* **Program**

Set of Instructions which may include:

* Operations
* Decisions
* Iteration etc.

Information (Data)

* **Programming Language (Java, C, C++, C#, PHP, JS, Python)**
* **Binary language (01)**
* **Compiler/Interpreter**

**IDE (Integrated Development Environment)**

* Text editor
* Compiler/interpreter
* Debugger

**Application**

* Front End (Presentation Layer)
* Back End (Business Layer)
* Database (Data Layer)

**Add two user given numbers**

Step 1: Ask a number with user & store

Step 2: Ask 2nd number with user & store

Step 3: Add two numbers

Step 4: Show output

**Variable & Datatypes**

name = "Ram"

age = 25

income = 50000.00

is\_married = False

* string
* numeric: integer, float
* Boolean

**Operators**

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

**Conditional / Decision Making Statements**

* if
* if… else…
* if… elif… else…

num1 = int(input('Enter a number: '))

num2 = int(input('Enter 2nd number: '))

if num1 == num2:

    print('Given numbers are equal')

elif num1 > num2:

    print('First number is larger')

else:

    print('Second number is larger')

**Looping Statements**

1. for loop

# for(initialization, final\_condition, increment)

for i in range(10):

    print(i)

for i in range(1, 11):

    print(i)

for i in range(1, 11, 2):

    print(i)

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

    print(i)

1. while loop

# initialization, final\_condition, increment

i = 1

while i <= 10:

    print(i)

    i += 1

**Collections in Python**

1. **List**: indexed, ordered, changeable, allows duplicate members

students\_list = ['Manish', 'Sudeep', 'Swarnim', 'Atut']

1. **Tuple**: indexed, ordered, unchangeable, allows duplicate members

students\_tuple = ('Manish', 'Sudeep', 'Swarnim', 'Atut')

1. **Set**: unindexed, unordered, unchangeable, but new members can be added, doesn’t allow duplicate members

students\_set = {'Manish', 'Sudeep', 'Swarnim', 'Atut'}

1. **Dictionary**: changeable, doesn’t allow duplicate key

employee\_dict = {'name':'Ram', 'age':25, 'income':50000.00, 'married':False}

1. **Range:** range(10)

my\_range = range(5, 50, 5)

**Multidimensional Collection**

students = [

            ['Ram', 'Shyam', 'Hari'],

            ['django', 'data science', 'AI'],

            [9812345678, 9876543211, 9845678314]

           ]

students1 = [

            ('Ram', 'django', 9812345678),

            ('Shyam', 'data science', 9812345678),

            ('Hari', 'AI', 9845678314),

            ]

students2 = [

            {'name':'Ram', 'course':'django', 'mobile':9812345678},

            {'name':'Shyam', 'course':'data science', 'mobile':9812345678},

            {'name':'Hari', 'course':'AI', 'mobile':9845678314}

            ]

students3 = {

            'name':['Ram', 'Shyam', 'Hari'],

            'course':['django', 'data science', 'AI'],

            'mobile':[9812345678, 9876543211, 9845678314]

            }

students4 = {

            1001:{'name':'Ram', 'course':'django', 'mobile':9812345678},

            1002:{'name':'Shyam', 'course':'data science', 'mobile':9812345678},

            1003:{'name':'Hari', 'course':'AI', 'mobile':9845678314}

            }

print(students2[1]['course'])

print(students3['course'][1])

print(students4[1002]['course'])

**Functions**

1. Parameterized

def checkOddEvenP(a):

    if a%2 == 0:

        print('Given number is Even')

    else:

        print('Given number is Odd')

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

checkOddEvenP(num)

1. Non – Parameterized

def checkOddEven():

    a = int(input('Enter a number'))

    if a%2 == 0:

        print('Given number is Even')

    else:

        print('Given number is Odd')

checkOddEven()

1. Return type

def addNumbers(a, b):

    return a+b

1. Non return type

def addNumbers(a, b):

    print(a+b)

**Decorators**

1. Instance method
2. Static method
3. Class method

**GIT**

* git init –> create a git repository
* git status –> shows status
* git add file\_name.ext / git add . –> add file/s to local repository
* git commit –m “message” –> commit the change with given message
* git remote add origin ‘url’ –> adds github repository
* git push origin branch\_name –> push change
* git branch –M bra \_name –> creates a branch
* git checkout branch\_name –> change branch
* git branch –> shows branch list
* git remote –v –> shows remote’s url
* git clone ‘url’
* git pull origin branch\_name

**OOP (Object Oriented programming)**

**Class**

Collection of attributes and behavior i.e. variables and functions.

It denotes a real time entity.

It defines state of an object.

It is blueprint of an object.

**Object**

Object is instance of a class.

Object may be considered as user defined data type.

**Features of OOP**

1. Encapsulation

class Account:

    \_\_name = ''

    \_\_ac\_no = ''

    \_\_balance = ''

#setter/getter

    def set\_name(self, name):

        self.\_\_name = name

    def set\_acc(self, ac):

        self.\_\_ac\_no = ac

    def set\_balance(self, balance):

        self.\_\_balance = balance

    def get\_name(self):

        return self.\_\_name

    def get\_acc(self):

        return self.\_\_ac\_no

    def get\_balance(self):

        return self.\_\_balance

1. Inheritance

|  |  |  |
| --- | --- | --- |
| class User:      name = ''      gender = ''      dob = ''      nationality = ''      address = ''      email = ''      phone = ''      username = ''      password = ''      def login(self):          pass | class Staff(User):      type = ''      shift = ''      salary = '' | class Student(User):      course = ''      reg\_id = '' |

1. Polymorphism

(Method Overriding)

|  |  |
| --- | --- |
| class Shape:      name = ''      no\_of\_side = ''      def \_\_init\_\_(self, name, sides):          self.name = name          self.no\_of\_side = sides      def area():          pass | class Triangle(Shape):      height = ''      base = ''      def \_\_init\_\_(self, name, sides, height, base):          super().\_\_init\_\_(name, sides)          self.height = height          self.base = base      def area(self):          return ((self.base\*self.height)/2) |
| class Rectangle(Shape):      length = ''      breadth = ''      def \_\_init\_\_(self, name, sides, length, breadth):          super().\_\_init\_\_(name, sides)          self.length = length          self.breadth = breadth      def area(self):          return self.length \* self.breadth | class Square(Shape):      length = ''      def \_\_init\_\_(self, name, sides, length):          super().\_\_init\_\_(name, sides)          self.length = length      def area(self):          return self.length\*\*2 |

1. Abstraction

from abc import ABC

class Shape(ABC):

    name = ''

    no\_of\_side = ''

    def \_\_init\_\_(self, name, sides):

        self.name = name

        self.no\_of\_side = sides

    def show\_info(self):

        return f'Shape: {self.name} \n No. of Sides: {self.no\_of\_side}'

    def perimeter(self, length):

        return self.no\_of\_side \* length

    @abstractmethod

    def area(self):

        pass

    @abstractmethod

    def volume(self):

        pass