Iris Dataset

- Install the `scikit-learn` package. (Anaconda users will have it pre-installed)
- Thereafter `import sklearn.datasets` to import the datasets submodule, and call the function datasets.load_iris() (https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html#sklearn.datasets.load_iris)

```
import sklearn.datasets
sklearn.datasets.load_iris()
```

In [25]:

```
import sklearn.datasets
dictionary=sklearn.datasets.load_iris()
data_array=dictionary["data"]
```

- The `load_iris()` function returns a dictionary containing various info about the Iris Dataset.
- The Iris Dataset is a dataset of 150 examples of the Iris flowering plant (https://en.wikipedia.org/wiki/Iris_(plant)).
- Each example consists of 4 numbers : sepal length (cm), sepal width (cm), petal length (cm) and petal width(cm). All the examples can be found as a numpy array at the key "data". (Can you guess the shape of the array?)
- Each example belongs to 1 out of 3 classes/categories which correspond to a sub-species of Iris - Iris-Setosa, Iris-Versicolour and Iris-Virginica. This information can be found as a numpy array at the key "target". (Can you guess the shape of the array?)
- The first 50 points are Iris-Setosa, next 50 Iris-Versicolour and last 50 Iris-Virginica.

- Submit the solutions to the next questions as a single file called `iris.py` .
- Write all the code sequentially, but use comments so that it is easy to check.

1. The `data_array` is cleanly divided in groups of 50 (0...49,50...99,100...149) . Take the first 40 points of each class and create a new array called `train_data` (first 40 points belong to class 0, next to class 1 etc)
2. Similarly put the remaining 10 points of each class are put in another dictionary called `test_data` .

---

6. Write a function `classify(train_data,test_data)` that predicts the class of a point from the `test_data` using the following logic.
    - Calculate the distance (according to this formula (https://en.wikipedia.org/wiki/Euclidean_distance#Definition)) of a `test_data` point with all the training data points.
    - The class predicted is the majority class of the closest (i.e. smallest distance) 30 points of the train_data. That is, for a test_data point A, if (out of the 120 train_data points) the closest 30 points

contains 15 of class 1 , 10 of class 0 and 5 of class 2 - then classifiy predicts the class of A to be 1 (the majority in top 30).
    - `classify` should return a 1-d array of 30 numbers (with value 0, 1 or 2).
7. Compute the accuracy of `classify` , by counting what percentage of predictions are correct (Remember the correct predictions are 0 for 0-9, 1 for 10-19, 2 for 20-29). Print the accuracy.

---

3. Plot the `train_data` using the `scatter()` function of `matplotlib` . Plot only the `0,1` columns as x and y respectively. All three classes (40 points each) should have different colours.
4. Generalize the above code to create 4C2=6 plots by choosing different columns of the data as x and y (ex : 0,1 ; 0,2; 2,3 etc - total 6 combinations.)
5. If possible generalize the above code to use the `subplots()` function so that all 6 plots can be neatly arranged.