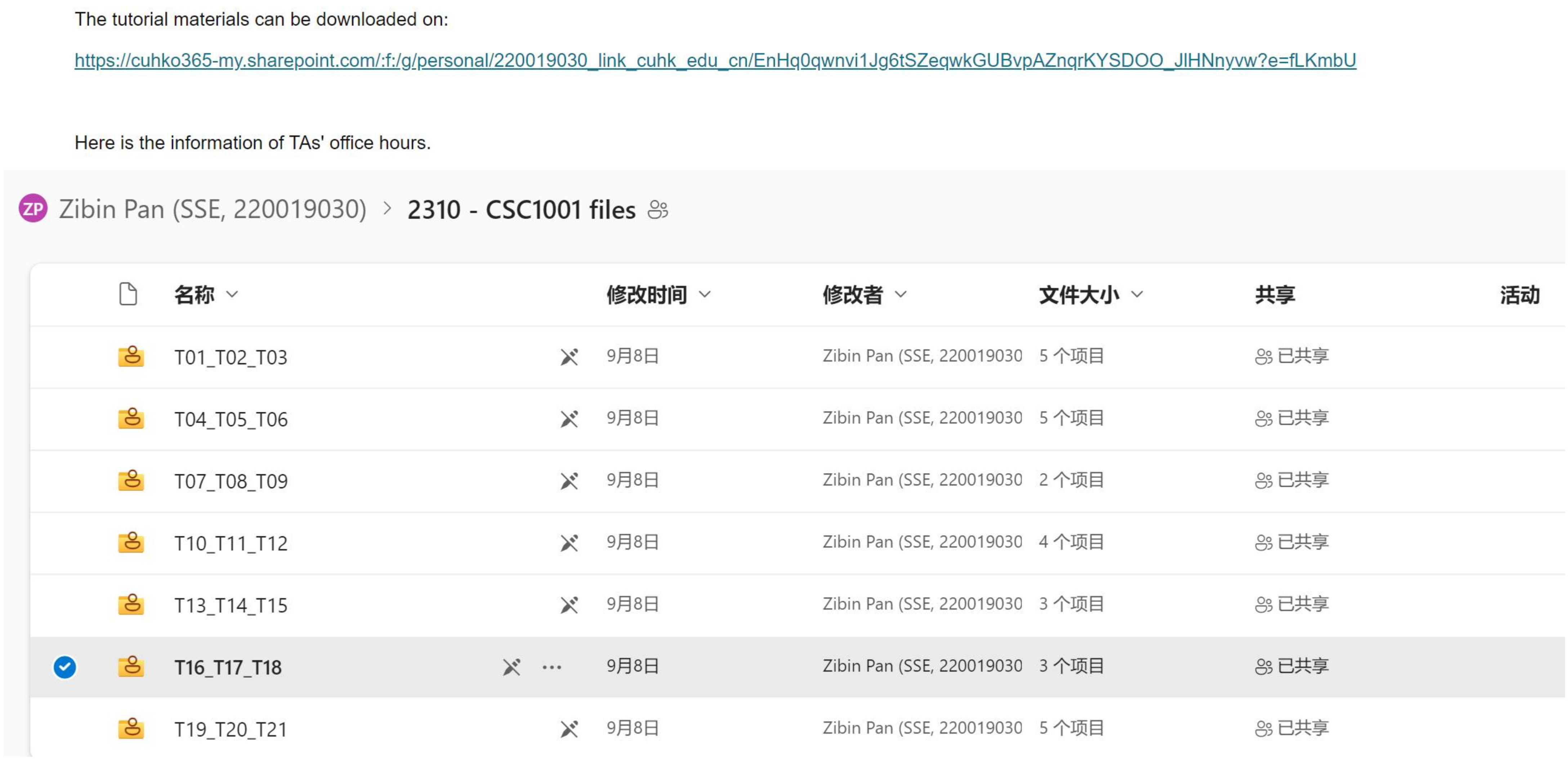
香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

INTRODUCTION TO COMPUTER SCIENCE: PROGRAMMING METHODOLOGY

TUTORIAL 12

DATA STRUCTURE-STACK AND QUEUE

- Download this slides and codes at:



The tutorial materials can be downloaded on:

https://cuhko365-my.sharepoint.com/:f:/g/personal/220019030_link_cuhk_edu_cn/EnHq0qwnvi1Jg6tSZeqwkGUBvpAZnqrKYSDOO_JlHNnyvw?e=fLKmbU

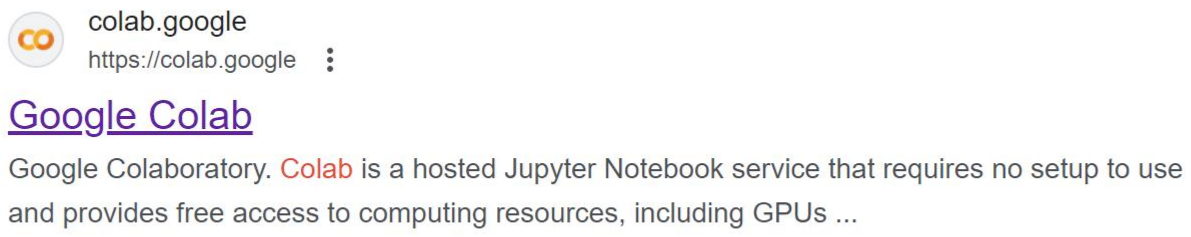
Here is the information of TAs' office hours.

ZP Zibin Pan (SSE, 220019030) > 2310 - CSC1001 files

名称	修改时间	修改者	文件大小	共享	活动
T01_T02_T03	9月8日	Zibin Pan (SSE, 220019030	5 个项目	已共享	
T04_T05_T06	9月8日	Zibin Pan (SSE, 220019030	5 个项目	已共享	
T07_T08_T09	9月8日	Zibin Pan (SSE, 220019030	2 个项目	已共享	
T10_T11_T12	9月8日	Zibin Pan (SSE, 220019030	4 个项目	已共享	
T13_T14_T15	9月8日	Zibin Pan (SSE, 220019030	3 个项目	已共享	
T16_T17_T18	9月8日	Zibin Pan (SSE, 220019030	3 个项目	已共享	
T19_T20_T21	9月8日	Zibin Pan (SSE, 220019030	5 个项目	已共享	

How to use .ipynb file?

Online: Google Colab

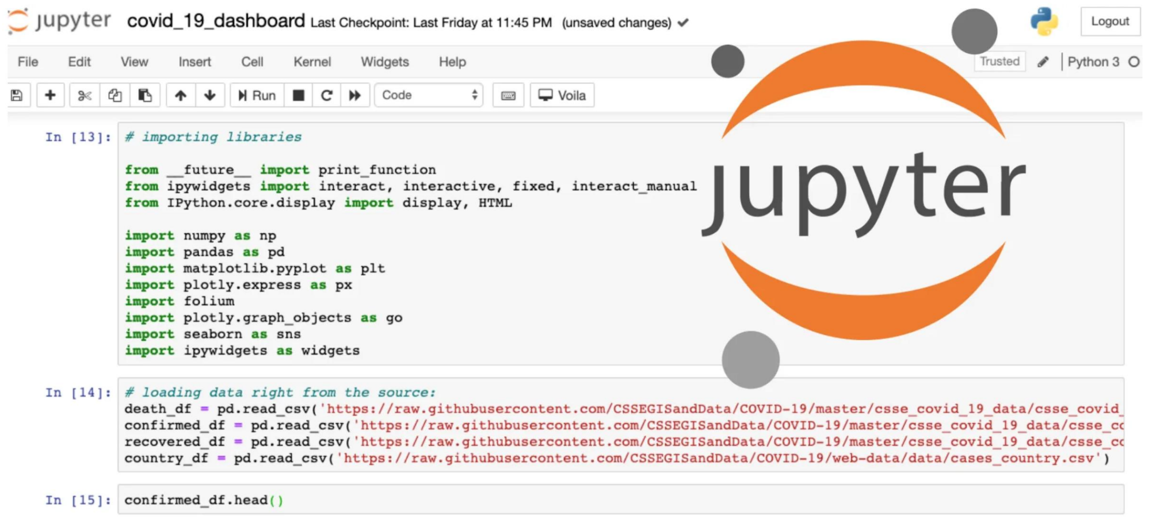


colab.google
https://colab.google

Google Colab

Google Colaboratory. Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs ...

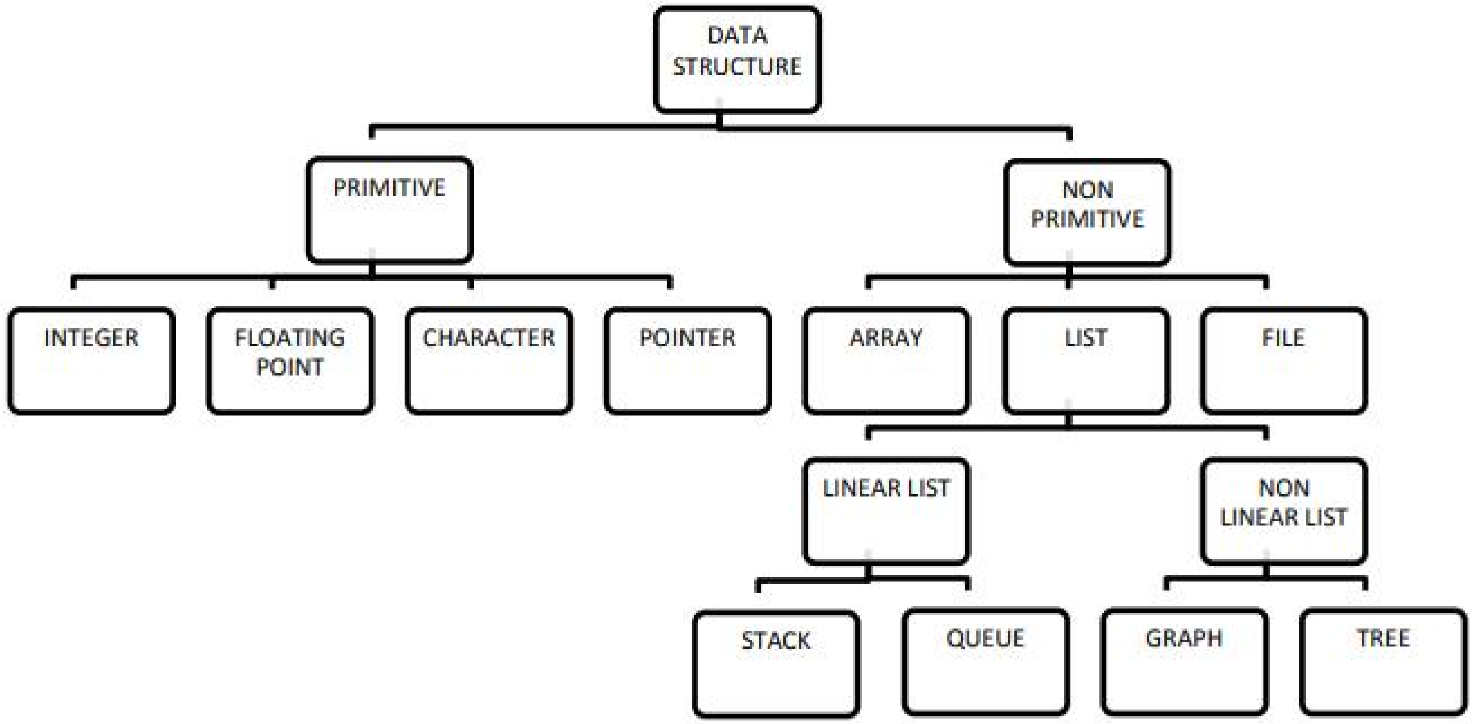
Offline: Jupyter Notebook(in Anaconda3)


jupyter covid_19_dashboard Last Checkpoint: Last Friday at 11:45 PM (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
Voila
jupyter

```
In [13]: # importing libraries

         from __future__ import print_function
         from ipywidgets import interact, interactive, fixed, interact_manual
         from IPython.core.display import display, HTML

         import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import plotly.express as px
         import folium
         import plotly.graph_objects as go
         import seaborn as sns
         import ipywidgets as widgets

In [14]: # loading data right from the source:
         death_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_
         confirmed_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_cc
         recovered_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_cc
         country_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/web-data/data/cases_country.csv')

In [15]: confirmed_df.head()
```

Classification of Data Structure

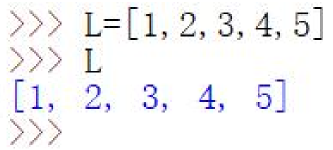
- Data structure is a representation of the logical relationship existing between individual elements of data.
- Algorithm + Data Structure=Program



- *Array* stores the same data types.
- *List* can stores a collection of different data types, by storing all of their pointers (pointing towards their addresses), which is very space consuming.

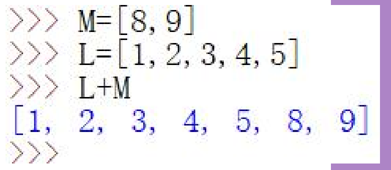
Operations on Data Structure

1. **Create**
   The create operation results in reserving memory for program elements. This can be done by declaration statement. Creation of data structure may take place either during compile-time or run-time. malloc() function of C language is used for creation.
2. **Destroy**
   Destroy operation destroys memory space allocated for specified data structure. free() function of C language is used to destroy data structure.
3. **Selection**
   Selection operation deals with accessing a particular data within a data structure.
4. **Updation**
   It updates or modifies the data in the data structure.
5. **Searching**
   It finds the presence of desired data item in the list of data items, it may also find the locations of all elements that satisfy certain conditions.
6. **Sorting**
   Sorting is a process of arranging all data items in a data structure in a particular order, say for example, either in ascending order or in descending order.
7. **Merging**
   Merging is a process of combining the data items of two different sorted list into a single sorted list.
8. **Splitting**
   Splitting is a process of partitioning single list to multiple list.
9. **Traversal**
   Traversal is a process of visiting each and every node of a list in systematic manner.



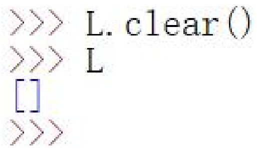
```
>>> L=[1,2,3,4,5]
>>> L
[1, 2, 3, 4, 5]
>>>
```

(Create)



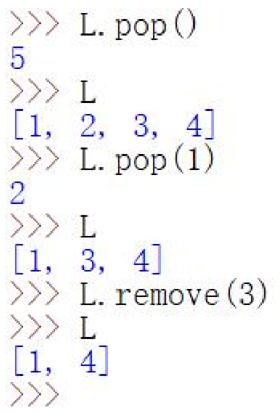
```
>>> M=[8,9]
>>> L=[1,2,3,4,5]
>>> L+M
[1, 2, 3, 4, 5, 8, 9]
>>>
```

(Merging)



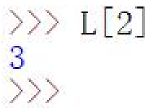
```
>>> L.clear()
>>> L
[]
>>>
```

(Destroy)



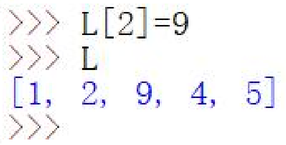
```
>>> L.pop()
5
>>> L
[1, 2, 3, 4]
>>> L.pop(1)
2
>>> L
[1, 3, 4]
>>> L.remove(3)
>>> L
[1, 4]
>>>
```

(Deletion)



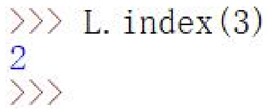
```
>>> L[2]
3
>>>
```

(Selection)



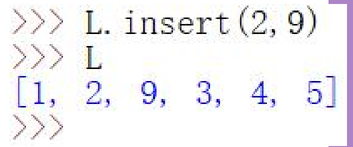
```
>>> L[2]=9
>>> L
[1, 2, 9, 4, 5]
>>>
```

(Updation)



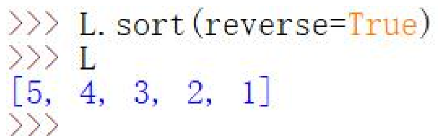
```
>>> L.index(3)
2
>>>
```

(Searching)



```
>>> L.insert(2,9)
>>> L
[1, 2, 9, 3, 4, 5]
>>>
```

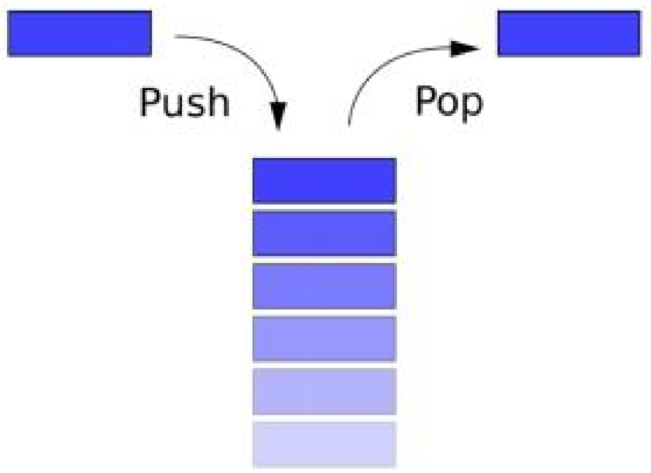
(Insertion)

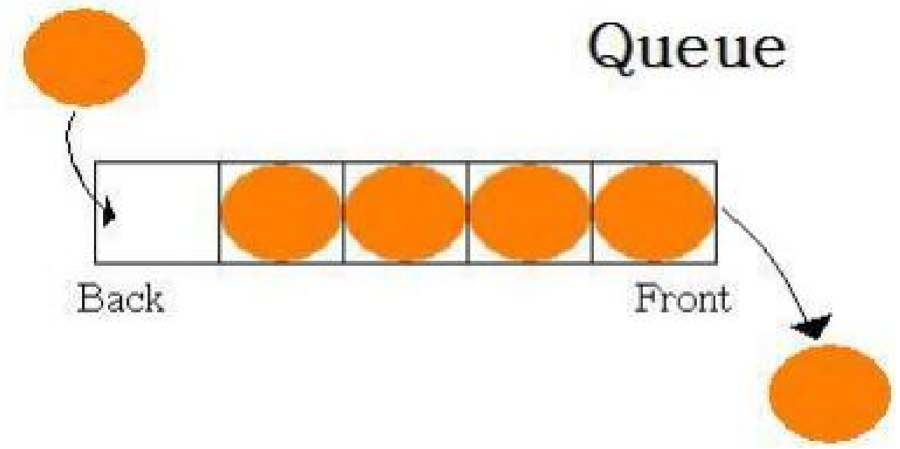


```
>>> L.sort(reverse=True)
>>> L
[5, 4, 3, 2, 1]
>>>
```

(Sorting)

Stack and Queue





- Uses of stacks?
  - Implement recursive algorithms
  - Compiling and running programs
  - Backtracking algorithms like constraint satisfaction (e.g. Sudoku(数独))

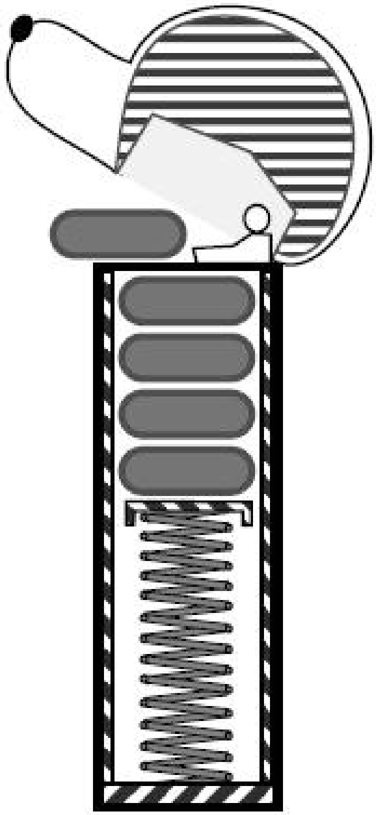
- Uses of queues?
  - Website requests
  - Simulations or game engines
  - Expanding algorithms like Dijkstra's

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Stack

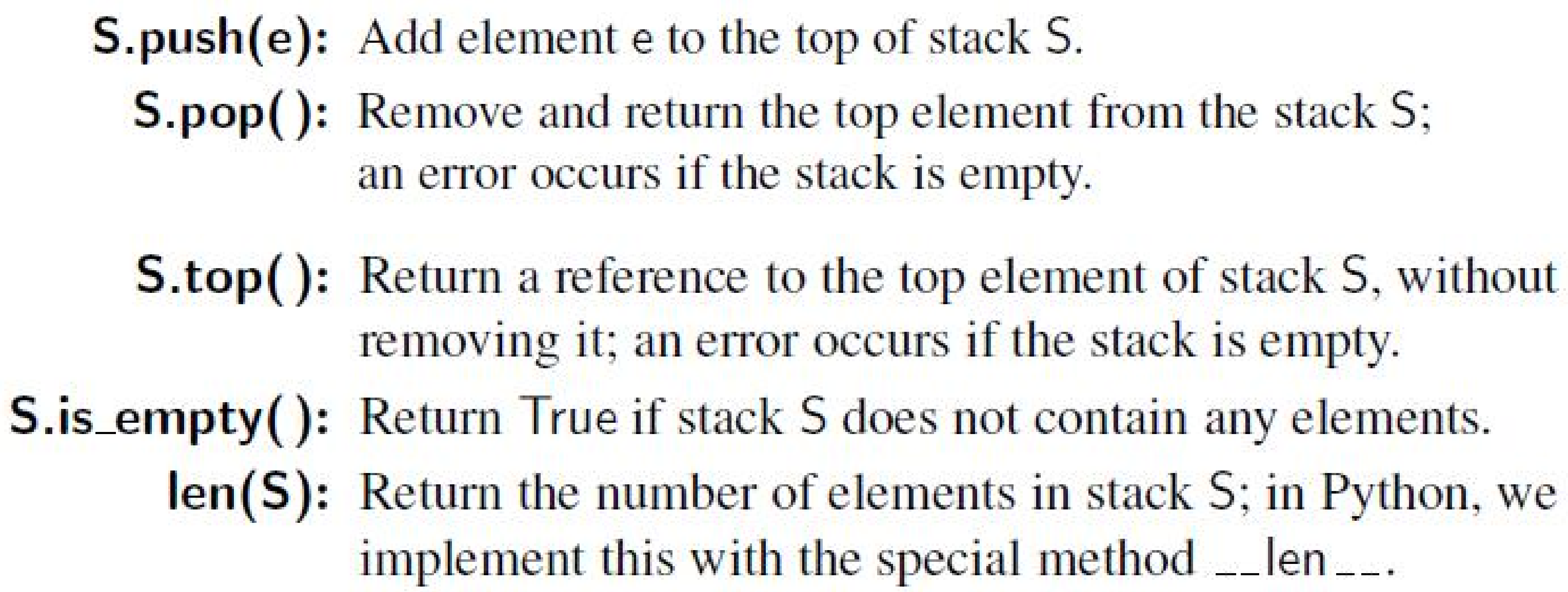
- A stack is a collection of objects that are inserted and removed according to the last-in, first-out (LIFO) principle

- A user may insert objects into a stack at any time, but may only access or remove the most recently inserted object that remains (at the so-called “top” of the stack)



Stack Class

- Generally, a stack may contain the following methods:
  - ➢ Refer to the *stack.py* file for definition of stack class.



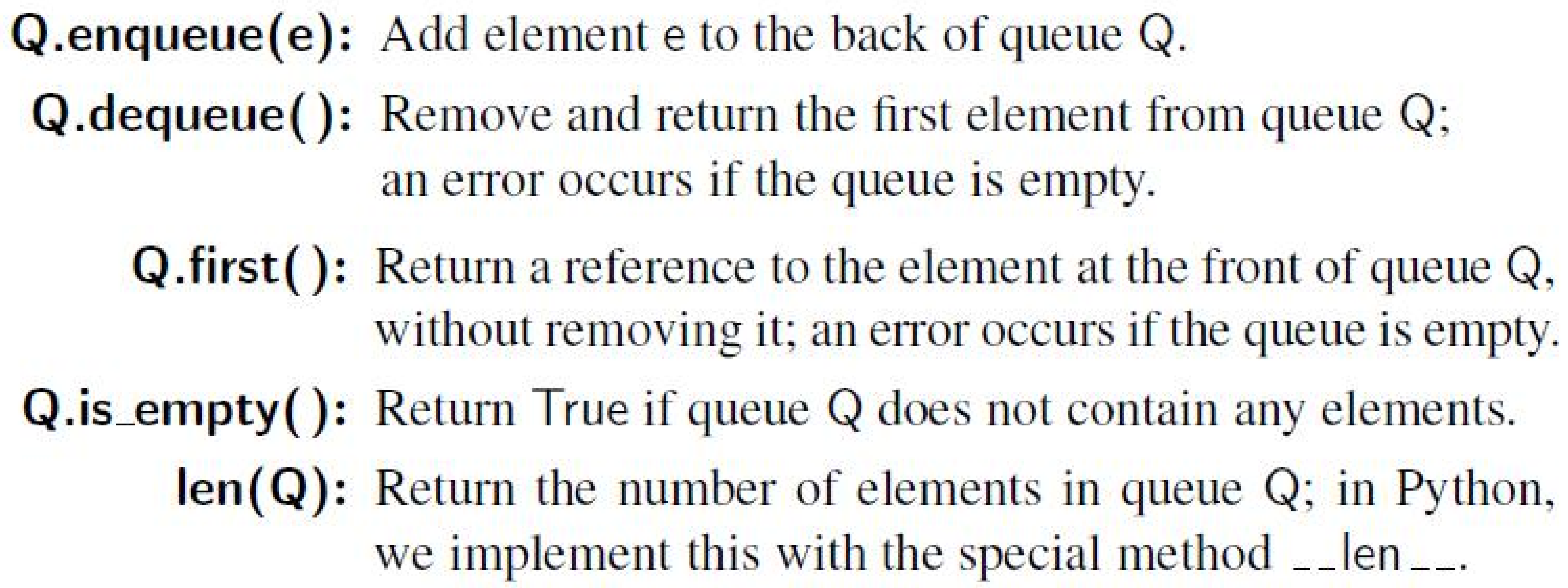
S.push(e):	Add element e to the top of stack S.
S.pop():	Remove and return the top element from the stack S; an error occurs if the stack is empty.
S.top():	Return a reference to the top element of stack S, without removing it; an error occurs if the stack is empty.
S.is_empty():	Return True if stack S does not contain any elements.
len(S):	Return the number of elements in stack S; in Python, we implement this with the special method __len__.

Queue

- Queue is another fundamental data structure
- A queue is a collection of objects that are inserted and removed according to the first-in, first-out (FIFO) principle
- Elements can be inserted at any time, but only the element that has been in the queue the longest can be next removed

Queue Class

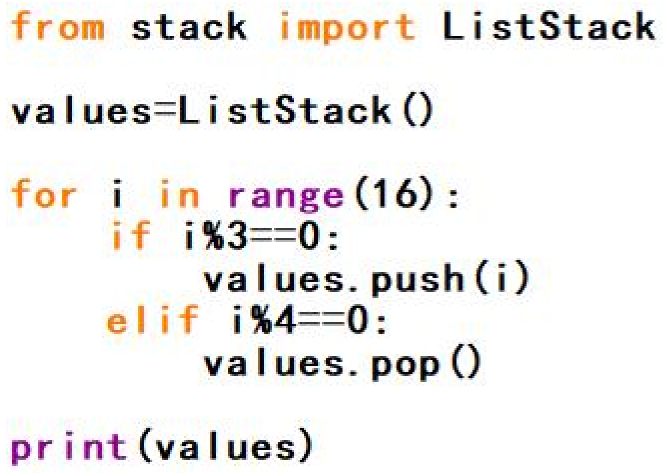
- The queue class may contain the following methods:
  - ➢ Refer to the *Queue.py* file for definition of queue class.



Q.enqueue(e):	Add element e to the back of queue Q.
Q.dequeue( ):	Remove and return the first element from queue Q; an error occurs if the queue is empty.
Q.first( ):	Return a reference to the element at the front of queue Q, without removing it; an error occurs if the queue is empty.
Q.is_empty( ):	Return True if queue Q does not contain any elements.
len(Q):	Return the number of elements in queue Q; in Python, we implement this with the special method __len__.

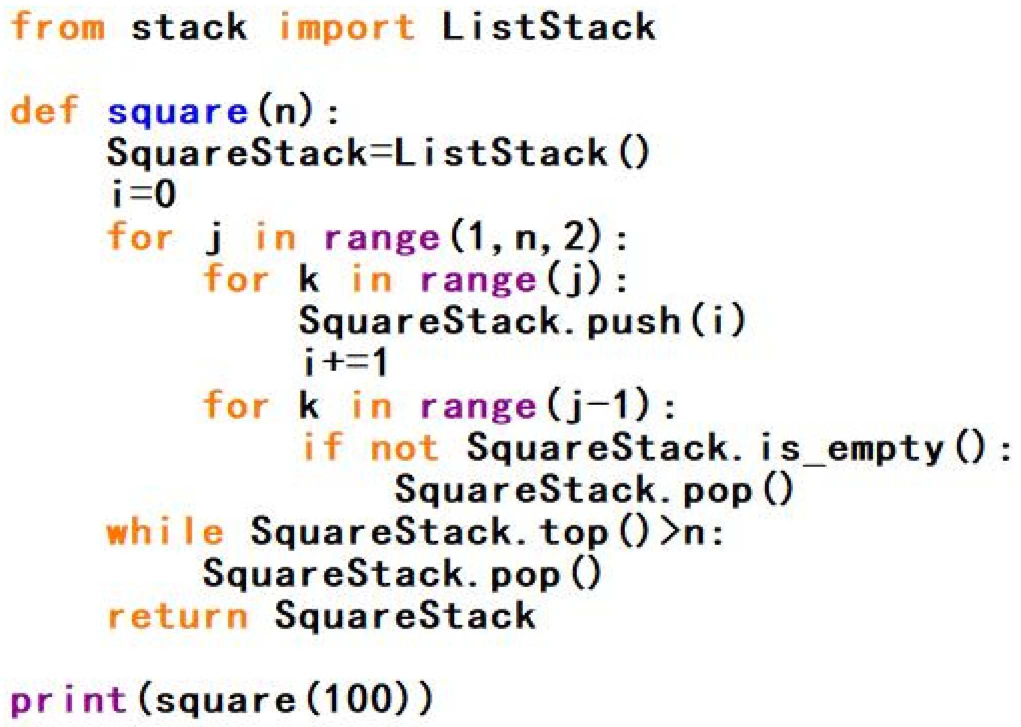
Q1: Create a Stack

- What is the outputs of the following programs?



```python
from stack import ListStack

values=ListStack()

for i in range(16):
    if i%3==0:
        values.push(i)
    elif i%4==0:
        values.pop()

print(values)
```

(a)



```python
from stack import ListStack

def square(n):
    SquareStack=ListStack()
    i=0
    for j in range(1,n,2):
        for k in range(j):
            SquareStack.push(i)
            i+=1
        for k in range(j-1):
            if not SquareStack.is_empty():
                SquareStack.pop()
    while SquareStack.top()>n:
        SquareStack.pop()
    return SquareStack

print(square(100))
```

(b)

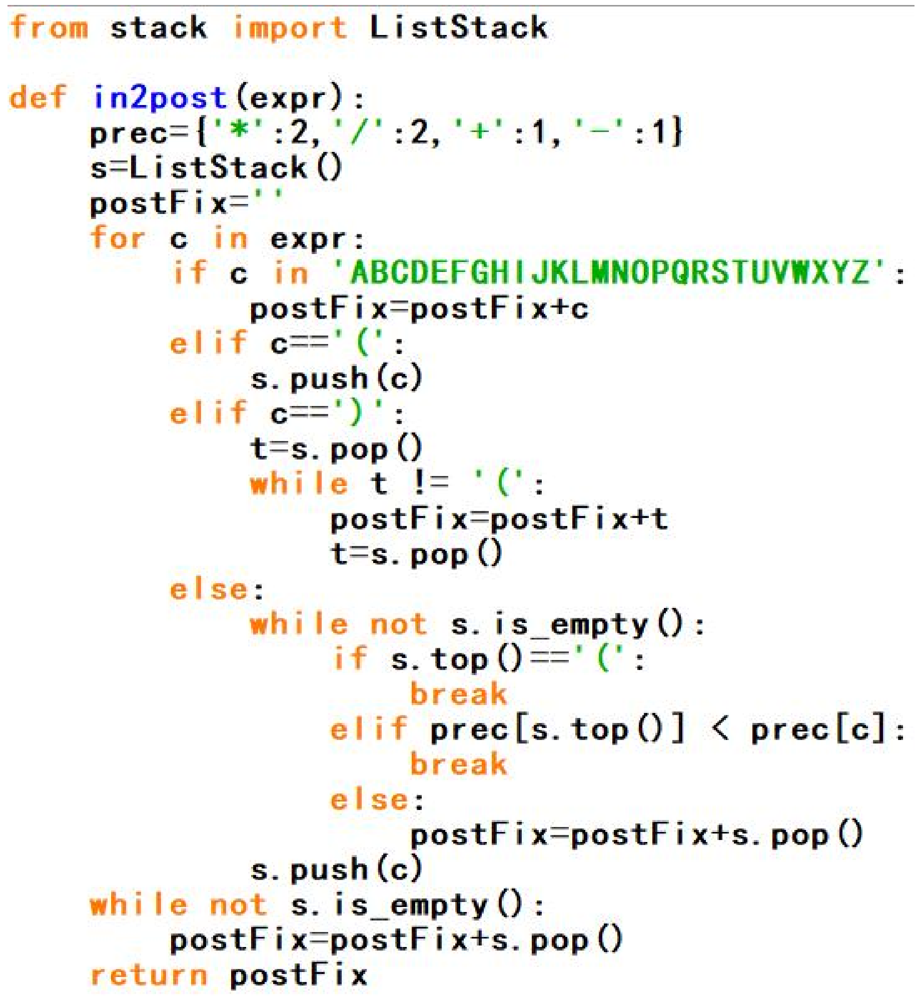
Q2: Infix and Postfix-I

- Three different notations can be used to represent a mathematical expression. The most common is the traditional algebraic or *infix* notation where the operator is specified between the operands A+B. The *prefix* notation places the operator immediately preceding the two operands +AB, whereas in *postfix* notation, the operator follows the two operands AB+. For example, *infix* notation (A+B)/(C-D)+E can be changed to *postfix* one AB+CD-/E+. Please answer the following questions:

Q2: Infix and Postfix-II

- i) Translate each of the following expressions into postfix ones:
  - (a) V*W*(X+Y)-Z (b) X-(Y+W*(Z/V))
  - (c) A-B*(C+D/E) (d) A/(B*C)-(D+E)
- ii) Translate each of the following postfix expressions into infix:
  - (a) ABC-D*+ (b) XYZ+AB-*-
  - (c) AB+CD-/E+ (d) AB+C-DE*+
- iii) Check the program attached next page, which is to implement translating infix to postfix, and verify your answers in i).

Q2: Infix and Postfix-III



```python
from stack import ListStack

def in2post(expr):
    prec={'*':2,'/':2,'+':1,'-':1}
    s=ListStack()
    postFix=''
    for c in expr:
        if c in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ':
            postFix=postFix+c
        elif c=='(':
            s.push(c)
        elif c==')':
            t=s.pop()
            while t != '(':
                postFix=postFix+t
                t=s.pop()
        else:
            while not s.is_empty():
                if s.top()=='(':
                    break
                elif prec[s.top()] < prec[c]:
                    break
                else:
                    postFix=postFix+s.pop()
            s.push(c)
    while not s.is_empty():
        postFix=postFix+s.pop()
    return postFix
```

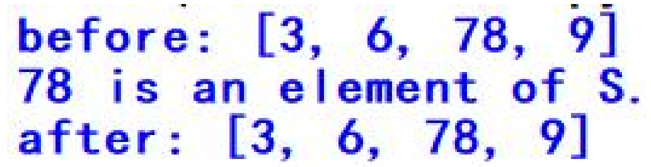
- iv) Evaluate the results of expressions in i) with V=1, W=3, X=7, Y=9, Z=10 and A=12, B=4, C=3, D=8, E=2. You can read the next page for instructions. If you like, you can try to write a general function def post2in(expr) to change *postfix* expressions to *infix* and verify your answers in ii).

Q2: Infix and Postfix-IV

- v) Using postfix is easy to use a stack to evaluate the result of the expression (i.e. replace each letter with actual data). To implement this, you keep pushing the letters and operators in the postfix expression into a stack and whenever you get an operator you pop the previous two expressions and calculate the postfix relative to this operator and push back the result to the stack and move on. Please write a function def evaluate(expr) to first invoke the in2post() to translate the expression expr to postfix and then following the above instructions to calculate the result.

Q3: Find Target In A Stack

- Suppose you have a stack S containing n elements and a queue Q that is initially empty. Describe how you can use Q to scan S to see if it contains a certain element x, with the additional constraint that your algorithm must return the elements back to S in their original order. You may only use one stack S and one queue Q.



```
before: [3, 6, 78, 9]
78 is an element of S.
after: [3, 6, 78, 9]
```

- *The stack S before and after the search is printed.*

Q4: Queuing Model Simulation-I

- We can model a queuing system by constructing a discrete event simulation. The simulation is a sequence of significant events that cause a change in the system. For example, in our airline ticket counter simulation, these events would include customer arrival, the start or conclusion of a transaction, or customer departure. The simulation is time driven and performed over a preset time period. The passing of time is represented by a loop, which increments a discrete time intervals. Thus, the time units must be small enough such that no event can occur between units.

Q4: Queuing Model Simulation-II

- A simulation is commonly designed to allow the user to supply parameters that define the conditions of the system. For a discrete event simulation modeling a queuing system, these parameters include:
  - The length of the simulation given in numbers of time units. The simulation typically begins at time unit zero.
  - The number of servers providing the services to the customers.
  - The expected service time to complete a transaction.
  - The distribution of arrival times, which is used to determine when customers arrive.

Q4: Queuing Model Simulation-III

- Consider a single server, with service time T under the exponential distribution $\mathbf{P(T = t) = \mu e^{-\mu t} (t \geq 0)}$, which means the average service time will be $\frac{1}{\mu}$ or average number of customers being served under unit time is $\boldsymbol{\mu}$. And the time $\boldsymbol{T}$ between a new arrival entering the queue and the previous arrival follows the same kind of distribution $\mathbf{P(T = t) = \lambda e^{-\lambda t} (t \geq 0)}$, which means the average number of arrivals under unit time is λ. The queuing model is denoted as M/M/1 model. What we concern is the average time each customer need to wait under these parameters. Follow

Q4: Queuing Model Simulation-IV

- i) Define a Customer class that has two private data fields id number self.__idNum and arrival time self.__arrivalTime.
- ii) Define a Server class that has two private data fields self.__customer to record which customer is under service and self.__stopTime to record the time when the service stops and two of all methods are def startService(self, customer, stopTime) and def stopService(self).
- iii) Define a Simulation class that has private data fields for μ and λ, and total time, the customer queue, total waiting time, total number of customers. The methods include def run(self), def printResults(self), def __handdleArrive(curTime), def

Q4: Queuing Model Simulation-V

- iv) Write a main() function to run your simulation class, setting $\boldsymbol{\mu} = \mathbf{10}$, $\lambda = \mathbf{9}$, total time=480mins(large enough). Calculate the total number of customers L in the queue at the end t=480mins, average waiting time T for each customer(waiting time is equal to time when leave the queue minus time when enter the queue).
- v) Repeat running your simulation class for N=1000 times(large enough), and calculate the average value of the quantity L and T. And compared to the theoretical results $\bar{L} = \frac{\rho}{1-\rho} = 9$, $\bar{T} = \frac{\rho}{\mu-\lambda} = 0.9$ mins,

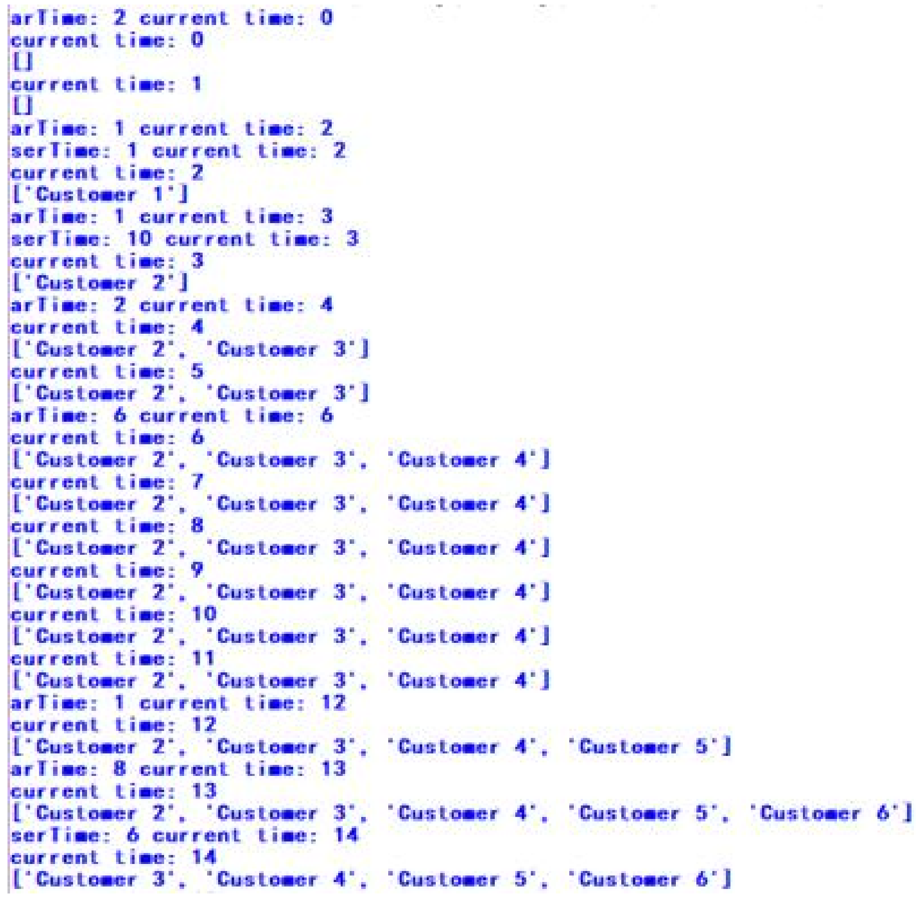
where $\rho = \frac{\lambda}{}$

Q4: Queuing Model Simulation-VI

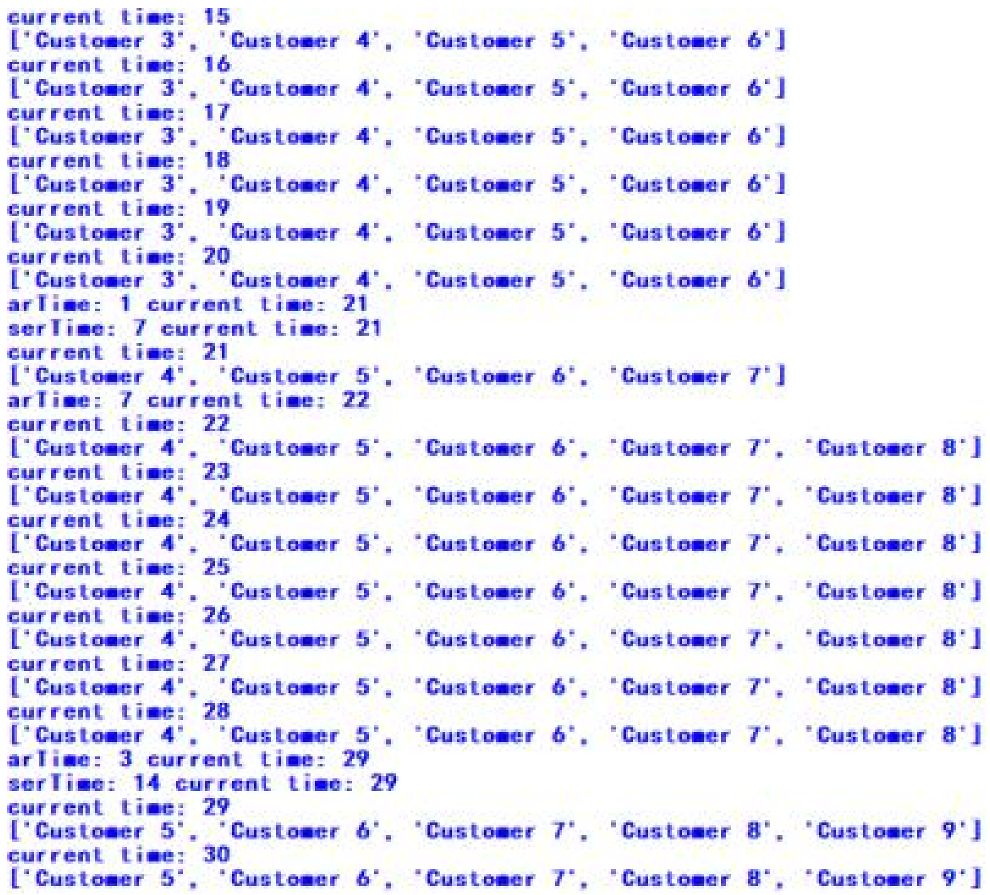
- vi) Define a function arrivalTime(lamda, timeUnit) to get the arrival time(or service time) following exponential distribution. Suppose time unit is 1 second and the probability for arrival at t=c(seconds)=c/60(mins) is $\int_{(c-\varepsilon)/60}^{(c+\varepsilon)/60} \lambda e^{-\lambda t} dt\,|_{\varepsilon=0.5} = \lambda e^{-\frac{\lambda c}{60}}\left(e^{\frac{\varepsilon}{60}} - e^{-\frac{\varepsilon}{60}}\right) \approx \lambda e^{-\frac{\lambda c}{60}} \cdot \frac{2\varepsilon}{60} = \frac{\lambda}{60} e^{-\frac{\lambda c}{60}}$, we let the probability for arrival at c=1 to be P(t=1)=$\frac{\lambda}{60} + \frac{\lambda}{60} e^{-\frac{\lambda}{60}}$, and let arrival at c=2, 3, 4⋯n-1 to be P(t=c)=$\frac{\lambda}{60} e^{-\frac{\lambda c}{60}}$, where $1 - \sum_{c=1}^{n-1} P(t=c) < 0.01$ and let $P(t=n) = 1 - \sum_{c=1}^{n-1} P(t=c)$ so that $\sum_{c=1}^{n} P(t=c) = 1$. Use a list to store possibility for all c=1, 2, 3, ⋯n and use

Q4: Queuing Model Simulation-VII

- **Sample Run: timeUnit =1 second, run totally for 30 seconds, print out the queue for each time. In this case, the arrival time generated is 2,1,1,2,6,1,8,1,7,3(s), which means there are customers arriving at 2, 3, 4, 6, 12, 13, 21, 22, 29(s), the service time generated is 1,10,6,7,14, which means Customer1 is served at 2s and leaves at 3s, …**



```
arTime: 2 current time: 0
current time: 0
[]
current time: 1
[]
arTime: 1 current time: 2
serTime: 1 current time: 2
current time: 2
['Customer 1']
arTime: 1 current time: 3
serTime: 10 current time: 3
current time: 3
['Customer 2']
arTime: 2 current time: 4
current time: 4
['Customer 2', 'Customer 3']
current time: 5
['Customer 2', 'Customer 3']
arTime: 6 current time: 6
current time: 6
['Customer 2', 'Customer 3', 'Customer 4']
current time: 7
['Customer 2', 'Customer 3', 'Customer 4']
current time: 8
['Customer 2', 'Customer 3', 'Customer 4']
current time: 9
['Customer 2', 'Customer 3', 'Customer 4']
current time: 10
['Customer 2', 'Customer 3', 'Customer 4']
current time: 11
['Customer 2', 'Customer 3', 'Customer 4']
arTime: 1 current time: 12
current time: 12
['Customer 2', 'Customer 3', 'Customer 4', 'Customer 5']
arTime: 8 current time: 13
current time: 13
['Customer 2', 'Customer 3', 'Customer 4', 'Customer 5', 'Customer 6']
serTime: 6 current time: 14
current time: 14
['Customer 3', 'Customer 4', 'Customer 5', 'Customer 6']
```

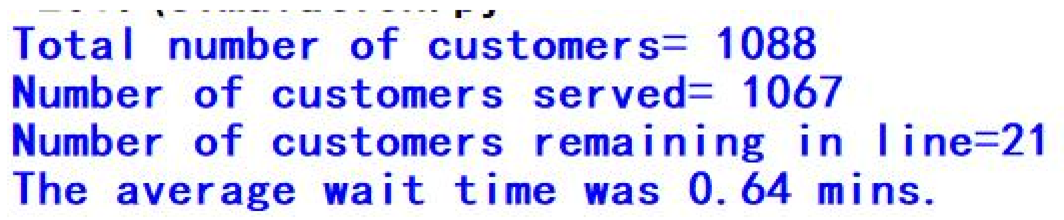


```
current time: 15
['Customer 3', 'Customer 4', 'Customer 5', 'Customer 6']
current time: 16
['Customer 3', 'Customer 4', 'Customer 5', 'Customer 6']
current time: 17
['Customer 3', 'Customer 4', 'Customer 5', 'Customer 6']
current time: 18
['Customer 3', 'Customer 4', 'Customer 5', 'Customer 6']
current time: 19
['Customer 3', 'Customer 4', 'Customer 5', 'Customer 6']
current time: 20
['Customer 3', 'Customer 4', 'Customer 5', 'Customer 6']
arTime: 1 current time: 21
serTime: 7 current time: 21
current time: 21
['Customer 4', 'Customer 5', 'Customer 6', 'Customer 7']
arTime: 7 current time: 22
current time: 22
['Customer 4', 'Customer 5', 'Customer 6', 'Customer 7', 'Customer 8']
current time: 23
['Customer 4', 'Customer 5', 'Customer 6', 'Customer 7', 'Customer 8']
current time: 24
['Customer 4', 'Customer 5', 'Customer 6', 'Customer 7', 'Customer 8']
current time: 25
['Customer 4', 'Customer 5', 'Customer 6', 'Customer 7', 'Customer 8']
current time: 26
['Customer 4', 'Customer 5', 'Customer 6', 'Customer 7', 'Customer 8']
current time: 27
['Customer 4', 'Customer 5', 'Customer 6', 'Customer 7', 'Customer 8']
current time: 28
['Customer 4', 'Customer 5', 'Customer 6', 'Customer 7', 'Customer 8']
arTime: 3 current time: 29
serTime: 14 current time: 29
current time: 29
['Customer 5', 'Customer 6', 'Customer 7', 'Customer 8', 'Customer 9']
current time: 30
['Customer 5', 'Customer 6', 'Customer 7', 'Customer 8', 'Customer 9']
```

Q4: Queuing Model Simulation-VIII

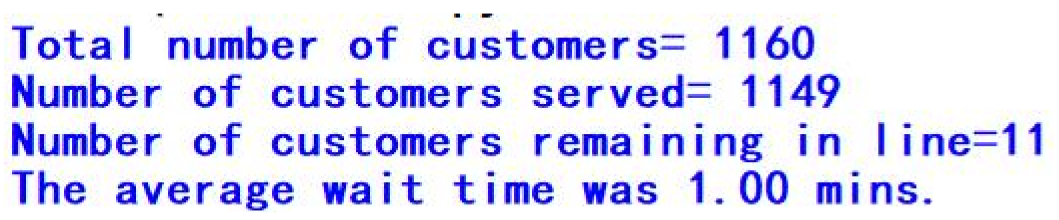
- Sample Run: timeUnit =0.1 second, run totally for 120 mins(120*600 seconds)(it should be longer to satisfy the assumption for theoretical result(t->∞)), print out the length of the queue at 120 mins, and the average waiting time for this simulation.

Trial 1



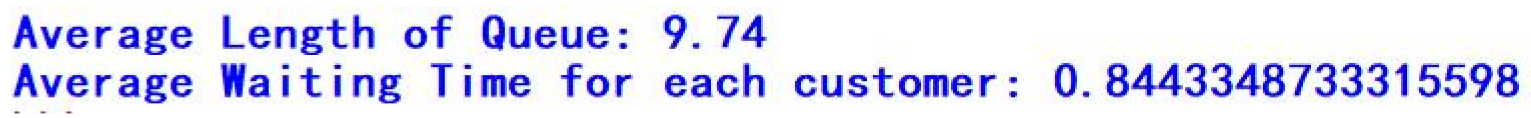
```
Total number of customers= 1088
Number of customers served= 1067
Number of customers remaining in line=21
The average wait time was 0.64 mins.
```

Trial 2



```
Total number of customers= 1160
Number of customers served= 1149
Number of customers remaining in line=11
The average wait time was 1.00 mins.
```

- Sample Run: Repeat the above process for 100 times(it should be a lot more but it is really time consuming) and average the results.



```
Average Length of Queue: 9.74
Average Waiting Time for each customer: 0.844334873315598
```