(Keyname, default value)
> Default Value Value to return if the Specified Key does not exist.

POPitem() : Removes the item that was inserted at last into the dictionary
> returns the removed value as tuple.
> Syntax : dictionary.popitem()

Eg employee = {'name': 'Ram', 'age': 30}
d. popitem() # returns & delete the last Keyvalue pair in dictionary.

* print(d.pop('age')) # return and delete the mentioned Key from dictionary.
* print (d.pop ('phone')) # raises Key from
* Print (d.pop ('phone', 'Key is not present'))
# returns Key is not present.
* del employee ('age')
# delete the Key 'age' and gets value

Adding/updating the dictionary

using Key and value Syntax.

d['Mysore'] = 26.5 # updating the dictionary key with new value:

Update(): updates the existing dictionaries with the specified items.

d.update({"Mysore": 26, "Cochin.": 28})

d.update(Mangalore = 24)

Set default([Keyname, [value]); returns the value of the items with the specified key.

* If the value is not specified then it adds None as the value else the value given will be added.
* If the key does not exist, it will insert the key with the specified value.

Eg. d = {'Bangalore': 25, 'Chennai': 35, "Delhi": 30}

d.set default("Mysore") # d = {'Bangalore': 25, "Chennai": 35, "Delhi": 30, "Mysore": None}

d.set default("Kolkata", 60) # d = {'Bangalore': 25, "Chennai": 35, "Delhi": 30, "Kolkata": 60}

FROM KEYS()

From keys(): returns a dictionary with the specified keys and the specified value.

> Syntax: dict.from(Keys, [value])

> keys Required. An iterable specifying the keys of the new dictionary

> Value optional: The value for all keys. default value is (none)

Eg. d = {'Bangalore': 25, 'chennai' = 35, "Delhi": 30}

d ['Bangalore'] >>> 25

d.get('Bangalore') >>> 25

Accessing a key that does not exist.

- d ["Noida"] >>> Key error
- print (d.get ("Noida")) >>> None.

get() method does not throw an error, but returns 'None' by default if the second argument is not given

- print (d.get ('Noida', 'The Key is not found in the dictionary'))

>>> "The Key is not found in the dictionary."

ITEMS(), KEYS(), Values()

items(): method returns a view object.

> Returns the key-value pairs of the dictionary as tuples in a list.

> The list will reflect any changes done to the dictionary.

> Syntax : dictionary.items()

Keys(): Returns a list of all the Keys present in the dictionary.

> List will reflect any changes done to the dictionary

> Syntax : dictionary.Keys()

Value(): Returns a list of all the Values.

> Syntax : dictionary-value()

check if the integer is palindrome or not

```
int = 151
if int [::-1] == int:
    print("palindrome")
else:
    print("not a palindrome")
```

```
a=str(num), num=1221
if a[::-1] == a:
    Print(f"number{num} is a palindrome")
else:
    Print(f"number {num} is not a palindrome")
```

Check if the given character is a special character or not.

```
a = Input("enter a character :")
if a.~~isalnum~~ alnum() != True:
    Print("It is a special character")
```

or.

```
if not a.alnum():
    Print("It is a special character")
```

---

check if the iterable is empty or not

```
l = [1, 2, 3]
if len(l) == 0:
    print("empty")
else:
    print("notempty")
```

```
if bool(l):
    Print("notempty")
else:
    print("empty")
```

(or)

```
if l:
    print("notempty")
else:
    print("empty")
```

```
a=2          (or)  a=int(input("enter a number :"))
if (a % 2=0):  → start of block
  print("even number")
else:
  Print("odd number")
Print(" end of if statement")
```

```
if a % 2 == 0 : print("even number")
```
space → In line if statement

```
if else
if a % 2 == 0:
    Print("even number")
else:
    Print("odd number")
```

```
String = "malayalam"  (or) String =input("entera string:")
if string[::-1] == string:   → if condition check bool of the string.
      print ("palindrome")
else:
      print("not a palindrome")
```

String starting with vowel

```
string =input("enter a string :")
if string[0][== "a" or string[0] == "e" or string[0] == "i"
                or string[0] == "O" or string[0] == "U"]
                or
if string[0] in "aeiouAEIOU":
      print("Starts with vowel")
else: Print("does not start with vowel")
```

* is: Returns true if both variables are pointing to the same id x is y.
* is not: Returns true if both variables are not pointing to the same id x is not y.

MEMBERSHIP OPERATORS

* used to test if an element is present in an iterable.
* In: Evaluates to true if it finds the element in the specified iterable and false otherwise.
* Notin: Evaluates to true if it does not find the element in the specified iterable and false otherwise.

BITWISEE OPERATOR.

> Bitwise AND (&) : sets each bit to 1 if both bits are 1

> Bitwise OR (|) : sets each bit to 1 if one of two bits is 1

> Bitwise XOR (^) : sets the output as 1 if any one bit is 1 else sets it to 0

> Bitwise NOT (~) : Inverts all the bits

> Bitwise right shift or zero fill left Shift (<<) : shift left by pushing zeros in from the right and let the leftmost bits fall of

> Bitwise left shift or Signed right shift (>>)

Shift right by pushing Copies of the leftmost bit in from the left, and let the rightmost bits fall off

IDENTITY OPERATOR

* used to check if two values are located on the same part of the memory.
* used to compare the objects. not if they are equal. but if they are actually the same object with the same memory location

OPERATORS

ARITHMETIC OPERATORS

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulus (%)
- Exponent or power (**)
- Floor (//) : performs division and rounds off the quotient to the nearest value.

If both operands +ve rounds off to the nearest lower value

Eg : 7//2 3

If any one operand -ve rounds off to nearest lower value with "-" sign

eg - 7//2 -4

Relational Operators

- Greater than (>) : True if left operand is greater than the right.
- Less than (<) : True if left operand is less than the right
- Equal to (==) : True if both operands are equal
- Not equal to (!=) : True if operands are not equal
- Greater than or equal to (>=) = True if left operand is greater than or equal to the right.
- Less than or equal to (<=) : True if left operand is Less than or equal to the right.

Properties	String	List	tuple	Set	Dictionary
Boundary	''/""/''' '''	[..]	(..)	{..}	{key : value}
Empty Datatype	str()/empty quotes	list()/[]	tuple()/()	set()	dict()/{}
Mutable	NO	Yes	NO	Yes	Yes
indexing and slicing	Yes.	Yes	Yes	No	NO
Duplicates	Allowed	Allowed	Allowed	Not allowed	Key duplication not possible
Datatypes allowed	All	All	All	Immutable hashable.	Keys - hashable ~~value~~ - any.

Method	args	Return type.
get()	('key', [value]	Value
key()	–	List of keys
values()	–	List of values
items()	–	List of tuples of key/value
update	({key:value}) (key - value)	None
Set default	(key, [value])	Value
fromkeys()	(iterable, [value])	Dictionary
PoPitem()	–	type of removed item.
POP()	(key, [value])	Removed value

Merging

If I want to merge i need to unpack with or without return keys.

pipeline() → is used in 3.10 Version if I want to merge 2 dict

Merging Dictionaries.

```
d1 = {'fname' : 'Steve', 'lname' : 'Jobs'}
d2 = {'age' : 56, 'Company' : 'apple'}
d3 = {**d1 , **d2}
```

(Or)

```
d3 = d1 | d2.
```

SEQUENCE V/s ITERABLES..

1. A Sequence is an object which can be indexed.
2. All Sequences are iterables, But all iterables are not sequences.

Inline

Print("Key is present" if Key in d else "not present")

Print("value is present" if value in d.values() else "Value is not present")

CONDITIONAL STATEMENTS.

> conditional statements in programming languages decides the direction of flow of program execution

> types.

* If statements.
* if else statements.
* elif statements.
* Nested if-else statement

Syntax

```
if condition :
    Statement 1
    Statement 2
    -------
    Statement n
```

Eg :

```
a = 3
if a > 2:
    print("entered the block")
```

INLINE if Conditional statement.

Syntax : if (condition) : statement 1 ; statement 2 ; ..... ; Statement n

inline if-else — used in comprehension.

iterable = "hello"

Syntax = {True block} if Condition else {false block}

print("not empty" if iterable else "empty")

---

① print("even number" if (a%2==0) else "odd number")

② print("palindrom" if (string[::-1]==string) else "not a palindrome")

③ print("starts with vowel" if string[0] in "aeiou AEIOU" else "does not starts with vowel")

④ print("it is a special character" if not a.alnum()) (wrong method)

⑤ if not a.alnum(): print("it is a special character")

---

write a program to check if the Key is present in the dictionary or not

d = {"a": 1, "b": 2}

Key = "C"

if Key in d.keys(): (Keylayer)

    print("Key is present")

else:

    print("not present")

Eg : a=3

if(a>2): print("greater"); print("done execution")

If-else Condition.

Syntax : if(Condition)

    ## true block ##

else:

    # False block ##

Eg : a=input()

if(a!=9):

    Print("not equal")

else :

    Print("equal")

INLINE IF-else Conditional Statement

Syntax : True statement if(Condition) else False statement

Eg: a=3

Print("greater" if (a>2) else "done execution").

Elif conditional statement

> Syntax : if(Condition):

        Statement

    elif (Condition):

        statement.

WHILE LOOP

* It is used when the number of iterations to be done is not known
* Steps to follows.
  → Keep a track of number of iterations with a reference variable.
  → Manually increment/decrement the reference variable

Syntax
```
while condition
    Statements.
```

2) write a program to print numbers from -10 to -1

(-10+1) → incremention

```
start=-10
end=-1
while start <= end:
    print (start)
    start+=1
```

3) write a program to ~~print~~ check whether the given number is prime numbers from 1 to 10 or not.

X
2 → 5%2 = 1
3 → 5%3 = 2
4 → 5%4 = 1
X

```
num = 5
i = 2
while i < num:
    if num % i == 0:
        Print ("not prime")
        break
    i += 1
else:
    Print ("no is prime")
```

```
ch = "a"
if ch in "aeiouAEIOU":
    d = dict.fromkeys(ch, ord(ch))
    print(d)
else:
    print("its not a vowel")
```

or

```
d[ch] = ord(ch)
d.update({"ch": ord(ch)})
```

---

elif -> when we want to check one by one condition
we go for elif -> when 1st condition fails.
then we go for elif.

LOOPS

WHILE LOOP

1) start = 1 ← even numbers from 1-10

```
start = 1
end = 10
while start <= end:
    if start % 2 == 0:
        print(start)
    start += 1
```

← odd numbers from 1 to 10

```
start = 1
end = 10
while start <= end:
    if start % 2 == 1:
        print(start)
    start += 1
```

```
y = int(input("enter the year:"))
if ((y % 400 == 0) or (y % 100 != 0 and y % 4 == 0)):
    print("its a leap year")
else:
    print("its not a leap year")
```

Check the no. of keys in the dictionary, if the number is even print the dictionary.

```
d = {"a": 1, "b": 2, "c": 3}
if len(d) % 2 == 0:
    print(d)
elif d.update(d=4):
    print(d)
```

1) or d.update({"d": 4})
2) d.setdefault("d", 4)
3) or d["d"] = 4

---

check if the

---

check if the entered character is vowel or not, if it is vowel than create a dictionary with char and its ascii value.

```
char = input("enter char:")
dic = {}
if char in "aeiouAEIOU":
    dic[char] = ord(char)
    print(dic)
else:
    print("not a vowel")
```

Without using inbuilt function

```
Char = "h"
if "a" <= char <= "z":
    Print (chr (ord (char) - 32))
elif "A" <= char <= "z":
    Print (chr (ord (char) + 32))
```

ord gives ASCII
lower(a) → 97
(A) 65
} ASCII

~~char~~.

Check if ~~entered character~~ the given year is leap year or not.

```
year = int (input ("enter the year :"))
if year % 4 == 0:
    Print ("Its a leap year")
else :
    Print ("its not a leap year")
```

(or)

```
y = int (input ("enter the year :"))
if y % 400 == 0) and (y % 100 == 0):
    Print ("its a leap year")
elif (y % 4 == 0) and (y % 100 != 0):
    Print ("its a leap year")
else :
    Print ("Not a leap year")
```

Check whether last 2nd digit is even or not

```
n = 2345
if isinstance(n,(int,str)) and int(str(n)[-2]) %2==0:
    print("its even")
else:
    print("its odd")
```

Check Greatest of 3 numbers.

```
a = 20
b = 30
c = 40
if a > b and a > c:
    Print("a is greater")
elif b > c and c > b:
    Print("b is greater")
else:
    Print("c is greater")
```

---

upper to lower and Viceversa.

```
a = "BHAVYA"
if a == a.upper():
    print(a.lower())
else:
    print("remains same")
```

```
a = "bhavya"
if a == a.lower():
    Print(a.upper())
else:
    Print("remains same")
```

```
a="holiday"
if isinstance(a, str):
    Print("its true")
else:
    Print("its false")
```

Check if the given string ends with vowel or not.

```
a = "Bharya"
if a[-1] in "aeiouAEIOU":
    Print("it ends with vowel")
else:
    Print("it ends with Cons")
```

or `if a[-1] in "AEIOU".upper()`

`if a[-1] in "aeiou".swapcase():`

Check if the list has even number of elements.

```
l = [1, 2, 3, 4, 5, 6, 7, 8]

if len(l) % 2 == 0:
    Print("it has even number of elements")
else:
    print("it has odd number of elements")
```

In a number check if the 1st number is even or odd.

```
n = 23456
if isinstance(n, int) and int(str(n)[0]) % 2 == 0:
    Print("its even")
else:
    Print("its odd")
```

else:
    Statement

> Eg: a = input()
    if (a!=9):
        Print("not equal")
    elif a>9:
        Print("greater")
    else:
        Print("equal")

> Ord()
- returns ASCII value of the specified character.
- Eg: ord("A") >>> 65

> chr()
- returns the character of specified ASCII value.
- Eg: char(65) >>> "A"

ACTS

Write a program to check if the given value is string or not

```
x = "Hello"
key = "H"
if key in x
```

```
a = input("enter the value:")
if isinstance(a,str):
    Print("is a string")
else:
    Print("not a string")
```

> To loop through a set of code a specified number of times, we use the range() function.
> Start index is always included.
> End index is always excluded.

SYNTAX FOR "FOR LOOP"

Syntax 1:

```
for ref_var in iterable:
    Statements.
```

Syntax 2:

```
for ref_var in ranges(SI, EI, step):
    Statements.
```

Note: _ (underscore) is called throw away variable. It is used for ignoring certain values.

LOOP WITH ELSE BLOCK

> The else keyword in any loop specifies a block of code to be executed when the loop is finished without any break.

> Syntax:

```
for value in iterable:
    Statements
else:
    Statements.
```

FORLOOP → range is inbuilt

```
l=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
range(1,11,1)
range(1,11)
>>list()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>list(range(1,11,1))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>List(range(1,11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>List(range(11))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>List(range(0,11,2))
[0, 2, 4, 6, 8, 10]
>>List(range(1,11,2))
[1, 3, 5, 7, 9]
```

```
>>str(range(11))
'range(0,11)'
>>range(11)
range(0,11)
```

Forloop.

* It is used when the number of iterations to be done is known
* There is no need to keep a track of iteration as for loop does it implicitly.
* No need to increment the reference variable manually.

Range()

range(start, end, step):

> Returns a range object sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Write a program to print fibonacci series till 10

a	b	c
0	1	1
1	1	2
1	2	3
2	3	5
3	5	8
5	8	13

```
a=0
b=1
while a<=10:
    print(a)
    c=a+b
    a=b
    b=c
```

```
a,b=0,1
while a<=10:
    print(a)
    c=a+b
    a,b=b,c
```

Fibonacci Series upto 10 numbers.

```
a=0
b=1
Count=0
while Count<=10:
    Print(a)
    c=a+b
    a,b=b,c
    Count+=1
```

Write a program to traverse through a string printing each character.

```
String="hello world"
i=0
while i<len(string):
    print(i,string[i])
    i+=1
```

Write a program to check multiple numbers (1 to 10) is prime or not

1st while	2nd while.	
num	i=2.	num/2 num **0.5
-> 1		
2		
3	2, 3	
4	2	
5	2, 3, 4	
6	2	
7	2, 3.	

```
num = 1
while num <=10:
    if num>1:
        i = 2
        while i<=num: (//2) | while i<=num//2:
            if num%i==0:
                break.
            i+=1
        else:
            print(num)
    num+=1
```

```
s = "hello world"
d = {}
for ch in s:
    d[ch] = ord(ch)
print(d)
```

→ To print ascii values with the string by creating dictionary

---

```
s = "hello hi"
d = {}
for ch in s:
    d[ch] = s.count(ch)
print(d)
```

→ when there is no inbuilt function.

→ How many times the character is present by inbuilt function.

```
s = "hello hi"
count = 0
for ch in s:
    count += 1
print(count)
```

→ without using inbuilt function.

```
s = "hello hi"
d = {}
for ch in s:
    d[ch] = s.count(ch)
Print(d)
```

```
s = "hello hi"
d = {}
for ch in s:
    if ch not in d:
        d[ch] = 1
    else:
        d[ch] += 1
Print(d)
```

write a program to print ascii values of each character in a string.

4) x = "hello"

```
for i in s:
    print(i)

for i in range(len(s)):
    print(s[i])
```

h | e | l | l | o

```
string = "helloworld"
for char in string:
    print(char, ord(char))
```

```
s = "hAi goOd morning"
for char in s:
    if char in "aeiouAEIOU":
        print(f"{char} is vowel")
    else:
        print(f"{char} is not a vowel")
```

Checking vowel or not

```
For char in s:
    if char in "aeiouAEIOU':
        if ord('a') <= ord(char) <= ord("z"):
            res = res + chr(ord(char) - 32)

        elif ord("A") <= ord(char <= ord("Z"):
            res = res + chr(ord(char) + 32)

        else:
            res = res + char
Print(res)
```

3) write a program to check whether prime number from 1 to 10.

```
for num in range(1,10):
    if num>1:
        for i in range(2, num//2):+1
            if num%i==0:
                break.
        else:
            print(num)
```

4) write a program to traverse through an iterable.

```
s = "hello"
i = 0
while i<len(s):
    print(i, s[i])
    i+=1
print()
```

→ whileloop

→ (To give 1 blank space)

```
for i in range(len(s)):
    print(i, s[i])
```

- for loop

((or))

```
for i in range(len(s)):
    print(i, s[i])
print()
for char in s:
    Print(char)
```

List.

Set

```
s={"hello", "hai", "world"}
for element in s:
    Print(element)
```

Dictionary

```
d={"a":1, "b":2, "c":3}
for key in d:
    Print(key)
```

to get values and keys we use

```
Print(key, d[key])
```

LOOP CONTROL STATEMENTS

> There are 3 loop control statements

1. Pass
2. break.
3. continue.

> Pass : performs no operation

> break : stops the execution of loop and breaks out of it.

> continue : skips the current execution and ~~Confines~~ continues with the next one.

1). Write a program to print even numbers from 1-50

```
For num in range(1,50):
    if num % 2 == 0:
        print(num)
```

for odd

```
if num % 2 == 1:
```

2) write a program to check whether the given number is prime or not

```
num = 12
for i in range(2, int(num ** 0.5)):
    if num % i == 0:
        print("not prime")
        break
else:
    print("prime")
```

int(1.42)

10) write a program to create a list with even length words.

Sentence = "hai good afternoon, welcome to afternoon Session"

```
l = S.split()
list_ = []
for word in l:
    if len(word) % 2 == 0:
        list_.append(word)
Print(list_)
```

---

11) WAP to create a list words ending with vowels.

S = "hai good afternoon, welcome to afternoon Session"

```
world = S.split()
l = []
for word in w:
    if word[-1] in "aeiouAEIOU":
        l.append(word)
Print(l)
```

12) WAP to create a set words ending with vowels.

Sentence = "hai good afternoon, welcome to afternoon Session"

```
w = Sentence.split()
s = set()
for word in w:
```

7) write a program to print first Occurance of the indexes of the substring.

```
s="hello world"
ch="o"
print(s.index(ch)) >inbuiltfunction

for i in range(len(s)): without using inbuitfunction
    if s[i]==ch:
        print(i)
        break.
```

8) 2nd Occurance.

```
s="helloworld"
ch="o"
count=0
for i in range(len(s)):
    if s[i]==ch:
        count+=1
        if count==2:
            print(i).
```

9) write a program to print words starting with vowels.

```
sentence = "it is tuesday"

l=sentence.split()
for word in l:
    if word[0] in "aeiouAEIOU":
        print(word)
```

```
for ch in s:
    if ch not in res:
        res.t=ch
Print (res)
Print (set(s))
```

without using inbuilt

---

5) write a program to print duplicate characters without using inbuilt method.

```
S = "hello world"
res = " "
duplicates = " "
for ch in s:
    if ch not in res:
        res t=ch
    else:
        duplicate t=ch
Print("The non duplicates are", res)
Print("The duplicates are", duplicates)
```

6) write a program to print all the indices of the given substring

```
S= "hello world"
ch = "O"
for i in range (len(s)):
    if s[i]==ch:
        print(i, end=" ")
```

Sentence = "hai how are you"

```
count_a=0
for b in sentence:
  if b.isalpha():
    count_at=1
print(count_a)
```

```
count = 0
l= Sentence.split()
Print (l)
For word in l:
  Count t=1
Print (Count)
```

3) write a program to print all the repeated characters.

```
S= "hello world"
for char in S:
    if S.count(char)>1:
        print(char ,end = "")
```

(or)

```
S= "hello world"
d={}
for ch in S:
  if ch not in d:
    d[ch]=1
  else:
   d(ch)+=1
for ch in d:
  if d(ch)>1:
   Print(d)
```

(or)

```
S="hello world"
for char in seb(S):
  if S.count(char)>1:
  Print(char, end="")
```

4) write a program to remove the duplicate or repeated character in the string.

```
S= "hell o world"
for ch in S:
    if S.count(ch)==1:
      rest=ch.
```

(or)

```
res=""
print(res)
```

Write a program to count the no. of digits and alphabets in the string.

(with using inbuilt function)

```
S = "hai 1234 python 23"

count_a = 0
count_d = 0

for i in s
    if i.isalpha():
        count_a += 1
    elif i.isdigit():
        count_d += 1
Print (Count_a , Count_d)
```

---

without using inbuilt function.

```
for i in S :
    if ord("a") <= ord(i) <= ord("z") or ord("A")
                    <= ord(i) <= ord("Z"):

        count_a += 1
    elif "0" <= i <= "9":
        Count_d += 1
Print (Count_a , Count_d)
```

Write a program to count the number of words in the sentence.

ENUMERATE (iterable)

- Returns an enumerate object.
- iterable can be a sequence, an iterator, or some other object which supports iteration.
- Returns a tuple containing a count (from start which defaults to 0) and the values obtained from iterating over the iterable.

print index and element of an iterable

```
iterable = ["apple", "google", "walmart", "flipkart", "gmail"]

for i in enumerate(iterable):
    print(i)
```

2) write a program to print the element present in even indices.

```
iterable = ["apple", "google", "walmart", "flipkart", "gmail"]

for i in enumerate(iterable):
    if i[0] % 2 == 0:
        print(i[1])
```

```
for index, item in enumerate(iterable):
    if index % 2 == 0:
        print(item)
```

```
for i in iterable[::2]:
    print(i)
```

```
for i in range(len(iterable)):
    if i % 2 == 0:
        print(iterable[i])
```

#concatenation

```
res = ""
for ele in sequence:
    res = ele + res.
print(res)
```

---

reversed() - inbuilt Class.

```
for ele in reversed(sequence):
    print(ele, end = " ")
```

Reversed(Sequence)

> Returns a revers iterator object.
> Sequence must be an object which has a - reversed_() method.

Printing hello world 5 times.

```
S = "hello world"
for _ in range(5):
    print(s)
```

(_) Throwaway variable.

write a program to print index and element of an iterable.

```
Iterable = ["apple", "google", "walmart", "flipkart", "gmail"]
for i in range(len(iterable)):
    print(i, iterable[i])
```

```
d={"a":1 , "b":"hello", "c":85 , "d":12.3,
   "e":[1,2,3]}

d1={}

for i in d:
   if isinstance(d[i],int):
      d1[i]=d[i]
print(d1)
```

(or)

```
d1={}
for i in d:
   if type(d[i])==int
```

```
for key,value in d.items():
   if isinstance(value,int):
      d1[key]=value[::-1]
   else:
      d1[key]=value
print(d1)
```

Write a program to create a dictionary if the values if of string reverse it

```
d1={}
for key,value in d.items():
   if isinstance(value,str):
      d1[key]=value[::-1]
   else:
      d1[key]=value
print(d1)
```

Write a program to print the elements in an iterable in reversed order

```
sequence="hello"
for ele in sequence[::-1]:
   print(ele,end=" ")
```

(or)

```
for ele in range(-1,-len(sequence)-1,-1):
   print(sequence[ele])
```

or

```
print(sequence[index] for index in range(-1,-len(sequence)-1,-1))
```

```
for word in w:
    if word[0] not in d:
        d[word[0]] = [word]
    else:
        d[word[0]].append(word)
print(d)
```

16) write a program to create a dictionary of character and its indices pair.

```
s = "helloworld"
d = {}
for i in range(len(s)):
    if s[i] not in d:
        d[s[i]] = [i]
    else:
        d[s[i]].append(i)
print(d)
```

17) write a program to create a dictionary of character and ascii value pair.

```
s = "hello word"
d = {}
for ch in s:
    d[ch] = ord[ch]
print(d)
```

18) write a program to create a dictionary with values of integer data type

```
if word[-1] = "aeiouAEIOU":
    s.add(word)
print(s)
```

13) write a program to create a list of tuples with word and its length pair.

```
Sentence="hai good afternoon , welcome to afternoon
          Session"
l = []
w = Sentence.split()
for war in w:
    l.append((word, len(word)))
Print(l)
```

14) create a dictionary with word and length pair only if the ~~vowels~~ words starting with vowe.

```
Sentence = "hai good afternoon , welcome to afternoon
                                        Session"
d = {}
w = Sentence.split()

for word in w:
    if word[0] in "aeiouAEIOU":
        d[word] = len(word)
Print(d)
```

15) create a dictionary with letter and the word starting with that letter pair

```
Sentence = "Python is a programming language"
d = {}
w = Sentence.split()
```

```
dd = defaultdict(list)
for index, element in enumerate(l):
    dd[element].append(index)
```

---

1) convert two lists into a dictionary.

2) temp = {"Bangalore": (26,32), "chennai": (29,35),
"Delhi": (31,36)}

```
for element in temp.items():
    print(element)

print()
for city, tem in temp.items():
    print(city, temp)

print()
for city, (min_, max_) in temp.items():
    print(city, min_, max_)
```

---

1) ~~write a program to create dict~~

Default dict()

```
from collections import defaultdict
dd = default dict (int)
for char in string:
    dd[char] += 1
Print (dd)
```

Tracing

Internally it will be written as

```
dd[char] = dd[char] + 1
dd[k] = 1 + 1
```

2

2) create a dictionary with word and its index pair.

```
l = ["apple", "google", "gmail", "apple", "gmail", "flipkart", "apple"]
d = {}
for index, ele in enumerate[l]:
    if d[index] = d[index] + 1
print [d]
    if element not in d:
        d[element] = [index]
    else:
        d[element] += [index]
```

or

```
d[element].append(index)
```

#Count()

```
d={}
for char in string:
    d[char]= sting.count(char)
print(d)
```

#without inbuilt method.

```
d={}
for char in string:
    if char not in d:
        d[char]=1
    else
        d[char]+=1
Print(d)
```

```
S='hello world'
d={}

for char in S:
  if char not in d:
    d[char]=1
  else:
    d[char]= d[char+1
    d[o] = d[o]+1
```

1+1

tracing

char	d.
h	{'h': 1}
e	{'h': 1, 'e': 1}
l	{'h': 1, 'e': 1, 'l': 1}
l	{'h': 1, 'e': 1, 'l': 2}
o	{'h': 1, 'e': 1, 'l': 2, 'o': 1}

Defaultdict()

defaultdict is used when the key has to be initialized and updated at the same time.

- unlike normal dictionary, here there is no need to check the condition for the presence of key
- Even if the key is not present defaultdict will create the key and initializes to its default value. Later the default value will get updated to the desired value.
- Syntax: From collections import defaultdict.

  d = defaultdict(datatype of value of the output)

* write a program to print a length of each word in the list in the form of tuples.

```
l2 = ["apple", "google", "flipkart"]
for element in l2:
    print((element, len(element)))
```

---

* creating a dictionary with ~~word~~ element and its count pair in the given iterable.

```
String = "hello world"
d = {}
for i in range
```

```
d[]
for i in range(len(l1)):
    d[l1[i]] = l2[i]
print(d)
```

```
for ele1,ele2 in zip(l1, l2):
    d[ele1]=ele2
Print(d)
```

ZIP()

> The zip() class takes iterable (can be one or more), aggregates them in a tuple, and return it.

> The zip() class returns a zip object which has list of tuples.

> If the passed iterators have different lengths, the iterator with the least items decides the length of the new iterator.

> Syntax : zip(iterator1, iterator2, iterator3...)

iterable : can be built-in iterable (like: list, String, dict)

Note:- To include all the elements of iterables of different length, we can use.

zip_longest() of itertools module, it will map the extra element with None by default

Syntax : zip_longest(iterator1, iterator2, iterator3..., fillvalue=None)

Write a program to create a dictionary of index and element pair, only if the elements are of odd length.

```
iterable = ["apple", "google", "walmart", "flipkart", "gmail"]
d = { }
for i in range (len(iterable)):
    if [i] % 2 == 1 : X
    print [i]
```

(not)

```
for index, ele in enumerate (iterable):
    if len(ele) % 2 == 1:
        d[index] = ele
print(d)
```

✓

* write a program to create a dictionary by making one list of elements as key and other list as value.

```
l1 = [10, 20, 30]
l2 = ["apple", "google", "flipkart"]
d = { }
for index, ele in enumerate (l1):
    d[l1[index]] = l2 [index]
Print(d)
```

write a program ~~with index and~~ word is given length
Keep it as it is else reverse it. wood pair.

Sentence = "hello good afternoon"

```
d={}
l=Sentence.split()

for index,ele in enumerate(l):
    if len(ele)%2==0:
        d[index]=ele
    else:
        d[index]=ele[::-1]

Print(d)
```

comprehension.

```
d1={index:ele if len(ele)%2==0 else ele[::-1]
    for index,ele in enumerate(l)}

print(d1)
```

Dictionary and Set Comprehension.

Set Comprehension

Syntax

```
{item for item in iterable}
```

Dictionary Comprehension

Syntax:

```
{key : value for item in iterable}
```

Create a dictionary with word and its length pair

```
Sentence = "hello good afternoon"
d = {}
l = Sentence.split()
for word in l:
    d[word] = len(word)
Print(d)
```

comprehension

```
d1 = {word : len(word) for word in l}
Print d1
```

2) create a dictionary with word and its length pair only it is of even length.

```
Sentence = "hello good afternoon"
d = {}
l = Sentence.split()
for word in l:
    if len(word) % 2 == 0:
        d[word] = len(word)
Print(d)
```

comprehension.

```
d1 = {word : len(word) for word in Sentence.
      Split() if len(word) % 2 == 0}
Print(d1)
```

```
if len(ch) % 2 == 0:
    lst.append(ch)
print(lst)
```

comprehension :-
```
lst1 = [ch for ch in names if len(ch) % 2 == 0]
print(lst1)
```

9) generating list of pi value with increasing decimal point comprehension.

```
from math import pi
print(pi)
print([round(pi, ch) for ch in range(1,9)])
a = 3.14159
print([round(a,ch) for ch in range(0,5)])
```

10) list comprehension to sum the factorial of numbers from 1-5.

```
import math
lst = [math.factorial(ch) for ch in range(1,5)]
print(lst)
```

```
From math import factorial
l = [factorial(ch) for ch in range(1,6)]
print(sum(l))
```

(or)

```
from math import factorial
l1 = []
sum_ = 0
for ch in range(1,6):
    sum_ = sum_ + factorial(ch)
print(sum_)
```

Without importing factorial.

```
fact = 1
num = 5
while num >= 1:
    fact = fact * num
    num -= 1
print(fact)
```

11) Reverse the item of list if the item is of odd length string.

```
names = ["apple", "google", "yahoo", "facebook",
         "yelp", "flipkart", "gmail", "instagram"]
lst = []
for ch in names:
    if len(ch) % 2 == 1:
        lst.append(ch[::-1])
print(lst)
print([ch[::-1] for ch in names if len(ch) % 2 == 1])
```

12) Reverse a item of a list if the list is of odd length string otherwise keep the item as it.

```
lst = []
for ch in names:
    if len(ch) % 2 == 1:
        lst.append(ch[::-1])
    else:
        lst.append(ch)
print([ch[::-1] if len(ch) % 2 == 1
       else ch for ch in names])
```

5) Filtering out those names which are less than 6 character.

```
names = ["apple", "google", "Yahoo", "gmail", "flipkart",
         "instagram", "microsoft"]
lst = []
for ch in names:
    if len(ch) < 6:
        lst.append(ch)
Print(lst)
```

comprehension

```
l2 = [a[ch]**ch for ch in range (len(a))]
print (l2)
```

```
(Print([ele**index for index, ele in
        enenumerate(a)]))
```

6) Build a list of tuples with string and its length pair

```
names = ["apple", "google", "Yahoo", "facebook", "yelp", "flipkart",
         "gmail", "instagram", "microsoft"]
lst = []
for ch in names:
    lst.append((ch, len(ch)))
Print(lst)
```

comprehension `[(ch, len(ch)) for ch in names]`

~~`lst1 = [ch for ch in names if len(ch) %2 == 0]`~~

```
print (lst1)
```

7) Build a list with only even length string.

```
names = ["apple", "google", "Yahoo", "facebook", "yelp"
         "flipkart", "gmail", "instagram", "microsoft"]
lst = []
for ch in names:
```

3] Filtering all the lng which starts with P

```
lng = ["Python", "java", "Perl", "PHP", "Python", "JS",
       "C++", "JS", "Python", "Ruby"]

lst = []
for ch in lng:
    if ch[0] == "P"
        lst.append(ch)

print(lst)
```

comprehension.

```
lst1 = [ch for ch in lng if ch[0] == "P"]
Print(lst1)
```

ii) Names starts with Consonants.

```
names = ["Laura", "Steve", "bill", "James", "bob",
         "greg", "Scott", "alex", "ive"]

lst=[]
for ch in names:
    if ch[0] not in "aeiouAEIOU":
        lst.append(ch)

print(lst)

#comprehension.
lst1 = [ch for ch in names if ch[0] not in
        "aeiouAEIOU"]

Print(lst1)
```

LIST COMPREHENSION

> List comprehension is an elegant way to define and create lists based on existing list

Syntax : [expression for item in list]

> It can identify when it receives a string or a tuple and work on it like a list.

> list comprehensions can utilize conditional statement to modify existing list (or other tuples).

1) Returns a list names starting with vowels in given string

```
lng = ["Laura", "steve", "bill", "James", "bob",
       "greg", "scott", "alex", "ive"]
lst = []
for ch in lng:
    if ch[0] in "aeiouAEIOU":
        lst.append(ch)
Print (lst)
```

using comprehension.

```
lst = [ch for ch in lng if ch[0] in "aeiouAEIOU"]
Print (lst)
```

3) create a list with words, if the words are of even length add it as it is, if it is odd length else reverse it.

```
String = "hello everyone it is thursday"
l = String.split()
l1 = []
for word in l:
    if len(word)%2==0:
        l1.append(word)
    else:
        l1.append(word[::-1])
Print(l1)
```

(or)

```
l2 = [word if len(word)%2==0 else word[::-1] for word in l]
print(l2)
```

4) create a list ~~with~~ if the number is even Square it and add else Cube it and add

```
l = [1, 2, 3, 4, 5, 6]
l1 = []
for i in l:
    if i%2==0:
        l1.append(i**2)
    else:
        l1.append(i**3)
Print(l1)
```

(or)

```
l2 = [i**2 if i%2==0 else i**3
      for i in l]
Print(l2)
```

COMPREHENSION

1) WAP to create a list of Square of number

```
l = [1, 2, 3, 4]
l1 = []
for i in l:
    l1.append(i**2)
Print(l1)
```

(or)

```
l1 = [i**2 for i in l]
Print(l1)
```

Syntax: [expression for item in list]

2) create a list of even number from 1 to 50

```
l = []
for i in range(1,51):
    if i%2 == 0:
        l.append(i)
Print(l)
```

(or)

```
l = [i for i in range(0,51) if i%2==0)
Print(l)
```

using defaultdict.

```
from collections import defaultdict
dd = defaultdict(list)

for element in items:
    w = element.split('-')
    value, key = w
    dd[key] += [value]      # dd[key].append(value)

print(dd)
```

2) Grouping even and odd numbers

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
from collections import defaultdict.
dd = defaultdict(list)

for i in n:
    if i % 2 == 0:
        dd[0] += [i]      # d[0].append(i)
    else
        d[1] += [i]       # d[1].append(i)
print(d)
```

```
n = [1, 2, 3, 4, 5, 6, 7, 8, 9]
from collection import defaultdict.
d = defaultdict(list)
for ele in n:
    rem = ele % 2
    if rem == 0:
        d[rem] += [ele]
    else:
        d[rem] += [ele]
print(d)
```

```
Print(max(l))
Print(l.sort())
Print(l)
Print(l[-1])
```

nth largest number

```
n = 5
for _ in range(n):
    for i in range(len(l)-1):
        if l[i] > l[i+1]:
            l[i], l[i+1] = l[i+1], l[i]
Print(l[-n])
```

1) Grouping flowers and animals in the below list.

```
items = ['lotus-flower', 'lilly-flower', 'cat-animal',
         'Sunflower-flower', 'dog-animal']
d = {}
for ele in items:
    word = ele.split('-')
    value, key = word
    if key not in d:
        d[key] = [value]
    else:
        d[key] += [value]
Print(d)
```

```
c = a + b
a, b = b, c
```

n^{th} fibonacci number.

```
a1, b1 = 0, 1
n=10
for i in range(n-1):
    a1, b1 = b1, a1+b1
Print(f"the {n}th fibonacci number is {a1}")
```

ii) to generate fibonacci series till the number 10

```
a2, b2 = 0, 1
n = 10
while a2 <= 10:
    print(a2, end = " ")
    a2, b2 = b2, a2 + b2
```

12) write a program to print the largest number in the given list

```
l = [98, 14, 62, 17, 56, 1, 5, 96]
max_ = 0
for i in l:
    if max_ < i:
        max_ = i
Print(max_)
```

2nd method

```
for i in range(len(l)-1):
    if l[i] > l[i+1]:
        l[i], l[i+1] = l[i+1], l[i]
Print(l[-1])
```

for i in range(len(l)-1):

comprehension.

```
d1 = {value : key for key, value in d.items()}
Print (d1)
```

9) convert 2 lists into a dictionary.

```
l1 = ["hello", "world"]
l2 = [10, 20]
d = {}
for key, value in zip(l1, l2):
    d[key] = value
print (d)
```

comprehension

```
d1 = {key : value for key, value in zip (l1, l2)}
Print(d1)
```

in range

```
d2 = {l1[i] : l2[i] for i in range (len(l1))}
print(d2)
```

10) write a program to print first 10 fibonacci numbers.

```
a, b = 0, 1
for i in range (0, 10):
    print (a, end=" ")
```

comprehension

```
d1 = {key : value[::-1] if isinstance(value, str)
      else value for key, value in d.items()}
print(d1)
```

7) create a dictionary with word and its count pair.

```
sentence = "hello hai how are you"
l = sentence.split()
d = {}
for word in l:
    d[word] = l.count[word]:
print d.
```

comprehension

```
d1 = {word : l.count[word] for word in l}
Print(d1)
```

8) write a program to flip the keys and values of the dictionary using dict Comprehension

```
d = {"a" : 1, "b" : "hello", "c" : 85, "d" : 12.3, "e" : (1,2,3)}
d1 = {}
for key, value in d.items():
    d1[value] = key
Print(d1)
```

5) ~~Build a list of~~

3) Create a dictionary with character and its ascii value.

```
S = "hello world"
d = {}
for ch in S:
    d[ch] = ord(ch)
Print(d)
```

comprehension.

```
d1 = { ch : ord(ch) for ch in s }
print(d1)
```

6) write a program to reverse the values in the dictionary if the value is of type string

```
d = { "a" : 1, "b" : "hello", "c" : 85, "d" : 12.3,
      "e" : [1,2,3] }
d1 = {}
for key, values in d.items():
    if isinstance (value, str):
        d1[key] = value[::-1]
    else:
        d1[key] = value.
Print(d1)
```

FUNCTION ANNOTATIONS.

↳ These are just conventions and this is only recommended not mandatory.

```
def add(a:int, b:int, c:int)->None:
    print(a+b+c)

print(add(1,2,3))   #6
add("hai","hello","world")
```

» Annotations are only type hints.
But it does not enforce type check!

```
def add(a:int,b:int)->int:
    return a+b
```

VARIABLE SCOPES

GLOBAL SCOPE VARIABLE →

* To update the global variable in local space is called global variable.

```
a=1
b=2
def add():
    global a
    a=a+b
    return a

Print(a)
Print(add())
Print(a)
```

→ VARIABLE NAME OF POSITIONAL ARGUMENTS

```
def function1(*args): # Packing.
    print(args)
    print(*args)  # 123 "hello"

function1(1, 2, 3, 'Hello')   → Return tuples.
                                # (1, 2, 3, "hello")
function1(1)                    # (1,)
function1()                     # ()
```

→ VARIABLE NUMBER OF KEYWORD ARGUMENTS.

```
def function(**kwargs):   # packing
    Print(kwargs)
```

→ Return dictionary

↓

but we cannot unpack the dictionary.

```
function at (a=1, b=2, c=3)
    print(*kwargs) - # abc
    Print(**kwargs) # Typeerror
```

→ (2 line Gap)

DEFAULT PARAMETER

Default Values indicate that the function argument will take that value if no argument value is passed during function call.

- The default value is assigned by using assignment (=) operator.

Example

```
def add(a, b, c = 0):
    print(a+b+c)

add(1, 2)      # 3
add(1, 2, 3)   # 6
```

* def greet (name, /, *, age):
    Print(f"My name is {name} and I am {age} years old")

greet ("Steve", age=23)

TYPES OF ARGUMENTS

calling a function using.

⇒ Positional arguments.
> Keyword arguments.
> Combination of both positional and Keyword arguments.
> Keywords and positional only arguments.
> Variable positional arguments - *args.
⇒ Variable Keywords arguments - **kwargs.

Note : / → is used to mention or denote that the parameter present before it must accept only positional arguments. (or) else it will throw type error

* → is used to denote that the parameters present after it must accept only Keyword arguments or else type error will be raised.

* * → is used in function ⇒ definition → packing.
* → is used in Calling function ⇒ unpacking.

greet("John", age=18)

greet(age=18, "John") → Syntax error

Positional arguments only

→ Parameters defined before Slash must be positional arguments

```
def greet(name, age, /):
    print(f"My name is {name} and I am {age} years old")

greet("John", 20)
greet(name="John", age=18) ← (It should throw error because here we are passing only positional arguments)
```

✓

```
def greet(name, /, age):
    Print(f"My name is {name} and I am {age} years old)

greet("Steve", 23)
greet("Steeve", age=23)
greet(name="Steve", age=23) # type error
```

Keyword arguments Only

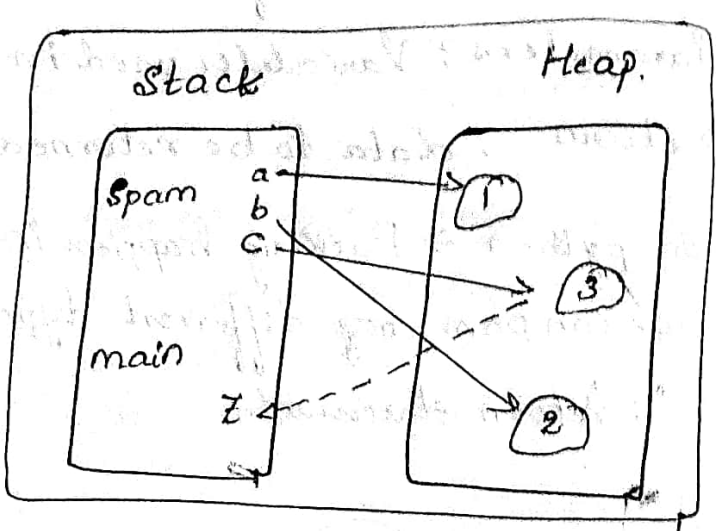
~~def greet(name, age)~~ → after star it should be keyword arguments only.

```
def Greet(*, name, age):
    Print(f"My name is {name} and I am {age} years old)

greet("Steeve", 23) → Type error.
greet(name="steeve", age=23)
```

Memory Management in functions.

```
def Spam(a, b):
    c = a + b
    return c
z = Spam(1, 2)
```

RAM



Positional arguments.

```
def greet(name, age):
    Print(f"My name is {name} and I am {age} years old")

greet("John", 18)
greet(18, "John")
```

Keyword arguments.

```
def print(values, end="\n", sep=" "):
greet(name="John", age=18)
greet(age=18, name="John")
```

combination positional and Keyword arguments.

always → first we should positional arguments.
and then only Keyword arguments

↓

If we change it will throw "Syntax error"

return data

func_name (arguments) ← function call

func_name : name of the function.

Parameters : Variables used in the function

return : data to be returned to the Caller.

* In python → Packing happens in tuple.
* we can pass different type of data types. in return statement.

Return statement.

* Statement used to get or return the values from a function to the Caller.
* The return statement is used to exit a function and go back to the place from where it was called.
* The statements after the return statements will not be executed once return statement is executed.
* It can return Single or multiple data.

> Syntax : return data.

or

return data1, data2 → returns tuple.

FUNCTIONS - OVERVIEW

> A function is a block of code which only runs when it is called.

> Function help break our program into smaller chunks.

- As a program grows larger and larger, functions make it more organized and manageable

> Function avoids repetition and makes the code reusable.

examples.

1)
```
def add(a,b):
    c=a+b
    return c

add(1,2)        # nothing will be printed.
Print(add(1,2)) # 3
```

2)
```
def add(a,b):
    Print(a+b)

add(1,2)          # 3
Print(add(1,2))   # 3, None.
```

Creating a function:

Syntax:
```
def func_name(parameters):   ← function definition/declaration
    Statement1
    Statement2
    .....
```

2) write a function to print message "Hai everyone" if the user input is not present and if the user input is present print the user input.

```
def greet (msg = "hai everyone"):
    Print(msg)

greet()
greet("hello good morning")
```

3) A function takes variable number of positional arguments as input. how to check if the arguments that are passed are more than 5

```
def length_(*args):
    Print(len(args)>5)
length_("Hello hai Everyone")
```

Sys.executable.

Sys.getrecursionlimit()

Sys.setrecursionlimit()

ex:-

```
count = 0
def greet():
    global count
    if count == 3:
        return
    print('hello')
    count += 1
    greet()
greet()
```

```
count = 0
def greet(a=count):
    print(a)
count = 1
greet()
```

greet(a=0) (count)

Print(a)

```
def len_(iterable):
    count = 0
    for _ in iterable:
        count += 1
    return count.
print(len_("Hello world"))
```

3) To search for a character in a given string and return the corresponding index.

```
def linear_search (seq, value):
    for index, item in enumerate(seq):
        if value == item:
            return f"{value} is present in position {index}"
    else:
        return f"{value} is not present in {seq}"

Print( linear_search ("hello", "z"))
```

s = "hello world"
ch = "0"
for ch in s:
print[i]

using inbuilt method

```
def linear_search (sequence, value):
    return sequence.index (value)
```

> write a function to count the number of positional and keyword arguments passed to it.

```
def sample (*args, **kwargs):
    Print (len (args))
    Paint (len(kwargs))
sample (1, 2, 3, 4, a=10, b=20, C=30)
```

NON-LOCAL SCOPES.

> NON-local Scopes are those which are neither local nor global.

> If a nested function has to access and modify the Variable of outer function, nonlocal Keyword should be used.

Example

```
def Spam():
    a=10

    def inner():
        non local a
        a+=1
```

1) write a function to return dictionary with character and its ascii value pair.

```
def char_ascii():
    pass
```

```
S = "Hello world"
d = { }
for ch in S:
    d[ch] = ord(ch)
print(d)
```

```
def char_ascii(string):
    d={char:ord(char)
       for char in string}
    return d.

Print(char_ascii("Hello"))
```

2) write a function to return length of any iterable without using any inbuilt function.

Local Scope Variable

```
def Spam():
    a = 2
    def wrapper():
        nonlocal a
        a = a + 10
        print(a)
    wrapper()
Spam()
```

LOCAL VARIABLE

A variable which is created inside a function is called a local Variable and it cannot be accessed outside the function.

Example :

```
def function():
    a = 10          # local Variable
```

GLOBAL VARIABLE

> A Variable created in global namespace i.e., outside a function.
> It can only be accessed inside a function but cannot be modified.
> In order to modify a global Variable inside a function then we need to use global Keyword.

Example :

```
a = 10
def func():
    global a
    a += 1
```

```
def last-digit(num)
    return num%10

res = last-digit(432)
print(res)
```

***

```
Last-digit = lambda num : num%10
res = last-digit(60123)
Print(res)
```

LAMBDA EXPRESSIONS

> An anonymous function is a function that is defined without a name.

> While normal functions are defined using the def keyword in Python, anonymous functions are defined using the lambda keyword.

> Syntax : lambda [arg1 [,arg2 ... argn]] : expression.

Normal function.

```
def add(a,b):
    return a+b # Single expression function.

Print add(1,2))
```

using lambda function

```
add = lambda a, b : a+b

print(add(1, 2))
```

2). Create a function named tail that takes a sequence and a number 'n' as input and returns the last 'n' elements from the given sequence as a list

```
def tail(sequence, n):
    return list(sequence[-n:])

print(tail("hello", 2))
```

3) write a function to check if the given number is fibonacci number.

```
def fib(number):
    a=0
    b=1

    while a <= number:
        if a == number:
            return f"{number} is a fibonacci number"

        c = a+b
        a,b = b,c
    else:
        return f"{number} is not a fibonacci number"

Print(file(19))
```

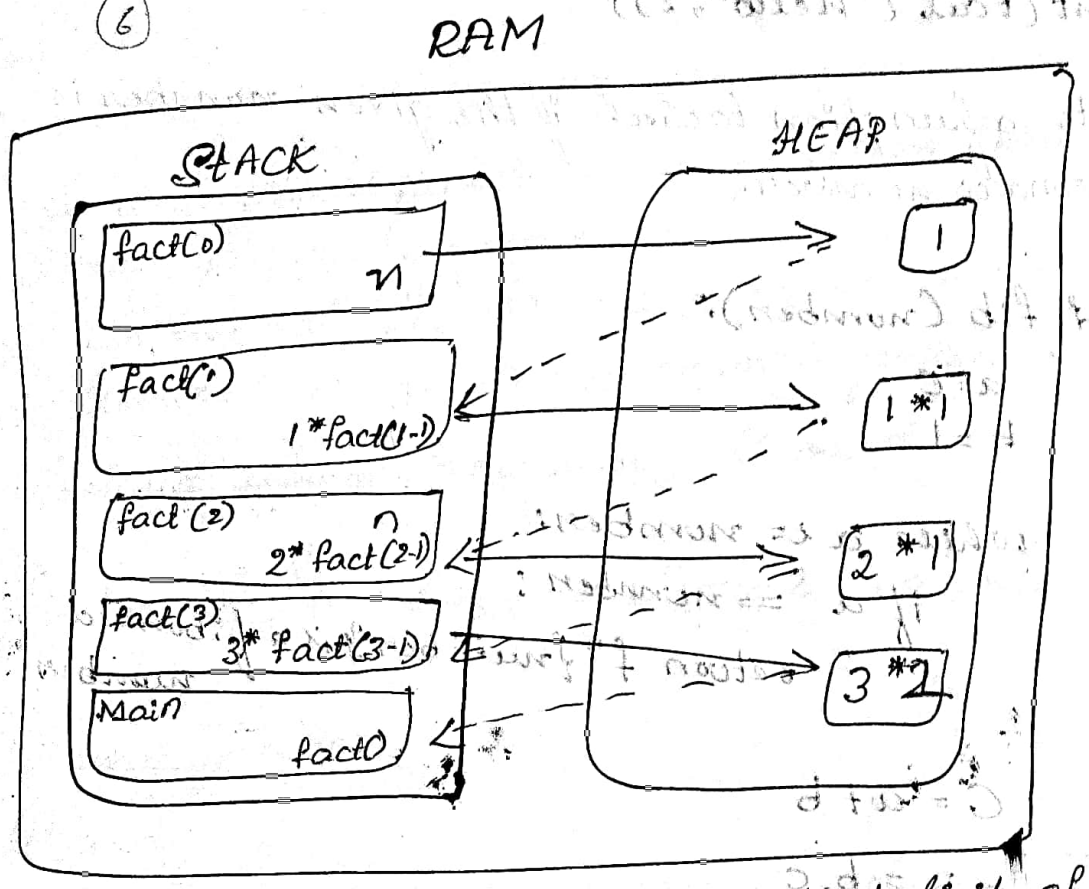
LAMBDA EXPRESSIONS / ANONYMOUS FUNCTIONS

* Memory Efficiency and Code Optimization.

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)
```

fact(3)

(6)



1) Write a function that returns last digit of an integer.

```
def last_digit(num):
    return num%10

Print(last_digit(1,2,3))
```

RECURSION

> It is function which calls itself until the condition is being satisfied.

> Used to avoid loops in some cases

> Python recursion limit can be found out with a function from the sys module called getrecursionlimit():

```
>>> from sys import getrecursionlimit
>>> getrecursionlimit()
1000
```

We can change it with setrecursionlimit():

```
>>> from sys import setrecursionlimit
>>> setrecursionlimit(2000)
>>> getrecursionlimit()
2000
```

Recursive functions typically follow this pattern:

> There are one or more base cases that are directly solvable without the need for further recursion.

> Each recursive call moves the solution progressively closer to a base case

```
def greet (Count = 0):          ← Print hello for
    if count == 3:                3 times.
        return

    Print ("hello")
    Count += 1

    greet (Count)

greet()
```

1) write a recursion function to print numbers from 1 to 10

```
def count_(number = 1):
    if number > 10:
        return.

    Print(number)
    number += 1
    Count_(number)

Count_()
```

2) write a recursion function to print factorial of a number.

```
def factorial (n, fact = 1, i = 1):
    if i > n:
        return fact

    fact *= i
    i += 1
    return factorial (n, fact, i)
Print ( factorial (5))
```

n = 5
fact = 1
i = 1
while i <= n:
    fact = fact * i
    i += 1
Print(fact)

Write a program to raise the elements of the list to the power of their indices.

$l_1 = [1, 2, 3, 4, 5]$

```
def power_(value):
    index, ele = value
    return ele**index
lambda value: value[1]**value[0]
map(power_, enumerate(numbers))
```

```
def power(value, index):
    return value**index
lambda value, index: value**index
print(list(map(power, numbers, range(len(numbers)))))
```

map will calculate each and every element and apply a functionality

lambda will accept single element.

1) `numbers = [1, 2, 3, 4, 5, 6, 7, 8]`

```
def even_(num):
    if num % 2 == 0:
        return num
print(list(map(even_, numbers)))
print(list(filter(even_, numbers)))
```

5) Write a program to convert the negative numbers into positive in the given list

```
numbers = [-9, 2, 3, -4, 8, -2, 7, -12]
print(list
(map(abs, numbers)))
```

6) write a program to return the word and its length pair in the given string.

```
Sentence = "This Is a Bunch of WORDS"
words = Sentence.split()
#normal function
def word_len(word):
    return word, len(word)
#lambda function
word_len_pair = lambda word: (word, len(word))
Print(list(map(word_len, words)))
```

7) To add the elements of two lists.

```
l1 = [1, 2, 3, 4]
l2 = [9, 5, 4, 6]
          num
def add_(num):
    return num[0]+num[1]
map(add_, zip(l1, l2))
# [(1,9), (2,5), (3,4), (4,6)]
```

```
add_=
#lambda num:
    num[0]+
    num[1]
```

```
def add_(n1, n2):
    return n1+n2
Print(list(map(add_, l1, l2)))
```

```
add_=
lambda n1, n2: n1+n2
```

```
l = list(range(10))
even_odd = lambda num: "even" if num % 2 == 0 else "odd"
res = map(even_odd, l)
Print(list(res))
```

2) check if the string is palindrome or not in the given list

```
list_ = ["madam", "dad", "hello", "google", "level"]
palindrome = lambda string: string == s[::-1]
Print(list(map(palindrome, list_)))
```

3) WAP to convert the string present in the list to upper case.

```
list_ = ["madam", "dad", "hello", "google", "level"]
upper_ = lambda word: word.upper()
print(list(map(upper_, list_)))
```

(or)

```
Print(list(map(str.upper, list_)))
```

4) write a program to swap the case of the words in the given sentence

```
Sentence = "This IS a Bunch Of wORds"
words = Sentence.split()
Print(list(map(str.swapcase, words)))
```

↓

```
Print(" ".join(res))
```

want output in string

1) write a lambda expression to check if the given string is palindrome or not.

```
Palindrome = lambda string : string == string[::-1]
Print (palindrome("mom"))
```

```
l=[1, 2, 3, 4, 5]
res =[]
sq = lambda num: num**2
res = map (sq, l)
print (list (res))
```

MAP()

Map applies a function to all the items in an input_list.

Syntax : map (function_to_apply, list_of_inputs)

> Map() can be applied to more than one list but the lists should have the same length.

> MAP() will apply its lambda function to the elements of the argument lists, i.e, it first applies to the elements with the 0th index, then to the elements with the 1st index until the nth index is reached.

1) write a program to check for even and odd numbers in the given list. using map.

Lambda expression to return square of a num.

```
Squares = lambda num: num**2
Print(Squares(5))
```

2) Lambda expression to return last element of any sequence.

```
last_ = lambda sequence : sequence[-1]
Print(last_("hello"))
```

4) Lambda expression to return last n elements of any sequence.

```
last_n = lambda seq, n: seq[-n:]
Print(last_n("hai hello", 3))
```

5) Lambda expression to check if the given number is even or odd.

```
even_odd = lambda num: f"{num} is even" if
           num % 2 == 0 else f"{num} is odd"
Print(even_odd(4))
```

6) Write a lambda expression to return Square and Cube of the given number.

```
Sq_cube = lambda num: (num**2, num**3)
print(sq_cube(4))
```

'FROM' Keyword.

> It is used when we need to import some module from different package

Syntax

from module_name import function

or

from package_name import module_name

or

from package_name.module_name import function

"as" Keyword.

> used to give an alias name for the module as well as function.

> Syntax

from module_name import function as alias_name

or

from package_name import module_name as alias_name

or

from package_name.module_name import function as alias_name

SORTED()

Sorted (iterable, *, key = None, reverse = False)

> Returns a new sorted list from the items in iterable.

> Has two optional arguments which must be

* program to return only positive values in the list using filter class.

```
numbers = [-2, -1, 0, 1, 2]

Pos = lambda num: num > 0

Print(list(filter(pos, numbers)))
```

None, false, 0 } default values

MODULES

> A module is a file consisting of python code.
> It can have functions, classes and variables.
> Any python file can be referenced as a module.

TYPES OF MODULES

User defined modules.
> created by user.
> can be accessed using import statement
> Syntax : import module_name.

Built in modules.
> Pre defined modules
> to install the module pip command can be used
> pip install module_name

IMPORT STATEMENT

> used to import modules in any programs
> It can be written anywhere in the module.
> Any number of modules can be imported in a module.

Syntax : import module-name

Print(list(filter(check_len, names)))

* write a program to build a list with only even length string using filter class.

```
names = ['apple', 'google', 'Yahoo', 'facebook', 'Yelp'
         'flipkart', 'gmail', 'Instagram']

def check_len(word):
    if len(word) % 2 == 0:          # return len(word) % 2 == 0
        return word.

Print(list(filter(check_len, names)))

even = lambda string : len(string) % 2 == 0
```

* Return the String if the string is starting with a vowel character"

```
names = ['laura', 'steve', 'bill', 'james', 'bob'
         'greig', 'scott', 'alex', 'ive']

vowels = lambda string : string[0] in
                         "aeiouAEIOU"

Print(list(filter(vowels, names)))

# vowels = lambda string : string[0].
                           lower()
                           in
                           "aeiou"
```

FILTER()

- The filter() function in python takes in a function and an iterable as arguments.
- This offers an elegant way to filter out all the elements of a sequence, for which the function returns true.
- Filter creates a list of elements for which a function returns true.

Syntax: filter(func, iterable)

Keypoints.

- unlike map(), only one iterable is required.
- The func argument is required to return a boolean type. if it doesn't, filter simply returns the iterable passed to it.
- Also, as only one iterable is required, it's implicit that func must only take one argument.
- filter passes each element in the iterable through func and returns only the ones that evaluate to true.

> write a program to extract the names whose length is greater than 5.

```
names = ["Python", "Programming", "Java", "Manual"]

def check_len(name):
    if len(name) > 4:          # return len(name) > 4
        return name
```

else:
    d[key].append(word) # d[key] += [word]

PATTERN PRINTING.

ex

```
"""
*
**
***
****
*****
"""
for row in range(5):
    for col in range(5+row):
        print("*", end="")
    print()
```

```
Printl)
for row in range(1,6):
    print("**" * row)
```

---

double of end index
or double of last line of star.

```
for row in range(1,6):
    Print(f"{'*'*row:^12}")
```

Equilateral triangle

```
for row in range(1,6):
    Print(f"{'*'*row:^10}")
```

Logic tracing.

```
    *
   * *
  * * *
 * * * *
* * * * *
```

write a program to print all the maximum values present in the list.

```
numbers = [7,4,9,2,8,9,6,4,5,9]
largest = max(numbers)
for num in numbers:
    if largest == num:
        print(num)
```

Print all the shortest words in the sentence.

```
Sentence = "hai, how are you doing today"
word = Sentence.split()
w = min(word), key = len)
```

or.

```
words = sorted(Sentence.split(), key = len)
min_, *rest, max_ = words.

for word in Sentence.split():
    if len(min_) == len(word):
        Print(word)
```

write a program to group anagram.

```
words = ['eat', 'silent', 'ate', 'hello', 'listen', 'tea']
d = {}
for word in words:
    key = "".join(sorted(word))
    if key not in d:
        d[key] = [word]
```

S = Sorted (dial_codes.items(), Key = lambda item : item[-1][-1])

6) write a program to find the longest word in the given sentence.

Sentence = "Python is a programming language and programming is fun"

words = Sentence.split()

Sorted (words, Key = len = S_words.

or)

min_, *rest, max_ = S_words.

Shortest, *rest, longest = S_words.

unpacking.

l = [1, 2, 3, 4, 5]

a = l[0]

b = l[1]

a, b = l[0], l[-1]

---

a, *c, b = l

a, b, *c = l

*a, c, *b → TypeError.

7) write a program to print a pair of longest non-repeated word and its length pair.

Sentence = "Python is a programming language and programming is fun"

words = Sentence.split()

~~S_words = Sorted(words, Key = le~~

d = {word: len(word) for word in words if words.count(word) == 1}

S_words = Sorted (d.items(), Key = lambda item: item[-1])

min_, *rest, max_ = S_words

2) Sort the dictionary based on the Value.

prices = {'ACME': 45.23, 'AAPL': 612.78,
'IBM': 205.55, 'HPQ': 37.20, 'FB': 10.75}

Print(sorted(prices.values())) → only values.

Print(sorted(prices.items(), key = lambda item:
item[-1]))

3) Sort the dictionary based on the length of the Key.

Prices = {'ACME': 45.23, 'AAPL': 612.78, 'IBM': 205.55
'HPQ': 37.20, 'FB': 10.75}

Print(sorted(prices.items(), key = lambda item:
or len(item[0]))

S = ——

4) Sort the dictionary based on the length of the Value.

dial_codes = {86: 'china', 91: 'India', 1: 'unitedstates'
62: 'Indonesia', 55: 'Brazil'}

S = sorted(dial_codes.items(), key = lambda
items: len(item[-1]))

5) Sort the dictionary based on last character of the values.

dial_codes = {86: 'china', 91: 'India', 1: 'unitedstates'
62: 'Indonesia', 55: 'Brazil'}

```
def firstchar(string):
    return string[0]
Snames = sorted(names, key = first_char)
```

(or)

```
f_char = lambda string : string[0]
```

4) Sort the list based on the last character of each element.

```
names=['apple', 'google', 'Yahoo', 'amazon', 'facebook',
    'instagram', 'microsoft', 'zomato']
Snames = sorted(names, key = lambda string
                    : string[-1])
```

5) Sort the list based on the first element of each tuple.

```
names = [('facebook', 8), ('apple', 5), ('instagram', 9)
         ('google', 6), ('Yahoo', 5)]
Print
S1 = sorted(names)
S = sorted(names, key = lambda element:
                    element[0])
```

6) Sort the dictionary based on the key.

```
Prices = {'ACME': 45.23, 'AAPL': 612.78, 'IBM':
          205.55, 'HPQ': 37.20, 'FB': 10.75}
Print(sorted(Prices))          #Only Keys List
Print(sorted(Prices.items()))  # both keys and values.
Print(sorted(price.items(), key = lambda item:
                    item[0]))
```

4) # Tuple

```
names = ("google", "amazon", "gmail", "walmart"
         "flipkart", "microsoft", "apple")

S_names = Sorted (names)
S_names = Sorted (names, reverse = True)
S_names = Sorted (names, reverse=true, key=len)
```

1) Write a program to sort a list in reversed order.

```
names = ["google", "amazon", "gmail", "walmart",
         "flipkart", "microsoft", "apple"]

Sorted (names, reverse = True)
↓
S_names =
```

2) Sort the list based on the length of the element.

↓

custom sorting.

```
names = ["google", "amazon", "gmail", "walmart",
         "flipkart", "microsoft", "apple"]

S_names = Sorted (names, key = len)
```

3) Sort the list based on first character of each element.

```
names = ['apple', 'google', 'Yahoo', 'amazon', 'facebook',
         'instagram', 'microsoft', 'zomato']
```

1st method - default sorting.

```
S_names = Sorted (names)
```

2nd method custom sorting

specified as keyword arguments.

* Key specifies a function of one argument that is used to extract a comparision key from each element in iterable (for example, key = str.lower) The default value is None (compare the elements directly).

* Reverse is a boolean value. if set to true, then the list elements are sorted as if each comparision were reversed.

Examples

1) # String.

word = "python"

Sorted_word = sorted (word) #list

Sorted_word = sorted (word, reverse = True)

2) # list

names = ["google", "amazon", "gmail", "walmart", "flipkart", "microsoft", "apple"]

S_names = sorted (names)

S_names = sorted (names, reverse = True)

S_names = sorted (names, reverse = True, key = len)

3) # Set

names = {"google", "amazon", "gmail", "walmart", "flipkart", "microsoft", "apple"}

S_names = sorted (names)

S_names = sorted (names, reverse = True)

S_names = sorted (names, reverse = True, key = len)

)S.path.getsize():

> In this method, Python will give us the size of the file in bytes.

> To use this method we need to pass the name of the file as a parameter.

FILES

> Files are named locations on disk to store related information.

> when we want to read from or write to a file, we need to open it first.

> When we are done, it needs to be closed so that the resources that are tied with the file are freed.

> Hence, in python, a file operation takes place in the following order.

* Open a file
* Read or write (perform operation)
* close the file.

Opening files in python.

> In python, files can be opened in two ways,

* Without Context manager

```
file_obj = open(filename, mode)
```

* with Context manager.

```
with open(filename, mode) as file_obj:
    Pass
```

Statistics.

path

Files.

os.popen (r" Path)

os.remove(r" path)

os.rename("old","new")

Methods related to files.

1) popen(file_name, mode): similar to open(), provides a pipe / gateway and accesses the file directly.

2) rename(old_name, new_name): renames the file with new name.

3) remove(): used to remove or delete a file path. This method can not remove or delete a directory. if the specified path is a directory then OS Error will be raised by the method.

Path

Print(os.path.exists(path)

Print(os.path.getsize(path

Path related methods.

1) os.path.exists():

> This method will check whether a file exists or not by passing the name of the file as a parameter.

> os module has a sub-module named PATH by using which we can perform many more functions.

```
S = "₹50,000/-"
res = ""
for char in S:
    if char.isdigit():
        res += char.x2
a = int(res)+2
Print(a)
```

FILEHANDLING.

```
import os
#directories.
Print(os.getcwd())
Print(os.chdir(r"C:\Users\Path"))
OS.mkdir('Demo')
Os.rmdir("Demo")
Print(OS.listdir())
```

Methods related to directory

1) getcwd(): get current working directory.
2) mkdir(dir_name): Creating a new directory
3) Chdir(dir_name): changing the directory.
4) rmdir(dir_name): removing directory.
5) listdir(): used to get the list of all files and directories in the specified directory.
If we don't specify any directory, then the list of files and directories in the current working directory will be returned.

```
Pat = ""
start = "a"
for row in range(4):
    var = chr(ord(start) + row)  # 97 + 0 = 97 = "a"
    Pat = Pat + var + "" # "" + "a" => "a", "a" + "b" = "ab"
    Print(Pat)
```

```
Pat = ""
start = "a"
end = "d"
for row in range(ord(start), ord(end + 1)):
    Pat = Pat + chr(row) + " "
    Print(Pat)
```

```
Pat = ""
for i in range(1, 6):
    Pat = Pat + str(i)
    Print(f"{Pat:>5}")
```

```
def prime_(n):
    return not any(C[n % i == 0 for i in range(2, n)])
Print(Prime_(9))
```

↳ any is a inbuilt function that returns true if any one item in iterable evaluates to boolean true.

- EVEN and Odd

```
for row in range(1,5):
    print("*")
    print("* " * row)
```

```
*
*
*
* * *
```

INVERTED TRIANGLE

```
for row in range(5, 0, -1):
    print("* " * row)
```

```
for row in range(5,0,-1):
    print(f'{"* " *row:>10}')
```

```
for row in range(5,0,-1):
    print(f'{"* " *row^10}')
```

Diamond pattern.

```
for row in range(1,6):
    print(f'{"* " *row:^10}')

for row in range(4, 0,-1):
    print(f'{"* " *row:^10}')
```

alphabet pattern printing.

```
for row in range(1,5):
```

6) count the number of words present in sample.txt

```
with open("sample.txt") as file:
    word_count=0
    for line in file:  ⟶ if line.strip():
        words=line.split()
        for word in words:
            word_count +=1
print(word_count)
```

(or)

```
with open("sample.txt") as file:
    word_count=0
    for line in file:  ⟶ if line.strip():
        words=line.split()
    word_count += len(words)
    Print(word_count)
```

7) write a program to print length of each line along with the line in a file.

```
with open("sample.txt") as file:

    for line in file:
        if line.strip():
            print(line, len(line))
                    # print(len(line), line)
```

8) write a program to create a dictionary with words and its count pair in a file.

```
with open("sample.txt") as file:
word_count={}
    for line in a file:
        if line.strip():
        words = line.split()
```

1) Write a program to read the contents of a file without loading a file into memory.

```
OS.chdir (r" path")
with open ("Sample.txt") as file:
    for line in file:
        print (line)
```

2) write a program to print line number along with the line in Sample.txt file

```
with open ("Sample.txt") as file:
    for line_no, line in enumerate (file, start=1):
        print (line_no, line, sep="--")
```

3) Print only non blank line in the file Sample.txt

```
with open ("Sample.txt") as file:
    for line in file:
        if line.strip():  #check empty lines.
            Print (line)
```

4) write a program to read the file in reversed order.

```
with open ("Sample.txt") as file:
    for line in reversed (list (file)):
        Print (line)
```

5) count the number of lines in the file Sample.txt

```
with open ("Sample.txt") as file:
    Count = 0
    for _ in file:
        Count += 1
print (Count)
```

Methods to read from a file:

1) read(): reads the data from starting to end of the file.

```
file = open("demo.txt", "r")
print(file.read())
```

Note: read() can have one argument which is an integer. It specifies the number of characters to be read from the file from the starting.

2) readline(): reads a single line from the file.

```
file = open("demo.txt", "r")
print(file.readline())
```

3) readlines(): returns entire text in the form of list seperating each line as an element

```
file = open("demo.txt", "r")
Print(file.readlines())
```

Example

* with open(f_path) as file:
  #traversing through a file by loading one line into memory
  for line in file:
      print(line)

* with open(f_path) as file:
  # traversing through one line at a time in an iterator(lazy iterable)
  Print(next(file))

File object Attributes.

1. file.closed : Returns true if file is closed, false otherwise.
2. file.mode : Returns access mode with which file was opened.
3. file.name : Returns name of the file.
4. file.readable(): Return true if file is opened in read mode.
5. file.writable(): Return true if file is opened in write mode.
6. file.close(): closes the file.

reading from a file

```
with open(f_path) as file:
    Print(file.read())      # entire file as a string
    Print(file.readline())  # one line at a time as a string.
    Print(file.readlines()) # entire file in list where lines will be separate elements.
```

Reading from files.

> By default files will be opened in "r" mode.

> Syntax: file = open("demo.txt", "r")

or

```
with open("demo.txt") as file:
```

iterator

Here file will be automatically closed once the control comes out of the with block.

opening a file

```
f_path = r" path.

# without context manager

f_obj = open (f_path)
print (f_obj.closed)  # false.
f_obj.close()
print (f_obj.closed)  # true

# with Context manager.

with open (f_path) as f:
    print ("inside with block", f.closed) #false
print ("outside With block", f.closed) #true.
```

modes:

There are four different methods (modes) for opening files.

"r" - Read - Default value. opens a file for reading, error if the file does not exist.

"a" - Append - opens a file for appending. creates the file if it does not exist.

"w" - Write - opens a file for writing, creates the file if it does not exist (overrides the content of file if already exists)

"x" - Create - creates the specified file, returns an error if the file exist.

Note [r+, w+, a+, x+ → both read and write operations can be done on the file.

```
        if line_no == n:
            print(line)

# Read first 6 lines
n=6
with open("access-log.txt") as file:
    for line_no, line in enumerate(file, start=1):
        if line_no <= n:
            print(line)
```

ISLICE

islice is a class in itertools modules which is used to perform slicing on any iterables. (collections, iterator objects like enumerate object, zip object, reversed and file objects etc)

Syntax

```
From itertools import islice
Object = islice(iterable, start, stop, step)
    (or)
    islice(iterable, stop)
```

Example

1) Read n^{th} line from a file.

```
n=15
with open("access-log.txt") as file:
    for line_no, line in enumerate(file, start=1):
        if line_no == n:
            print(line)

# Islice
```

```
if words[0] not in d:
    d[words[0]]=1
else:
    d[words[0]]+=1
```

Sorted (d.items(), key = lambda item : item[1])

min_ , *rest , _max.

Print(_max)

(OR)

```
with open("access-log.txt") as file:
    l=[]
    for line in file:
        if line.strip():
            words=line.split()
            l.append(words[0])
```

C = Counter(l)

Print(c) # Counter object → dictionary of element of its count pair in descending order.

Print(C.most_common()) # list of tuples of element and its count pair

Print(C.most_common(n)) # list of tuple of the n maximum repeated elements (n → integer)

1) Read nth line from a file.

```
n=15
with open("access-log.txt") as file:
    for line_no, line in enumerate(file, start=1):
```

```
with open("access-log.txt") as file:
    d = {}
    for line in file:
        if line.strip():
            words = line.split()
            if words[0] not in d:
                d[words[0]] = 1
            else:
                d[words[0]] += 1

    print(d)
```

(or)

```
#default dictionary
with open("access-log.txt") as file:
    dd = defaultdict(int)
    for line in file:
        If line.strip():
            words = line.split()
            dd[words[0]] += 1

    Print(dd)
```

* write a program to print the most occured IP address in the access-log file.

```
with open("access-log.txt") as file:
    d = {}
    for line in file:
        if line.strip():
            words = line.split()
            for word in words:
```

```
For word in words.  -# traversing through the word in a line
    d[word]=0
    if word not in word_count:
        word_count[word]=1
    else:
        word_count[word]+=1
```

(or)

```
From collections import defaultdict
with open("Sample.txt") as file:
    d= default dict(int)
    for line in file:
        If line.strip():
            For word in line.split():
                d[word]+=1
Print(d)
```

* write a program to extract the IP addresses from accesslog file.

```
with open("access-log.txt") as file:
    for line in file:
        if line.strip():
            words = line.split()
            Print(words[0])
```

* write a program to create a dictionary with IP addresses and its count pair

write(): writes the data into the file.

> Data must be a string

> Returns the number of characters written in the file.

writelines(): write an iterable data into file. (list, tuple or dictionary)

> Data must has string as its elements.

> Returns none.

WRITING INTO FILES

> To write we need to use "w" mode.

> Syntax: file = open("demo.txt", "w")

(or)

with open("demo.txt", "w") as file:

> Functions to perform write operation

* write()
* writelines()

Examples write(), writelines()

```
import os
os.chdir(r"path")
Print(os.getcwd())
with open("example.txt", "a") (or) "w" as file:
    print(file.write("tomorrow is holiday\n"))
    file.writelines({"apple\n", "is good for health\n", "orange\n"})
with open("example.txt", "r+") as file:
    file.write("all the best")
    for line in file:
        print(line)
```

with Keyword arguments.

C=counter (A=3, B=5, C=2) → {'B':5, 'A':3, 'C':2}

Deque (Doubly Ended Queue) is the optimized list for quicken append and pop operations from both sides of the container.

Syntax : deque (list)

deque.

```
with open ("Sample.txt") as file:
    lines = deque (file, n)
    Print (list (lines))
```

```
S = "AABBCCCDAA CDA"
C = 1
a = ""
for i in range (len(S) - 1):
    if S[i] == S[i+1]:
        C += 1
    else:
        a += str(C) + S[i]
        C = 1
Print (a)
```

Read last n lines from a file.

```
n=4
with open("access-log.txt") as file:
    lines_count = 0

    for _ in file:
        lines_count += 1

    file.seek(0)
    res = islice(file, lines_count-n, lines_count)
    print(list(res))
```

Note ⇒ DEQUE

COLLECTION MODULE

1) Counters.

2) A counter is a subclass of the dictionary.

* It is used to keep the count of the elements in an iterable in the form of an unordered dictionary where,
The key represents the element in the iterable and value represents the count of that element in the iterable

1) Syntax to create a counter object.
From collections import Counter

2) with sequence of items.
C = Counter(['B','B','A','B','C','A','B','B','A','C']))
output ⇒ {"B":5, "A":3, "C":2}

3) with dictionary
C = Counter({'A':3, 'B':5, 'C':2}) → {'B':5, 'A':3, 'C':2}

```
with open ("access-log.txt") as file:
    lines = islice (file, n-1, n)
    Print (list (lines))
```

2) Read first n lines.

```
n=6
with open ("access-log.txt") as file:
    for line_no, line in enumerate (file, start=1):
        if line_no <= n:
            print(line)
```

Using islice

```
with open ("access-log-txt") as file:
    lines = islice (file, n)
    print (list (lines))
```

3) write a program to read 10th to 15th lines.

```
Start=10
End=15
with open ("access-log.txt") as file:
    lines = islice (file, start-1, end)
    for line in lines:
        print(line).
```

#enumerate

```
with open ("access-log.txt") as file:
    for line_no, line in enumerate (file, start=1):
        if 10 <= line_no <= 15:
            Print (line)
```

```
    count = 0
    for line in file:
        count += 1
with open("sample.txt") as file:
    for line_no, line in enumerate(file):
        if count - n <= line_no <= count:
            print(line)
n_lines(3)
```

(or)

```
def n_lines1(n):
    with open("sample.txt") as file:
        count = 0
        for line in file:
            count += 1
        file.seek(0)
        lines = islice(file, count-n, count)
        return(list(lines))
Print(n_lines1(3))
```

4) write a program to print longest and non repeated word in the sentence,"

```
s = "See and saw went to see a sea"
l = s.split()
l1 = []
for i in l:
    if l.count(i) == 1:
        l1.append(i)
Print(sorted(l1)[-1])
```

(or)

```
l1 = [i for i in l if
      l.count(i) == 1
print(sorted(l1)[-1])
```

MID ASSESMENT

1) write a program to print only even lines in a file (consider filename as Sample.txt)

```
import os
from itertools import islice
from Collections import defaultdict.

OS.chdir (r"C:path")
def even_lines():
    with open ("sample.txt") as file:
        for line_no, line in enumerate (file):
            if line_no % 2 == 1:
                print(line)
even_lines()
```

2) write a program to print the nth fibonacci number.

```
n = 10
a, b = 0, 1
for i (in range (n-1):
    C = a+b
    a, b = b, c

Print(a)
```

3) write a function that returns the nth line and last n lines from a file.

```
os.chdir (r"C:path")
def n_lines(n):
    with open("sample.txt") as file:
```

```
data = ["Apple", 456]; ["Orange", 555]
w_obj.writerows(data)
```

default standard - "\r\n"

DictWriter()

~~os.chdir(r"abcxyz")~~

```
with open("data.csv", "w", newline="") as file:
    obj = csv.DictWriter(file, ["en", "salary"])
    obj = writeheader()
    obj.writerow({"en": "John", "salary": 50})
    obj.writerows([{"en": "Bhavya", "salary":
                    500000}, {"en": "santhosh", "salary":
                    1000000}])
```

(or)

```
with open("shashank.csv", "w", newline="") as file:
```

↳ Can create new file and we can write.

writerows():

```
data = [('apple', 'google', 'Yahoo'), ('microsoft', 'netflix'
                                       , 'gmail')]
with open(file_name, 'w') as csv_file:
    csv_writer = csv.writer(csv_file)
    csv_writer.writerows(data)
```

Example :-

```
import os
import csv
path = r"abcxyz"
with open(path) as file:
    obj = csv.reader(file)
    print(obj)
    for data in obj:
        print(data)

with open(path) as file:
    obj = csv.DictReader(file)
    print(obj)
    for data in obj:
        print(data)
```

writer(), writerow(), writerows()

```
Path1 = r"abcxyz"
with open(path1, "a", newline="") as file:
    w_obj = csv.writer(file)
    w_obj.writerow(["EmployeeName", "ID"])
    w_obj.writerow(["John", 256])
```

using DictReader()

```
with open(filename, 'r') as csv_file:
    rows = csv.DictReader(csv_file)
```

rows → iterator object which holds each data record in the form of python dictionary.

Writing into csv files.

There are 2 methods,

1. csv.writer(csv_file)
2. csv.DictWriter(csv_file, fieldnames)

Note : Below are some supporting methods to write data into csv file,

1. writer_obj.writerow(): writes a single data into csv file. Data can be a list or dictionary.
2. writer_obj.writerows(): writes multiple data into csv file. Data should be list of iterables.
3. writer_obj.writeheader(): writes header in the file using the fieldnames specified.

```
import csv
```

Using writer():

```
with open(file_name, "w") as csv_file:
    csv_writer = csv.writer(csv_file)
    csv_writer.writerow([data])
```

Using DictWriter():

```
with open(file_name, 'w') as csv_file:
    csv_writer = csv.DictWriter(csv_file, [field_names])
    csv_writer.writeheader()
    csv_writer.writerow({'name': "xyz", 'age': 10})
```

tell() and seek()

> tell()- returns the current position of the cursor in the file.

* seek(pos) - navigates to the specified position.

Handling csv files.

* CSV (comma seperated values) is a simple file format used to store tabular data, such as a spreadsheet or database.
* A csv file stores tabular data (numbers and text) in plain text.
* Each line of the file is data record.
* Each record consists of one or more fields, seperated by commas.

For working csv files in python, there is an inbuilt module named csv.

Reading csv files

There are 2 methods.

1. csv.reader(csvfile)
2. csv.DictReader(csv_file)

```
import csv.
#using reader()
with open(csv_file,"r") as file:
    rows = csv.reader(file)
```

rows → iterator object which holds each data record in the form of python list

```
else:
    res += str(c) + s[i]
    c = 1
print(res)
```

4) write a program to get the following output from the list

```
l = [1, 2, 3, 4, 5, 6, 7, 8, 9]
res = []
for i, j in enumerate(l):
    if i % 2 == 0:
        res.append(j)
    else:
        res.append(j)
    print(res)
    res = []
if len(l) % 2 == 1:
    print(res)
```

5) Find all max numbers from the below list.

```
numbers = [1, 2, 3, 4, 0, 3, 2, 4, 2, 1, 0, 4]
l = list(sorted(numbers))
Print([i for i in l if i == l[-1]])
```

```
words = ["silent", "sea", "case", "listen", "Pea", "ape"
         "fare", "fear"]
def anagram(words):
    d = defaultdict(list)
    for word in words:
        if len(word) % 2 == 1:
            key = "".join(sorted(word))
            d[key].append(word)
    return d.
print(anagram(words))
```

2) write a program to print the following pattern.

```
* * * * *
 * * * *
  * * *
   * *
    *
```

```
for i in range(5, 0, -1):
    print(f'{" * " * i:^10}')
```

3) write a program to get the following output

```
S = 'AABBCCCDAACD"
C = 1
res = ""
for i in range(len(s)-1):
    if s[i] == s[i+1]:
        ct = 1
```

```
d={}
for i in l.:
    c=0
    for j in l:
        if i==j :
            ct=1
    d[i]=c
print(d)
```

a) write a python program to Sum the Squares of the first 20 natural numbers.

```
Sum_=0
for i in range(21):
    Sum+= i**2
Print(Sum_)
```

b) write a dictionary comprehension to Create a dictionary with word as its Key and if the # word is of numeric type reverse it else add the word as it is in the Value.

```
S='12 plus 18 equals to 30"
print {i:i[::-1]if i.isdigit()else i for i in s.split()}
```

Part B

) write a function to group anagrams which are of Odd length.

5) Sort the dictionary based on the last character of the key.

```
Prices = {'ACME': 45.23, 'AAPL': 612.78, 'IBM': 205.55,
          'HPQ': 37.20, 'FB': 10.75}
print(dict(sorted(prices.items(), key = lambda
          item : item[0][-1])))
```

6) Build a list with only even length string using filter function.

```
names = ['apple', 'google', 'Yahoo', 'facebook', 'Yelp'
         'flipkart', 'gmail', 'instagram', 'microsoft']
print(list(filter(lambda i : len(i)%2==0, names)))
```

7) write a python program to return a list of elements raised to the power of their indices using enumerate class.

```
numbers = [32, 65, 39, 8, 1]
l = []
for i, j in enumerate(numbers):
    i.append(j**i)
print(i)
```

8) Write a program to create a dictionary with word and the number of occurances of word in the string without using inbuilt method.

```
Sentence = "hello world welcome to python hello
            hi hello hello"
l = Sentence.split()
```

logging function name decorator.

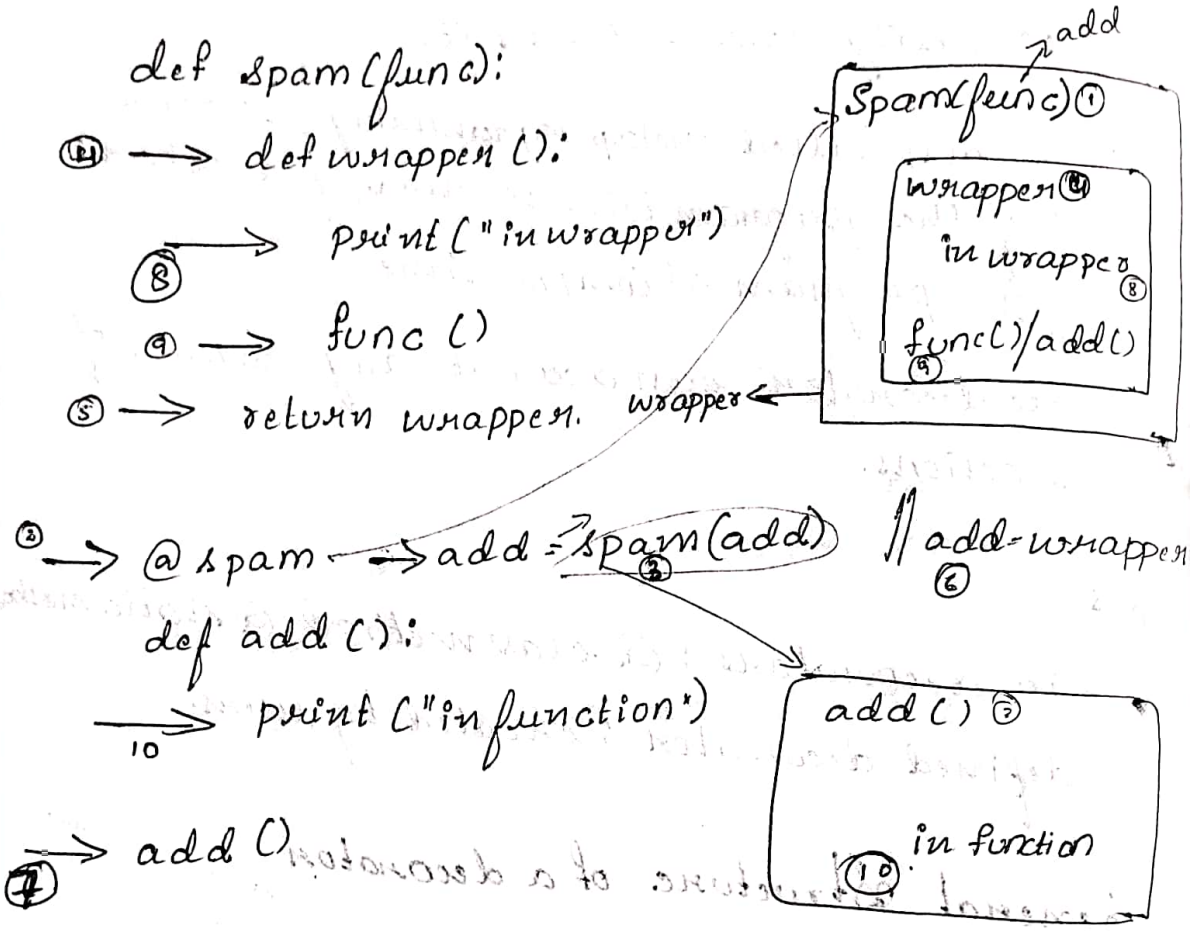
```
def log_(func):
    def wrapper(*args, **kwargs):
        print("function name:", func.__name__)
        res = func(*args, **kwargs)
        return res.
    return wrapper.

@log_    # multiply = log_(multiply)
def multiply(a,b):
    return a*b

print(multiply(7,3))
```

2) write a decorator function to input 5 seconds of delay before executing any function.

```
# delay decorator.
import time
def delay(func):
    def wrapper(*args, **kwargs):
        time.sleep(5)
        func=(*args, **kwargs)

    return wrapper
@delay    # push = delay(push)
def push(element):
    l=[]
    l.append(element)
    print(l)
push("hello")
```

Working procedure of a Decorator.

```
def spam(func):
    def wrapper():
        print("in wrapper")
        func()
    return wrapper

@spam
def add():
    print("in function")

add()
```



1) Write a decorator function to log a message (function name) before executing any function.

Note `__name__` → it is the magic method to get a function name,

dundeer method.

→ which returns name of any Programmable entity attached to it.

Decorators overview

- A decorator takes in a function, adds some functionality and returns it.
- This is also called metaprogramming because a part of the program tries to modify another part of the program at compile time.
- A single decorator can decorate any number of functions.

Types

- Built in decorators : @classmethod, @staticmeth
- user defined decorator : created by users.

General Structure of a decorator

2 param 3 argu-types so we just write.

```
def outer (func):
    def inner (*args, **kwargs):
        func (*args, **kwargs)
    return inner
```

* This process of calling a function using some other name or variable is called as monkey patching.

Note :- when a function is decorated with decorator function (@decoratorfunction) There will be 2 major changes happening

1) The parameter of outer function (func) will start pointing to main function.
2) The ~~main~~ Variable pointing to main function starts pointing to wrapper function.

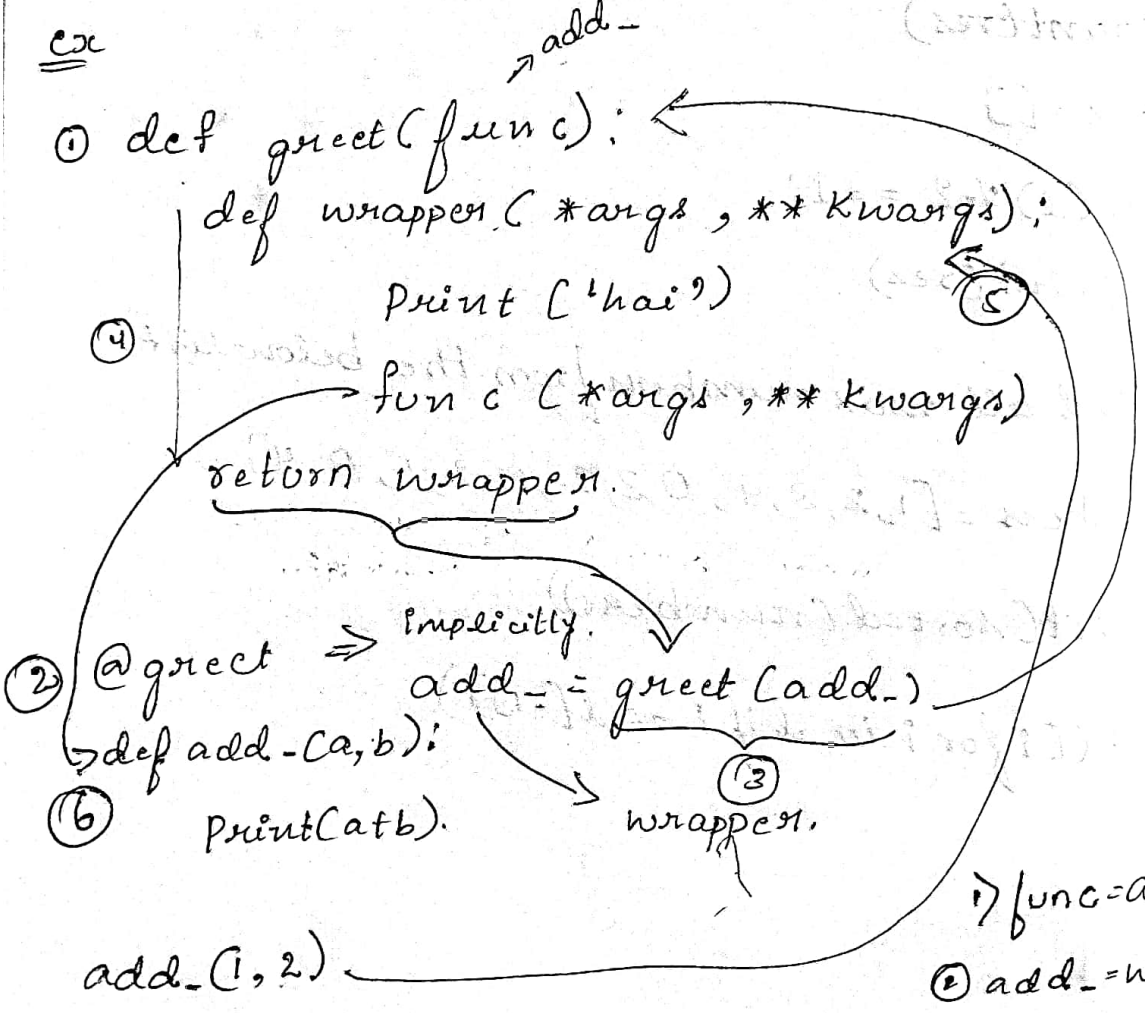
FIRST CLASS OBJECT

> First class objects are the one which is treated as an other object in python like Strings, lists, dicts etc.

> You can pass a function to another function, You can return a function from another function, just like any other functions.

> A decorator is a function, which takes another function as an argument, adds some extra functionality, and returns another function without altering the source code of original function.

ex



```
@ no_dec_fun
def push (element):
    l=[]
    l.append (element)
    Print (l)

Push ("hello")
Push ("world")
Print (d)
```

12. WADF to check if the number is positive, if positive perform addition else...

```
sm, ml = 0, 1
def n_positive (*arg):
    def positive (func):
        def wrapper (*args, **kwargs):
            global sm, ml
            for i in arg:
                if i > 0:
                    sm += i
                else:
                    ml *= i
            func (*args, **kwargs)
        return wrapper
    return positive.
@n_positive (1, 4, 6, -3, 5, -3)
def operation ():
    print (sm, ml)
operation ()
```

```
@no_dec_fun
def push(element):
    l=[]
    l.append(element)
    print(l)
push("hello")
print(f"{c} functions decorated")
```

11. WADF to create a dictionary of function name and number of function call pair

```
d={}
def no_dec_fun(func):
    def wrapper(*args, **kwargs):
        if func.__name__ not in d:
            d[func.__name__]=1
        else:
            d[func.__name__]+=1
        func(*args, **kwargs)
    return wrapper.
@no_def_fun
def sub(a,b):
    print(a-b)
sub(5,7)
sub(6,4)
sub(8,3)
```

write a decorator to execute a function for n no. of time.

```
def n_times(n):
    def repeat_(func):
        def wrapper(*args, **kwargs):
            for _ in range(n):
                func(*args, **kwargs)
        return wrapper
    return repeat_

@n_times(3)
def display():
    print("in display")

display()
```

11) WAP to count no. of decorated functions

```
C=0
def no_dec_fun(func):
    global c
    Ct=1
    def wrapper(*args, **kwargs):
        func(*args, **kwargs)
    return wrapper

@no_dec_fun
def Sub(a,b):
    print(a-b)
Sub(5,7)
```

delay for "n" Seconds.

```
import time
def n_delay(n):
    def delay(func):
        def wrapper(*args, **kwargs):
            time.sleep(n)
            func(*args, **kwargs)

        return wrapper.

    return delay.
@n_delay(1)          #@ delay     #push = delay(push)
def push(element):
    l=[]
    l.apper(element)
    Print(l)

@n_delay(3)
def display()
    Print("in display")

Push("hello")
display()
```

7) write a decorator function to count the number of function call of the main function.

```
count=0
def no_of_function_calls(func):
    def wrapper(*args, **kwargs):
        global count
        count+=1
        func(*args, **kwargs)
    return wrapper.

@no_of_function_calls.
def sub(a,b):
    print(a-b)

Sub(1,2)
Sub(3,9)
Sub(7,2)
print(count)
```

Syntax:

```
*** delay for n seconds ***
def outer(parameters of decorator):
    def decorator(main_func):
        def wrapper(parameters_of_mainfunc):
            ** additional functionality **
            mainfunc(parameters)
        return wrapper
    return decorator
```

h) write a decorator function which counts the number of arguments passed to a function.

```
def counts_(func):
    def wrapper(*args, **kwargs):
        print(f"number of args given to {func.__name__} is are:", len(args)+
              len(kwargs))
        func(*args, **kwargs)

    return wrapper.

@counts_
def add(a,b):
    print(a+b)

add(1, 2)
```

i) write a decorator that returns only the positive value on performing substrations.

```
def positive_(func):
    def wrapper(*args, **kwargs):
        return abs(func(*args, **kwargs))

    return wrapper.

@positive_
def sub(a,b):
    print(a-b)

print(sub(1,2))
```

3) write a decorator function that executes any function for 3 times

```
def thrice_(func):
    def wrapper(*args, **kwargs):
        for i in range(3):
            func(*args, **kwargs)
    return wrapper.
@thrice_      # spam = ~~repeat~~ thrice_(spam)
def spam():
    print("in spam")
spam()
```

4) write a decorator function which calculates the execution time of any function.

```
import time.
def execution_time(func):
    def wrapper(*args, **kwargs):
        start = time.time()
        func = (*args, **kwargs)
        end = time.time()
        return f'the function {func.__name__}
            is executed in {end-start} seconds'
    return wrapper
@execution_time
def display():
    time.sleep(3)
    print("in display")
print(display())
```

```
            yield str(i)[::-1]
        else:
            yield i
print(list(rev_ind(items)))
```

#

```
l1 = (str(i)[::-1] if isinstance(i, (int, float, complex)) else i for i in item)
```

5) Generating list of PI values with increasing decimal point number.

```
From math import pi
def pi_(n):                    → (1, n, -1) → reversing.
    for i in range(n):
        yield round(pi, i)
print(list(pi_(4)))

a = (round(pi, i) for i in range(n))
print(a)
```

3) Generate 10 fibonacci numbers.

```
def fibo(n):
    a,b = 0,1
    for _ in range(n):
        yield a
        c = a+b
        a,b = b,c
        yield f"{n}th fibo number is {a}"
```

Print(list(fib(10)))

for nth fibonacci number.

```
def fibo(n):
    a,b = 0,1
    for _ in range(n-1):
        c = a+b
        a,b = b,c
    yield f"{n}th fibo number is {a}"

Print(list(fibo(10)))
```

4) Generate a list → if individual datatype, reverse it else keep it as it is.

```
items = ["flipkart", 2021, "gmail", 1.2, [1,2,3]
         , 2+3j, True]
def rev_ind(lis):
    for i in lis:
        if isinstance(i, (int, float, complex, bool)):
```

1. Write a generator function generate only the strings with odd length in the given list

```
names = ["bob", "steve", 'alex', "maya", 'john']
def odd_length(string):
    for i in string:
        if len(i) % 2 == 1:
            yield i
print(list(odd_len(names)))

# using generator expression.
l = (i for i in names if len(i) % 2 == 1)
print(l)
print(list(l))
```

2. Generate a tuple of values in the given list. only numeric

```
items = ["flipkart", 2021, "gmail", 1.2, [1,2,3],
         2+3j, True]
def num_value(values):
    for i in values:
        if isinstance(i, (int, float, complex))
            and not isinstance(i, bool):
            yield i
Print(tuple(num_value(items)))

#
g = (i for i in items if isinstance(i, (int, float,
                                        complex)))
Print(tuple(g))
```

GENERATORS OVERVIEW

> Python generator is used to create python iterators
> This is done by defining a function but instead of the return statement returning from the function, use the "yield" keyword.
> A Generator is similar to a function returning an array.

Features of Generators

* A Generator is a function that returns an iterator. It generates values using the 'yield' keyword.
2) They don't take memory of a list. They are LAZY iterables. Generators are used for saving memory.
3) When called on next() function, it raises StopIteration exception when there are no more values to generate.
4) 'yield' Keyword suspends or pauses the execution of the function. But 'return' Statement ends the function.

Yield v/s return.

Return	Yield
. Stops the execution	1. Pauses the execution
. cannot return to the function blocks	2. can return to the function block
. Multiple return Statements Cannot be written in a function.	3. Multiple yield Keywords Can be written in a function.
. can return multiple elements	4. can return multiple elements

GENERATOR

```
def func(a,b):
    yield a+b
    yield a*b
    yield a-b

res = func(1,2)
print(next(res))
print(next(res))
print(next(res))

res = func(2,2)
print(list(res))
```

) Generate ~~list of~~ Square number of a given list using generator function.

```
l = [1, 2, 3, 4, 5]
def Square_(l1):
    for i in l1:
        yield i**2
res = Square_(l1)  } # print(list(Square_(l)))
Print(list(res))

# l = (i**2 for i in l1)
Print(list(l1))
```

14) WADF to reverse a string using delay function.

```
import time

def delay_func():

    def wrapper(*args, **kwargs):
        time.sleep(3)
        func(*args, **kwargs)

    return wrapper

@ delay_
    def reverse(string):
    print(string[::-1])

reverse_("bhavvvyyya")
```

if i want to pass the parameter in decorator function then

```
def n_delay(n):
        sleep(n)

@n_delay(3)
```

} extracting to add to a normal function.

```
@positive
def add(a,b):
    return a+b
Print(add(1,2))
Print(add(-4,5))
Print(add(9,3))
```

13) WADF to check if the args are iterables or not, if it is iterable return length else 'not iterable'.

```
def iterable_(func):
    def wrapper(arg):
        if isinstance(arg, (str, list, tuple, set, dict)):
            Print(len(arg))
        else:
            Print("not iterable")
        func(arg)
    return wrapper.

@iterable
def args(a):
    Print(a)
args("happy")
```

(or)

```
sm, ml = 0, 1
def positive(func):
    def wrapper(*args, **kwargs):
        global sm, ml
        for i in args:
            if i > 0:
                sm += i
            else:
                ml *= i
        func(*args, **kwargs)
    return wrapper

@positive
def operation(a, b, c, d):
    print(sm, ml)

operation(4, 6, 9, -4)
```

(or)

```
def positive(func):
    def wrapper(*args, **kwargs):
        c = 0
        for i in args:
            if i > 0:
                c += 1
            else:
                return "Not positive"
        if c > 1:
            return func(*args, **kwargs)
    return wrapper
```

SERIALIZATION IN PYTHON

* The Serialization process is a way to convert a data structure into a linear form that can be stored or transmitted over a network.

* This process is also referred to as marshalling

* The reverse process, which takes a stream of bytes and converts it back into a data structure is called deserialization or unmarshalling.

(whatever the data stores will not be modified.)

JSON (Java Script Object Notation)

in P

>JSON is a lightweight data format for data interchange which can be easily read and written by humans, easily parsed and generated by machines.

-> It is a complete language-independent text format.

>JSON is most commonly used for client-server communication because;

-> It is human readable.

-> It stores data in Key-Value pair.

-> JSON is language-independent.

-> It transfers the data as it is over the network

Serializing the data

The json module provides the following two methods to serialize python objects into JSON format.

Decorator

WAPF to check if the args are iterables or no, if it is iterable return length

```
def iterable_(func):
    def wrapper(arg):
        if isinstance(arg,(str,list,tuple,set,dict)):
            print(len(arg))
        else:
            print("not iterable")
    func(arg)
    return wrapper

def iter(func):
    def wrapper(arg):
        if isinstance(arg,str):
            print(arg[::-1])
        else:
            print("not a string")
    func(arg)
    return wrapper.

@iterable_
@iter
def args(a):
    print(a)
args("happy")
```

```
d[i[0]] = int(i[1])
res = sorted(d.items(), key = lambda i: i[-1])
print()
Print()
Print(res[-3:])
```

E) countries with less than 10K Vaccinated people:

```
path =
with open(path) as file:
    rows = csv.reader(file)
    header = next(rows)
    for i in rows:
        if i[0].strip() and i[1].strip() and
            int(i[1]) < 10000:
                print(i[0], ":", i[1])
```

F) Get the latest updated Country with its total vaccinations and number of people Vaccinated.

```
from datetime import datetime.
Path = r"C[path]",
with open(path) as file:
    dates = []
    rows = csv.reader(file)
    header = next(rows)
    for i in rows:
        if i[10].strip():
            dates.append(i[10])
    dates.sort(key = lambda date: datetime.strp
                time(date, "%y-%b-%d"))
    Print(dates)
```

```
        header = next(rows)
        for i in rows:
            if i[5].strip():
                tct = int(i[5])
Print(f'total vaccinations of all countries:{tc}')
```

B. Total Vaccination by countries.

```
Path = r"C:
with open(path) as file:
    rows = csv.reader(file)
    header = next(rows)
    for i in rows:
        Print(f'{i[0]}:{i[5]}')
```

C. names of countries and WHO regions.

```
Path = r"C
with open(path) as file:
    rows = csv.reader(file)
    header = next(rows)
    for i in rows:
        Print(f'{i[0]}:{i[2]}')
```

D. country and person vaccinated and get top 3 countries with most vaccinated people

```
Path = r"C:
with open(path) as file:
    d = {}
    rows = csv.reader(file)
    header = next(rows)
    for i in rows:
        if i[0].strip() and i[6].strip():
```

5) WAP to Sort the Shares in test.csv file based on the Share prices:

```
Path = r"C: "
with open (path) as file:
    l=[]
    rows = csv.reader(file)
    header = next(rows)
    for i in rows:
        if i[2].strip():
            l.append(float(i[2]))
Print(f'Shares: {sorted(l)}')
```

6) WAP to add the all the shares in the test.csv file.

```
with open (path) as file:
    l = 0
    rows = csv.reader(file)
    header = next(rows)
    for i in rows:
        if i[1].strip():
            l+=int(i[1])
Print(f' total number of Shares are{l}')
```

7) Analysing Vaccination details.

A. total Vaccination of all countries:

```
Path = r"C: "
with open (path) as file:
        tc = 0
        rows = csv.reader(file)
```

WAP to group male and female employees in the employees file :

```
male = []
female = []
with open (path) as file :
        rows = csv.reader (file)
        header = next (rows)
        for i in rows:
                if i[1] == "male":
                        male . append (i[0])
                else :
                        female .append (i[0])
        Print (f"male - employees : {male}, \n\n
                female employees : {female}")
Print()
```

4) WAP to group employees based on their team

```
d = { }
with open (path) as file :
        rows = csv.reader (file)
        header = next (rows)
        for i in rows:
                if i[2] not in d:
                        d[i[2]] = [i[0]]
                else:
                        d[i[2]] . append (i[0])
        Print (d)
```

CSV FILEHANDLING PROGRAMS.

1. write a program to read all the names of the employees in employee.csv file.

```
import os
import csv
path = r"abcxyz"
with open(path) as file:
    rows = csv.reader(file)
    header = next(rows)
    for data in rows:
        print(data[0], end = " ")
```

```
with open(p) as file:
    rows = csv.DictReader(file)
    for data in rows:
        print(data["name"])
```

2) write a program to print only the salaries that are >70000

```
with open(path) as file:
    rows = csv.reader(file)
    header = next(rows)
    for i in rows:
        if int(i[3]) > 70000:
            print(i[3])
```

using dictreader.

```
with open(p) as file:
    r = csv.DictReader(f)
    for data in r:
        if data["Pay"] > str(70000):
            print(data["Pay"], end=" ")
```

```
except ValueError as msg:
        Print (msg)
```

```
l=[1,2,3]
try:
   print(l[2])
    l.remove(1)

except (IndexError, ValueError) as Msg:
        Print (msg)

else:
   Print('no errors')

finally:
     Print ("executed")
```

```
a=1
b=0
```

```
try:
    if b>0:
      pass
    else:
       print(a/b)
except ZeroDivisionError:
     raise zerodivision Error('error occured')
```

```
try:
```

try - except, else, finally, raise

1. Default except block

```
a = 1
b = 0
try:
    print(a/b)
    l.append(10)
except:
    print("error handled")
```

2. Specific except block

```
except ZeroDivisionError as msg:
        print(msg)
except NameError:
    print("name error handled")
```

3) Generic except block.

```
except (ZeroDivisionError, NameError):
        print("error handled")
```

```
l = [1, 2, 3]
try:
    print(l[3])
    l.remove(4)
```

1) Multiple except block

```
except IndexError as msg:
        print(msg)
```

Finally block

> It is a block which will get executed even when the exception is raised or not.

> we can add try and except block inside finally

Syntax
```
try:
    statements
except:
    statements
finally:
    statements
```

Else block

It is a block which will get executed even when the exception is not raised.

Syntax:
```
try:
    statements.
except:
    statements.
else:
    statements.
```

User defined exceptions or custom exception

* custom exceptions can be created by inheriting exception class.

* Syntax:
```
class user-exception_name (Exception):
    pass
```

```
try :
    Statements
        try:
            Statements
        except :
            Statements.
except:
    nested try_except block
```

"as" Keyword

> Used to give alias name for the exception names written in the except block.

> Syntax : except <exception name> as alias_name:

Raise Keyword

> used to raise a specific exception whenever the condition is matched.

> Once an exception is raised, it searches for the specific except block and handles the exception

Syntax : raise error_name("message")

Syntax:

```
try:
    statements
except <exception_name>:
    statements
```

3) Generic except block

> Handles all types of exceptions in a single except block.

Syntax:

```
try:
    statements
except Exception/Base Exception:
    statements.
```

4) Multiple except block

> A single try block can have multiple except blocks.

Syntax:

```
try:
    statements
except exception1:
    statements.
except exception2:
    statements
```

Nested try and except block

Syntax:

OVERVIEW

> An event which causes the termination of the program.

> To handle unexpected termination of the program exception handling is done.

> To handle the exception, try and except block is used.

Types

1. Default except block
2. Specific except block.
3. Generic except block
4. Multiple except block

1) Default except block

Syntax

```
try:
    statements

except:
    statements
```

2) Specific except block

Handles specific exceptions only.

```
data = "hello"
with open("demo.pkl", "wb") as file:
    pickle.dump(data, file)
```

```
# deserialize - load, loads.
python_obj = pickle.loads(pck_obj)
Print(python_obj)
Print(type(python_obj))

with open("demo.pkl", "rb") as file:
    data = pickle.load(file)
    Print(data).
```

PICKLE

> "Pickling" is the process whereby a python object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into python object.

METHODS

1. dump () : pickle.dump(data, fp)
2. dumps() : pickle.dumps(data)
3. load () : Pickle.load (fp)
4. loads () : Pickle.loads (data)

Note :- The file should be opened in "binary" mode. (wb, rb, ab) and the file extension will be .pkl

Example

```
import Pickle.

# Serialize -dump, dumps
data = "hello"
Pck_obj = pickle.dumps(data)
Print (pck_obj)
Print ( type (pck_obj))
```

```
data = None.
json_obj = json.dumps(data)
Print (json_obj)
Print (type (json_obj))

data = {"username" : "user1", "Password" : "pwd"}
data2 = None
with open ("Sample.json", "w") as file :
    json.dump (data, file)
    json.dump (data2, file)
    Print ("data dumped Successfully)

# de-serialization - load, loads

data = {"username" : "user1", "Password" : "Pwd"}
json_obj = json.dumps (data)
Print (json_obj)
Print (type (json_obj))

Python_obj = json.loads (json_obj)
Print (python_obj)
Print (type (python_obj))

data = {"username" : "user1", "Password" : "Pwd"}
with open ("sample.json") as file :
    data = json.load (file)
    Print (data)
    Print (type (data)
```

1. JSON.dump(object, file): used to write Python object as JSON formatted data into a file.
   - the file type can be anything including text, JSON or even binary file.

2. JSON.dumps(object): Serializes any python object into JSON formatted String.

De-Serializing the Data

The Json module provides the following two methods to de serialize JSON data into python objects.

1. json.load(file): used to read JSON object as python objects from a file.
   * The file type can be anything including text, JSON or even binary file.

2. json.loads(object) de serializes a JSON formatted String into python objects.

Example

```
import json.

# Serialization - dump, dumps
data = {"username": "user1", "Password": "Pwd"}
Json_obj = json.dumps(data)
Print(json_obj)
Print(type(json_obj))
```

custom exception or user defined exception

```
class UserNotPresent(Exception):
    pass
userinput = input("enter the name")
if userinput == "shashank":
    print(userinput)
else:
    raise Exception("user is invalid")
```