**Program**

Information & Algorithm

Set of instruction to complete a certain task.

* Operations
* Decision
* Iteration

Step1: Take a number with user store as a

Step2: Take operation with user (+ or – or \* or /)

Step3: Take next number with user as b

Step4: if operation is +:

Print a+b

If operation is -:

Print a-b

If operation is \*:

Print a\*b

If operation id /:

Print a/b

**Programming Language (Java, C, C#, Python, PHP)**

* High Level Language
* Complier/Interpreter
* Binary language (0, 1)

**Web Application**

Front (UI/UX)

* HTML CSS Bootstrap JavaScript jQuery
* React JS, Vue JS
* React Native, Flutter

Back

* Python, django, flask
* PHP, laravel, CI
* ASP.net, JSP, Perl, Ruby on Rails, Node JS

Data

- mysql, sqlite, oracle, mssql

- mongodb, postgresql

**IDE (Integrated Development Environment)**

* Text Editor
* Compiler/Interpreter
* Debugger

Pycharm, VS Code

**Variable & Data types**

String

name = 'Ram Kumar'

Numeric(Integer, Float)

age = 25

salary = 50000.0

Boolean

is\_married = False

Type Casting

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

b = int(input('Enter 2nd number'))

print(a+b)

* Do not start with a number
* No special characters
* No spaces, we can use underscore instead
* Case sensitive

**Operator**

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

**Conditional/Decision Making Statements**

1. If …
2. if … else …
3. if … elif … else …

a = 10

b = 20

if a > b:

    print('A is greater than B')

elif b > a:

    print('B is greater than A')

else:

    print('A and B are equal')

**Looping Statements**

1. for

num = 5

for i in range(1, 11):

    print(num, '\*', i, '=', i\*num)

1. while

num = 5

j = 1

while j <= 10:

    print(num, '\*', j, '=', j\*num)

    j += 1

**Collection**

1. List

Indexed, ordered, changeable

students\_list = ['ram', 'shyam', 'hari', 'krish', 'john', 'shyam']

student = ['ram', 'ktm', 'ram@email.com', 98123456780]

1. Tuple

Indexed, ordered, unchangeable

students\_tuple = ('ram', 'shyam', 'hari', 'krish', 'john', 'shyam')

1. Set

unindexed, unordered, unchangeable but new value can be added

students\_set = {'ram', 'shyam', 'hari', 'krish', 'john', 'shyam'}

1. Dictionary

Indexed, ordered, changeable

student\_dict = {'name': 'ram', 'address': 'ktm', 'email': 'ram@email.com', 'mobile': 98123456780}

**Multidimensional Collection**

students1 = [

    ['ram', 'ktm', 'ram@email.com', 98123456780],

    ['shyam', 'lalitpur', 'shyam@email.com', 98123456780],

    ['hari', 'ilam', 'hari@email.com', 98123456780],

    ['krishna', 'butwal', 'krish@email.com', 98123456780],

]

students2 = (

    ['ram', 'ktm', 'ram@email.com', 98123456780],

    ['shyam', 'lalitpur', 'shyam@email.com', 98123456780],

    ['hari', 'ilam', 'hari@email.com', 98123456780],

    ['krishna', 'butwal', 'krish@email.com', 98123456780],

)

students3 = [

    ('ram', 'ktm', 'ram@email.com', 98123456780),

    ('shyam', 'lalitpur', 'shyam@email.com', 98123456780),

    ('hari', 'ilam', 'hari@email.com', 98123456780),

    ('krishna', 'butwal', 'krish@email.com', 98123456780),

]

students4 = (

    ('ram', 'ktm', 'ram@email.com', 98123456780),

    ('shyam', 'lalitpur', 'shyam@email.com', 98123456780),

    ('hari', 'ilam', 'hari@email.com', 98123456780),

    ('krishna', 'butwal', 'krish@email.com', 98123456780),

)

students5 = [

    {'name': 'ram', 'address': 'ktm', 'email': 'ram@email.com', 'mobile': 98123456780},

    {'name': 'shyam', 'address': 'lalitpur', 'email': 'shyam@email.com', 'mobile': 98123456780},

    {'name': 'hari', 'address': 'ilam', 'email': 'hari@email.com', 'mobile': 98123456780},

    {'name': 'krishna', 'address': 'butwal', 'email': 'krish@email.com', 'mobile': 98123456780}

]

students6 = {

    'name': ['ram', 'shyam', 'hari', 'krishna'],

    'address': ['ktm', 'lalitpur', 'ilam', 'butwal'],

    'email': ['ram@email.com', 'ram@email.com', 'ram@email.com', 'ram@email.com'],

    'mobile': [98123456780,98123456780,98123456780,98123456780]

}

**Function**

def function\_name():

    pass

1. Non Parameterized

def add():

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

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

    print(a, '+', b, '=', a+b)

add()

1. Parameterized

def add\_p(a, b):

    print(a, '+', b, '=', a+b)

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

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

add\_p(num1, num2)

1. Return Type

def add\_p(a, b):

    return a+b

1. Non Return Type

def add\_p(a, b):

    print(a+b)

**Decorator**

**Library modules**

Examples: math, datetime, re, random

**GIT**

* git init
* git status
* git add README.md
* git commit -m "first commit"
* git remote add origin https://github.com/mesujitg/python\_april\_8\_00.git
* git push -u origin main
* git config –-global user.email “you@example.com”
* git config –-global user.user “username”
* git branch (shows branches)
* git branch branch\_name (creates branch)
* git checkout branch\_name (moves to a branch)
* git branch -dbranch\_name (deletes branch)
* git clone <https://github.com/mesujitg/python_april_8_00.git>
* git pull origin branch\_name
* git remote -v

**OOP (Object Oriented Programming)**

**Class**

* represents a real time entity
* collection of attributes(variables) and behavior(functions/methods)
* it is blueprint of an object

**Object**

* instance of an object
* user defined data type

class Cat:

    color = ''

    height = ''

    def moves(self):

        print('Cat Walks')

    def make\_sound(self):

        print('Cat Meow')

c = Cat()

c.moves()

c.sound()

**Features of OOP**

1. Encapsulation

class Rectangle:

    \_\_length = ''

    \_\_breadth = ''

    # def \_\_init\_\_(self, l, b):

    #     self.\_\_length = l

    #     self.\_\_breadth = b

    # setter functions

    def setLength(self, l):

        self.\_\_length = l

    def setBreadth(self, b):

        self.\_\_breadth = b

    # getter functions

    # def getLength(self):

    #     return self.\_\_length

    # def getBreadth(self):

    #     return self.\_\_breadth

    def area(self):

        return (self.\_\_length \* self.\_\_breadth)

l = int(input('Enter length of Rectangle'))

b = int(input('Enter breadth of Rectangle'))

r = Rectangle()

r.setLength(l)

r.setBreadth(b)

print('Area of given Rectangle is ', r.area())

1. Inheritance

|  |  |
| --- | --- |
| class User:      name = ''      address = ''      phone = ''      email = ''      username = ''      password = ''      def login():          pass      def logout():          pass | class Student(User):      course = ''      fee = ''      def enroll():          pass      def attend():          pass      def upgrade():          pass      def pay():          pass      def leave():          pass |
| class Staff(User):      salary = ''      time = ''      def add\_staff():          pass      def update\_staff():          pass      def delete\_staff():          pass      def get\_staff():          pass |

1. Polymorphism

Method Overloading (python doesn’t support)

Method Overriding

|  |  |
| --- | --- |
| class Shape:      no\_of\_side = ''      length = ''      name = ''      def \_\_init\_\_(self, n, name):          self.no\_of\_side = n          self.name = name      def get\_perimeter(self):          return (self.no\_of\_side \* self.length)      def get\_area(self):          return 0      def get\_angle(self):          return (((self.no\_of\_side-2) \* 180)/self.no\_of\_side) | class Triangle(Shape):      base = 0      height = 0      def \_\_init\_\_(self, b, h, n, name):          self.base = b          self.height = h          super().\_\_init\_\_(n, name)      def get\_area(self):          return (1/2 \* self.base \* self.height) |
| class Rectangle(Shape):      length = ''      breadth = ''      def \_\_init\_\_(self, l, b, n, name):          self.length = l          self.breadth = b          super().\_\_init\_\_(n, name)      def get\_area(self):          return (self.length \* self.breadth) | t = Triangle(20, 10, 3, 'Triangle')  print(t.get\_area())  r = Rectangle(20, 30, 4, 'Rectangle')  print(r.get\_area()) |

1. Abstraction

|  |  |
| --- | --- |
| from abc import ABC, abstractmethod  class Shape(ABC):      no\_of\_side = ''      length = ''      name = ''      def \_\_init\_\_(self, n, name):          self.no\_of\_side = n          self.name = name      def get\_perimeter(self):          return (self.no\_of\_side \* self.length)      @abstractmethod      def get\_area(self):          pass | class Rectangle(Shape):      length = ''      breadth = ''      def \_\_init\_\_(self, l, b, n, name):          self.length = l          self.breadth = b          super().\_\_init\_\_(n, name)      def get\_area(self):          return (self.length \* self.breadth)      def get\_perimeter(self):          return (2 \* (self.length + self.breadth)) |
| class Circle(Shape):      radius = ''      def \_\_init\_\_(self, r, n, name):          self.radius = r          super().\_\_init\_\_(n, name)      def get\_area(self):          return (math.pi \* self.radius \* self.radius)      def get\_perimeter(self):          return (2 \* math.pi \* self.radius) | class Square(Shape):      length = ''      def \_\_init\_\_(self, l, n, name):          self.length = l          super().\_\_init\_\_(n, name)      def get\_area(self):          return (self.length \* self.length) |