"PYTHON"

I want to speak with the computer
Language = Python
- Guido van Rossum
- 1991.

Syntax :-

```
import pandas as pd.
```

Programming language application
Gaming, Banking, Machine learning

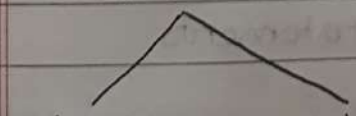
Basic variables.

Data → collection of the facts.
Ex: My name is poorna.

How to store the data
"Poorna" → String
56 → Integer
True/False → Boolean

variables.
Data/values can be stored in temporary storage spaces called variables.

Student = "Poorna"



Name associated & address associated
Student → Name of the variable.

10 = a
20 = b
a+b = 30
a-b = -10
a*b = 200
a/b =
multiple operations on variables

3) Decision Making statements.

if	else
it's raining :	go out and play
sit inside	

if	else
marks > 70 :	give practice test
Get ice-cream	

if --- else pseudo code

```
if(condition){
statements to be executed
}

else{
statements to be executed
}
```

If the "if' condition is true
at that time it will execute the statements inside if
else if it is false
It will execute the statements inside else

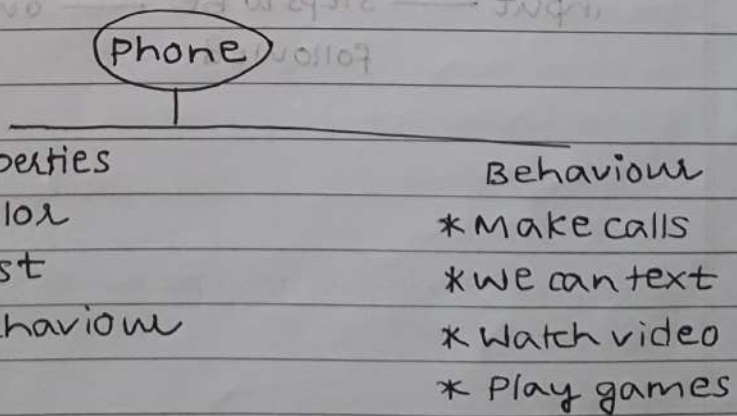
4) looping statements.
are used to repeat a task multiple times.
while, Repeat, each

while loop pseudo code

```
while(TRUE){
keep executing statements
}
```

5) Functions in programming
Function is a block of code which performs a specific task.

6) OOP concepts.
I am surrounded with the objects
Mobile, laptop, bag, bike, dog, cat
object have
→ Property
→ Behaviour
class is a template for real world entities.



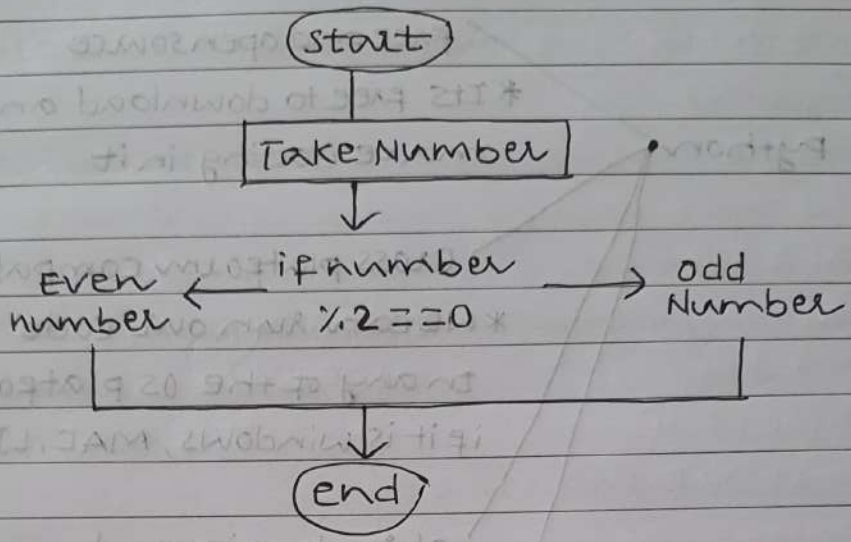
object
It has a specific template
* objects are specific instances of a class

Phone would be the class
specific instances would be the objects
Brands of phone Apple
samsung
Nokia
google pixel
Tesla
Apple is the brand of phone which is object
O/P b/w objects
All having
Different values for properties
Different values for Behaviour

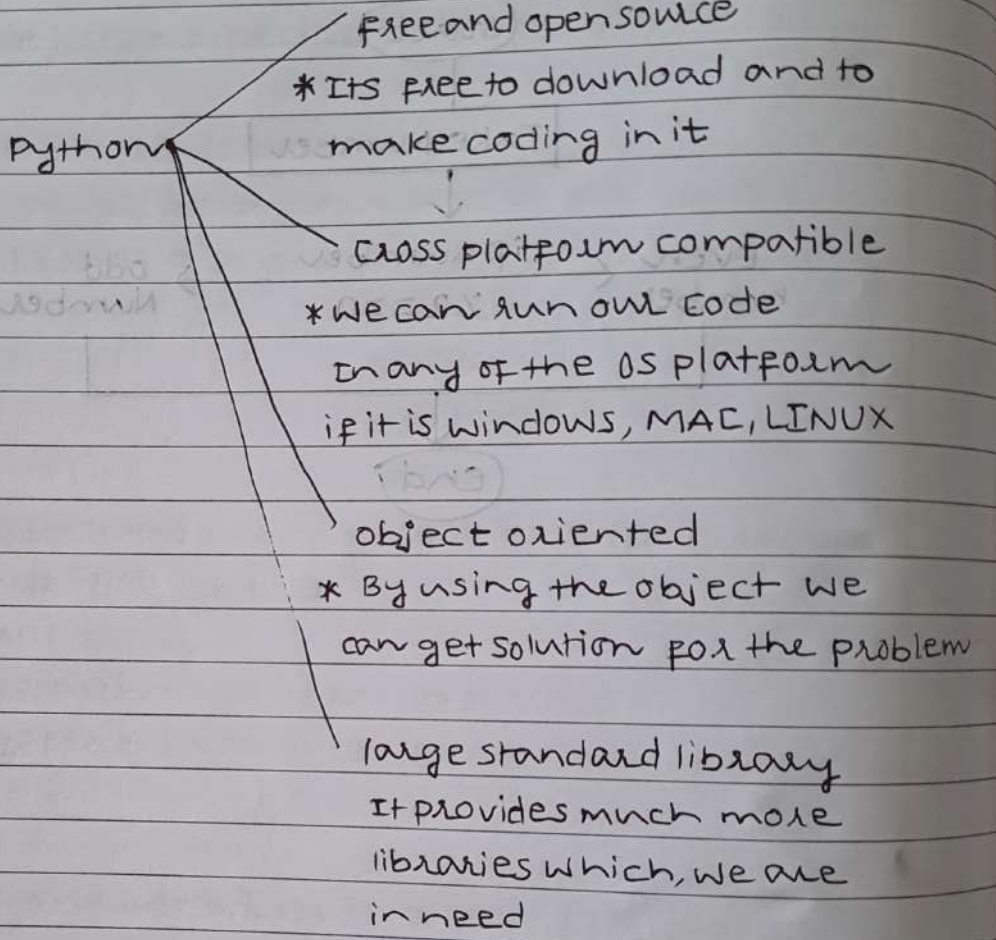
6) Algorithmic approach to solve problem:
step by step approach
to solve a problem is known as Algorithm.

input —— steps to be followed —— output

Algorithm to find if number even/odd.



7 Introduction to python.



IDE - Integrated Development Environment
Python, Pycharm, Anaconda
↓ webpage Distribution

8) Intro to Jupyter notebook.

kernel executes the program.

Print("This is poorna")
This is poorna

9) Variables and Data-types in python.

variables.

a = "Poorna"

a 'a' is a variable

→ Poorna

a = "chandru"

a

→ chandru

a = "Gowda"

a

→ Gowda

Data types in python

every variable is associated with the data

1) Integers (int) Integer values

1, 2, 3, 4, 5, 100, 1000, 999

2) Float →

6.19, 3.14, 9.32, 8.674

3) Boolean →

True, False

4) String (" " mention it inside double quote)

"Poorna" 'chandru' 'Earth'

* a1 = 100
a1
→ 100
type(a1)
→ int

* a1 = 3.14
a1
→ 3.14
type(3.14)
→ Float

a1 = 3.14
type(a1)
→ Float

* a1 = True
a1
→ True
type(a1)
→ Bool

* a1 = "Poorna"
a1
→ 'Poorna'
type(a1)
→ Str

* Complex number
Both Real and Imaginary part
a1 = 6+9j
a1
→ 6+9j
type(a1)
→ Complex

classmate
Date
Page

(10) operators in python:

Arithmetic operator
Relational operator
logical operator
We can perform various types of operations by using above mentioned operators.

Arithmetic operator
a=10
b=20
Arithmetic or mathematic operations
(+, −, *, /)
a+b ⟶ 30 (10+20)
a−b ⟶ −10 (10−20)
b−a ⟶ 10 (20−10)
a*b ⟶ 200 (10*20)
a/b ⟶ 0.5 (10/20) ⟶ we should use forward slash

Relational operators
(>, <, ==, !=)
a=10
b=20
a>b ⟶ False, b>a ⟶ True (20>10)

a<b ⟶ True (10<20)

a==b ⟶ False (10 is not equal to 20)

a!=b ⟶ True

a=100
b=100
a==b → True (100==100, a==b)

Logical operators.
→ logical operator "and", "or" and "not"

→ Bitwise operator "&", "|"

& And operator
a= True (1)
b=False (0)
a&b → False
b&a → False
b&b → False
a&a → True

w.r.t the And operator we get a true value only when both the operands are true

| or operator
It will give use the true result when either of the operands is true or both of the operands are true

we will get a False result w.r.t the or operator only when both the operands are false
a= True (1)
b= False (0)
a|b → True (either one value having true)
b|a → True (a is true)
a|a → True (both the operands are true
b|b → False

(11) Tokens in python

smallest meaningful component in a program

Keywords

Identifiers

Literals

operators

(1) Keywords

→ are special reserved words

It will give meaningful information for the compiler or interpreter.

False	class	Finally	is	Return
None	continue	For	lambda	Try
True	def	From	Nonlocal	While
and	del	Global	Not	With
as	elif	If	or	yield

(2) Identifiers.

are names used for variables, functions or objects

Rules →

No special character except _ (identifiers)

Identifiers are case sensitive

First letter cannot be a digit

Poorna is his Identifier

Poorna is being Identified by his name

(3) Python literals

→ constants in python (Do not change)

a1 = "poorna" ↳ It is a string literal

↳ a1 is a variable

a1 = 123 → Numeric literal

a1 = True → Boolean literal

(12) strings in python

→ are sequence of characters enclosed within single quotes (' '), double quotes (" ") or triple quotes (''' ''')

'Hello World'

" This is poorna"

''' I am going to kerala tomorrow '''

b1 = 'Hello world'

b1 → 'Hello world'

b1 = " This is poorna"

b1 → ' This is poorna'

b1 = ''' This is a multiline
string
'''

b1 → ' this is a multiline\nstring\n'

Extracting the Individual characters.

Indexing in python starts with 0, not 1.

```
                    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
My_string = " My _ name _ is _ poorna
                   -17-16-15-14-13-12-11-10-9-8-7-6-5-4-3-2-1
```

My_string[0] → 'M'

My_string[-1] → 'a'

My_string[4] → 'n'

I want to extract name

So the indexing is done like this

n starts with 3

ends with e → 6

— → 7

The first element would be inclusive (n)

The last element would be exclusive (-)

My_string[3:7] → 'name'

* Now I want poorna

My_string[11:17] → 'poorna'

String Functions:

len(My_string) → 17

My_string.upper() → 'MY NAME IS POORNA'

My_string.lower() → 'my name is poorna'

* Replacing a substring

```
my_string.replace("y","a")
→'Ma name is poorna'
```

* Number of occurrences of substring

```
new_string = "hello hello hello world"
new_string.count("hello")
→ 3
```

* Finding the Index of substring

```
s1 = "This is poorna"
s1.find("poorna")
→ 8
```

* Splitting a string

```
fruit = "I like apples, Mangoes, banana, grapes,
         cherries"
fruit.split(", ")
→['I like apples', 'Mangoes', 'banana', 'grapes', 'cherries']
```

(13) Tuples in python.

Data-structures in python

- Tuple
- List
- Dictionary
- set

Tuple

is an ordered collection of elements enclosed within ()

* Tuples are Immutable
  once you create the tuple, you cant change the value inside the tuple
* Heterogenous mixture of different elements.

#Tuple

tup1 = (100, "b", True, "c", False)

tup1 ⟶ (100, 'b', True, 'c', False)

type(tup1) ⟶ tuple.

Extract Individual elements

tup1[0] ⟶ 100

tup1[1] ⟶ 'b'

tup1[-1] ⟶ False

tup1[1:3] ⟶ 'b', True

tup1[2] = "hello"

we will get error "tuple" object does not support item assignment.

* Finding length of Tuple

```
tup1 = (100, "b", True, "c", False)
len(tup1) → 5
```

* concatenating tuples

```
tup1 = (1,2,3)
tup2 = (4,5,6)
tup1 + tup2
→ (1,2,3,4,5,6)
```

* Repeating Tuple elements

```
tup1 = ("Poorna", 300)
tup1*3
→ ('Poorna', 300, 'Poorna', 300, 'Poorna', 300)
```

* Reapeating and concatenating

```
tup1 = ("Poorna", 300)
tup2 = (4,5,6)
tup1*3 + tup2
→('Poorna',300,'Poorna',300,'Poorna',300,4,5,6)
```

* Tuple Functions.

minimum value

```
tup1 = (1,2,3,4,5)
min(tup1)
→ 1
```

Maximum value

```
tup1 = (1,2,3,4,5)
max(tup1)
→ 5
```

(14) List in python.

list is an ordered collection of elements enclosed within [ ]

* lists are mutable
  we can add, subtract, multiply values inside the list.

```
LI = [1, "Poorna", 3.14, True, 5+9j]
type(LI) ⟶ list
```

* Extracting individual elements

```
LI[-1] ⟶ 5+9j
LI[1:4] ⟶ ['Poorna', 3.14, True]
```

* Modifying a list

  changing the element of 0th index

```
LI = [1, "a", 2, "b", 3, "c"]
LI[0] ⟶ 100
LI ⟶
  ['100', 'a', 2, 'b', 3, 'c']
```

  Appending a new element

```
LI = [1, "a", 2, "b", 3, "c"]
LI.append("poorna")
LI
⟶ [1, 'a', 2, 'b', 3, 'c', 'Poorna']
```

  Popping the last element

```
LI = [1, "a", 2, "b", 3, "c"]
LI.pop()
LI
⟶ [1, 'a', 2, 'b', 3]
```

* Reversing elements of a list

```
L1 = [1, "a", 2, "b", 3, "c"]
L1.reverse()
L1
→ ['c', 3, 'b', 2, 'a', 1]
```

* Inserting element at a specified index

```
L1 = [1, "a", 2, "b", 3, "c"]
L1.insert(1, "poorna")
→ [1, 'poorna', 'a', 2, 'b', 3, 'c']
```

* Sorting a list

```
L1 = ["Mango", "cherry", "banana", "guava"]
L1.sort()
→ ['banana', 'cherry', 'guava', 'Mango']
```

* Concatenating lists

```
L1 = [1, 2, 3]
L2 = ["a", "b", "c"]
L1+L2
→ [1, 2, 3, 'a', 'b', 'c']
```

* Repeating elements

```
L1 = [1, "a", True]
L1*3
→ [1, 'a', True, 1, 'a', True, 1, 'a', True]
```

(15) Dictionary in python:

is an unordered collection of key-value pairs enclosed with {}

* Dictionary is mutable

key ← pair → value

```
fruit = {"apple": 50, "banana": 30, "orange": 40, "peach": 100}
type(fruit) ⟶ dict
```

* Extracting keys

```
fruit = {"apple": 50, "banana": 30, "orange": 40, "guava": 60}
fruit.keys()
⟶
dict_keys(['Apple', 'orange', 'banana', 'guava'])
```

* Extracting values

```
fruit = {"apple": 50, "banana": 30, "orange": 40, "guava": 60}
fruit.values()
⟶
dict_values([50, 30, 40, 60])
```

```
fruit.items()
⟶
dict_items([('apple', 50), ('banana', 30), ('orange', 40), ('guava', 60)])
```

* Adding a new element.

```
fruit={"Apple":10,"orange":20,"Banana":30}
fruit["Mango"]=50
fruit ⟶
{'Apple':10,'orange':20,'Banana':30,'Mango':50}
```

* Changing an existing element

```
fruit={"Apple":10,"orange":20}
fruit["Apple"]=100
fruit ⟶
{'Apple':100,'orange':20}
```

* Update one dictonary's element with another

```
fruit1={"Apple":10,"orange":20}
fruit2={"cherry":30,"Banana":40}

fruit.update(fruit2)
fruit1 ⟶
{'Apple':10,'orange':20,'cherry':30,'Banana':40}
```

* Popping an element (Removing)

```
fruit={"Apple":10,"orange":20,"Banana":30}
fruit.pop("orange")
fruit ⟶
{"Apple':10,'Banana':30}
```

Set in python

Set is an unordered and unindexed collection of elements elclosed with { }

* Duplicates are not allowed in set

```
s1 = {1, "Poorna", "Poorna", 1}
s1 ⟶
{1, "Poorna"}
```

We can see, duplicates are not allowed in set

Set operations.

* Adding a new element

```
s1 = {1, "a", True, 2, "b", False}
s1.add("Hello")
s1 ⟶
{1, 2, False, 'Hello', 'a', 'b'}
```

⟶ There is no proper sequence in set so the indexing is not possible in sets.

* updating multiple elements

```
s1 = {1, "a", True, 2, "b", False}
s1.update([10, 20, 30])
s1 ⟶
{1, 10, 2, 20, 30, False, 'a', 'b'}
```

* Removing an element

```
s1 = {1, "a", True, 2, "b", False}
s1.remove("b")
s1
{1, 2, False, 'a'}
```

set functions.

* union and Intersection

* union

s1 = {1, 2, 3}

s2 = {"a", "b", "c"}

s1.union(s2) ⟶

{1, 2, 3, 'a', 'b', 'c'}

Basically concatenating two sets.

* Intersection

s1 = {1, 2, 3, 4, 5, 6}

s2 = {5, 6, 7, 8, 9}

s1.intersection(s2) ⟶

{5, 6}

To find out the common elements between two sets.

If statement in python

* If statement,

  if
  it's raining
  sit inside

  else
  go out and play

①
```
a=10
b=20
if a>b:
    print("a is greater than b")
if b>a:
    print("b is greater than a")

if a>b:
    print("a is greater than b")
else:
    print("a is not greater than b")
```

Rwn →

```
a is not greater than b
```

elseif → we will use it when
we want to check the multiple statements.

classmate
Date
Page

(2)
```
a=10
b=20
c=30
if(a>b and & a>c):
    print("a is the greatest")
elif (b>a & b>c):
    print("b is the greatest")
else:
    print(" c is greatest")
```
Run → c is the greatest

* Now can see how can we use this Conditional statements with List, Tuple and dictionary.

→ tupl=(1,2,3,4) (with tuple)
```
if 2 in tupl:
    print(" 2 is present in tuple")
```
Run → 2 is present in tuple

→ tupl=(1,2,3,4)
```
if 6 in tupl:
    print(" 6 is present in tuple")
else:
    print("6 is not present in tuple")
```
Run → 6 is not present in tuple

with list if with list

*
```
li = [1,2,3,4,5]
if li[1]==2:
    li[1]=li[1]+100
li ⟶ [1,102,3,4,5]
```

*
```
li=[1,2,3,4,5]
if li[4]==10:
    li[1]=li[1]+100
else:
    li[4]=li[4]+500
li ⟶ [1,102,3,4,505]
```

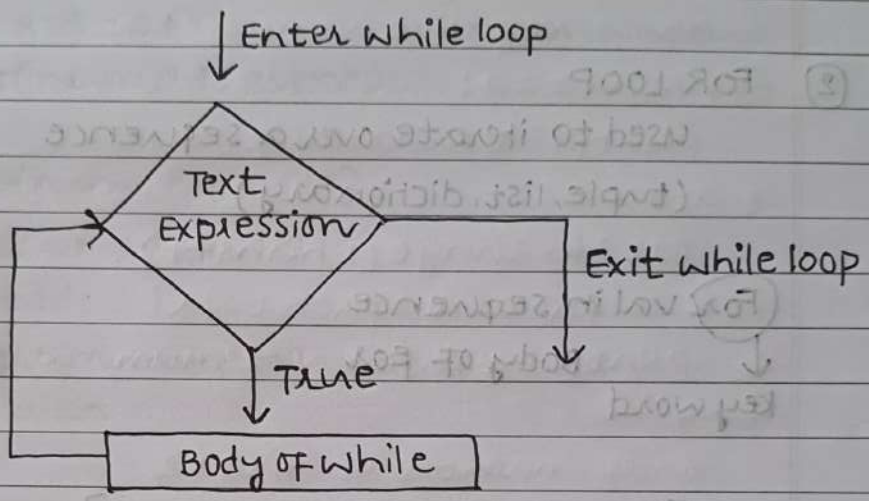
if with dictionary

*
```
d1 = {"a":1,"b":2,"c":3}
if d1["b"]==2:
    d1["b"]=d1["b"]+100
d1 ⟶ {'a':1,'b':102,'c':3}
```

LOOPING STATEMENTS IN PYTHON:

used to repeat a task multiple times.

① While loop =



Syntax

```
while condition :
        Execute statements
```

*
```
i=1
while i<=10:
    print(i)
    i=i+1
```
→ 1 2 3 4 5 6 7 8 9 10

*
```
i=1
n=2
while i<=10:
    print(n,"*",i,"=",n*i)
    i=i+1
```
→
2*1 = 2
2*2 = 4
2*3 = 6
2*4 = 8
2*5 = 10
2*6 = 12
2*7 = 14
2*8 = 16
2*9 = 18
2*20 = 40
2*10 = 20

* `li = [1,2,3,4,5]` `i=0`

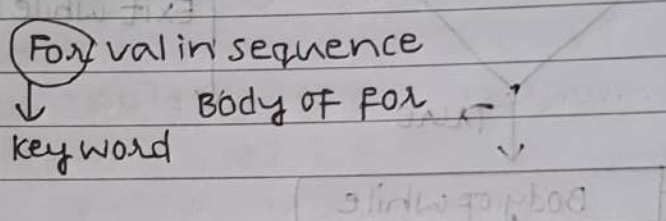
```
while i < len(li):
    li[i] = li[i] + 100
    i = i+1
```

li ⟶ [101, 102, 103, 104, 105]

② FOR LOOP

used to iterate over a sequence

(tuple, list, dictionary)



keyword

*
```
li = ["apple", "Banana", "orange"]
for i in li:
    print(i)
```

⟶ apple
banana
orange

* nested for loop

```
li = ["orange", "blue", "green"]
l2 = ["book", "chair", "phone"]
for i in li:
    for j in l2:
        print(i, j)
```

⟶ orange book
orange chair
orange phone
blue book
blue chair
blue phone
green book
green chair
green phone

Basic problems in python

① check even or odd:

```
num = int(input("Enter a number: "))
if(num%2)==0:          %→Remainder
   print(num," is even")
else:
   print(num," is odd")
```

→ Enter a number: 5
  5 is odd

→ Enter a number: 8
  8 is even

② check positive, negative or 0

```
num = float(input("Enter a number: "))
if num>0:
   print("positive number")
elif num==0:
   print("zero")
else:
   print("Negative number")
```

→ Enter a number: 8
  Positive number

→ 0
  zero

→ Enter a number: -5
  Negative number

③ Factorial of a number.

```
num = int(input("Enter a number: "))
factorial = 1
if num<0:
    print("Sorry, Factorial doesn't exist for
        Negative number")
elif num==0:
    print("The Factorial of 0 is 1")
else:
    for i in range(1,num+1):
        factorial = factorial*i
    print("The factorial of", num, "is", factorial)
```

⟶

Enter a number 4 4X3X2

24

⟶

Enter a number

3628800

(4) Reversing a number

```
n=int(input("Enter number: "))
rev=0
while(n>0):
    dig=n%10
    rev=rev*10+dig
    n=n//10
print("Reverse of the number:", rev)
```

n = 123 → 321

(5) check if it is a palindrome (12321)

```
n=int(input("Enter a number: "))
temp=n
rev=0
while(n>0):
    dig=n%10
    rev=rev*10+dig
    n=n//10
if(temp==rev):
  print("The number is a palindrome!")
else:
  print("The number is not a palindrome!")
```

121/10 → rem(1)	n	rev
12/10 → 2	121	0*10+1 = 0+1 = 01
	12	1*10+2 = 10+2 = 12
	1	12*10+1 = 120+1 = 121

dig
1
2
1

Enter a number: 121
The number is a palindrome

⑥ Fibonacci 0 1 1 2 3 5 8

```
n=int(input("Enter number: "))
a=0
b=1
if n<0:
    print("Incorrect input")
elif n==0:
    print(a)
elif n==1:
    print(a)
else:
  for i in range(2,n):
      c=a+b
      a=b
      b=c
    print(b)
```

⟶ Enter a number: 7
8

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

classmate
Date
Page

Functions in python.

Block of code which performs a specific task.

Normal function
Lambda function

Normal function syntax

```
def function_name:
        Execute statements
```

Lambda function syntax

```
Lambda arguements : expression
```

*
```
def hello():
    print("Hello World")
hello()
```
Hello World

*
```
def add10(x):
    return x+10
add(10(10)
```
(def — keyword; x — parameter)

20

```
def even_odd(x):
    if x%2==0:
        print(x,"is even")
    else:
        print(x,"is odd")
even_odd(5)
```

5 is odd

Lambda functions. (filter, map, reduce)

```
g = lambda x: x*x*x
print(g(7))
```

343

Main function of lambda operations

lambda functions with (filter) takes two parameter

```
li=[5,7,22,97,54,62,77,23,73,61]
final_list=list(filter(lambda x:(x%2!=0),li))
print(final_list)
```

1st 2nd

extracting the odd numbers.

→

[5, 7, 97, 77, 23, 73, 61]

#lambda with map

```
li=[5,7,22,97,54,62,77,23,73,61]
final_list = list(map(lambda x:x*2,li))
print(final_list)
```

⟶

```
[10,14,44,194,108,124,154,46,146,122]
```

lambda with reduce

```
from functools import reduce
li=[5,8,10,20,50,100]
sum = reduce((lambda x,y:x+y),li)
print(sum)
```

⟶ 193

Object oriented programming

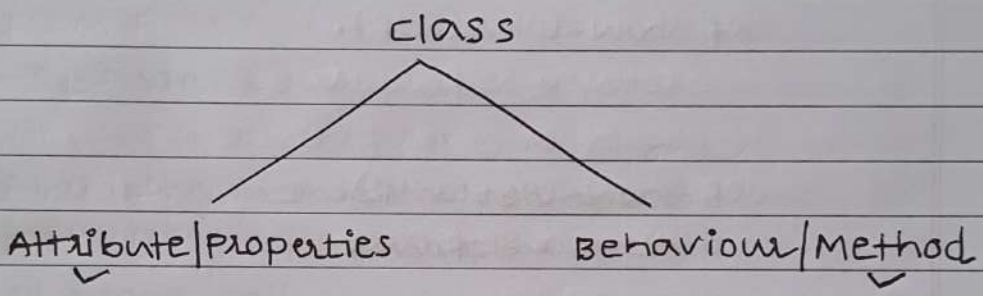
classmate
Date
Page

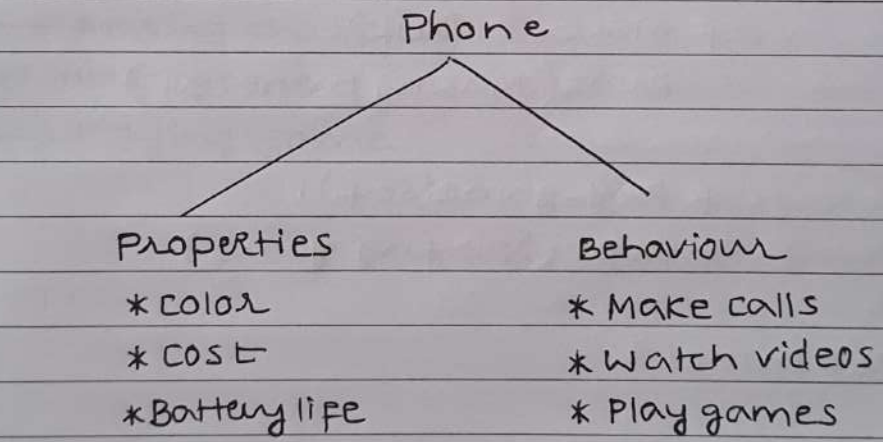
OOPs in python:

we are surrounded with objects
like (mobile, laptop, bike, key, pen, dog, book)

classes:-

class is a template for real world entities

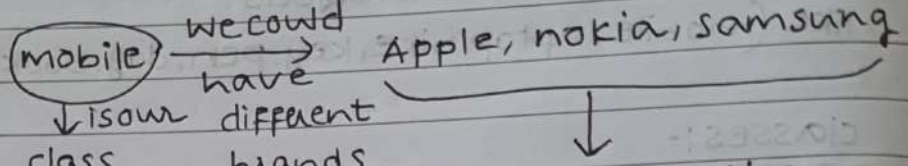




class is a user defined datatype
↳ It is a datatype which we can create by ourselves

objects :-

objects are specific instances of a class.



Creating the first class.

```
#creating a class phone

class phone :
    def make_call(self):
        print("Making phone call")
    def play_game(self):
        print("playing game")

p1 = phone()

#invoking methods through object
p1.make_call
Making phone call
p1.play_game
Playing game
```

methods

Referencing call

keyword

name of object

name of the method

Adding parameters to a class method

```
class Phone:
    def set_color(self, color):
        self.color = color

    def set_cost(self, cost):
        self.cost = cost

    def show_color(self):
        return self.color

    def show_cost(self):
        return self.cost

    def make_call(self):
        print("Making phone call")

    def play_game(self):
        print("playing game")

P2 = Phone()

P2.assign
P2.set_color('blue')
P2.set_cost(500)
P2.show_color() → 'blue'
P2.show_cost() → 500
```

creating a class with constructor.

```
class Employee:
    def __init__(self, name, age, salary, gender):
        self.name = name
        self.age = age
        self.salary = salary
        self.gender = gender

    def employee_details(self):
        print("Name of employee is ", self.name)
        print("Age of employee is ", self.age)
        print("salary of employee is ", self.salary)
        print("Gender of employee is", self.gender)

e1 = Employee("Poorna", 23, 1,00,000, "Male")
e1.employee_details()
```

↳ init method acts as constructor
↓
spcl type of function

↓ instantiating the "e1" object

e1.employee_details() → Invoking 'employee_details' method.

```
Name of employee is poorna
age of employee is 23
salary of employee is 1,00,000
gender of employee is Male
```

Inheritance in python

With Inheritance one class can derive the properties of the another class.

```
class vehicle:
    def __init__(self,mileage,cost):
        self.mileage=mileage
        self.cost = cost

    def show_details(self):
        print("I am a vehicle")
        print("Mileage of vehicle is", self.mileage)
        print("cost of vehicle is", self.cost)

V1=vehicle(500,500)
V1.show_details()
```

⟶

```
I am a vehicle
Mileage of vehicle is 500
cost of vehicle is 500
```

creating the child class

```
class car(vehicle):
    def show_car(self):
        print("I am a car")

c1 = car(200,1200)
c1.show_details()
I am a vehicle
Mileage of vehicle is 200
cost of vehicle is 1200

c1.show_car()
I am a car
```

→ Instantiating the object for child class

→ Invoking the child class method

over-riding init method.

```
class car(vehicle):
    def __init__(self, mileage, cost, tyres, hp):
        super().__init__(mileage, cost)
        self.tyres = tyres
        self.hp = hp

    def show_car_details(self):
        print("I am a car")
        print("Number of tyres are", self.tyres)
        print("value of hp is", self.hp)

#invoking show_details()
 method from parent class

c1 = car(20, 12000, 4, 300)
c1.show_details()
```

⟶

```
I am a vehicle
Mileage of vehicle is 20
cost of vehicle is 12000
```

```
#invoking show_car_details()
 method from child class

c1.show_car_details()
```

⟶

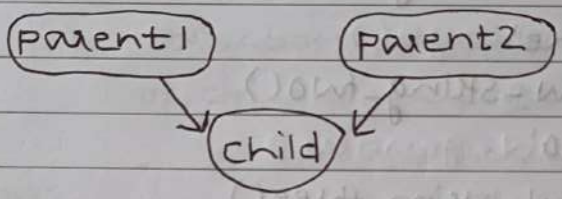
```
I am a car
Number of tyres are 4
value of hp is 300
```

Types of Inheritance

Single Inheritance
Multiple Inheritance
Multi-level Inheritance
hybrid Inheritance

Multiple Inheritance

The child inherits from More than 1 parent class.




```
class parent1():
    def assign_string_one(self,str1):
        self.str1 = str1

    def show_string_one(self):
        return self.str1
```
(Parent class 1)

```
class parent2():
    def assign_string_two(self,str2):
        self.str2 = str2
    def show_string_two(self):
        return self.str2
```
(Parent class 2)

```
class derived(Parent1, parent2):
    def assign_string_three(self,str3):
        self.str3 = str3
    def show_string_three(self):
        return self.str3
```

Creating the object of the child class

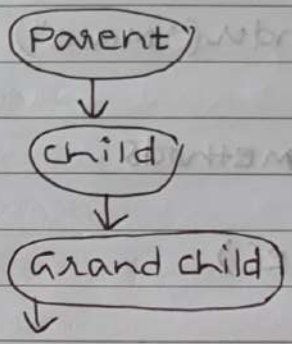
```
d1. Derived
d1.assign-string-one("one")
d1. assign-string-two("two")
d1. assign-string-three("three")
```

Invoking methods

```
d1.show_string-one()
→ 'one'
d1.show-string-two()
→ 'two'
d1.show_string-three()
→ 'three'
```

Multi-level Inheritance

We have parent, child and grandchild Relationship



It have both the properties of child and the parent.

Parent class

```
class parent():
    def assign_name(self,name):
        self.name = name
    def show_name(self):
        return self.name
```

child class:

```
class child(parent):
    def assign_age(self,age):
        self.age = age
    def show_age(self):
        return self.age
```

Grand child class

```
class grandchild(child):
    def assign_gender(self,gender):
        self.gender = gender
    def show_gender(self):
        return self.name
```

gl.ch grandchild()

gl.assign_name("Poorna")
gl.assign-age(25)
gl.assign-gendu("Male")

Invoking class methods

gl.show-name()
⟶ Poorna
gl.show-age()
⟶ 25
gl.show-gendu()
⟶ Male

Continuation of python

(26) **Array in python**

All the value of same type.

if we have int array, we should only have int array

if we have float array, we should only have float array.

```
import array as arr
arr.array
                (or)        i - signed integer (-1 to +1)
from array import *         I - unsigned integer
vals = array('i', [5,9,8,42])
print(vals)
```

* `print(vals.buffer_info())`
  → (Address, size)
* `print(typecode)`
  → i → we are working with integer
*
```
from array import *
vals = array('i', [5, 9, -8, 4, 2])
vals.reverse()
print(vals)
```
→ array('i', [2, 4, -8, 9, 5])

```
vals = array('i', [5,9,-8,4,2])
for i in range(5):
    print(i)
```
→ 5
9
-8
4
2

* When we dont know the range

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

```
for e in vals:
    print(e)
```

* Can we work with characters
    U - unicode

*
```
from array import *
vals = array('u',['a','e','i'])
for e in vals:
    print(e)
```
→ a e i

* If I want to create a new array using the old array

```
from array import *
vals = ('i',[5,9,8,4,2])
newarr = array(vals.typecode,(a for a in vals))
for e in newarr:
 print e
```

---

using while loop

```
While i<len(newarr):
    print(newarr[i])
    i=i+1
```

(27) inserting element in array
searching for element in array
in python

we can create a blank array

```
from array import *
arr=array('i',[])
n=int(input("Enter the length of the array"))
for i in range(5): range(n):
    x=int(input("Enter the next value"))
    arr.append(x)
print(arr)
val=int(input("Enter the value for search"))
k=0
for e in arr:
    if e==val:
        print(k)
        break
    k=k+1
print(arr.index(val))
```

single dimensional
one row and multiple columns

multi dimensional array
multiple rows and multiple columns

Exception Handling

What is

compile time error → a = hello" (Problem in syntax)

Runtime error → L=[1,2,3]print(L[4])

logical error → 4+4|2 (Problem with logic)

⑥

Exceptions:- Runtime errors as exception

There is an error occuring during the execution of the particular task

Exception handling.

keywords -

try:

except:

else: } optional

finally: }

try:-

Contains operations

except:-

what has to be done

write that code here.

Types of error

1) n,x = 5,6,"Poorna"

value error = too many values to unpack

2) L=[1,2,3]

L[4]

index error = list index out of range

python.org

classmate
Date
Page

(3) b=10
a

Name error: name a is not defined.

(4) import poorna

modwle not found error = no modwle named 'poorna'

(5) 2+'3'

Typeerror: unsupported operand type for +
'int' and 'str'.

Program

(*) import math
#enter negetive number to check what happens

num=int(input ("Enter number to comput of
factorial: "))
try:
print(math.factorial(num))
except valueerror:
print ("cannot compute the factorial of
negetive numbers")

⊛ Program

```
import math

num = int(input("Enter no to compute factorial of : "))
valid_input = false

while not valid_input:
try:
        print(math.factorial(num))
        valid_input = True
except valueerror
        print("cannot compute the factorial
                of negative nos")
        num = int(input("please re-enter: "))
```

⊛ Program

```
def getmonth():
    month = int(input("Enter current month (1-12): "));
    if month<1 or month>12:
            raise value error
    return month

valid = False
monthname = ("Jan", "Feb", "march", "Ap", "may", "June",
            "July", "Aug", "sept", "Nov", "Dec")
while not valid
    try:
        month = getmonth()
        print("The month you entered is",
            month_name[month-1])
        valid1 = True
    except valueError:
        print("Invalid month enter\n")
```

(*) program

```
def getMonth():
    Month = int(input("Enter current month (1-12): "))
    if month<1 or month>12:
      raise valueError("Invalid Month value")
    return month

valid=false
month_name=("Jan",
            "Dec")
While not valid:
try:
    month =getMonth()
    print("The month you entered is",
          month_name[month-1])
    valid = True
except valueError as ve_msg:
    print(ve_msg)
          ↳ This an object of value error
```

(*) Program

→ creating my own exception class

```
class UserDefinedException(Exception):
    def __init__(self, message):
        self.message = message
try:
    s=input("Enter name: ")
    if(s==""):
        raise UserDefinedException("No empty string Allowed")
    else:
        print(s)
except UserDefinedException as msg:
    print("Error Message", msg)
```

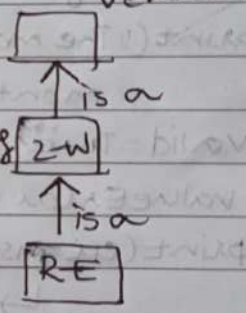
① Guess the output

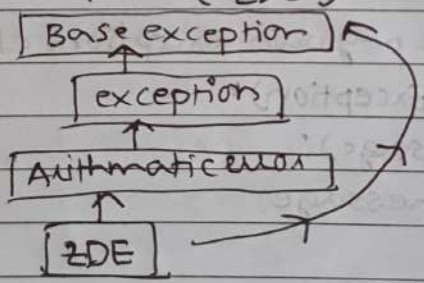
```
try:
    a = 5
    b = 5/0
except ZeroDivisionError:
    print("ZDE")
except BaseException:
    print("BE")
except Exception:
    print("E")
```

```
try:
    a = 5
    b = 5/0 → raise ZDE
except BaseException:
    print("BE")
except Exception:
    print("E")
except ZeroDivisionError:
    print("ZDE")
```

ZDE is a Base exception variable

If tries to matching





ZDE is a Baseexception error

② The else part will executed when no exception has been caught

4)
```
try:
    a=5
    b=5|5
except ZDE:
    print("ZDE")
except BaseException:
    print("BE")
except Exception:
    print("E")
else:
    print("ELSE")
```

5)
```
try:
    a=5
    b=5|0
except(BaseException,Exception,ZeroDivisionError):
```

6)
```
try:
    a=5
    b=5|5
except:
    print("Hi")
else:
    print("Hello")
```

7)
```
try:
    a=5
    b=5|5
else                    → It will give us compile time error
    print("Hello")
```

⑧

```
try:
    a=5
    b=5/0
except:
    print("Hi")
else:
    print("Hello")
finally:
    print("Namasthe")
```

```
try:
    cook-biriyani()
except salt less error
except overcooked error
finally:
    switchoff-stove
```

whether there is exception is executed or not finally will be executed.

—| end |—

of exception handling

(29) Ways of creating Arrays in Numpy
array(), linespace(), logspace(),
arange(), zeros(), ones()

```
from numpy import *
arr = array([1,2,3,4,5])
print(arr)
```
in array all the values should be of same type

* linspace() (start, stop, step)
arr = linspace(0,15,16) → dividing into the parts

* arange()
```
arr = arange(1,15,2)
print(arr)
```
→ 1, 3, 5, 7, 9, 11, 13

* logspace()
```
arr = logspace(1,40,5)
print(arr)
```

* zeros() ones() → more efficient

```
arr = ones(5)
print(arr)
```
→ [1. 1. 1. 1. 1.]

(30) Copying an Array in python

* From numpy import *
```
arr = array([1,2,3,4,5])
arr = arr+5
print(arr)
→ [6 7 8 9 10]
```

* vectorized operation
```
from numpy import *
arr1 = array([1,2,3,4,5])
arr2 = array([5,4,3,2,1])
arr3 = arr1+arr2
print(arr3)
→[6,6,6,6,6]
```

* Mathematical operations in numpy
```
from numpy import *
arr1 =array([1,2,3,4,5])
print(arr1)
print(log(arr1))
print(sin(arr1))
print(cos(arr1))
print(sqrt(arr1))
print(sum(arr1))
print(min(arr1))
print(max(arr1))
unique, sort
```

* we can also concatinate
```
from numpy import *
arr1 =([1,2,3,4,5])
arr2 =array([6,1,9,3,2])
print( concatenate([arr1,arr2]))
```

classmate
Date
Page

Copying

```
from numpy import *
arr1 = array([1,2,3,4,5])
arr2 = arr1.view()
print(arr1)
print(arr2)
```

view is a function that creats new array

shallow copy:

The both the array are interlinked

```
from numpy import *
arr1 = array([2,6,8,1,3])
arr2 = arr1.view()
arr1[1] = 7
print(arr1) → 2,7,8,1,3
print(arr2) → 2,7,8,1,3
```

Deep copy

The both the arrays are not interlinked

```
from numpy import *
arr = array([1,2,3,4,5])
arr2 = arr1.copy()
arr1[1] = 7
print(arr1) → 1,7,3,4,5
print(arr2) → 1,2,3,4,5
```

(31) Working with matrix

```
from numpy import *
arr = array([
        [1, 2, 3],
        [4, 5, 6]
        ])
print(arr)
print(arr.ndim) → show the no of dimension
print(arr.shape) → 2 row, 3 columns
print(arr.size) → 6 (counts the size)
```

I want to create 2d array into 1D array

* From numpy import *

```
arr = array([
        [1, 2, 3],
        [4, 5, 6]
        ])
arr2 = arr.flatten()
print(arr2)
[1, 2, 3, 4, 5, 6]
```

arr2 is 1D

* To create 3D array from 1D array

```
from numpy import *
arr = array([
        [1, 2, 3, 6, 2, 9],
        [4, 5, 6, 7, 5, 3],
        ])
arr2 = arr.flatten()
arr3 = arr2.reshape(3, 4)
print(arr3)
```

Matrix operations.

```
from numpy import *
m=matrix("1,2,3; 6,4,5; 1 6 7")
print(diagonal(m)) → [1 4 7]
print(m.min()) → 1
Print(m.max()) → 7
```

```
from numpy import *
m1=matrix('1 2 3; 6 4 5; 1 6 7')
m2= matrix('1 2 3; 6 8 5; 2 6 7')
m3 = m1 * m2 ;
Print(m3)
```

(32) Functions in python:

I want to define a function, later on I want to call that function

func name

```
* def greet():
      print("Hello")
      print("Good Morning")
  greet()
```

sweet

parameter or argument

```
* def add(x,y):
      c=x+y
      print(c)
  add(5,4) ⟶ 9
```

function can return a value

```
* def add(x,y):
      c=x+y
      return c
  result = add(5,4)
  print(result)
```

* from one function we can return multiple values

```
def add_sub(x,y):
    c=x+y
    d=x-y
    return c,d
result1, result2 = add_sub(5,4)
print(result1, result2)
```

(33) Function arguements in python

How to pass a parameter to function

```
def update(x):
    x=8
    print(x)
update(10)
```

(34) Types of Arguements.

```
def add(a,b):
    c=a+b
    print(c)
add(5,6) ⟶ 11
```

a,b → Formal Arguement

Arguement (Actual Arguement)

Position

```
def person(name, age):
    print(name)
    print(age)
person("Poorna", 23)
```

Keyword

```
person(age=28, name="Poorna")
```

default

```
* def person(name,age=18):
    print(name)
    print(age)
person("Poorna") ⟶ Poorna.
```

variable length arguement

```
def sum(a,b):
    c=a+b
    print(c)
sum(5,6)
```

```
def sum(a, *b)
    c=a
    for i in b:
        c=c+i
    print(c)
sum(5,6,34,78) ⟶ 123
```

(35) **kwargs

↑function

```
def person():
def person(name, **data):
    print(name)
    print(data)
person("navin", age=28, City="mumbai", mDb=00)
```

(36) Global keywords.

scope

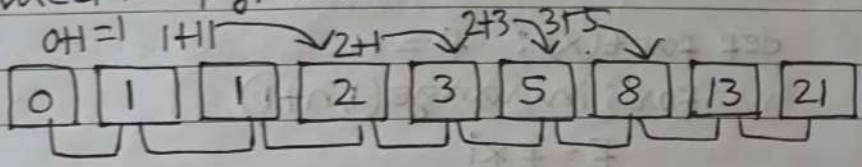
(a=10) → This a this is outside the fun is global

```
def something():
    (a=15) → Inside the fun is local variable
    print(a) ↳ we cannot use this out side

print(a)
```

(37) Pass list to a function

(38) Fibonacci in python.



We should add 2 numbers to get next number

```
def fib(n):
    a=0
    b=1
    print(a)
    print(b)
    for i in range(2,n):
        c=a+b
        a=b
        b=c
        print(c)
fib(5)
```

(39) Factorial of a number.

$5! = 5*4*3*2*1 = 120$

$4! = 24$

```
def fact(x):  f=1
    for i in range(1,n+1)
        f = f*i
    return f
x=4
result = fact(x)
print(result)
```

(40) Recursion in python

```
def greet():
    print("Hello")

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

A function calling itself is Recursion.

(41) factorial using recursion.

```
def fact(n):
    if n==0
        return 1
    return n* fact(n-1)
result = fact(5)
print(result) ⟶ 120
```

(42) lambda ⟶ Anonymus function
function without names are called as Anonymus function (or) lambda.
"functions are objects" in python

```
f= lambda a ; a*a
result = f(5)
print(result) ⟶ 25
```

(43) lambda,

(43) lambda
Filter
map
reduce.

```
def is_even(n):
    return n%2==0
nums=[3,2,6,8,4,6,2,9]

evens = filter(
evens = list(filter(is_even,nums))
print(evens)
```

```
nums = [3,2,6,8,4,6,2,9]
evens = list(filter(lambda n: n%2==0,nums))
print(evens)
doubles=list(map(lambda n:n*2,evens))
```

filter=

reduce

```
from functools import reduce

nums=[3,2,6,8,4,6,2,9]
evens=list(filter(lambda n:n%2==0,nums))
doubles=list(map(lambda n:n*2,evens))
print(doubles)
sum=reduce(lambda a,b: a+b,doubles)
print(sum)
```

(44) Decorators:

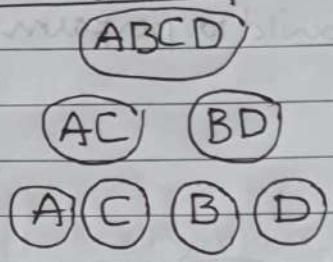
functions are build to perform certain task

```
def div(a,b):
    print(a/b)
def smart_div(func):          # It is taking it as a parameter
    def inner(a,b):
        if a<b:
            a,b = b,a
        return func(a,b)
    return inner

div1 = smart_div(div)
div1(2,4)
```

We can change the behaviour of existing function

45 Modules in python.



1 module will be file

```
a=9
b=7
```

new file

```
def add(a,b):
    return(a+b
def sub(a,b):
    return a-b
def multi(a,b):
    return a*b
def div(a,b):
    return a/b
```

```
import calc
a=9
b=7
c=calc.add(a,b)
```

```
from calc import *
a=9
b=7
c=sub(a,b)
```

→ 2

(46) `__name__ = "__main__"`
special variable `__name__`

```
print(__name__)
```
→ `__main__` → starting point of program

(47) continued part

demo
```
from calc import add
def fun1():
    add()
    print("from fun1")
def fun2():
    print("from fun2")
fun1()
fun2()
def main()
    fun1()
    fun2()
main()
```
→ in calc main
result 1 is __main__
result 2 is

calc
```
def add():
    print("result 1 is", __name__)
def sub():
    print("result 2 is")
def main():
    print("in calc main")
    add()
    sub()
if __name__ == "__main__":
    main()
```

(6) Iterators in python

one value at a time

```
nums = [7, 8, 9, 5]
nums[0] ⟶ 7
nums[3] ⟶ 5
for i in nums:
    print(i) ⟶ 7
                8
                9
                5
```

```
nums = [7, 8, 9, 5]
it = iter(nums)
print(it.__next__()) ⟶ 7
print(next(it)) ⟶ 8
```

lets create own class.

```
class topten:
    def __init__(self):
        self.num = 1

    def __iter__(self):
        return self

    def __next__(self):
        if self.num <= 10:
            val = self.num
            self.num += 1
            return val
        else:
            raise StopIteration

values = Topten()
for i in values:
    print(i)
```

(62) Generators.

top10 perfect square

```
def topten():
    n=1
    while n<=10:
        sq = n*n
        yield sq
        n+=1
values = topten()
for i in values:
    print(i)
```