Variables and Data Types

In [1]:
```
# Variable assignment
name = "Rachit"
age = 25
is_student = True

# Data types
string_var = "Hello, World!"
integer_var = 42
float_var = 3.14159
boolean_var = True
list_var = [1, 2, 3, 4, 5]
tuple_var = (1, 2, 3)
dictionary_var = {"name": "John", "age": 25}

print(f"Variable : {string_var} , data type : {type(string_var)}")
print(f"Variable : {integer_var} , data type : {type(integer_var)}")
print(f"Variable : {float_var} , data type : {type(float_var)}")
print(f"Variable : {boolean_var} , data type : {type(boolean_var)}")
print(f"Variable : {list_var} , data type : {type(list_var)}")
print(f"Variable : {tuple_var} , data type : {type(tuple_var)}")
print(f"Variable : {dictionary_var} , data type : {type(dictionary_var)}")
```

```
Variable : Hello, World! , data type : <class 'str'>
Variable : 42 , data type : <class 'int'>
Variable : 3.14159 , data type : <class 'float'>
Variable : True , data type : <class 'bool'>
Variable : [1, 2, 3, 4, 5] , data type : <class 'list'>
Variable : (1, 2, 3) , data type : <class 'tuple'>
Variable : {'name': 'John', 'age': 25} , data type : <class 'dict'>
```

Control Flow:

if elif and else

In [2]:
```
var = 10

if isinstance(var, int):
    var_type = "Integer"
elif isinstance(var, float):
    var_type = "Float"
elif isinstance(var, str):
    var_type = "String"
elif isinstance(var, bool):
    var_type = "Boolean"
else:
    var_type = "Unknown"

print(f"Variable: {var}, Type: {var_type}")
```

```
Variable: 10, Type: Integer
```

for loop

In [3]:
```
# Nested loops to create a pattern
rows = int(input("Enter the row number :"))

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

```
Enter the row number :5
*
**
***
****
*****
```

while loop

In [4]:
```
# User input validation using a while loop
password = "rachit"
input_password = input("Enter the password: ")

while input_password != password:
    print("Incorrect password. Try again.")
    input_password = input("Enter the password: ")

print("Access granted!")
```

```
Enter the password: Rachit
Incorrect password. Try again.
Enter the password: rachit
Access granted!
```

Functions

In [5]:
```
# Function definition
def greet(name):
    print("Hello, " + name + "!")

# Function call
greet("Rachit")
```

```
Hello, Rachit!
```

Decorators

In [6]:
```
# Authorization Decorator:
def check_authorization(username, password):
    name = "Rachitmore"
    pwd = "rachitmore"
    if username == name and password == pwd:
        return True
    else:
        return False

def authorization_decorator(func):
```

```
    def wrapper(username, userpassword):
        try:
            if check_authorization(username, password):
                return func(username, password)
            else:
                raise PermissionError("Unauthorized access")
        except Exception as e:
            return e
    return wrapper

@authorization_decorator
def protected_function(username, password):
    print("Access granted")

name = "Rachitmore"
password = "rachitmore"
protected_function(name, password)
```

```
Access granted
```

Lists and List Manipulation

In [7]:

```
# List creation
data_science = ["Python", "MySql", "Statistics", "Machine Learning", "Deep Learning"]

# Accessing elements
print(data_science[0])  # Output: Python

# Modifying elements
data_science[0] = "R language"
print(data_science)  # Output: ["R language", "Statistics", "Machine Learning", "Deep L

# Appending and removing elements
data_science.append("Cloud")
data_science.remove("MySql")
print(data_science)  # Output: ['R language', 'Statistics', 'Machine Learning', 'Deep L

# Slicing a list
print(data_science[1:4])  # Output: ['Statistics', 'Machine Learning', 'Deep Learning']

# Iterating over a list
for discipline in data_science:
    print(discipline)
```

```
Python
['R language', 'MySql', 'Statistics', 'Machine Learning', 'Deep Learning']
['R language', 'Statistics', 'Machine Learning', 'Deep Learning', 'Cloud']
['Statistics', 'Machine Learning', 'Deep Learning']
R language
Statistics
Machine Learning
Deep Learning
Cloud
```

Input and Output

In [8]:
```
# Accepting user input
name = input("Enter your name: ")
print("Hello, " + name + "!")

# Displaying output
age = int(input("Enter your age: "))
print("Your age is", age)
```

```
Enter your name: Rachit
Hello, Rachit!
Enter your age: 25
Your age is 25
```

File Handling

In [9]:
```
import csv
import json

# Writing and Reading txt Files
file = open("example.txt", "w")
file.write("Hello, World!")
file.close()

file = open("example.txt", "r")
content = file.read()
file.close()
print(content)

# Writing and Reading csv Files
data = [
    ['Name', 'Age', 'City'],
    ['John', '25', 'New York'],
    ['Alice', '32', 'London'],
    ['Bob', '28', 'Paris']
]

with open('data.csv', 'w', newline='') as file:
    csv_writer = csv.writer(file)
    csv_writer.writerows(data)
    file.close()

with open('data.csv', 'r') as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        print(row)
    file.close()

# Writing and Reading JSON file
data = {
    'name': 'John',
    'age': 30,
    'city': 'New York'
}

with open('data.json', 'w') as file:
    json.dump(data, file)
    file.close()
```

```
with open('data.json', 'r') as file:
    data = json.load(file)
    file.close()
    print(data)
```

```
Hello, World!
['Name', 'Age', 'City']
['John', '25', 'New York']
['Alice', '32', 'London']
['Bob', '28', 'Paris']
{'name': 'John', 'age': 30, 'city': 'New York'}
```

Exception Handling

In [10]:
```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
    print("Result:", result)
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
except ValueError:
    print("Error: Invalid input.")
```

```
Enter a number: 0
Error: Cannot divide by zero.
```

Iterators

In [11]:
```
# function definition
class NumberIterator:
    def __init__(self, limit):
        self.limit = limit
        self.current = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.current < self.limit:
            number = self.current
            self.current += 1
            return number
        else:
            raise StopIteration

# driver code
# Using the custom iterator
iterator = NumberIterator(5)
for num in iterator:
    print(num)
print("")
```

```
0
1
2
3
4
```

Generators

In [12]:
```python
# function definition
def fibonacci_generator():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

# driver code
# Using the generator
fib_gen = fibonacci_generator()
for _ in range(5):
    print(next(fib_gen))
```

```
0
1
1
2
3
```

Object-Oriented Programming (OOP)

In [13]:
```python
# Class definition
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def drive(self):
        print("Driving", self.brand, self.model)

# Object creation
my_car = Car("Tata Motors", "Nexon")

# Accessing attributes
print(my_car.brand)  # Output: Tata Motors

# Calling methods
my_car.drive()  # Output: Driving Tata Motors Nexon
```

```
Tata Motors
Driving Tata Motors Nexon
```

Inheritance

In [14]:
```python
# Parent class
class Animal:
    def __init__(self, name):
```

```
        self.name = name

    def eat(self):
        print(f"{self.name} is eating.")


# Child class inheriting from parent
class Dog(Animal):
    def __init__(self, name, breed):
        # Calling the parent class constructor
        super().__init__(name)
        self.breed = breed

    def bark(self):
        print("Woof! Woof!")

    def info(self):
        print(f"Name : {self.name} and Breed : {self.breed}")


# Creating objects
animal = Animal("Animal")
dog = Dog("Charlie", "Golden Retriever")

# Accessing parent class methods
animal.eat()

# Accessing child class methods
dog.eat()
dog.bark()
dog.info()
```

```
Animal is eating.
Charlie is eating.
Woof! Woof!
Name : Charlie and Breed : Golden Retriever
```

Encapsulation

In [15]:

```
class BankAccount:
    def __init__(self, account_number, balance = 100000):
        self._account_number = account_number
        self._balance = balance

    def deposit(self, amount):
        if amount > 0:
            self._balance += amount
            print(f"Deposited {amount}. New balance: {self._balance}")

    def withdraw(self, amount):
        if amount > 0 and amount <= self._balance:
            self._balance -= amount
            print(f"Withdrew {amount}. New balance: {self._balance}")
        else:
            print("Insufficient funds.")

    def get_balance(self):
        return self._balance
```

```
# Creating an instance of the BankAccount class
account = BankAccount("1234567890")

# Accessing methods with encapsulated attributes
balance = account.get_balance()
print("Current balance:", balance)
account.withdraw(20000)
account.deposit(50000)
```

```
Current balance: 100000
Withdrew 20000. New balance: 80000
Deposited 50000. New balance: 130000
```

Polymorphism

In [16]:
```
class Shape:
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2


class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height


# Create instances of different shapes
circle = Circle(5)
rectangle = Rectangle(4, 6)

# Call the area method on different shapes
print("Area of the circle:", circle.area())
print("Area of the rectangle:", rectangle.area())
```

```
Area of the circle: 78.5
Area of the rectangle: 24
```

Abstraction

In [17]:
```
from abc import ABC, abstractmethod

# Abstract parent class
class Shape(ABC):
    @abstractmethod
    def area(self):
        pass
```

```
# Concrete classes implementing Shape
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

# Create instances of different shapes
circle = Circle(5)
rectangle = Rectangle(4, 6)

# Call the area method on different shapes
print("Area of the circle:", circle.area())
print("Area of the rectangle:", rectangle.area())
```

```
Area of the circle: 78.5
Area of the rectangle: 24
```

Modules and Packages

In [18]:
```
# Importing modules
import math

print(math.sqrt(16))  # Output: 4.0

import mymodule
print(dir(mymodule))

mymodule.greet("Rachit")  # Output: Hello, John!
```

```
4.0
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__pack
age__', '__spec__', 'greet']
Hello Rachit
```

Working with Databases

In [21]:
```
# SQLite example
import sqlite3

# Connecting to a database
conn = sqlite3.connect("mydb.db")

# Creating a cursor object
cursor = conn.cursor()

# Executing SQL queries
```

```
cursor.execute("CREATE TABLE IF NOT EXISTS students (name TEXT, age INTEGER)")

# Inserting data
cursor.execute("INSERT INTO students VALUES (?, ?)", ("Rachit", 25))
cursor.execute("INSERT INTO students VALUES (?, ?)", ("Ankur", 25))
cursor.execute("INSERT INTO students VALUES (?, ?)", ("Jonny", 26))
cursor.execute("INSERT INTO students VALUES (?, ?)", ("Rahul", 26))
cursor.execute("INSERT INTO students VALUES (?, ?)", ("Priya", 23))

# Executing SQL queries
cursor.execute("Select * from students")

for i in cursor1:
    print(i)

# Committing the changes
conn1.commit()

# Closing the connection
conn1.close()
```

```
('Rachit', 25)
('Rachit', 25)
('Rachit', 25)
('Rachit', 25)
('Rachit', 25)
```

Regular Expressions

In [22]:

```
import re
class Validate:
    def __init__(self, username, email, phone, url, date):
        self.username = username
        self.email = email
        self.phone = phone
        self.url = url
        self.date = date
        self.data_validate()

    # Validating email addresses
    def validate_email(self):
        pattern = r"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"
        result = re.match(pattern, self.email)
        if result:
            print(f"{self.email} is valid.")
        else:
            print(f"{self.email}is invalid.")

    # Extracting phone numbers
    def validate_phone_numbers(self):
        pattern = r"\d{3}-\d{3}-\d{4}"
        result = re.findall(pattern, self.phone)
        if result:
            print(f"{self.phone} phone numbers found:", result)
        else:
            print("No phone numbers found or invalid phone number.")

    # Data validation
    def validate_username(self):
```

```
        pattern = r"^[a-zA-Z0-9_ ]+$"
        result = re.match(pattern, self.username)
        if result:
            print(f"{self.username} is valid.")
        else:
            print(f"{self.username} is invalid.")

    # Validating URLs
    def validate_url(self):
        pattern = r"^http(s)?://"
        result = re.match(pattern, self.url)
        if result:
            print(f"{self.url} is valid.")
        else:
            print(f"{self.url} is invalid.")

    # Data extraction
    def validate_dates(self):
        pattern = r"\d{1,2}[/-]\d{1,2}[/-]\d{2,4}"
        result = re.findall(pattern, self.date)
        if result:
            print(f"{self.date} Valid:")
        else:
            print(f"{self.date} invalid.")

    # Data extraction
    def data_validate(self):
        self.validate_username()
        self.validate_email()
        self.validate_phone_numbers()
        self.validate_url()
        self.validate_dates()

print("Valid details \n")
obj1 = Validate("Rachit More","rachitmore3@gmail.com","123-456-7890","https://www.examp
print("\nInvalid details \n")
obj2 = Validate("Rachit@More","rachitmore@gmail","qwerty","www.example","No dates")
```

```
Valid details

Rachit More is valid.
rachitmore3@gmail.com is valid.
123-456-7890 phone numbers found: ['123-456-7890']
https://www.examples.ai is valid.
07/05/2023 Valid:

Invalid details

Rachit@More is invalid.
rachitmore@gmailis invalid.
No phone numbers found or invalid phone number.
www.example is invalid.
No dates invalid.
```

In [ ]: