

"PYTHON"

classmate

Date _____

Page _____

I want to speak with the computer

Language = Python

- Guido van Rossum

Syntax:-

- 1991.

import pandas as pd.

Programming language application :-

Gaming, Banking, Machine learning

Basic variables.

Data → collection of the facts.

Ex: My name is pooja.

How to store the data

"Pooja" → string

56 → Integer

True/False → Boolean

variables.

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

student = "Pooja"

Diagram:
A line from 'student' in the code above branches into two lines pointing to 'Name associated' and 'address associated'.

Name associated & address associated

student → Name of the variable.

$$\begin{aligned} 10 &= a \\ 20 &= b \end{aligned} \left\{ \begin{aligned} a+b &= 30 \\ a-b &= -10 \\ a*b &= 200 \\ a/b &= - \end{aligned} \right. \begin{array}{l} \text{multiple operations} \\ \text{on variables} \end{array}$$

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.

Phone

Properties

* color

* cost

* Behaviour

Behaviour

* Make calls

* we can text

* Watch video

* Play games

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

of blw objects

All having

Different values for properties

Different values for behaviour

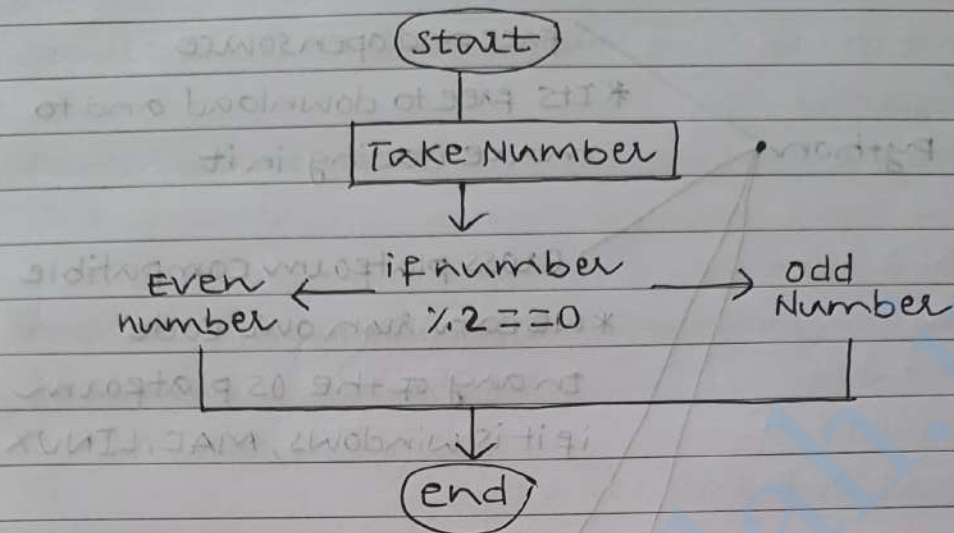
6) Algorithmic approach to solve problem

step by step approach

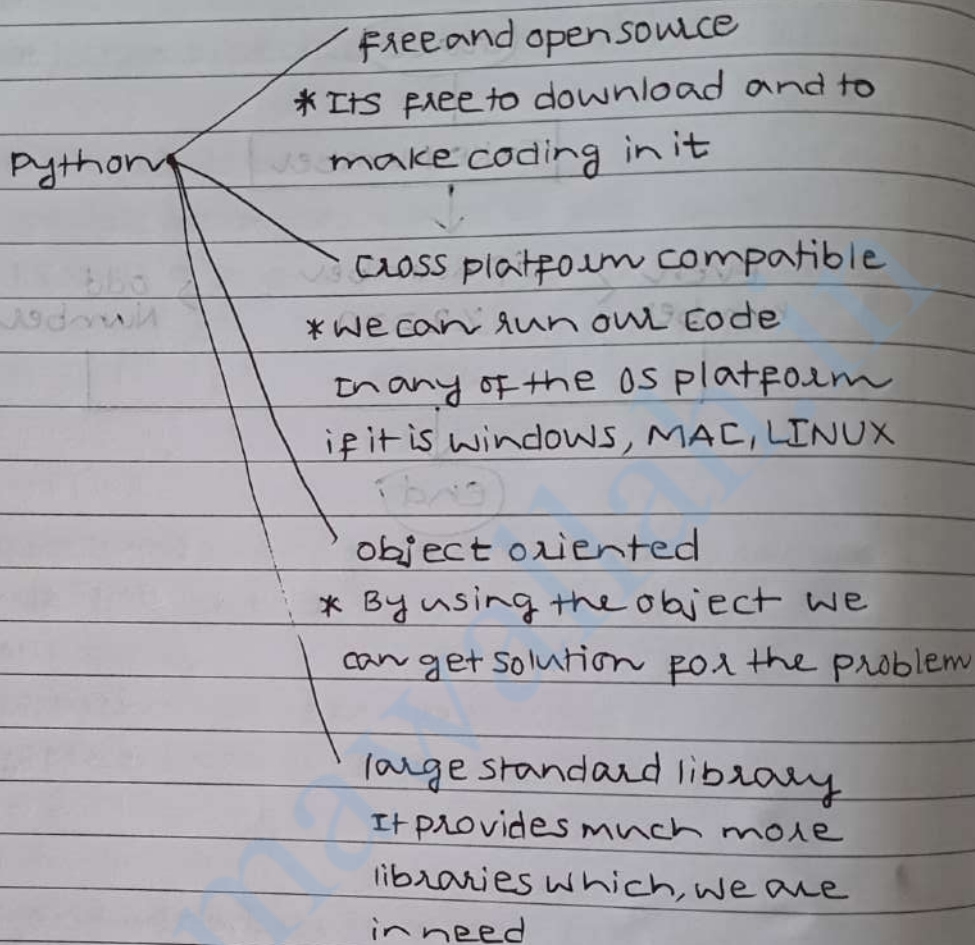
to solve a problem is known as Algorithm.

input — steps to be — output
followed

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 poozna")
```

This is poozna

9) Variables and Data-types in python.

variables.

```
a = "Poozna"
```

a

'a' is a variable

→ Poozna

```
a = "chandan"
```

a

→ chandan

```
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)

"Poozna" "chandan" "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 = "Poolna"`

`a1`

→ 'Poolna'

`type(a1)`

→ str

* Complex number

Both Real and Imaginary part

`a1 = 6+9j`

`a1`

→ 6+9j

`type(a1)`

→ complex

(10) operators in python:

Arithmetic operator

Relational operator

logical operator "and", "or", "not"

We can perform various types of operations
by using above mentioned operators.

Arithmetic operator

 $a = 10$ $b = 20$

Arithmetic or mathematic operations

 $(+, -, *, /)$ $a + b \rightarrow 30 (10 + 20)$ $a - b \rightarrow -10 (10 - 20)$ $b - a \rightarrow 10 (20 - 10)$ $a * b \rightarrow 200 (10 * 20)$ $a / b \rightarrow 0.5 (10 / 20) \rightarrow$ we should use forward slash

Relational operators

 $(>, <, ==, !=)$ $a = 10$ $b = 20$ $a > b \rightarrow \text{False}, b > a \rightarrow \text{True} (20 > 10)$ $a < b \rightarrow \text{True} (10 < 20)$ $a == b \rightarrow \text{False} (10 \text{ is not equal to } 20)$ $a != b \rightarrow \text{True}$

$a = 100$

$b = 100$

$a == b \rightarrow \text{True} (100 == 100, a == b)$

Logical operators.

$\rightarrow$ logical operator "and", "or" and "not"

$\rightarrow$ Bitwise operator "&", "|"

& And operator

$a = \text{True} (1)$

$b = \text{False} (0)$

$a \& b \rightarrow \text{False}$ w.r.t the And operator

$b \& a \rightarrow \text{False}$ we get a true value only

$b \& b \rightarrow \text{False}$ when both the operands are

$a \& a \rightarrow \text{True}$ 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

$a = \text{True} (1)$

$b = \text{False} (0)$

$a | b \rightarrow \text{True}$ (either one value having true)

$b | a \rightarrow \text{True}$ (a is true)

$a | a \rightarrow \text{True}$ (both the operands are true)

$b | b \rightarrow \text{False}$

- (ii) 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 (7)

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 can't change the value inside the tuple
- * Heterogeneous 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 = ("Poojha", 300)

tup1 * 3

→ ('Poojha', 300, 'Poojha', 300, 'Poojha', 300)

* Repeating and concatenating

tup1 = ("Poojha", 300)

tup2 = (4, 5, 6)

tup1 * 3 + tup2

→ ('Poojha', 300, 'Poojha', 300, 'Poojha', 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, "Poojna", 3.14, True, 5+9j]
```

`type(LI) → list`

* Extracting individual elements

`LI[-1] → 5+9j`

`LI[1:4] → ['Poojna', 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("poojna")`

`→ [1, 'a', 2, 'b', 3, 'c', 'poojna']`

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 dictionary's element with another

```
fruit1 = {"Apple": 10, "orange": 20}
```

```
fruit2 = {"cherry": 30, "Banana": 40}
```

```
fruit1.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 enclosed with $\{ \}$

* Duplicates are not allowed in set

$S1 = \{1, "Poorna", "Poorna", 1\}$

$S1 \rightarrow$

$\{1, "Poorna"\}$

We can see, duplicates are not allowed in set

Set operations

* Adding a new element

$S1 = \{1, "a", True, 2, "b", False\}$

$S1.add("Hello")$

$S1 \rightarrow$

$\{1, 2, False, 'Hello', 'a', 'b'\}$

$\rightarrow$ There is no proper sequence in set so the indexing is not possible in sets.

* Updating multiple elements

$S1 = \{1, "a", True, 2, "b", False\}$

$S1.update([10, 20, 30])$

$S1 \rightarrow$

$\{1, 10, 2, 20, 30, False, 'a', 'b'\}$

* Removing an element

$S1 = \{1, "a", True, 2, "b", False\}$

$S1.remove("b")$

$S1$

$\{1, 2, False, 'a'\}$

set functions.

* Union and Intersection

* Union

$$S1 = \{1, 2, 3\}$$

$$S2 = \{"a", "b", "c"\}$$

$S1.union(S2) \rightarrow$ two sets.

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

* Intersection

$$S1 = \{1, 2, 3, 4, 5, 6\}$$

$$S2 = \{5, 6, 7, 8, 9\}$$

$S1.intersection(S2) \rightarrow$ sets.

$$\{5, 6\}$$

To Find out the common elements between two

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")

elif $b > a$:

print("b is greater than a")

if $a > b$:

print("a is greater than b")

else:

print("a is not greater than b")

Run →

a is not greater than b

else if → we will use it when

we want to check the multiple statements.

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.

→ tup1 = (1, 2, 3, 4)

(with tuple)

if 2 in tup1:

print("2 is present in tuple")

Run → 2 is present in tuple

→ tup1 = (1, 2, 3, 4)

if 6 in tup1:

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

```
* l1 = [1, 2, 3, 4, 5]
if l1[1] == 2:
    l1[1] = l1[1] + 100
l1 -> [1, 102, 3, 4, 5]
```

```
* l1 = [1, 2, 3, 4, 5]
if l1[4] == 10:
    l1[1] = l1[1] + 100
else:
    l1[4] = l1[4] + 500
l1 -> [1, 102, 3, 4, 505]
```

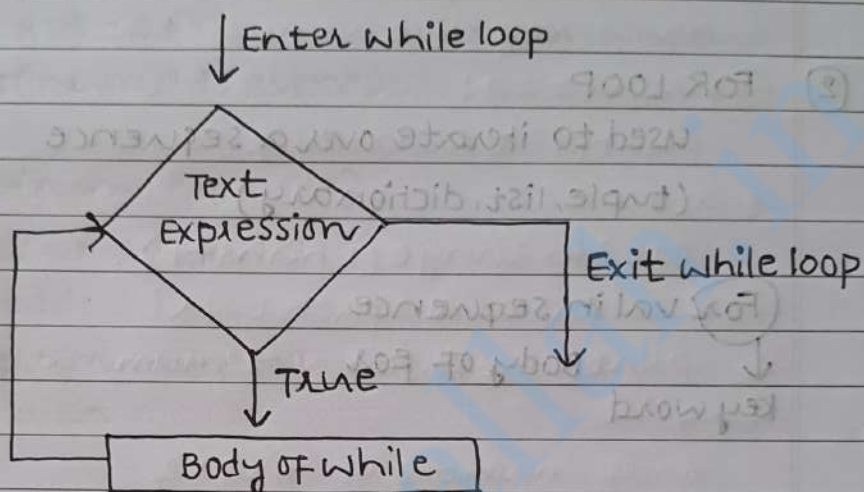
if with dictionary

```
* 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*10=20

* `l1 = [1, 2, 3, 4, 5]` `i = 0`
`while i < len(l1):`
`l1[i] = l1[i] + 100`
`i = i + 1`
`l1 → [101, 102, 103, 104, 105]`

(2) FOR LOOP

used to iterate over a sequence
 (tuple, list, dictionary)

For val in sequence
 ↓
 Body of for
 keyword

* `l1 = ["apple", "banana", "orange"]`
`for i in l1:`
`print(i)`
 → apple
 banana
 orange

* nested for loop

`l1 = ["orange", "blue", "green"]`
`l2 = ["book", "chair", "phone"]`
`for i in l1:`
`for j in l2:`
`print(i, j)`

→ orange book
 chair
 phone
 blue book
 chair
 phone
 green book
 chair
 phone

Basic problems in python

① check even or odd:

```

num = int(input("Enter a number: "))
if (num % 2) == 0:
    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 4 × 3 × 2

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)

```

$n = 123 \rightarrow 321$

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!")

$121/10 = \text{rem}(1)$
 $121/10 = 2$

n
 121
 12
 1

rev

$0 \times 10 + 1 = 0 + 1 = 01$

$1 \times 10 + 2 = 10 + 2 = 12$

$12 \times 10 + 1 = 120 + 1 = 121$

dig

1

2

1

Enter a number: 121

The number is a palindrome

⑥ fibonacci 0 1 1 2 3 5 8

$n = \text{int}(\text{input}(\text{"Enter number: "}))$

$a = 0$

$b = 1$

if $n < 0$:

$\text{print}(\text{"Incorrect input"})$

elif $n == 0$:

$\text{print}(a)$

elif $n == 1$:

$\text{print}(a)$

else:

 for i in range(2, n):

$c = a + b$

$a = b$

$b = c$

$\text{print}(b)$

→ Enter a number: 7

8

a

b

c

0

1

1

1

2

3

2

3

5

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 arguments : expression

* def hello():
 print("Hello world")

hello()

Hello world

* def add10(x):
 return x+10

keyword add(10(10))

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)
```

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 _____

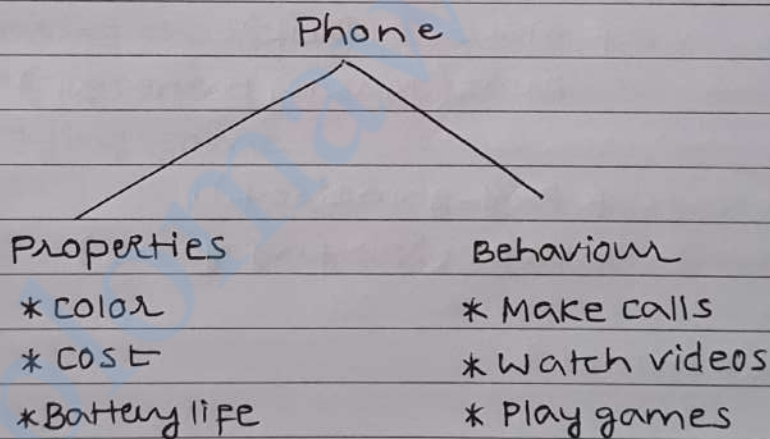
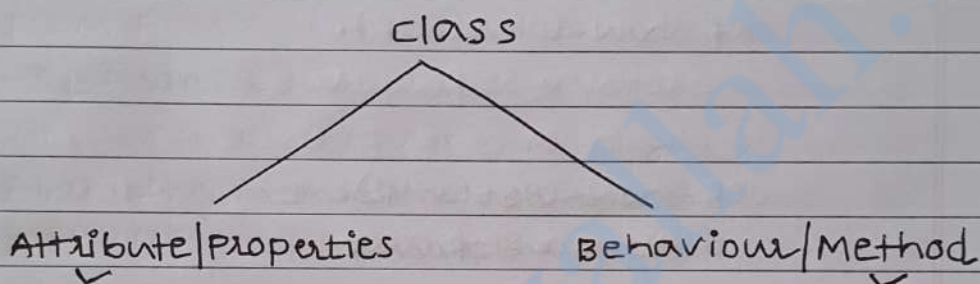
OOPs in python.

we are surrounded with objects

like (mobile, laptop, bike, key, pen, dog, book)

classes:-

class is a template for real world entities



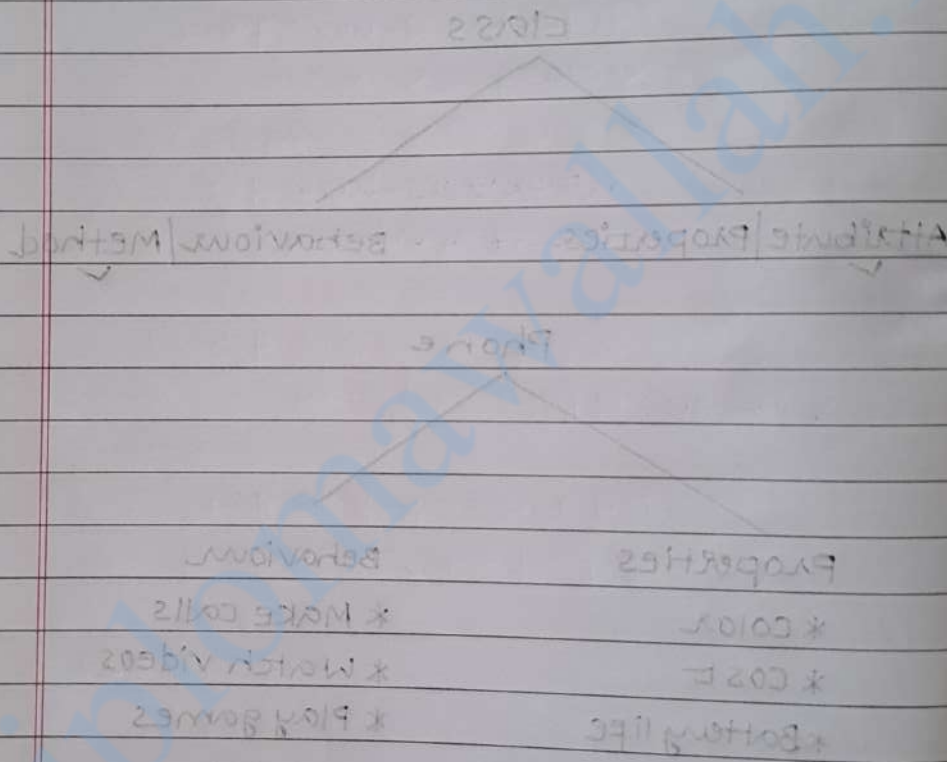
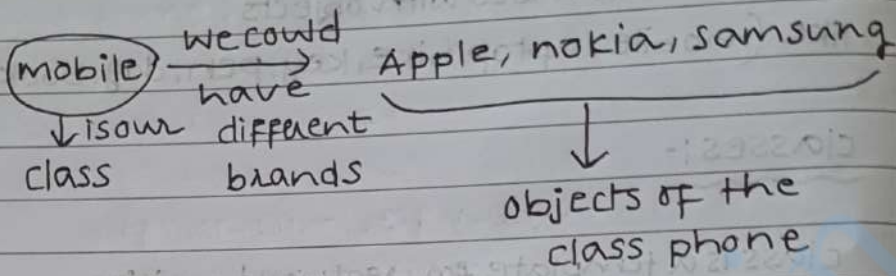
class is a user defined datatype

↳ It is a datatype which we can create by ourselves

oriented programming

objects:-

objects are specific instances of a class.



class is a user defined datatype
It is a datatype which we can
create our own

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")

keyword

methods

Referencing
call

name of object
p1 = phone()

invoking methods through object

name of object

name of the method

p1.make-call

→ making phone call

p1.play-game

→ Playing game

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):
```

↳ init method acts as constructor

```
    self.name = name
    self.age = age
    self.salary = salary
    self.gender = gender
```

↓
Spl type of
function

```
    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, 100,000, 'Male')
```

↓ 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) → Instantiating the object

c1.show-details() for child class

I am a vehicle

Mileage of vehicle is 200

Cost of vehicle is 1200

c1.show-car() → Invoking the child

I am a car 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

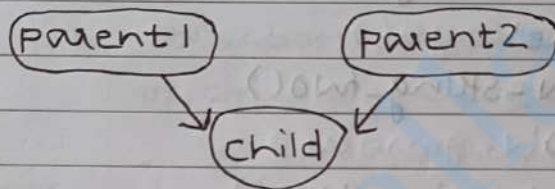
Multiple Inheritance

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
```

(parent class 1)

```
    def show_string_one(self):
```

```
        return self.str1
```

```
class parent2():
```

```
    def assign_string_two(self, str2):
```

```
        self.str2 = str2
```

```
    def show_string_two(self): (parent class 2)
```

```
        return self.str2
```

```
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'

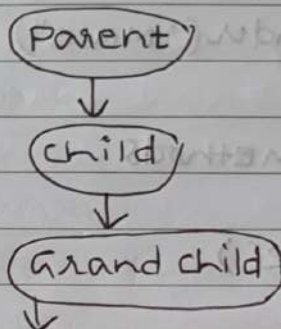
d1.show-string-three()

→ 'three'

Multi-level Inheritance

We have parent, child and

grandchild relationship.



It has 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
```

g1.ch grandchild()

g1.assign_name("poorna")

g1.assign_age(25)

g1.assign_gender("male")

Invoking class methods

g1.show_name()

→ poorna

g1.show_age()

→ 25

g1.show_gender()

→ male

Continuation of python

classmate

Date

Page

(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

(01)

i - signed integer (-1 to +1)

I - unsigned integer

```
from array import *  
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 don't know the range

```
for i in range(len(vals)):
```

```
    print(vals[i])
```

```
for e in vals:
```

```
    print(e)
```

* Can we work with character

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:
```

```
    printe
```

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)
```

```
<<2>> val = int(input("Enter the value for search"))
```

```
<<2>> 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

classmate

Date _____

Page _____

What is

compile time error $\rightarrow a = \text{hello}$ (Problem in syntax)

Runtime error $\rightarrow L = [1, 2, 3] \text{ print}(L[4])$

logical error $\rightarrow 4 + 4 / 2$ (Problem with logic)
(5)

Exceptions:- Runtime errors as exception

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

Exception handling.

keywords -

try:

except:

else:

finally: } optional

try:-

contains operations

except:-

What has to be done

write that code here.

Types of error

1) $n, x = 5, 6, \text{"PooLha"}$

value error = too many values to unpack

2) $L = [1, 2, 3]$

$L[4]$

index error = list index out of range

python.org

(3) b=10

a

Name error: name a is not defined

(4) import poorna

module not found error = no module named 'poorna'

(5) 2+'3'

Type error: unsupported operand type(s) for +
'int' and ('str')Program

(*) import math

enter negative number to check what happens

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

(*) Program

```

import math

num = int(input("Enter no to compute factorial if : "))

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 month (1-12) : "))
    if month < 1 or month > 12:
        raise ValueError
    return month

valid = False
month_name = ("Jan", "Feb", "March", "Apr", "May", "June",
              "July", "Aug", "Sept", "Nov", "Dec")

while not valid:
    try:
        month = getmonth()
        print("The month you entered is",
              month_name[month-1])
        valid = True
    except ValueError:
        print("Invalid month entry\n")

```

④ 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 eu-msg:
```

```
    print(eu-msg)
```

↳ This is an object of ValueError

④ 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

② try:

a = 5

b = 5 | 0

except ZeroDivisionError:

print("ZDE") ✓

except BaseException:

print("BE")

except Exception:

print("E")

③ try:

a = 5

b = 5 | 0 → raise ZDE

except BaseException:

print("BE")

except Exception:

print("E")

except ZeroDivisionError:

print("ZDE")

Base exception

exception

ArithmeticError

ZDE

ZDE is a Base exception

If tries to match

ZDE

is a

ZDE

is a

RE

ZDE is a Base exception error

④ 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:
    print("Hello")

```

→ It will give us compile time error.

```

⑧ try:
    a = 5
    b = 5/0
except:
    print("Hi") ✓ except overcooked error
else:
    print("Hello")
finally:
    print("Namaste") ✓
    whether there is exception is executed or not
    Finally will be executed.
  
```

— | end | —
of exception handling

(29)

Ways of creating Arrays in Numpy

array(), linspace(), 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 concatenate

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

copying

```

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

```

view is a function that creates new arrays

✓ shallow copy

The both the arrays 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 *
arr1 = 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 *

```
arr1 = array([
    [1, 2, 3, 6, 2, 9],
    [4, 5, 6, 7, 5, 3],
])
```

arr2 = arr1.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

```

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

func name

Sweet

```

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

parameter or argument

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 arguments in python

How to pass a parameter to function

def update(x):

x = 8

print(x)

update(10)

(34) Types of Arguments

→ a, b → Formal Argument

def add(a, b):

c = a + b

print(c)

add(5, 6) → 11

Argument (Actual Argument)Position

def person(name, age):

print(name)

print(age)

person("Pooja", 23)

Keyword

person(age=28, name="Pooja")

default

* def person(name, age=18):

print(name)

print(age)

person("Pooja") → Pooja.

variable length argument

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", mob=00)
```

36) Global keywords.

scope

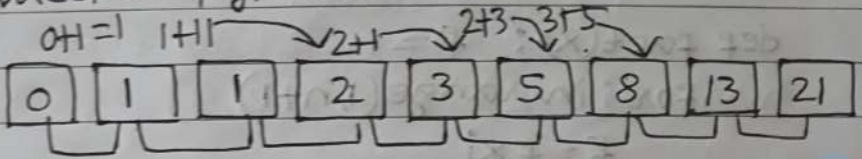
(a=10) → This a this is outside the fun is a global

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

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, x+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 → Anonymous function
function without names are called as
Anonymous 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]

filter = evens = list(filter(lambda n: n%2==0, nums))

print(evens)

doubles = list(map(lambda n: n*2, evens))

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)
```

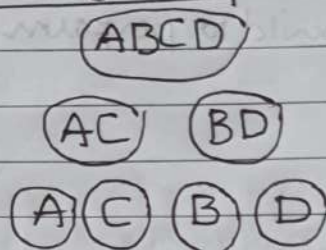
```
    return inner
```

```
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

$a = 9$

$b = 7$

$c = \text{calc.add}(a,b)$

from calc import *

$a = 9$

$b = 7$

$c = \text{sub}(a,b)$

→ 2

(46)

--name-- == "--main--"
special variable --name--

print(--name--)

→ --main-- → starting point of program

(47)

continued part

demo from calc import add

calc

def fun1():
 add()
 print("from fun1")

def add():

print("result 1 is", --name--)

def fun2():

print("from fun2")

def sub():

print("result 2 is")

fun1()

def main():

fun2()

print("in calc main")

def main():

add()

→ fun1()

sub()

fun2()

if --name-- == "--main--":

main()

main()

→ in calc main

result 1 is --main--

result 2 is

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.
top 10 perfect squares

```
def topten():
    n = 1
    while n <= 10:
        sq = n * n
        yield sq
        n += 1
```

values = topten()

for i in values:

print(i)