15/10/23 PYTHON => Guido van Rosum

Language

* It runs slow when compared to other compiler Python
* By C programming Language is created
* To make python faster generator is used.
* Related to python => cython, Jython, Iron Python.

Interpreter.
* It will execute line by line

Example:

```
print("Jawan")
print(Leo)
print("Vikram")
```

RAM
↓
Temporary Storage

Once you execute python file information will get stored in RAM and once the execution is completed the information in RAM will be deleted.

R - Read
E - Evaluate
P - Print
L - Loop.

REPL is a python interpreter

go → created by google
programming language (Intespreter).
Java Script => Intespreter
created for Netscape company.

official Documentation
↓
python.org
MDN, W3C

* Go & Java script is Interpereter Language like python
* Each Language has its own documentation

Version => major . minor . patch

major ↑ features, minor ↑ features, ↑ like security updates

avi-game-dev.vercel.app

Python 2.0
Python 3.0
Python 3.10.12
} Versions

Python 2.0 Vs Python 3.0

print "Hello"
keyword

print("Hello")
built in function

5/2 => 2

5/2 => 2.5

PEP => Normal people can suggest feature to improve in python.
python enhancement proposal.

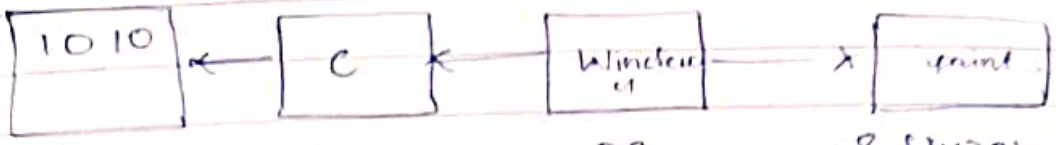
Eg: Sum = 1+5
print(Sum) ✓
print(Sum = 100)
print(Sum := 100) ✓ 3.8

There is a operator called walrus operator (:=) because of the animal walrus
:= => allows you to assign values in function calls.

* Python doesnot supports pointers because the of security issues instead id() function is used to fetch the address.
* Main application of python is AI, ML,

Application:
① Large no. of Libraries in python
② AI/ML libraries
③ Web development
④ Games
⑤ Automations
⑥ Scientific calculation

TENSOR FLOW
provided by Google
For AI & ML projects



Machines drivers — OS C++ — Software (Python).

Program:

```
import time
for x in range(5):
    time.sleep(1)
    print(x, end=" ")
```

sep → applicable in between the elements.

Expected output will be achieved ⇒ `print(x, flush=True, end=" ")`

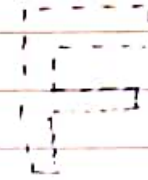
OUTPUT: applicable at the end of the

blinks for 5 sec and the output will display as

0 1 2 3 4

print() → it is in single line → 0 1 2 3 4 (Stack) → flushed → Output

CLI Vs GUI

CLI
- use ascii art
- Eg. 
- CLI
- * We mostly prefer ascii arts in CLI
- Eg: people use CLI in servers.
- * CLI is faster / flexible when compared to GUI

GUI
- * GUI is mainly used for to improve user experience
- Eg: Windows, Android
- 1-4 files at same time).

Ctrl + D
Duplicate

Single → will place cursor on the next line
Double → word
Triple → Entire line.

print()
* value .....
Eg. print (A, B, C)
* sep = " "
It is used to place values inbetween strings values passed in function
* end = "\n" "\n"
It is used to keep string at the end of values.
* flush = False.

Ctrl + Arrows → used to move between words
Shift + Arrows → Select characters
Ctrl + shift + Arrows → Select + Move

-i → to open interpreter in after the execution of code.

PROGRAM (ANIMATION).

① import time
itr = 0
li = ['\\', '|', '/', '-']
for x in range(40):
    print(" " * x + li[itr % 4] + ">=[ISRO]")
    itr += 1
    time.sleep(0.25)

② import art
art.tprint("MARVEL")

/ \ -> forward slash is to keep single line code to multiple lines.

6/10/23 PROGRAM

```
a=1
b=2
temp=a
a=b
b=temp
```

```
a=1
b=2
a,b,t=10,20,a+b
print(t)
```

3//

a,b,t=1,2,a+b
Name Error: 'a' is not defined
In python when you press enter then only memories get allocated.

```
adhi=30
roopa=20
anu=10
madhu=1
total=adhi+1
roopa+1
anu+1
madhu
print(total)
```

control + back slash (\)
=> to get # comments & uncomment the line.

; is used to write multiple codes in a single line.

Eg: a+2 a=1+2; print(a)

I python => Intelligent python

* Python will not support increment & decrement operators

a=5 -> a++ = 6
++a = 5

Eg: Unary minus & Unary Plus.

30++10--+10--100+-10 = 140

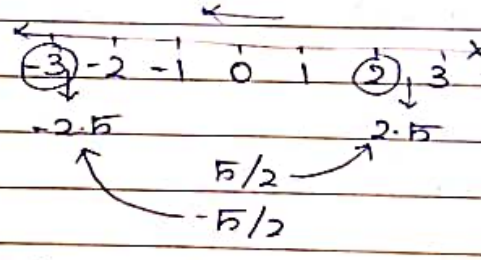
+- (Unary) has more priority than +- (Add, Sub)

BODMAS

* As Python is used for scientific calculation. Mostly python, execute same as normal mathematics.
* Priority
  Unary will follow (r to l)
  * / // % (l to r)
  + - (l to r)

* In multiplication where a*b will result in int when a & b are int.
* a*b will result in float if any of the value is float

* In division where a/b results always in float type.
  Eg: 4/2 = 2.0
  Floor division
  Eg: 5//2 => 2
  -5//2 => -3
  5//-2 => -3
  -5//-2 => 2



* In floor division the result is always towards negative
Eg: a//b
  (i) first get the result of a/b.
  (ii) The result which you get convert into integer type.
  (iii) Don't forget to move -ve side.

a//b -> a/b

output is always float.

5//2 2

5.0//2 2.0

* The result sign of a%b is not as expected because, Internally it uses floor division

$D_{(nt)} = D_{(or)} * Q + R$ $\boxed{Q = a//b}$

$-5//2 \rightarrow -5/2 = -2.5 = -3 \Rightarrow Q$

$-5 = (2 \times -3) + R$

$R = 1$

$5\%-2 \rightarrow 5/-2 = -2.5 = -3 \Rightarrow Q$

$5 = (-2 \times -3) + R$

$5 - 6 = R$

$R = -1$

$-5\%2 \rightarrow -5/-2 = 2.5 = 2 \Rightarrow Q$

$-5 = (-2 \times 2) + R$

$-5 = -4 + R$

$R = -1$

The result sign for a%b will be same as denominator sign.

$5\%2 \Rightarrow 1$

$-5\%2 \Rightarrow 1$

$5\%-2 \Rightarrow -1$

$-5\%-2 \Rightarrow -1$

$-11\%3 \Rightarrow -11/3 = -4 - 3.7 = -4$

$-11 = (3 \times -4) + R$

$-11 + 12 = R$

$R = 1$

$0\%-0 \Rightarrow 0$

$0 \Rightarrow R$

Error: division by zero.

$-4\%-4 \Rightarrow -4/-4 = 1$

$-4 = (-4 \times 1) + R$

$-4 + 4 = R$

$R = 0$

'=' Equalto

```
>>> a = 5
>>> b = 5
>>> c = 5
>>> id(a)
  140149672886640
>>> id(b)
  140149672886640
>>> id(c)
  140149672886640
>>> d = 6
>>> id(d)
  140149672886672.
```

Assignment operator
'=' => a = 5.
5 a is not assigned to a
Here 5 is refering to a.

* Compairson Operator
  >, >=, <, <=, ==, !=
  30 > 20 > 10.
  Only used once
  Compairson Operator works from left to right.

CODE:

```
def fn(x):
  print("value", x)
  return x
>>> fn(30) > fn(20) > fn(10)
value 30
value 20
value 10
OUT: True
```

Chaining Operation.
  23 == 65 == 456

* Membership Operator.
=> in, not in
Eg:
  li = [0,9]
  0 in li
=> True.
  5 in li
=> False.

* In 'is' operation the value between -5 to 256 will return True

```
>>> a = -5
>>> b = -5
>>> a is b
True
```

4.5 => int(4.5) = 4.

32 ≤ x ≤ 100 => 32 <= x <= 100.

Python is just similar to math

"Sakthi" and "Sakthi" is not Sakthipriya.

Similar to English Language so it is called as High level Language.

Comparison Operator, Relational Operator and Membership Operator. it supports chaining operation will written always boolean type (True False)

Exponentiation - (** 1)

* Logic Operation

Logical	False	True
int	0	± integers
float	0.0	± integers, 0.001
bool	False	True
Complex	0+0j	remaining values (ex: 4+ij)
string	" " empty string	remaining string.

If there is a space it is true otherwise it is false.

List	[]	[1,2,3]
Tuple	()	(1,2,3)
Dictionary	{}	{'a': 54}

* AND Operator

```
>>> 50 and 90 and 34 and 60 and 8
                T      T      T      T
>>> 8.
```

>>> 50 and 90 and 0 and 60 and 8

= 0

>>> def fn(x):
... print(x)
... return(x)

>>> fn(56) and fn(90) and fn(0) and fn(44) and fn(8)

56
90
0

* While using and first
  (i) See the value on left
  (ii) If it is true move forward, if it is false stop there
  (iii) If it is false stop there & return the value.

* OR
  * If it is false move forward if it is true no stop there.
  * If it is true stop there & return the value.

Eg:

>>> 0 or 45 and 0 or 456 and 0.01 or 'hi' and '' or 5+66+10 and 55+90

= 0.01

* NOT Operator.
  In Logical Operator not is special because it will return in boolean where and & or ~~written~~ return values.

Eg:
>>> not not not not 45
True
>>> not not not
False.

Bitwise Operator.
>> → Right Shift
<< → Left Shift
DATA TYPES.
① Int
* Int consist of +ve value -ve value & zero
Usage of _ in numbers
>>> 100000 = amount

>>> amount = 1_00_000
* Here _ is used make no. more readable
* We cannot use _ side by side.
>>> 00_00__000
Invalid.
* you should use only one _ and it should always placed between the digits.
>>> 0_0_
>>> _000
Syntax Error.
* you should leading zeros in decimal integer literal are not permitted
>>> time = 05.
Invalid

Binary.

0 0 1 0
2^3 2^2 2^1 2^0
8 0 2 1 =>2

a = 0B1001
Here a = 9 is stored

```
impor pyperclip
pyperclip.copy(str(dir(pygame)))
```

```
>>> 0.0          >>> 12.
= 0.0            = 12.0
>>> 0.           >>> .12
= 0.0            = 0.12
```

9/10/23. Exponential

```
>>> e          >>> 0b10e1x
>>> e1x        >>> -2**2 → -4
>>> 1e x       >>> (-2)**2 → 4
>>> 1e1 ✓      >>> a**-b => 1/a^b
>>> 1E1 ✓      >>> a**0.5 => a^1/2 => √a.
```

COMPLEX NUMBER.

* In complex number by default all the values are float.
* Real part will accept both float & integer
* Imaginary part will accept only float

```
0001+0001j
Error: leading zeros in decimal integer literals
>>> 1+001j.                    are not permitted
(1+1j).
```

* We should not use '_' before j

```
>>> 1+0001_j
Error:
```

6.2 ..... $\times 10^{-17} + ij$

$4 - 4j^2$

8

```
>>> (-1) ** 0.5
(6.123233.....e-17 + 1j)
```

Complex donot support a > b, a <= b.
Complex supports -a, +a, a+b, a-b, a * b, a/b, a ** b.

```
>>> fn(6) > fn(0) > fn(20) > fn(300) > fn(400)
value 6
value 0
value 20
False
```

```
def fn(x):
    print("value:", x)
    return x
```

```
>>> a = 2+2j
>>> a > 2
```

* Comparision between 'complex' & 'int'. 'complex' & complex will not supported

* In complex comparision operator is not supported

Identity Operator => is, is not
It is used to check reference.

Boolean

* In boolean we have two values True & False
* >>> True + 1
  => 2
  Because internally True refers to 1.
* >>> False + 0
  => 0
  Because False refers to 0.

* C is a loosely typed language

```
int a = 1;
float b = a;
```

* Java is a strictly typed datatype

```
int a = 1;
float b = a;
Error
```

* Python is a no typed — Dynamic datatype

STRING OPERATION.

Eg:

```
path = "C:\new\backup\tree"
r -> raw
>>> print(path)
C:
ebackup  ree
```

```
>>> path = r"C:\new\backup\tree"
>>> print(path)
C:\new\backup\tree
```

Eg:

```
>>> len('hai')
3
>>> len('\new')
3
>>> len(r'\new')
```

Example.

```
import time
li = ['|', '/', '-', '\\']
for x in range(40):
```

```
import time.
str = '\/-\\'
for x in range(10):
    time.sleep(0.5)
    print(str[x%4],
    end=" ", flush=
    True)
```

print(li[x % 4])
time.sleep(0.5).

Example:

① import time
print("loading : ", end=" ")
for x in range(100):
    print(str(x)+"% ", end=" ", flush=True)
    time.sleep(0.5)
    print("\b"*3, end=" ", flush=True)

② import time
l = "loading:"
len_l = len(l)
for x in range(100):
    print(l+str(x)+"% ", end=" ", flush=True)
    time.sleep(0.05)
    print("\b"*(len_l+4), end=" ", flush=True)

pdb → python debugger
bug, debug, breakpoint()
vs code → Visual studio code → debugging tools.
Extension → Python (Microsoft)

Requirements:
* Python Software
* Vs code

* Open vs code in any folder, create a file main.py
* Write some sample code in main.py
* In left side there will be a icon run & debug.
* Breakpoint → Go to the line where you want repoint hover Hover line no.

Classes & Objects

* In class we have datamembers & member function. Where datamember function is also called method.
* Datamethodmember is also called as attributes.

Example:

```
class Student:
    db={
        'sinchan' : ['py', 'c'],
        'bochan' : ['r', 'js'],
        'kazama' : ['js', 'cpp'],
        'neni' : ['js']
    }
    def update(person, lang):
        Student.db[person].append(lang)
            append(lang)
class Trainer:
    db={'vijay' : ['js'],
        'surya' : ['r'],
        'ajith' : ['py'],
        'samantha' : ['c', 'js'],
        'nayanthara' : [cpp, 'tcl/tk']
    }
    def update(person, lang):
        Trainer.db[person].append(lang)

Class Cat:
    Pass
mia = Cat()
coco = Cat()
Simba = Cat()
```

Instantiation
↓
Instance of class.
↓
Process of creating objects from class

* Here mia is object and cat is a class

Instantiation

The process of creating objects from class is called instanciation.

Eg:

```
class Cat:
    ears=2
    legs=4
    tail=1
    fur_color=" "
    def sounds():
        return "meow meow"
    def hunt_prey(food):
        return food+" yummy..."
```

```
>>> mia.
<__main__.cat Object at 0x7fde78eg7eb0>
>>> type(mia).
<class '__main__.cat'>
>>> mia.fur_color="grey"
>>> simba.fur_color="orange"
>>> coco.fur_color="black".
>>> coco.fur_color
'black'.
>>> Cat.sounds()
'meow meow'
>>> mia.sounds()
TypeError: Cat.sounds() takes 0 positional arguments but 1 was given
```

* If any changes happen in class all objects get affected

Eg:

```
>>> cat.legs = 2
>>> mia.legs
2
>>> simba.legs
2
```

Variables can be assigned like this

```
>>> cat.paws = "cute"
```

10/10/23

①

```
class Cat:
def sounds(x):
    return "meow..."
mia = Cat()
simba = Cat()
print (mia.sounds())
```

Cat.sounds get called

even though I am not passing anything

Some value is getting passed.

```
res = Cat.sounds(mia)
print(res)
```

②

```
class Cat:
    def sounds(self):
        return "meow..."
mia = Cat()
simba = Cat()
print(mia.sounds())
res = cat.sounds(mia)
print(res)
```

③
```
mia = Cat()
simba = Cat()
Cat.paws=4
mia.sleep=18;
print(simba.paws)
print(simba.sleep)
```
error.

④
```
class Cat:
    def sounds(self):
        self.sleep=23;
        print("sleeping with sounds", self.s)
mia= Cat()
simba= Cat()
mia.sounds()
simba.sounds().
```

Class → has its own attributes & methods
variable.
class variable → attributes that belongs to class.

Object → has its own attributes & methods
Instance variable → attributes that belongs to object.
Instance method will have 'self'

* class variable:
  * Variable which belongs to class is called class variable.
  * This is common / accessable to all object
* Static methods. (class method)
  * If there is no self keyword then the method is called static method

* Instance Variable
  * Variable belongs to that object alone

Eg:
mia = 18. sleep = 18 where mia - obj
sleep - attribute

Eg:
self. sleep = 24 where self → object
sleep → attribute.

* Instance methods
  * method which accept self arg we can say instance methods.
  * It belong to object alone

Eg:
```
def sounds(self):
    pass
```

Creating Instance Variable.

①
```
class Leo:
    def casting(self, name):
        self.actor = name
leo_das = Leo()
harold_das = leo()
leo_das.actor = "vijay"
harold_das.actor = "arjun"
Leo.casting(leo_das, "actorvijay")
Leo_das.casting("actor vijay")
harold_das.casting("actor arjun")
print(harold_das.actor)
```

Instance var created outside the class

class name should always start with capital letter.

① Class Leo:
    location = "kashmir"
    def casting(self, name):
        self.actor = name.
Leo.director = "Lokesh"
vijay = Leo()
trisha = Leo()
print(vijay.director)
# after 3 years
Leo.director = "GVM"
print(Vijay.director)
print(trisha.director)
# after 4 years.
arjun = Leo()
# class variable director is
# converted to instance variable.
# only to arjun object.
arjun.director = "Vignesh"
print(arjun.director)
print("VIJAY house: direc:", vijay.director)
Leo.director = "Lokesh Kanagaraj"
# arjun doesnot have access to
# class variable (director)
print(arjun.director) # Output = Vignesh.

② Class Sanji:
    hair_color = "yellow"
    eye_brows = "curly"
    habbit = "smoking"
    def memories(): → Static method.
        print("a boy with "+Sanji.eye_brows+"
                                eye_brows")
        print("a boy with"+age 5.eye_brows+"
                                eyebrows").

print("Only sad story... so don't ask me")

age5 = sanji()
age10 = sanji()
age30 = sanji()
sanji.memories()

③ class Ben:
    @classmethod
    def transform(cls):
        print("transformed to alien :", x)
Ben10000 = Ben
Albedo = Ben
fourarms = Ben()
# Ben.transform(Albedo)
Print(Ben)
print(forearms)
Output
<class '__main__.Ben'>
<main__.Ben object at 0x000002014B48900>

def fn():
    return 100
res = fn()
another_fn = fn()
    instacemethod
mia.sounds()
object method

* By default obj get passed => Instance method
    classmethod
cat.sounds() => By default class get passed
class method

https://tinyurl.com/kit-opps
https://tinyurl.com/kit-student-oops

Class method
@classmethod
def fn(cls):
    pass

Instance method
def fn(self):
    pass

>>> cat.sound()
cat.sound(cat)

>>> mia.sounds()
Cat.sound(mia)

11/10/23

dunder → double+underscore

PROGRAM:

```
class Soldier:
    def __init__(self, val): # constructor
        self.strength = val
                ⟶ def __del__(self):
p1 = Soldier(9)                 print("destructor")
p2 = Soldier(5)
p3 = Soldier(4)
p4 = Soldier(7)
```

Constructor:
* Constructor will get call when object is created.
* In python to create a constructor we need to use method __init__
* __init__ is called as magic methods
* Magic method is also called as dunder methods

Destructor.

* Destructor will get call when object is removed from memory.

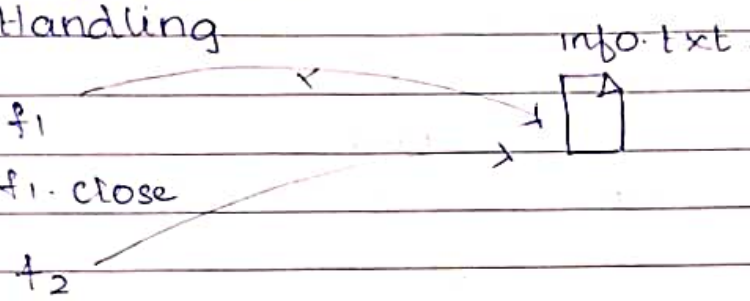
* To create a destructor we have to use a method __del__

Eg:

```
>>> p1 = soldier(9)
>>> del p1
```

Usage of destructor

1. File Handling




```
f1 = open("info.txt","r")
.....
f1.close()
f2 = open("info.txt","r")
.....
f2.close()
```

PROGRAM:

```
class Soldier:
    def __gt__(self, obj):
        if self.strength > obj.strength:
            return self
        else:
            return obj
    def __init__(self, val, name): #constructor
        self.strength = val
        self.name = name
```

```
def def __del__(self): #destructor
        print("destructor")
p1 = Soldier(8,'kakashi')
p2 = Soldier(9,'naruto')
p3 = Soldier(7,'gara')
p4 = Soldier(10,'pain')

leader = p1>p2>p3>p4
print(leader.name)
```

Operator Overloading

* Operator Overloading is used to overwrite the functionality of operators.

Eg: '+' operator is used to add 2 numbers

Example.

```
class KTM:
    chassis = "trellis frame"
class Triumph:
    engine = 598
class Bajaj(KTM, Triumph): #inheritance
    def newBike():
        return "RS 600 {0} cc engine with {1}
                                chassis".\
        format(Bajaj.engine, Bajaj.chassis)

print(Bajaj.newBike())
```

Inheritance.

* Inheritance is called as when some features used from parent class called is used by child class.

* KTM is & Triumph is parent class & Bajaj
* Chassis is a feature of KTM & engine is feature of triumph.

CONCEPTS & IT'S USAGES

1. Oops
   * Use oops only if for that is managed by more than 1 person.
   * Even if the project is done by 1 person but you are going to maintain this project for long time (Go oops if the project is for long time)

2. Generators.
   If you want to execute your code faster go for generators.

3. Functions
   Functions are used to avoid repetition of code
   for memory management

4. String formatting.

5. Dictionary.
   Use when you have datas in key-value pairs

6. List.
   When we want to store more than one value in variables we have to go for list. It should be editable. Duplicates are allowed.

7. Tuples.
   Same as list but the values are not mutable.

8. Set
   No duplicates are allowed.

Help function:

* Help(x) where x is a module
* I will get list of methods, classes & variables
* Help(x) where x is a function
* We will get brief details about what functions will do.
* Help(x) where x is a class.
* It will give list of all methods & class variables
* Help(x) where x is a object
* We will get list of methods, class variables along with instant variables

dir

* dir(x) where x is a module, we will get list of method, class, global variable without any description.

9 Frozen set

Duplicates are not changed allowed & values can be changed.

10. Range

Range is used to generate n number from start to end.

Database.

Unstructured db	Structured db
Couch db	MySQL
Mongo db	postgreSQL
	SQ Lite
	Oracle db.

SQLite.

* Open browser
* Search SQLite code for python.
* Copy the code & past in file.
* Run the code
* If there is no SQLite 3 in system
* Download SQLite3 from google.

FUNCTION

* In functions once you execute the function for the variable which is inside the function memories get allocated.
* Logic inside the function will get executed.
* After the execution of function memories created for variable will get deleted.

Generators:

* Generators are used to avoid wastage of memory.
* Generators are used to execute faster
* When you use generator python don't create all value & store it in RAM, Instead of that just assume the values are there.

PROGRAM

```
def fn():
    print("A")
    yield 456.
    print("B")
    print("C")
    print("D")
    yield 200.
```

cordova

```
yprint("A")
yprint("A")
yield 200.

O = fn() # O is store with generator obj
res1 = next(O) # O res1 will stored with 456
# A
res2 = next(O) # res2 will stored with 200.
# B
# C
# D
res 3 = next(O) # res3 will stored with 300
# A
# A.
print(res1, res2, res3)

yprint(" ..... ")
x = fn() # x is also a generator obj
for i in x:
    yprint(i)
```

* Generator can be used in a for loop where the next function is internally called automatically.

Scope.
* Builtin scope
* Global scope
* Local scope.

Builtin scope.
* yprint, input, some sum are ex. of built in scope.

* They are called as builtinscope because whenever executing the file functions are available by default

Global scope.

* A variable defined in a file is executing first will have global scope
* variable which has global scope can be accessed from any where

Local scope

* A variable which is defined inside the function is called Local scope

12/10/23 CODE:

```
>>> r= input ("Enter the value: ")
Enter the value: 4.56
>>> print(r)
4.56.
```

* Once you entered the input the input will be accepted only if enter key is pressed
* In input function we can pass any datatype.

Eg: input(4.5)
input(4.5)

* We mostly prefer string as input functions argument.
* keep a space at end of string to make it meaningful

Eg. input(" Enter a name ")
↳ space.

* you can use := instead of single space to make it meaningful

Eg:

```
>>> a = input ("a = ")
a = 56
>>>(a)
56
```

Program :

```
a = input("Enter the value of a:")
b = input("Enter the value of b:")
c = a+b
print("Result :",c)
```

Output:

Enter the value of a : 56
Enter the value of b : 100
Result 56100.

* By default input will return in string type

Input Constrain

$-1000 < a < 1000$

$-1000 < b < 1000$.

Program.

```
a: int(input(-000)
b: int(input(10)
c = l*b
print("Area of Rectangle is
  Area of Rectangle is[-0000]
```

Sample Output:

1. print("Area of Rectangle is[", c, "]", sep=" ")
2. print("Area of Rectangle is ", [c], sep=" ")

Sublimetext

3. print ("Area of Rectangle is [" ?] ". format(area))
4. print ("Area of Rectangle is [" + str(area) + "]")

① Write a Program to get n number of array inputs & give the sum of elements in array

Input Sample:
5
1. 10 -20 40 2

Output Sample
Sum: 33

PROGRAM:

① line1 = input()
line2 = intput
line1 = int(input())
li = line2.split(" ")
total = 0
for x in li:
    total = total + int(x)
t = 0
for x in arr:
    x = int(x)
    t = t + x

line1 = input()
line2 = input()
arr → ["10", "20", "30", "40", "50"]

② n = input(input())
arr = input()
li = arr.split(" ")
t = 0
for x in arr.split(" "):
    t = t + int(x)

④ n = int(input)
print(sum(list(Map(int,

③ n = int(input())
arr = input()
to = 0
for x in input().split(" "):
    t = t + int x
print(b)

④ n = int(input)
print(sum(list(Map(int,
input().split(" "))))

PROGRAM:

```
class Ball:
    def __init__(self, color):
        self.color = color
def paintRed(ball):
    ball.color = "red" # dipping ball in paint
    return ball
bag = [
    Ball("white"),
    Ball("white"),
    Ball("white"),
    Ball("white")
]
newBag = map(paintRed, bag)

for ball in newBag:
    print(ball.color)

    def __str__(self):
        return "A very pulpy looking ball with" +
                                        self.color.
    def __repr__(self)
        return "A very pulpy looking ball with" +
                                        self.color
```

FUNCTION PROGRAMMING

```
def custom_map(fn, items):
    newItems = []
    for item in items:
        res = fn(item)
        newItems.append(res)
    return newItems
print(sum(custom_map(int, input().split(" "))))
```

```
>>> [[x for x in input().split(" ")] for y in range(4)]
1  2  3 4
1  2  3  4
1  2  3  34
1  2  34  4
[['1','2','3','4'],['1','2','3','4'],['1','2','3','34'],
['1','2','34','4']]
```

13/10/23 PROGRAM:

```
import time

def deleteline(letter_count):
        print("\b" * letter_count, end=" ")
def load_animation(n):
        for no in range(n+1):
                value = "Loading" + str(no) + "%."
                ln = len(value)
                print(value, end=" ", flush = True)
                deleteLine(n)
                time.sleep(0.025)
```

Return type

```
def fn():      ->  ____
    ____ > doc       ||
    ____           object
    ____           (class
    return
```

```
def deleteLine(letter_count) -> int :
```

Eg: -> float
-> boolean
-> complex
-> None

Eg:

```
>>> def fn() -> int : pass
```

```
>>> def fn() -> list : pass
>>> def fn() -> tuple : pass
>>> def fn() -> set : pass

->dict : pass
-> bin : pass
```

* In this arrow mark we can use classes, functions and objects.

```
>>> a=1
>>> def fn() -> a : pass

>>> def fn() -> 1234 : pass
```

Ex:

```
def fn(x) : ---> Int, Float
    if x==0 :
        return 123
    else :
        return 123.4678
```

Scenario where 2 types int, float is being passed.

Polymorphism. Overloading.

Ability of an object to do different things at a different scenario.

Python doesn't support polymorphism.

Over riding

class A -> Parent class.
bicycle

class B -> child class
bicycle -> Only class method
o = B()
o.bicycle()

* bicycle method in Parent class is over ridden by method in child class.

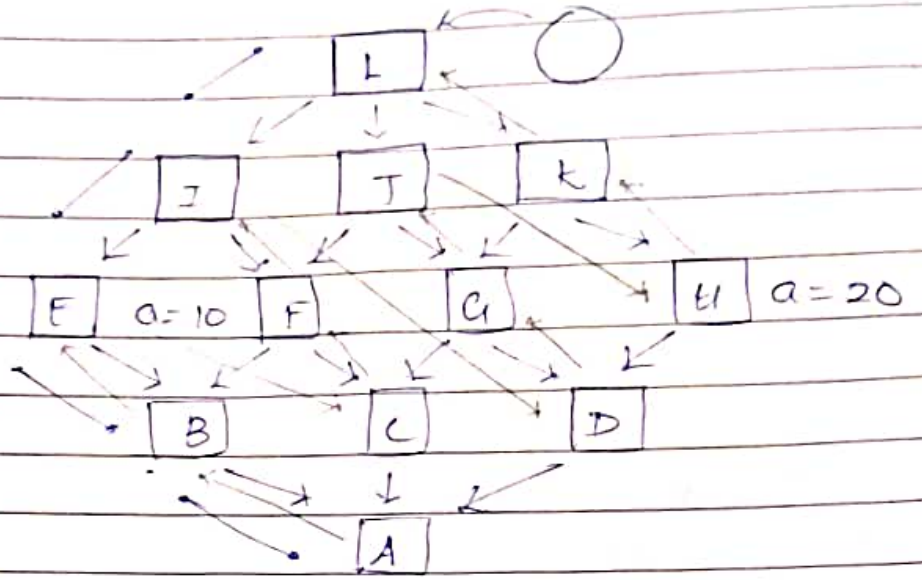
Definition:
* A method in parent class is over ridden by method in child class.

Polymorphism => many forms

Overloading

Ex:
```
def fn():
    print("no arg")

def fn(x):
    print("1 arg", x)
def fn(x,y):
    print("2 arg", x, y)
```



Obj = A()
print

MRO - Method Resolution Order.

```
>>> for x in A.__mro__
    print(x)
```

```
<class '__main__.A'>
<class '__main__.B'>
<class '__main__.E'>
<class '__main__.C'>
<class '__main__.F'>
<class '__main__.I'>
<class '__main__.D'>
<class '__main__.G'>
<class '__main__.J'>
<class '__main__.H'>
<class '__main__.K'>
<class '__main__.L'>
<class 'object'>
```

16/11/23 FUNCTIONAL PROGRAMMING.

LIST:

li = [Arjun, Aravind, Preethi]

Ex:

```
val = "apple"
li = []
li[:] = val
```

Output:

["a", "p", "p", "l", "e"]

String Methods:

capitalize	isascii	lstrip	splitlines
casefold	isdecimal	maketrans	splitswith
center	isdigit	partition	strip
count	isidentifier	removeprefix	swapcase
Encode	islower	removesuffix	title
endswitch	isnumeric	replace	translate
expandtabs	isprintable	rfind	upper
find	isspace	rindex	zfill.
format	istitle	rjust	
format_map	isupper	rpartition	
index	join	rsplit	
isalnum	ljust	rstrip	
isalpha	lower	split	

WALRUS OPERATOR (:=) => Used only in python-3

```
>>> a = 0    => 0
>>> a = a+1  => 1
>>> a        => 0
```

Example with walrus operator

```
def fn(x):
    print(x)
>>> a = 10
>>> fn(a)      => 10
>>> fn(a := a+1)
>>> a          => 11
```

fn(a = a+1) which gives the updated value of a. It will works in Java & C

Example without walrus Operator.

```
def fn(x):
    print(x)
>>> a=10
>>> fn(a)  =>10
>>> fn(a = a+1)
Type error
>>> a
10
```

EXAMPLE:

```
li = [10, 20, 30, 40]
a = 0
>>> li[a = a+1]   => Syntax Error.
>>> a
0
>>> li[a := a+1]
20
>>> a
1
>>> li[a := -a]
40
```

```
>>> li = [111, 222, 333]
>>> a = 0
>>> li[a := a+1; a := a+1]
Syntax Error
>>> a
```

```
>>> li[(a:=a+1) : (a:=a+1)]
>>> [2 2 2]
>>> a
2
>>> a=10
>>> a=a+1 => No output
>>> a+1
12
>>> a:=a+1
Syntax Error
>>> a
11
>>> (a:=a+1)
12
>>> a
12

>>> a=0
>>> a=a:=a+1 => x Syntax error
>>> a=(a:=a+1) working
>>> a
1
```

a := a+1 = a => Syntax Error

(a := a+1) = a => Syntax Error

As you know that walrus operator uses ':' '=' character. In any expression has more than one ':' or '=' there is a possibility for syntax error.

```
>>> 1+2 = a        >>> a:=a+1=a
Syntax error       Syntax error
```

17/11/23

Walrus Operator (or) Assignment Expression (or) Named Expression

* (:=) introduced in PEP 572, Python 3.8

```
>>> a=0
>>> (a:=a+1) => did only if brackets are there
>>> a:=a+1 is invalid
```

because already a=a+1 is there. To avoid confusion. This will through error

a = 1+2
↓ variable, ↓ expression

Exception case

```
>>> a=fn()  → Here the fn() returns
                value 2.
>>> a
2
>>> a := fn()
>>> a
 Syntax error.
```

```
>>> (a := fn())
>>> a
 2
```

FUNCTIONS:

* We use to write logics inside the functions.
* Used for code reusability.

User defined function.

Syntax

```
def fn name (argu...):
        -------> statements
```

Pre defined function

Eg:

```
print input.
```

```
def fn():
    pass
```

```
>>> type(fn)
<class 'function'>
```

Here def keyword is used, where it is taken from define.

User defined

```
def fn():
    print("Hai")
```

```
>>> fn()
Hai
```

```
>>> fn(1)
Error
```

>>> fn
<function fn at 0x.....>
fn name
hashed address

① def fn1():
    print("cheese")
② def fn2():
    print("bread")
    fn1()
    print("bread")

③ fn2() → function call.
bread
cheese
bread
} Output

Argument => value accepted in function definition is called argument
Parameters => value passed in function call is called parameters.

def fn(arg1, arg2):
    .....
    .....
fn(para1, para2)

LIST
li = [10, 20, 30]
indexing
li[0] → 10
li[-1] → 30
li[0.0] → Type Error
li[true] → Name Error: name 'true' is not defined

li[True*0] => 10
li[True] => 20
li[-100] => Error
>>> True+True
2.

li = [10,20,30]
li[:len(li)-1] => [10,20]
li[-1] => 30
>>> 0
0
>>> 0.0
-0.0

Slicing.

li = [10,20,30]
li[:len(li)] => [10,20,30]
li[:3] => [10,20]
li[0:3] => [10,20]

li[x:y:z]
x → inclusive
y → exclusive
start:end:step.
step → default(1)

li[:] => [10,20,30]
li[::] => [10,20,30]
li[0:len(li):1]
start end step

>>> li[::0]
ValueError: slice step cannot be zero.

li[0:100] => [10,20,30]
li[-100:100] => [10,20,30]
li[-2] => 20.
li[0] => 10
li[0:-2] => [10]
R→L => []

slicing won't do boundary check.
slicing always follows L → R.

```
>>> li[-90:-90] = [90]
>>> li[90:90] = [90].
>>> li
[90, 1, 2, 3, 4, 5, 90]
>>> li[90:90] = 'avinash'
>>> li
[90, 1, 2, 3, 4, 5, 90, 'a', 'v', 'i', 'n', 'a', 'sh']
>>> li[90:90] = 90.
TypeError: can assign only a iterable.
>>> li = [1,2]
>>> li[0] = [0,0,0]
>>> li
[[0,0,0], 2]
>>> li[0] = 'marvel'.
>>> li
['marvel', 2]
>>> li = [1,2,3,4,5,6]
>>> li[::2]
[1,3,5]
>>> li[::2] = [10, 30, 50]
>>> li
[1,2,3,4,5,6]
```

```
>>> li[::2] = [10,20]
value error: attempt
to assign sequence of
size 2 to extended
slice of size 3.
```

```
>>> li[::2] = [1,2,34,4,5,5,56,6,6,6]
>>> li
```

```
>>> li[0:0]
```

```
li[x:y] => step=1
When step value is 1
attempt to sequence
```

```
>>> li[2:2] = [4100, 4004, 400]
>>> li.
[1, 2, 3, 4]
>>> li[2]
3
>>> li[2:2] => []
```

size error will not come.

```
>>> li
[1, 2, 3, 4]
>>> li[2 2] = [999, 999, 999]
>>> li
[1, 2, 999, 999, 999, 3, 4]
```

18/11/23

EXAMPLES (DISCUSSION ON QUESTIONS)

① `>>> a = input()`
`' '"`
`>>> a`

what is the output?

Ans: `'\' "'`

* whatever In the above question the input is given as ' followed by "
* So while printing the inputvalue output should be exactly as the input value

Example:

```
>>> 77\ 90
Syntax error: Unexpected character after
              line continuation character
```

② `>>> 122\`
`... +1\`
`... 89`
`221`

* We have to remove the slash to write the expression in single line without commiting space

FUNCTION CALL:

```
def tree(branch1, branch2="lkjhg"):
    print("branch1:", branch1)
    print("branch2:", branch2)

tree(branch2="bird2", branch1="bird1")
```

no of arguments in func def shouldbe equal
No of arguments passed.

```
def fn():
    print("hi");
a = fn()
print(a)
```

```
def fn():
    pass
print(print("hello"), end=" ")
```

OUTPUT:

Nonehello.

```
def fn():
    a=10
    b=20
    return a,b.

x,y= fn()
print(x,y).
```

positional and keyword arguments.

```
def fn(x,y):
    print("values:", x, y).
fn()
fn(1)
fn(1,2)
fn(y=1, x=2)
fn(x=1, y=2)
fn(1, y=2)
fn(1, x=2).
```

FUNCTION.

* positional
* keyword
* default
* var-arg or arbitary arguments

=> we can pass different count of arg. in function calls

```
def fn(x,y):
    pass
>>> fn(10,20)
```
→ Nothing get print.

RETURN STATEMENT ⇒ • return any value
• return no value

```
a = fn(10,20)
>>> a
```
→ Nothing get print
```
>>> print(a)
None.
```

eg: a = fn() → def fn():
>>> print(a) pass
None.

* By default if there no return statement It will return None.

```
def fn():
    return 4
    return 5
a = fn()
>>> a.
→ 4
  5
```

⇒ we can use multiple return statement in single fn.

```
def fn(x):
    if x > 0:
        return 1
a = fn(-4)
>>> a
None.
```

* In other language (C, C++) the function 'fn' does not return anything you are trying to store in variable 'a'

TRICKS: (Slicing).

* li[x:y:z] will always results in list[...]
* Now in the below statement

li[x:y:z] = [...] → sequence type (rightside)

unicode

* The result which we get here is those items will be update in the above code.

Eg: li [10,20,30,40]
li [1:4:2] = [100, 200].
⇓
[20,40]

>>> li
[10, 100, 30, 200]

Error Cases:
case(i): Li=[10,20,30,40,50,60].
>>> li[0:] = 456
Type Error: always should be iterable
case(ii):
>>> li[::2] = [10,20,40,50]
Error => Left side count does not match with right side count.

STRING METHODS.

>>> val.	isdecimal();	isdigit();	isnumeric() (true when it is only mathematical)
0	True	True	True.
2^4	False	True.	True
¼	False	False	True.

>>> '0.ab'. isidendifier()
False
>> 'if'. isidentifier()
True

* isidentifier is a function will results in True if it follows rules for creation of variable

```
>>> "\'\\"'''".isprintable()
True.

>>> \b, \t, \n will result in False.

>>> "    ".isspace()
* True
>>> "".isspace()
False.

>>> "tony stark".rjust(20)
'            tony stark'
>>> "tony stark".ljust(20)
'tony stark          '

>>> "  hai  ".strip()
'hai'
>>> "----hai---".strip()
'---hai---'
>>> "----hai----".strip("-")

>>>'pakistan border India'.partition()
TypeError
>>> 'pakistan border India'.partition('border')
('pakistan', 'border', ' India')
>>> 'p bor i bor china'.partition('bor')
('p', ' bor', ' i bor china')
>>> 'p bor i bor china'.rpartition('bor')
('p bor i ', 'bor', ' china')
```

20/11/23 FUNCTION:

```
1 def total(x):
2     if x==0:
3         return x
4     return x+ total(x-1)
5 print(total(4))
```

Output: 10

function call occurs 5 times.

function call at 5, 3, 2, 1, 0

total(x->) 5 4 3 2 1 0

5, 4, 3, 2, 1, 0

Output: 6

```
def fn(a,b): → fn(a,b="value"):
    pass                  ↳ default argument
fn(40)
```

print(40); → default argument.
end= ? "\n"
sep = ? " "

```
def wish(name="guest"):
    print("GM", name);
```

wish() → GM guest
wish("students") → GM students
wish(name="Akash") → GM Akash
wish("Akash") → GM Akash
wish(msg="GE") → Type Error

```
def wish(msg, name="guest"):
    print(msg, name)
```
→ Type Error

```
wish() →
wish("Akash", "GE")
wish(msg="GM")
```

def print(
achive
we can't adure print without var-args

var-args => variable argument.

Sample code with print function.

1. print()
2. print(10)
3. print(20)
4. print(10,20)
5. print(sep=" ")

Another Example for default argument.

→ function definition

```
def wish(name="guest", msg):   syntaxerror
    print(msg, name)
print("line1")
wish("name='kavin') => TypeError
print("line 2")
#wish(name='kavin', 'GM')
print("line 3")

def wish(name="guest", msg):
def wish(msg, name="guest"):
#try use cases with above 2 func definition

def wish(msg, name="guest"):
    pass
print("line1")
wish(name='kavin') => TypeError
print("line 2")
wish(name='kavin', 'GM') => Syntax Error
print("Line3")
```

ARGUMENTS

1. Positional arguments : in func call - without assignment
2. keyword arguments : in func call - with assignment
3. default arguments : in func def - with assignment
4. non default arguments : in fun def - without assignment

* always keyword arg. follows positional argument.
* always default arg follows non default args only if default, non default present.

```
def fn():
    print ("hai")
def fn(x=90):
    print("hai 2")
fn()
# the output is "hai 2"
# because the last function definition will
# get called.
```

* We can achieve overloading in python using var-args.

Mutable vs Immutable

① >>> t = (10, [1,2,3], 30)
>>> t[1][2] = 233
>>> t[1]
Output:
1 2 233

② a = 1
b = 1
a ? b
it will result True if
(a) is
(b) ==

→ before running the script
↓
checking all the lines of code & values

In compile
>>> a = 1000
>>> b = 1000
>>> a is b
True.

In interpreter
>>> a = 1000
>>> b = 1000
>>> a is b
False

→ I can see that value 1000 is used more than once.
To avoid wastage of memory, 1000 is created only one

CODE TO PRINT a-z variable

```
>>> for i in rang(ord("a"), ord("z")+1):
    print(chr(i), "=")
```

Example

a = 1000
:
z = 1000

print(a is b) => True → id(a) == id(b) => True
'is' used for check reference
'==' used to check values

```
>>> a = 999
>>> b = 999
>>> a is b
False
>>> a=b=999
>>> a is b
True
```

```
>>> a = 888 ; b = 888
>>> a is b.
True
```

* This is True because same as python file.
* In single line we are using 888 more than once. Hence reference is created only once

* If a is b results in true a == b is also true.

```
>>> a=10000.00
>>> b=10000
>>> a==b
True
>>> a is b
False.
>>> a= 5/2 ; b=2.5
>>> a==b
True
>>> a is b.
True.
```

```
>>> a=0j
>>> b=0+0j
>>> a is b
False
>>> a==b
True
```

```
>>> a=0j ; b= 0+0j
Ts >>> a is b.
True
>>> a=1; b= True
>>> a is b.
False
>>> a ==b
True.
```

* In ascii values a is b is True for values between 0 to 255

```
>>> a = '\x00'
>>> b = '\x00'
>>> a is b
True.
>>> chr(256)
'Ā'
>>> a = 'Ā'
>>> b = 'Ā'
>>> a is b
False
```

```
>>> chr(255) => 'ÿ'
>>> a= 255 'ÿ'
>>> b= 'ÿ'
>>> a is b
True
```

* In case of string, ascii characters which has range of 0 to 255 will result True.

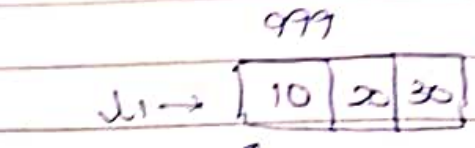
```
>>> a = (1,2)
>>> b = (1,2)
>>> a is b
False
```

```
>>> a=(1,2) ; b=(1,2)
>>> a is b
True
```

Mutable & Immutable has no effects on 'is & '=='

COPY

```
>>> l1 = [10,20,30]
>>> l2 = l1
>>> l1[0] = 999
>>> l2
[999, 20, 30]
```



Shallow copy

```
>>> l1 = [10, 20, 30]
>>> l2 = [*l1]
>>> l1[0] = 999
>>> l2
[10, 20, 30]
>>> l1
[999, 20, 30]
```

Example:

① l1 = [10,20,30]
l2 = l1
l1 = l1 + [40]
print(l1)
print(l2)
Output:
[10, 20, 30, 40]
[10, 20, 30]

② l1 = [10, 20, 30]
l2 = l1
l1 += [40]
print(l1)
print(l2)
Output
[10, 20, 30, 40]
[10, 20, 30, 40]

FUNCTIONS:

var-arg: if there is *fn in function call/function def, then it is called var-arg.
eg: def fn(*a): pass.

* in function call we can provide positional arguments after keyword arguments. (Syntax error).
* in function def we can provide positional (non-default) arguments after default arg(Syntax error).
* in fun def after if we are providing var-arg then we can provide non default arg. var after var arg.
* If we are providing non-default or default arg. after var-arg then it should be called by using keyword only
* for var-arg we cannot call it using keyword only arg. var.