**Department of Operational Research**

University of Delhi

# Practical File - Part1

MOR105: Python Programming

***Submitted By***

Name: **Om Gupta**

Form No.: DUPG0032347

Department: North

Campus: North

Section: A

Batch: 2024-26

Master of Operational Research

***Submitted To***

Dr. Gurjeet Kaur

Department of Operational Research

# INDEX

1. a. Write a program to enter name and display as “Hello, Name”.

In [4]:
```
name = input("Enter your name: ")
print(f"Hello, {name}!")
```

```
Hello, Om!
```

1. b. WAP to create hashtag after taking two input strings from user.

> We are considering a hashtag to have the following properties:
>
> - It should be concatenation of first and last three characters of both the strings respectively.
> - It should be in lower case.
> - It should not have any spaces.

In [5]:
```
str1 = input("Enter first string: ")
str2 = input("Enter second string: ")
hashtag = (str1.replace(" ","")[:3]+str2.replace(" ","")[-3:]).lower()
print(f'Hastag created from "{str1}" and "{str2}" is "#{hashtag}"')
```

```
Hastag created from "N o R th" and "Ca MPUs" is "#norpus"
```

2. WAP to compute the roots of a quadratic equation.

In [3]:
```
a = float(input("Enter the coefficient of x\N{SUPERSCRIPT TWO}: "))
if a==0:
    print("Not a quadratic equation.\nThe coefficient of x\N{SUPERSCRIPT TWO},
          needs to be non-zero.")
else:
    b = float(input("Enter the coefficient of x: "))
    c = float(input("Enter the constant term: "))
    # Formatting the equation for maximum readability.
    print(f"Given equation is:\n{a}x\N{SUPERSCRIPT TWO}
          {"" if b==0 else str(" + " if b>0 else " - ")+str(abs(b))+"x "}
          {"" if c==0 else str("+ " if c>0 else "- ")+str(abs(c))}")
    from math import sqrt
    # Discriminant
    d = b*b-4*a*c
    if d>0:
        d = sqrt(d)
        root1 = (-b+d)/(2*a)
        root2 = (-b-d)/(2*a)
        print(f"Roots are x = {root1:2.2f}, {root2:2.2f}")
    elif d == 0:
        d = sqrt(d)
        root1 = (-b+d)/(2*a)
        print(f"Root is x = {root1:2.2f}")
    else:
        print(f"Since, discriminant ({d}) < 0\nTherefore, no real roots exist.")
```

```
Given equation is:
2.0x² - 2.0x + 2.0
Since, discriminant (-12.0) < 0
Therefore, no real roots exist.
```

3. WAP to play Stone, Paper, Scissors with Computer.

In [42]:
```
u = int(input(f"1. Stone\n2. Paper\n3. Scissors\nEnter your choice (1/2/3): "))
if u>3 or u<1:
    print("Incorrect Choice. Please choose from 1/2/3.")
else:
    from random import randint
    c = randint(1,3)
    l = ['Stone', 'Paper', 'Scissors']
    print(f"You Picked -> {l[u-1]}\nComputer Picked -> {l[c-1]}")
    print("\nResult -> ",end="")
    if u==c:
        print("Draw")
    elif (u==1 and c==3) or (u==2 and c==1) or (u==3 and c==2):
```

```
        print("You Won!")
    else:
        print("Computer Won!")
```

```
You Picked -> Paper
Computer Picked -> Stone

Result -> You Won!
```

4. Write a program for BMI Calculator with Categorization of underweight, normal weight and overweight

> BMI Categories
>
> Underweight: BMI < 18.5
>
> Normal weight: 18.5 <= BMI < 24.9
>
> Overweight: 25 <= BMI < 29.9
>
> Obesity: BMI >= 30

In [52]:

```
# Input: weight (kg) and height (m)
weight = float(input("Enter your weight in kg: "))
height = float(input("Enter your height in meters: "))

print(f"Weight: {weight}kg")
print(f"Height: {height}m")
bmi = weight / (height ** 2)

print(f"Your BMI is: {bmi:.2f}")

# Categorization
if bmi < 18.5:
    print("You are underweight.")
elif 18.5 <= bmi < 24.9:
    print("You have a normal weight.")
elif 25 <= bmi < 29.9:
    print("You are overweight.")
else:
    print("You are obese.")
```

```
Weight: 70.0kg
Height: 1.69m
Your BMI is: 24.51
You have a normal weight.
```

5. WAP to demonstrate exception handling of Zero Division Error

In [55]:

```
a = int(input("Enter Dividend: "))
b = int(input("Enter a Divisor: "))

try:
    c = a/b
    print(f"{a}/{b} = {c}")
except ZeroDivisionError as e:
    print("Division by 0 is not possible.")
```

```
Division by 0 is not possible.
```

6. WAP to demonstrate OOPs using user defined Cuboid Class

In [60]:

```
class Cuboid:
    def __init__(self, l, b, h):
        self.ln = l
        self.br = b
        self.hi = h
    def peri(self):
        """Compute perimeter of the cuboid."""
        return 4*(self.ln+self.br+self.hi)
    def vol(self):
        """Compute volume of the cuboid."""
        return self.ln*self.br*self.hi
```

```python
c1 = Cuboid(1,2,3)
perimeter = c1.peri()
volume = c1.vol()
print(f'Perimeter is {perimeter}')
print(f'Volume is {volume}')
```

```
Perimeter is 24
Volume is 6
```

7. WAP to demonstrate inheritance in OOPs using user defined Rectangle and Cuboid Class

In [59]:

```python
class Rectangle:
    def __init__(self, l, b):
        self.ln = l
        self.br = b
    def peri(self):
        """Compute perimeter of the rectangle."""
        return 2*(self.ln+self.br)
    def area(self):
        """Compute area of the rectangle."""
        return self.ln*self.br

class Cuboid(Rectangle):
    def __init__(self, l, b, h):
        super().__init__(l, b)
        self.hi = h
    def vol(self):
        """Compute volume of the cuboid."""
        return self.area()*self.hi
    def peri(self):
        """Compute perimeter of the cuboid."""
        return 4*(self.ln+self.br+self.hi)

c2 = Cuboid(3,2,3)
perimeter = c2.peri()
volume = c2.vol()
print(f'Perimeter is {perimeter}')
print(f'Volume is {volume}')
```

```
Perimeter is 32
Volume is 18
```

8. Write a Program to implement Inheritance. Create a class Employee inherit two classes Manager and Clerk from Employee.

In [65]:

```python
class Employee:
    def __init__(self, empid, name, age, salary):
        self.empid = empid
        self.name = name
        self.age = age
        self.salary = salary

    def display_info(self):
        print(f"Employee ID: {self.empid}")
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Salary: {self.salary}")

class Manager(Employee):
    def __init__(self, empid, name, age, salary, department):
        super().__init__(empid, name, age, salary)
        self.department = department

    def display_manager_info(self):
        self.display_info()
        print(f"Department: {self.department}")
        print("Position: Manager")

class Clerk(Employee):
    def __init__(self, empid, name, age, salary, experience):
        super().__init__(empid, name, age, salary)
        self.experience = experience
```

```
    def display_clerk_info(self):
        self.display_info()
        print(f"Experience: {self.experience} years")
        print("Position: Clerk")

manager = Manager(32347 ,"Rahul", 45, 120000, "HR")
clerk = Clerk(12345, "Ashish", 30, 40000, 5)

print("Manager Details:")
manager.display_manager_info()

print("\nClerk Details:")
clerk.display_clerk_info()
```

```
Manager Details:
Employee ID: 32347
Name: Rahul
Age: 45
Salary: 120000
Department: HR
Position: Manager

Clerk Details:
Employee ID: 12345
Name: Ashish
Age: 30
Salary: 40000
Experience: 5 years
Position: Clerk
```

9. WAP to demonstrate the demarcation of class variable and instance variable in OOP using Employee Class

In [73]:

```
class Employee:
    # Class variable: shared by all instances
    company_name = "Bajaj Tech"
    emp_count = 0

    def __init__(self, name, age, salary):
        # Instance variables: unique to each instance
        self.name = name
        self.age = age
        self.salary = salary
        Employee.emp_count += 1 # Accessing class variable

    def display_info(self):
        """Display information of the Employee."""
        print(f"Employee Name: {self.name}")
        print(f"Employee Age: {self.age}")
        print(f"Employee Salary: {self.salary}")
        print(f"Company: {Employee.company_name}")  # Accessing class variable


emp1 = Employee("Rahul", 45, 120000)
emp2 = Employee("Ashish", 30, 40000)

print("Employee 1 Details:")
emp1.display_info()

print("\nEmployee 2 Details:")
emp2.display_info()

# Accessing the class variable directly from the class
print(f"\nNumber of Employees (Accessed via Class): {Employee.emp_count}")

# Modifying the class variable
Employee.company_name = "New Bajaj Tech"

# Display information again after modifying the class variable
print("\nAfter changing the company name for all employees:\n")

print("Employee 1 Details:")
emp1.display_info()
```

```
print("\nEmployee 2 Details:")
emp2.display_info()
```

```
Employee 1 Details:
Employee Name: Rahul
Employee Age: 45
Employee Salary: 120000
Company: Bajaj Tech

Employee 2 Details:
Employee Name: Ashish
Employee Age: 30
Employee Salary: 40000
Company: Bajaj Tech

Number of Employees (Accessed via Class): 2

After changing the company name for all employees:

Employee 1 Details:
Employee Name: Rahul
Employee Age: 45
Employee Salary: 120000
Company: New Bajaj Tech

Employee 2 Details:
Employee Name: Ashish
Employee Age: 30
Employee Salary: 40000
Company: New Bajaj Tech
```

10. Write a Program to determine EOQ using various inventory models.

In [76]:
```
from abc import ABC, abstractmethod
from math import sqrt
class DetModels(ABC):
    @abstractmethod
    def get_quantity():
        pass
    @abstractmethod
    def get_total_cost():
        pass
    def get_cycle_time(self, Q, lam):
        return Q/lam

class EOQ(DetModels):
    def get_quantity(self, A, lam, I, C):
        return int(sqrt((2*A*lam)/(I*C)))
    def get_total_cost(self, A, lam, I, C, Q):
        holding_cost = (I * C * (Q / 2))
        ordering_cost = A * (lam / Q)
        total_cost = holding_cost + ordering_cost
        return total_cost

class EPQ(DetModels):
    def get_quantity(self, A, lam, I, C, si):
        return int(sqrt((2*A*lam*si)/(I*C*(si-lam))))
    def get_total_cost(self, A, lam, I, C, Q, si):
        holding_cost = (I * C * Q * (si - lam)) / (2 * si)
        setup_cost = A * (lam / Q)
        total_cost = holding_cost + setup_cost
        return total_cost

class EOQ_Short(DetModels):
    def get_quantity(self, A, lam, I, C, pi):
        return int(sqrt((2*A*lam*(pi + I*C))/(I*C*pi)))
    def get_total_cost(self, A, lam, I, C, Q, pi):
        Q_r = Q * (I * C) / (pi + I * C)
        holding_cost = I * C * (Q / 2)
        ordering_cost = A * (lam / Q)
        shortage_cost = pi * (Q - Q_r) / 2
        total_cost = holding_cost + ordering_cost + shortage_cost
        return total_cost
```

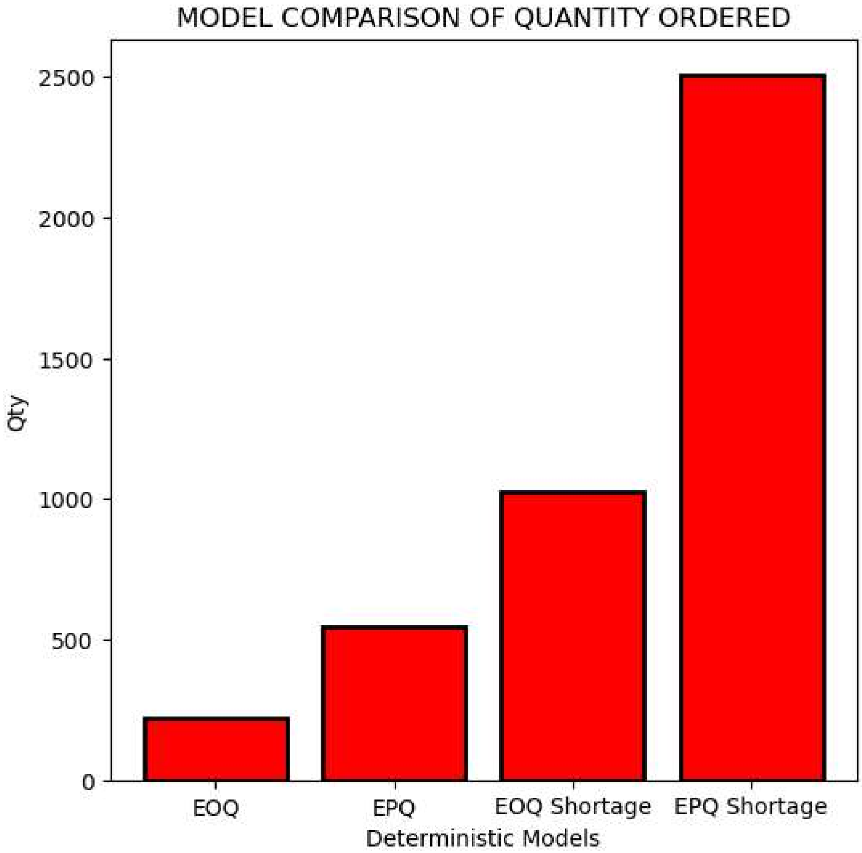
```python
class EPQ_Short(DetModels):
    def get_quantity(self, A, lam, I, C, si, pi):
        return int(sqrt((2*A*lam*(pi + I*C)*si)/(I*C*pi*(si-lam))))
    def get_total_cost(self, A, lam, I, C, Q, si, pi):
        Q_r = Q * (I * C) / (pi + I * C)
        holding_cost = (I * C * Q * (si - lam)) / (2 * si)
        setup_cost = A * (lam / Q)
        shortage_cost = pi * (Q - Q_r) / 2
        total_cost = holding_cost + setup_cost + shortage_cost
        return total_cost
```

In [79]:

```python
# Sample Run
A = float(input("Enter ordering cost per order (A): ")) # 100
lam = float(input("Enter demand rate (lambda): ")) # 10000
I = float(input("Enter inventory carrying cost rate (I): ")) # 0.2
C = float(input("Enter unit cost of item (C): ")) # 200

print("\nRunning for EOQ")
my_eoq = EOQ()
Q = my_eoq.get_quantity(A, lam, I, C)
tc = my_eoq.get_total_cost(A, lam, I, C, Q)
T = my_eoq.get_cycle_time(Q, lam)
print(f"Quantity = {Q} units")
print(f"Total Cost = ${tc:2.2f}")
print(f"Cycle Time = {T*365:2.2f} days")

si = float(input("Enter production rate (s): ")) # 12000
print("\nRunning for EPQ")
my_epq = EPQ()
Q2 = my_epq.get_quantity(A, lam, I, C, si)
tc = my_epq.get_total_cost(A, lam, I, C, Q, si)
T = my_epq.get_cycle_time(Q, lam)
print(f"Quantity = {Q2} units")
print(f"Total Cost = ${tc:2.2f}")
print(f"Cycle Time = {T*365:2.2f} days")

pi = float(input("Enter shortage cost per unit (p): ")) # 2
print("\nRunning for EOQ with Shortage")
my_eoq_short = EOQ_Short()
Q3 = my_eoq_short.get_quantity(A, lam, I, C, pi)
tc = my_eoq_short.get_total_cost(A, lam, I, C, Q, pi)
T = my_eoq_short.get_cycle_time(Q, lam)
print(f"Quantity = {Q3} units")
print(f"Total Cost = ${tc:2.2f}")
print(f"Cycle Time = {T*365:2.2f} days")

print("\nRunning for EPQ with Shortage")
my_epq_short = EPQ_Short()
Q4 = my_epq_short.get_quantity(A, lam, I, C, si, pi)
tc = my_epq_short.get_total_cost(A, lam, I, C, Q, si, pi)
T = my_epq_short.get_cycle_time(Q, lam)
print(f"Quantity = {Q4} units")
print(f"Total Cost = ${tc:2.2f}")
print(f"Cycle Time = {T*365:2.2f} days")
```

```
Running for EOQ
Quantity = 223 units
Total Cost = $8944.30
Cycle Time = 8.14 days

Running for EPQ
Quantity = 547 units
Total Cost = $5227.64
Cycle Time = 8.14 days

Running for EOQ with Shortage
Quantity = 1024 units
Total Cost = $8954.92
Cycle Time = 8.14 days

Running for EPQ with Shortage
Quantity = 2509 units
Total Cost = $5238.26
Cycle Time = 8.14 days
```

In [80]:
```python
# Visualising Q for each model
import matplotlib.pyplot as plt

l = [Q, Q2, Q3, Q4]
names = ["EOQ", "EPQ", "EOQ Shortage", "EPQ Shortage"]

plt.figure(figsize=(6,6))
plt.bar(names, l, color = 'r',edgecolor='k',linewidth=2)
plt.ylabel("Qty")
plt.xlabel("Deterministic Models")
plt.title("MODEL COMPARISON OF QUANTITY ORDERED")
plt.show()
```




11. Write a Program to determine different characteristics using various Queueing models.

In [81]:
```python
from math import factorial

class MM1:
    def __init__(self, lam, mu):
        self.lam = lam  # Arrival rate
        self.mu = mu    # Service rate
        self.rho = lam / mu  # Traffic intensity

    def get_pn(self, n):
        """ Probability of having n customers in the system """
        return (1 - self.rho) * (self.rho ** n)

    def L(self):
        """ Average number of customers in the system """
        return self.rho / (1 - self.rho)

    def Lq(self):
        """ Average number of customers in the queue """
        return (self.rho ** 2) / (1 - self.rho)

    def W(self):
        """ Average time a customer spends in the system """
        return 1 / (self.mu - self.lam)

    def Wq(self):
```

```python
        """ Average time a customer spends waiting in the queue """
        return self.rho / (self.mu - self.lam)


class MM1k:
    def __init__(self, lam, mu, k):
        self.lam = lam  # Arrival rate
        self.mu = mu    # Service rate
        self.k = k      # Capacity
        self.rho = lam / mu

    def get_p0(self):
        """ Probability of zero customers in the system """
        if self.rho == 1:
            return 1 / (self.k + 1)
        else:
            return (1 - self.rho) / (1 - self.rho ** (self.k + 1))

    def get_pn(self, n):
        """ Probability of having n customers in the system """
        p0 = self.get_p0()
        return p0 * (self.rho ** n)

    def L(self):
        """ Average number of customers in the system """
        p0 = self.get_p0()
        L = 0
        for n in range(self.k + 1):
            L += n * self.get_pn(n)
        return L

    def Lq(self):
        """ Average number of customers in the queue """
        return self.L() - self.rho

    def W(self):
        """ Average time a customer spends in the system """
        return self.L() / (self.lam * (1 - self.get_pn(self.k)))

    def Wq(self):
        """ Average time a customer spends waiting in the queue """
        return self.Lq() / (self.lam * (1 - self.get_pn(self.k)))


class MMc:
    def __init__(self, lam, mu, c):
        self.lam = lam  # Arrival rate
        self.mu = mu    # Service rate
        self.c = c      # Number of servers
        self.rho = lam / (c * mu)

    def get_p0(self):
        """ Probability of zero customers in the system """
        summation = sum((self.lam / self.mu) ** n / factorial(n) for n in range(self.c))
        last_term = (self.lam / self.mu) ** self.c / (factorial(self.c) * (1 - self.rho))
        return 1 / (summation + last_term)

    def get_pn(self, n):
        """ Probability of having n customers in the system """
        p0 = self.get_p0()
        if n < self.c:
            return p0 * (self.lam / self.mu) ** n / factorial(n)
        else:
            return p0 * (self.lam / self.mu) ** n / (factorial(self.c) * self.c ** (n - self.c))

    def Lq(self):
        """ Average number of customers in the queue """
        p0 = self.get_p0()
        return p0 * (self.rho ** self.c) * self.lam / (factorial(self.c) * (1 - self.rho) ** 2)

    def L(self):
        """ Average number of customers in the system """
        return self.Lq() + self.lam / self.mu

    def W(self):
```

```python
        """ Average time a customer spends in the system """
        return self.L() / self.lam

    def Wq(self):
        """ Average time a customer spends waiting in the queue """
        return self.Lq() / self.lam


class MMck:
    def __init__(self, lam, mu, c, k):
        self.lam = lam  # Arrival rate
        self.mu = mu    # Service rate
        self.c = c      # Number of servers
        self.k = k      # Capacity
        self.rho = lam / (c * mu)

    def get_p0(self):
        """ Probability of zero customers in the system """
        sum1 = sum((self.lam / self.mu) ** n / factorial(n) for n in range(self.c))
        sum2 = ((self.lam / self.mu) ** self.c) / factorial(self.c) * sum(
            (self.lam / (self.c * self.mu)) ** n for n in range(self.k - self.c + 1)
        )
        return 1 / (sum1 + sum2)

    def get_pn(self, n):
        """ Probability of having n customers in the system """
        p0 = self.get_p0()
        if n < self.c:
            return p0 * (self.lam / self.mu) ** n / factorial(n)
        else:
            return p0 * (self.lam / self.mu) ** n / (factorial(self.c) * self.c ** (n - self.c))

    def Lq(self):
        """ Average number of customers in the queue """
        p0 = self.get_p0()
        return p0 * (self.rho ** self.c) * self.lam / (factorial(self.c) * (1 - self.rho) ** 2)

    def L(self):
        """ Average number of customers in the system """
        return self.Lq() + self.lam / self.mu

    def W(self):
        """ Average time a customer spends in the system """
        return self.L() / self.lam

    def Wq(self):
        """ Average time a customer spends waiting in the queue """
        return self.Lq() / self.lam
```

In [83]:

```python
# Sample Run
mm1 = MM1(lam=2, mu=3)
print("Model: M/M/1")
print("L:", mm1.L())
print("Lq:", mm1.Lq())
print("W:", mm1.W())
print("Wq:", mm1.Wq())

mm1k = MM1k(lam=2, mu=3, k=5)
print("\nModel: M/M/1/k")
print("L:", mm1k.L())
print("Lq:", mm1k.Lq())
print("W:", mm1k.W())
print("Wq:", mm1k.Wq())

mmc = MMc(lam=5, mu=6, c=3)
print("\nModel: M/M/c")
print("L:", mmc.L())
print("Lq:", mmc.Lq())
print("W:", mmc.W())
print("Wq:", mmc.Wq())

mmck = MMck(lam=4, mu=5, c=2, k=10)
print("\nModel: M/M/c/k")
print("L:", mmck.L())
print("Lq:", mmck.Lq())
```

```python
print("W:", mmck.W())
print("Wq:", mmck.Wq())
```

```
Model: M/M/1
L: 1.9999999999999998
Lq: 1.333333333333333
W: 1.0
Wq: 0.6666666666666666

Model: M/M/1/k
L: 1.4225563909774435
Lq: 0.7558897243107768
W: 0.7472353870458135
Wq: 0.3970510795155344

Model: M/M/c
L: 0.848130785803916
Lq: 0.014797452470582666
W: 0.1696261571607832
Wq: 0.0029594904941165332

Model: M/M/c/k
L: 1.1809752084643919
Lq: 0.38097520846439187
W: 0.29524380211609796
Wq: 0.09524380211609797
```