1(a).To generate a python code for SVM algorithm

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

# Load dataset
X, y = datasets.load_iris(return_X_y=True)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Train the SVM model
model = SVC(kernel='linear').fit(X_train, y_train)

# Print accuracy
print(model.score(X_test, y_test))
```

Output

0.9555555555555556

1(b).Design a python code to check leap year

```
year = 2024
print(year % 4 == 0 and (year % 100 != 0 or year % 400 == 0))
```

Output

True

2(a).implement KNN algorithm to classify the dataset using python

```
import numpy as np

# Example dataset
X_train = np.array([[1, 2], [2, 3], [3, 4], [6, 7], [7, 8], [8, 9]])
y_train = np.array([0, 0, 0, 1, 1, 1])
X_test = np.array([[5, 5], [1, 1]])

# KNN algorithm
def knn_predict(X_train, y_train, X_test, k=3):
    distances = np.linalg.norm(X_train[:, np.newaxis] - X_test, axis=2)
    nearest_indices = np.argsort(distances, axis=0)[:k]
    nearest_labels = y_train[nearest_indices]
    predictions = np.array([np.bincount(labels).argmax() for labels in nearest_labels])
```

```
    return predictions

# Predictions
predictions = knn_predict(X_train, y_train, X_test)
print("Predictions:", predictions)
```

Output

```
Predictions: [0 0 0]
```

2(b).write a python program tom find the factorial of a number

```
def factorial(n):
    return 1 if n == 0 else n * factorial(n - 1)

# Example usage
number = 5
print("Factorial of", number, "is:", factorial(number))
```

Output

```
Factorial of 5 is: 120
```

3(a).generate a python code for k-means algorithm

```
import numpy as np

def k_means(X, k):
    centroids = X[np.random.choice(len(X), k, replace=False)]
    while True:
        clusters = [[] for _ in range(k)]
        for x in X:
            clusters[np.argmin(np.linalg.norm(x - centroids, axis=1))].append(x)
        new_centroids = np.array([np.mean(cluster, axis=0) for cluster in clusters])
        if np.allclose(centroids, new_centroids):
            return centroids
        centroids = new_centroids

# Example usage
X = np.array([[1, 2], [1, 1], [2, 2], [8, 7], [8, 8], [9, 8]])
k = 2
centroids = k_means(X, k)
print("Centroids:", centroids)
```

Output

```
Centroids: [[8.33333333 7.66666667]
 [1.33333333 1.66666667]]
```

3(b).write a python program to print the fibonacci sequence

```
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        print(a, end=" ")
        a, b = b, a + b

# Example usage
fibonacci(10)
```

Output
0 1 1 2 3 5 8 13 21 34

4(a).to implement maximum margin classifier algorithm using python

```
from sklearn import datasets
from sklearn.svm import SVC

# Load dataset
X, y = datasets.load_iris(return_X_y=True)
X, y = X[y != 0], y[y != 0]  # Consider only binary classification

# Train Maximum Margin Classifier
clf = SVC(kernel='linear')
clf.fit(X, y)

# Print coefficients and intercept
print("Coefficients:", clf.coef_)
print("Intercept:", clf.intercept_)
```

Output
Coefficients: [[-0.59549776 -0.9739003 2.03099958 2.00630267]]
Intercept: [-6.78097119]

4(b). Develop a python code to check armstrong number

```
def is_armstrong(num):
    return num == sum(int(x) ** len(str(num)) for x in str(num))

# Example usage
number = 153
print(f"{number} is Armstrong:", is_armstrong(number))
```

Output
153 is Armstrong: True

5(a). To generate python code for random forest algorithm

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load dataset
X, y = load_iris(return_X_y=True)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predict on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

Output

```
Accuracy: 1.00
```

5(b). Write python program to make a simple calculator

```
 def calculator(num1, operator, num2):
    if operator == '+':
        return num1 + num2
    elif operator == '-':
        return num1 - num2
    elif operator == '*':
        return num1 * num2
    elif operator == '/':
        if num2 != 0:
            return num1 / num2
        else:
            return "Cannot divide by zero!"
    else:
        return "Invalid operator!"

# Example usage
num1 = 10
operator = '/'
```

```
num2 = 5
result = calculator(num1, operator, num2)
print("Result:", result)
```

Output
Result: 2.0

6(a). Develop a program for perceptron algorithm using python

```
import numpy as np

class Perceptron:
    def __init__(self, lr=0.01, n_iters=100):
        self.lr, self.n_iters = lr, n_iters
    def fit(self, X, y):
        self.weights, self.bias = np.zeros(X.shape[1]), 0
        for _ in range(self.n_iters):
            for x_i, y_i in zip(X, y):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = np.where(linear_output >= 0, 1, 0)
                update = self.lr * (y_i - y_predicted)
                self.weights += update * x_i
                self.bias += update
    def predict(self, X): return np.where(np.dot(X, self.weights) + self.bias >= 0, 1, 0)

# Example usage
X = np.array([[2, 3], [3, 3], [3, 4], [5, 6], [6, 6], [6, 7]])
y = np.array([0, 0, 0, 1, 1, 1])

perceptron = Perceptron()
perceptron.fit(X, y)
predictions = perceptron.predict(X)
print("Predictions:", predictions)
```

Output
Predictions: [0 0 0 1 1 1]

6(b). To design a python program to display calendar

```
import calendar

def display_calendar(year, month):
    print(calendar.month(year, month))

# Example usage
year = 2024
month = 6
display_calendar(year, month)
```

Output

```
    June 2024
Mo Tu We Th Fr Sa Su
             1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

7(a). Generate a python code gradient descent algorithm

```
import numpy as np

# Gradient Descent function
def gradient_descent(X, y, lr=0.01, n_iters=100):
    n_samples, n_features = X.shape
    weights = np.zeros(n_features)
    bias = 0

    for _ in range(n_iters):
        linear_model = np.dot(X, weights) + bias
        dw = (1 / n_samples) * np.dot(X.T, (linear_model - y))
        db = (1 / n_samples) * np.sum(linear_model - y)
        weights -= lr * dw
        bias -= lr * db

    return weights, bias

# Example usage
X = np.array([[1], [2], [3], [4]])
y = np.array([2, 4, 6, 8])

weights, bias = gradient_descent(X, y)
print("Weights:", weights)
print("Bias:", bias)
```

Output

```
Weights: [1.8211826]
Bias: 0.5247844642513884
```

7(b). To write a python program to find sum of natural numbers using recursion

```
def sum_of_natural_numbers(n):
    return n + sum_of_natural_numbers(n - 1) if n > 0 else 0

# Example usage
number = 5
```

```
print("Sum of natural numbers up to", number, "is:", sum_of_natural_numbers(number))
```

Output

```
Sum of natural numbers up to 5 is: 15
```