**Program**

* Set of instructions to complete a certain task

**Information & Algorithm**

- Operations

- Decisions

- Iteration

**WAP to calculate two user given numbers according to given operation.**

Step1: Ask a number with user as num1

Step2: Ask operation with user as oper

Step3: Ask 2nd number with user as num2

Step4: If oper is addition

num1+num2

if oper is subtraction

num1-num2

Step5: Show output

**WAP to show multiplication table of a given number**

Step1: num = 5

Step2: do num\*i, from i=1 to i=10

**Programming Language**

C C++ C# Java PHP Python Perl

* High Level Language
* Compiler/Interpreter
* Machine Level Language (binary (0,1))

**Web Application**

1. Front

HTML CSS Bootstrap JavaScript jQuery

AJAX

React JS Vue JS

React Native, Flutter

1. Back

Python – Django/Flask

PHP – Laravel / CI

ASP

1. Data

Mysql, mssql, oracle, sqlite

**IDE (Integrated Development Environment)**

* Text Editor
* Compiler/Interpreter
* Debugger

Pycharm, VS Code

**Variables & Data types**

String

name = 'ram kumar'

Numeric(integer, float)

age = 25

salary = 50000.0

Boolean

is\_married = False

Rules for naming variable

1. Should not start with a number
2. No special characters
3. Space is not allowed, we can use underscore to connect multiple words
4. Case sensitive

Type Casting

c = float(input('Enter a number'))

d = float(input('Enter 2nd number'))

print(c+d)

**Operator**

|  |  |
| --- | --- |
| Arithmetic | **+ - \* / \*\* % //** |
| Assignment | **= += -+ \*= /= \*\*= %= //=** |
| Relational | **== < > <= >= !=** |
| Logical | **or and not(!)** |
| Concatenation | **+ ,** |
| Membership | **in not in** |
| Identity | **is is not** |

**Conditional/Decision Making Statement/Selection**

1. if …

a = 10

b = 10

if a==b:

    print('Given numbers are equal')

1. if … else …

a = 10

b = 20

if a==b:

    print('Given numbers are equal')

else:

    print('Given numbers are not equal')

1. if … elif … else …

a = 10

b = 20

if a>b:

    print('A is greater')

elif b>a:

    print('B is greater')

else:

    print('Given numbers are equal')

**Looping Statements**

1. for

for i in range(10):

    print(i)

for i in range(1, 11):

    print(i)

for i in range(1, 11, 3):

    print(i)

for i in range(10, 0, -1):

    print(i)

1. while

i = 1

while i <= 10:

    print(i)

    i += 1

i = 10

while i >= 1:

    print(i)

    i -= 1

**Collections in Python**

1. List

Indexed, ordered, changeable (mutable), allows duplicate members

students\_list = ['bigyan', 'harish', 'arpana', 'bhoomiksha', 'bigyan']

1. Tuple

Indexed, ordered, not-changeable (immutable), allows duplicate members

students\_tup = ('bigyan', 'harish', 'arpana', 'bhoomiksha', 'bigyan')

1. Set

Unindexed, unordered, not-changeable (immutable) but new elements can be added, no duplicate value

students\_set = {'bigyan', 'harish', 'arpana', 'bhoomiksha', 'bigyan'}

1. Dictionary

Indexed, Ordered, changeable, no duplicate key

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Address** | **Email** | **Phone** |
| Bigyan | ktm | [bigyan@email.com](mailto:bigyan@email.com) | 9841123456 |

student\_dict = {'name':'bigyan', 'address':'ktm', 'email':'bigyan@email.com', 'phone':9841123456}

1. Range

range(5), range(1, 6), range(5, 51, 5)

**Multidimensional Collection**

students = [

            ['ram', 'shyam', 'hari'],

            ['krishna', 'gopal', 'deepak', 'abc'],

            ['bini', 'arpana', 'bigyan']

        ]

students1 = (

            ['ram', 'shyam', 'hari'],

            ['krishna', 'gopal', 'deepak', 'abc'],

            ['bini', 'arpana', 'bigyan']

        )

students2 = [

            ('ram', 'shyam', 'hari'),

            ('krishna', 'gopal', 'deepak', 'abc'),

            ('bini', 'arpana', 'bigyan')

        ]

students3 = (

            ('ram', 'shyam', 'hari'),

            ('krishna', 'gopal', 'deepak', 'abc'),

            ('bini', 'arpana', 'bigyan')

        )

studens4 = [

    ['bigyan', 'ktm', 'bigyan@email.com', 9841123456],

    ['bini', 'lalitpur', 'bini@email.com', 9841123456],

    ['bhoomiksha', 'baneshwor', 'bhoomi@email.com', 9841123456],

    ['harish', 'ktm', 'harish@email.com', 9841123456],

    ]

students5 = [

    ['bigyan', 'bini', 'bhoomiksha', 'harish'],

    ['ktm', 'lalitpur', 'baneshwor', 'ktm'],

    ['bigyan@email.com', 'bini@email.com', 'bhoomi@email.com', 'harish@email.com'],

    [9841123456, 9841123456, 9841123456, 9841123456]

]

studens6 = [

    {'name':'bigyan', 'address':'ktm', 'email':'bigyan@email.com', 'mobile':9841123456},

    {'name':'bini', 'address':'lalitpur', 'email':'bini@email.com', 'mobile':9841123456},

    {'name':'bhoomiksha', 'address':'baneshwor', 'email':'bhoomi@email.com', 'mobile':9841123456},

    {'name':'harish', 'address':'ktm', 'email':'harish@email.com', 'mobile':9841123456},

    ]

students7 = {

    'name':['bigyan', 'bini', 'bhoomiksha', 'harish'],

    'address':['ktm', 'lalitpur', 'baneshwor', 'ktm'],

    'email':['bigyan@email.com', 'bini@email.com', 'bhoomi@email.com', 'harish@email.com'],

    'mobile':[9841123456, 9841123456, 9841123456, 9841123456]

}

**Function**

def function\_name():

    pass

1. Non parameterized

def add():

    num1 = int(input('Enter first number'))

    num2 = int(input('Enter second number'))

    print(num1, '+', num2, '=', num1 + num2)

add()

1. Parameterized

def add(num1, num2):

    print(num1, '+', num2, '=', num1 + num2)

add(5, 3)

add(10, 15)

add(20, 200)

Positional arguments & Keyword arguments

def sub\_p(num1, num2):

    print(num1, '-', num2, '=', num1 - num2)

a = int(input('Enter first number'))

b = int(input('Enter second number'))

sub\_p(a, b)

print(sub\_p(num1=20, num2=200))

1. Non return type

def add\_nr(num1, num2):

    print(num1, '+', num2, '=', num1 + num2)

1. Return type

def add\_r(num1, num2):

    return num1 + num2

**Decorator (@)**

In Python decorator (@) is used to define special methods.

1. Instance Method
2. Static Method
3. Class Method

class Triangle(Shape):

    def \_\_init\_\_(self, b, h, name, n):

        self.base = b

        self.height = h

        super().\_\_init\_\_(name, n)

    def get\_area(self):

        return 1/2 \* (self.base \* self.height)

    def get\_perimeter(self):

        pass

    @staticmethod

    def get\_area\_by\_sides(a, b, c):

        s = a+b+c

        return sqrt(s \* (s-a) \* (s-b) \* (s-c))

    @classmethod

    def get\_triangle(cls, l, b, name, n):

        return cls(l, b, name, n)

**Git – Version Control System**

* git init
* git add README.md / git add .
* git commit -m "first commit"
* git branch -M main
* git remote add origin https://github.com/mesujitg/python\_april\_04\_24\_9\_30.git
* git push -u origin main
* git config -–global user.email ‘youremail@email.com’
* git config --global user.name ‘your username’
* git status

**…or push an existing repository from the command line**

* git remote add origin https://github.com/mesujitg/python\_april\_04\_24\_9\_30.git
* git branch -M main
* git push -u origin main
* git clone https://github.com/mesujitg/python\_april\_04\_24\_9\_30.git
* git pull origin master
* git branch (shows all branches)
* git branch branch\_name (create new branch)
* git checkout branch\_name (moves to new branch)
* git merge branch\_name (merge a branch)

**OOP (Object Oriented Programming)**

**Class**

* Blueprint of an object (a real time entity)
* Collection of attributes(properties) and behavior of a real time entity i.e. variables and functions

**Object**

* Instance of class
* User defined datatype

**Features of OOP**

1. Encapsulation
2. Inheritance
3. Polymorphism
4. Abstraction

**Encapsulation**

class Student:

    # \_\_name = ''

    # \_\_address = ''

    # \_\_email = ''

    # \_\_course = ''

    # \_\_institute = ''

    def \_\_init\_\_(self):

        self.\_\_name = ''

        self.\_\_address = ''

        self.\_\_email = ''

        self.\_\_course = ''

        self.\_\_institute = 'Broadway Infosys'

    # setter

    def setValues(self, n, a, e, c):

        self.\_\_name = n

        self.\_\_address = a

        self.\_\_email = e

        self.\_\_course = c

    def setName(self, n):

        if self.\_\_name.isalpha():

            self.\_\_name = n

    def setAddress(self, a):

        self.\_\_address = a

    def setEmail(self, e):

        self.\_\_email = e

    def setCourse(self, c):

        self.\_\_course = c

    # getter

    def getName(self):

        return self.\_\_name

    def getAddress(self):

        return self.\_\_address

    def addStudent(self):

        students.append({'name': self.\_\_name, 'address': self.\_\_address,

                        'email': self.\_\_email, 'course': self.\_\_course})

    def showStudents(self):

        for s in students:

            print(students.index(s)+1, '.', s['name'], s['address'], s['email'], s['course'])

    def updateStudent(self, index):

        students[index]['name'] = self.\_\_name

        students[index]['address'] = self.\_\_address

        students[index]['email'] = self.\_\_email

        students[index]['course'] = self.\_\_course

    def deleteStudent(self, index):

        students.pop(index)

**Inheritance**

class User:

    name = ''

    address = ''

    email = ''

    mobile = ''

    username = ''

    password = ''

    role = ''

    def login():

        pass

class Student(User):

    course = ''

    def add\_student():

        pass

class Staff(User):

    salary = ''

    shift = ''

    type = ''

    department = ''

    def add\_staff():

        pass

**Polymorphism**

1. Method Overloading

class Calculate:

    def add(self, a, b, c=0):

        return a + b + c

c = Calculate()

print(c.add(10, 25))

print(c.add(10, 25, 35))

1. Method Overriding (highlighted with yellow)

|  |  |
| --- | --- |
| from abc import ABC, abstractmethod  class Shape(ABC):      def \_\_init\_\_(self, name, n):          self.name = name          self.no\_of\_sides = n      def get\_info(self):          print('Shape: ', self.name)          print('No. of Sides: ', self.no\_of\_sides)      def get\_volume(self):          return 0      @abstractmethod      def get\_area(self):          pass | class Triangle(Shape):      def \_\_init\_\_(self, b, h, name, n):          self.base = b          self.height = h          super().\_\_init\_\_(name, n)      def get\_area(self):          return 1/2 \* (self.base \* self.height) |
| class Rectangle(Shape):      def \_\_init\_\_(self, l, b, name, n):          self.length = l          self.breadth = b          super().\_\_init\_\_(name, n)      def get\_area(self):          return self.length \* self.breadth | class Square(Shape):      def \_\_init\_\_(self, l, name, n):          self.length = l          super().\_\_init\_\_(name, n)      def get\_area(self):          return self.length\*\*2 |
| class Circle(Shape):      def \_\_init\_\_(self, r, name, n):          self.radius = r          super().\_\_init\_\_(name, n)      def get\_area(self):          return (3.14 \* self.radius\*\*2) | class Cube(Shape):      def \_\_init\_\_(self, l, name, n):          self.length = l          super().\_\_init\_\_(name, n)      def get\_area(self):          return 6 \* self.length\*\*2      def get\_volume(self):          return self.length\*\*3 |

**Abstraction**

In above examples **Shape** is abstract class (it inherits ABC i.e. Abstract Base Class). **get\_area()** is abstract method which is compulsorily implemented in all child classes (highlighted with green).