Python

⇒ Computer's main component → RAM (microprocessor) ⇒ #1) Generic programming
→ GPU
→ CPU

bit → 0 / 1 — collection of 8 bits ⇒ 1 byte

→ Each ascii character need 8 bits to represent.

1 KB = 1024 Bytes | 1MB = 1024 Killobytes | 1GB = 1024 Mega Bytes

→ Computer's have memory but no imagination
→ Each cpu has some set of instruction which it can perform
⇒ Machine only known '0' & '1' so ~~it~~ code, and human know characters, so programming language is one who converts character code in machine code

Ex: Python, ruby, java, C++, html, css, go, kotlin etc

⇒ Various programming languages exist & each is their have own set of works that it can perform
⇒ high level language human language like programming lang.

Programming lang ⇒ [Compiler or interpreter] → machine code

→ Compiler gives executable file while interpreter give output asap after debug completion

---

Terminal (OS) Ex linux, windows, mac os

⇒ we can perform basis os related activity using terminal. and for this we only use keyboard
⇒ Terminal can be able to do alot without using GUI in OS

Navigation Pwd, ls, cd, cd..,

mkdir ⇒ create new folder
→ file locations can be relative or absolute
⇒ rm -r folder -y ⇒ use to delete folder

CP ⇒ copy
mv ⇒ move
⇒ while moving or copying we can rename

File operations in terminal

⊙ cat → see content of a file ⊙ Touch ⇒ new file creation
⊙ echo ⇒ echo "hello world"
we can use inset (>) option to inset value in file
↓
use write (not append) to tally · overwrite take place.

to append
⊙ echo "hello world" → append >> filename.py

> ⇒ write	>> ⇒ append

⇒ nano and vi are some text editors in ubuntu & call them from terminal.

Internet # Community (like stackoverflow)

Start of python programming

⇒ To execute python you need to have python interpreter on your os

basics.

Each programming language has semantics & syntax.
→ To display we use print() → Print function.
⇒

Python datatypes

Data structure ⇒ a well arranged, created & stored structurised data which has various operations.
⇒ Structured data is important

(A) integer (int) (B) string (str) (C) float "float" (d) boolean "bool" (E) [10,2,3] "list"
(True, False) (list of any datatype)

Key word ⇒ (F) (10,2,8) ↓ "tupple" (D) Dictonary (a key value pair) { "no": 8624969701 }
(keyword = dict)

Key words ⇒

⇒ comment → not executable content in code file (green colour) → ignored by interpretor.

⇒ for dictionary we use keyvalue pair comma seperated

a = { "grs" : 8624 , "abhi" : 7775 , "adesh" : 9199 } ⇒ such like that

and we can use ~~a.grs() or a['d']~~ a["grs"] ⇒ this will give value for value inside dictionary

⇒ All of these have own pros & cons and all are useful.

Console IO

Say ~~dict~~ d = { "grs" : 8624,
"abhi" : 9775,
"adesh" : 9199,
}

~~we can~~ to access dictionary element we can use key

Ex:- d["grs"] or d["abhi"] or d["adesh"]

⇒ If you want to pass argument in terminal you can use library

⇒ Input function is mostly use to take input from terminal ↑ takes input in string format you can also send msg along with it

a = int (input ("enter your age"))

int → typecast

input → msg will display in terminal

If statement

If val not in d: → wrong input

else:
print value

⇒ with if we are going to test that is key value is there in dictionary or not.

↳ To put new value it is also easy

d["newkey"] = newvalue → this will add new entry in dictionary

⇒ If statement use [Conditional statement] and on basis of it we will run next code.

def keyword → used to create fun

repeating the code which can use many time & call it by it's name. and each function has own task (internal code)

```
def oddorEven(a){
    If (a%2==0):
        print("even")
    else return;
    print("odd")
}
oddorEven(10)
```

→ conditional statement
→ function declaration
⇒ function has parameter
→ a is parameter (positional argument)
→ function call

Variables

these are containers & they are used to store values

⇒ try to use expressive name for variables.
⇒ there should no space in variable (you can use (_) underscore, lower(camelcase) (even hyphens are invalid in python
⇒ never start with number or special character

`_abc` ⇒ valid ⇒ variables can start with underscore

⇒ order of codes matters in python.
⇒ To find out type of variable (type function) is used
type(variable) ⇒ some datatype

⇒ All datatype in python are class and all variables are object of it.

#Strings

collection of characters call as string Ex ⇒ a = "abc"

⇒ you can also index the string like array

a = " a b c d "
    0 1 2 3

len(a)

len function used to find out length of string or array

⇒ indexing starts from 0 → zero

⇒ you can use negative indexing -1 ⇒ last element

a[0] ⇒ 'a' a[-1] = 'd'

⇒ to use quotes \' \" used → escape sequence

""" —"— """ ⇒ tripple quotes used to create multiline string in python

#String operations

string slicing can used to fetch substring from string

name [start : end : increment/decrement].

this will display string from start index to end -1

⇒ end index should incremented by 1

⇒ default start = 0 default end ⇒ len(string) + 1

name [:] ⇒ Default values will be used

⇒ for slicing we can use negative values also

a = "hello" ; b = "world"

c = a + " " + b

string concatenation

⇒ "Ha" * 8 so this is "HaHaHaHaHaHaHaHa"

⇒ len() ⇒ used to calculate length of string or array.

String methods (for string)

⇒ we also have built in function from python

upper() ⇒ make all character upper alphabets ⇒ BaSe ⇒ BASE

lower() ⇒ m —"— lower case ⇒ BAse ⇒ base

capitalize() ⇒ make first letter capital base ⇒ Base

to find substring you can use in keyword

```
if "abc" in "abcdef":
    print("yes")
else:
    print("no")
```

o/p => yes

→ letter or substring both can be searched using (in) keyword

Numbers

num1 = 1 (↑ integer) num2 = 10.6 (↓ float)

Ex: a = 1+2j ⇒ complex

↓ for this we can grab real & imginary part

a.real ⇒ give real value of complex no (property)

a.imag ⇒ gives imaginary value

⇒ type cast is also used to interconvert datatype between each other if possible (cast)

int(), float(), str() ⇒ few those functions used for typecast

Numerical operations

+ ⇒ addition - ⇒ sub * ⇒ mul

/ ⇒ float division // ⇒ integer division

% ⇒ module (get remainder) ~~expona~~ power ⇒ **

⇒ abs() → get absolute value for a number. 2**3 ⇒ 8 or can use pow(2,3) → pow function

⇒ max ⇒ find max among Inputs.

⇒ min ⇒ find min

⇒ round (a number, val no. of digits aftr float)

a = 3.3333 round(a, 2) ⇒ 3.33

is - integer function()

is_integer() ⇒ find out a variable is integer or not.

lists in python

you can add any datatype in a single list

a = [10, 20, True, "hello", 10.6] ⇒ this is valid

⇒ we can fetch values in array with index

a[0] ⇒ 10 ⇒ The square brackets used to create list

⇒ we can also store duplicate in single a list (which we can't do in set)

list operations

to access elements we use index

a = ["hello", "abc", "dc", "marvel"]

a[0] = "bt" → replace by bt

⇒ you can use string sliding on array also.

→ remove (a value to remove) ⇒ used to remove value from list

⇒ pop (index) ⇒ pop (remove) element at a index

⇒ append (value) ⇒ add new value at end of list

⇒ del (a[index]) ⇒ this will also delete that specific value from array

⇒ you can also add list

a = ["abc"], b = ["bdc"] → a = a + b

↓

a = ["abc", "bdc"]

⇒ to add multiple value you can use extend fun

extend ("val1", "val2", "val3")

↑ extend add multiple values to list at once

⇒ len() ⇒ function to get length of list

⇒ sum() ⇒ find sum of all numbs in integer list

⇒ max() ⇒ find max from list

⇒ min() ⇒ find min from list

sort() ⇒ sort list.

to sort in reverse dirn

list.sort(reverse = True)

#Boolean values

True or False are used in many case.

In integer 0 ⇒ false 1 ⇒ true
↳ anything which is non zero ⇒ true
↳ zero ⇒ false

for decimal ⇒ 0.0 ⇒ false
anything other ⇒ true

for string ⇒ "" ⇒ false ⇒ empty string ⇒ false
" " ⇒ true nonempty string ⇒ true

#Tupples

- these are immutable means once created there is no scope of change in future.

→ tuppke is basically a immutable list.

tpl = (4,8,9,10) → a tuple

⇒ If you do
tpl[0] = 10 will throw error

⇒ So If you want unchangable data, so we use tupple

⇒ you can read tupple using indexing like list
index() ⇒ find index of a value

⇒ you can't change element in tuple but you can extend a tuppls

Ex: tpl = (4,8,9) + (8,9)
tpl = (4,8,9,8,9) → extended tupple

⇒ If you want to change tupple you can type cast it and all

set

⇒ non indexed ⇒ can't perform slicing ⇒ each element exist only one time
⇒ set comes with many math functions
- function add() ⇒ add a value
⇒ a set can have data of any type
a = { 1, True, "hell" } ⇒ possible

⊙ remove(val) ⇒ remove specific value from set
⊙ pop() ⇒ returns first value of set
⊙ discard() ⇒ so if you want to delete element but you are not sure it is present in set or not so use discard
↓
If element not present shll not raise error

---

set operations

Some set function like intersection, union can used on two set and useful in some case like autocorrect.

functions ⇒ set1.intersection(set2)
set1.difference(set2)
set1.union(set2)

Note ⇒ ⊛ set1.difference(set2) is different than set2.difference(set1)

⇒ symmetric difference is sum of above both

set1.difference(set2) ⇒ that mean item that are in (set1) but not in (set2)

Dictionary

keyword ⇒ dict.

It is a key value pair (just like javascript objects)

{key : value, key : value, key : value}

→ dictionary contain comma seperated key value pairs.

Ex: item = {
↑ dictionary
"name" : "ghansham",
"edu" : "graduation",
"job" : False,
}

→ you can have dictionary, array set inside a dictan

a = {
"name" : {
dictionary → "fname" : "ghansham",
"lname" : "salunkhe",
}
array → "edu" : ["10th", "12th", "B.Tech"]
"m" : {1,2,3,4}, → set
"n" : (1,2,3,3,4,5) → tuple
}

⇒ commas are important

operations in dictionary

you can use key ~~value~~ to extract value from dictan

- a["name"] ⇒ {fnam ——— 11 ———} give value

- you can reassign value also

a["name"] = "ghansham salunkhe"

Can use get

dict.get("key") to get value for a key

⇒ a.pop("key") ⇒ used to remove a element from dictionary

⇒ delete [[a["key"]]) ⇒ also used to delete key value pair

⇒ a.keys() ⇒ give list of keys in dictionary

⇒ update function used to update a key value pair

a.update ({"key" : "value"})

dictionary are easy yet helpfull keep them in mind.

Decision control

(A) elsf conditions

find max among a,b,c

→ bracket not mandatory in python

```
if (a>b and a>c):
    return a;
elif (b>c):
    return b
else:
    return c
```

pseudo code

⇒ elif ladder is used to do multiple condition check and display result according to multiple conditions

---

to combine condition we use ~~&&~~ (and) (or) keys

Ex: a>b or a>c ⇔ a>b and a>c

== equal to	!= not equal to	equality & inequality operator
> ~~greater~~ than (~~le~~greater)	< less than	>= & <= equality + another

⇒ we use and & or keywords to join conditions.

and			Or		
T	T	T	T	T	T
T	F	F	T	F	T
F	T	F	F	T	T
F	F	F	F	F	F

a=15

a>10 and a<20

true

(true)

Join two conditions

Say
0 → 10 ⇒ child
10 → 18 ⇒ teen
18 → 35 → young
35 → 60 → sr. citizen
60 → > → old

(and & or used to join conditions)

write code for this

Read more

ternary operation

conditions: if (code if true) else (code if false)

Ex: find bigg among two a, b

a > b if a (If true) else b (if false)

⇒ use ternary to find best among 3

Ex: a > b and b > c if a else (b > c if b else c)

⇒ conditional operation

a = ["hell", "bt", "shit"]

if "bt" in a:
    print ("present")
else:
    print ("not present").

also you can use not in

→ not keyword can used with condition to find negation (reverse any condition

Ternary

(code if true) if (condition) else (code if false)

① find big among two

c = a if a>b else b

find bigg among 3

d = a if a>b and a>c else (b if b>c else c)

=) Important to know & keep in mind that how ternary operator work

---

Match Cases

match parameter :

case first :
    do code

case two :
    code

case _ :
    default()

}

=) not so much (imp)

just keep in mind

loops

→ used inorder to run some code multiple times

(A) while loop

```
while condition :
    code to repeat
```

⇒ increment / decrement must be kept under consideration as else we can go in infinite loop

Break ⇒ break ~~code~~ loop for some condition

continue ⇒ if we want to skip a specific condition we use continue keyword (skip something)

pass ⇒ placeholder for code

If at some position we want to type but not yet at that time we put pass there

(#) also there exist

```
while condition :
    code
else :
    code
```

while else

when condition false else run one time & loop ends

(jump)

⇒ if condition of continue is true at that time code below continue will skipped

---

for loop

run from start to end-1

Ex: (A)

```
for i in range(start, end):
    code
```

c = [1, 2, 3, 4, 5]

(B)

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

↑ ~~i is~~ i has value 1, 2, ... 5

(C) we can access data from list, dictionary, tuple set for all iteration we can use for loop

⇒ no increment or decrement important in for loop it will do it by itself.

Arrays

functions in python

→ these are some argument which we can use in function

(A) parameters the values which pass to function

⇒ used to transfer data to function.

→ the order in which pass argument to function that matter

(#) keyword arguments

we can set default value so at declaration

```
def function (param1="value", param2="value")
```

so if we not past value explicitly default value will used.

⇒ If we pass argument value like

function (10,"20") ⇒ order matter but

function (param2="value", param1="now")

↑

(*) as we using ~~arra~~ keyword argument from function to pass parameter value at such time order not matters

⇒ say some time you don't have idea of how many parameters you get at that time we can use arbitary positional argument

Ex *val ⇒

```
def func (*val, str) {
}
```

you can pass any n no. of values and you can use *val as tuple

⇒ *val avoid problem of positional arguments.

```
def maxamongAll (*val):
    return max(val)
```

easy & nice.

*val ⇒ adjust multiple values at once

val ⇒ is considered as tuple in function

Print

if you want to print in single line

Print ("content", end = " ") → space

by default end = "\n"

try this

If you have comma seperated values in prin

print (1, 2, 3, 4, sep = " + ") → by default sep = " "

here

Remember

=> sep & end are two parameters in function of Print

Arbitary keyword arguments

use setdefault() if you want to set some default key value pair

If we want to pass key value pairs to function so we use key word argument

** used → a dichonary

def function (**kargs):

print (kargs)

you can pass

function (name = "grs", age = 7)

such like that

=> with * you can pass a tupple & with ** you can create a dichonary inside function

=> Internally stored as dichionary

=> you can pass any no. of arguments as parameter and whole considered as a dichonary

Return (not mandatory)

you can return some value from function & for that we can use return

lambda functions also called anonymous functions

one line function ⟶ not more than that

Say def ~~fn~~ root(num):
    return pow(num, 0.5)

so in lambda fn

⇒ lambda num : pow(num, 0.5)
(num: parameter; pow(num, 0.5): code for body of function)

and you can store lamba function as variable

fun = lambda m : pow(m, 0.5)

⇒ fun(); to function call lambda function

⇒ small function created in lamba

⇒ say you have array & for each element you want to call function at such time you can use lambda function

---

Classes ⇒ A template for data creation

⇒ class is blueprint of object
⇒ we can create any number of objects from single class

Ex: For declaring class, class keyword used, className should capitalized

```
class Hell:
    pass
```

~~def __init(self)~~
~~def __init(self)~~

a = Hell()

new keyword not needed in Py to create Object

this is used to create object a of class Hell

Each class has attribute/fields and functions

⇒ when object create init function from class will be called

```
class Hell:
    def __init():
        print("init called");
```

A = Hell() ⇒ output ⇒ init called.

⇒ Self is notation to current object and managed by interpreter so don't worry about it.

⇒ If you want to give some attributes to class so add them in __init__() fun

```
class Hell:
    def __init__(self, name, mis):
        self.name = name
        self.mis = mis

a = Hell(name = "grs", mis = 1119030303)
```

⇒ Such like that we can create objects and add attributes to class

Such like that & you can access class attribute or function using dot operator

```
print(a.name, a.mis)
```

Add new attribute.

It is very easy and just like dictionary

```
a.newattribute = value
```

and this will add new attribute in that object not class.

→ also `a.name = "sham"` → change/update value

Delete (same like dictionary)

`del(a.name)` ⇒ name attribute will deleted from class

object functions

class has some attributes & member functions

```
def func(self):
    code
```

→ inside class & you can easily create fun inside classes

Ex: rectangle class

```
class Rectangle:
    def __init__(self, length, breadth):
        self.length = length
        self.breath = breadth

    def area(self):
        return self.length*self.breadth

    perimeter = lambda self : (self.length + self.breadth)*2
```

thus you can give attribute & member function to classes

Classes & objects.

inheritance is basic facility provided by classes

```
class A:
    code

#inherit class B
class B(A):
    code
```

→ now B will inherit class A

see inheritance In Python code — github

⇒ to access function from sv parent

super() keyword is used

Modules

as ⇒ alias

Import & from keyword make your access to another file's data, library.

import filename as &

from

you can use function, data or classes from another file

you are familiar with this already

① import ~~from~~ from from

you can create folder and store all source file code there and treat that folder as library & file as library

just add `__init__.py` file in that folder and that folder will act like a package

strings (more about strings) → Backward slash

slashes can be used for escape sequence

\n ⇒ new line \t ⇒ tab

\" ⇒ double quotes \' ⇒ single quotes

\\ ⇒ backward slash

⇒ r"\\\\" r"\\\\\\"

⇒ r" string " ⇒ r create raw string so (\n \t) are not considered as escape sequence

⇒ raw string are useful in case of (Url)

% string substitution (we use in CPP)

These are different string formats %d ↓int %c ↓ch %s ↓string

(A) name = "grs" ; age = 17

% | print ("%s is %d ! " %(name, age))

after % order matters

such like that you can use string formatting in python

easy do a search & you get it

(B) fstring

print (f "{name} is {age} ! "

(C) format ("{0} is {1} ! ", format (name, age))