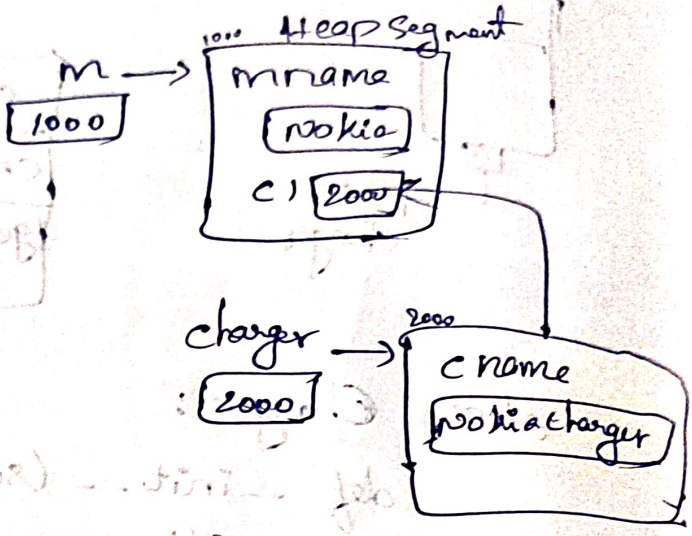
```
del m
Print("After deleting")
Print(m.charge.cname)
charge.getCharger()
```

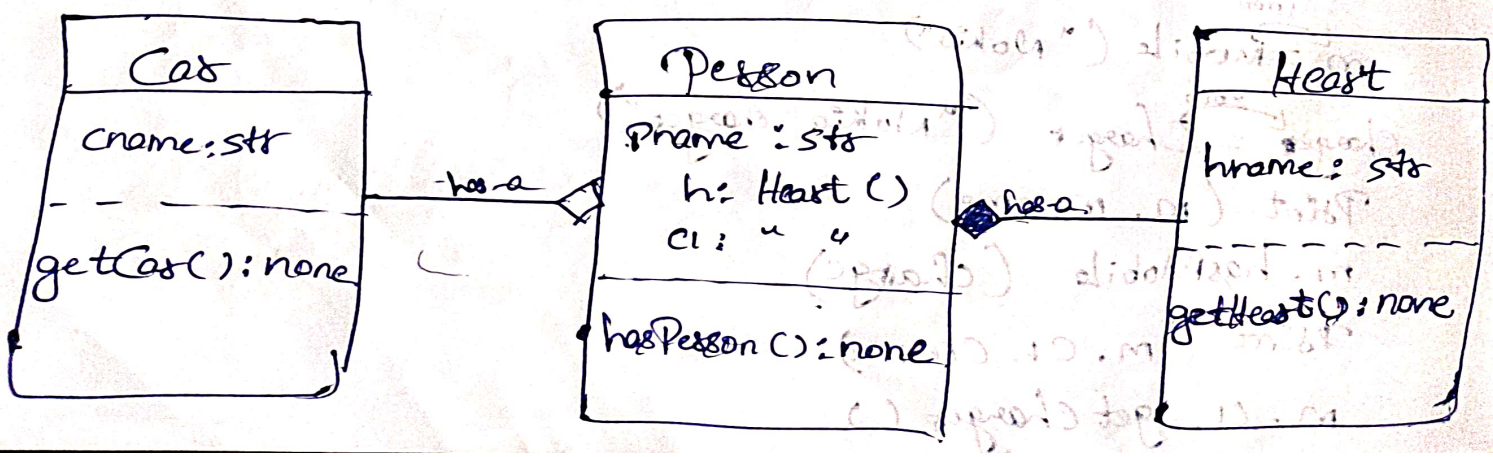


o/p

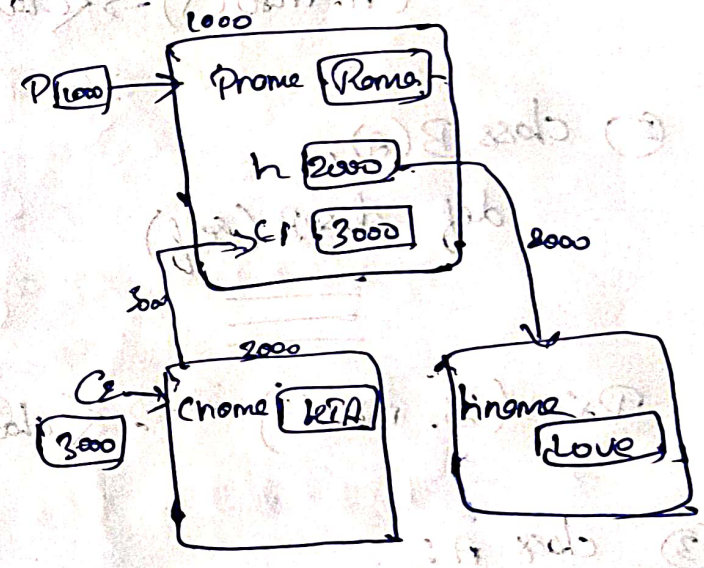
Mobile is ready
Charger is ready
Nokia
Both Mobile and charger
Nokia charger
charger is used for charging.
After deleting
Nokia charger
charger is used for charging

Note:- Aggregate object will be passed as Parameters to the main object, method.

Combining aggregate and Composed object



```
class Car :
    def __init__(self, name):
        self.cname = name
        Print("Car is Ready")

    def getCar(self):
        Print("Car is Ready to Drive")

class Heart;
    def __init__(self, name):
        self.hname = name
        Print("Heart is Ready")

    def getHeart(self):
        Print("Heart is Person")

class Person:
    def __init__(self, name);
        self.pname = name
        self.h = Heart("Love")
        self.c1 = " "
        Print("Person is Ready")

    def hasPerson(self, c):
        self.c1 = c
        Print("Person & Heart Connected")

P = Person("Rome")
C2 = Car("KIA")
P.hasPerson(C2)
Print(P.pname)
```



```
Print(P.h.hname)
Print(P.c1.cname)
P.h.getHeart()
P.c1.getCar()
del P
Print(c2.cname)
c2.getCar()
```

Searching on Method

Method Resolution Order (mro())

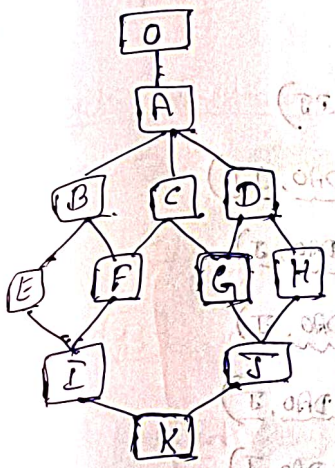
①
```
class A:
    def dispA(self):
        Print("A")
```
Print(A.mro()) → <class A> <class object> then error

②
```
class B(A):
    def dispB(self)
```
Print(B.mro()) → <class B> <class A> <class object> then error

③
```
class A:
    def dispA(self):

class B:
    def dispB(self):
```

```
class C(A,B):
    def dispC(self):
```
Print(c.mro()) → <class C> <class A> <class B> <class object>
then error

Note: If parent are multiple then we use a mro algorithm or C3 linearization algorithm.



mro(A) = A, O
mro(B) = B A O
mro(C) = C A O
mro(D) = D A O
mro(E) = E B A O
mro(F) = F B C A O
mro(G) = G C D A O
mro(H) = H D A O
mro(I) = I E F B C A O
mro(J) = J G H C D A O
mro(K) = I J E F G H B C D A O

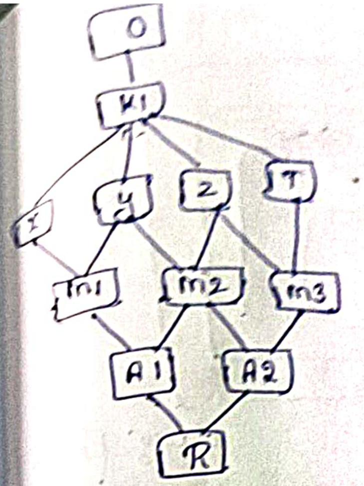
mro(I) = I + C3 algo (mro(E), mro(F), EF)
= I + C3 algo (EBAO, FBCAO, EF)
= IE + C3 algo (BAO, FBCAO, F)
= IEF + C3 algo (BAO, BCAO)
= IEFB + C3 algo (AO, CAO)
= IEFBC + C3 algo (AO, AO)
= IEFBCA + C3 algo (0, 0)

mro(I) = IEFBCAO → If Parents are multiple we can use this method.

mro(J) = J + C3 algo (mro(G), mro(H), GH)
= J + C3 algo (GCDAO, HDAO, GH)
= JG + C3 algo (CDAO, HDAO, H)
= JGH + C3 algo (DAO, DAO, H)
= JGHC + C3 algo
= JGHCD + C3 algo
= JGHCDA + C3 algo
= JGHCDAO

mro(K) = K + C3 algo (mro(I), mro(J), IJ)
= K + C3 algo (IEFBCAO, JGHCDAO, IJ)
= KI + C3 algo (EFBCAO, JGHCDAO, J)
= KIE + C3 algo (FBCAO, JGHCDAO, J)
= KIEF + C3 algo (BCAO, JGHCDAO, J)
= KIEFB + C3 algo (CAO, JGHCDAO, J)
= KIEFBJ + C3 algo (CAO, GHCDAO)
= KIEFBJG + C3 algo (CAO, HCDAO)
= KIEFBJGH + C3 algo (CAO, CDAO)
= KIEFBJGHC + C3 algo (AO, DAO)
= KIEFBJGHCD + C3 algo (AO, AO)
= KIEFBJGHCDA + C3 algo (O, O)

mro(K) = KIEFBJGHCDAO

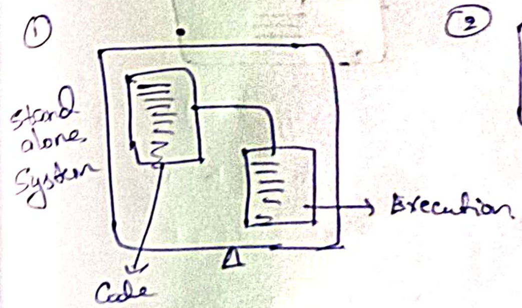


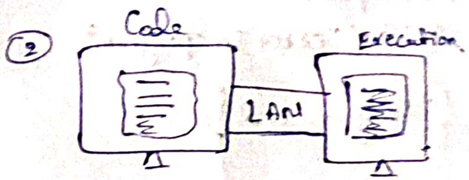
mro(R) = RA1M1 A2 M2 YM3ZTK1O

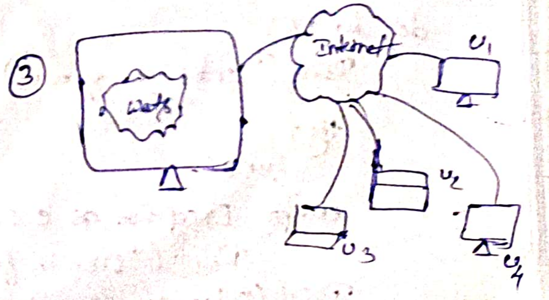
Cyclic Inheritance

-> In Python doesn't support cyclic Inheritance

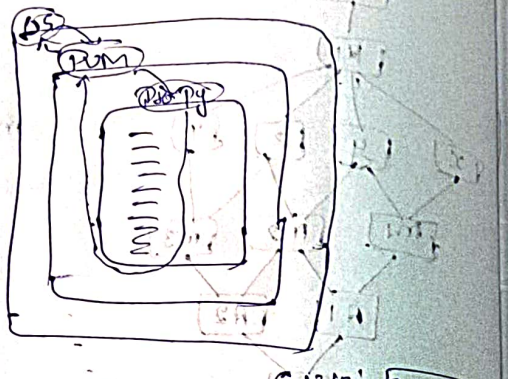
· Exception · Handling :

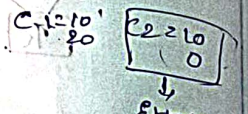






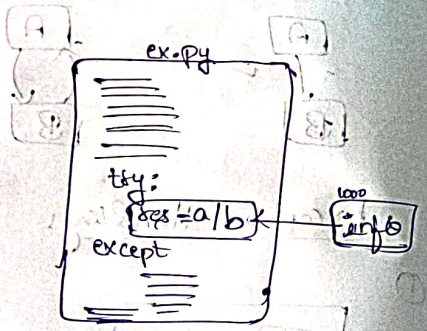
```
Print ("Enter a value 1")
a= int(input())
Print ("Enter a value 2")
b = int(input())
c = a/b
Print ("Res is", c)
Print ("Program end")
```





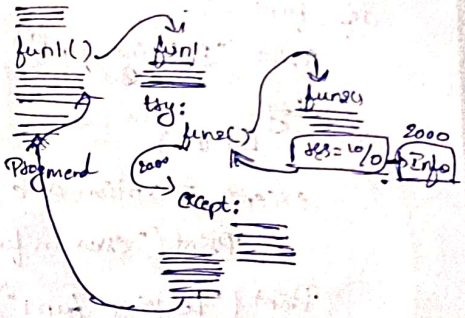
By Using try and except to overcome error.

```
Print ("Enter a value 1")
a = int(input())
Print ("Enter a value 2")
b = int(input())
try:
    res = a/b
    Print(res)
except Exception as e:
    Print ("error in Program")
Print ("Program end")
```



Program

```
def fun1():
    Print ("Entering fun1")
    try:
        fun2()
    except Exception as e:
        Print ("Error in fun1")
    Print ("Leaving fun1")
def fun2():
    Print ("Entering fun2")
    res = 10/0
    Print (res)
    Print ("Leaving fun2")
Print ("Pgm started")
fun1()
Print ("Pgm end")
```



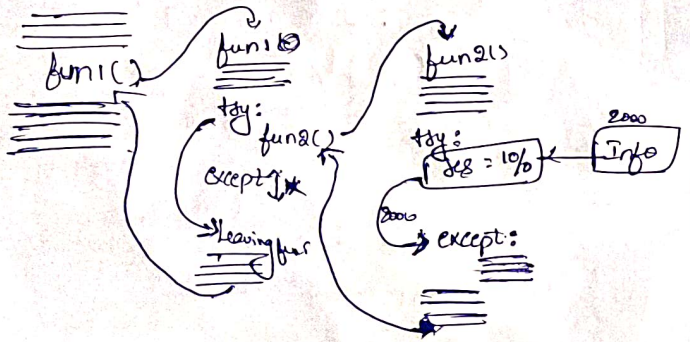
Else block after except

```
Print ("Enter a value 1")
a = int(input())
Print ("Enter a value 2")
b = int(input())
try:
    res = a/b
    Print ("The result =", res)
except Exception as e:
    Print ("Error Occured")
else:
    Print ("No error occured")
```

o/p a=10
5
a=0
No error occured

Reshowing the Error

```
def fun1():
    Print("Entering fun1")
    try:
        fun2()
    except Exception as e:
        Print("error in fun1")
    Print("Leaving fun1")
def fun2():
    Print("Entering fun2")
    try:
        res = 10/0
        Print(res)
    except Exception as e:
        Print("error in fun2")
        raise e
    Print("Leaving fun2")
Print("Program started")
fun1()
Print("Program end")
```



Note:- without raise keyword the program will not printing a error in fun1.

* with using raise keyword the program will printing a error in fun1 in above program.

Handling Specific Exceptions

Exception
- → 10/five ⇒ ValueError
- → 10/0 ⇒ ZeroDivisionError
- → 10/5 ⇒ Exceptions

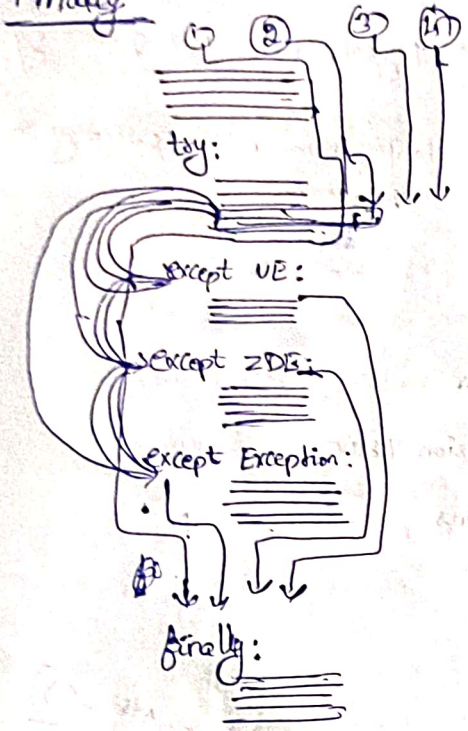
```
try:
    Print("Enter a value1")
    a = int(input())
    Print("Enter a value2")
    b = int(input())
    res = a/b  <- 3000 Info
    Print(res)
except ValueError as e:
    Print("It is ValueError")
    Print(e)
    Print(e.__str__())
except ZeroDivisionError as e:
    Print("It is a ZeroDivisionError")
except Exceptions as e:
    Print("It is a error")
```

e.__str__()

[e.__str__()]

It is used to read the error.

Finally



Case 1 : $\frac{10}{5}$ Execute normally

Case 2 $\frac{10}{Five}$ Value Error

Case 3 $\frac{10}{0}$ → ZDE

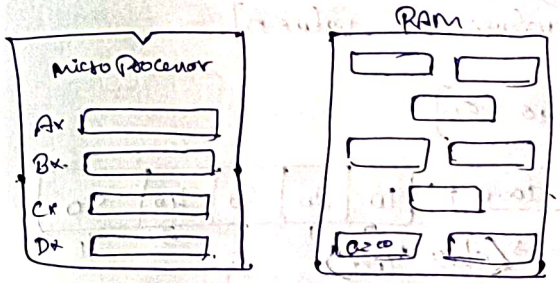
Case 4 $\frac{10}{"5"}$ → Exception

Finally Block

```
def fun1():
    Print("Entesing fun1")
    try:
        fun2()
    except Exception as e:
        Print("Error in fun1")
        raise e
    finally:
        Print("leaving fun1")
```

```
def fun2():
    Print("Entesing fun2")
    try:
        res = 10/0        e: 1000 Info
        Print(res)
    except Exception as e:
        Print("Error in fun2")
        raise e
    finally:
        Print("leaving fun2")
Print("Program started")
try:
    fun1()
except Exception as e:
    Print("error in main")
Print("Program end")
```

List In Python



1 value = 1 var

1000 vals = 1000 vars

1 Searchings } it's difficult in RAM

1 Sortings

Array

a | 1 | 2 | 3 | 4 | 5 |

1000 value = 1 variable

(Searching)

Homogeneous

Memory size is fix.

Not dynamic in nature.

List: It is similar to array.
It can store any type of value
It is a Hetesogeneous
It memory size is not fixed.
It is Dynamic in nature.

Linked list

a | 1 | 1.5 | Rama | 20 | 4.5 |

In Python we can use list not an array.

* Searching and Sorting is easy in list.

Syntax:

int float str complex

list_name = [value1, value2 ..... valuen]

Ex: L = [10, 20, 30, 40, 50]

L | 1000 | → | 10 | 20 | 30 | 40 | 50 |
0 1 2 3 4

Print(type(L)) ⟶ <class 'list'>

Print(L)

L1 = [10, 28.5, "Rama", True]

Print(Len(L1)) ⟶ 4

L1 | 2000 | → | 10 | 28.5 | Rama | True |
0 1 2 3

Print(L1)

Print(L[1]) → 20

Print(L[4]) → 40

Print(L[-1]) → 40

Print(L[-4]) → 20

L3 = [ ] ← empty list.

Feature:

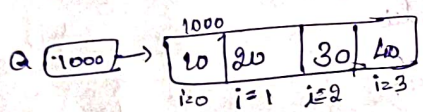
1) Hetesogeneous
2) Dynamic in nature
3) Follows the Insertion order.
4) Duplicates are allowed
5) Mutable

→ **Hetesogenous:** In the one list we can store any type of values.

② **Dynamic in nature**

```
a = [ ]
i = 0
while (True):
    Print("Enter a value")
    data = int(input())
    a.insert(i, data)
    i = i + 1
```

```
Print ("Do you wish to Continue")
Print ("Press 1 : "Yes")
Print (" Press 2 : No")
choice = int (input ())
if (choice == 1):
    Continue
else :
    break
Print(a)
```

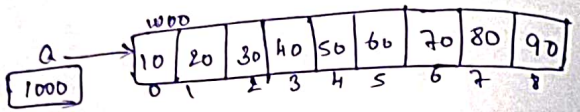


data → 10 20 30 40

choice 1, 1, 1, 2

Indexing and Slicing in list

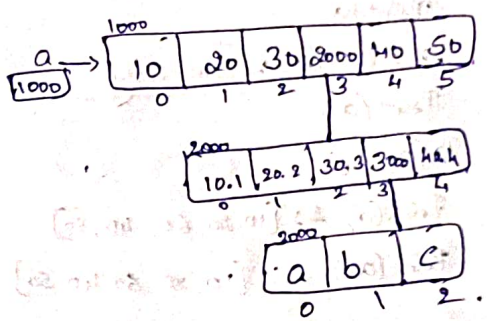
a = [10, 20, 30, 40, 50, 60, 70, 80, 90]




```
Print (len(a))
Print (a)
Print (a[1])
Print (a[4])
Print (a[-1])
Print (a[8])
Print(a[0:3])
Print (a[4:6])
Print (a[7:])
Print (a[:])
Print (a[:5])
Print (a[5:])
Print (a[::-1])
Print (a[8:4]) → No o/p
Print (a[8:4:-1])
Print (a[-3:-8]) → No o/p
Print (a[-3:-8:-1])
Print (a[-8:-3])
Print (a[:5:-1])
Print (a[5::-1])
Print (a[::2])
Print (a[0:7:0]) → Error
```

a = [10, 20, 30, [10.1, 20.2, 30.3, ['a', 'b', 'c'], 40.4], 40, 50]

```
Print (a)
Print (len(a)) -> 6
1D → Print (a[2]) → 30
2D → Print (a[3][2]) → 30.3
3D → Print (a[3][3][2] → c
```



Arithmetic Operation on list

① a = [10, 20, 30]
b = [40, 50, 60]
c = a+b
Print (c) → [10, 20, 30, 40, 50, 60]

{ → unsupported operand
}

② a = [1, 2, 3]
b = a*3
Print (b) → [1, 2, 3, 1, 2, 3, 1, 2, 3]

③ a1 = [55, 66, 77, 88]
Print (55 in a1) → True.
Print (65 in a1) → False
Print (77 not in a1) → False.
Print (80 not in a1) → True.
Print (88 in a1) → True

Copy Method

① a = [10, 20, 30, 40, 50]

Print(a)

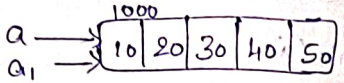
a1 = a #shallow Copy

Print(a1)

a[2] = 60

Print(a) → [10, 20, 60, 40, 50]

Print(a1) → [10, 20, 60, 40, 50]



② a = [10, 20, 30, 40, 50]

~~a1~~ = a

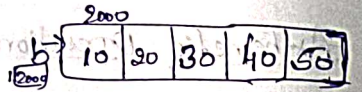
b = a.copy() #deepCopy

a[2] = 60

Print(a) → [10, 20, 60, 40, 50]

Print(a1) → [10, 20, 60, 40, 50]

Print(b) → [10, 20, 30, 40, 50]



Inbuilt Functions in List

1) All Function. all() → Give the result True when all the element in the list are non zero

2) Any function any() → Give the result True only one of the element in the list are non zero.

3) Sorted() → Give the value in the Assending value

a = [10, 20, 30, 40, 50]

Print(len(a)) → 5

Print(min(a)) → 10

Print(max(a)) → 50

Print(all(a)) → True

b = [10, 20, 0, 30, 40]

Print(all(b)) → False

Print(any(b)) → True

c = [0, 0, 0, 0]

Print(all(c)) → False

Print(any(c)) → False.

d = [22, 10, 15, 60, 25]

Print(sorted(d)) → [10, 15, 22, 25, 60]

Print(sorted(d, reverse = True)) → [60, 25, 22, 15, 10]

Inbuit Methods in list

a = [10, 20, 30, 40, 50]

Print(a)

a.insert(2, 25)

Print(a) → [10, 20, 25, 30, 40, 50]

a.append(60)

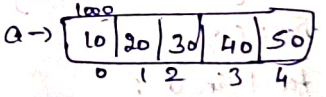
a.append(70)

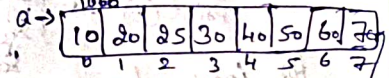
a.append(90, 100, 110) # Error

b = [90, 100, 110]

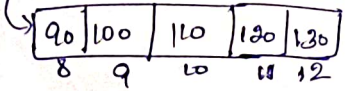
a.extend(b)

a.extend([120, 130])





In append we can add only single value.



```
a = [10, 20, 30, 40, 50, 60, 70]
Print(a)
a.remove(30)
Print(a) ⟶ [10, 20, 40, 50, 60, 70]
a.pop()  # last Remove
Print(a) ⟶ [10, 20, 40, 50, 60]
a.pop(1) # index
Print(a) → [10, 40, 50, 60]
           0   1   2   3
a.count(10)
Print(a.count(10)) ⟶ 1
Print(a.index(40)) → 1
a.clear()
Print(a) ⟶ [ ]
del a
Print(a) ⟶ "Error a is not define"
```

a →

10	20	30	40	50	60	70
0	1	2	3	4	5	6

Traversing a List :-

```
a = [10, 20, 30, 40]
Print(a)
for i in a:
    Print(i)
```

o/p

```
10
20
30
40
```

```
for index, value in enumerate(a):
    Print(index, " ", value)
```

o/p

```
0 10
1 20
2 30
3 40
```

list() :

```
a = list(range(10))
b = list(range(5, 15))
c = list(range(5, 15, 2))
Print(a)
Print(b)
Print(c)
```

o/p

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
[5, 7, 9, 11, 13]
```

list():

```
a = list(range(10))
b = list(range(5, 15))
c = list(range(5, 15, 2))
Print(a)
Print(b)
Print(c)
```

O/P

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
[5, 7, 9, 11, 13]
```

02/06

List Comprehension:-

①
```
a = [10, 20, 30, 40, 50]
b = [ ]
for data in a.:
    newdata = data + 1
    b.append(newdata)
```

Print(b) ⟶ [11, 21, 31, 41, 51]

②
```
a = [1, 2, 3, 4, 5]
b = [ ]
for data in a:
    newdata = data * data.
    b.append(newdata)
```

Print(b) ⟶ [1, 4, 9, 16, 25]

③ a = ["Krishna", "Arjuna", "Bhima"]

```
b = [ ]
for data in a:
    newdata = data.upper ( )
    b.append (newdata)
```

Print (b) → ["KRISHNA", "ARJUNA", "BHIMA"]

→ Write a program of list Comprehension to print only digits from given string.

```
a = "abc123def456"
b = [ ]
for data in a:
    if data.isdigit ( ):
        b.append (data)
```

Print (b)

```
a = "abc123def456"
b = [ ]
for data in a:
    if data.isalpha ( ):
        b.append (data)
```

Print (b)

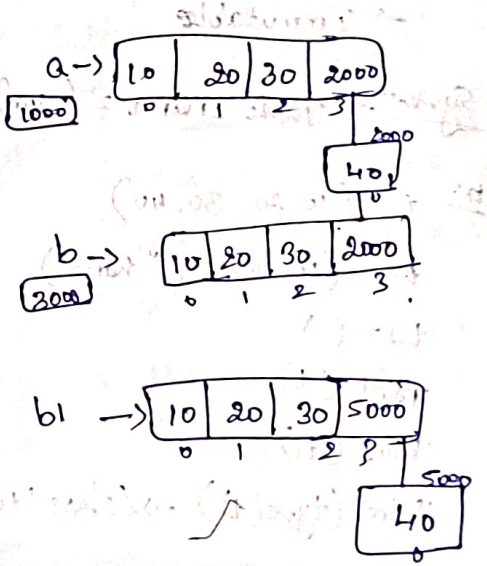
→ Write a program of list Comprehension to print only even numbers from the list.

```
a = range (5, 20)
b = [ ]
c = [ ]
for data in a:
    if (data % 2 == 0):
        b.append (data)
    else:
        c.append (data)
```

Print (b)
Print (c)

Achiving deepCopy when the list Contains reference.

```
a = [10, 20, 30, [40]]
a1 = a  # shallow copy
Print (a)
Print (a1)
a[0] = 100
a[3][0] = 400
Print (a)
Print (a1)
b = a.copy ( )
a1 = a
a[0] = 100
a[3][0] = 400
Print (a)
```



```
import copy.
a = [10, 20, 30, [40]]
b1 = copy.deepcopy(a)
a1 = a
a[0] = 100
a[3][0] = 400
Print(a)
Print(a1)
Print(b1)
```

3/6 **Tuples in Python** :- To store the multiple value

Features

- → Heterogeneous
- → Not dynamic in nature
- → Allows Duplicates
- → It follows insertion order
- → Immutable.

Syntax:- `Tuple_name = (value1, value2, ..... valuen)`

Ex:

```
t = (10, 20, 30, 40)
t1 = (10, 20.2, "Rama")
t2 = ()
Print(t)
Print(t1)
Print(type(t)) → <class 'tuple'>
t4 = (10)
Print(type(t4)) → <class 'int'>
t5 = (10,)
Print(type(t5)) → <class 'tuple'>
```

Indexing and Slicing in Tuple :-

```
a = (10, 20, 80, 40, 50, 60)
Print(a)
Print(len(a)) → 6
Print(a[1]) → 20
Print(a[3]) → 40
Print(a[5]) → 60
Print(a[-1]) → 60
Print(a[-3]) → 40
Print(a[0:2])
Print(a[3:5])
Print(a[:3])
Print(a[3:])
Print(a[::-1])
Print(a[-5:-2])
Print(a[-5:-2:-1]) → No o/p
Print(a[4:0:-1])
Print(a[0:2:-1]) → No o/p
Print(a[:])
Print(a[0:3:0]) → Error
```

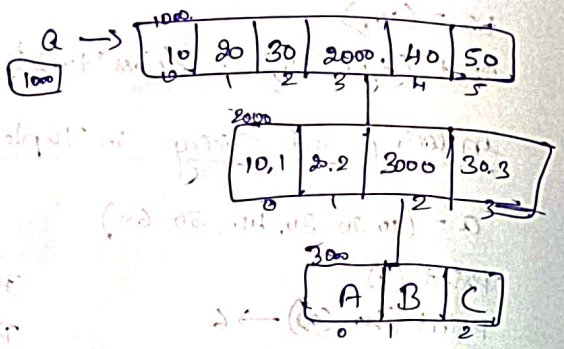
Insertion and Deletion in Tuple

```
a = (10, 20, 30, 40, 50, 60)
b = a[0:2] + (25,) + a[2:]
Print(b)
c = a[0:2] + a[3:]
Print(c)
```

List inside tuple

```
stud_info = ("Rama", [40, 50, 60])
name, ia_marks = stud_info.
Print(name)
Print(ia_marks)
```

```
a = (10, 20, 30, (10.1, 20.2, ("A", "B", "C"), 30.3), 40, 50)
Print (a)
Print (a[2])
Print (len(a))
Print (a[3][3])
Print (a[3][2][2])
Print (a[3][2])
Print (a[3])
```



04/06

* zip() : zip() function

```
names = ["Kohli", "ABD", "Dhoni", "Salt"]
runs = [9000, 8000, 5000, 4000]
Country = ["IND", "SA", "IND", "ENG"]
ipl_t = ["RCB", "RCB", "CSK", "RCB"]
c_info = list (zip (names, runs, Country, ipl_t))
Print (c_info)
```

O/P

[('Kohli', 9000, IND, RCB), ( ), ( ), ( )]

ABD Dhoni Salt

* zip_longest ( )

```
from itertools import zip_longest
names = ["Kohli", "ABD", "Dhoni", "Salt"]
runs = [9000, 8000, 5000, 4000]
Country = ["IND", "SA", "IND", "ENG"]
ipl_t = ["RCB", "RCB"]
c_info = list (zip_longest (names, runs, Country, ipl_t, fillvalue = "#"))
Print (c_info)
```

O/P

[('Kohli', 9000, 'IND', 'RCB'), ('ABD', 8000, 'SA', 'RCB'), ('Dhoni', 5000, 'IND', '#'), ('Salt', 4000, 'ENG', '#')]

→ In the middle like '5000' is removed in the end value salts runs will store in '#'

Sets In Python

* Feature
  - → Heterogeneous
  - → Dynamic in nature
  - → Mutable
  - → Doesn't follow insertion order
  - → Duplicates are not allowed

* Syntax:-

```
set_name = {value1, value2 . . . . valuen}
```

Program

```
S1 = {10, 20, 30, 40, 50}
Print (S1) → {30, 20, 40, 50, 10} → It is not following insertion order.
S2 = {10, 20, 10, 30, 20 10}
Print (S2) → {10, 30, 20} → It Connot print multiple value
S3 = { }
S4 = {10, "Rama", 20.2}
Print (S4) → {Rama, 20.2, 10}
```

```
S5 = {10, 20, 30, 40, 50}
Print(S5) → {50, 30, 10, 20, 40}
Print(len(S5)) → 5
Print(type(S5)) → <class 'set'>
Print(S2[2]) → # Error
Print(S2[0:3]) → # Error
```

→ Traversing a set:

```
a = {20, 30, 40, 50}
for i in a:
    Print(i) →  40
                50
                20
                30

for index, value in enumerate(a):        o/p
    Print(index, " ", value)             0  40
                                         1  50
                                         2  20
                                         3  30
```

→ Add and delete elements from sets

```
a = set()
for i in range(5):
    Print("Enter a value")
    data = int(input())
    a.add(data)
Print(a)
a.update([60, 70, 80])
Print(a)
Print("Enter a value to discard")
d1 = int(input())
a.discard(d1)
d2 = int(input())
a.remove(d2)
```

5/6 Subset and superset and disjoint

```
S1 = {1, 2, 3, 4}
S2 = {3, 4, 5, 6}
S3 = S1.union(S2)
Print(S3) → {1, 2, 3, 4, 5, 6}
S4 = S1.intersection(S2)
Print(S4) → {3, 4}
S5 = S1.difference(S2)
Print(S5) → {1, 2}
S6 = S2.difference(S1)
Print(S6) → {5, 6}
S7 = S1.symmetric_difference(S2)
Print(S7) → {1, 2, 5, 6}
```

---

(2)
```
S1 = {1, 2, 3, 4}
S2 = {3, 4, 5, 6}
S3 = {5, 6, 7, 8}
Print(S1.isdisjoint(S2)) → False
Print(S2.isdisjoint(S3)) → False
Print(S1.isdisjoint(S3)) → True
```

(3)
```
S1 = {1, 2, 3, 4}
S2 = {1, 2}
Print(S1.issubset(S2)) → False
Print(S1.issuperset(S2)) → True
Print(S2.issuperset(S4)) → False
Print(S2.issubset(S4)) → True
```

Frozenset

Keyword = frozenset

```
a = {10, 20, 30, 40, 50}
str1 = frozenset([10, 20, 30, 40, 50])
Print (str1) ——> frozenset ({50, 30, 20, 10, 40})
str1.add (60)
str1.discard (30)   } Error.
```

```
s1 = {10, 20, 30, [40, 50], 60}
Print (s1) → Error.  → List cannot be defined inside set
s2 = {10, 20, (30, 40), 50}
Print (s2) ——> {50, 20, 10, (30, 40)} → Tuple is allowed in set.
s3 = {10, 20, {30, 40}, 50}
Print (s3) ——> Error  Nested set is not supported
```

Dictionaries in Python

{ key : value }

Syntax:

dict_name = {key1 : value1, key2 : value2 ..... keyn : valuen,}

Example:

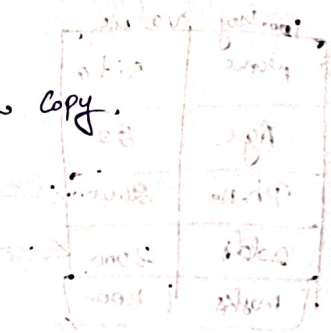
```
emp_info = {"Name" : Rama", "Age" : 25, "height" : 5.6,
            "Address" : "Mandya"}
Print (emp_info)
Print (type (emp_info)) → <class 'dict'>
Print (emp_info ['Name'])
emp_info ["Mobile"] = 1234
Print (emp_info)
```

```
student = {"Name" : "Rama", "Age" : 22, "Height" : 5.6, "Add" : "Bng"}
Print (student)
Print (student ["Age"])
for i in student:
    Print (i)
for i in student:
    Print (student [i])
for i in student.values ():
    Print (i)
for index, value in student.items ():
    Print (index, " ", values)
```

6/6

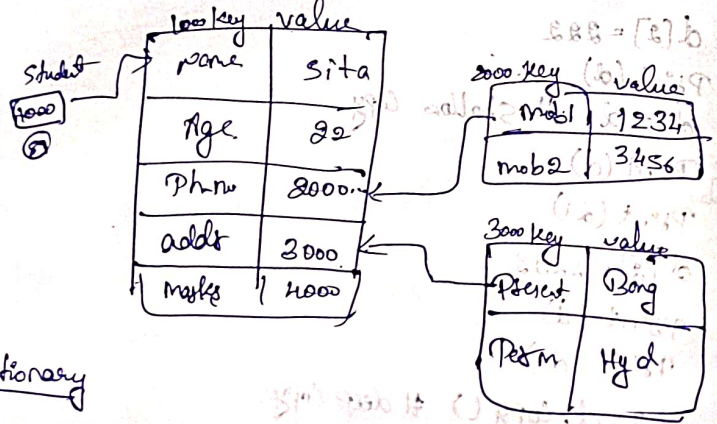
Copy Methods in Dictionary :-

```
d = {1:11, 2:22, 3:33, 4:44}
Print (d)
d[2] = 222
Print (d)
d1 = d.  # shallow copy.
Print (d)
Print (d1)
d[2] = 2222
Print (d)
Print (d1)
d2 = d.copy ()  # deep copy.
d[2] = 21
Print (d)
Print (d1)
Print (d2)
```



Add and Delete in Dictionary

```
Student = {"name": "Sita", "Age": 22, "ph_num": {"mob1": 1234,
                                                 "mob2": 3456},
           "addr": {"Present": "Bang",
                    "Perm": "Hyd"
                   }
          }
Print (Student)
Print (Student["Age"])
Print (Student["ph_num"]["mob1"])
Print (Student["addr"]["perm"])
Student["marks"] = {1:80, 2:40, 3:90, 4:100}
Print (Student)
del (Student["marks"])
Print (student)
```



Nested Dictionary

Copy methods in Dictionary

```
Student = {}  -> Same as previous
Print (Student)
S1 = Student  # shallow copy.
S1 = ["ph_num"]["mob1"] = 6666.
Print (Student["ph_num"]["mob1"]
Print (S1["ph_num"]["mob1"])
S2 = Student.copy()
S1 = ["addr"]["perm"] = "Mysore"
Print (Student["addr"]["perm"]) -> Mysore
Print (Student["addr"]["perm"]) -> Mysore
Print (S2["addr"]["perm"]) -> Mysore
```

* When the dictionary Contains the reference the deep copy also acts as the shallow copy.

→ Achiving the deepCopy. when the dictionary Contains the reference.

```
import copy.
S2 = copy.deepcopy (Student)
Print (S2["addr"]["perm"]) -> Hyd
```

Zip() on Dictionary

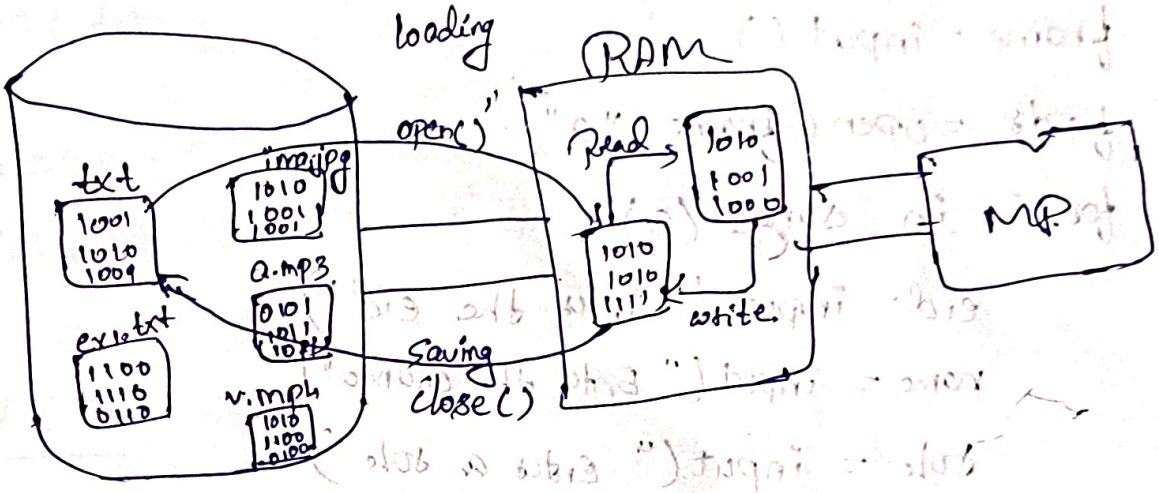
```
emp_id = [101, 102, 103, 104]
emp_name = ["Jaggu", "Raju", "Chutki", "Venky"]
emp_ph = [1234, 5678, 8765, 4321]
emp_addr = ["Tumkur", "Dubai", "Mysore", "Tirupati"]
info = list (zip (emp_name, emp_ph, emp_addr))
emp_det = dict ((zip (emp_id, info1))
Print (emp_det)
```

Core Python Continue

File Processing in Python

Types of files
1) Text file
2) Binary file.

[file Container with holds some data]



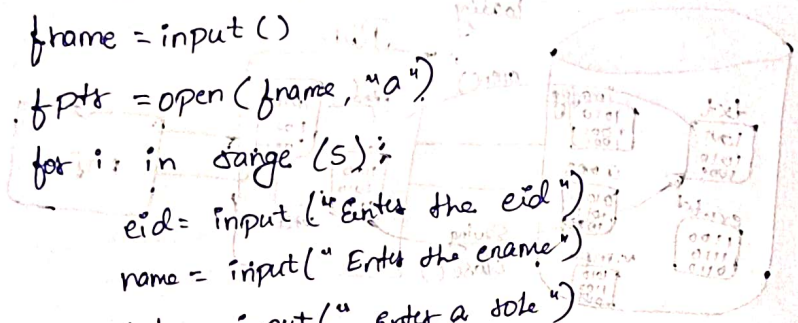
write a program to read 5 names from the keyboard and save it in a txt file.

```
→ Print("Enter a file name.")
  fname = input()
  fptr = open(fname, "w") → For create a new file.
  for i in range(5):
      Print("Enter a name")
      data = input()
      fptr.write(data + "\n")
  fptr.close()
  Print("5 names are written to file")
```

* fptr = open(fname, "a") → For rewrite the word in already exiting file. → Add the sentence or word to the same file.

write a program to Collect information of 5 employee and store all the details in a single line.

```
print ("Enter a file name")
fname = input()
fptr = open(fname, "a")
for i in range(5):
    eid = input("Enter the eid")
    name = input("Enter the ename")
    role = input("Enter a role")
    mob = input("Enter a mobile number")
    addr = input("Enter the address")
    fptr.write(eid + "\t" + name + "\t" + role + "\t" +
               mob + "\t" + addr + "\n")

fptr.close()
print("5 roles are inserted")
```



10/6

* Read() method

1) read()
2) read(bytes)
3) readline()
4) readlines()

Program(1)

```
print("Enter the file name")
fname = input()
fptr = open(fname, "r")
data1 = fptr.read()
print(data1)
```

It will print all the data in the file.

=> data2 = fptr.read(10)
print(data2)

It will print number of character based on input including space.

=> data3 = fptr.readline()
print(data3)

It will print only one line from the file.

=> data4 = fptr.readlines()
print(data4)

It will print all the data from the file in list.

* Split():

```
str1 = "Rama is eating"
print(len(str1)) --> 14
str2 = str1.split()
print(str2) --> ['Rama', 'is', 'eating']
print(len(str2)) --> 3
fptr = open("emp.txt", "r")
print(fptr.name) --> emp.txt
print(fptr.mode) --> r
print(fptr.writable()) --> False
print(fptr.readable()) --> True
print(fptr.closed()) --> False
fptr.close()
print(fptr.closed) --> True
```

Reading file (Image) in file processing

```
fptr = open("car.jpeg", "rb")
data = fptr.read()
Print(data)
```

o/p

Binary formed of the image.

```
fptr = open("car.jpg", "rb")
data = fptr.read()
fptr1 = open("newcar.jpeg", "wb")
data1 = fptr1.write(data)
fptr.close()
fptr1.close()
Print("Copied")
```

* rb : Reading the binary file (read binary)
* wb : writing the binary file (write binary)