```
import numpy as np
import random
import pandas as pd
from datetime import datetime
from dataclasses import dataclass, field
from typing import Callable
```

```
from google.colab import drive

drive.mount('/content/drive')
```

```
    Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
trip_stats_df = pd.read_csv('/content/drive/MyDrive/trip_stats.csv')
start_station_df = pd.read_csv('/content/drive/MyDrive/start_station_probs.csv')
trip_stats_df
```

	start	end	count	mean	std
0	11 St & Washington St	11 St & Washington St	142	25.929108	39.186350
1	11 St & Washington St	12 St & Sinatra Dr N	44	56.655303	149.709313
2	11 St & Washington St	14 St Ferry - 14 St & Shipyard Ln	48	12.481597	16.335279
3	11 St & Washington St	4 St & Grand St	47	7.348582	2.465807
4	11 St & Washington St	6 St & Grand St	25	5.890000	1.983590
...	...	...	...	...	...
5141	Willow Ave & 12 St	Stevens - River Ter & 6 St	23	8.784783	2.983604
5142	Willow Ave & 12 St	Vesey Pl & River Terrace	1	16.250000	NaN
5143	Willow Ave & 12 St	Warren St	7	26.807143	5.739514
5144	Willow Ave & 12 St	Washington St	11	35.895455	29.392104

```
start_station_df
```

	Unnamed: 0	start_station_name
0	South Waterfront Walkway - Sinatra Dr & 1 St	0.044679
1	Grove St PATH	0.043504
2	Hoboken Terminal - Hudson St & Hudson Pl	0.033629
3	Hoboken Terminal - River St & Hudson Pl	0.029832
4	Newport Pkwy	0.027035
...	...	...
76	Dey St	0.002670
77	Jackson Square	0.001816
78	Bergen Ave & Stegman St	0.001457
79	Grant Ave & MLK Dr	0.000689
80	JCBS Depot	0.000010

81 rows × 2 columns

```
start_station_df = start_station_df.rename(columns={"Unnamed: 0": 'start_station_
start_station_df
```

	start_station_name	probability
0	South Waterfront Walkway - Sinatra Dr & 1 St	0.044679
1	Grove St PATH	0.043504
2	Hoboken Terminal - Hudson St & Hudson Pl	0.033629
3	Hoboken Terminal - River St & Hudson Pl	0.029832
4	Newport Pkwy	0.027035
...	...	...
76	Dey St	0.002670
77	Jackson Square	0.001816
78	Bergen Ave & Stegman St	0.001457
79	Grant Ave & MLK Dr	0.000689
80	JCBS Depot	0.000010

81 rows × 2 columns

```
riders = 3500
lam = 2.38
mu = 2.78
sigma = 0.619
stations = 81
initial_bikes = 10
max_bikes = 10
```

```
locations_and_probs = []
locations = []
station_count = len(start_station_df)
for i in range(station_count):
  tup = (start_station_df['start_station_name'].loc[start_station_df.index[i]],
         start_station_df['probability'].loc[start_station_df.index[i]])
  locations_and_probs.append(tup)
  locations.append(start_station_df['start_station_name'].loc[start_station_df.in

locations_and_probs = sorted(locations_and_probs, key=lambda location: location[0
print(locations_and_probs)
locations = sorted(locations)
print(locations)
```

```
[('11 St & Washington St', 0.0176445454368894), ('12 St & Sinatra Dr N', 0.02
['11 St & Washington St', '12 St & Sinatra Dr N', '14 St Ferry - 14 St & Ship
```

```
start_probabilities = []
for i in range(81):
  start_probabilities.append(locations_and_probs[i][1])
start_probabilities
```

```
 0.0107595797160558,
 0.0123327312629882,
 0.0192856726679484,
 0.0128668259239837,
 0.0133814989609431,
 0.0040396977995299,
 0.0026704733049777,
 0.0075355901260463,
 0.0071083143972498,
 0.0056225601584804,
 0.0033016760861543,
 0.012128804210608,
 0.0132164151566353,
 0.0006894676532851,
 0.0435044378410922,
 0.0229854920468449,
 0.01866418069879,
 0.0053215249859193,
 0.0072539765775214,
 0.0236846705121482,
 0.0336285420186836,
 0.0298316145196061,
 0.0169550777836042,
 0.0109052418963273,
 9.710812018100952e-06,
 0.0018159218473848,
 0.0093806444094855,
 0.0072248441414671,
 0.0088465497484899,
 0.0074870360659558,
 0.0051078871215211,
 0.0202567538697585,
 0.0053118141739012,
 0.015488745168871,
 0.0087008875682184,
 0.0073899279457748,
 0.0120025636543727,
 0.0235292975198586,
 0.0090601876128881,
 0.0099438715065353,
 0.0066324846083629,
 0.0056128493464623,
 0.0099632931305715,
 0.0143040261026627,
 0.0254714599234788,
 0.027034900658393,
 0.0064771116160733,
 0.0120219852784089,
 0.0073607955097205,
 0.0074384820058653,
 0.017032764279749,
 0.0446794460952824,
 0.0092835362893045,
 0.0073996387577929,
 0.0027287381770863,
 0.0125657907514226,
 0.0146050612752238,
 0.0172658237681834,
 0.0099924255666258]
```

```
df = trip_stats_df[trip_stats_df['start'] == locations[0]]
df
```

	start	end	count	mean	std
0	11 St & Washington St	11 St & Washington St	142	25.929108	39.186350
1	11 St & Washington St	12 St & Sinatra Dr N	44	56.655303	149.709313
2	11 St & Washington St	14 St Ferry - 14 St & Shipyard Ln	48	12.481597	16.335279
3	11 St & Washington St	4 St & Grand St	47	7.348582	2.465807
4	11 St & Washington St	6 St & Grand St	25	5.890000	1.983590
...	...	...	...	...	...
59	11 St & Washington St	Van Vorst Park	3	30.083333	2.953717
60	11 St & Washington St	Warren St	16	19.421875	7.679571
61	11 St & Washington St	Washington St	16	36.632292	31.654605
62	11 St & Washington St	West St & Chambers St	1	28.700000	NaN
63	11 St & Washington St	Willow Ave & 12 St	32	13.725521	22.164499

64 rows × 5 columns

```
new_location_probabilities = []
for i in range(81):
  df = trip_stats_df[trip_stats_df['start'] == locations[i]]
  new_location_probabilities.append(df['count'].to_numpy()/df['count'].to_numpy()
new_location_probabilities
```

```
            0.0045283 , 0.0045283 , 0.00301887, 0.01886792, 0.00226415,
            0.0045283 , 0.0045283 , 0.06490566, 0.00301887, 0.00528302,
            0.01962264, 0.01886792, 0.02566038, 0.02566038, 0.07245283,
            0.05056604, 0.00150943, 0.02264151, 0.0045283 , 0.00377358,
            0.00226415, 0.01660377, 0.00377358, 0.00301887, 0.04377358,
            0.0354717 , 0.04981132]),
     array([0.01669086, 0.02249637, 0.04281567, 0.02539913, 0.00072569,
            0.01741655, 0.01741655, 0.02612482, 0.02394775, 0.00870827,
            0.04644412, 0.00072569, 0.02394775, 0.00072569, 0.00145138,
            0.03338171, 0.10812772, 0.01959361, 0.03701016, 0.00290276,
            0.06023222, 0.00072569, 0.00580552, 0.00072569, 0.0275762 ,
            0.00217707, 0.00580552, 0.00507983, 0.00580552, 0.00145138,
            0.0137881 , 0.07764877, 0.06531205, 0.02830189, 0.00072569,
            0.00072569, 0.00145138, 0.02685051, 0.01451379, 0.01596517,
            0.00507983. 0.00145138. 0.01306241. 0.00072569. 0.00072569.
```

```
             0.00072569, 0.00362845, 0.01306241, 0.00217707, 0.0065312 ,
             0.00072569, 0.06966618, 0.02249637, 0.00943396, 0.00072569,
             0.00362845, 0.00217707, 0.00725689]),
      array([0.00240385, 0.00240385, 0.00961538, 0.00240385, 0.03125   ,
             0.02163462, 0.00240385, 0.00480769, 0.01923077, 0.00240385,
             0.00480769, 0.10336538, 0.00721154, 0.00240385, 0.00480769,
             0.01923077, 0.00240385, 0.11778846, 0.02163462, 0.00721154,
             0.00240385, 0.01201923, 0.00240385, 0.15865385, 0.03605769,
             0.00961538, 0.01923077, 0.00480769, 0.00240385, 0.00240385,
             0.00240385, 0.02403846, 0.00961538, 0.00721154, 0.00480769,
             0.00721154, 0.03846154, 0.00240385, 0.06009615, 0.00480769,
             0.00240385, 0.01442308, 0.01442308, 0.04807692, 0.00480769,
             0.00961538, 0.00480769, 0.01923077, 0.00480769, 0.00480769,
             0.01201923, 0.00240385, 0.00480769, 0.02163462, 0.00480769,
             0.00721154, 0.00961538, 0.00721154, 0.00240385]),
      array([0.00727273, 0.02909091, 0.00363636, 0.00363636, 0.00727273,
             0.00727273, 0.01090909, 0.01818182, 0.01454545, 0.00727273,
             0.00363636, 0.00363636, 0.01454545, 0.08363636, 0.00363636,
             0.00363636, 0.00363636, 0.01090909, 0.01090909, 0.02545455,
             0.00363636, 0.02909091, 0.00363636, 0.00363636, 0.00363636,
             0.00363636, 0.27272727, 0.05454545, 0.00727273, 0.00727273,
             0.00363636, 0.02181818, 0.00363636, 0.03636364, 0.00727273,
             0.00727273, 0.01818182, 0.00363636, 0.01090909, 0.00363636,
             0.02909091, 0.09090909, 0.00363636, 0.05818182, 0.00727273,
             0.00727273, 0.02545455]),
      array([0.00257732, 0.0064433 , 0.00257732, 0.00128866, 0.00128866,
             0.01546392, 0.02190722, 0.00386598, 0.00257732, 0.00515464,
             0.01804124, 0.0128866 , 0.00386598, 0.03221649, 0.0064433 ,
             0.00128866, 0.00128866, 0.0257732 , 0.0193299 , 0.00128866,
             0.00128866, 0.07860825, 0.0064433 , 0.01159794, 0.0064433 ,
             0.0128866 , 0.31056701, 0.02706186, 0.02190722, 0.00257732,
             0.00386598, 0.01030928, 0.0257732 , 0.00773196, 0.00128866,
             0.01804124, 0.0064433 , 0.00128866, 0.00773196, 0.0064433 ,
             0.0128866 , 0.00773196, 0.00128866, 0.00902062, 0.05025773,
             0.00515464, 0.01159794, 0.0128866 , 0.0064433 , 0.00773196,
             0.01159794, 0.01417526, 0.01159794, 0.00257732, 0.01675258,
             0.0064433 , 0.00257732, 0.00386598, 0.00386598, 0.00386598,
             0.00386598, 0.01546392, 0.01159794, 0.0128866 ]),
      array([0.00273224, 0.00409836, 0.00409836, 0.00409836, 0.00136612,
             0.00136612, 0.00136612, 0.00136612, 0.00136612, 0.00273224,
             0.00136612, 0.00136612, 0.01092896, 0.00136612, 0.02322404,
             0.00956284, 0.00136612, 0.00136612, 0.06967213, 0.02459016,
             0.00136612, 0.00273224, 0.00409836, 0.11748634, 0.00136612,
             0.00136612, 0.0204918 , 0.04918033, 0.01092896, 0.03551913,
             0.00819672, 0.01092896, 0.01092896, 0.00136612, 0.01229508,
```

```
new_location_indices = []
for i in range(81):
  df = trip_stats_df[trip_stats_df['start'] == locations[i]]
  new_location_indices.append(df['end'].to_numpy())
new_location_indices
```

```
       'Hoboken Terminal - River St & Hudson Pl', 'Hudson St & 4 St',
       'Jersey & 3rd', 'Jersey & 6th St', 'Lafayette Park',
       'Leonard Gordon Park', 'Liberty Light Rail', 'Madison St & 1 St',
       'Madison St & 10 St', 'Mama Johnson Field - 4 St & Jackson St',
       'Manila & 1st', 'Marin Light Rail', 'Marshall St & 2 St',
       'Monmouth and 6th', 'Morris Canal', 'Newark Ave', 'Newport PATH',
       'Newport Pkwy', 'North Moore St & Greenwich St', 'Oakland Ave',
       'Paulus Hook', 'Pershing Field', 'Riverview Park',
       'South St & Whitehall St',
       'South Waterfront Walkway - Sinatra Dr & 1 St',
       'Southwest Park - Jackson St & Observer Hwy',
       'Stevens - River Ter & 6 St', 'Van Vorst Park',
       'Vesey Pl & River Terrace', 'Warren St', 'Washington St',
       'West St & Chambers St', 'Willow Ave & 12 St'], dtype=object),
 array(['11 St & Washington St', '12 St & Sinatra Dr N',
       '14 St Ferry - 14 St & Shipyard Ln', '4 St & Grand St',
       '6 St & Grand St', '7 St & Monroe St', '8 St & Washington St',
       '9 St HBLR - Jackson St & 8 St', 'Adams St & 11 St',
       'Adams St & 2 St', 'Baldwin at Montgomery',
       'Bergen Ave & Stegman St', 'Bloomfield St & 15 St',
       'Brunswick & 6th', 'Brunswick St', 'Christ Hospital',
       'Church Sq Park - 5 St & Park Ave',
       'City Hall - Washington St & 1 St', 'Clinton St & 7 St',
       'Clinton St & Newark St', 'Columbus Dr at Exchange Pl',
       'Columbus Drive', 'Columbus Park - Clinton St & 9 St',
       'Communipaw & Berry Lane', 'Dey St', 'E 59 St & Madison Ave',
       'Essex Light Rail', 'Glenwood Ave', 'Grand St', 'Grand St & 14 St',
       'Grove St PATH', 'Hamilton Park', 'Harborside', 'Heights Elevator',
       'Hilltop', 'Hoboken Ave at Monmouth St',
       'Hoboken Terminal - Hudson St & Hudson Pl',
       'Hoboken Terminal - River St & Hudson Pl', 'Hudson St & 4 St',
       'JC Medical Center', 'Jersey & 6th St', 'Madison St & 1 St',
       'Madison St & 10 St', 'Mama Johnson Field - 4 St & Jackson St',
       'Manila & 1st', 'Marin Light Rail', 'Marshall St & 2 St',
       'Monmouth and 6th', 'Montgomery St', 'Morris Canal', 'Newark Ave',
       'Newport PATH', 'Newport Pkwy', 'North Moore St & Greenwich St',
       'Oakland Ave', 'Paulus Hook', 'Pershing Field', 'Riverview Park',
       'South Waterfront Walkway - Sinatra Dr & 1 St',
       'Southwest Park - Jackson St & Observer Hwy',
       'Stevens - River Ter & 6 St', 'Van Vorst Park',
       'W Broadway & W Houston St', 'Warren St', 'Washington St',
       'Willow Ave & 12 St'], dtype=object),
 array(['11 St & Washington St', '12 St & Sinatra Dr N',
       '14 St Ferry - 14 St & Shipyard Ln', '4 St & Grand St',
       '5 Corners Library', '6 St & Grand St', '7 St & Monroe St',
       '8 St & Washington St', '9 St HBLR - Jackson St & 8 St',
       'Adams St & 11 St', 'Adams St & 2 St', 'Baldwin at Montgomery',
       'Bloomfield St & 15 St', 'Christ Hospital',
       'Church Sq Park - 5 St & Park Ave', 'City Hall',
       'City Hall - Washington St & 1 St', 'Clinton St & 7 St',
       'Clinton St & Newark St', 'Columbus Dr at Exchange Pl',
       'Columbus Park - Clinton St & 9 St', 'Dey St', 'Dixon Mills',
       'Essex Light Rail', 'Grand St', 'Grand St & 14 St',
       'Grove St PATH', 'Hamilton Park', 'Heights Elevator',
       'Hoboken Ave at Monmouth St',
       'Hoboken Terminal - Hudson St & Hudson Pl',
       'Hoboken Terminal - River St & Hudson Pl', 'Hudson St & 4 St',
       'JC Medical Center', 'Jersey & 3rd', 'Lafayette Park',
       'Leonard Gordon Park', 'Madison St & 1 St', 'Madison St & 10 St',
       'Mama Johnson Field - 4 St & Jackson St', 'Manila & 1st',
```

```
waiting_for_bike = [0]*81
waiting_to_return = [0]*81
bikes_available = [10]*81
now = datetime.now()
current_time = now.strftime("%H:%M:%S")
current_time
```

```
'01:27:58'
```

```
def static_vars(**kwargs):
    def decorate(func):
        for k in kwargs:
            setattr(func, k, kwargs[k])
        return func
    return decorate
@static_vars(t=0)
def now():
    return now.t
print(now())
```

```
0
```

```
def set_time(t_new=0):
    now.t = t_new
    return now()
def get_time():
    return now()
```

Events:

Arrive to pick up bike

Return bike

```
class FutureEventList:
    def __init__(self):
        self.events = []

    def __iter__(self):
        return self

    def __next__(self) -> Event:
        from heapq import heappop
        if self.events:
            return heappop(self.events)
        raise StopIteration

    def __repr__(self) -> str:
        from pprint import pformat
        return pformat(self.events)

    def len(self) -> int:
      return len(self.events)
```

```
class Rider:
  def __init__(self, start_index, arrival_time):
    self.start_index = start_index
    self.arrival_time = arrival_time
    self.end_index = None
    self.bike_rental_time = None
    self.end_time = None
    self.ended = False
    self.return_time = None
    self.rented = False

  def __lt__(self, other):
    return self.arrival_time < other.arrival_time

  def set_end_index(self, index):
    self.end_index = index
  def set_end_time(self, time):
    self.end_time = time
  def set_bike_rental_time(self, time):
    self.bike_rental_time = time



def schedule(e: Event, fev: FutureEventList): # inserts `e` into `fev`
    from heapq import heappush
    heappush(fev.events, e)


@dataclass(order=True)
class Event:
    t: int
    f: Callable=field(compare=False)
    r: Rider
```

```
def initialize_events(size, event_list):
  initial_arrival_times = np.random.exponential(scale=1.0/2.38, size=size)
  initial_locations = np.random.choice(81, size=size, p=start_probabilities)
  initial_arrival_times = np.cumsum(initial_arrival_times)
  #print(initial_arrival_times[size-1])
  for i in range(size):
    rider = Rider(initial_locations[i], initial_arrival_times[i])
    riders.append(rider)
    schedule(Event(rider.arrival_time, arrive, rider), event_list)


def initial_state():
    return {'waiting_for_bike': [[]]*81,
            'waiting_to_return': [[]]*81,
            'bikes_available': [max_bikes]*81,
            'bikes_rented': 0,
            'bikes_returned': 0,
            'bikes_lost': 0,
            'max_waitlist_length': [0]*81,
            'bikes_rented_location': [0]*81}


def simulate(state, event_list, verbose=False):
    i = 0
    for e in event_list:

        set_time(e.t)
        #print(get_time())
        if e.t > 1440:
          break

        i = i+1
        e.f(state, e.r, event_list)
```

```
def arrive(s, rider, fev):
  if s['bikes_available'][rider.start_index] == 0:
    temp = s['waiting_for_bike'][rider.start_index].copy()
    temp.append(rider)
    s['waiting_for_bike'][rider.start_index] = temp

  if len(s['waiting_for_bike'][rider.start_index]) > s['max_waitlist_length'][rid
    s['max_waitlist_length'][rider.start_index] = len(s['waiting_for_bike'][rider

  if s['bikes_available'][rider.start_index] > 0:
    s['bikes_available'][rider.start_index] -= 1

    s['bikes_rented'] += 1
    s['bikes_rented_location'][rider.start_index] += 1
    rider.rented = True
    rider.bike_rental_time = now()
    #get next location and travel time
    travel_time = np.random.lognormal(mean=2.78, sigma=0.619)
    #get a new location, but doesn't necessarily correspond since not all starts
    temp_index = np.random.choice(len(new_location_probabilities[rider.start_inde
    new_location = new_location_indices[rider.start_index][temp_index]
    if new_location in locations:
      new_index = locations.index(new_location)
      rider.end_index = new_index
      rider.return_time = now() + travel_time
      schedule(Event(now() + travel_time, return_bike, rider), fev)
    else:
      s['bikes_lost'] += 1
  if len(s['waiting_to_return'][rider.start_index]) > 0 and s['bikes_available'][

    new_rider = s['waiting_to_return'][rider.start_index].pop(0)
    schedule(Event(now(), return_bike, new_rider), fev)
```

```
def return_bike(s, rider, fev):

  if s['bikes_available'][rider.end_index] == max_bikes:
    temp = s['waiting_to_return'][rider.end_index].copy()
    temp.append(rider)
    s['waiting_to_return'][rider.end_index] = temp
  if s['bikes_available'][rider.end_index] < max_bikes:
    rider.ended = True
    rider.end_time = now()
    s['bikes_available'][rider.end_index] += 1
    s['bikes_returned'] += 1
    if len(s['waiting_for_bike'][rider.end_index]) > 0:
      new_rider = s['waiting_for_bike'][rider.end_index].pop(0)
      schedule(Event(now(), arrive, new_rider), fev)


max_bikes = 10
riders = []
event_list = FutureEventList()
state = initial_state()
print(state)
initialize_events(3500, event_list)
#print(state['bikes_rented'])
print(sum(state['bikes_available']))
simulate(state, event_list)
#print(sum(state['waiting_for_bike']))
#print(state['bikes_available'])
#print(state['waiting_to_return'])
#print(state['waiting_for_bike'])
print(state['bikes_available'])
#print(len(state['waiting_to_return']))
print(state['bikes_rented'])
print(state['bikes_returned'])
print(state['bikes_lost'])
print(state['max_waitlist_length'])
```

```
{'waiting_for_bike': [[], [], [], [], [], [], [], [], [], [], [], [], [], [],
810
[10, 10, 4, 8, 10, 9, 10, 3, 10, 6, 7, 0, 10, 7, 9, 10, 10, 0, 10, 10, 10, 0,
3397
3170
1
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 7, 0, 0, 0, 13, 0, 0, 12,
```

Testing the model

```
max_bikes = 10
riders = []
event_list = FutureEventList()
state = initial_state()
initialize_events(1, event_list)
rider = riders[0]
print(rider.start_index)
simulate(state, event_list)
print(state['bikes_available'][rider.start_index])
print(state['waiting_to_return'][rider.end_index])
print(rider.rented)
print(rider.ended)
```

```
68
9
[<__main__.Rider object at 0x7a74c24eb040>]
True
False
```

As you can see, with just one person renting a bike, they successfully rent a bike, the bike number at that location decreases. When the rider tries to return to a location, it gets added to the waiting to return list because no one else has rented a bike from there yet and we cannot go over max capacity of 10. Additionally, the rider rented variable correctly is True and the rider.ended is false.

```
max_bikes = 0
riders = []
event_list = FutureEventList()
state = initial_state()
initialize_events(1, event_list)
rider = riders[0]
print(rider.start_index)
simulate(state, event_list)
print(state['bikes_available'][rider.start_index])
print(state['waiting_for_bike'][rider.start_index])
print(rider.rented)
```

```
    52
    0
    [<__main__.Rider object at 0x7a74c2293ac0>]
    False
```

In this test, there are no bikes to begin with, when the first rider arrives at their initial position, they are unable to get a bike so they are added to the waiting for bike queue, and rented is still false

```
max_bikes = 0
event_list = FutureEventList()
state = initial_state()
rider1 = Rider(0, 1)
rider2 = Rider(0, 2)
print(rider1)
print(rider2)
schedule(Event(rider1.arrival_time, arrive, rider1), event_list)
schedule(Event(rider2.arrival_time, arrive, rider2), event_list)
print(event_list)
simulate(state, event_list)
print(state['waiting_for_bike'][rider1.start_index])
state['waiting_for_bike'][rider1.start_index].pop(0)
print(state['waiting_for_bike'][rider1.start_index])
```

```
<__main__.Rider object at 0x7a74c2291e70>
<__main__.Rider object at 0x7a74cc2cec20>
[Event(t=1,
       f=<function arrive at 0x7a74bad8c280>,
       r=<__main__.Rider object at 0x7a74c2291e70>),
 Event(t=2,
       f=<function arrive at 0x7a74bad8c280>,
       r=<__main__.Rider object at 0x7a74cc2cec20>)]
[<__main__.Rider object at 0x7a74c2291e70>, <__main__.Rider object at 0x7a74c
[<__main__.Rider object at 0x7a74cc2cec20>]
```

If two riders try to pick up a bike when there are none left, the first one to arrive becomes first in line for a bike and pop correctly removes them from the list.

```
max_bikes = 0
event_list = FutureEventList()
state = initial_state()

rider1 = Rider(0, 1)
rider2 = Rider(1, 2)
schedule(Event(rider1.arrival_time, arrive, rider1), event_list)
schedule(Event(rider2.arrival_time, arrive, rider2), event_list)
simulate(state, event_list)
print(state['waiting_for_bike'])
```

```
[[<__main__.Rider object at 0x7a74cc2cf4f0>], [<__main__.Rider object at 0x7a
```

The waiting for bike array adds the waiting riders to the correct indices.

```
max_bikes = 10
riders = []
event_list = FutureEventList()
state = initial_state()
initialize_events(3500, event_list)
print(sum(state['bikes_available']))
simulate(state, event_list)
print("bikes available", sum(state['bikes_available']))
print("bikes rented but not returned yet", state['bikes_rented'] - state['bikes_r
```

```
    810
    bikes available 554
    bikes rented but not returned yet 256
```

We start with 810 bikes, 10 at each station. At the end of the simulation if we add up the bikes which are remaining available at station with the number rented - number returned (because some people are still waiting to return theirs, some are still riding, and a couple went to end destinations which are not in our set of starting locations) and it still equals 810 which is good.

```
max_bikes = 10
riders = []
event_list = FutureEventList()
state = initial_state()
rider1 = Rider(0, 1)
rider1.return_time = 1
rider1.end_index = 0
schedule(Event(rider1.return_time, return_bike, rider1), event_list)
simulate(state, event_list)
print(state['bikes_available'][rider1.end_index])
print(state['waiting_to_return'][rider1.end_index])
state = initial_state()
state['bikes_available'][0] = 9
schedule(Event(rider1.return_time, return_bike, rider1), event_list)
print(state['bikes_available'][rider1.end_index])
simulate(state, event_list)
print(state['bikes_available'][rider1.end_index])
print(state['waiting_to_return'][rider1.end_index])
```

```
10
[<__main__.Rider object at 0x7a74baa3b970>]
9
10
[]
```

This is two tests: the first shows that if we try to return a rider when there are already the maximum bikes, the bikes stays at 10 and the rider gets added to the waitlist to return. The second shows if there is not the max number of bikes, the return is successful and the bike count goes up 1.

```
max_bikes = 10
riders = []
event_list = FutureEventList()
state = initial_state()
rider1 = Rider(0, 1)
rider2 = Rider(0, 5)
rider1.return_time = 1
rider1.end_index = 0
schedule(Event(rider1.return_time, return_bike, rider1), event_list)
print(event_list)
simulate(state, event_list)
print(state['bikes_available'][rider1.end_index])
print(state['waiting_to_return'][rider1.end_index])
schedule(Event(rider2.arrival_time, arrive, rider2), event_list)
print(event_list)
simulate(state, event_list)
print(state['bikes_available'][rider1.end_index])
print(state['waiting_to_return'][rider1.end_index])
```

```
[Event(t=1,
       f=<function return_bike at 0x7a74bad8d120>,
       r=<__main__.Rider object at 0x7a74c243bdf0>)]
10
[<__main__.Rider object at 0x7a74c243bdf0>]
[Event(t=5,
       f=<function arrive at 0x7a74bad8caf0>,
       r=<__main__.Rider object at 0x7a74c2438e20>)]
10
[]
```

This tests the calling of a return when the number of bikes goes below the maximum. We first have a bike try to return at index 0, it is unable, so the rider is added to the waiting to return queue. Then a rider arrives to pick up a bike, they take a bike successfully, the rider from the return queue is able to return their bike, so the bikes available stays at ten and they are no longer in the queue.

```
max_bikes = 10
riders = []
event_list = FutureEventList()
state = initial_state()
state['bikes_available'][0] = 0
rider1 = Rider(0, 5)
rider2 = Rider(0, 1)
rider1.return_time = 5
rider1.end_index = 0
schedule(Event(rider2.arrival_time, arrive, rider2), event_list)
print(event_list)
simulate(state, event_list)
print(state['bikes_available'][0])
print(state['waiting_for_bike'][0])
schedule(Event(rider1.return_time, return_bike, rider1), event_list)
print(event_list)
simulate(state, event_list)
print(state['bikes_available'][0])
print(state['waiting_for_bike'][0])
```

```
[Event(t=1,
       f=<function arrive at 0x7a74bad8caf0>,
       r=<__main__.Rider object at 0x7a74c259b1f0>)]
0
[<__main__.Rider object at 0x7a74c259b1f0>]
[Event(t=5,
       f=<function return_bike at 0x7a74bad8d120>,
       r=<__main__.Rider object at 0x7a74c2598700>)]
0
[]
```

This is very similar to the previous test. This shows that if we have no bikes available and someone arrives, the bikes does not decrement, and the rider is added to the waiting for bike queue. If a rider then returns a bike to that location, the bike count temporarily goes up allowing the first rider to exit the queue and rent the new bike, resetting the bike count to 0.

```
max_bikes = 10
riders = []
event_list = FutureEventList()
state = initial_state()
state['bikes_available'][0] = 0
rider1 = Rider(0, 5)
rider2 = Rider(0, 1)
rider1.return_time = 5
rider1.end_index = 0
schedule(Event(rider2.arrival_time, arrive, rider2), event_list)
schedule(Event(rider1.return_time, return_bike, rider1), event_list)
print(event_list)
simulate(state, event_list)
print(rider2.arrival_time, rider2.bike_rental_time)
```

```
[Event(t=1,
       f=<function arrive at 0x7a74bad8caf0>,
       r=<__main__.Rider object at 0x7a74bab31090>),
 Event(t=5,
       f=<function return_bike at 0x7a74bad8d120>,
       r=<__main__.Rider object at 0x7a74bab302b0>)]
1 5
```

This is the same scenario, but I am using it to show that the arrival time and bike rental time variables are correctly stored in the rider. Their initial arrival is at t=1, the next rider returns at t=5 and therefore the rider should have a bike rental time of 5 which is correct. This rider has a waiting time of 4 minutes.

Through these tests, I have shown that events are scheduled correctly, bikes are not lost during simulation (except when they go to locations which are not in the start location list), the waiting for bike and waiting to return queues work correctly, if there are no bikes then you cannot rent a bike, if there are bikes available then you can rent one, and that my variables for tracking waiting time work correctly. I also show my rented and ended variables work correctly, although they are not used until the next section.

```
#2.2

trials = 50
probability_of_success = []
average_wait_time = []
for i in range(trials):
  max_bikes = 10
  riders = []
  event_list = FutureEventList()
  state = initial_state()
  initialize_events(3500, event_list)
  simulate(state, event_list)
  success_rate = state['bikes_rented']/3500
  probability_of_success.append(success_rate)
  arrival_times_sum = 0
  rental_times_sum = 0
  for rider in riders:
    if rider.rented == True:
      arrival_times_sum += rider.arrival_time
      rental_times_sum += rider.bike_rental_time
  avg_wait_time = (rental_times_sum - arrival_times_sum)/state['bikes_rented']
  average_wait_time.append(avg_wait_time)
```

```
import scipy.stats as st
```

```
print(probability_of_success)
print(average_wait_time)
success_mean = np.mean(probability_of_success)
print(success_mean)
wait_mean = np.mean(average_wait_time)
print(wait_mean)
success_stdev = np.std(probability_of_success)
wait_stdev = np.std(average_wait_time)
success_interval = st.t.interval(confidence=0.90, df=len(probability_of_success)-
                 loc=success_mean,
                 scale=st.sem(probability_of_success))
wait_interval = st.t.interval(confidence=0.90, df=len(average_wait_time)-1,
                 loc=wait_mean,
                 scale=st.sem(average_wait_time))
print('probability of getting a bike, interval', success_interval)
print('average wait time, interval', wait_interval)
```

```
[0.9585714285714285, 0.978, 0.9397142857142857, 0.9702857142857143, 0.94314285
[7.971747976641453, 8.767593760193842, 13.01671032634102, 9.183268131798757, 1
0.9571885714285715
8.872282201692624
probability of getting a bike, interval (0.9536677933414494, 0.960709349515693
average wait time, interval (8.222527693370965, 9.522036710014282)
```

2.3 The problem statement is a bit vague, so what I will be doing is simulating and determining what the minimum number of bikes at each station which lead to an average wait time of 0 with the condition that there is no maximum bikes at a location, so there is never any waiting to return

```
max_bikes = 0
riders = []
event_list = FutureEventList()
state = initial_state()
initialize_events(3500, event_list)
max_bikes = 1000000
simulate(state, event_list)
print(state['max_waitlist_length'])
print(state['waiting_for_bike'])
print(state['bikes_rented_location'])
```

```
[75, 93, 69, 38, 11, 24, 51, 48, 53, 22, 39, 13, 35, 29, 4, 50, 38, 29, 29, 55
[[<__main__.Rider object at 0x7a74bb0f8970>, <__main__.Rider object at 0x7a74b
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
max_bikes = 30
riders = []
event_list = FutureEventList()
state = initial_state()
initialize_events(3500, event_list)
max_bikes = 1000000
simulate(state, event_list)
print(state['max_waitlist_length'])
print(state['waiting_for_bike'])
print(state['bikes_rented_location'])
arrival_times_sum = 0
rental_times_sum = 0
for rider in riders:
  if rider.rented == True:
    arrival_times_sum += rider.arrival_time
    rental_times_sum += rider.bike_rental_time
avg_wait_time = (rental_times_sum - arrival_times_sum)/state['bikes_rented']
print(avg_wait_time)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[[], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [],
[53, 81, 62, 35, 16, 25, 46, 40, 48, 22, 29, 18, 33, 26, 2, 42, 19, 28, 19, 4
0.0
```

If each location has 30 bikes and no upper bound, there is no wait time

```
max_bikes = 25
riders = []
event_list = FutureEventList()
state = initial_state()
initialize_events(3500, event_list)
max_bikes = 1000000
simulate(state, event_list)
print(state['max_waitlist_length'])
print(state['waiting_for_bike'])
print(state['bikes_rented_location'])
arrival_times_sum = 0
rental_times_sum = 0
for rider in riders:
  if rider.rented == True:
    arrival_times_sum += rider.arrival_time
    rental_times_sum += rider.bike_rental_time
avg_wait_time = (rental_times_sum - arrival_times_sum)/state['bikes_rented']
print(avg_wait_time)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[[], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [],
[64, 75, 74, 41, 8, 25, 48, 55, 34, 18, 32, 18, 30, 27, 4, 35, 31, 38, 25, 41,
0.0
```

```
max_bikes = 25
riders = []
event_list = FutureEventList()
state = initial_state()
initialize_events(3500, event_list)
max_bikes = 1000000
simulate(state, event_list)
print(state['max_waitlist_length'])
print(state['waiting_for_bike'])
print(state['bikes_rented_location'])
arrival_times_sum = 0
rental_times_sum = 0
for rider in riders:
  if rider.rented == True:
    arrival_times_sum += rider.arrival_time
    rental_times_sum += rider.bike_rental_time
avg_wait_time = (rental_times_sum - arrival_times_sum)/state['bikes_rented']
print(avg_wait_time)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[[], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [],
[56, 84, 67, 28, 9, 16, 39, 57, 29, 27, 43, 19, 24, 20, 7, 37, 25, 38, 15, 49,
0.4325768222931226
```

With 25 bikes at each station, we can get a perfect simulation where no one waits, but sometimes we don't

```
max_bikes = 26
wait_time_list = []
for i in range(25):
  riders = []
  event_list = FutureEventList()
  state = initial_state()
  initialize_events(3500, event_list)
  max_bikes = 1000000
  simulate(state, event_list)
  print(state['max_waitlist_length'])
  arrival_times_sum = 0
  rental_times_sum = 0
  for rider in riders:
    if rider.rented == True:
      arrival_times_sum += rider.arrival_time
      rental_times_sum += rider.bike_rental_time
  avg_wait_time = (rental_times_sum - arrival_times_sum)/state['bikes_rented']
  wait_time_list.append(avg_wait_time)
print(sum(wait_time_list))
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0.0
```

Running the simulation 25 times with 26 bikes at each station to start and no maximum bikes, we get an overall wait time of 0. Therefore, if we initialize each station with 26 bikes, no one will have to wait to get a bike

When running the simulation 10 times with 26 bikes at each starting location and no max capacity, no riders ever have to wait for a bike