⇒ Array:

→ it contain Similar type of data elements.

→ will not be getting this array by default we need to import module to get array.

⇒ import array.
array.array()

(or) from array import *

we can work with all the functions

instead of calling array everytime we can write it as

Ex: import array as arr.
arr.array()

Syntax:

vals = array((datatype 'i'), [5, 9, 8, 4, 2])
print(vals)

O/P :- (1630 3920, 5)
Address of Array, count/size

Methods

print(vals.reverse()) ✗ - not work like this.

print(vals.typecode)

```
vals = array('i', [5, 9, -8, 4, 2])
vals.reverse()
print(vals)
```

printing value one by one

```
vals = array("i", [1, 5, 3, 2, 4])
for i in range(5):
    print(vals[i])
```

→ it represent the index size and print.

→ variable name represent the values.

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

Taking old Array values to new Array.

```
vals = Array("i", [10, 2, 4, 5, 6])
newArray = array(vals.typecode, (a for a in vals))
```

This takes one-one value from the vals.

typecode → Type of values

```
for a in newArray:
    print(a)
```

* Using while loop

from Array import * → where we can use all the functions.

Array ~~declare~~ import ~~a 2 2d~~ if it is not default one.

Ex:

arr = array ('i', [1, 5, 6, 8, 7])

↑ data type code, it's a int

newArray = array (arr.type code, [a for a arr])

↑ Tells which type of ~~code~~ data type is this code belongs to.

↓ it used to check one by one values to print.

where

i = 0 → 'i' starts from

while i < len(newArray):

    print(newArray[i])

    i += 1

-> Empty array

arr = array ('i', [])

⇒ Functions

function is used to reduce the code where we can call any number of times

Syntax:

①
```
def greet():
    print("Hello")
    print("Good morning")
```
} Action.

```
greet()
greet()
```
} → calling function

②
```
def add(x, y):
    c = x + y
    print(c) / return c
```
used when we assigned to another variable

```
result = add(5, 4) → calling.
print(result)
```

③ working with two

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

Patterns

```
n=6
for i in range(0,n):
    for j in range(0,i+1):
        print("*", end=" ")
    print("")
```

```
0
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

```
n=6
for i in range(0,n):
    for j in range(0,i+1):
        print(i, end=" ")
    print("")
```

⇒ Prime number

```
num=7
for i in range (2,num):
    if num%i==0:
        print("it is in not prime")
else:
    print("it is prime")
```

⇒ Armstrong number

To get in row format

```
print("#", end="")
print("#", end="")
```

O/P:

```
# # #
# # #
# # #
# # #
```

```
for i in range(4):
    for j in range(4):
        print("#", end="")
    print()
```

This loop runs 16 times

① i=0,1,

② j=0,1,2,3 Again starts from 0,1,2,3,

print(#)

16 time if runs

Come out of the loop. Then go to the first line of

The code:

Object oriented programming

- it aims to implement real world entities like inheritance, polymorphisms, encapsulation, etc.

main concepts of oops

- class
- objects
- polymorphism
- Encapsulation
- Inheritance
- Data abstraction

Defining class

```
class computer:
    def config(self):
        print("i5, 16gb, 1TB")

a = "s"
com1 = computer()
print(type(com1))
```

where we can write attributes → variables
Behavior → methods (functions)
in oops we call them as methods.

com1 → object of computer class

o/p: <class 'str'> → predefined.
<class '__main__.computer'> → our class. user defined.

Ex:-

class computer : ⇒ class declaration

def config (self): → Argument.
→ method (function) declaration

print ('i5, 16gb, 1TB')

com1 = computer() → object

com2 = computer()

→ classname.method(object)

computer.config(com1) → to call the method.

computer.config(com2)

com1.config()
com2.config()
} mostly we use this method to declare object

We can write in two ways.

→ normally we call function directly using.

function name.

Ex:- config().

But in oops in call function (method) by using classname and object. because one class can contain many objects so, we have to mention the object name also. in ~~the method~~ that

Ex:- computer.config(com1).

com1 is acts as a parameter. where it is self. in program.

another behiour to call method (Function)

com1.config() → in this config will take twice

com1 has parameter

init is called it self automatically.

→ it is used to initialized the variables.

Init -- method

working with variables

Special

variables methods

--name-- --init--

class Computer:

→ it call automatically.

def __init__(self, cpu, ram):

self.cpu = cpu

self.ram = ram

object

def config(self):

print("config is", self.cpu, self.ram)

Constructor

com1 = computer('i5', 16) → object creation.

com2 = computer('Ryzen 3', 8)

↓ object.

calling methods:

com1.config()

com2.config()

- here we have to assign two variables ~~obj~~ Args with object (self)

Ex:- self.cpu = cpu

self.ram = ram

Explanation

Heare we are using cpu and ram has two arguments and we are passing three argument. here com1.config() in this config takes com1 has Argument (object, value, Ram)