```
# Required Libraries
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt


# Load the Iris dataset
iris = load_iris()
data = iris.data
print(data)
target = iris.target
print(iris.target)

#create list to hold WSS/SSE/inertia values for each k
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=1, n_init=10)
    kmeans.fit(data)
    sse.append(kmeans.inertia_)

#visualize results
plt.plot(range(1, 11), sse)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()


# Perform K-means clustering and take cluster centers
kmeans_model = KMeans(n_clusters=3, random_state=1, n_init=10).fit(data)
labels = kmeans_model.labels_
clusters_sklearn = kmeans_model.cluster_centers_
print(labels)
print(kmeans_model.cluster_centers_)

# Function to plot final clusters
def plot_clusters_sklearn(data, labels, clusters):
 # Plot data points, choosing first two features for visualization
 plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis', label='Data points')
 # Plot cluster centers, also focusing on the same two features
 plt.scatter(clusters[:, 0], clusters[:, 1], s=200, color='red', marker='X', label='Centers')
 plt.title('Visualizing Clusters with Matplotlib using Iris Dataset')
 plt.xlabel('Sepal Length (cm)')
 plt.ylabel('Sepal Width (cm)')
 plt.legend()
 plt.grid(True)
 plt.show()

# Visualize the clusters
plot_clusters_sklearn(data, labels, clusters_sklearn)
```

Exception Handling

1. Write a python code to show ZeroDivisionError.
2. Write a python code to show IndexError.
3. Write a python code to run a certain block of code if the code block inside try runs without any errors.
4. Write a python code to run a certain block of code no matter whether there is an exception or not.
5. Write a python code to raise Exceptions (Python's built-in Exception objects) by yourself to indicate that something went wrong.
6. Write a python code to define custom exceptions by creating a new class that is derived from the built-in Exception class.
7. You're going to write an interactive calculator! User input is assumed to be a formula that consist of a number, an operator (at least + and -), and another number, separated by white space (e.g. 1 + 1). Split user input using str.split(), and check whether the resulting list is valid:

   ✓ If the input does not consist of 3 elements, raise a FormulaError, which is a custom Exception.

   ✓ Try to convert the first and third input to a float (like so: float_value = float(str_value)). Catch any ValueError that occurs, and instead raise a FormulaError

   ✓ If the second input is not '+' or '-', again raise a FormulaError

   If the input is valid, perform the calculation and print out the result. The user is then prompted to provide new input, and so on, until the user enters quit.

   An interaction could look like this:

   ```
   >>> 1 + 1
   2.0
   >>> 3.2 - 1.5
   1.7000000000000002
   >>> quit
   ```