Python

Intro:

* Python is an interpreted, object-oriented, high-level Programming language that can be used for a wide variety of applications
* Python is a powerful general-purpose programming language
* First developed in late 1980s by Guido Van Rossum
* Python is an open source programming language
* Guido Van Rossum Named it after the BBC comedy TV series "Monty Python's flying circus"

Advantage:

* easy to learn and use
* large number of libraries available that can be used in your projects today
* High level language
* Huge community
* object-oriented language
* Portable across operating system

Organizations using Python

* Google
* Microsoft
* facebook
* youtube
* Mozilla
* cisco
* Quora etc....

Python Based jobs

- dat Analysis
- Artificial Intelligence
- Automation
- Web Application
- Desktop Application
- Hacking
- School students

High level languages

High level languages like C, C++, Java, Python etc... are very near to English. It makes programming process easy. However it must be translated into machine language before execution. This translation process is either conducted by either a compiler or an interpreter. which is also known as source code.

Machine Code

Machine languages are very close to the hardware. Every computer has its machine language. a Machine language programs are made up of series of binary pattern (Eg: 110110) It represents the simple operations which should be performed by the computer. Machine language programs are executable so that they can be run directly.

Compiler (C++)

A compiler transforms code written in a high level programming language into a machine code or any other intermediate code. It converts the text that a programmer writes into a format the CPU can understand. The conversion manner is different. It converts the entire source code in one go and reports all the errors of the program along with the line numbers. After all the error are removed, the program is recompiled and after that the compiler is no needed in memory as the object program is available

Interpreter (Python)

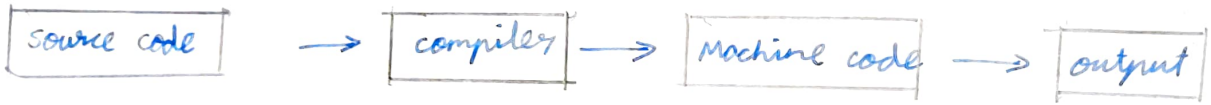
The Interpreter converts the source code line by line during Run Time, translates it into machine code or virtual machine code. Interpreter allows evaluation and modification of the program while it is executing. If there is error in any line, it reports it at the same time and program execution cannot Resume until the error is rectified. Interpreter must always be present in the memory every time the programs run. It is first interpreted and then executed.

Diff btw Compiler & Interpreter

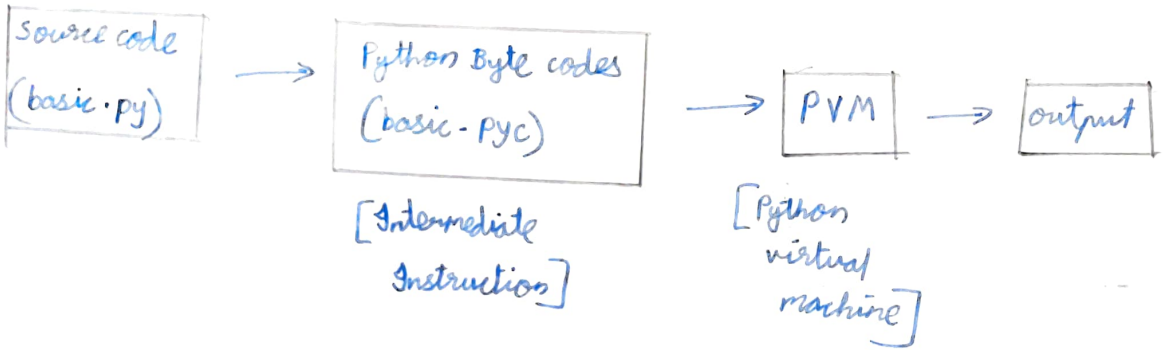
Compiler	Interpreter
• Read whole program at a time	• Read's the program line by line
• Translates whole program at a time into machine code	• Translates only one line of the program at a time into machine code.
• Takes more time to analyze the source code	* Takes less time to analyze source code
* faster	* slower
* Requires more space	* requires less space
* Languages like C, C++ use compiler	* Languages like Python, Ruby use interpreter.

what computers do after entering our source code...

Compiler works



Interpreter works



Spyder

Input :

```
Print ("I Love You LAPPy")
```

Output :

```
I Love You LAPPy
```

Variable :

- Variables are containers for storing data values
- Python have no command for declaring a variable
- A variable name must start with a letter or the underscore (_) character
- a variable can't start with a number
- A variable name can only have alpha-numeric characters and underscores. (A-Z, 0-9, _)
- Variable names are case sensitive (Ram, ram, RAM are different)

Eg :

```
name = "Ram"
user_name = "Ram"
name2 = "Ram"
```

Example program :

Input:

```
a = 25
b = 25
c = a + b
Print ("Total :", c)
```

Output:

Total : 50

---

what is Id & type

Eg prog:

Input :

```
a = 10
Print (a)
Print (type (a))
Print (id (a))
```

output :

```
10
<class 'int'>
140733886576
```

→ data type

→ Memory location

Keywords:

Keywords are the reserved words in python

Keywords cannot be used as a variable name, function name or any other identifier (class etc...).

Keywords are case sensitive in python.

Keywords are...

False await else import pass None break except in raise True class finally is return and try for continue lambda as def from nonlocal while assert del global not with async elif if or yield

Program to view all keywords

Input:

```
import keyword
print (keyword.kwlist)
```

Python keywords.py

output:

shows all keyword...

Program for getting input...

Input:

```
name = input()
Print(name)
Print(type(name))
```

→ shows its datatype

Output

```
____
```

← Type anything, it will returns it

eg: Bro

```
Bro
<class 'str'>
```

Addition by getting input.....

```
a = int(input("Enter the value of A:"))
b = int(input("Enter the value of B:"))
c = a + b
Print(c)
```

Output:

```
Enter the value of A: 100
Enter the value of B: 43
143
```

Multiplication of decimal numbers....

```
a = float (input ("value of A: "))
b = float (input ("value of B: "))
c = a * b
Print (c)
```

Output:

Value of A: 10.23
Value of B: 13.54
138.5142

Multiple inputs in a single line....

```
name1, name2, name3 = input ("Enter 3 names:").split()
Print ("Name1 :", name1)
Print ("Name2 :", name2)
Print ("Name3 :", name3)
```

Splits

output:

Enter 3 names : Ferrari Rolex Apple
Name 1 : Ferrari
Name 2 : Rolex
Name 3 : Apple

2:45 mins
1
hr

Nested If in Python

i (col)

j (row)

end = " " makes the words joined

" " new line

5 rows so we have to run it 5 times

In Laptop

Many Inbuilt functions and

Packages are saved in

Lapy with examples...

20.6.23

Python-OOP

Class :

A class is a blueprint or serves as a template from which individual objects are created

Object :

Object is an instance of a class which consists of methods and properties

Example for class :

class car :

Data Member (states):

Model

Colour

brand

model : swift
colour : red
Brand : BMW

Member function or Method (Behaviors)

Start

Accelerate

Brake

syntax :

class demo():

Pass

LAPPY...

syntax

```
class demo():
    pass
a = 10
print(type(a))
```

everything we use here is class type

'pass' is a keyword used to stop error as there is no definition

demo → class name

```
print(type(demo))
print(isinstance(a, int))
```

true

type of class [demo] is 'type'

```
class car():
    pass
swift = car()
print(isinstance(swift, car))
print(type(swift))
```

car → class name

→ instance [object]

→ gives True

using 'isinstance' to find whether it is a class' object

→ car

Single Inheritance:

In C++ note

Multiple Inheritance:

In C++ note

Multilevel Inheritance

In C++ Note

Diamond problem:

In C++ note ...

read
reading mode

save a code in a file and know its location

Go to Python application [spyder]

Projects → open

code:

r = read

```
file = open("Lappy.txt", "r")
Print(file.read())
```

output:

data read from lappy.txt

— — — —

```
file = open("Loppy.txt", "r")
Print(file.readline())
```

output

reads the first line

~~some of~~ more & more are saved in Loppy → view

NXT concept →

Install "SQLite Browser"

⇒ search this in a search engine

Download according to Sappy [ram etc...

⇒ install it

Graph Turtle

first "import Turtle"

The statements will be written as turtle.

if wanted to use our desired words use 'import Turtle as t

---

use of "import time"

delays the screen closing time

* its a package

---

to create a screen

turte . getscreen()

---

time . sleep (your timing)

delays the execution for a given number of seconds.

float division

30/20

>>> 1.5

Integer division

30//20

>> 1

---

(1) From sympy import isprime

sympy is a library which can perform symbolic variable, Algebric Manipulations, Equation solving, calculus, Linear algebra, Trignometry, Geometry, it works not only for numerical values but also

---

use of import

* is used to bring in external

modules or packages into your code

---

isprime():

* It is commonly used to check whether a given num is prime or not.

Full form of sympy

↓

Symbolic python

means it allows you to work with

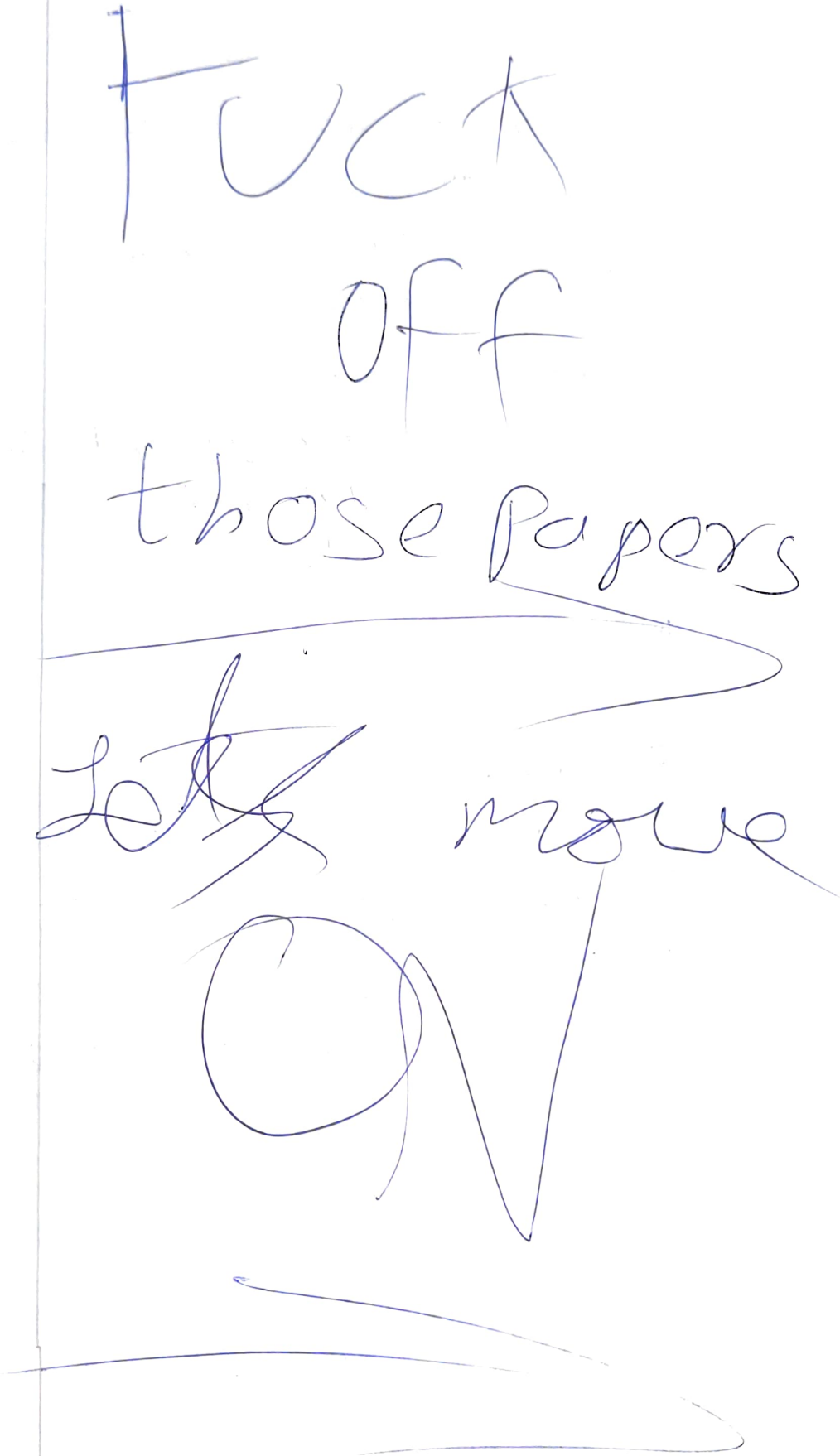
CSV → common separated values
↳ simple file format used to store tabular data such as database or database.

② import numpy : (Numerical python)
used for numerical and array operation.

③ ~~import~~

③ import Pandas : (Data Manipulation and Analysis)
used for data Manipulation and analysis
makes easy to work in SQL tables.
* loading and reading filelike (CSV, SQL, Excel)
* Data analysis: perform various task
* can be easily integrated with libraries like Matplotlib and Seaborn.

④ ~~seaborn~~ import seaborn : (Statistical data Visualization)
Seaborn is built on top of Matplotlib and provides a higher level interface.


fuck
off
those papers
Let's move
on

Libraries and Modules

1. Sympy : (symbolic Python

allows to perform symbolic calculation, manipulate algebric expressions, solve equations, perform various mathematical operations symbolically rather than numerically.

i) symbolic Algebra

Eg:

```
from sympy import isprime
Print(isprime(3))
```

Out:

True

ii) solving Eqn

iii) calculus :

includes, differentiation, integration, can find derivatives, integrals, limits. etc...

iv) linear algebra :

includes, Matrix, finding determinants...

v) Trignometry & Algebric simplification

vi) Polynomials :

can solve or create

vii) cat complex Numbers :

Supports and allows to perform symbolic operations with complex numbers.

Note:

some modules must be used by importing specific functions or items from them rather than importing the entire module. avoid naming conflict or to reduce memory usage when you only need a specific part of a module.

2. Random:

[from Random import randint

This function generates pseudo random numbers. The numbers appear to be random but are generated using an algorithm and a speed value.

Eg: from random import randint
Print (randint (3, 14))

import random
Print (random.randint(3,14))

Out:
a random value will be choose from 3 to 14

(for integer value →

(for float value →

~~we use as~~
~~import random~~
~~Print (random.random (3, 14))~~

Eg: from random
Print (random.uniform (2.5, 5.6))

Out:
random float value ...

3. OS : (operating system)

Provides a way to interact with operating system, allowing you to perform various file and directory operations, access environment variables and execute system commands.

i) working with Directory ~~Dictionaries~~ and files :

os.getcwd()
↳ get current working directory ~~dictionary~~

os.listdir()
↳ list files and dictionaries in a specified ~~data~~ directory

os.mkdir()
↳ create a new dictionary

os.path.join()
↳ join one or more path components to create a complete path

ii) creating and removing directories

os.mkdir()
↳ create a new directory

os.makedirs()
↳ create a directory and its parent directories if they don't exist

os.rmdir()
↳ Remove an empty directory

iii) File operations :-

os.rename()

↳ rename a file or directory

os.remove()

↳ Remove (delete) a file

iv) Environment variables :

os.environ

↳ Access and modify environment variables

Note :

MyDocuments → Directory

Resume.docx → file

Invoices → directory

Invoice1.pdf → file

Photos → directory

Vacation → directory

Beach.jpg → file

Mount.jpg → file

⇒ Directories ⇒ a folder in graphical user interface (GUI)

⇒ Directories provide structure and organization

⇒ files hold actual data

⇒ Both are the building blocks of file system, allowing us to store, organize, and manage data on our computer.

APP using kivy

create an virtual

To install

Python -m pip install --upgrade pip wheel
setuptools virtualenv

Python -m virtualenv kivy_venv

kivy_venv | scripts | activate

Python -m pip install ~~docutils~~ docutils pygments

pypiwin32 kivy.deps.sdl2 kivy.deps
.glew;

Python -m pip install kivy.deps.gstreamer

string

```
character_name = "wick"

Print("John" + character_name)
```

output:

John wick

datas

strings
desimal/digits
Boolean
exponent

```
phrase = "MD boss"
Print(phrase.upper().isupper())
```

output:

True

```
Phrase = "MD boss"
print(Phrase.index("M"))
print(Phrase.index("boss"))
print(Phrase.index("z"))
```

Output

0

3

~~not found~~

```
def cube(n):
    return n*n*n
Print(cube(3))
```

output

27

```
def cube(n):
    n*n*n
cube(3)
```

output

27

```
def sq(n):
    return n*n
R=sq(2)
Print(R)
```

output

4

```
def code(n):
    return n+n
    print("Bro")
res= code(2)
print(res)
```

output

4

"Bro" won't be printed as return breakes the code

Dictionaries

```
Rat = {"one": "white", "two": "black"}
print(Rat)
print(Rat.get("two"))
print(Rat["one"])
```

2-D list

~~number grid = [1, 2, 3, 4~~

```
num_grid = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    [10]
]
print(num_grid[0][0])
print(num_grid[2][1])
```

index of element

index of the chosen element

output

8

nested for loop

 same without any print()

```
for row in num_grid:
    Print(row)
```

output

```
for row in num_grid:
    for col in row
        Print(col)
```

Giraffe language 

```
E = "aeiouAeiou"
L = input("Enter the words to change:")
for i in L:
```

try:
    value = 10/0
    num = int(input("Enter a num:"))
    print(num)
except ZeroDivisionError as err:
    print("Divide by zero")
except ValueError:
    print("invalid input")

Reading txt file

```
F = open("fun.txt", "r")
print(F.readable())
F.close()
```

output

True

```
F = open("Fun.txt", "r")
print(F.read() for readline())
F.close
```

output
reads all reads first line

```
F = open("Fun.txt", "r")
Print(F.readline())
Print(F.readline())
Print(F.readlines())
Print(F.readlines()[1])
F.close()
```

output:

1. reads first line
2. " second "
3. reads in array
4. reads the array of given index

```
F = open("Fun.txt", "r")
For File in F.readlines
    Print(File)
F.close
```

output

reads all

.txt
.html

} can be used for both

appending a txt file / .html

↳ adding new data to existing file

```
F = open("fun.txt", "a")
F.write("\nHave a lot of fun")
F.close()
```

output

Nothing will be on console

use reading ← text will be modified and saved

use read to be in console

writing a txt file or html by .html

↳ overwrites your file / creates a new file

```
F = open("Fun.txt", "w")
F.write("Have fun")
F.close
```

output

will be written or created a new file

can use r+w
r+a
r & t is default

modules & library

```
import useful_tools
Print (useful_tools.roll_dice(10))
```

list of python modules in website

in cmd prompt

```
Pip install python-Docx
Pip uninstall
```

Concatenating

first = "1"
second = "2"
print(first + second)

Output

12

first = int("2")
second = int("2")
print(first + second)

output

3

more values to be
a constant
then go for
tuple

Array

Arrays are similar to list but, one difference "all values must be of same type" either in list we can have any type of value".

ie) int array means only int value
float " " " float "

but one advantage in array
ie) they don't have any specific size
ie) fixed size

ie) you can expand, shrink

for eg: we have five value
if you want ten value
then u can expand

to add an element we can use append()

can access an element using its index

the place it is useful

in college → 10 stud → a proj/give,
→ to calculate mark1 ... mark10
in this case array will be more useful

array() is not by default

so we import array

in this we have a function called array()

instead of using array.array()
all time use as
{ import array as ar
ar.array() }

or we can specify the function we wanted to work with from a library →

```
from array import array *
vals = array(...)
```

↳ first mention type, the mention values

* ⇒ represents, to work with all functions of the library array

typecode	ctype	pythontype	Min. size in byte
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'f'	float	float	4
'd'	double	float	8

```
from array import array *
vals = array('i', [5,9,8,4,2])
Print(Vals)
```

out: ✓

(what if one of the element is float)

```
from array import array *
vals = array('i', [5.6,7,8,9])
Print(Vals)
```

out: error

---

but if one of the element is '-' with the type 'i' it works but with type 'I' it shows error

ie) cannot convert '-' value to unsigned int

buffer_info() → gives size of array

from . . . .

```
vals = array('i', [5,6,7,8])
Print(vals.buffer_info())
```

output:

(57571891011, 4)

address

size

within tuple separated by comma

→

typecode() → to know the type working with array

append(element) → to add a value

remove(value) → to remove a value

reverse() → to reverse the value

eg:

```
from array . . . . *
vals = . . . . .
vals.reverse()
Print(vals)
```

output: ✓

from . . . . .

vals = ...

Print(vals[0])

↳ index

output: what is index '0'

actually it is like list

---

to print all using index value

from . . .

vals = ...

for i in range(5): (or) len(vals)

Print(vals[i])

(or)

vals

output:

*

*

*

*

---

of course u can work with chars but use 'u' as it is unicode

```
from array import *
vals = array('u', ['a', 'b', 'c'])
Print(vals)
```

---

to create a new array with same values in existing array

```
from . . . .
vals = . . .('i', [. . . .])
new_vals = array(vals.typecode, (a for a in vals))
Print(new_vals)
```

{a for a in vals}

means take one and assign again & again till last is done

{new_array = array(vals.typecode, (a for a in vals))

take datatype of vals

take and assign element of that array

to assign a squrat of existing array in new_array

Just do

new_array = array(vals.typecod, (a*a for a in vals))

Note :

while loop

Step 1: initialization
" 2: condition
" 3: increment or decrement

An array is basically a data structure which hold more than one value at a time. it is a collection or ordered series of elements of the same type

extend ( ) → to add more elements at end of an array

insert( ) → add to desired position

concadination

```
a = array(i, [5, 6])
b = array(i, [7, 8])
d = array(i, i)
d = b + c
Print(d)
```

concadinating different data type leads to error

---

slicing array

```
import
a = array(i, [0, 1, 2, 3, 4, 5])
Print(0:3)
Print(0:-2)
```

Print(a[::-1])

↳ it does not reverse an array, it forms a reverse copy of array

---

```
for x in a[0:-3]:
    print(x)
```

Prints specific element

syntax:

Var = Array(Typecode, [Elements])

a = array ('i', (1, 2, 2, 3, 4))

a.remove (2) will be removed

Print (a)

output

('i', 1, 2, 3, 4)

array with numpy

```
import numpy as np
arr = np.array([1,2,3,4], dtype=np.float64)
my_array = np.array([1,2,3,4])
print(my_array[0])
my_array[2] = 10
print(my_array)
```

can access using index

can modify

arrays not only to store data
also can perform operations
in entire array

numpy provides many functions for array & sort

min & max is one of theses functions

```
import ... as np
my_array = np.array([1,2,3,4])
Print (np.min(my_array))
Print (np.max(my_array))
Print (np.sort(my_array))
Print (np.flip(my_array))
```

→ for sorting in ascending order

↓ for sorting in descending order

array can be used to find mean, median and Matrix

```
import . . . .
my_ar = np.array([1,2,3,4,5])
Print (np.mean(my_ar))
print(np.median(my_ar))
print (np.std (my_ar))
```

→ for standard deviation

→ for Matrix

```
import numpy as np
matrix1 = np.array([1.1,2.2,3.3],[4.4,5.5,6.6])
matrix2 = np.array([7.7,8.8],[9.9,10.10],[11.11,12.12])
result = np.dot(matrix1, matrix2)
Print(result)
```

for Matrix multiplication

output:

```
[[66.913  71.896]
 [161.656  174.262]]
```

to transpose a Matrix

```
transposed_matrix = np.transpose(matrix2)
Print(tran..._matrix)
```

we can transpose the result too

```
from numpy import *
array0=array([6,7,8,9,10])
arr1=array([1,2,3,4,5])
arr1=arr1+5
Print(arr1)
arr=arr1+arr0
```

5 will be adde to each element

adding 2 arrays

```
Print(log(array0))
Print(sqrt(array1))
Print(sum(array0))
```

Find log value

Find Square root

Find Sum

- id(arr)
- flip
- sort — Dsc / Asc
- Max
- Min

unique()

to find unique element

Copy array

```
arr1 = arr(. . . . . )
arr = arr1
Print(arr)
```

both arr and arr1 have same address

```
from num import *
arr1 = . . . .
arr2 = arr1.view()
Print(arr2)
```

→ arr1 in different address

two copy →
- shallow copy
- deep ,,

shallow copy

```
from num
arr1 =
arr2 = arr1.view()
arr1[1] = 7
Print(arr1)
Print(arr2)
```

changing element in arr1
changes in arr2 too

don't want this

shallow copy

instead of view()
use copy() for deep copy

arr2 = arr1.copy()

if changes happen in arr1

it does not change in arr2

---

faster than list

---

np.arange(8)

→ output

[1, 2, 3, 7, 8]

→

import

→ Numerical python

for → linear algebra,
Matrix, fourier transform
functions related arrays

creates n-d array

ie) n numbers of dimensional array

working of ndarray faster than list

Version of numpy

Print(numpy.__version__)

Most of the part of numpy in created by C & C++
only least by python

array( )

↓

Pass list/tuple

* 0-D Array (ie containing only one element)

```
import numpy as ap
a = ap.array(54)
Print(a)
```

ap -> alias name
ie) Alternate name

↓

54

* 1-D Array

```
import ....
a = ap.array([1,2,3])
```

→ [1,2,3]

↳ we can call this as an array of 0-D array elements

* 2- D array

```
import . . .
a = ap . array([1,2,3],[4,5,6])
Print(a)
```

↓

we call as array of 1d array elements

* 3- D array

```
import . . . .
a = ap . array([1,2,3],[4,5,6],[[7,8],[9,10]])
Print(a)
```

↓

array of 2d array

Dimension - Level of depth of array

↓

ndim()

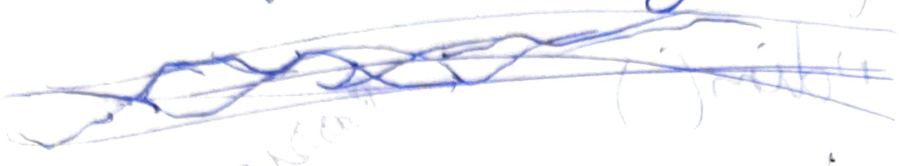
↓

type(*)

Basic Primitive

array does not support multi dimen array

Thats why we use numpy



```
from numpy import *
arr = linspace(0, 16, 16)
                start, stop, step
```

can use range( )
but, in range(##) 16 is excluded rather here its included. and the step is not like in range( ) but, its separates 16 times from 0 to 16
ie) breaking the range into parts

```
Print(arr)
```

Try to understand

```
from numpy import *
arr = arrange(1, 15, 2)
print(arr)
```

first element, last, step

here, the ~~step~~ step is not like in linespace( )
but like in range( )
ie) it doesn't separate the range but, increment or decrement

---

```
from ...
arr = logspace(1, 40, 5)
Print(arr)
Print('%.2f' % arr[4])
```

Try, to understand

```
from numpy import *
arr = zeros(5)
arr1 = ones(4)
Print(arr)
Print(arr1)
```

```
from numpy import *
arr1 = array([
        [1,2,3],
        [4,5,6]
    ])
```

```
# check whether it is 2-d array
Print(arr1.ndim) -> 2
Print(arr1.shape) -> (2,3)
                      ↓   ↓
                    row  column
Print(size) -> size of entire block
```

convert 2d array to 1d array

```
arr2 = arr1.flatten()
Print(arr2.ndim) ---> 1
```

convert 1-d to 3 d array

```
arr3 = arr2.reshape(3, 4)
Print(arr3)
```

row ↓ column →

```
arr4 = arr2.reshape(2, 2, 3)
Print(arr4)
```

↓

2 array having 3 array

Try to understand

$$\begin{bmatrix} 9 & 6 & 7 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

column matrix

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$[4\ 6\ 7\ 8]$$

row matrix

Matrix

```
from numpy import *
arr = array([[1,2,3,4],
             [5,6,7,8]])
m = matrix(arr)  #Method 1
Print(m)
m1 = matrix('1 2 3 4;
             5 6 7 8')
Print(m1)
```

↓

4 row
2 column

```
m2 = matrix('1 2 3; 4 5 6;
             10 11 12')
Print(m2)
```

↓

3 row
3 column

print diagonal elements

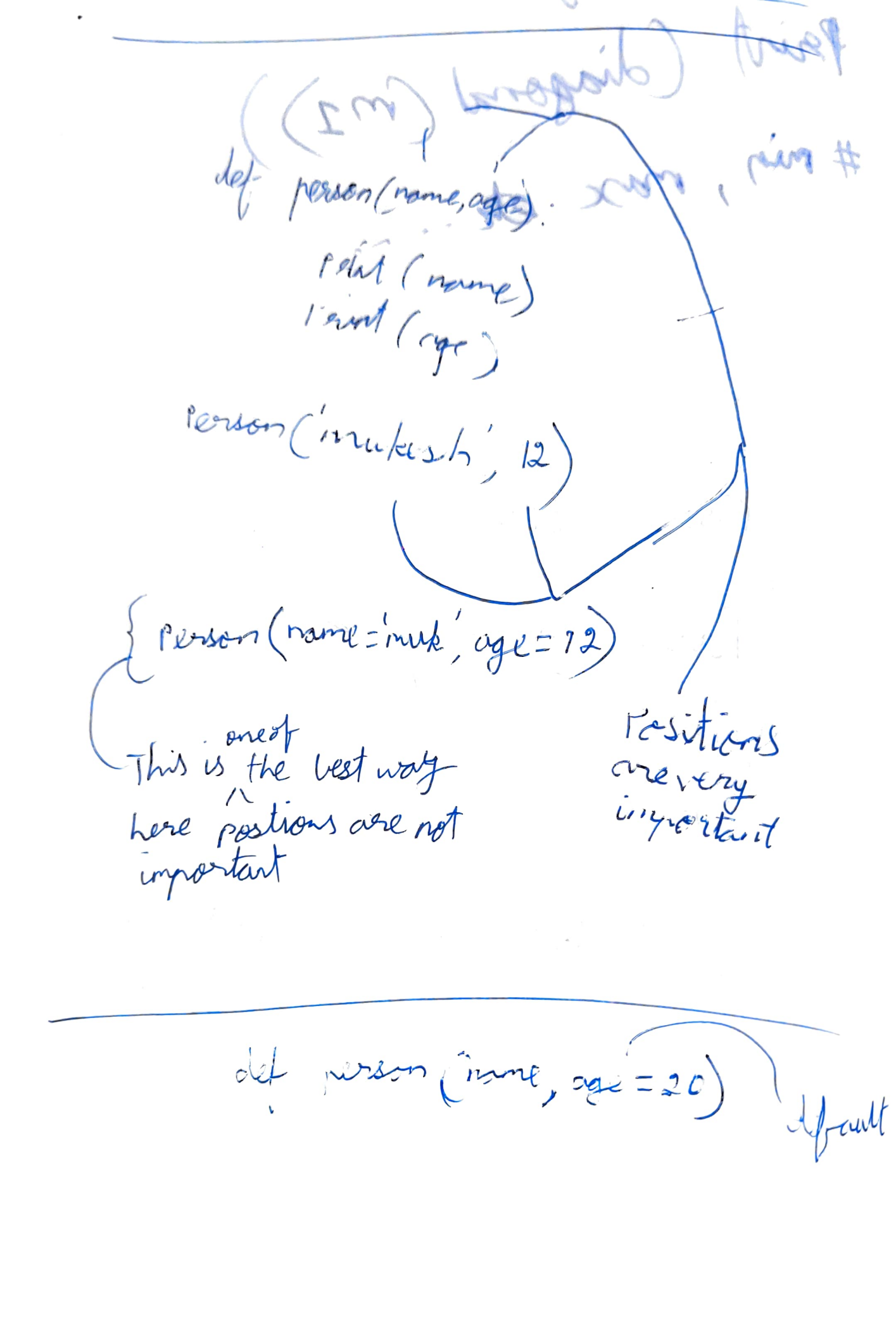
print (diagonal (m1))

min, max etc....

Pass by Value

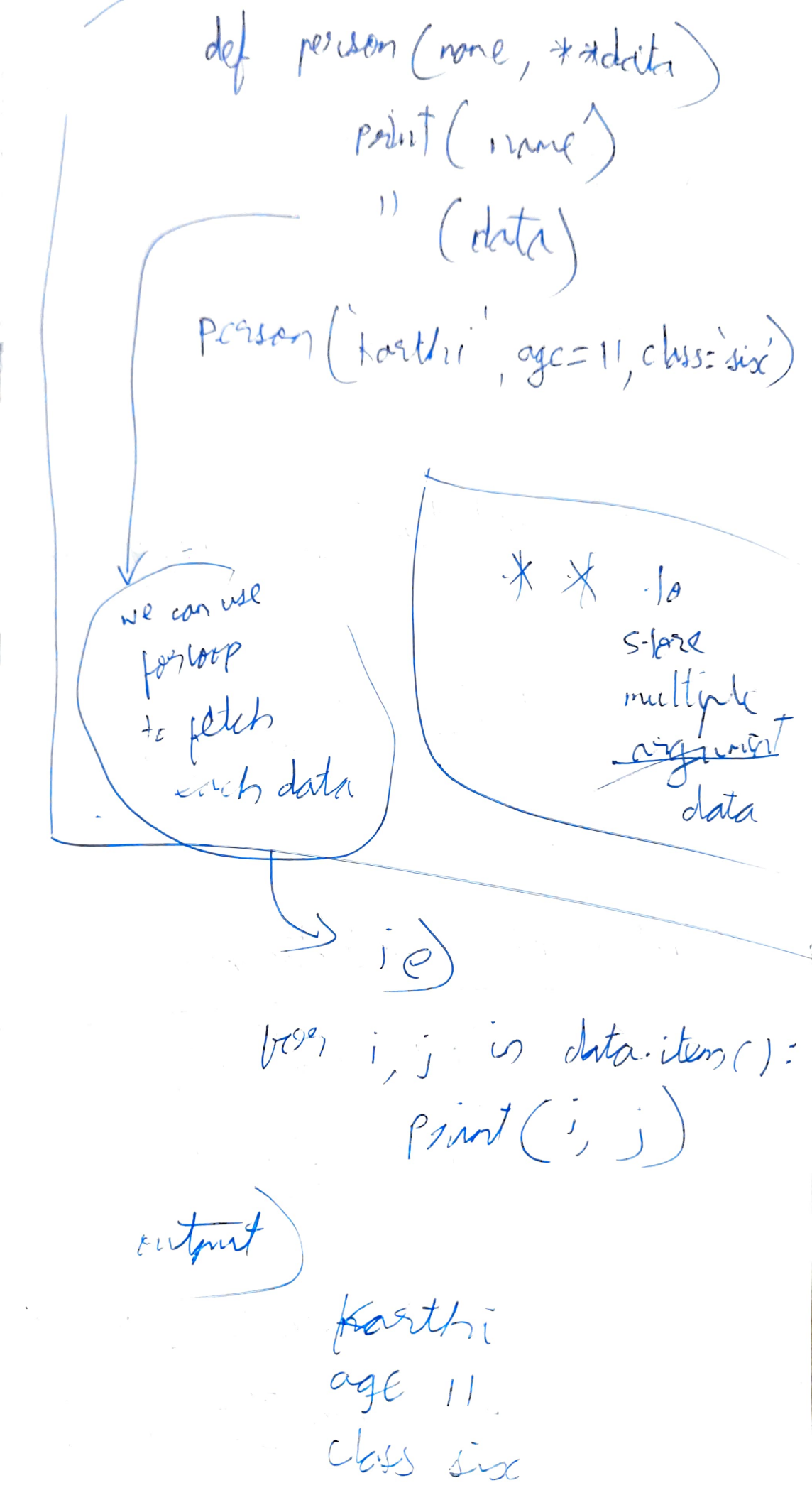
Pass by reference

We won't use in Python



def person(name, age=20)

default


def person(name, **data)
print(name)
print(data)
person('karthi', age=11, class='six')
we can use for loop to fetch each data
** to store multiple argument data
for i, j in data.items():
print(i, j)
output
karthi
age 11
class six

for loop trick

Trick using list

list1 = [1, 2, 3, 4]

" 2 = [5, 6, 7, 8]

for item1, item2 in zip(list1, list2):

    print(item1, item2)

another logic using dictionary

data = {'numbers': [1, 2, 3, 4],
        'letters': ['a', 'b', 'c', 'd']
}

for i in range(len(data['numbers'])):

    print(data['...'][i], data['...'][i])

```
# tuple in list
data = [(1, 'a'), (2, 'b')]
for num, letter in data:
    print(number, letter)
```

---

new using item() function

```
data = { 'k1' : 'happy'
         'k2' : 'sad' }
for key, value in data.item():
    print key, value
```

One cool example

```
class item:
    def __init(self, num, letter):
        self.num = num
        self.letter = letter

data = [item(1, 'a'), item(2, 'b'),
        item(3, 'c')]

for item in data:
    print(item.number, item.letter)
```

There is no global 'a' & local 'a' once we use 'global' keyword. it will be 'a' global variable can be accessed both inside & outside of a function

Next page

if wanna edit a global variable inside a function, then use 'global' keyword

to just access inside function we don't need to use 'global' keyword

2 eg's

```
a=10
def No():
    a=15
    print(a,'fn')
No()
print('out',a)
```

output

```
15 fn
out 10
```

```
a=10
def No():
    global a
    a=15
    print(a,'fn')
NO()
print('out',a)
```

```
15 fn
out 15
```

To break that ⭐

we use globals() function where we can both access 'a' as local and global.

```
b = 6
a = 10
def Nothing():
    a = 5
    x = globals()
    Print('in', a)
    Print('G', x)

Nothing()
Print(a)
Print(b)
```

~~can~~ it access both the variable 'a' and 'b'

if wanna access only one, then

⭐ → x = globals()['b'] (or) x = globals()['a']

if you wanna change the value in `x`, the don't change like

```
{ x = 16 (or) x = 5
```

change like this

```
{ globals()['a'] = 12
  globals()['b'] = 100
```

because if you change like

```
{ x = 16
```

it will assign a new variable

Recursion

There's a limit for recursion, it's (1000)

if want beyond 1000 then import 'sys' & use sys.setrecursionlimit( )

limit ← is numbers

```
import sys
sys.setrecursionlimit(2000)
def greet():
    Print('Hello')
greet()
```

To ensure if recurs 2000 times

```
import sys
sys.setrecursionlimit(2000)
i=0
def greet():
    global i
    i += 1
    print('Hello')
    print(i)
greet()
```

~~output~~

recursion means a function calling itself

New

Filter() → To filter and then

Map() → To change values

reduce() → for eg: ~~three~~ multiple value to one

belongs to functools module

along with lambda

① Filter()

* it takes two arguments

filter(function, iterable)

↓ for logic

↓ ie) list / sequence in our upcoming example

filter() depends upon true or false

if true, returns if not doesn't

Example:

```
def is_even(n):
    return n%.2==0:
lst = [1,2,3,4,5,6,7,8,9,10]
even = list(filter(is_even, lst))
Print(even)
```

output:

[2, 4, 6, 8, 10]

→ we coustomly use this function for filter, in case of this to be replaced, we use lambda. to replace coustomity

so,

~~lst = [1,2,3,4,5,6,7,8,9,10]~~

~~even: (lambda n: n%.2.~~

```
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even = list(filter(lambda n: n%.2==0, lst))
Print(even)
```

(2) Map ( )

take some value and apply some function

ie) to change values

Eg: find even and double it

It also takes 2 arguments, same as before

map (function, iterable)

~~def update (n):~~

~~return n +~~

```
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even = list(filter(lambda n: n%2==0, lst))
double = list(map(lambda n: n*2, lst))
print(even)
print(double)
```

(3) reduce()

To reduce Multiple value to one or as ~~one~~ accordingly to our desired output

{from functools import reduce

it also takes two argument

{reduce(function, iterable)

```
from functools import reduce
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even = list(filter(lambda n: n%2==0, nums))
double = list(map(lambda n: n*2, even))
sum = reduce(lambda a, b: a+b, double)
print(sum)
```

first add 1 & 2 & then 3 with 3 & 4 & 10 with 5 & 6 .....

we didn't use list as we expect a single output

Tkinter

* You cannot die as a rat in this race
* Experience & skills speaks
* At the end of your day dreams must be tr

* kill Love, Murder Pain, Feel dreams and work for it

* if you can't afford sacrifice and change you will be a looser

* Take risk for your dreams

* forget family, friends, Love, Trust, money & everything except Books, tutorials and Laptop

* There's No Time

* Don't waste your time on Practicing for Rat Race

* knowledge wins