

" PYTHON "

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 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.

"PYTHON"

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

$$a/b = -$$

3) Decision Making statements:

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

if marks > 70 : give practice test
else 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 time it goes back to start *

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
- * Write 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 objects

All having

Different values for properties

Different values for behaviours

6) Algorithmic approach to solve problem

Step by step approach

To solve a problem is known as Algorithm.

Input — steps to be — output

followed by

Angular JS

Java Script

Node JS

Javascript

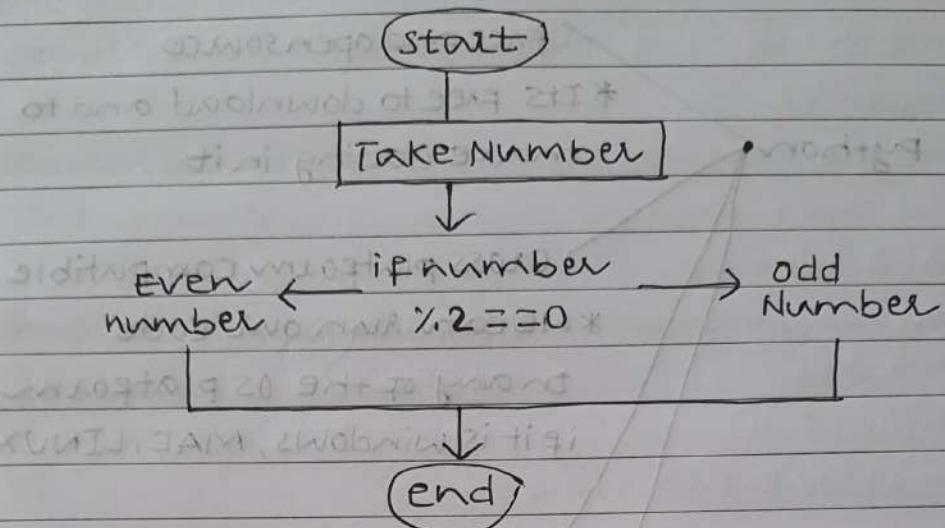
React JS

ES6

Angular SPA

Java Script

Algorithm to find if number even/odd.



7 Introduction to python.

Python

free and open source

* ITS free to download and to make coding in it

cross platform compatible

* we can run our code

in many of the OS platform
if it is windows, MAC, LINUX

object oriented

* By using the object we can get solution for the problem

large standard library

It provides much more libraries which we are in need

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

→ 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'

* $a1 = 100$

$a1$

$\rightarrow 100$

$type(a1)$

$\rightarrow int$

* $a1 = 3.14$

$a1$

$\rightarrow 3.14$

$type(3.14)$

$\rightarrow float$

$a1 = 3.14$

$type(a1)$

$\rightarrow float$

* $a1 = True$

$a1$

$\rightarrow True$

$type(a1)$

$\rightarrow bool$

* $a1 = "Poojna"$

$a1$

$\rightarrow 'Poojna'$

$type(a1)$

$\rightarrow str$

* complex number

Both Real and Imaginary part

$a1 = 6+9j$

$a1$

$\rightarrow 6+9j$

$type(a1)$

$\rightarrow complex$

(10) operators in python.

Arithmetic operators

Relational operator

logical operators "and", "or", "not"

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

Arithmetic operators

$a = 10$

$b = 20$

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.

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

→ 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 ($0 \& 0 = 0$) or $\leftarrow 0 \& 0$

| or operator

It will give us 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) $b | b \rightarrow \text{False}$

(1) 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 (''' ''')

$b1 = 'Hello world'$

$b1 = "This is poorna"$

$b1 = """I am going to kerala tomorrow"""$

$b1 = 'Hello world'$

$b1 \rightarrow 'Hello world'$

$b1 = "This is poorna"$

$b1 \rightarrow 'This is poorna'$

$b1 = """This is a multiline
string$

'''

$b1 \rightarrow 'this is a multiline\nstring\nin'$

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")

→ 'My name is poorna'

* Number of occurrences of substring

new_string = "hello hello hello world"

new_string.count("hello") → [P]

→ 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)

* 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)`

* Repeating 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.

`l1 = [1, "Pooja", 3.14, True, 5+9j]`

`type(l1) → list`

* Extracting individual elements

`l1[-1] → 5+9j`

`l1[1:4] → ['Pooja', 3.14, True]`

* Modifying a list

changing the element of 0th index

`l1 = [1, "a", 2, "b", 3, "c"]`

`l1[0] → 100`

`l1 →`

`[100, 'a', 2, 'b', 3, 'c']`

Appending a new element

`l1 = [1, "a", 2, "b", 3, "c"]`

`l1.append("pooja")`

`l1`

`→ [1, 'a', 2, 'b', 3, 'c', 'pooja']`

Popping the last element

`l1 = [1, "a", 2, "b", 3, "c"]`

`l1.pop()`

`l1`

`→ [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, "Poojna", "Poojna", 1}

s1 →

{1, "Poojna"}

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

$$s1 = \{1, 2, 3\}$$

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

$s1.union(s2) \rightarrow$ two sets.

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

* Intersection

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

To find out the common

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

elements between two

$s1.intersection(s2) \rightarrow$ sets.

$$\{5, 6\}$$

If statement in python.

* If statement,

if

it's raining

else

go out and play

sit inside

① a = 10

b = 20

if a > b:

print("a is greater than b")

else 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")

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 $l1[1] == 2:$: (Condition & Block)
 $l1[1] = l1[1] + 100$: (Assignment Statement)
 $l1 \rightarrow [1, 102, 3, 4, 5]$: (Change in List)

- * $l1 = [1, 2, 3, 4, 5]$
 if $l1[4] == 10:$: (Condition & Block)
 $l1[1] = l1[1] + 100$: (Assignment Statement)
 else:
 $l1[4] = l1[4] + 500$: (Assignment Statement)
 $l1 \rightarrow [1, 102, 3, 4, 505]$: (Change in List)

if with dictionary

- * $di = \{"a": 1, "b": 2, "c": 3\}$
 if $di["b"] == 2:$: (Condition & Block)
 $di["b"] = di["b"] + 100$: (Assignment Statement)
 $di \rightarrow \{"a": 1, "b": 102, "c": 3\}$: (Change in Dictionary)

* $l = [1, 2, 3, 4, 5]$: $i = 0$

while $i < \text{len}(l)$:

$l[i] = l[i] + 100$

$i = i + 1$

$l \rightarrow [101, 102, 103, 104, 105]$

② FOR LOOP

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

For val in sequence

Body of for

keyword

* $l = ["apple", "Banana", "orange"]$

for i in l :

print(i)

→ apple
banana
orange

* nested for loop

$l = ["orange", "blue", "green"]$

$l2 = ["book", "chair", "phone"]$

for i in l :

for j in $l2$: → orange book

print(i, j)

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: → 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 zero

```
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")
```

clif 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 $4 \times 3 \times 2$

24

Enter a number

3628800

; 0<num>3

(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!")
```

$121 // 10 - \text{temp}(1)$

$121 // 10 - 2$

n

121

12

1

rev

$0 * 10 + 1 = 0 + 1 = 1$

$1 * 10 + 2 = 10 + 2 = 12$

$12 * 10 + 1 = 120 + 1 = 121$

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(b)
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	2	3	
2	3	5	

10-140-160x0
classmate
classmate

11 12 13 14 15 16 17 18 19 20

fib
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6774
10955
17711
28661
46372
75033
121375
196408
317783
514191
831974
1345945
2177919
3523864
5696783
9220647
14917429
24134076
39051505
63185581
102237086
165422567
267659653
433082220
700712273
1133794473
1834506746
3068299219
5002805965
8071105184
13073911019
21145016103
34218927122
55363943225
89582867347
144946810572
234533677919
379477488491
614011166410
993488654901
1607500320011
2599988974912
4207489294923
6807478269835
10014967568188
16822445837023
26837413405211
43659859242232
70497272647443
114157131669755
184654354337218
308811485607473
513465839944691
822277725552164
1335743535496855
2158021260988515
3503764796485370
5661785957473885
9165550754959255
14827336712433100
24092887467002355
38910224179435455
62903111646437800
101813335845772255
164716447492109055
266530783337881255
431247230829990310
700000000000000000
1131247230829990310
1831247230829990310
2962500000000000000
4893747230829990310
7855747230829990310
12748747230829990310
2050447230829990310
33252947230829990310
5375747230829990310
8695000000000000000
14070747230829990310
22765747230829990310
3752847230829990310
60293147230829990310
97821547230829990310
1580147230829990310
25582947230829990310
41184347230829990310
66767247230829990310
108549547230829990310
175116747230829990310
283666247230829990310
45878247230829990310
742448647230829990310
1201231147230829990310
1943679747230829990310
3145311147230829990310
5088690847230829990310
8234000000000000000
1332269747230829990310
2155739747230829990310
3510000000000000000
5665739747230829990310
9175739747230829990310
1484147230829990310
24683247230829990310
403247230829990310
6500000000000000000
1053247230829990310
1703247230829990310
275647230829990310
4512747230829990310
72692147230829990310
118819547230829990310
191618747230829990310
303438247230829990310
50687647230829990310
8103147230829990310
131719147230829990310
213438247230829990310
355157347230829990310
568595547230829990310
9237520829990310
1592347230829990310
258469547230829990310
417704247230829990310
676173747230829990310
115327747230829990310
188645247230829990310
303972747230829990310
492618247230829990310
796590747230829990310
129318247230829990310
208636747230829990310
337955247230829990310
546591747230829990310
884547230829990310
143114247230829990310
231538247230829990310
372656747230829990310
6041947230829990310
976841247230829990310
1553659747230829990310
2517319247230829990310
4030938747230829990310
654825747230829990310
1057821247230829990310
1715643747230829990310
2773466247230829990310
454693247230829990310
7320395747230829990310
1186735247230829990310
19227719747230829990310
3045543747230829990310
5068315747230829990310
813663147230829990310
13204419747230829990310
2137223747230829990310
3554665747230829990310
568231747230829990310
9236635247230829990310
1511895747230829990310
2423783747230829990310
3935675747230829990310
635945147230829990310
1029512747230829990310
1669459747230829990310
2708819247230829990310
4417638747230829990310
7126458247230829990310
1154384747230829990310
1866773747230829990310
302315747230829990310
4989541747230829990310
8012983747230829990310
129924247230829990310
2198583747230829990310
3597125747230829990310
579566747230829990310
95912347230829990310
1538587747230829990310
25571747230829990310
411575747230829990310
6673342747230829990310
108487147230829990310
1752245747230829990310
283412247230829990310
4668247230829990310
750247230829990310
1216671747230829990310
1975119247230829990310
3250237747230829990310
5225355247230829990310
8475592747230829990310
13700947230829990310
2237643747230829990310
3677738247230829990310
5955475747230829990310
961125247230829990310
1552263747230829990310
2504431247230829990310
4008663747230829990310
6512905247230829990310
10521547230829990310
1603419247230829990310
2656533747230829990310
4312955247230829990310
6929377747230829990310
1124831247230829990310
1737753747230829990310
2764585247230829990310
4502367747230829990310
7267950247230829990310
1177028747230829990310
1843812247230829990310
3000634747230829990310
4944467247230829990310
7948330747230829990310
1289275747230829990310
2083113247230829990310
3366226747230829990310
5532453247230829990310
8900000000000000000
1443247230829990310
2266583747230829990310
3730866747230829990310
6297750247230829990310
1009551747230829990310
1619335247230829990310
2638668747230829990310
4277236247230829990310
6955904747230829990310
11211761747230829990310
1812551747230829990310
2924103747230829990310
4738655247230829990310
7667307747230829990310
1240596247230829990310
2001192747230829990310
3202388747230829990310
5103584247230829990310
8206966747230829990310
1331055247230829990310
2151413747230829990310
3402827247230829990310
5554243747230829990310
8956650247230829990310
1451128747230829990310
2302552247230829990310
3604075747230829990310
5905600247230829990310
9507123747230829990310
1541275247230829990310
2462548747230829990310
4024126247230829990310
6486703747230829990310
1056333247230829990310
1692996747230829990310
2755960247230829990310
4418923747230829990310
7137886247230829990310
1155684747230829990310
1871571247230829990310
3027145747230829990310
4894720247230829990310
7919442747230829990310
1281416747230829990310
2062833247230829990310
3325666747230829990310
5388500247230829990310
8777002747230829990310
1416553747230829990310
2293387247230829990310
3786770747230829990310
6274553247230829990310
10049110747230829990310
1612464747230829990310
2625928247230829990310
4248491747230829990310
6872455247230829990310
1111594747230829990310
1832843247230829990310
2964685747230829990310
4800528247230829990310
7765156747230829990310
1256571247230829990310
2013143747230829990310
3226715247230829990310
5140457747230829990310
8361110247230829990310
1350156747230829990310
2186312247230829990310
3562464747230829990310
5724827247230829990310
9287250747230829990310
15019675247230829990310
2403930747230829990310
3905893247230829990310
6309825747230829990310
10214682747230829990310
1632435747230829990310
2664873247230829990310
4329708747230829990310
7009512247230829990310
1132923747230829990310
1835855247230829990310
2968707747230829990310
4707560247230829990310
7676312747230829990310
1244583247230829990310
2009465747230829990310
3218951247230829990310
5128433747230829990310
8347866247230829990310
13476291747230829990310
2185158747230829990310
3572717247230829990310
5755474747230829990310
9311251247230829990310
1506663747230829990310
2413325247230829990310
3927687747230829990310
6345350247230829990310
10273025747230829990310
1644665247230829990310
2669327747230829990310
4338690247230829990310
7007352747230829990310
1131101747230829990310
1832474247230829990310
2964836747230829990310
4707709247230829990310
7675481747230829990310
1244320747230829990310
2009193247230829990310
3218565747230829990310
5127438247230829990310
8345810747230829990310
1347353247230829990310
2185025747230829990310
3572388247230829990310
5755260747230829990310
9310933247230829990310
1506865747230829990310
2413538247230829990310
3927900747230829990310
6345673247230829990310
1027434747230829990310
1644207247230829990310
2669879747230829990310
4338752247230829990310
7007424747230829990310
1131211747230829990310
1832484247230829990310
2964856747230829990310
4707729247230829990310
7675501747230829990310
1244323247230829990310
2009195747230829990310
3218568247230829990310
5127440747230829990310
8345813247230829990310
1347356247230829990310
2185128747230829990310
3572391247230829990310
5755263747230829990310
9310936247230829990310
1506866747230829990310
2413539247230829990310
3927911747230829990310
6345683247230829990310
1027437247230829990310
1644210747230829990310
2669883247230829990310
4338755747230829990310
7007428247230829990310
1131214747230829990310
1832487247230829990310
2964860747230829990310
4707733247230829990310
7675510747230829990310
1244327247230829990310
2009199247230829990310
3218573247230829990310
5127446747230829990310
8345825747230829990310
1347360247230829990310
2185141747230829990310
3572393247230829990310
5755268747230829990310
9310942247230829990310
1506869747230829990310
2413540247230829990310
3927921747230829990310
6345691247230829990310
1027438747230829990310
1644222747230829990310
2669895247230829990310
4338758747230829990310
7007434247230829990310
1131217747230829990310
1832491247230829990310
2964863747230829990310
4707741247230829990310
7675520747230829990310
1244329247230829990310
2009198247230829990310
3218576247230829990310
5127448747230829990310
8345831747230829990310
1347363247230829990310
2185144747230829990310
3572396247230829990310
5755273747230829990310
9310948247230829990310
1506871247230829990310
2413543247230829990310
3927931747230829990310
6345696747230829990310
1027440247230829990310
1644233747230829990310
2669906247230829990310
4338763747230829990310
7007447247230829990310
1131220747230829990310
1832498247230829990310
2964866747230829990310
4707751247230829990310
7675530747230829990310
1244330247230829990310
2009199747230829990310
3218579247230829990310
5127451747230829990310
8345841747230829990310
1347366247230829990310
2185147747230829990310
3572399247230829990310
5755278747230829990310
9310958247230829990310
1506874247230829990310
2413546247230829990310
3927941747230829990310
6345699747230829990310
1027447247230829990310
1644243747230829990310
2669916247230829990310
4338773747230829990310
7007454247230829990310
1131223747230829990310
1832500247230829990310
2964876747230829990310
4707761247230829990310
7675540747230829990310
1244336247230829990310
2009203247230829990310
3218583247230829990310
5127458747230829990310
8345851747230829990310
1347371247230829990310
2185151747230829990310
3572400247230829990310
5755283747230829990310
9310968247230829990310
1506876247230829990310
2413553247230829990310
3927951747230829990310
6345700747230829990310
1027450247230829990310
1644253747230829990310
2669926247230829990310
4338783747230829990310
7007464247230829990310
1131226747230829990310
1832508247230829990310
2964879747230829990310
4707771247230829990310
7675550747230829990310
1244343247230829990310
2009209247230829990310
3218586247230829990310
5127468747230829990310
8345861747230829990310
1347376247230829990310
2185158747230829990310
3572403247230829990310
5755289747230829990310
9310978247230829990310
1506877247230829990310
2413556247230829990310
3927954747230829990310
6345706747230829990310
1027457247230829990310
1644263747230829990310
2669933247230829990310
4338813747230829990310
7007471247230829990310
1131229747230829990310
1832516247230829990310
2964880747230829990310
4707781247230829990310
7675560747230829990310
1244353247230829990310
2009213247230829990310
3218593247230829990310
5127478747230829990310
8345871747230829990310
1347380247230829990310
2185161747230829990310
3572406247230829990310
5755293747230829990310
9310988247230829990310
1506878247230829990310
2413563247230829990310
3927957747230829990310
6345710747230829990310
1027458247230829990310
1644273747230829990310
2669940247230829990310
4338823747230829990310
7007484247230829990310
1131231747230829990310
1832520247230829990310
2964887747230829990310
4707791247230829990310
76755707472

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

`add10(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]
```

(odd numbers)

Q5

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 _____

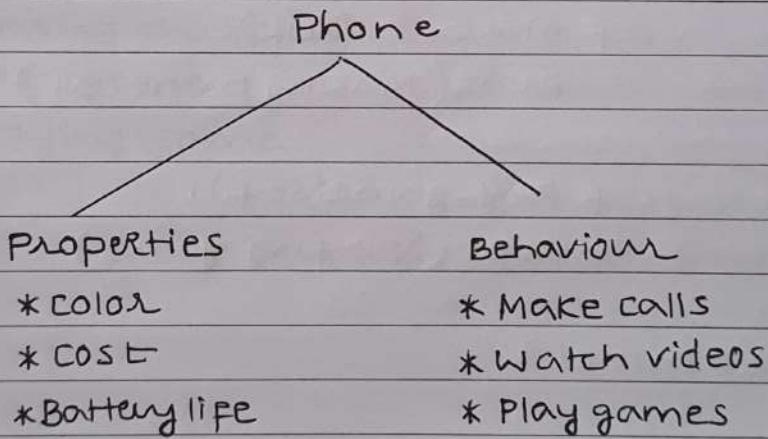
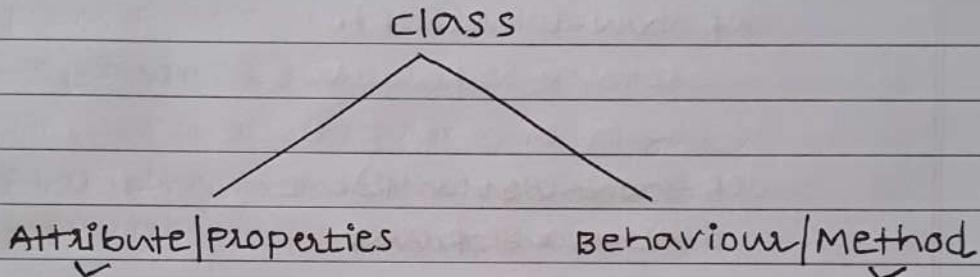
OOPS in python

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

Pramana sang bastri

objects:-

objects are specific instances of a class.

mobile → we could have different brands → Apple, nokia, samsung
 ↓ ↓ ↓
 class objects of the
 ↓ ↓ ↓
 objects of the
 class phone

22/01/20

phone

- ~ behaviors
- ~ * make calls
- ~ * receive calls
- ~ * move it
- ~ * play games
- ~ * take photo
- ~ * edit photo

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")

↑ methods

→ Referencing
call

keyword

name — pl = phone()

of object

invoking methods through object

name of object → name of the method

pl.make-call

→ Making phone call

pl.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.type_of
 function

 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, 100000, "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 100000

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)
```

```
VI = vehicle(500, 500)
```

```
VI.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

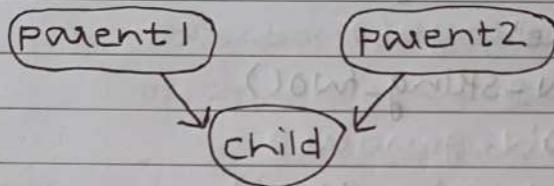
Multiple Inheritance

Multi-level Inheritance

Hybrid Inheritance

Multiple Inheritance

The child inherits from more than 1 parent class.



```
class Parent1():
    pass
```

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

(Parent class 1)

```
    def show_string_one(self):
        return self.str1
```

```
class Parent2():
    pass
```

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

```
    def show_string_two(self): (Parent class 2)
        return self.str2
```

```
class Derived(Parent1, Parent2):
    pass
```

```
    def assign_string_three(self, str3):
        self.str3 = str3
```

```
    def show_string_three(self):
        return self.str3
```

creating the object of the child class

di. derived

di. assign-string-one("one")

di. assign-string-two("two")

di. assign-string-three("three")

Invoking methods

di.show-string-one()

→ 'one'

di.show-string-two()

→ 'two'

di.show-string-three()

→ 'three'

(parent class)

: (child class)

late2, 3192 → new2

(parent class)

: (child class)

late2, 3192 → new2

(parent class)

: (child class)

late2, 3192 → new2

: (parent class)

: (child class)

late2, 3192 → new2

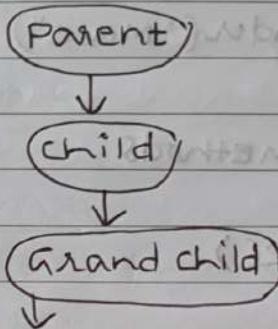
: (parent class)

late2, 3192 → new2

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
```

gl.ch grandchild()

gl.assign-name("Pooja")

gl.assign-age(25)

gl.assign-gender("Male")

Invoking class methods

gl.show-name()

→ Pooja

gl.show-age()

→ 25

gl.show-gender()

→ 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 don't 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) → $\frac{a}{i}$

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

from array import *

vals = (1, [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

CLASSEmate

Date _____

Page _____

What is

Compile time error \rightarrow `a = hello"` (Problem in syntax)

Runtime error \rightarrow `L=[1, 2, 3] print(L[4])`

Logical error \rightarrow `4+4/2` (Problem with logic)
⑥

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, "Pooja"`

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 **NameError:** name a is not defined(4) `import poornav`**ModuleNotFoundError:** No module named 'poornav'(5) $2 + '3'$ **TypeError:** unsupported operand type for +
'int' and 'str'Program(*) `import math`

enter negative number to check what happens

```
num = int(input("Enter number to compute factorial:"))
print(math.factorial(num))
```

```
try:
    num = int(input("Enter number to compute 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
```

```
monthname = ("Jan", "Feb", "March", "Apr", "May", "June",  
            "July", "Aug", "Sept", "Nov", "Dec")
```

```
while not valid:
```

```
    try:
```

```
        month = getmonth()
```

```
        print("The month you entered is",  
              monthname[month - 1])
```

```
        valid = True
```

```
    except ValueError:
```

```
        print("Invalid month entry")
```

(*) 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 e:
```

```
    print(e)
```

↳ It is an object of ValueError

(*) 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("ErrorMessage", 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:

$a = 5$

$b = 5/0 \rightarrow$ raise ZDE

except BaseException:

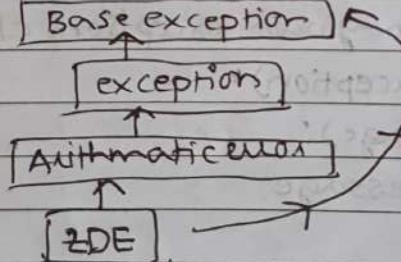
 print("BE")

except Exception:

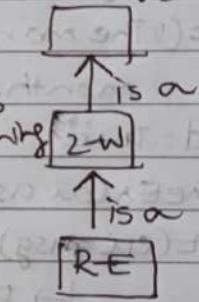
 print("E")

except ZeroDivisionError:

 print("ZDE")



ZDE is a Base exception
vehicle



- ④ The else part will be 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")

try:

(5) $a = 5$

$b = 5 / 0$

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") ✓ except overcooked error

else:

print("Hello")

finally:

print("Namasthe") ✓

whether there is exception is executed or not

Finally will be executed.

try:

cook-biryani()

except salt less error

finally:

switchoff-stove

→ end →

of exception handling

- (29) Ways of creating Arrays in Numpy
 $\text{array}()$, $\text{linspace}()$, $\text{logspace}()$,
 $\text{arange}()$, $\text{zeros}()$, $\text{ones}()$

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

- * $\text{linspace}()$ (start, stop, step) → dividing into the parts

```
arr = linspace(0, 15, 16)
```

```
→ 1, 3, 5, 7, 9, 11, 13
```

- * $\text{arange}()$

```
arr = arange(1, 15, 2)
```

```
print(arr)
```

```
→ 1, 3, 5, 7, 9, 11, 13
```

- * $\text{logspace}()$

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

```
print(arr)
```

- * $\text{zeros}()$ $\text{ones}()$ → 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)
```

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

view is a function that creates new arrays

shallow copy: \rightarrow (memory) same

The both the arrays are interlinked

```
from numpy * import *
```

```
arr1 = array([2, 6, 8, 1, 3])
```

```
arr2 = arr1.view()
```

```
arr1[1] = 7
```

```
print(arr1)  $\rightarrow$  2, 7, 8, 1, 3
```

```
print(arr2)  $\rightarrow$  2, 7, 8, 1, 3
```

[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)  $\rightarrow$  1, 7, 3, 4, 5
```

```
print(arr2)  $\rightarrow$  1, 2, 3, 4, 5
```

[1, 2, 3, 4, 5]

[1, 7, 3, 4, 5]

[1, 2, 3, 4, 5]

[1, 7, 3, 4, 5]

[1, 2, 3, 4, 5]

[1, 2, 3, 4, 5]

[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) → shows the no. of dimension

print(arr.shape) → 2 rows, 3 columns

print(arr.size) → 6 (counts the size)

I want to convert 2d array into 1d array

(*) From numpy import *

arr = array([

[1, 2, 3],

[4, 5, 6]

])

arr2 = arr1.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 = 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

function 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 arguments in python

How to pass a parameter to functions

def update(x):

x = 8

print(x)

update(10)

8 → (8,) → (10,) → 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("Poojna", 23)

Keyword

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

default

* def person(name, age=18):

print(name)

print(age)

person("Poojna") → Poojna.

variable length argument

<code>def sum(a,b):</code>	<code>def sum(a,*b)</code>
----------------------------	----------------------------

`c=a+b`

`print(c)`

`sum(5,6)`

<code>c=a</code>	<code>:(x) addtion 3+6</code>
------------------	-------------------------------

`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 global

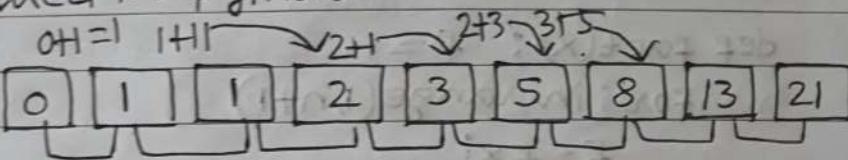
`def something():`

`(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, 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 → 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):  
    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

(ABCD)

(AC) (BD)

(A) (C) (B) (D)

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

```

from calc import add
def fun1():
    print("from fun1")
def fun2():
    print("from fun2")
fun1()
fun2()
def main():
    → fun1()
    fun2()
main()
→ in calc main
result1 is __main__
result2 is

```

```

calc
def add():
    print("resut1 is", __name__)
def sub():
    print("resut2 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 top10():

n = 1

while n <= 10:

sq = n * n

yield sq

n += 1

values = top10()

for i in values:

print(i)