→ Guido van Rossum
↓ 1989 and 1991
→ General purpose, high-level programming language.
→ BBC's TV show Monty's Python flying circus.
→ Short and unique
→ High-level, interpreted interactive and object-oriented scripting language.

→ Features:-
open source
dynamically typed
simple syntax
Easy to read
easy to code
portable (same code can be used in different machines
Suppose you write a program and in windows, if you want to run on Mac also, no need to change).
→ Large standard Library

Applications:-
→ Desktop
→ web, AI, machile learning.
→ Games, IoT, etc.

Disadvantages:-
Not used in Android.

python-2 or Python3:-
raw_input
long
less library functions
not simple code, ASCII

input which is used in and def except if not return true } except this all
nothing except Assert del finally import or try False
available Handling Break elif, for in pass while None others are
more class. else from is print with match (lowercase
simple continue except global lambda as
code, unicode raise yield nonlocal

PEP-8:-
Python enhancement proposal-8 → It is the 8th proposal in the python enhancement proposal's.
→ It gives rules, identation

Pip:- python installation package
→ By having this we can easily write the projects.
→ If anything not there in pip will install, uninstall
→ i.e essential tool for python developers, working with external libarries and ensuring projects can be easily set-up & maintained.
→ show information, upgrade, List installed packages.

→ Maximum length 79-characters
→ No use of curly braces or semicolons, it follows identation.

2-types of comment line:-
→ singleline
#g → multiline " " " multiline """

obc → cba
bcc
bca
bcc

variables:- Storage purpose
No special characters are allowed except underscore, keywords are not used as

multiple values:-
x, y, z = "pupa", "deri", "santhoshi"
x = y = z = "pupa"

keywords → soft-keyword [_, match, case
hard-keyword
↓ sometime they behave like variable names also
35-keywords
import keyword
keyword.kwlist
len(keyword.kwlist)
whatever the situation not a matter, always behave as keyword.

PEP-8:-
Python enhancement proposal-8 → It is the 8th proposal in the python enhancement proposal's.
→ It gives rules, identation

→ to use unicode need to add
u' ⇒ u'hello"

x = 5
y = "john"
print(x+y)
⇒ error.
x = 5, y = "john"
print(x, y)
5 john

unpacking tuple:-
fruits = ("apple", "green", "red")
x, y, z = fruits
print(x) → we can unpack tuple
print(y) also, it will give.
print(y) → we can upack sets
→ whereas in dictionary if we unpack then only keys we can see.

Global:-
variables that are declared outside of function, global keyword
→ we can use anywhere, if we want

```
x="awesome"
def myfunc():
    x="fantastic"
    myfunc().
print("python is "+x) → awesome
```

→ If we change or use same variable in inside a function, then it will allow only inside the function.
→ If we want change, then use that variable or give that variable as global and change.

```
global x
x='fantastic'
myfunc()
```

Built-in Data types:-

Text type:- string
int :- int.
Numeric type : int, float, complex
Sequence type : List, tuple, range.
byte type:- memoryview, bytes, bytearray
mapping type: dict
set type : set, frozenset.
none type : None
Boolean type : bool
range:- → immutable
→ range(start, stop, increment)
included → excluded

→ If we want to find type of data use x=10
print(type(x))

bytes:- 0 to 256 [immutable]
→ Directly stored in hard-disk
→ group of byte numbers just like an array.
→ All the functions that work on strings will works on bytes
→ we can convert any type to bytes except complex, float.

```
a=[1,2,3,4]
alist=bytes(a)
```

(list) → bytes

→ we can access b[0], b[1],

```
x=bytes(5)
```

↓
It will given default 5-zeros in hexadecimal format.

```
x="hello"
bstr=bytes(str, 'utf-8')
str="hello"
bstr=bytes(str, 'utf-8')
```

→ 8 means 1 byte = 8 bits

→ utf stands for unicode transformation format.
→ unicode it can represents all languages in bytes format.
→ It will transform that code into specified binary format.

```
bstr=bytes(str, 'utf-8') → b'hello'
range=range(20)
brange=bytes(range)
print(brange)
```

↓
It will print all values upto 20, if any is value is having value, then it will represent with that value.

b'\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09 → is having value],
Where \x09 → is having value],
like that.

→ if printable charaters are there it will print like that only, otherwise it print hexadecimal format.
→ If we want to convert dictionary values to bytes, then only keys will be considered.
→ wherever we use bytes there we can use bytearray also but only difference is here we can change [mutable].
→ And range also varies.

```
str="rupa"
bastr=bytearray(str, encoding='utf-8')
print(bastr)
```

if we want to change string value then `bastr[1]='a'`

↓ gives error

If we want to print normal then `str.decode()` → hello

we have to give in these
basta[i] = ord('a')

memoryview:-

it work only on bytes & bytearray.
→ here only one i.e read only()
↓
it gives true for immutable & false for mutable ones.

x=memoryview(bytes(5))
print(x) → 0x00000[E14EBDBICD
print(type(x))

→ This is useful for handling large data efficiently, such as binary data, large arrays, or data shared between different components.

Complex:-

real, imaginary used like AC. resistance, voltage, polynomial equations, signal processing etc.

c=10+5j
(.real.⇒)10.
c.imag ⇒ 5.
complex(10,-2) ⇒10-2j
Complex(True, False) ⇒ (1+0j

Compound data type:-

→ Here datatype can store more than one value

List:- mutable, insertion order preserved.
→ heterogenous.
→ duplicates allowed, growable
→ values should be enclosed with

x=["apple", "banana", "grapes"]
print(x).
type(x)
→ list memory will be allocated 0,4,8,16,24,32.

Taking i/p:- → float:
a=list(map(int(input().split())

→ we can access.
a=[10,20,30,40] ⇒ a[0]
a[1]
a[2]

slist=[1,10,20,30].
sum(slist) → 81
s2list=[1,2,4.3,True)
→ Here for addition only bool, int, float boolean only for these we can find sum.

List constructor

a=list(("apple", "banana"))
print(a)

Indexing:- print(thislist[1])
↓ indexing

Slicing:-
= positive
→ negative
s[start:stop:increment)
↓ include → exclude

→ Slicing we can get substring

~~this~~ checking value:-

s=["apple", "banana"]
if "apple" in s:
print("yes this is there")

change item val
a[1]="blueberry"
print(a) → "apple", "banana"

→ change a range of item Variables:-

thislist=["apple", "cherry", "kiwi"]
thislist[1:3]=["blackcurrant", "watermelon"]
print(thislist) →
["apple", "blackcurrant", "watermelon", "kiwi"]

→ If we have give values extra, then also it will print.
→ If you insert less items than you replace the new items will be inserted where you specified and remaining will move accordingly.

List methods:-

append → It will add elements at the end of list.
→ Here we can able to add only one

item,
a.append([1,3,5])
a.append("apple", "banana")
↙ Takes exactly one argument
a.append(("apple", "banana"))
a.append({1:"rupa", 2:"devi"})

[1,2,3], {1:"rupa", 2:"devi"}

a.append({1,2,3}) → error, it takes only one argument
a.append({1,2,3}, {3,4,5})

Clear():-
→ It will clear the elements,
→ empty list will be like that only
a.clear() → it does not take any arguments.

Copy():- copy also takes any arguments.
2 types of copy
→ deepcopy
→ shallow copy
for these we have to import copy module.

thislist=["apple", "banana"]
mylist = thislist.copy()
Print(mylist)

array = [12, 19, 13, [90, 45]]
array1 = array.copy()
Print(array1)
[12, 19, 13, [90, 45]]
array.append(890)
Print(array1)
[12, 19, 13, [90, 45]]
array[3][0] = 23
Print(array)
[12, 19, 13, [23, 45], 890]

deepcopy:- Here any changes made to a copy of the object do not reflect the original object

import copy
array2 = copy.deepcopy(array)
print(array2)
[12, 19, 13, [23, 45], 890]
array[3][0] = 45
→ Syntax error
array[3][1] = 34
Print(array)
[12, 19, 13, [23, 34], 890]
Print(array2)
[12, 19, 13, [23, 45, 890]

shallow copy:- Changes made to a copy of an object do reflect in the original object.

count:- how many times that the element is there in list.
Str: count("appl")
1.
Sto.count([1,2,3]) → 1
Sto. count({1:"rupa", 2:"devi"}) → 0
It takes exactly one argument, exactly one argument. But in strings it takes 3 arguments

index():- it will give that where the particular element is located, here we can give ranges also
array = [2, 90, 3, 4, 10, 11, 'AS', 7]
array.index(3) → start value
array.index(3, 1, 4) → end value, always excluded
A(A.index(60)) = 100
Print(A)

extend():- No. of elements, it will add

Adding lists:-
list1 = ['a', 'b', 'c'] here in extend we add list to dictionary, it will
list2 = takes only keys.
→ we can add iterable (set, dictionary, tuple)

del:- del. thislist[0] → we have to specify index, if not, whole will be deleted.

insert():-
if we want insert element at specified index, we use insert.

```
insert(index, "value")
```

pop():-
It will delete the element that which we have specified, if we are not specified the element, then. by default, last element will be deleted.
→ And whatever the element we delete that will it print.

remove():-
remove the specified element
a.remove("apple") → in list atleast only one time will remove.
→ If more than one item with the specified value, then it will delete the first occurence one.

```
reversed(sequence)
```

reverse → Here for this we have to use, another one
↓
It doesn't take any argument
this list
→ It will reverse current sorting order.
→ reversed function return an iterator that access the given sequence in the reverse order.
→ If you want you can convert this iterator to a list if you need to.

sort()
By default it will sort in ascending order.
→ If we want in ~~asce~~ desending

```
a=[1,2,3,4]
a.sort()
print(a)
a.sort(reverse=True)
print(a)
```

→ ~~If~~ consider in a list, if we are having capital and small
→ If for example small letters having highest precedence, then only, it first give preference to capital, then after only small letters consider if we don't like that

reverse

```
a.sort(key=str.lower)
x=["apple","banana"]
for i in x:
    print(i)
print(i for i in x)
for i in range(len(x)):
    print(i, => [len(i) for i in x]
```

using while

```
i=0
while i<len(this list)
    print(thislist[i])
    i=i+1
```

List comprehension:-
List, it is very short and concise way.

```
newlist=[expression for x in iterable condition]
fruits=["apple","banana","chery","kiwi"]
newlist=[]
for x in fruits:
    if ~~x in newlist~~
    if 'a' in x:
        newlist.append(x)
print(newlist)
```

or

```
x=[x for x in supd. if 'a' in x]
print(x)
```

↙ here condition is optional.

List comprehension:-

```
L=[0 for i in range(10)]
=> 0 0 0 0 0 0 0 0 0 0
```

[i**2 for i in range(1,8)] → [1,4,9,16,25,36,49]

[i*10 for i in L] → [10,140,50,90,120]

[c*2 for c in string] → ['HH', 'ee', 'll', 'lss', 'oo']

[m[0] for m in M]

L=[[i,j] for i in range(2) for j in range(2)] → [[0,0],[0,1],[1,0],[1,1]]

```
for i in range(2):
    for j in range(2):
        print([i,j])
```

[x.upper() for x in fruits]

Tuple :- immutable, order preserved, duplicates allowed.

→ we are not able to add any data.

→ Here tuples are more faster, than list because here we don't add any elements and delete.

→ If we want to perform any operations then convert that into other types and convert into other data types and apply.

→ Here we only perform operation
index
count

Why sort() is not acceptable where sorted() accepted in tuple

Here sorted() is the thing that it will take new tuple, add that elements into another tuple.

→ It doesn't modify the tuple it will create new one.

→ we can access tuple as list

→ Tuple Constructor:-

a=tuple("apple","banana")

tuple adding:-

a=("rupa","jen")

b=("santhoshi")

a+b= a=("apple",)

+ operation is applicable only for list and tuple

- operator worked for set

If no. of variables is less than the no. of values, you can add an * to the variable name and values will be assigned

fruits=("apple","banana","orange","grapes","raspberry")

(green,yellow,*red)=fruits

print(green) → apple

print(yellow) → banana

print(red) → orange, grapes, raspberry

→ (green,*yellow,red)=fruits

print(green)

("apple","banana","orange","grapes","raspberry")

print(yellow) → print(red)

```
for x in fruits
    print(x)
```

```
for i in range(len(fruits)):
    print(fruits(i))
```

```
i=0
while(i<len(fruits)):
    print(fruits(i))
    i=i+1
```

multiplication

* is possible for list and tuple

Dictionary:- Duplicate keys are not allowed. If we try to try are trying to insert an entry with duplicate key the old value will be replaced with new value.

```
x = {"name": "John", "age": 36}
Print(x) → {'name': 'John', 'age': x}
type(x)
```

→ A dictionary is a key-value pair set arranged in any order.
→ It stores a specific value for each key. value is any python object, while the key can not
→ key is any primitive data type.
→ comma and curly braces are used.

```
d = {1: 'Jimmy', 2: 'Alex', 3: 'Tom',
     4: 'Mike'}
Print(d) →
Print(d[1]).
Print(d[4])
Print(d.keys())
Print(d.values())
dict = {1: 'hello', 2: 'devi', 3: 'santhoshi'}
dict[0] → Key Error.
```

dict Constructor:-

```
thisdict = dict(name = "John", age = 36,
                Country = "Norway")
Print(thisdict)
thisdict = {"brand": "Ford",
            "model": Mustang,
            "year": 1964}
x = thisdict["model"]
x = thisdict.keys()
x = thisdict.values()
x = thisdict.items()
thisdict = {"brand": "Ford",
            "model": "Mustang",
            "year": 1964}
if "model" in thisdict:
    print("yes, model is one of this dict")
```

update:-

```
thisdict.update({"year": 2021})
```

Add

```
thisdict["color"] = "red"
```

pop()

It will remove specified value, here it dictionary it takes any arguments, but only first one is removed and it will display the element what we delete.

```
thisdict.pop("model")
```

popitem():-

It will delete the element at last.

Del() :- it will delete the specified index, if we don't specify index it will delete the whole.

```
del thisdict["model"]
```

Clear():-

It doesn't take any argument.

sorted()
→ only for keys

sum():-
only for keys.

loop dictionaries:-

```
for x in thisdict:
    print(x)
```

```
for x in thisdict:
    print(x)
for x in thisdict
    print(thisdict[x])
```
↓ it gives values.

```
for x in thistedict:
    print(len(thisdict[x]))
```
↓ length of values

```
for x in thisdict.values():
    print(x)
for x,y in thisdict.items():
    print(x,y)
```

Nested dictionaries:-

```
myfamily = {
  "child1": {
    "name": "Emil",
    "year": 2004
  },
  "child2": {
    "name": "Tobias",
    "year": 2007
  },
  "child3": {
    "name": "Linus",
    "year": 2011
  }
}
print(myfamily["child2"]["name"])
```

Set:- fast compared to list & tuples. here in sets we don't use duplicate elements

→ Here insertion order is not preserved.

→ Duplicates are not allowed.

→ heterogenous objects are allowed

→ index concept is not applicable.
↓ because unordered.
→ growable in nature

→ we can able to add only immutable ones.

→ true and 1 are considered same value and are treated as duplicates.

→ False and 0 are treated as same.

set constructor:-

```
R = set(1,2,3,4)
len(R)
1 in R → True
2 in R → True.
```

Empty set:-

```
a = set()
set1.add[1,2,3] → error
set1.add(1,2,3) → {1,2,3}.
```

Add exactly takes one argument

Sorted:- The sorted function always returns a list, regardless of the type of input iterable. This does not change the original set but gives you a sorted view of its elements.

```
myset = {3,1,4,1,5,9}
sorted_list = sorted(my_set)
print(sorted_list)
```
[1,3,4,5,9]

```
nums = [3,1,4,1,5]
sorted_nums = sorted(nums)
```
[1,1,3,4,5]

```
tuple = (3,1,4,1,9)
sorted_tuple = sorted(my_tuple)
my_string = "hello"
sorted_string = sorted(my_string)
```
↓
['e','h','l','l','o']

isdisjoint():-

set1 = {1,2,3}

set2 = {2,3,4}

>> set1.isdisjoint(set2)

False.

) set2.remove(3)

print(set2)

{2,4}

issubset():-

set1 = {1,2,3,4,5,6}

set2 = {2,3}

set2.issubset(set1)

true

issuperset

set1.issuperset(set2)

true.

set1 >= set2

true

set1 == set2

False

set1 <= set2 → False

set1.union(set2)

→ {1,2,3,4,5,6}

frozenset:- It is similar to set but we are notable to change

thisset = {"apple", "banana", "cherry"}

for x in thisset:

print(x)

~~thisset = {"apple"}~~.

update:-

Here we can add mutuable ones also.

thisset = {"apple", "banana", "cherry"}

tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

print(thisset)

remove:-

thisset.remove("banana")

if item is not there it will raise error.

discard():-

thisset.discard("banana")

Will not raise an error.

pop():- last element delete, but here we ~~dont~~ can't say what element will delete

clear():-

del:- del thiset

union:- set1 = {"a", "b", "c"}

set2 = {"d", "e", "f"}

set3 = set1.union(set2)

print(set3)

We can use | operator instead of union, but | it will takes only same iterable.

You can add how many you want.

myset = set1.union(set2, set3, set4)

update():- update also do same what what union, but difference is update ~~to~~ doesn't take new variable, whereas union take

→ It doesn't return a new set.

Intersection:- &(but only allow same iterabl

set1.intersection(set2)

only print duplicates

Intersection_update():-

will also keep only duplicates instead of returning new set, it will update the preset set only

difference:-

It will return a new set that will contain only the items from set that are not present in other set.

set3 = set1.difference(set2)
- [we can use this operator]
→ but works only for same iterable

difference_update:-
Instead of updating returning new set
it will update the present one

Symmetric differences:-
Keep elements that are not present in both sets.
^[we can use this also]

None:-
it is something like null value in java,
if value is not there then to handle such cases we use None

separator:-
print(a,b,c, sep="y")
10y20y30 → here definetely we have to mention comma

String format:-
print("{}, {}, {}".format(a,b,c)) → separated by comma
print(f"{a}, {b}, {c}")

separated by comma
print("the result of adding", a, "and", b, "is", c)

f
print(f"the result of adding {a} and {b} is {c}")

formated string:-
print("a value is %d" %a)
print("b value is %d and c value is %d" %(b,c))

replacement:-
print("hello {} your salary is {} and your friend {} is".format(name, salary, gf))

type conversions:-
converting one data type to other.
→ we can convert any datatype into int except complex.
→ we can convert any type into string except complex

random:-
import random → we have to import
from random import randint
print(random.randint(a,b)) → will return a random integer between a,b here a,b both are included
print(random.randrange(a,b)) → Here b is excluded

from math import *
import random
random.random()

Built-in:-
abs(), round()
abs(3+4i) = √(9+16) = √25 = 5

max:-
max(iterable, key=function)
here function is optional. this function may be lambda or user defined function) built-in function.
max(["fggg", "rupa", "devi"]) → rupa → it compare maximum length, ASCII

max(['shhhh', 'gggg', 'dddd'], key=len)
apple.

max([1,3,1,9],[10,20,0,3],[0,0,100,1])
⇒ (10,20,0,3)

max([1,3,1,9],[2,3,4,5],[3,4,5,6], key=lambda i: i[2]) ⇒ [1,3,1,9]

max([1,3,1,9],[2,3,4,5],[4,5,6], key=lambda i: i[1])

txt = "Rupa"
print("Ru" in text) ⇒ True
if "Ru" in txt:
    print("True it is there")
if "pu" not in txt:
    print("txt is not there")

Address:-
id()

lower to upper:-
a = "hello"
print(a.upper())
print(a.lower())

Strings():-
→ In python char type is also represented by strings, char data type is not possible.
→ slicing and indexing concept is also applicable for strings.
→ But we are not able to change the elements.

In a given text if we want to find out place of specified element:-
s = input("Enter some text").
for i in range(len(s)):
    if s[i] == 'a':
        print(i)

word = "python"
word[0] = 'j'
→ type Error

If we want to change
j + word[1:]
A = "hello world"
print(A[1])

For x in banana:
    print(x)

a = "hello world"
print(len(a))

strip():- it will remove both leading & trailing characters
"###Hello###".strip('#')
Hello

→ strip() → character see the first character, if first character not match, it will not see the rest of elements
→ strip() not only for single word, but for multiple words, in multiple it check any one of the one will match it will remove.

lstrip():- leading whitespace only
rstrip():- trailing only

replace:- replace not only used for single character. it used for multiple for group of characters.
a = "hello world"
a.replace("h", "i"))
a.replace("santhoshi", "rupadevi"))
String.replace(old, new, count))
a.replace('3', 'e', 2)
development

a = "hello world"
b = a.split(',')
print(b)

b="Hello World"
b=a.split(',')
print(b)

c="sanmoshi,rapaderi"
c.split("moshi")
['san', 'rapaderi']

Capitalize():
→ here it capitalize the first characters in given string.
→ only capitalize the first word.
print(a.capitalize())

Case fold():
It will convert small case here only
Convert capital to small.

Title:
a="dggg gyhyhh" → print(a.title())
→ Dggg, Gyhyhh
It converts whole name to each word capitd

Center:- split():- maxlimit specifies the maximum number of splits, the default value-1
→means no limit on the number of splits
If the delimiter is not found enter string will be given as it is.

Center:-
Here we can able to add exactly one character long.
str.center(35, '%.%.%.')
↓gives error.
str="demo"
str.center(10, '#')
:###demo###:

ljust:- returns a left justified version of string.
s1.ljust(45) → just add spaces to the left

rjust:-
~~right~~ returns right justified version of string.

count():-
returns the number of times a specified value occurs in a string.
str="this is demo"
print(str.count("demo")) → 2
print(str.count("demo",7,25)) → 1
↓ start ↓ stop

encode():
used to convert a string into bytes. This method is particularly useful in when dealing with text needs to be stored transmitted or processed in a specific encoding format.
encode() → encodes the string using the specified encoding
If no encoding is specified UTF-8 will be used → the default value errors is strict.
string.encode(encoding=encoding, errors=strict)

errors:-
backslashreplace:-
text="Hello World! ~~abcd~~ !"
encoded_text=text.encode('ascii', errors='backslashreplace')
print(encoded_text)
b'Hello World!\\U4f60\\U597d\\uff01'
\\U4e16\\U754c\\uff01'
It replaces characters that cannot be encoded with their unicode escape sequence (e.g \U1234).

Ignore: ignores that cannot be encoded.

namereplace:-
text="Hello, World! ~~abcd~~ !"
encoded_text=text.encode('ascii', error='namereplace')
↓replaces the character with a text explaining the character which (i.e we are not able to mention start, stop index in list, which is possible in string)

strict:- this is the default string.
→ It raises a unicode error if it encounter a charaters that cannot be encoded, then it will give error.

replace:-
errors="replace"
↓
replaces with question mark.

xmlcharrefreplace:-
↓
replaces the charaters with the xml charater.
for xml/html we will use this xmlcharrefreplace.

endswith():-
str.endswith("text")
↓True
str.endswith("text",12,17)
true.

expandstab():-

find():- searches the string for the specifed value and returns the position of where it was found
str.find("text")
→ if text is not find then it will give -1,
→ whereas in index if text not found it will error.
→ str.find("text",12,17))

isalnum():- returns true if all charaters in the string are the alphanumeric.
print(str.isalnum())

isalpha():- returns true if all charaters in the strings are alphabets

isascii:- charaters in the sting are ASCII

isdecimal:- returns true if all charaters in the string are decimal

isdigits():- returns true if all the charaters in the string are digits, i.e subscipt/superscript.

isidentifier():- returns true if it is perfect identifer.

islower():- returns true if is it lowercase.

isnumeric:- returns true if all charaters in the string are numeric → like subscipt, superscript, roman, numeral, fraction ect

isprintable():- returns true if it is printable charater.

isspace():- returns true if all charaters in the string are whitespace

istitle():- returns true if the string follows the rules of a title.

isupper():- If all charaters are upper.

- zfill():- It fills the string with a specified number of 0 values at beginning

str = "wow"
print(str.zfill(7))
000wow
print(str.zfill(10))
0000000wow

swapcase():
lowercase becomes uppercase & uppercase becomes lowercase
st.swapcase())
Starts with.
st.startswith("one",2,45))
rindex:-
It will search from right side, i.e it give first prefference to right side.
rsplit():-
which occurs from right-side onwards
rfind():-
It will search from right side
join:-
Difference b/w join and add
→ join ~~add in betr~~ will add elements ~~st~~ in between the strings.
→ whereas add will add elements at the end of the string

```
for x in range(10,0,-1)
  print(x)
else in for:-
for € in range(6):
  print(x)
else:
  Print("Finally Finished")
```

else statement will not execute if loop statement is stooped by a break statement.

Templates:-
formatter

unicode division:-

0-31 → non printable characters
127 - left array
32-47 - special symbols
48-57 - 0-9
58-64 - special symbols
65-90 - 'A'-'Z'
91-96 - special symbols
97-122 - 'a'-'z'
123-126 - special symbols

unicode of single digit:-
ord('5')
ord('A')

unicode of double
chr(96)
chr(126)

import string:-
import string
print(string.ascii_letters)
print(string.ascii_lowercase)
Print(string.ascii_uppercase)
print(string.ascii_punctuations)
Print(string.octdigits)
Print(string.digits)
print(string.hexdigits)
capwords

Functions:- Block of code, which runs when it is called. you can pass data known as parameters into a function

Creating:-
def keyword
def function_name(parameters)
function_name(call)

Information can be passed into functions as arguments:- Arguments are specified nas a function name, inside the parenthis you can add as many elements you want, just separate them with a comma

```
def my_function(fname):
    Print(fname + "refsness")
my_function("emil")
myfunction("tobias")
my_function("linus")
```

→ An argument is the value that is sent to the function when it is called.

Exception handling:-

Try, except, else, finally

Try → block code for errors
except → handle the error
else → else block will execute if there is no error.
finally → irrespective of the error what happen finally will be executed regardless error occur or not final block will execute

→ For one try block we mention as many exception as we want which one is correct suited that will execute

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The try except is finished")
```

Operators:-

Walrus operator: (:=

x:=3

↓

means assignment & expression done on same same line.

print(x:=3) → 3

s=3

print(s) → 3

→ Assignment operator takes 2-lines whereas walrus takes one operator line.

Arithematic:-

+, -, *, /, //, x**y, %

↓

Here 5/2 ⇒ 2.5

↓

floor division ← 5//2 ⇒ 2.

Comparision:- It will compare 2-elements

>, <, ==, !=, =>, <=

Logical operator (and, or, NOT)

print(0 or 1) ⇒ 1

print(False or 'hey') ## hey

print('hi' or 'hey') #hi

print([] or False) ## False

print(False or []) []

In or returns value of first operand and first operand is not a [False, 0, '', []] otherwise it returns last operand.

And:- Here it doesn't see second element, if first in false.

print(0 and 1) ⇒ 0.

print(1 and 0) ⇒ 0

print(False and 'hey') False

print('hi' and 'hey') hey

print([] and False) []

print(False and []) False

→ math package is used for normal

→ whereas cmath is used for complex numbers

→ decimal package is used for decimals & Floating point numbers

→ Fraction package is used for rational numbers.

Membership:- in, not in

↓ returns true if specified value is not there

returns true if specified value is there

Identity:- is, is not

↓ returns true if both variables are not the same object

Returns true if both variables are the same object

Bitwise operators:-

Python conditions:-

→ if

→ elif

→ else

Ternary

a>b ? True : False

↑ condition true

Pass statement:-

Interpreter completely ignores the comment, whereas pass is not ignored, if statement can't be empty, but if you for some reason have an a statement without no content put in pass statement to avoid getting Error.

Condition false

Python loops:-

for

while

while-else:- with while statement we can run a block of code once when the condition no longer is true

```
i=1
while(i<6):
    print i
    i=i+1
else:
    print(i, "is no longer less than 6")
```

For:-

```
for x in range(11)
    print(x)

for x in range(21):
    if(x%2!=0):
        print(x)
```

memory view:-

memory view object allows you to access the internal data of an object that supports the buffer protocal without copying it.

↓ such as bytes, bytearray or an array from the array modul.

```
data = bytearray(b'Hello world!')
view = memoryview(data)
print(view[0]) → 72. ASCII code for H
sliced_view = view[0:5]
print(sliced_view.tobytes())
```

↓

b'Hello'

Uses:-

memory views avoid copying data, which can be more efficient for large data structures.

→. They allow different parts of your program to access and modify the same data without creating multiple copies.

→ memoryview provides a way to interact with binary data efficiently offering direct access to underlying data buffers, which is mainly useful for performance-critical applications and large datasets.