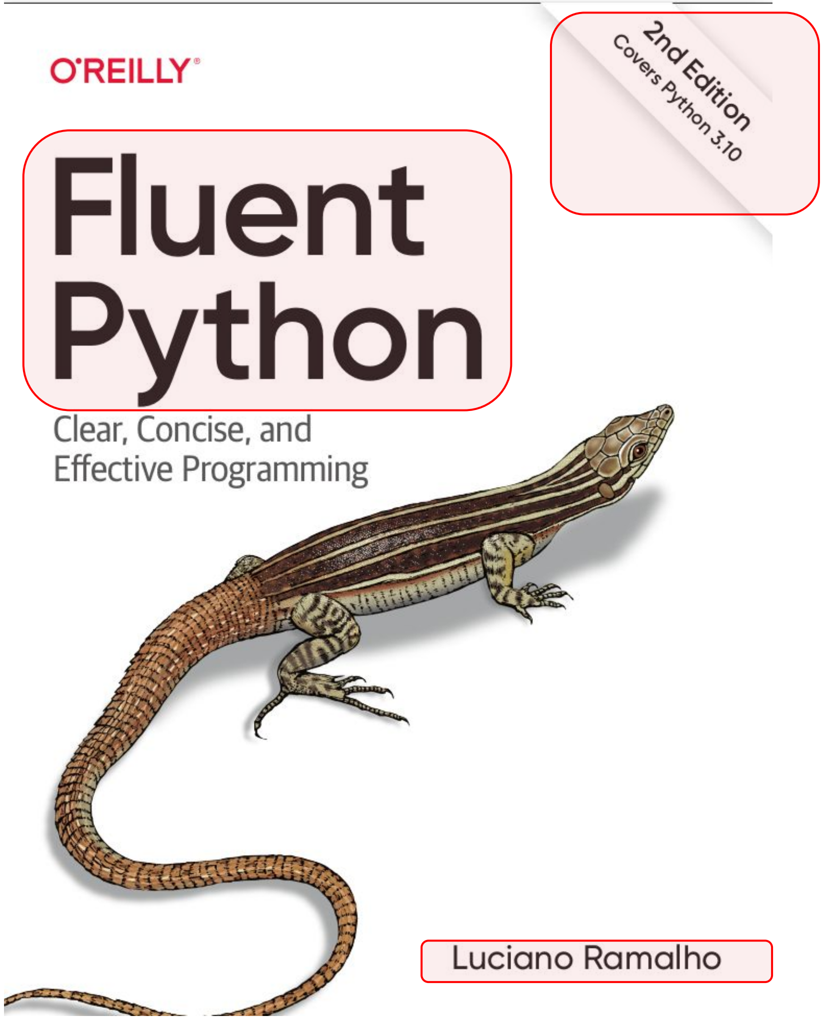
Advanced Applied Python

Typed Python, Interfaces, ABC and @abstractmethod, Profiling and Logging, Generators and list comprehensions , @staticmethod, @property, @property.setter, @classmethod, Use global variables in configuration files, Python Standard Library, Python/General OOP practices



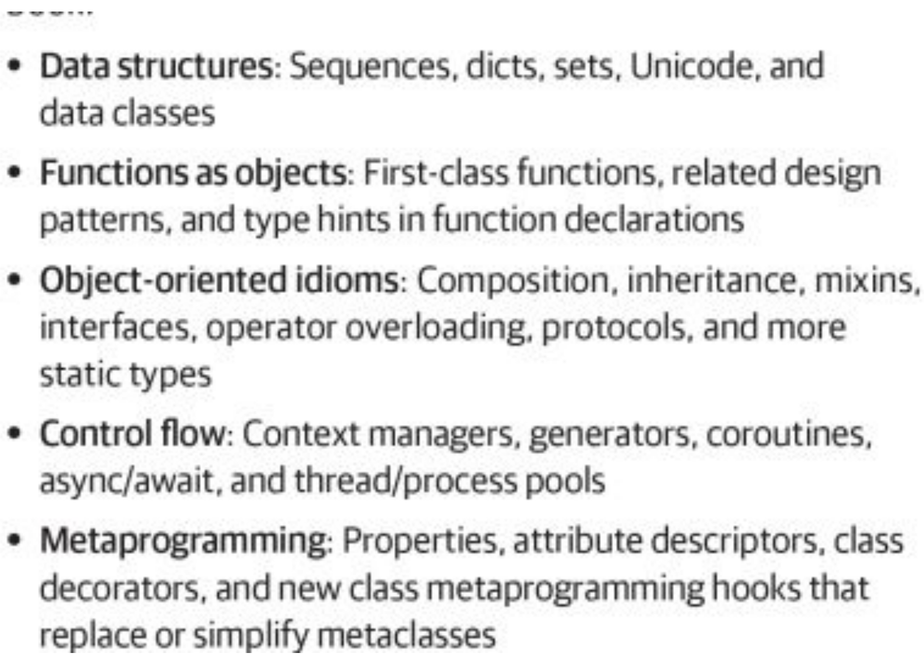
O'REILLY

2nd Edition
Covers Python 3.10

Fluent Python

Clear, Concise, and
Effective Programming

Luciano Ramalho



- **Data structures**: Sequences, dicts, sets, Unicode, and data classes
- **Functions as objects**: First-class functions, related design patterns, and type hints in function declarations
- **Object-oriented idioms**: Composition, inheritance, mixins, interfaces, operator overloading, protocols, and more static types
- **Control flow**: Context managers, generators, coroutines, async/await, and thread/process pools
- **Metaprogramming**: Properties, attribute descriptors, class decorators, and new class metaprogramming hooks that replace or simplify metaclasses

Structure

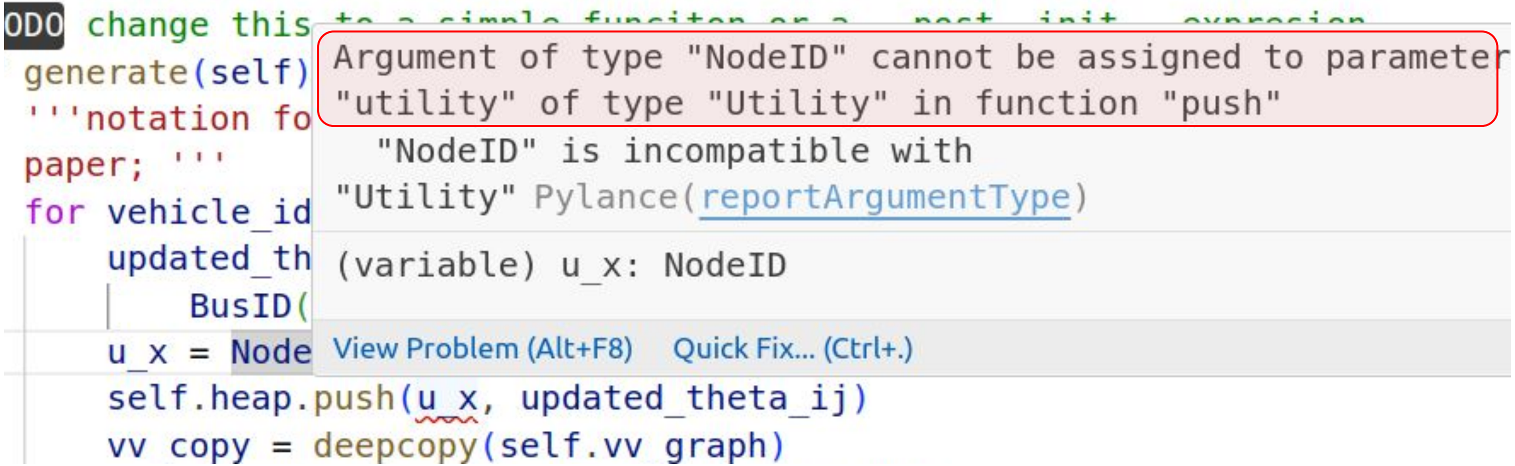
- Typed Python
- Interfaces, ABC and @abstractmethod
- Profiling and Logging
- Generators and list comprehensions
- @staticmethod, @property, @property.setter, @classmethod
- Use global variables in configuration files
- Python Standard Library
- Python/General OOP practices

- Worked example/Workshop

Typed Python

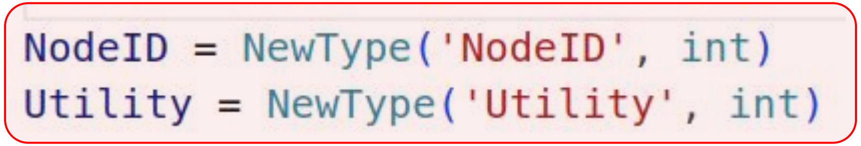
Examples why this is important + main way of impementing typing

Why this is important - static type checking logic



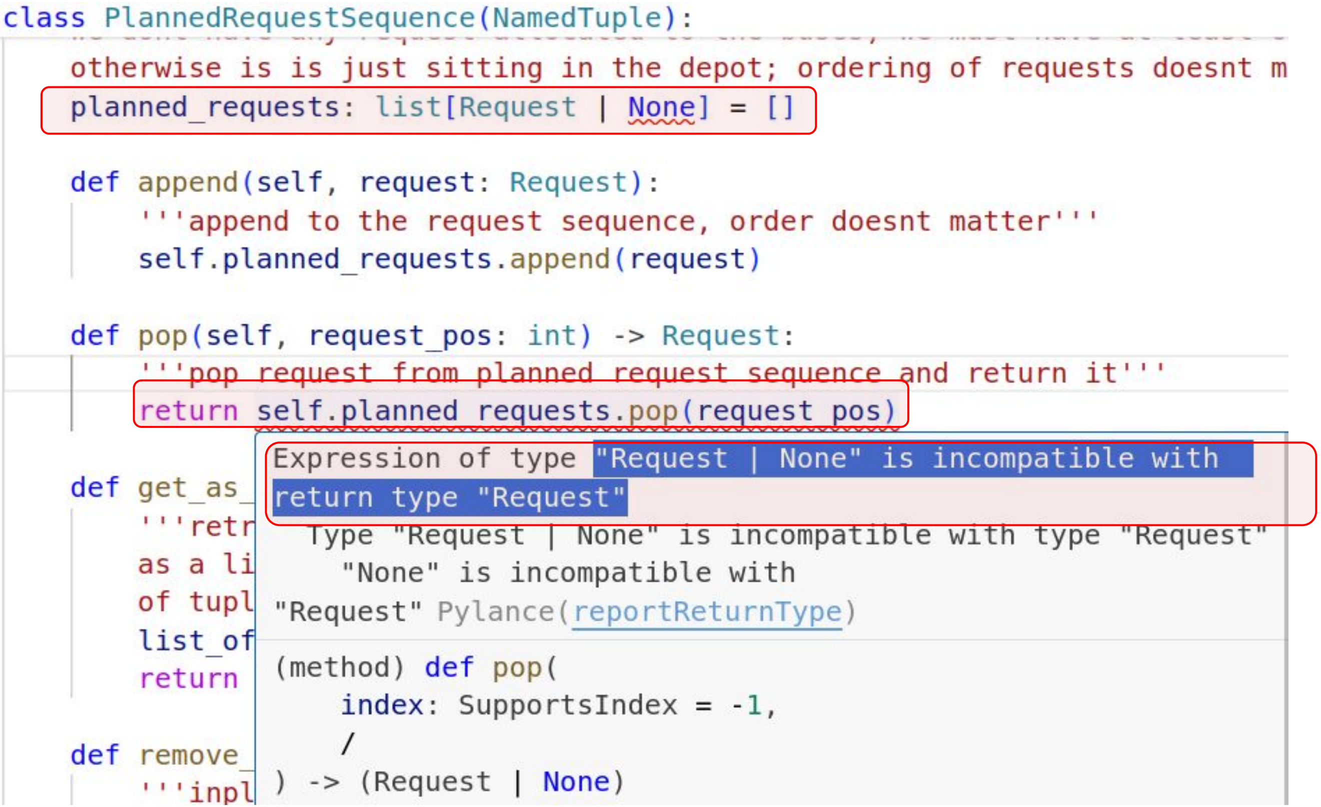
```
ODO change this to a simple funciton or a __post_init__ expresion
generate(self)
'''notation fo
paper; '''
for vehicle_id
    updated_th
        BusID(
    u_x = Node
    self.heap.push(u_x, updated_theta_ij)
    vv_copy = deepcopy(self.vv_graph)
```

```
Argument of type "NodeID" cannot be assigned to parameter
"utility" of type "Utility" in function "push"
  "NodeID" is incompatible with
"Utility" Pylance(reportArgumentType)

(variable) u_x: NodeID
```



```
NodeID = NewType('NodeID', int)
Utility = NewType('Utility', int)
```

Automatic, Static, Edge case detection

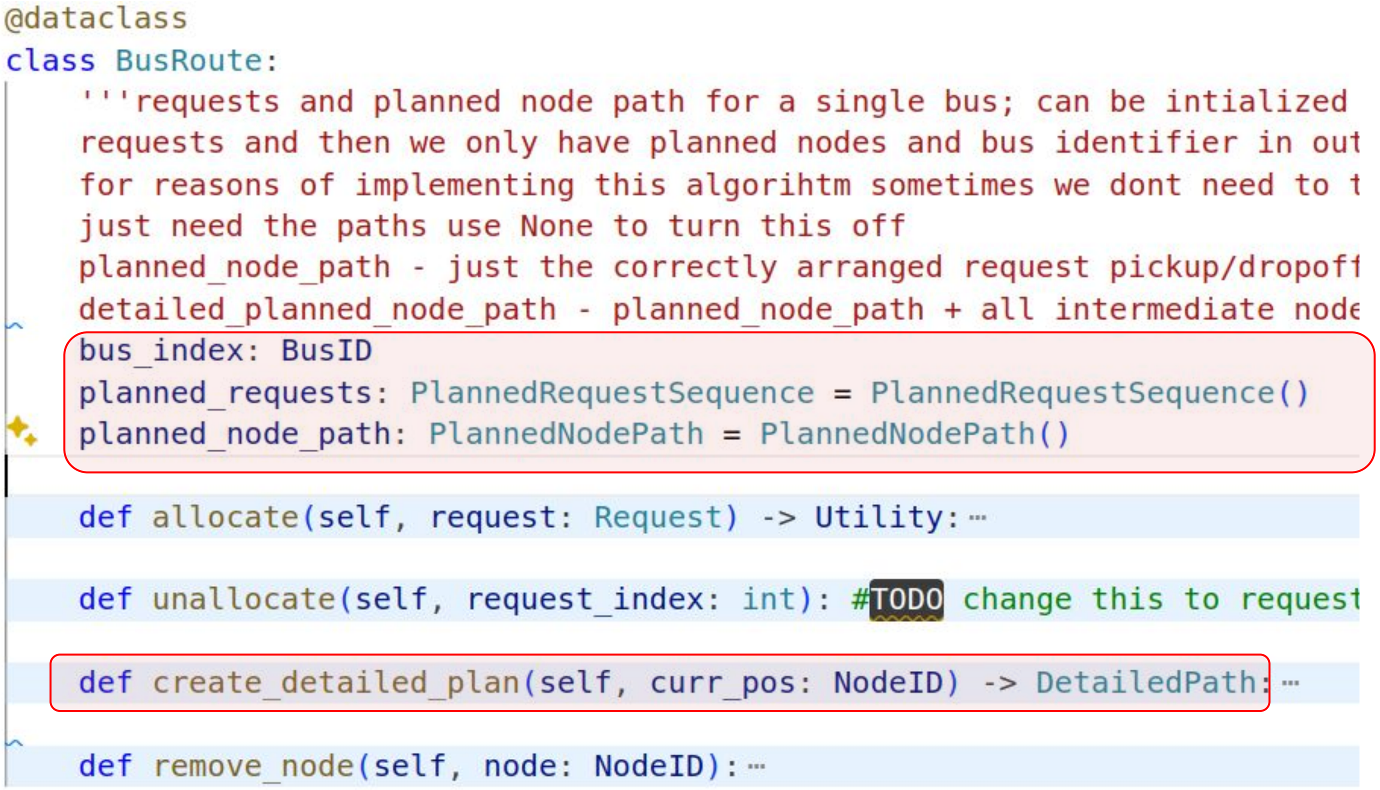


```
class PlannedRequestSequence(NamedTuple):
    otherwise is is just sitting in the depot; ordering of requests doesnt m
    planned_requests: list[Request | None] = []

    def append(self, request: Request):
        '''append to the request sequence, order doesnt matter'''
        self.planned_requests.append(request)

    def pop(self, request_pos: int) -> Request:
        '''pop request from planned request sequence and return it'''
        return self.planned_requests.pop(request_pos)

    def get_as_
        '''retr
        as a li
        of tupl
        list_of
        return

    def remove_
        '''inpl
```

```
Expression of type "Request | None" is incompatible with
return type "Request"
  Type "Request | None" is incompatible with type "Request"
    "None" is incompatible with
"Request" Pylance(reportReturnType)
```

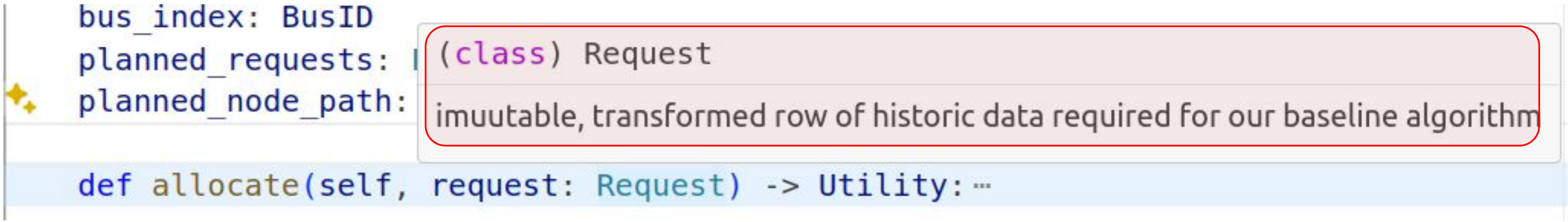
```
(method) def pop(
    index: SupportsIndex = -1,
    /
) -> (Request | None)
```

Increased readability



```
@dataclass
class BusRoute:
    '''requests and planned node path for a single bus; can be intialized
    requests and then we only have planned nodes and bus identifier in out
    for reasons of implementing this algorihtm sometimes we dont need to t
    just need the paths use None to turn this off
    planned_node_path - just the correctly arranged request pickup/dropoff
    detailed_planned_node_path - planned_node_path + all intermediate node
    bus_index: BusID
    planned_requests: PlannedRequestSequence = PlannedRequestSequence()
    planned_node_path: PlannedNodePath = PlannedNodePath()

    def allocate(self, request: Request) -> Utility:…

    def unallocate(self, request_index: int): #TODO change this to request

    def create_detailed_plan(self, curr_pos: NodeID) -> DetailedPath:…

    def remove_node(self, node: NodeID):…
```

Improved documentation

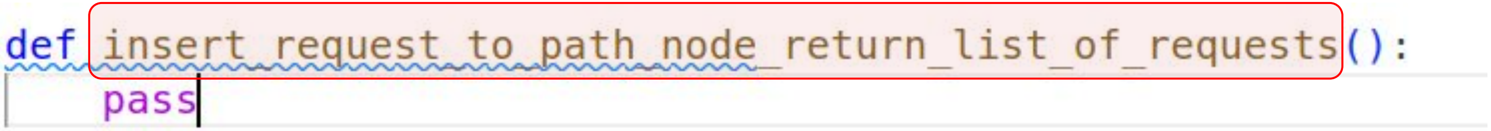


```
    bus_index: BusID
    planned_requests: 
    planned_node_path: 

    def allocate(self, request: Request) -> Utility:⋯
```

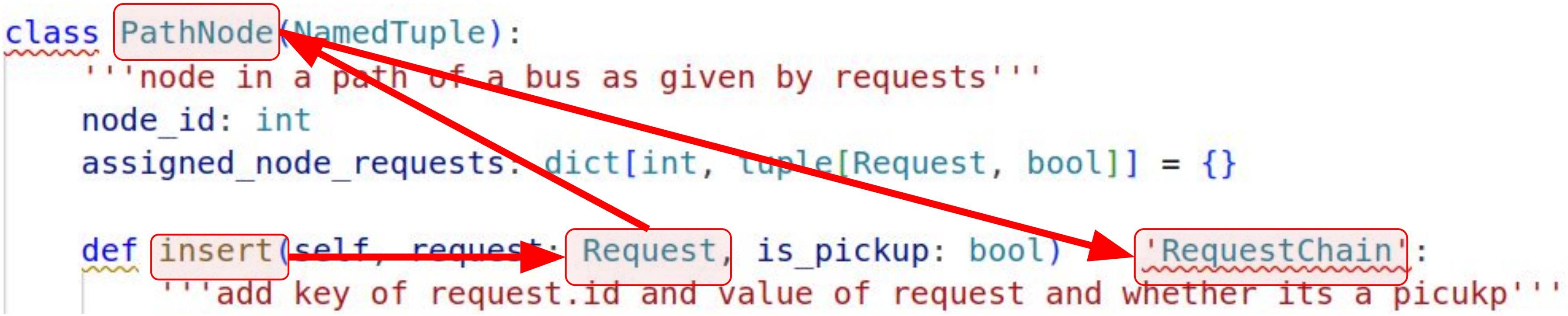
```
(class) Request
```

imuutable, transformed row of historic data required for our baseline algorithm

Simplification of Method/function names

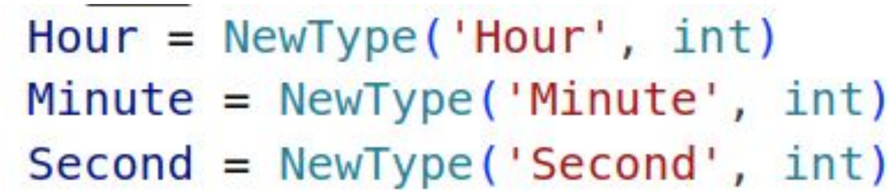


```
def insert_request_to_path_node_return_list_of_requests():
    pass
```

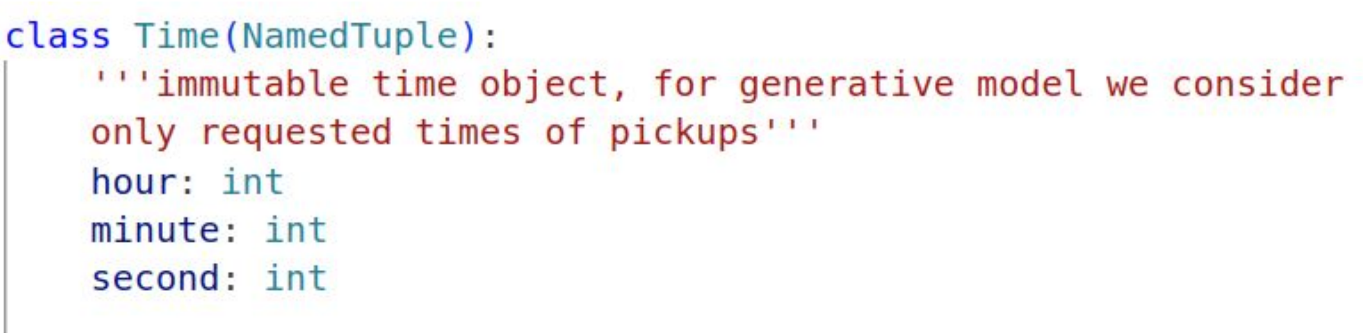


```
class PathNode(NamedTuple):
    '''node in a path of a bus as given by requests'''
    node_id: int
    assigned_node_requests: dict[int, tuple[Request, bool]] = {}

    def insert(self, request: Request, is_pickup: bool) -> 'RequestChain':
        '''add key of request.id and value of request and whether its a picukp'''
```

NewType vs Class based type setting

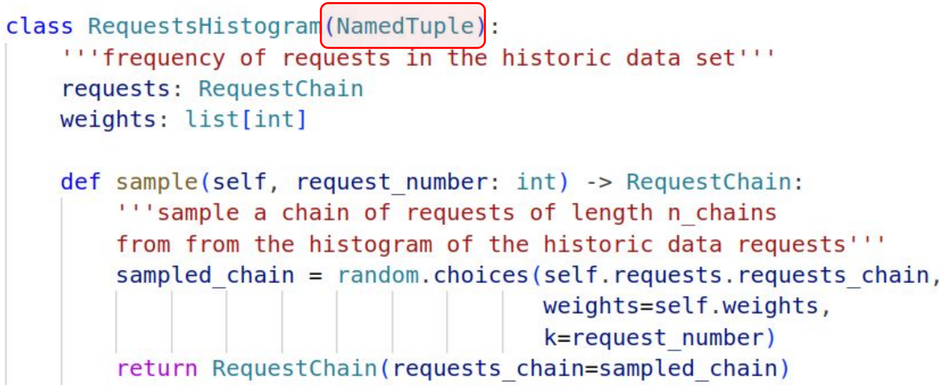


```python
Hour = NewType('Hour', int)
Minute = NewType('Minute', int)
Second = NewType('Second', int)
```

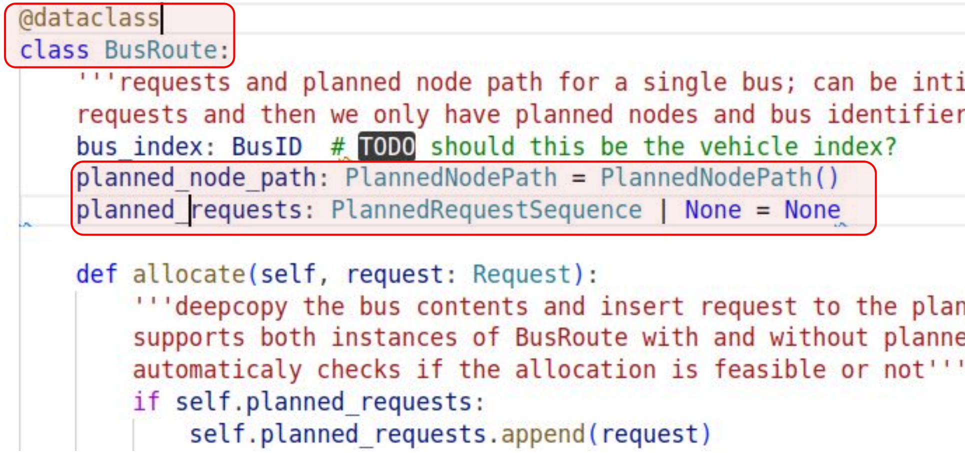


```python
class Time(NamedTuple):
    '''immutable time object, for generative model we consider
    only requested times of pickups'''
    hour: int
    minute: int
    second: int
```

NamedTuples vs. @dataclasses



```python
class RequestsHistogram(NamedTuple):
    '''frequency of requests in the historic data set'''
    requests: RequestChain
    weights: list[int]

    def sample(self, request_number: int) -> RequestChain:
        '''sample a chain of requests of length n_chains
        from from the histogram of the historic data requests'''
        sampled_chain = random.choices(self.requests.requests_chain,
                                       weights=self.weights,
                                       k=request_number)
        return RequestChain(requests_chain=sampled_chain)
```

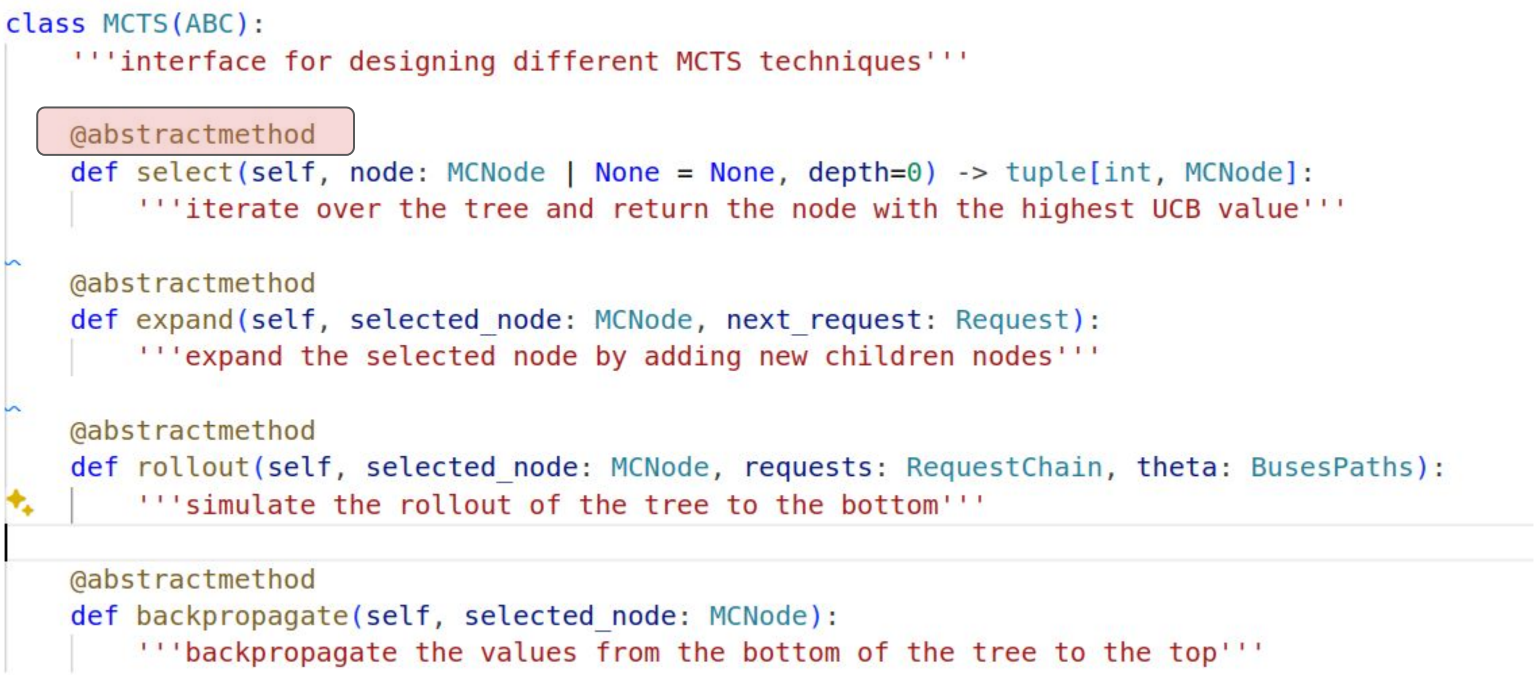


```python
@dataclass
class BusRoute:
    '''requests and planned node path for a single bus; can be inti
    requests and then we only have planned nodes and bus identifier
    bus_index: BusID  # TODO should this be the vehicle index?
    planned_node_path: PlannedNodePath = PlannedNodePath()
    planned_requests: PlannedRequestSequence | None = None

    def allocate(self, request: Request):
        '''deepcopy the bus contents and insert request to the plan
        supports both instances of BusRoute with and without planne
        automaticaly checks if the allocation is feasible or not'''
        if self.planned_requests:
            self.planned_requests.append(request)
```

Interfaces, ABC and @abstractmethod

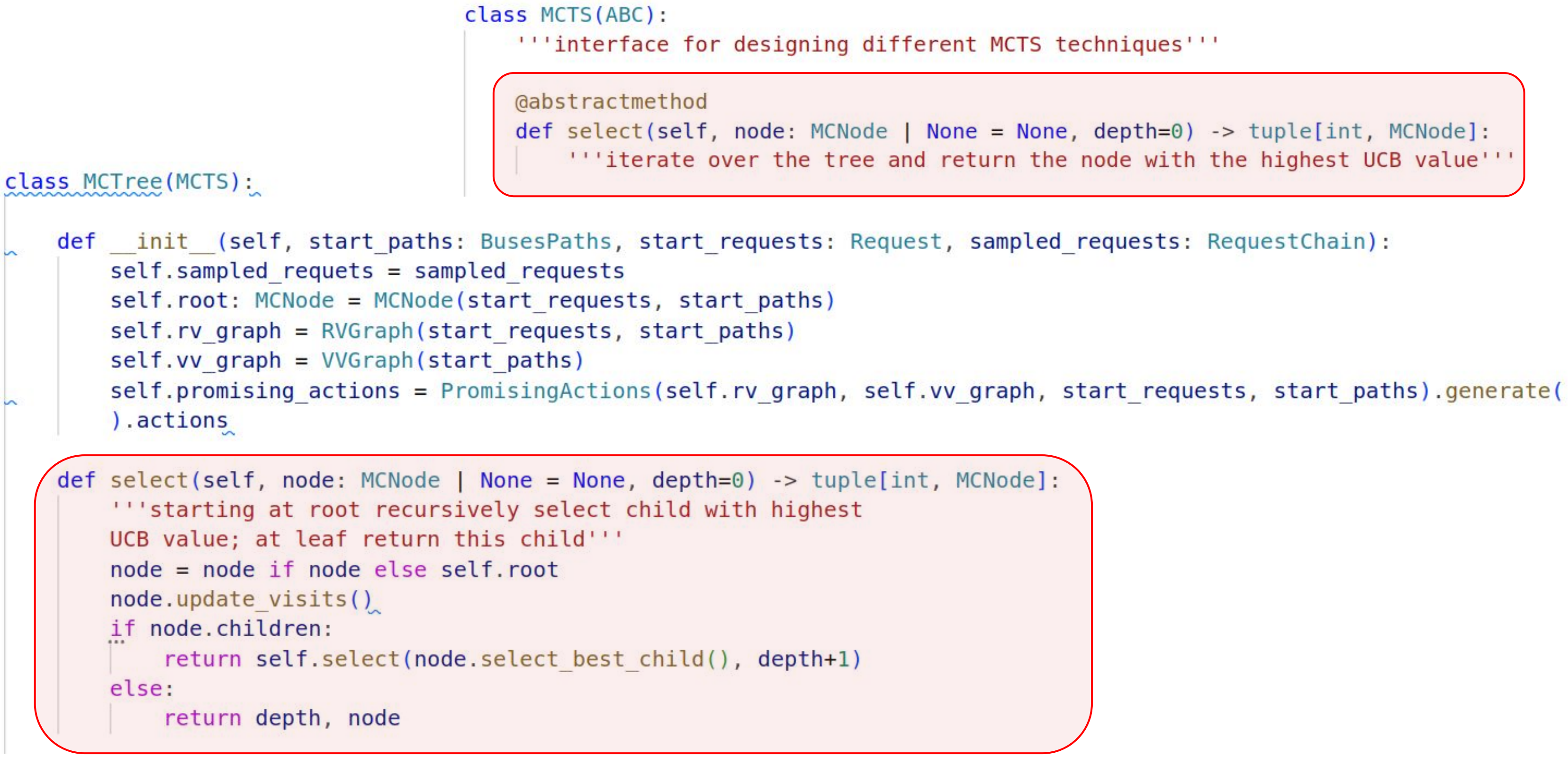
How to improve high level documentation of you code

Interface for Class structure



```
class MCTS(ABC):
    '''interface for designing different MCTS techniques'''

    @abstractmethod
    def select(self, node: MCNode | None = None, depth=0) -> tuple[int, MCNode]:
        '''iterate over the tree and return the node with the highest UCB value'''

    @abstractmethod
    def expand(self, selected_node: MCNode, next_request: Request):
        '''expand the selected node by adding new children nodes'''

    @abstractmethod
    def rollout(self, selected_node: MCNode, requests: RequestChain, theta: BusesPaths):
        '''simulate the rollout of the tree to the bottom'''

    @abstractmethod
    def backpropagate(self, selected_node: MCNode):
        '''backpropagate the values from the bottom of the tree to the top'''
```

Interface for Class structure



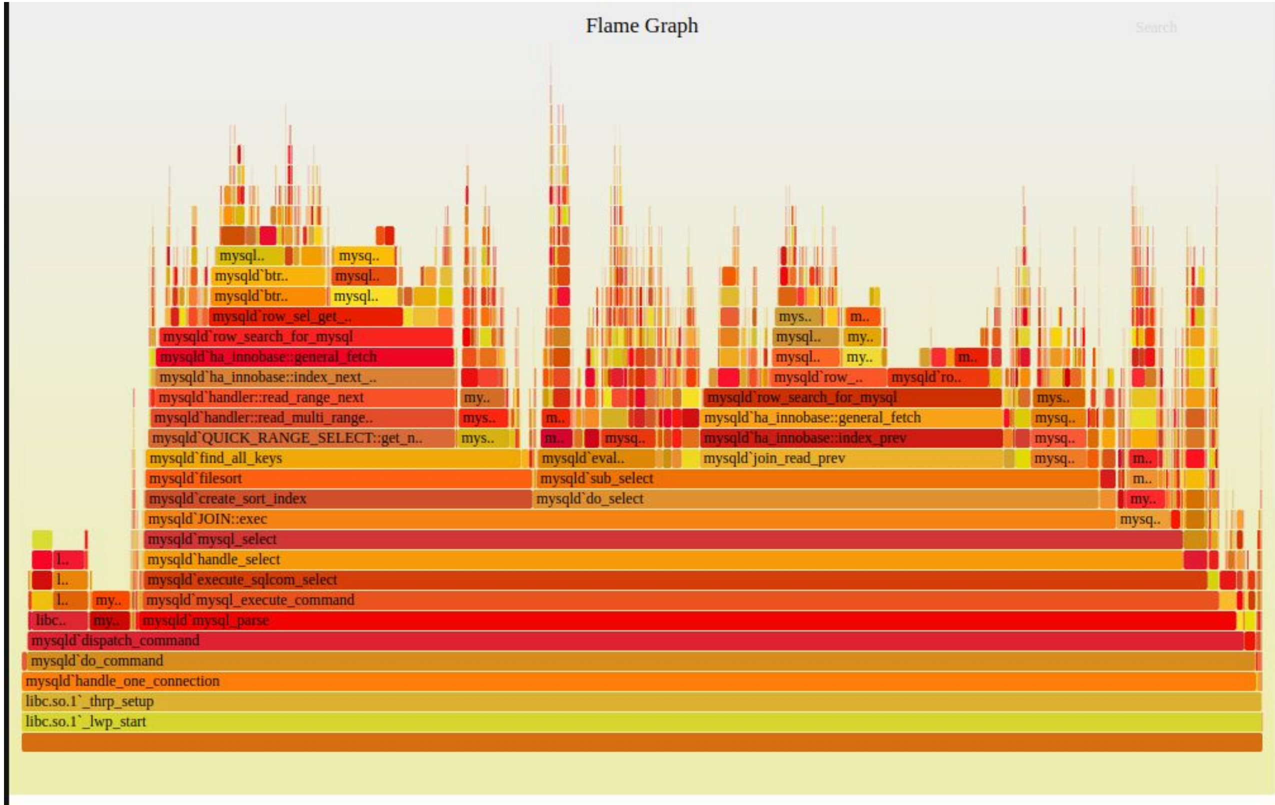
```python
class MCTS(ABC):
    '''interface for designing different MCTS techniques'''

    @abstractmethod
    def select(self, node: MCNode | None = None, depth=0) -> tuple[int, MCNode]:
        '''iterate over the tree and return the node with the highest UCB value'''
```

```python
class MCTree(MCTS):

    def __init__(self, start_paths: BusesPaths, start_requests: Request, sampled_requests: RequestChain):
        self.sampled_requets = sampled_requests
        self.root: MCNode = MCNode(start_requests, start_paths)
        self.rv_graph = RVGraph(start_requests, start_paths)
        self.vv_graph = VVGraph(start_paths)
        self.promising_actions = PromisingActions(self.rv_graph, self.vv_graph, start_requests, start_paths).generate(
        ).actions

    def select(self, node: MCNode | None = None, depth=0) -> tuple[int, MCNode]:
        '''starting at root recursively select child with highest
        UCB value; at leaf return this child'''
        node = node if node else self.root
        node.update_visits()
        if node.children:
            return self.select(node.select_best_child(), depth+1)
        else:
            return depth, node
```

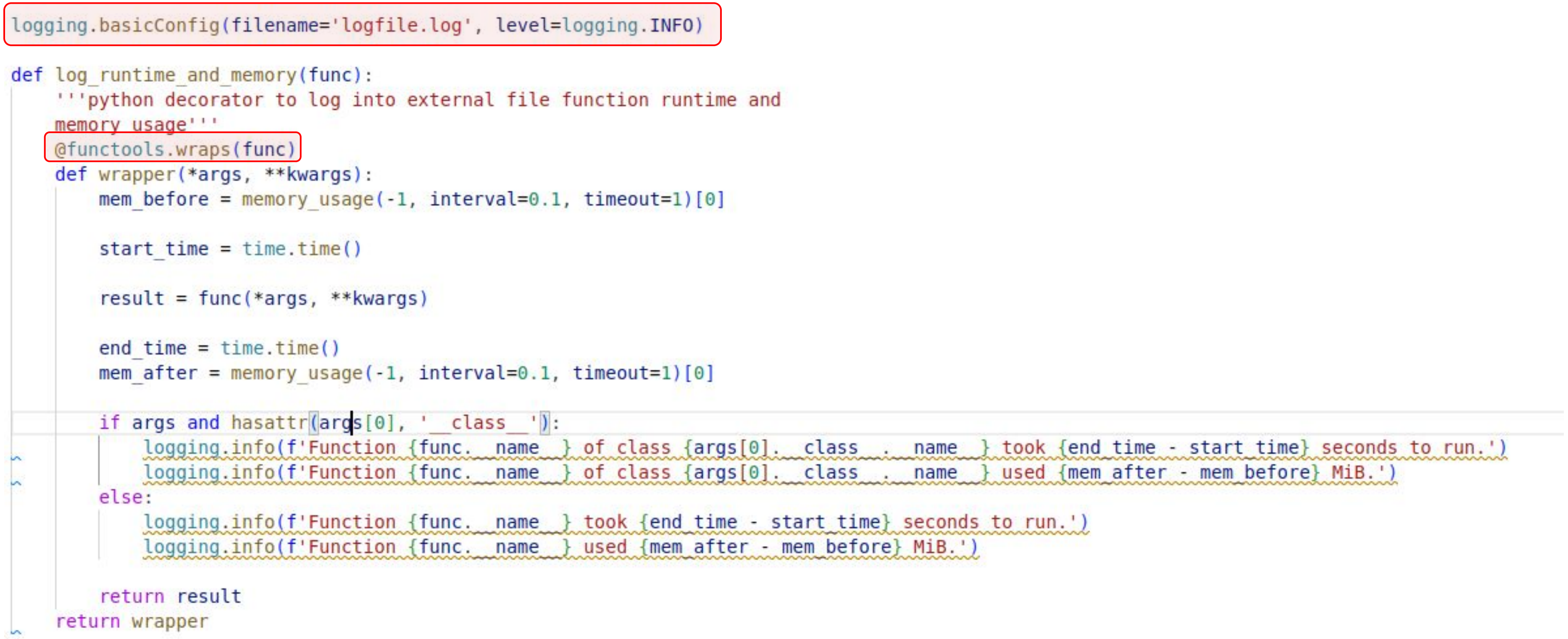
Profiling and Logging

Flame graph

CPU, Memory Flame Graph tells us where to focus

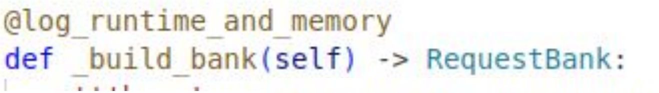


Use specialized decorators and loggers

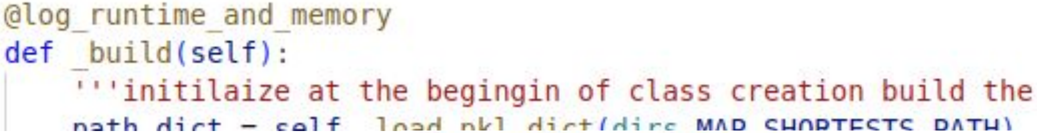


```
logging.basicConfig(filename='logfile.log', level=logging.INFO)

def log_runtime_and_memory(func):
    '''python decorator to log into external file function runtime and
    memory usage'''
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        mem_before = memory_usage(-1, interval=0.1, timeout=1)[0]

        start_time = time.time()

        result = func(*args, **kwargs)

        end_time = time.time()
        mem_after = memory_usage(-1, interval=0.1, timeout=1)[0]

        if args and hasattr(args[0], '__class__'):
            logging.info(f'Function {func.__name__} of class {args[0].__class__.__name__} took {end_time - start_time} seconds to run.')
            logging.info(f'Function {func.__name__} of class {args[0].__class__.__name__} used {mem_after - mem_before} MiB.')
        else:
            logging.info(f'Function {func.__name__} took {end_time - start_time} seconds to run.')
            logging.info(f'Function {func.__name__} used {mem_after - mem_before} MiB.')

        return result
    return wrapper
```

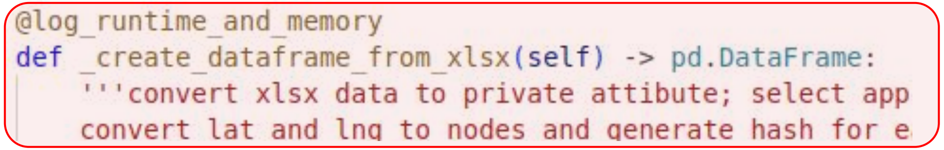
Use specialized decorators and loggers



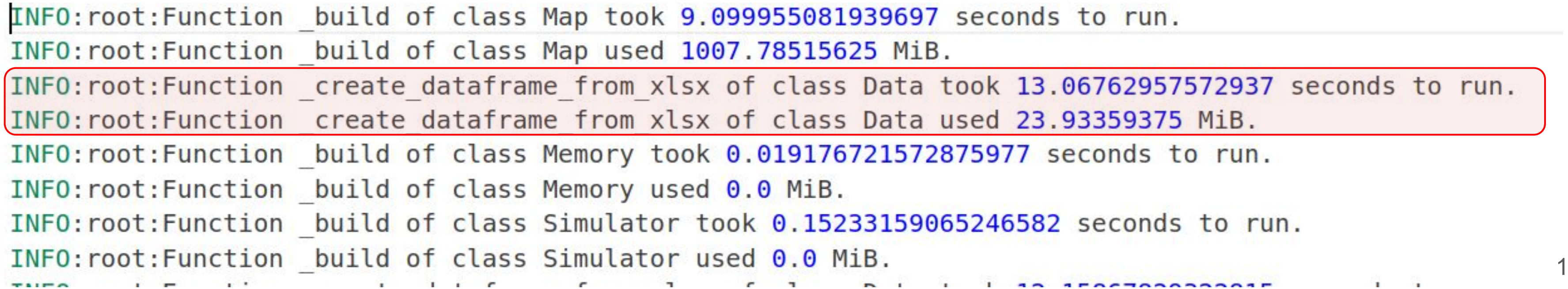
```
@log_runtime_and_memory
def _build_bank(self) -> RequestBank:
```



```
@log_runtime_and_memory
def _build(self):
    '''initilaize at the begingin of class creation build the
    path_dict = self._load_pkl_dict(dirs.MAP_SHORTESTS_PATH)
```

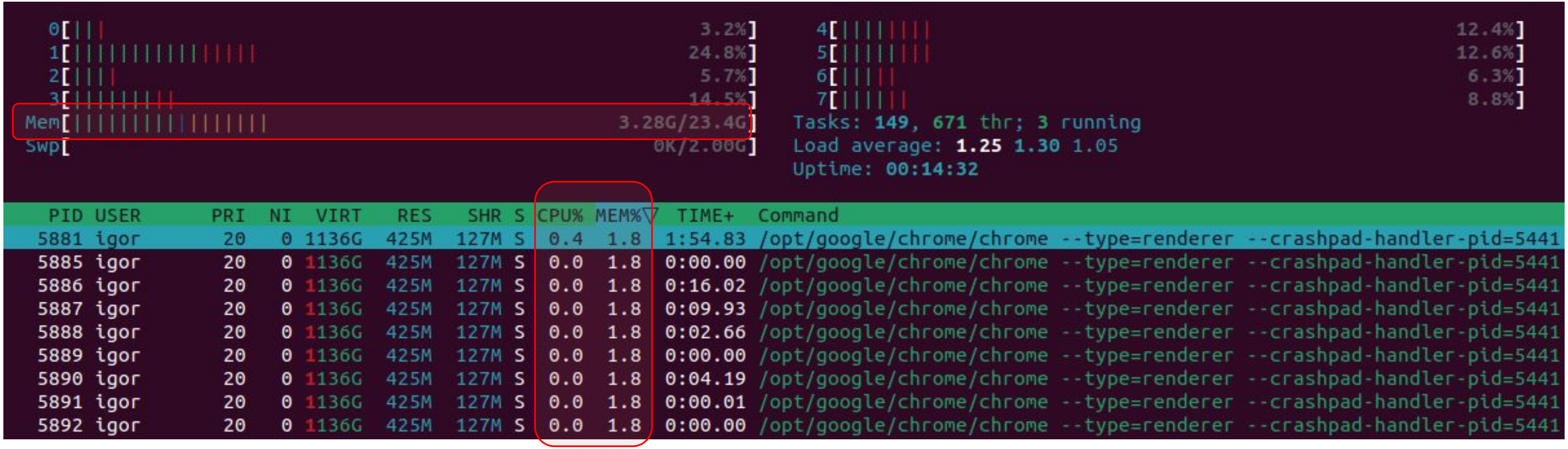


```
@log_runtime_and_memory
def _create_dataframe_from_xlsx(self) -> pd.DataFrame:
    '''convert xlsx data to private attibute; select app
    convert lat and lng to nodes and generate hash for e
```



```
INFO:root:Function _build of class Map took 9.099955081939697 seconds to run.
INFO:root:Function _build of class Map used 1007.78515625 MiB.
INFO:root:Function _create_dataframe_from_xlsx of class Data took 13.06762957572937 seconds to run.
INFO:root:Function _create_dataframe_from_xlsx of class Data used 23.93359375 MiB.
INFO:root:Function _build of class Memory took 0.019176721572875977 seconds to run.
INFO:root:Function _build of class Memory used 0.0 MiB.
INFO:root:Function _build of class Simulator took 0.15233159065246582 seconds to run.
INFO:root:Function _build of class Simulator used 0.0 MiB.
```

Actively monitor the global system memory and CPU requirements (VSC takes a LOT of RAM)

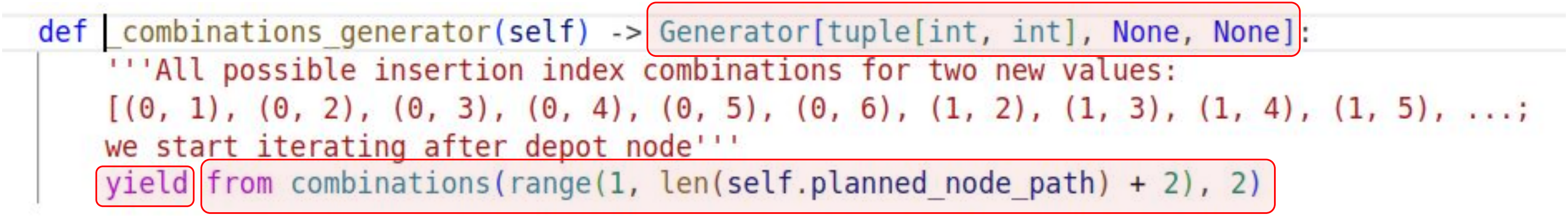


```
   0[|||                                         3.2%]     4[||||||||                                              12.4%]
   1[||||||||||||||||                           24.8%]     5[||||||||                                              12.6%]
   2[||||                                        5.7%]     6[|||||                                                  6.3%]
   3[|||||||||                                  14.5%]     7[||||||                                                 8.8%]
 Mem[|||||||||||||||||                    3.28G/23.4G]   Tasks: 149, 671 thr; 3 running
 Swp[                                        0K/2.00G]   Load average: 1.25 1.30 1.05
                                                         Uptime: 00:14:32

   PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%▽  TIME+  Command
  5881 igor       20   0 1136G  425M  127M S  0.4  1.8  1:54.83 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=5441
  5885 igor       20   0 1136G  425M  127M S  0.0  1.8  0:00.00 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=5441
  5886 igor       20   0 1136G  425M  127M S  0.0  1.8  0:16.02 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=5441
  5887 igor       20   0 1136G  425M  127M S  0.0  1.8  0:09.93 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=5441
  5888 igor       20   0 1136G  425M  127M S  0.0  1.8  0:02.66 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=5441
  5889 igor       20   0 1136G  425M  127M S  0.0  1.8  0:00.00 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=5441
  5890 igor       20   0 1136G  425M  127M S  0.0  1.8  0:04.19 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=5441
  5891 igor       20   0 1136G  425M  127M S  0.0  1.8  0:00.01 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=5441
  5892 igor       20   0 1136G  425M  127M S  0.0  1.8  0:00.00 /opt/google/chrome/chrome --type=renderer --crashpad-handler-pid=5441
```

Generators and list comprehensions

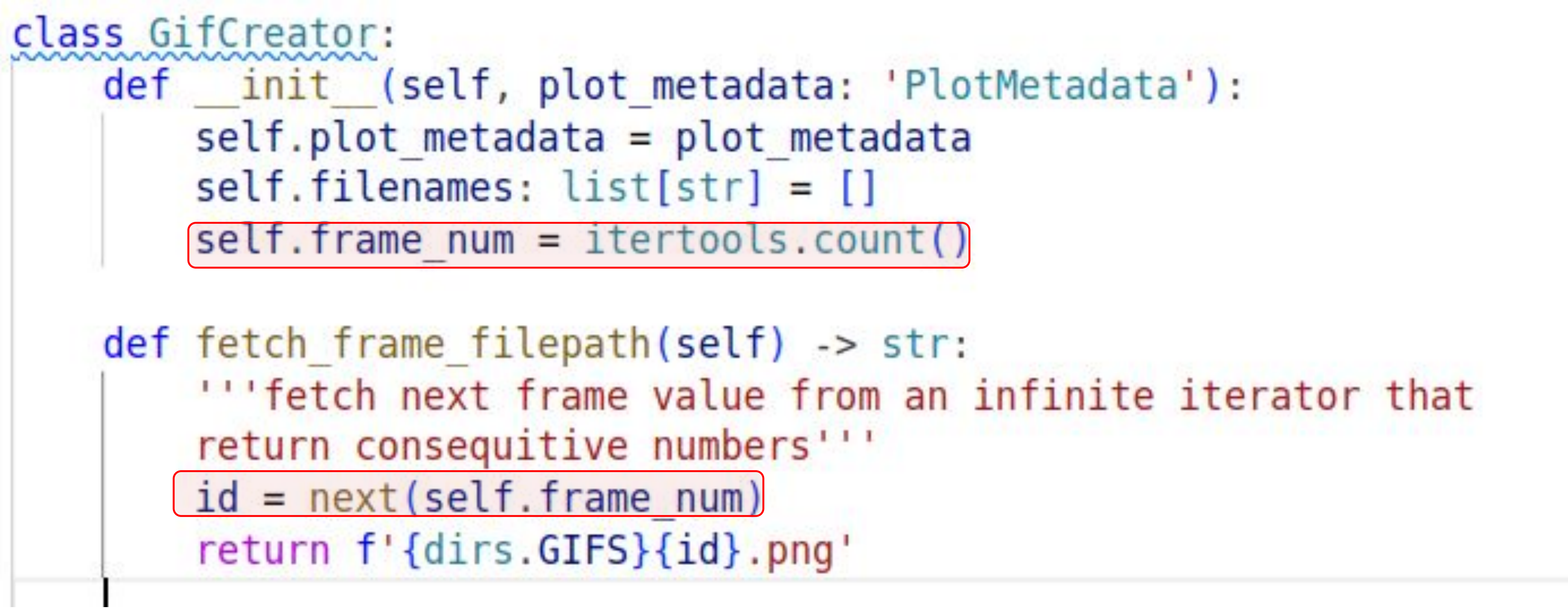
Cleaner way to deal with nested loop and robust way to achieve consistency in fetching values

Using Generator functions



```python
def _combinations_generator(self) -> Generator[tuple[int, int], None, None]:
    '''All possible insertion index combinations for two new values:
    [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (1, 2), (1, 3), (1, 4), (1, 5), ...;
    we start iterating after depot node'''
    yield from combinations(range(1, len(self.planned_node_path) + 2), 2)
```

Generator instances (just like deque) allow for consistent tracking of values inside them

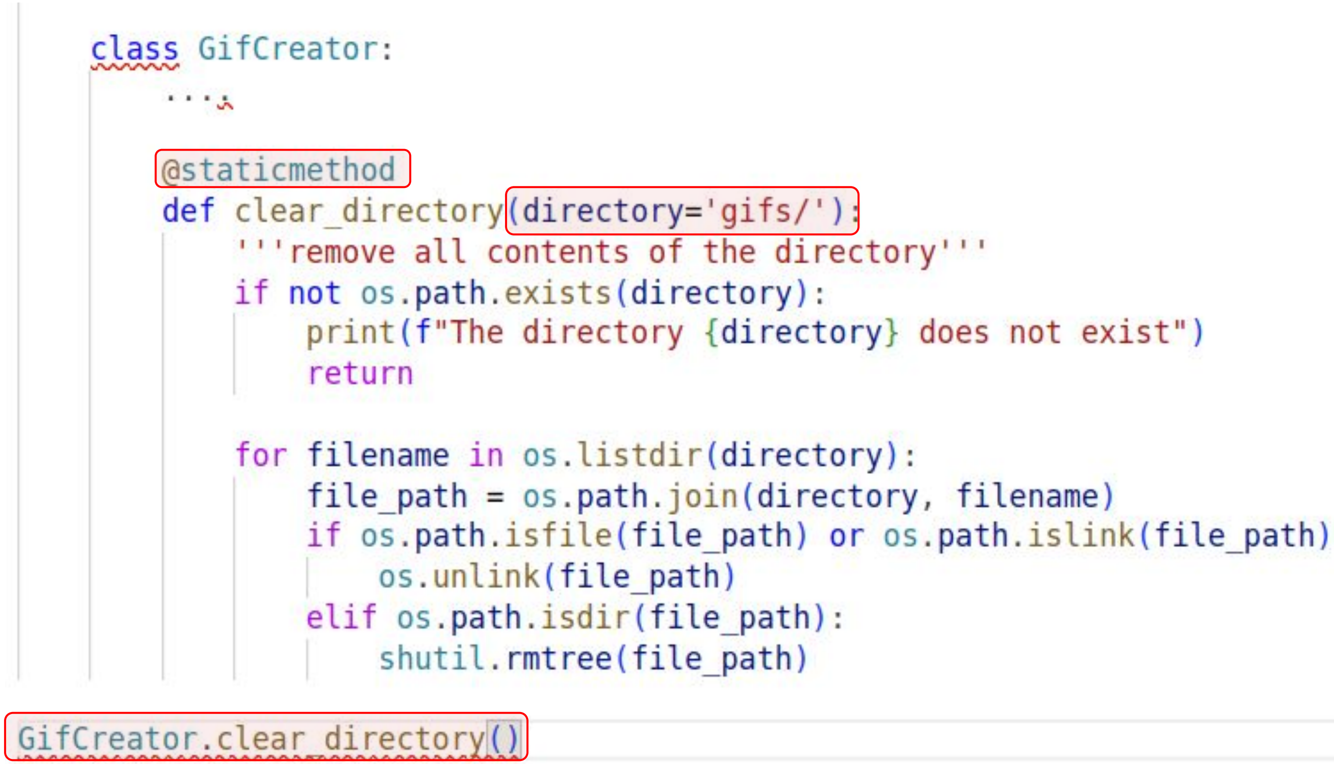


```
class GifCreator:
    def __init__(self, plot_metadata: 'PlotMetadata'):
        self.plot_metadata = plot_metadata
        self.filenames: list[str] = []
        self.frame_num = itertools.count()

    def fetch_frame_filepath(self) -> str:
        '''fetch next frame value from an infinite iterator that
        return consequitive numbers'''
        id = next(self.frame_num)
        return f'{dirs.GIFS}{id}.png'
```

@staticmethod, @property, @property.setter, @classmethod

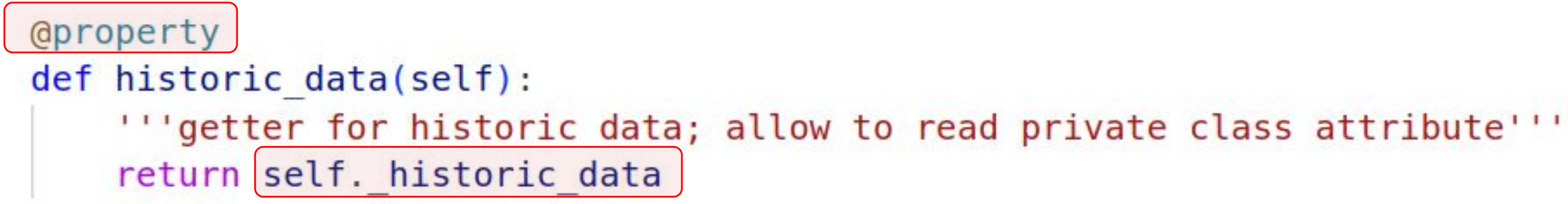
Main methods for customising our classes method

@staticmethod good way to group similar function together

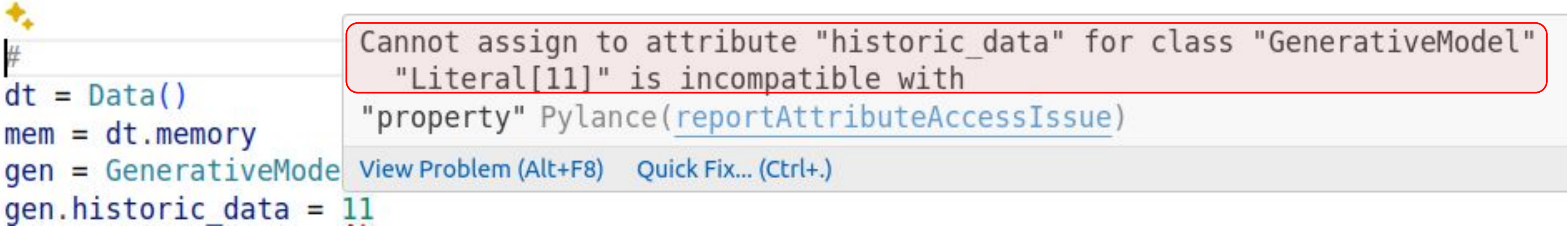


```
class GifCreator:
    ....

    @staticmethod
    def clear_directory(directory='gifs/'):
        '''remove all contents of the directory'''
        if not os.path.exists(directory):
            print(f"The directory {directory} does not exist")
            return

        for filename in os.listdir(directory):
            file_path = os.path.join(directory, filename)
            if os.path.isfile(file_path) or os.path.islink(file_path)
                os.unlink(file_path)
            elif os.path.isdir(file_path):
                shutil.rmtree(file_path)

GifCreator.clear_directory()
```

@property - method to access class attributes and make them immutable



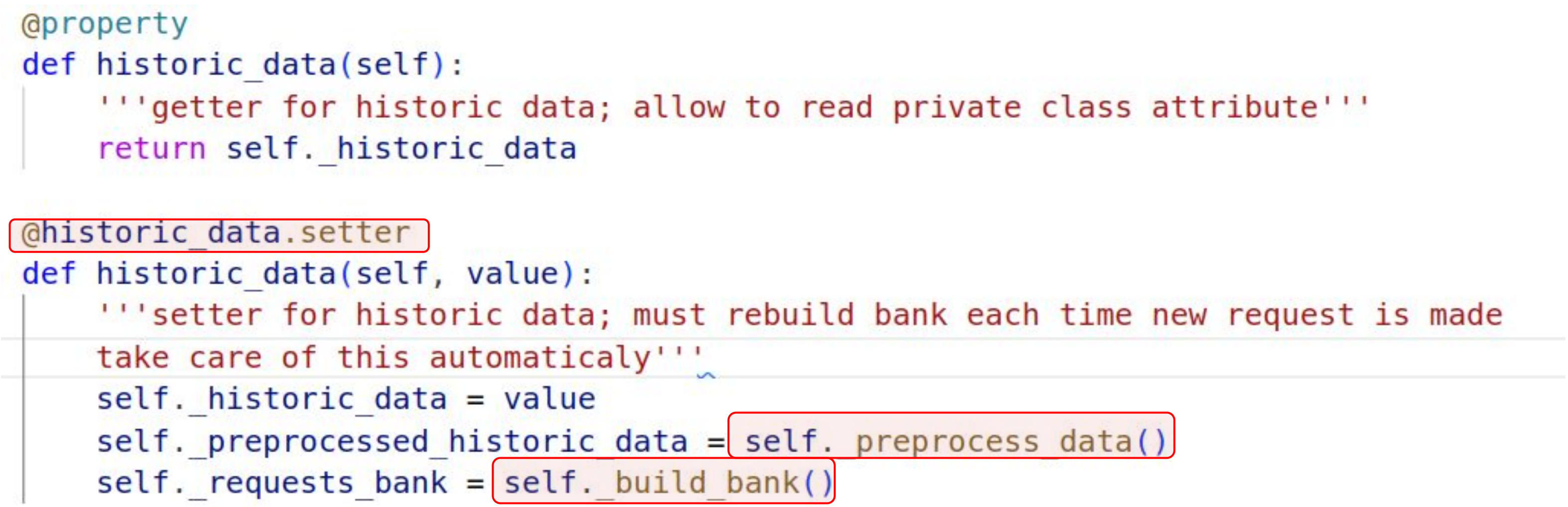
```
@property
def historic_data(self):
    '''getter for historic data; allow to read private class attribute'''
    return self._historic_data
```



```
#
dt = Data()
mem = dt.memory
gen = GenerativeMode
gen.historic_data = 11
```

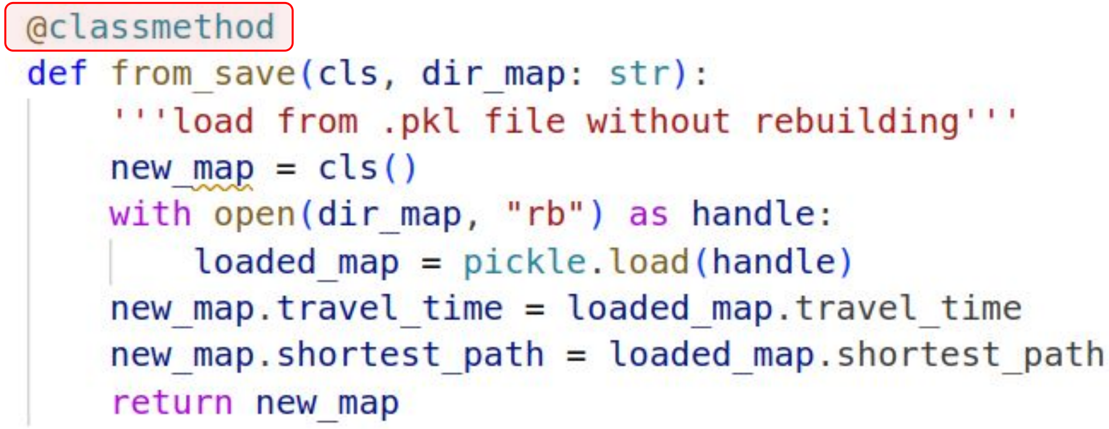
```
Cannot assign to attribute "historic_data" for class "GenerativeModel"
  "Literal[11]" is incompatible with
"property" Pylance(reportAttributeAccessIssue)
```

@<property>.setter - automatising setting the attributes values when we create them

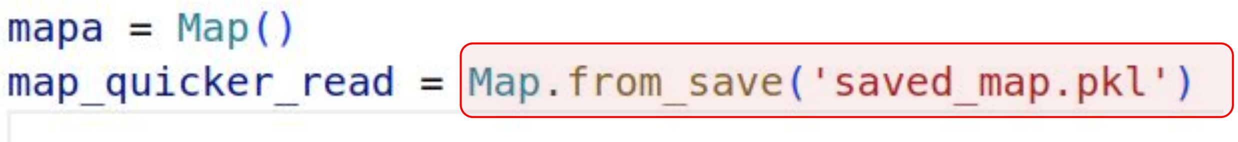


```
@property
def historic_data(self):
    '''getter for historic data; allow to read private class attribute'''
    return self._historic_data

@historic_data.setter
def historic_data(self, value):
    '''setter for historic data; must rebuild bank each time new request is made
    take care of this automaticaly'''
    self._historic_data = value
    self._preprocessed_historic_data = self._preprocess_data()
    self._requests_bank = self._build_bank()
```

@classmethod - custom way of initializing code



```
@classmethod
def from_save(cls, dir_map: str):
    '''load from .pkl file without rebuilding'''
    new_map = cls()
    with open(dir_map, "rb") as handle:
        loaded_map = pickle.load(handle)
    new_map.travel_time = loaded_map.travel_time
    new_map.shortest_path = loaded_map.shortest_path
    return new_map
```

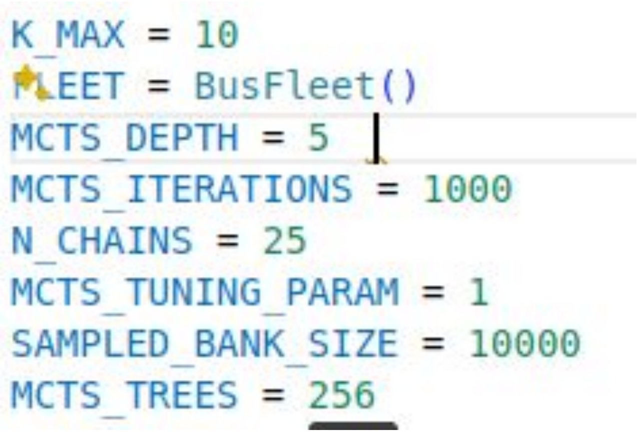


```
mapa = Map()
map_quicker_read = Map.from_save('saved_map.pkl')
```

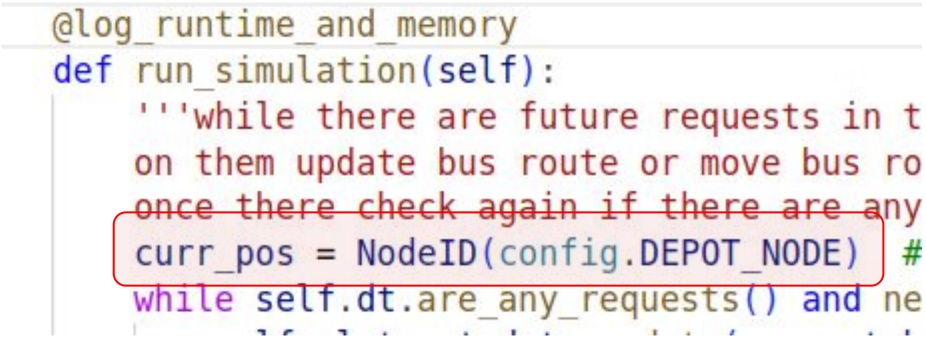
Use global variables in configuration files

config.py and dirs.py for metaparameters and directories

config.py and dirs.py

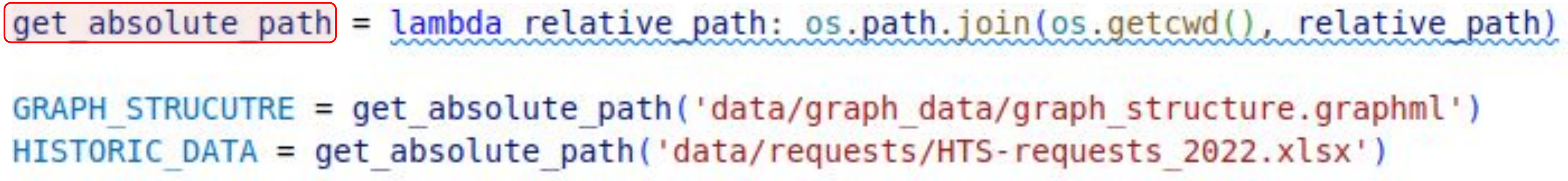


```
K_MAX = 10
FLEET = BusFleet()
MCTS_DEPTH = 5
MCTS_ITERATIONS = 1000
N_CHAINS = 25
MCTS_TUNING_PARAM = 1
SAMPLED_BANK_SIZE = 10000
MCTS_TREES = 256
```



```
@log_runtime_and_memory
def run_simulation(self):
    '''while there are future requests in t
    on them update bus route or move bus ro
    once there check again if there are any
    curr_pos = NodeID(config.DEPOT_NODE)  #
    while self.dt.are_any_requests() and ne
```

dirs.py keep relative path piped to absolute ones

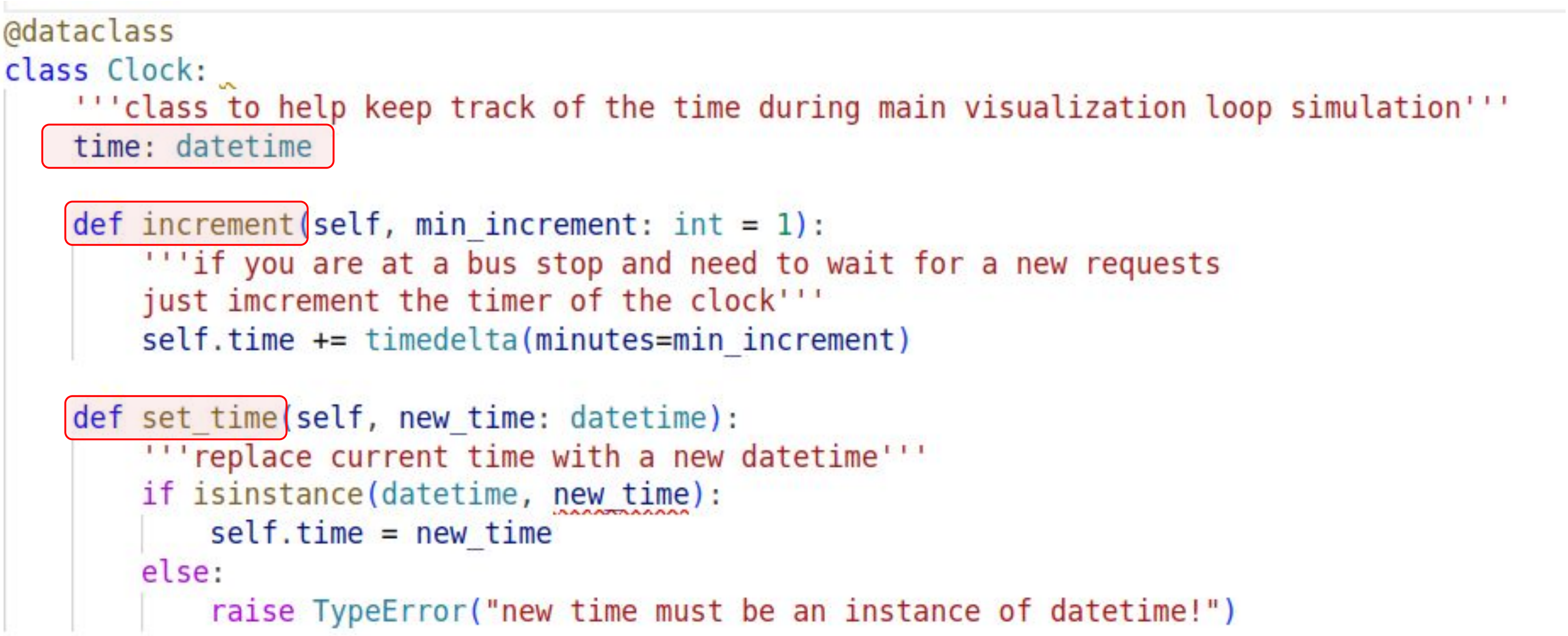


```
get_absolute_path = lambda relative_path: os.path.join(os.getcwd(), relative_path)

GRAPH_STRUCUTRE = get_absolute_path('data/graph_data/graph_structure.graphml')
HISTORIC_DATA = get_absolute_path('data/requests/HTS-requests_2022.xlsx')
```

Python/General OOP practices

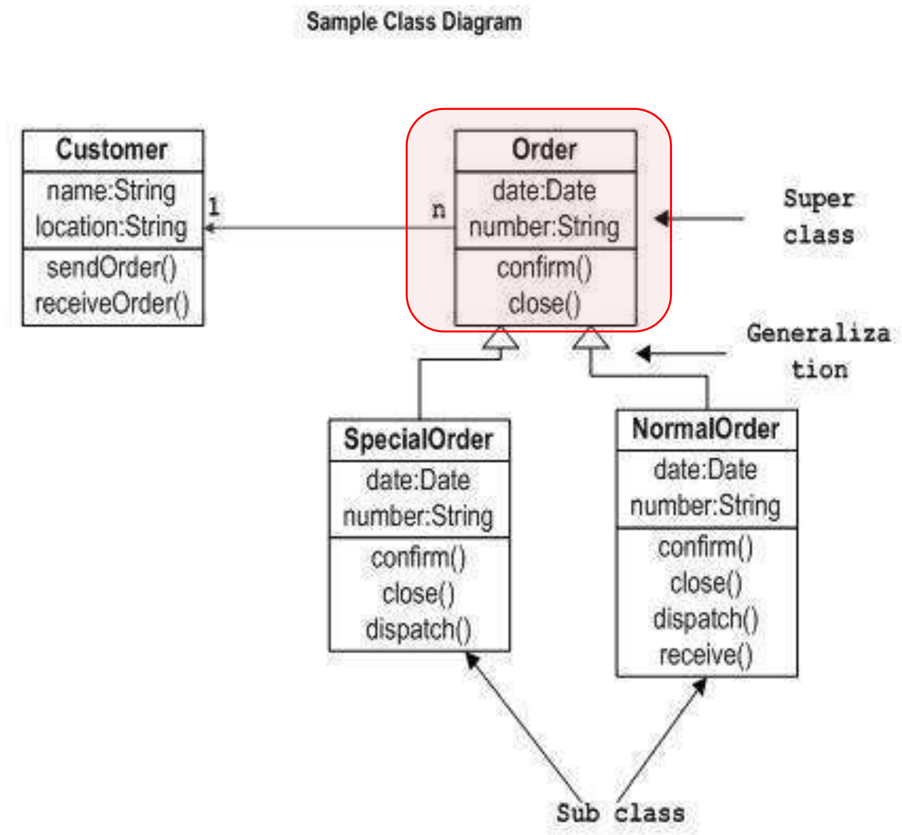
When to define classes, design choices for setting the namespaces

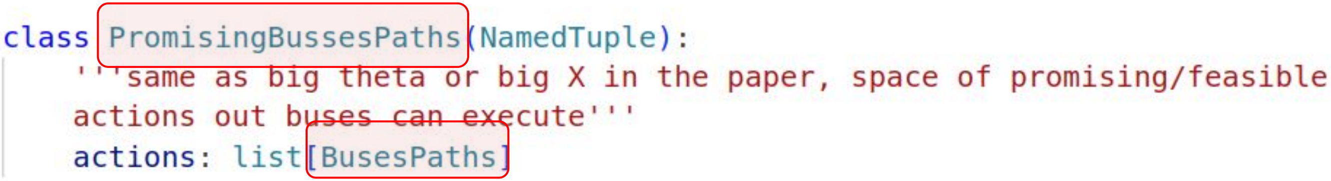
If it has data and methods for this data then it could be a class



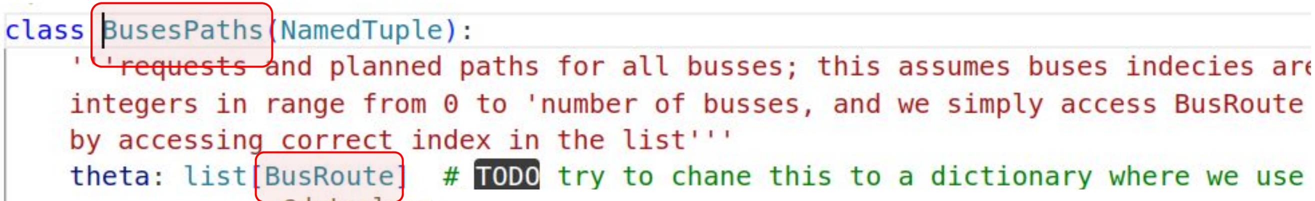
```python
@dataclass
class Clock:
    '''class to help keep track of the time during main visualization loop simulation'''
    time: datetime

    def increment(self, min_increment: int = 1):
        '''if you are at a bus stop and need to wait for a new requests
        just imcrement the timer of the clock'''
        self.time += timedelta(minutes=min_increment)

    def set_time(self, new_time: datetime):
        '''replace current time with a new datetime'''
        if isinstance(datetime, new_time):
            self.time = new_time
        else:
            raise TypeError("new time must be an instance of datetime!")
```

Methods in namescopes should be self explanatory

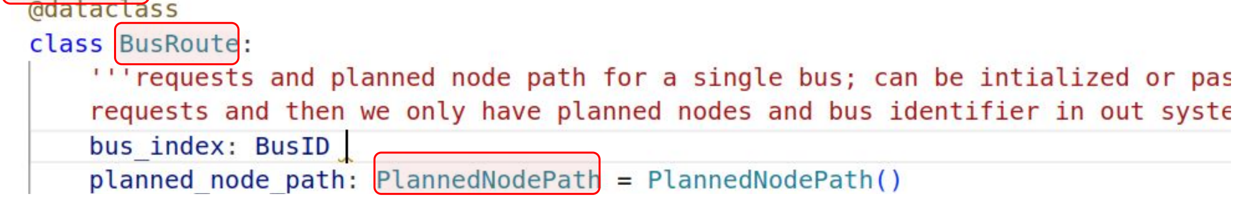




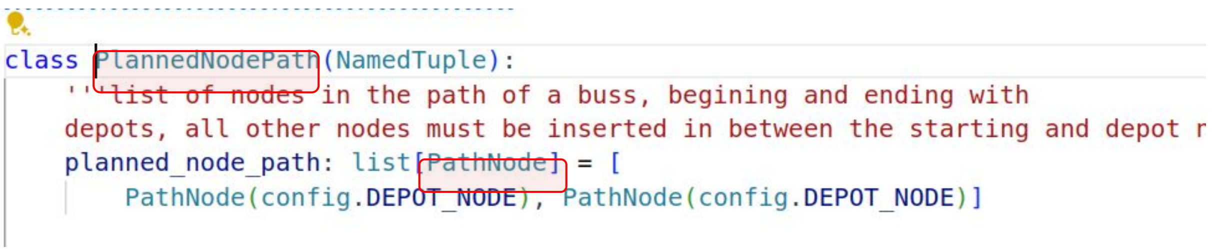
```
class PromisingBussesPaths(NamedTuple):
    '''same as big theta or big X in the paper, space of promising/feasible
    actions out buses can execute'''
    actions: list[BusesPaths]
```



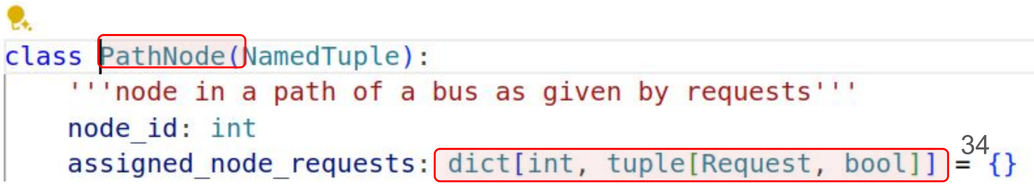
```
class BusesPaths(NamedTuple):
    '''requests and planned paths for all busses; this assumes buses indecies are
    integers in range from 0 to 'number of busses, and we simply access BusRoute
    by accessing correct index in the list'''
    theta: list[BusRoute]  # TODO try to chane this to a dictionary where we use
```



```
@dataclass
class BusRoute:
    '''requests and planned node path for a single bus; can be intialized or pas
    requests and then we only have planned nodes and bus identifier in out syste
    bus_index: BusID
    planned_node_path: PlannedNodePath = PlannedNodePath()
```



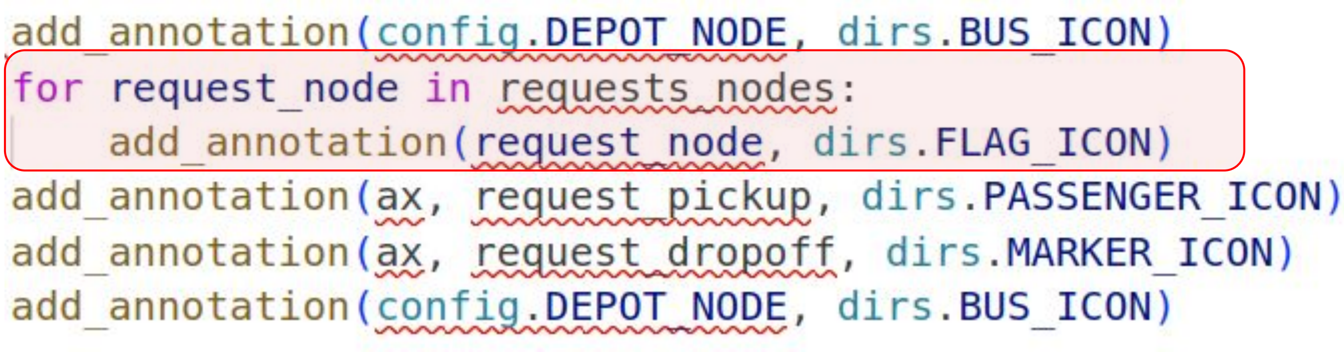
```
class PlannedNodePath(NamedTuple):
    '''list of nodes in the path of a buss, begining and ending with
    depots, all other nodes must be inserted in between the starting and depot r
    planned_node_path: list[PathNode] = [
        PathNode(config.DEPOT_NODE), PathNode(config.DEPOT_NODE)]
```



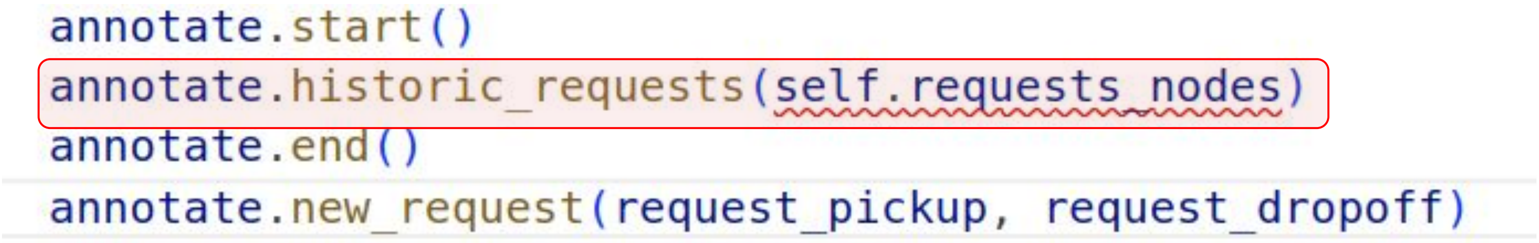
```
class PathNode(NamedTuple):
    '''node in a path of a bus as given by requests'''
    node_id: int
    assigned_node_requests: dict[int, tuple[Request, bool]] = {}
```

Operations with namescopes should be self explanatory

Natural language should guide layers of abstraction level

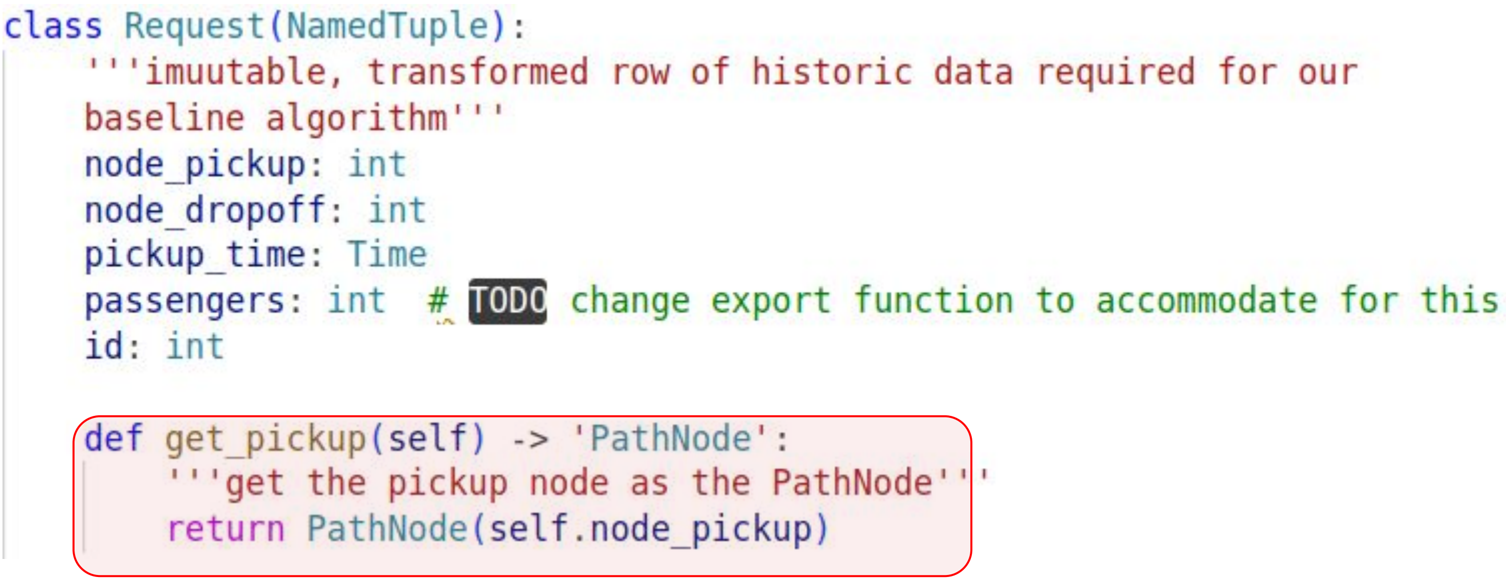


```
add_annotation(config.DEPOT_NODE, dirs.BUS_ICON)
for request_node in requests_nodes:
    add_annotation(request_node, dirs.FLAG_ICON)
add_annotation(ax, request_pickup, dirs.PASSENGER_ICON)
add_annotation(ax, request_dropoff, dirs.MARKER_ICON)
add_annotation(config.DEPOT_NODE, dirs.BUS_ICON)
```



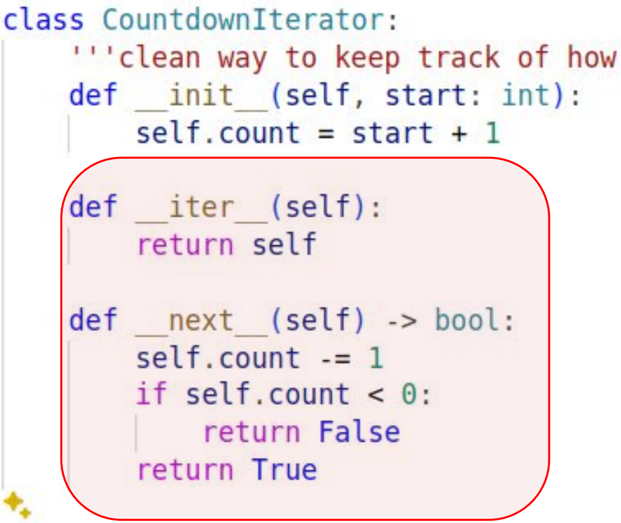
```
annotate.start()
annotate.historic_requests(self.requests_nodes)
annotate.end()
annotate.new_request(request_pickup, request_dropoff)
```

Define smart getter methods to make it more readable

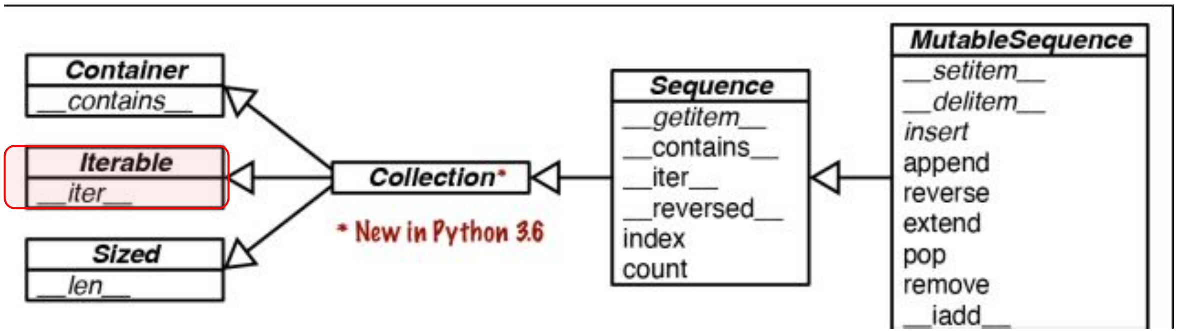


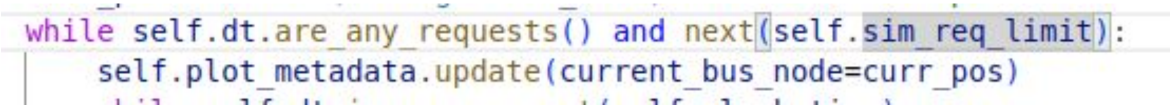
```
class Request(NamedTuple):
    '''imuutable, transformed row of historic data required for our
    baseline algorithm'''
    node_pickup: int
    node_dropoff: int
    pickup_time: Time
    passengers: int  # TODO change export function to accommodate for this
    id: int

    def get_pickup(self) -> 'PathNode':
        '''get the pickup node as the PathNode'''
        return PathNode(self.node_pickup)
```

Pythonic data model allows for using common operations on specific classes



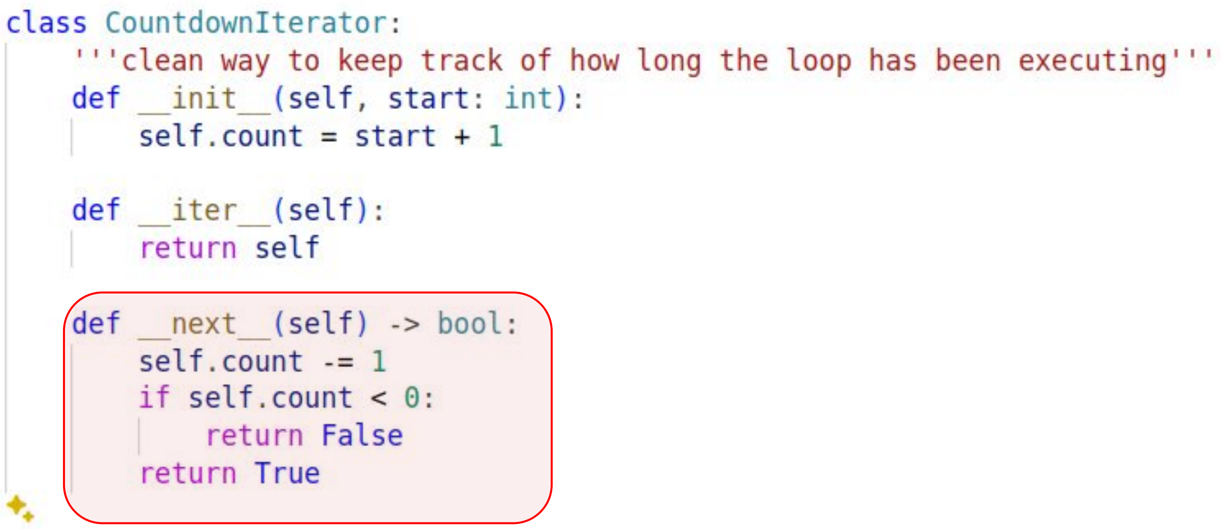
```
class CountdownIterator:
    '''clean way to keep track of how long the loop has been executing'''
    def __init__(self, start: int):
        self.count = start + 1

    def __iter__(self):
        return self

    def __next__(self) -> bool:
        self.count -= 1
        if self.count < 0:
            return False
        return True
```



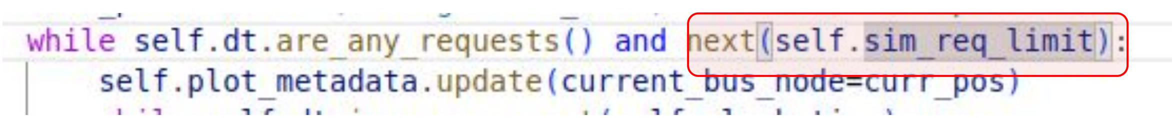


```
while self.dt.are_any_requests() and next(self.sim_req_limit):
    self.plot_metadata.update(current_bus_node=curr_pos)
```

Operator overloading for better customisation of standard methods

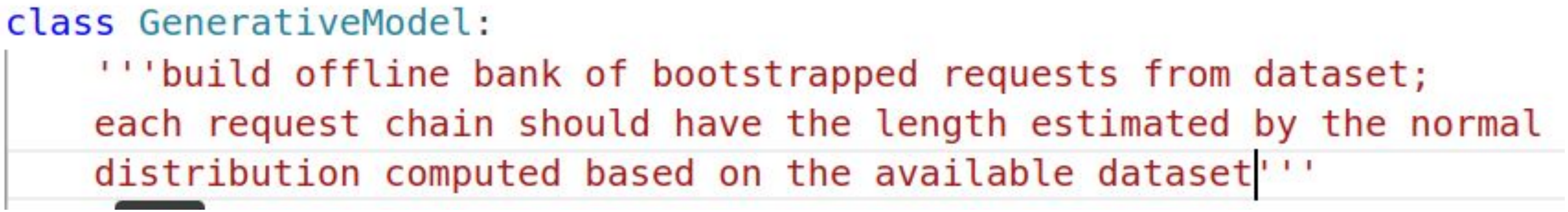


```python
class CountdownIterator:
    '''clean way to keep track of how long the loop has been executing'''
    def __init__(self, start: int):
        self.count = start + 1

    def __iter__(self):
        return self

    def __next__(self) -> bool:
        self.count -= 1
        if self.count < 0:
            return False
        return True
```



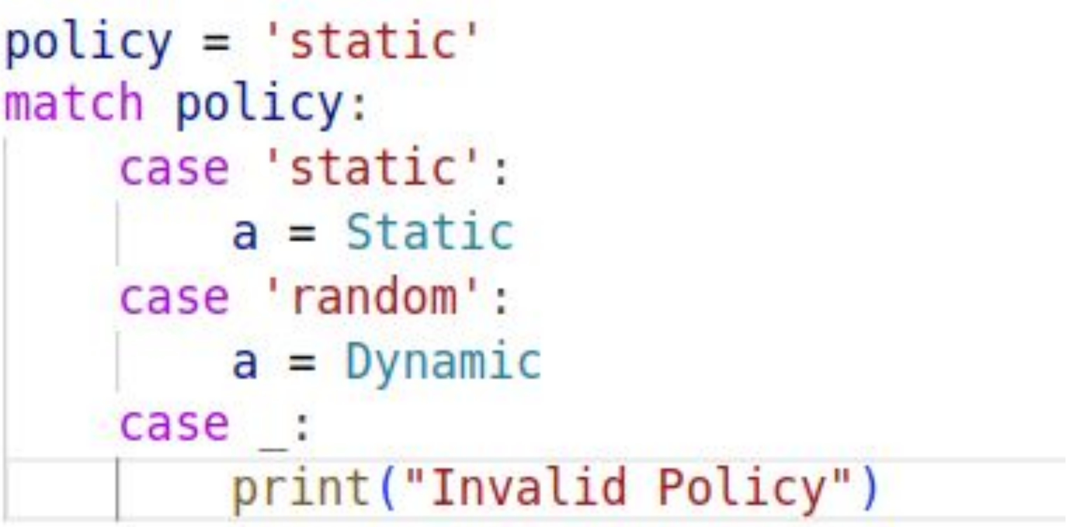
```python
while self.dt.are_any_requests() and next(self.sim_req_limit):
    self.plot_metadata.update(current_bus_node=curr_pos)
```

Good docstrings are key - image explaining this to you in future



```python
class GenerativeModel:
    '''build offline bank of bootstrapped requests from dataset;
    each request chain should have the length estimated by the normal
    distribution computed based on the available dataset'''
```

Match-case is the improved if-elif-else due to pattern matching

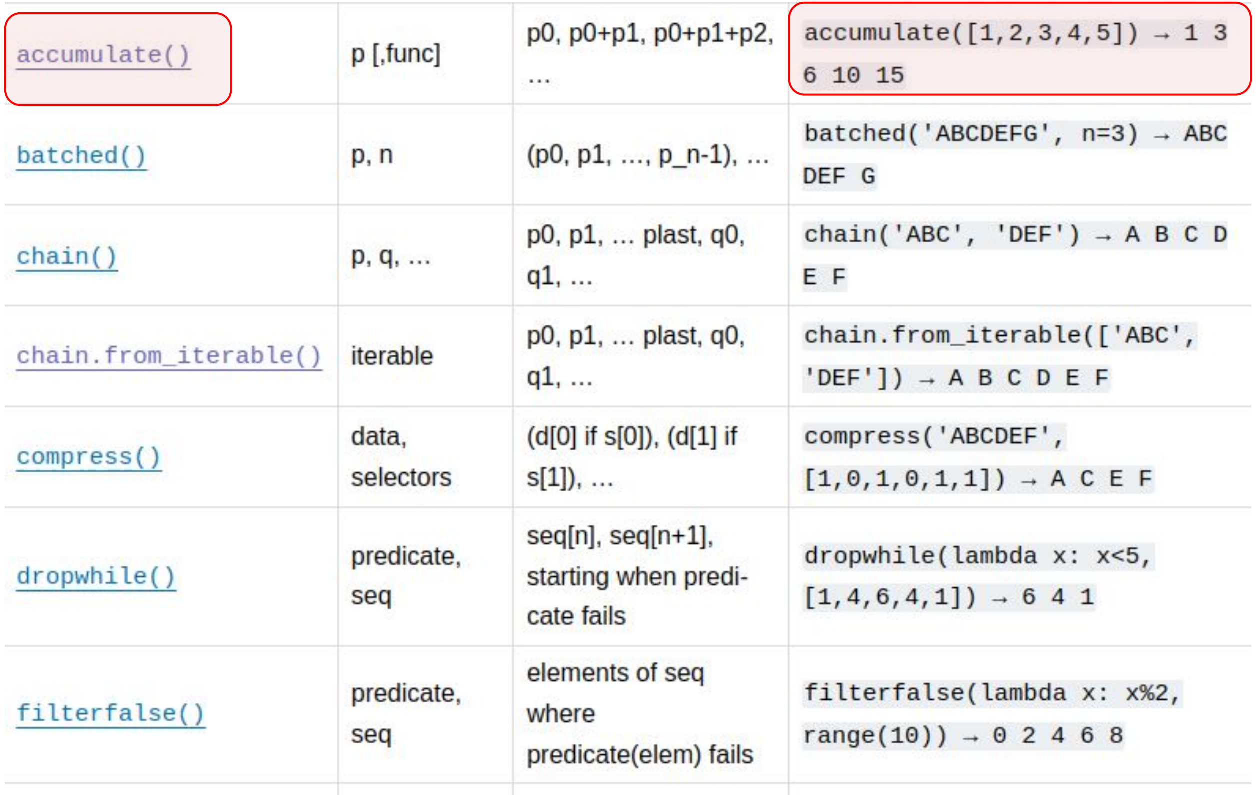


```
policy = 'static'
match policy:
    case 'static':
        a = Static
    case 'random':
        a = Dynamic
    case _:
        print("Invalid Policy")
```

Python Standard Library

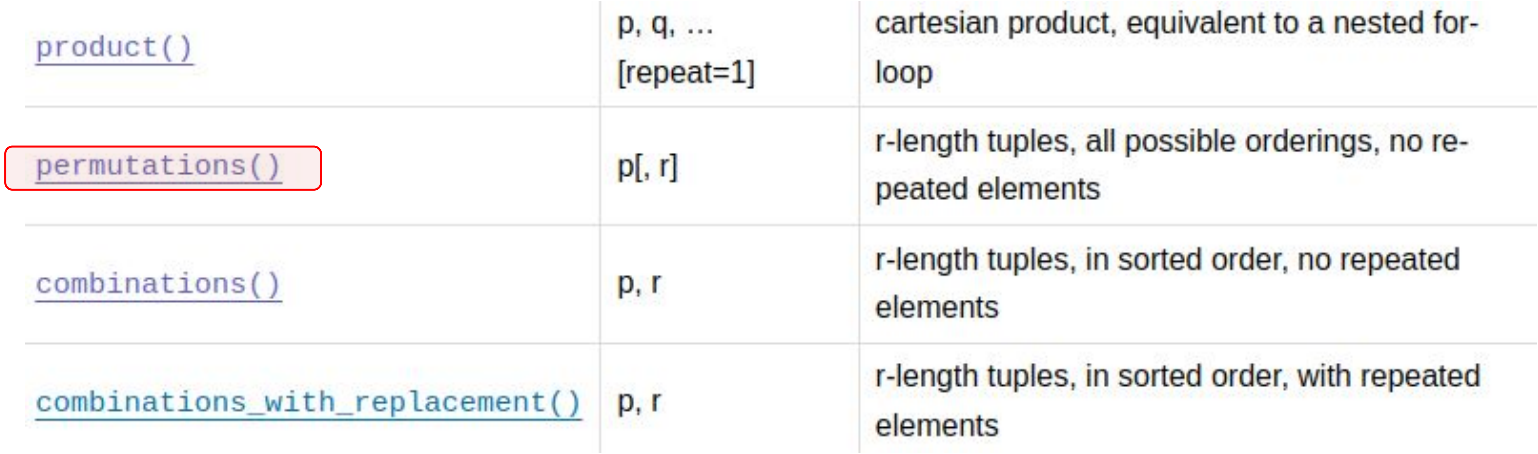
Itertools, functools, collections

Know python standard library well - itertools, functools, collections…



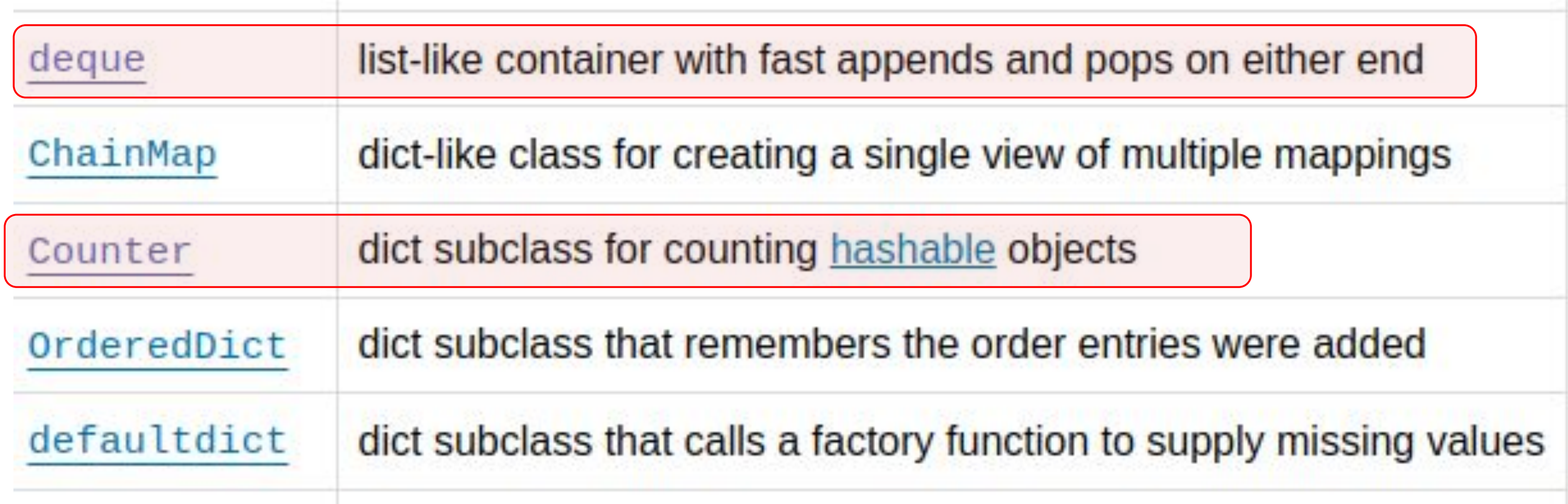
accumulate()	p [,func]	p0, p0+p1, p0+p1+p2, …	accumulate([1,2,3,4,5]) → 1 3 6 10 15
batched()	p, n	(p0, p1, …, p_n-1), …	batched('ABCDEFG', n=3) → ABC DEF G
chain()	p, q, …	p0, p1, … plast, q0, q1, …	chain('ABC', 'DEF') → A B C D E F
chain.from_iterable()	iterable	p0, p1, … plast, q0, q1, …	chain.from_iterable(['ABC', 'DEF']) → A B C D E F
compress()	data, selectors	(d[0] if s[0]), (d[1] if s[1]), …	compress('ABCDEF', [1,0,1,0,1,1]) → A C E F
dropwhile()	predicate, seq	seq[n], seq[n+1], starting when predicate fails	dropwhile(lambda x: x<5, [1,4,6,4,1]) → 6 4 1
filterfalse()	predicate, seq	elements of seq where predicate(elem) fails	filterfalse(lambda x: x%2, range(10)) → 0 2 4 6 8

and….



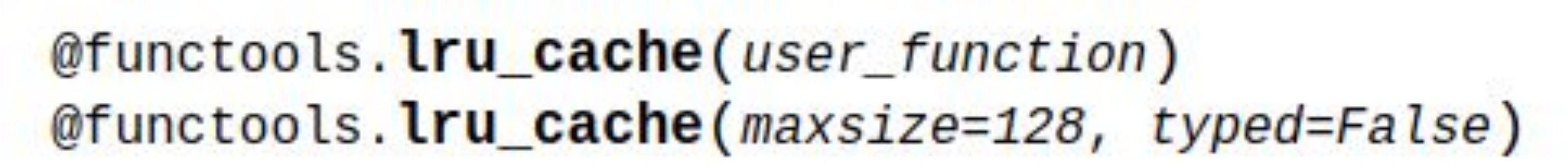
product()	p, q, … [repeat=1]	cartesian product, equivalent to a nested for-loop
permutations()	p[, r]	r-length tuples, all possible orderings, no repeated elements
combinations()	p, r	r-length tuples, in sorted order, no repeated elements
combinations_with_replacement()	p, r	r-length tuples, in sorted order, with repeated elements

collections



deque	list-like container with fast appends and pops on either end
ChainMap	dict-like class for creating a single view of multiple mappings
Counter	dict subclass for counting hashable objects
OrderedDict	dict subclass that remembers the order entries were added
defaultdict	dict subclass that calls a factory function to supply missing values

Functools - inbuild caching support



```
@functools.lru_cache(user_function)
@functools.lru_cache(maxsize=128, typed=False)
```

Worked example

Let's apply every single topic for make SOTA code in python

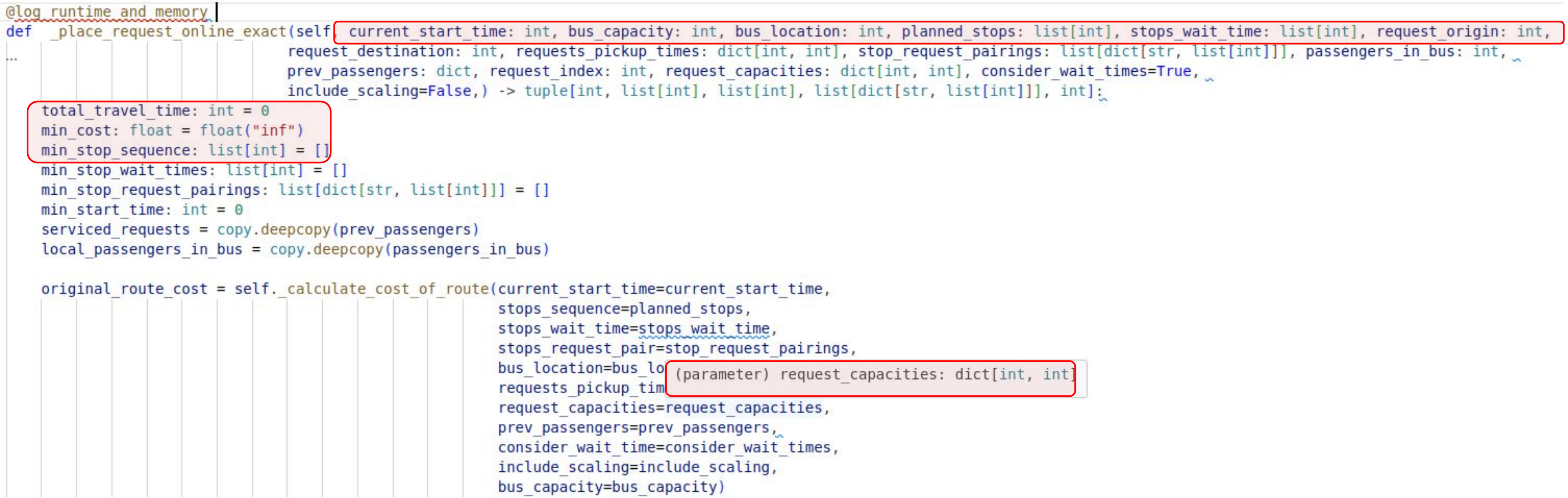
Structure

- Typed Python
- Interfaces, ABC and @abstractmethod
- Profiling and Logging
- Generators and list comprehensions
- @staticmethod, @property, @property.setter, @classmethod
- Use global variables in configuration files
- Python Standard Library
- Python/General OOP practices

- Worked example/Workshop

Worked example: Simple starting code

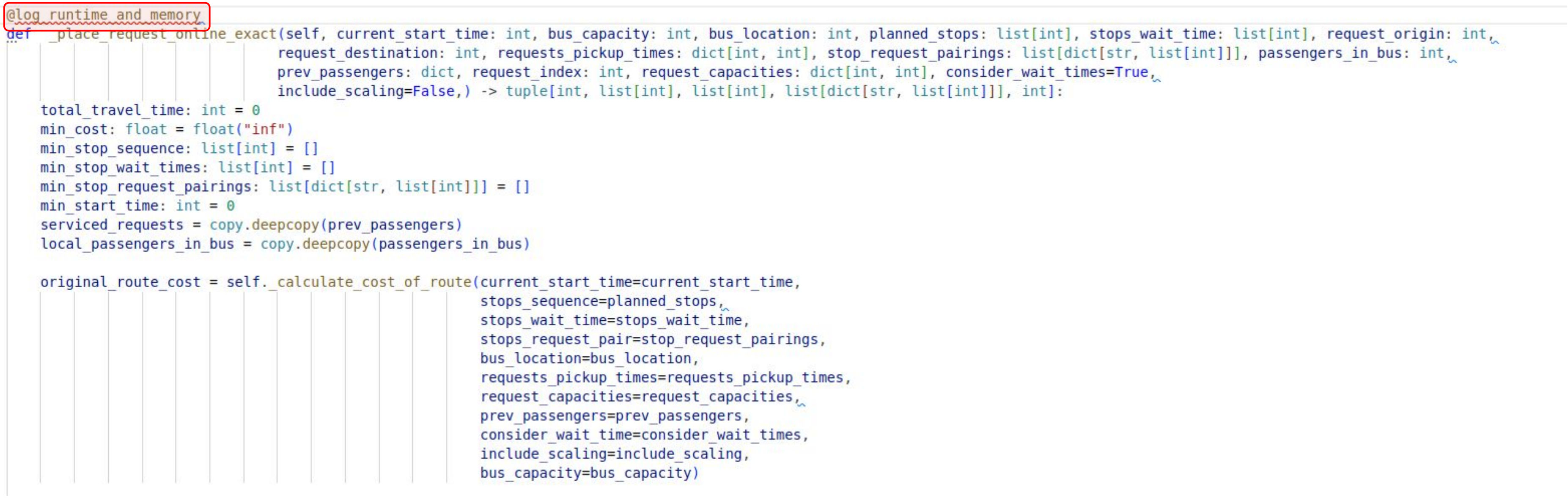
Worked example: add base types



```
@log_runtime_and_memory
def _place_request_online_exact(self, current_start_time: int, bus_capacity: int, bus_location: int, planned_stops: list[int], stops_wait_time: list[int], request_origin: int,
                                request_destination: int, requests_pickup_times: dict[int, int], stop_request_pairings: list[dict[str, list[int]]], passengers_in_bus: int,
                                prev_passengers: dict, request_index: int, request_capacities: dict[int, int], consider_wait_times=True,
                                include_scaling=False,) -> tuple[int, list[int], list[int], list[dict[str, list[int]]], int]:
    total_travel_time: int = 0
    min_cost: float = float("inf")
    min_stop_sequence: list[int] = []
    min_stop_wait_times: list[int] = []
    min_stop_request_pairings: list[dict[str, list[int]]] = []
    min_start_time: int = 0
    serviced_requests = copy.deepcopy(prev_passengers)
    local_passengers_in_bus = copy.deepcopy(passengers_in_bus)

    original_route_cost = self._calculate_cost_of_route(current_start_time=current_start_time,
                                                        stops_sequence=planned_stops,
                                                        stops_wait_time=stops_wait_time,
                                                        stops_request_pair=stop_request_pairings,
                                                        bus_location=bus_lo
                                                        requests_pickup_tim
                                                        request_capacities=request_capacities,
                                                        prev_passengers=prev_passengers,
                                                        consider_wait_time=consider_wait_times,
                                                        include_scaling=include_scaling,
                                                        bus_capacity=bus_capacity)
```

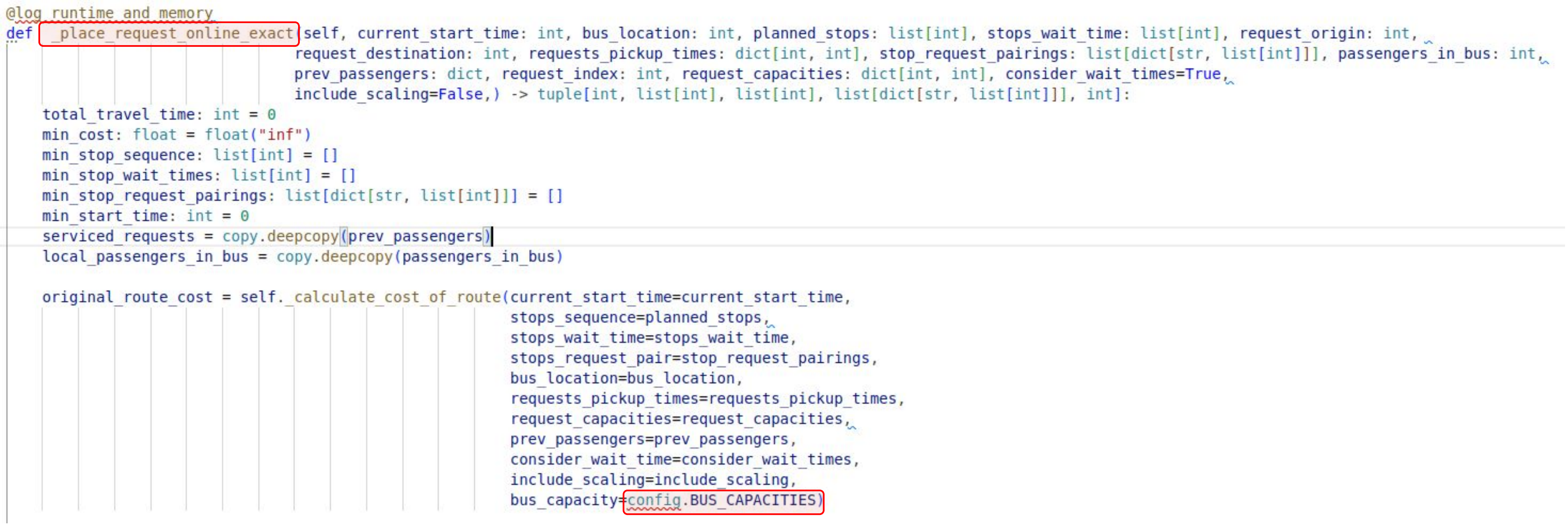
```
(parameter) request_capacities: dict[int, int]
```

Worked example: profile to get a sense of time and memory (optional)

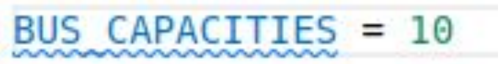


```python
@log_runtime_and_memory
def _place_request_online_exact(self, current_start_time: int, bus_capacity: int, bus_location: int, planned_stops: list[int], stops_wait_time: list[int], request_origin: int,
                                request_destination: int, requests_pickup_times: dict[int, int], stop_request_pairings: list[dict[str, list[int]]], passengers_in_bus: int,
                                prev_passengers: dict, request_index: int, request_capacities: dict[int, int], consider_wait_times=True,
                                include_scaling=False,) -> tuple[int, list[int], list[int], list[dict[str, list[int]]], int]:
    total_travel_time: int = 0
    min_cost: float = float("inf")
    min_stop_sequence: list[int] = []
    min_stop_wait_times: list[int] = []
    min_stop_request_pairings: list[dict[str, list[int]]] = []
    min_start_time: int = 0
    serviced_requests = copy.deepcopy(prev_passengers)
    local_passengers_in_bus = copy.deepcopy(passengers_in_bus)

    original_route_cost = self._calculate_cost_of_route(current_start_time=current_start_time,
                                                        stops_sequence=planned_stops,
                                                        stops_wait_time=stops_wait_time,
                                                        stops_request_pair=stop_request_pairings,
                                                        bus_location=bus_location,
                                                        requests_pickup_times=requests_pickup_times,
                                                        request_capacities=request_capacities,
                                                        prev_passengers=prev_passengers,
                                                        consider_wait_time=consider_wait_times,
                                                        include_scaling=include_scaling,
                                                        bus_capacity=bus_capacity)
```

Worked example: add global configuration variables

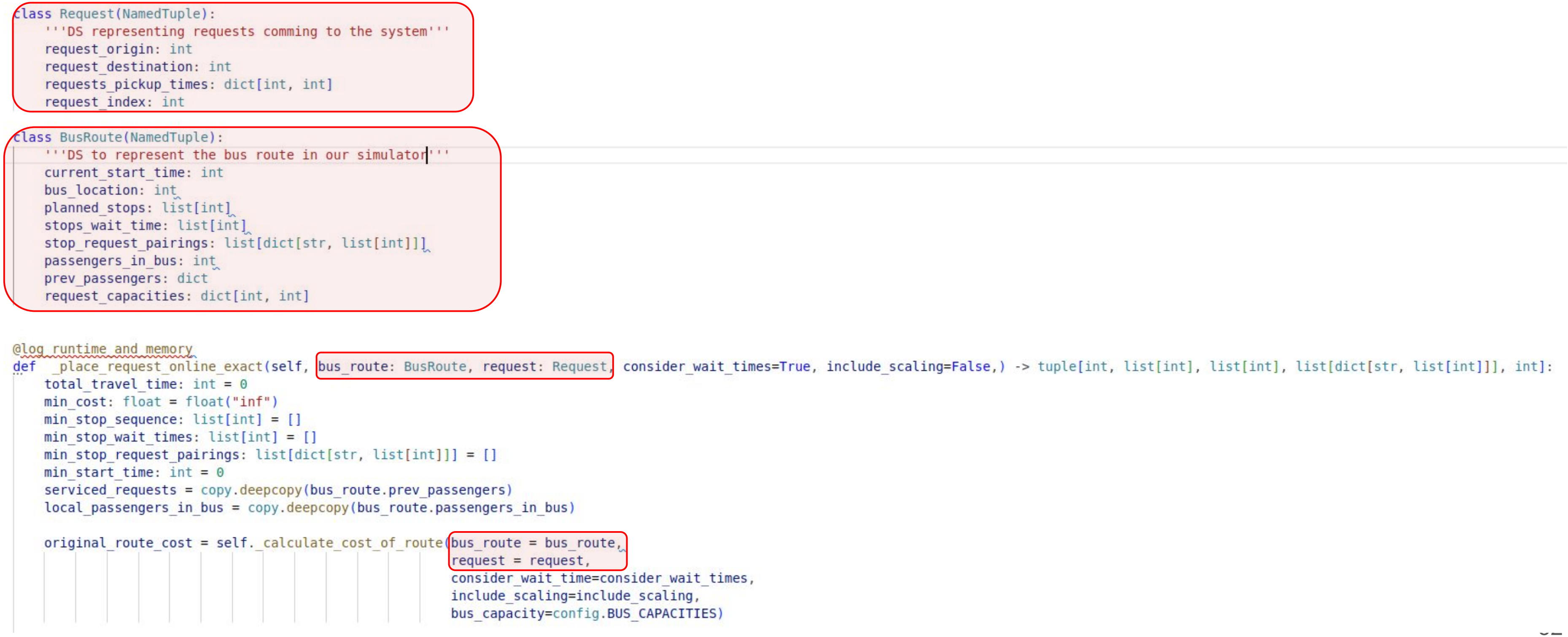


```
@log_runtime_and_memory
def _place_request_online_exact(self, current_start_time: int, bus_location: int, planned_stops: list[int], stops_wait_time: list[int], request_origin: int,
                                request_destination: int, requests_pickup_times: dict[int, int], stop_request_pairings: list[dict[str, list[int]]], passengers_in_bus: int,
                                prev_passengers: dict, request_index: int, request_capacities: dict[int, int], consider_wait_times=True,
                                include_scaling=False,) -> tuple[int, list[int], list[int], list[dict[str, list[int]]], int]:
    total_travel_time: int = 0
    min_cost: float = float("inf")
    min_stop_sequence: list[int] = []
    min_stop_wait_times: list[int] = []
    min_stop_request_pairings: list[dict[str, list[int]]] = []
    min_start_time: int = 0
    serviced_requests = copy.deepcopy(prev_passengers)
    local_passengers_in_bus = copy.deepcopy(passengers_in_bus)

    original_route_cost = self._calculate_cost_of_route(current_start_time=current_start_time,
                                                        stops_sequence=planned_stops,
                                                        stops_wait_time=stops_wait_time,
                                                        stops_request_pair=stop_request_pairings,
                                                        bus_location=bus_location,
                                                        requests_pickup_times=requests_pickup_times,
                                                        request_capacities=request_capacities,
                                                        prev_passengers=prev_passengers,
                                                        consider_wait_time=consider_wait_times,
                                                        include_scaling=include_scaling,
                                                        bus_capacity=config.BUS_CAPACITIES)
```



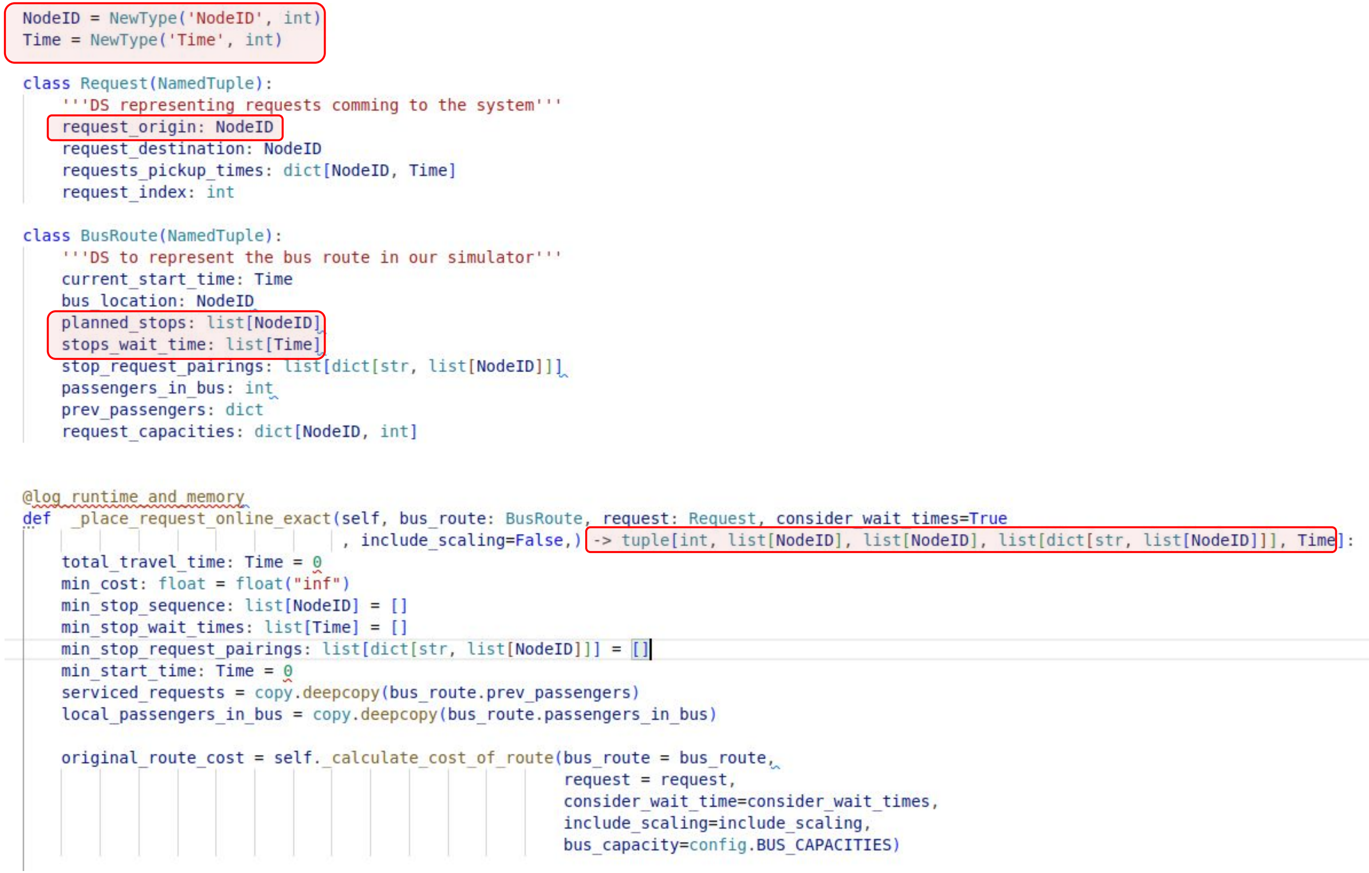
```
BUS_CAPACITIES = 10
```

Worked example: create data classes to improve data representation



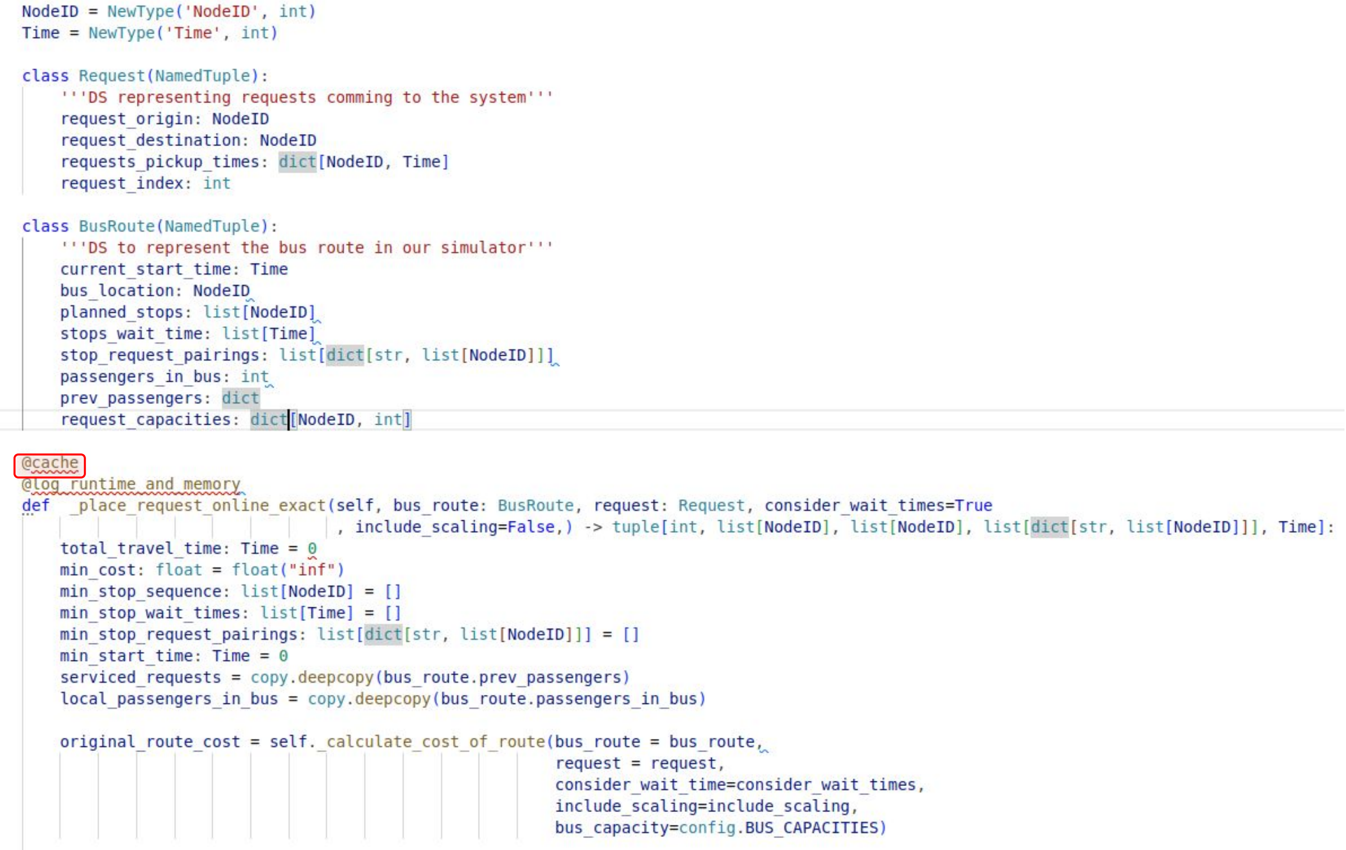
```python
class Request(NamedTuple):
    '''DS representing requests comming to the system'''
    request_origin: int
    request_destination: int
    requests_pickup_times: dict[int, int]
    request_index: int

class BusRoute(NamedTuple):
    '''DS to represent the bus route in our simulator'''
    current_start_time: int
    bus_location: int
    planned_stops: list[int]
    stops_wait_time: list[int]
    stop_request_pairings: list[dict[str, list[int]]]
    passengers_in_bus: int
    prev_passengers: dict
    request_capacities: dict[int, int]


@log_runtime_and_memory
def _place_request_online_exact(self, bus_route: BusRoute, request: Request, consider_wait_times=True, include_scaling=False,) -> tuple[int, list[int], list[int], list[dict[str, list[int]]], int]:
    total_travel_time: int = 0
    min_cost: float = float("inf")
    min_stop_sequence: list[int] = []
    min_stop_wait_times: list[int] = []
    min_stop_request_pairings: list[dict[str, list[int]]] = []
    min_start_time: int = 0
    serviced_requests = copy.deepcopy(bus_route.prev_passengers)
    local_passengers_in_bus = copy.deepcopy(bus_route.passengers_in_bus)

    original_route_cost = self._calculate_cost_of_route(bus_route = bus_route,
                                                        request = request,
                                                        consider_wait_time=consider_wait_times,
                                                        include_scaling=include_scaling,
                                                        bus_capacity=config.BUS_CAPACITIES)
```

Worked example: add intuitive basic types aliases



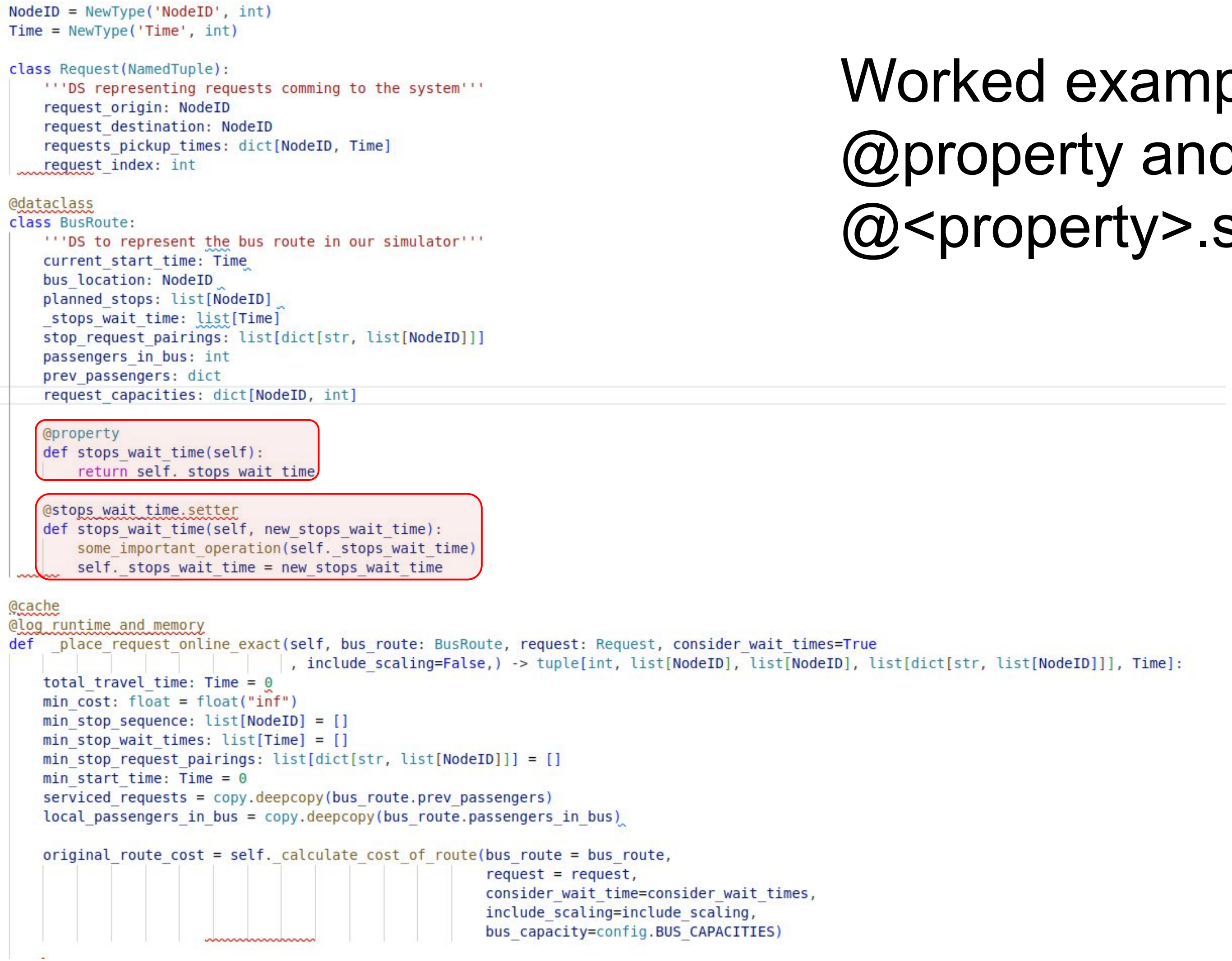
```python
NodeID = NewType('NodeID', int)
Time = NewType('Time', int)

class Request(NamedTuple):
    '''DS representing requests comming to the system'''
    request_origin: NodeID
    request_destination: NodeID
    requests_pickup_times: dict[NodeID, Time]
    request_index: int

class BusRoute(NamedTuple):
    '''DS to represent the bus route in our simulator'''
    current_start_time: Time
    bus_location: NodeID
    planned_stops: list[NodeID]
    stops_wait_time: list[Time]
    stop_request_pairings: list[dict[str, list[NodeID]]]
    passengers_in_bus: int
    prev_passengers: dict
    request_capacities: dict[NodeID, int]


@log_runtime_and_memory
def __place_request_online_exact(self, bus_route: BusRoute, request: Request, consider_wait_times=True
                                 , include_scaling=False,) -> tuple[int, list[NodeID], list[NodeID], list[dict[str, list[NodeID]]], Time]:
    total_travel_time: Time = 0
    min_cost: float = float("inf")
    min_stop_sequence: list[NodeID] = []
    min_stop_wait_times: list[Time] = []
    min_stop_request_pairings: list[dict[str, list[NodeID]]] = []
    min_start_time: Time = 0
    serviced_requests = copy.deepcopy(bus_route.prev_passengers)
    local_passengers_in_bus = copy.deepcopy(bus_route.passengers_in_bus)

    original_route_cost = self._calculate_cost_of_route(bus_route = bus_route,
                                                        request = request,
                                                        consider_wait_time=consider_wait_times,
                                                        include_scaling=include_scaling,
                                                        bus_capacity=config.BUS_CAPACITIES)
```

Worked example: use python STL functools to cache results

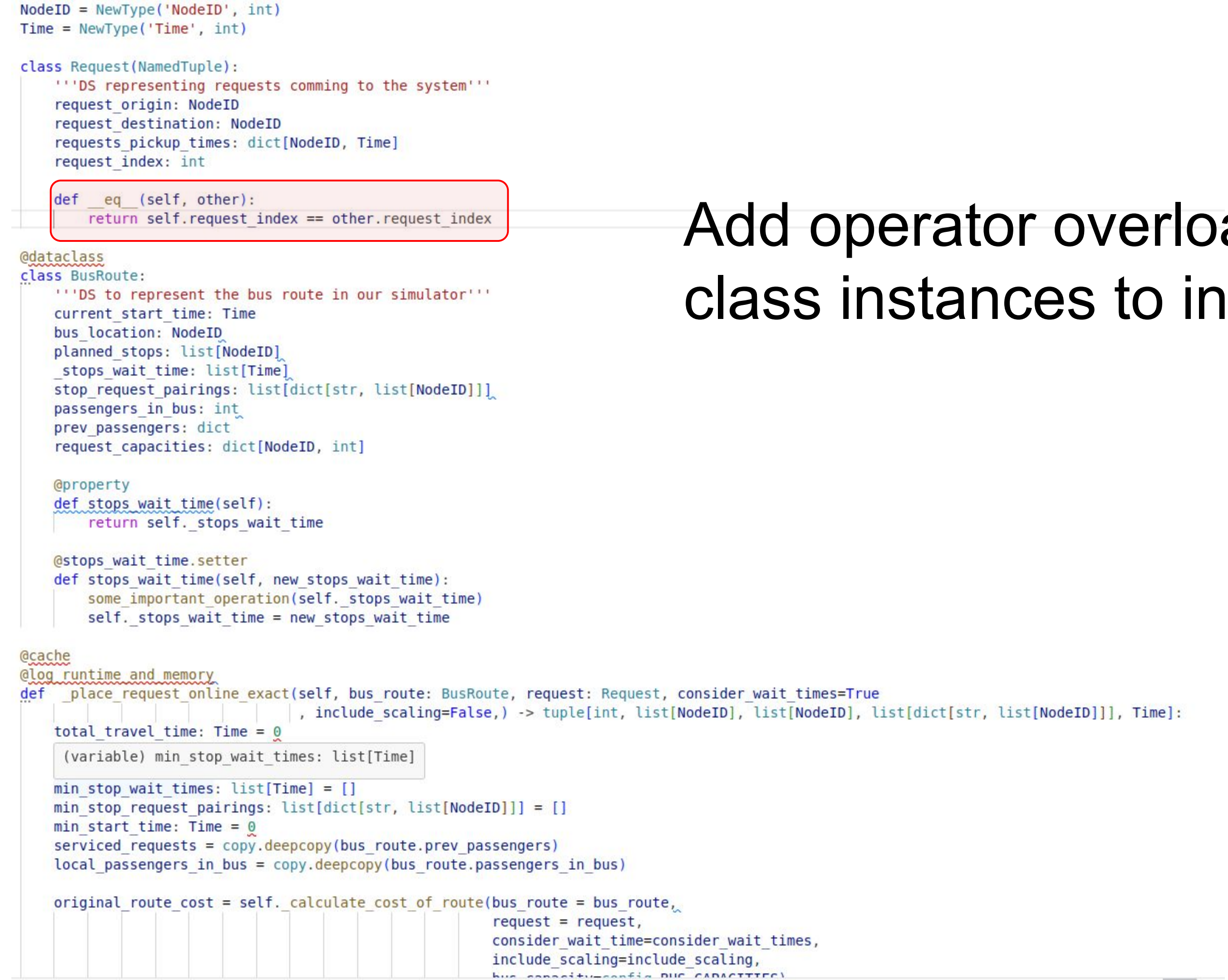


```python
NodeID = NewType('NodeID', int)
Time = NewType('Time', int)

class Request(NamedTuple):
    '''DS representing requests comming to the system'''
    request_origin: NodeID
    request_destination: NodeID
    requests_pickup_times: dict[NodeID, Time]
    request_index: int

class BusRoute(NamedTuple):
    '''DS to represent the bus route in our simulator'''
    current_start_time: Time
    bus_location: NodeID
    planned_stops: list[NodeID]
    stops_wait_time: list[Time]
    stop_request_pairings: list[dict[str, list[NodeID]]]
    passengers_in_bus: int
    prev_passengers: dict
    request_capacities: dict[NodeID, int]

@cache
@log_runtime_and_memory
def __place_request_online_exact(self, bus_route: BusRoute, request: Request, consider_wait_times=True
                                 , include_scaling=False,) -> tuple[int, list[NodeID], list[NodeID], list[dict[str, list[NodeID]]], Time]:
    total_travel_time: Time = 0
    min_cost: float = float("inf")
    min_stop_sequence: list[NodeID] = []
    min_stop_wait_times: list[Time] = []
    min_stop_request_pairings: list[dict[str, list[NodeID]]] = []
    min_start_time: Time = 0
    serviced_requests = copy.deepcopy(bus_route.prev_passengers)
    local_passengers_in_bus = copy.deepcopy(bus_route.passengers_in_bus)

    original_route_cost = self._calculate_cost_of_route(bus_route = bus_route,
                                                        request = request,
                                                        consider_wait_time=consider_wait_times,
                                                        include_scaling=include_scaling,
                                                        bus_capacity=config.BUS_CAPACITIES)
```

Worked example: setup @property and @<property>.setter methods



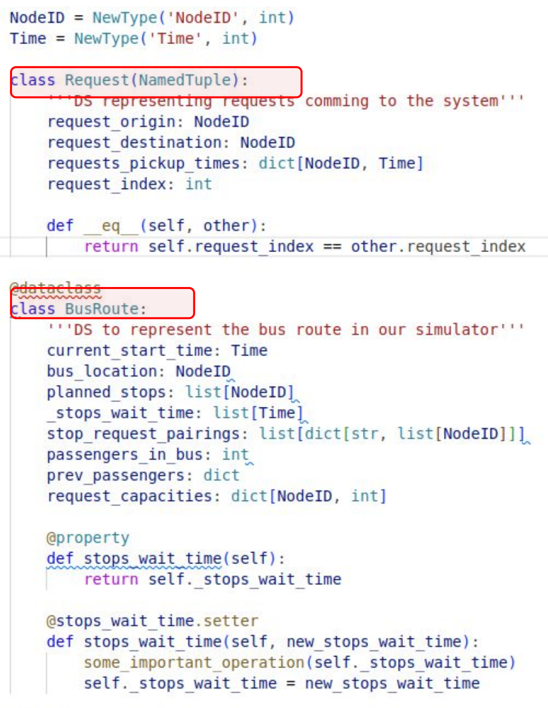
```python
NodeID = NewType('NodeID', int)
Time = NewType('Time', int)

class Request(NamedTuple):
    '''DS representing requests comming to the system'''
    request_origin: NodeID
    request_destination: NodeID
    requests_pickup_times: dict[NodeID, Time]
    request_index: int

@dataclass
class BusRoute:
    '''DS to represent the bus route in our simulator'''
    current_start_time: Time
    bus_location: NodeID
    planned_stops: list[NodeID]
    _stops_wait_time: list[Time]
    stop_request_pairings: list[dict[str, list[NodeID]]]
    passengers_in_bus: int
    prev_passengers: dict
    request_capacities: dict[NodeID, int]

    @property
    def stops_wait_time(self):
        return self._stops_wait_time

    @stops_wait_time.setter
    def stops_wait_time(self, new_stops_wait_time):
        some_important_operation(self._stops_wait_time)
        self._stops_wait_time = new_stops_wait_time

@cache
@log_runtime_and_memory
def  _place_request_online_exact(self, bus_route: BusRoute, request: Request, consider_wait_times=True
                                , include_scaling=False,) -> tuple[int, list[NodeID], list[NodeID], list[dict[str, list[NodeID]]], Time]:
    total_travel_time: Time = 0
    min_cost: float = float("inf")
    min_stop_sequence: list[NodeID] = []
    min_stop_wait_times: list[Time] = []
    min_stop_request_pairings: list[dict[str, list[NodeID]]] = []
    min_start_time: Time = 0
    serviced_requests = copy.deepcopy(bus_route.prev_passengers)
    local_passengers_in_bus = copy.deepcopy(bus_route.passengers_in_bus)

    original_route_cost = self._calculate_cost_of_route(bus_route = bus_route,
                                                         request = request,
                                                         consider_wait_time=consider_wait_times,
                                                         include_scaling=include_scaling,
                                                         bus_capacity=config.BUS_CAPACITIES)
```



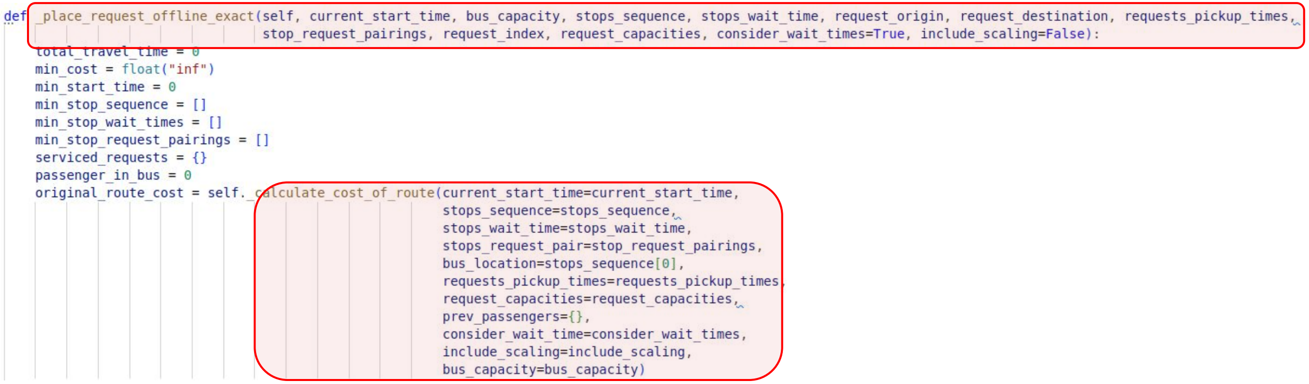
```python
NodeID = NewType('NodeID', int)
Time = NewType('Time', int)

class Request(NamedTuple):
    '''DS representing requests comming to the system'''
    request_origin: NodeID
    request_destination: NodeID
    requests_pickup_times: dict[NodeID, Time]
    request_index: int

    def __eq__(self, other):
        return self.request_index == other.request_index

@dataclass
class BusRoute:
    '''DS to represent the bus route in our simulator'''
    current_start_time: Time
    bus_location: NodeID
    planned_stops: list[NodeID]
    _stops_wait_time: list[Time]
    stop_request_pairings: list[dict[str, list[NodeID]]]
    passengers_in_bus: int
    prev_passengers: dict
    request_capacities: dict[NodeID, int]

    @property
    def stops_wait_time(self):
        return self._stops_wait_time

    @stops_wait_time.setter
    def stops_wait_time(self, new_stops_wait_time):
        some_important_operation(self._stops_wait_time)
        self._stops_wait_time = new_stops_wait_time

@cache
@log_runtime_and_memory
def _place_request_online_exact(self, bus_route: BusRoute, request: Request, consider_wait_times=True
                                 , include_scaling=False,) -> tuple[int, list[NodeID], list[NodeID], list[dict[str, list[NodeID]]], Time]:
    total_travel_time: Time = 0
    (variable) min_stop_wait_times: list[Time]
    min_stop_wait_times: list[Time] = []
    min_stop_request_pairings: list[dict[str, list[NodeID]]] = []
    min_start_time: Time = 0
    serviced_requests = copy.deepcopy(bus_route.prev_passengers)
    local_passengers_in_bus = copy.deepcopy(bus_route.passengers_in_bus)

    original_route_cost = self._calculate_cost_of_route(bus_route = bus_route,
                                                        request = request,
                                                        consider_wait_time=consider_wait_times,
                                                        include_scaling=include_scaling,
                                                        bus_capacity=config.BUS_CAPACITIES)
```

Add operator overloading to allow class instances to interact better

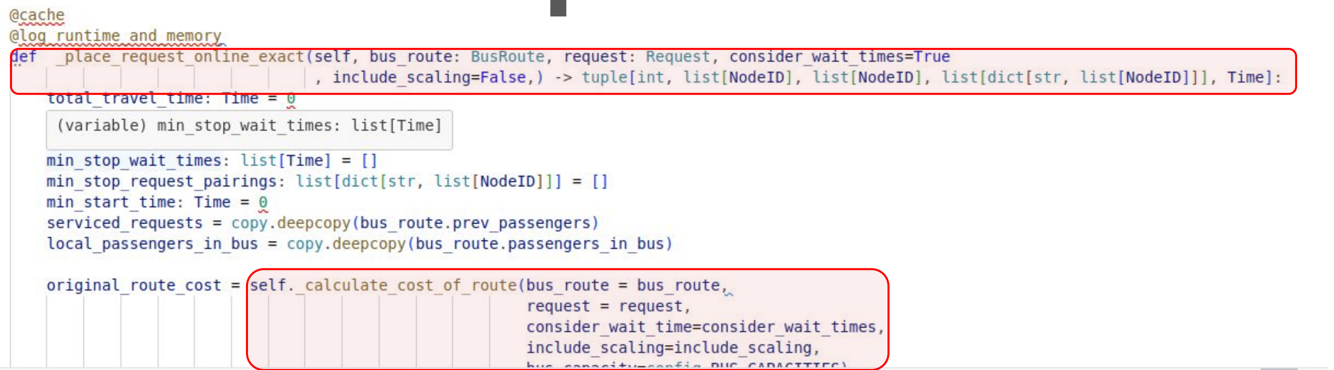
Worked example: comparison



```python
NodeID = NewType('NodeID', int)
Time = NewType('Time', int)

class Request(NamedTuple):
    '''DS representing requests comming to the system'''
    request_origin: NodeID
    request_destination: NodeID
    requests_pickup_times: dict[NodeID, Time]
    request_index: int

    def __eq__(self, other):
        return self.request_index == other.request_index

@dataclass
class BusRoute:
    '''DS to represent the bus route in our simulator'''
    current_start_time: Time
    bus_location: NodeID
    planned_stops: list[NodeID]
    _stops_wait_time: list[Time]
    stop_request_pairings: list[dict[str, list[NodeID]]]
    passengers_in_bus: int
    prev_passengers: dict
    request_capacities: dict[NodeID, int]

    @property
    def stops_wait_time(self):
        return self._stops_wait_time

    @stops_wait_time.setter
    def stops_wait_time(self, new_stops_wait_time):
        some_important_operation(self._stops_wait_time)
        self._stops_wait_time = new_stops_wait_time
```



```python
def _place_request_offline_exact(self, current_start_time, bus_capacity, stops_sequence, stops_wait_time, request_origin, request_destination, requests_pickup_times,
                                 stop_request_pairings, request_index, request_capacities, consider_wait_times=True, include_scaling=False):
    total_travel_time = 0
    min_cost = float("inf")
    min_start_time = 0
    min_stop_sequence = []
    min_stop_wait_times = []
    min_stop_request_pairings = []
    serviced_requests = {}
    passenger_in_bus = 0
    original_route_cost = self._calculate_cost_of_route(current_start_time=current_start_time,
                                                        stops_sequence=stops_sequence,
                                                        stops_wait_time=stops_wait_time,
                                                        stops_request_pair=stop_request_pairings,
                                                        bus_location=stops_sequence[0],
                                                        requests_pickup_times=requests_pickup_times,
                                                        request_capacities=request_capacities,
                                                        prev_passengers={},
                                                        consider_wait_time=consider_wait_times,
                                                        include_scaling=include_scaling,
                                                        bus_capacity=bus_capacity)
```



```python
@cache
@log_runtime_and_memory
def _place_request_online_exact(self, bus_route: BusRoute, request: Request, consider_wait_times=True
                                , include_scaling=False,) -> tuple[int, list[NodeID], list[NodeID], list[dict[str, list[NodeID]]], Time]:
    total_travel_time: Time = 0
    (variable) min_stop_wait_times: list[Time]
    min_stop_wait_times: list[Time] = []
    min_stop_request_pairings: list[dict[str, list[NodeID]]] = []
    min_start_time: Time = 0
    serviced_requests = copy.deepcopy(bus_route.prev_passengers)
    local_passengers_in_bus = copy.deepcopy(bus_route.passengers_in_bus)

    original_route_cost = self._calculate_cost_of_route(bus_route = bus_route,
                                                        request = request,
                                                        consider_wait_time=consider_wait_times,
                                                        include_scaling=include_scaling,
                                                        bus_capacity=config.BUS_CAPACITIES)
```

Thank you for your attention!

igor.sadalski@gmail.com