CLASSES

- A recipe for creating objects,
- A structure to hold information,
- Many objects can be created from same recipe,
- Consists of methods and attributes.

`__init__`

- A methods that is called when the object is created,
- Its arguments are used to create the object,
- Usually sets a number of attributes.

METHODS

- Functions that are associated with an object,
- Called with '.',
- Can call other methods and attributes,
- First argument must be `self`,
- Can return something or change the object.

`self`

- A way of accessing information associated with the object.

```python
import numpy as np
import random

class AdvertisingCampaign:
    """
    A Class to create an AdvertisingCampaign object.
    Holds information about the cities and their adjacency matrix.
    Contains methods to minimise the number of cities required to cover
    the set.
    """
    def __init__(self, cities, adj_matrix):
        """
        Initialises the object.
        """
        self.cities = cities
        self.number_of_cities = len(self.cities)
        self.adj_matrix = adj_matrix
        self.best_score = len(self.cities)
        self.best_solution = np.array([1] *  self.number_of_cities)
        self.num_broadcasts_to_try = np.linalg.matrix_rank(self.adj_matrix)

    def evalutate_solution(self, solution):
        """
        Gives a score to a potential solution.
        If solution leaves any city out, returns self.number_of_cities,
        otherwise it returns the number of cities used for broadcasts.
        """
        coverage = np.matmul(solution, self.adj_matrix)
        if 0 in coverage:
            return self.number_of_cities
        return sum(solution)

    def new_solution(self):
        """
        Randomly generate a new potential solution
        with self.number_broadcasts_to_try broadcasts.
        """
        number_empty = self.number_of_cities - self.num_broadcasts_to_try
        sol = [1] * self.num_broadcasts_to_try + [0] * number_empty
        random.shuffle(sol)
        return np.array(sol)

    def optimise(self, num_itrs):
        """
        For num_itrs iterations, keep generating random potential
        solutions with self.number_broadcasts_to_try broadcasts. If solution
        is valid, reduce the number of broadcasts to try by 1. Keep track
        of best solution.
        """
        for iteration in range(num_itrs):
            solution = self.new_solution()
            score = self.evalutate_solution(solution)
            if score <= self.best_score:
                self.best_solution = solution
                self.best_score = score
                self.num_broadcasts_to_try = self.best_score - 1

    def print_solution(self):
        """
        Prints out the best solution.
        """
        for i, city in enumerate(self.cities):
            if self.best_solution[i] == 1:
                print(self.cities[i])



with open('french_cities.txt', 'r') as f:
    cities = f.read()
    cities_list = cities.split('\n')
adjacency_matrix = np.genfromtxt('french_distances.csv', delimiter=',')


random.seed(0)
R = AdvertisingCampaign(cities_list, adjacency_matrix)
R.optimise(10000)
R.print_solution()
```

ATTRIBUTES

- Variables that are associated with the object,
- Accessed with '.',
- Can be changes or created by methods,
- Set or changed with `self`.

LIBRARIES

- Packaged, pre-written code,
- Must be imported with `import`,
- A wide variety available.

FILES

- Can load external data,
- Many different ways of reading these.

OTHER?

`with`

`enumerate`

Seeds