KARUME DANIEL

SCT211-0072/2022

Bsc. COMPUTER SCIENCE

1. THE VOLUME OF A SPHERE

- The volume of a sphere is computed mathemtically using the formula defined below:

$$V = \frac{4}{3}\pi r^3$$

- Where π is equal to 3.14159... and r is the radius of the sphere
- The snippets of Python code below demonstrate how the volume of a sphere can be computed programatically.

In [4]:
```
# Defining a constant to hold the value of PI
PI = 3.14159
```

In [5]:
```
# Get the radius of the sphere by prompting the user
radius_of_sphere = float(input("Enter the radius of a sphere: "))
```

```
Enter the radius of a sphere: 7
```

In [6]:
```
# Compute the volume of the sphere
volume_of_sphere = 4 / 3 * (PI * radius_of_sphere ** 3)
print(f"Sphere volume: {volume_of_sphere:.2f}")
```

```
Sphere volume: 1436.75
```

2. COMPUTING COMPOUND INTEREST

- Compound interest is calculated using the formula provided below:

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

- Using this formula, it is possible to compute compound interest using a Python script. This is demonstrated in the code below.

In [7]:
```
def get_compound_interest_inputs():
    """
    Prompts the user for the inputs needed to calculate compound interest.

    Returns:
      A tuple containing the principal amount, annual interest rate, number of
      times interest is compounded per year, and number of years.
    """

    principal = float(input("Enter the principal amount: "))
    rate = float(input("Enter the annual interest rate (as a decimal): "))
    times_compounded = int(input("Enter the number of times interest is compounded per y
    years = int(input("Enter the number of years: "))
```

```
    return principal, rate, times_compounded, years

# Call the function to prompt the user for input
principal, rate, times_compounded, years = get_compound_interest_inputs()
```

```
Enter the principal amount: 100000
Enter the annual interest rate (as a decimal): 0.12
Enter the number of times interest is compounded per year: 1
Enter the number of years: 5
```

In [8]:
```
def compound_interest(principal, rate, times_compounded, years):
    """
    Calculates the compound interest earned on a principal amount.

    Args:
      principal: The initial amount invested (float).
      rate: The annual interest rate (float).
      times_compounded: The number of times interest is compounded
       per year (integer).
      years: The number of years the money is invested (integer).

    Returns:
      The future value of the investment, including interest (float).
    """

    # Calculate the annual growth factor.
    growth_factor = 1 + (rate / times_compounded)

    # Calculate the future value.
    future_value = principal * (growth_factor ** (times_compounded * years))

    return future_value

# Call the function that computes the compound interest
future_value = compound_interest(principal, rate, times_compounded, years)

# Output the compound interest
print(f"Future value after {years} years: {future_value:.2f}")
```

```
Future value after 5 years: 176234.17
```

3. ASCII CAT

- Below is a Python script that prints out a cat using ASCII characters.

In [ ]:
```
print(r"""
  |\_/|
 / @ @ \
( > º < )
 `>>x<<´
 /  O  \
""")
```

```
  |\_/|
 / @ @ \
( > º < )
 `>>x<<´
 /  O  \
```

4. SOLVING A SYSTEM OF EQUATIONS USING

GAUSS JORDAN ELIMINATION

- The system of equations below can be solved using the Gauss Jordan Elimination technique. This is demonstrated in a step wise manner below.

System of Equations:

$$\begin{aligned} x_1 + 2x_2 - x_3 &= -2 \\ 2x_1 + 7x_2 - 8x_3 &= -16 \\ -2x_2 + 2x_3 &= 2 \end{aligned}$$

Augmented Matrix:

$$\begin{bmatrix} 1 & 2 & -1 & | & -2 \\ 2 & 7 & -8 & | & -16 \\ 0 & -2 & 2 & | & 2 \end{bmatrix}$$

Step 1: Subtract 2 times the first row from the second row:

$$\begin{bmatrix} 1 & 2 & -1 & | & -2 \\ 0 & 3 & -6 & | & -12 \\ 0 & -2 & 2 & | & 2 \end{bmatrix}$$

Step 2: Add 3 times the third row to twice the second row:

$$\begin{bmatrix} 1 & 2 & -1 & | & -2 \\ 0 & 3 & -6 & | & -12 \\ 0 & 0 & -6 & | & -18 \end{bmatrix}$$

Step 3: Subtract the third row from 6 times the first row:

$$\begin{bmatrix} 6 & 12 & 0 & | & 6 \\ 0 & 3 & -6 & | & -12 \\ 0 & 0 & -6 & | & -18 \end{bmatrix}$$

Step 4: Subtract the third row from the second row:

$$\begin{bmatrix} 6 & 12 & 0 & | & 6 \\ 0 & 3 & 0 & | & 6 \\ 0 & 0 & -6 & | & -18 \end{bmatrix}$$

Step 5: Subtract 4 times the second row from the first row:

$$\begin{bmatrix} 6 & 0 & 0 & | & -18 \\ 0 & 3 & 0 & | & 6 \\ 0 & 0 & -6 & | & -18 \end{bmatrix}$$

Step 6: Multiply:

- The first row by $\frac{1}{6}$
- The second row by $\frac{1}{3}$
- The third row by $-\frac{1}{6}$

$$\begin{bmatrix} 1 & 0 & 0 & | & -3 \\ 0 & 1 & 0 & | & 2 \\ 0 & 0 & 1 & | & 3 \end{bmatrix}$$

Solution:

$$x_1 = -3$$
$$x_2 = 2$$
$$x_3 = 3$$

Below is an implementation of the Gauss Jordan Elimination Technique using Python's numpy module.

In [9]:
```
# Import numpy
import numpy as np
```

In [10]:
```
# Define the coefficient matrix and the constant vector
A = np.array([[1, 2, -1],
              [2, 7, -8],
              [0, -2, 2]])
b = np.array([-2, -16, 2])
```

In [11]:
```
# Combine the coefficient matrix and the constant vector into an augmented matrix
augmented_matrix = np.column_stack((A, b))
```

In [12]:
```
# Solve the augmented matrix
solution = np.linalg.solve(A, b)
```

In [13]:
```
# Print the solutions
x1, x2, x3 = solution
print("x1 =", x1)
print("x2 =", x2)
print("x3 =", x3)
```

```
x1 = -3.0
x2 = 2.0
x3 = 3.0
```