# Syntax

## Documentation

Good documentation of sections in functions and methods.

## Adherence to conventions

Your codebase is mostly complaint with PEP8. Good job!

## Organisation of code (abstraction)

Good use of abstraction made for easier code reading.

Some classes carry too many responsibilities, e.g. WebInterface besides encapsulating data for display, also ends up being responsible for processing data for display. That should be delegated to another function/class (e.g. an Adapter class)

Code would be made easier to overview and refactor by abstracting request-handling into a standard data structure/class.

e.g. in main.py:

...

# PathInfoRequest is subclassed from BaseRequest.

# A PathInfoRequest instance encapsulates parameters for a request

# (for detailed information on a path.

req = BusInfoRequest.from\_request(request)

if req.isempty():

return redirect("/")

# sort\_paths only accepts a valid Request object (subclassed from BaseRequest).

result = sort\_paths(request)

ui.update\_from\_result(result)

...

# 

# Object-Oriented Programming

## Encapsulation

Code is well organised into classes with specific responsibilities.

### Improvement: encapsulate bus stop data, and bus route data in BusStop and BusRoute classes

Would eliminate the need for bus\_utils.py and make the code more understandable.

Suggested UML for encapsulating bus stop, bus route, and bus operator data:

| BusStop | BusRoute |
| --- | --- |
| +busStopCode: str  +roadName: str  +description: str  +coordinate: tuple(lat, lon) | +serviceNo: str  +direction: int  -busStops: list |
| +distanceFromPoint(lat, lon)  +fromRecord() [classmethod]  +toRecord()  +isMrtStop() | +fromRecords()  +toRecords()  +hasBusStop() |

## Inheritance & Polymorphism

Not used in code.

### Improvement: replace code-branching with polymorphism

if-checks are scattered throughout the code and make it harder to understand.

Where multiple complex cases need to be handled, consider refactoring to abstract classes (Request, Result, Handler, ...) and use polymorphism for easier understanding.

e.g. for handling various kinds of requests:

class Handler:

'''

Base class for a Handler. Handlers take in a Request object and return a Result object.

Handler subclasses must implement the following interface:

Methods:

process(self, req: Request) -> Result:

Takes in req, a Request object.

Returns a Result object.

'''

def process(req: Request):

# This method must be overridden by subclasses

raise NotImplementedError

class SortPathHandler(Handler):

def process(self, req: Request) -> Result:

'''req.is\_valid() should be called in main before being passed to Handler'''

data = ...

return Result.from\_dict(data) # Result class here will replace WebInterface

in main.py:

...

req = ...

handler = SortPathHandler()

result = handler.process(req) # This line will be identical across different routes.

# future extensions/additions of handlers will be easier to implement.

ui.update\_from\_result(result)

...

# 

# Algorithms and Data Structures

## Use of sorting and searching algorithms/data structures

No standard algorithms implemented. Good use of a graph data structure.

## Performance of implementation

App does not appear to carry out unnecessary steps.

# Design

## Data modelling techniques

Some use of data modelling helped to organise the code.

Lack of relation arrows makes it difficult to understand the class hierarchy and flow of logic in the program.

## ER modelling techniques for data

Appropriate ER model used.

Bus stops should have a many-to-many relationship with bus routes, since routes consist of multiple stops, and each stop can be on multiple routes.

## Usability

Not able to run app because of missing dependencies.

App design keeps the functionality clear.

Error messages provided for erroneous input.

User choices restrained for useability. Can improve by helping user select defaults for each request.

# Web Programming

## HTTP methods and HTML forms

Appropriate use of HTML form elements and HTTP methods.

## HTML, formatting and CSS

Appropriate use of HTML page elements.

# Databases

## Data verification and data validation

Validation is carried out before being used in a query.

## Data security

Placeholders used appropriately in queries to SQLite database.