MBTA exploration

For the assignment, we provide two main exectuable files:

- `metromap.py`, which contains source code and prints answers when executed,
- `tests.py`, which provides test classes and executes tests when called interactively.

The code *should* not have any external dependencies, but it might require python3 due to `urllib`.

Question 1

For the first part of the assignment, we use the `Map` class, which we designed to interact with the MBTA API. Among its attributes, `Map` accounts for `routes`, a dictionary of all the subway routes of interest. The method `get_routes()` composes the API request for `routes` through the `get()` method, which also accepts arguments for the API filters. By default, `get_routes()` downloads the routes of type `0` and `1`, as requested by the assignment.

Answer to Question 1 from terminal output:

```
====================================
=========QUESTION 1==================
====================================
Mattapan Trolley
Green Line E
Orange Line
Red Line
Green Line D
Green Line C
Blue Line
Green Line B
```

Question 2

For the second question, we are required to explore the stops located along each subway route. We then defined the `Stop` class to store subway stop information. `Stop` instances have a `parents` attribute, which stores the set of route(s) traveling through a `Stop` instance.

Interaction with the API for subway stops occurs through the `get_stops()` method of the `Map` class. `get_stops()` downloads information for all the subway stops located along each of the downloaded routes.

Two helper methods, `max_stops()` and `min_stops()` find the routes with the maximum and minimum number of stops, as requested by the assignment. Answer to the first part of Question 2 from terminal output:

```
====================================
=========QUESTION 2a=================
====================================
Route Green-B has the maximum number of stops: 24
Route Mattapan has the minimum number of stops: 8
```

A third helper method, `print_connecting_stops()`, outputs to terminal the subway stops that are crossed by at least two subway routes (i.e., those with a `parents` set with more than one element), as requested by the assignment.

Answer to the second part of Question 2 from terminal output:

```
====================================
=========QUESTION 2b=================
====================================
Ashmont --> Mattapan, Red
Boylston --> Green-D, Green-E, Green-C, Green-B
Haymarket --> Green-E, Orange, Green-C
North Station --> Green-E, Orange, Green-C
Park Street --> Green-D, Green-E, Red, Green-C, Green-B
Hynes Convention Center --> Green-D, Green-C, Green-B
Arlington --> Green-D, Green-E, Green-C, Green-B
Copley --> Green-D, Green-E, Green-C, Green-B
Kenmore --> Green-D, Green-C, Green-B
Government Center --> Blue, Green-E, Green-C, Green-D
Downtown Crossing --> Red, Orange
State --> Blue, Orange
```

Question 3

For the final part of the assignment, we are requested to find a possible way for a subway user to travel from a given starting stop to a given destination stop. The method `calc_route_adjacency()` reformulates the information on stops connecting multiple subway lines in the form of subway adjacency. In other words, the `neighbors` attribute of each `route` instance is filled with subway lines that are accessible with at most one trip (i.e., from start stop to connecting stop). The method `trip_between_stops()` uses the adjacency information to progressively "visit" all subway routes that are reachable from the starting stop. The code makes sure that all subway lines are "visited" only once. The code also stores the path followed to reach each of the lines, which can be then rendered "backwards" to answer the assignment. Example output from terminal for Question 3:

```
====================================
=========QUESTION 3==================
====================================
Brookline Village, Sullivan Square --> Green-D, Green-E, Orange
```