Overview

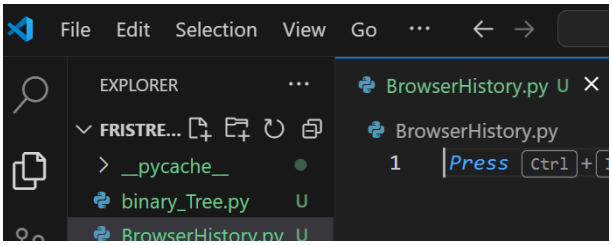
In this worksheet, we will implement the concepts of data structures. The provided Python code implements a basic web browsing history program using a LifoQueue to simulate stacks for backward and forward navigation. Initially, two LifoQueue objects are created to store the backward and forward history, along with a variable to track the current page. The user is prompted to enter the number of URLs that they have recently visited and is in the history list.

Pre-requisites:

1. Github account
2. Git
3. VSCode
4. Python
5. Basic understanding of Python syntax.
6. Basic understanding of abstract data structures
7. Familiarity with git commands

Instructions:

1. **Open the cloned folder in VSCode**.
    a. Create a python file called “BroswerHistory.py”



2. **Write a program**: Write the following code in the file created above.
    a. Import the LifoQueue class from the queue module. LifoQueue is a type of queue where elements are retrieved in a last-in-first-out (LIFO) order.

```
1   from queue import LifoQueue
2
```

    b. Then an instance of LifoQueue called backward_history and forward_history is created which will store the backward and forward navigation history respectively. Additionally, a variable called current_page is initialized to None, indicating that there is no current page initially.

```
3   backward_history = LifoQueue()
4   forward_history = LifoQueue()
5   current_page = None
6
```

    c. Prompts the user to enter the number of URLs that they have visited and stores the input as an integer in the variable NoOfVisits.

```
7   # Visit function
8   NoOfVisists = int(input("Enter the number of url history: "))
```

d. Then, print a message to prompt the user to enter the url history that they last visited. A loop will iterate NoOfVisits times, allowing the user to input multiple URLs and store the input in the variable url. The line “print(f"Visiting {url}")”, prints a message indicating that the program is visiting the entered URL. The line “backward_history.put(current_page)”, adds the current page (if any) to the backward history before updating the current page with the newly entered URL. The line “current_page = url”, updates the current_page variable with the newly entered URL.

```
9   print("Enter URLs to visit:")
10  for i in range(NoOfVisists):
11      url = input("URL: ")
12      print(f"Visiting {url}")
13      backward_history.put(current_page)
14      current_page = url
```

e. Print the url of the current page:

```
15
16  # Display current page
17  print(f"Current page: {current_page}")
18
```

f. The while loop will continue as long as the user inputs 'yes' when prompted to go back. Then checks if the backward_history stack is not empty, ensuring that there are previous pages available to navigate. Then, adds the current page to the forward history before navigating back, allowing the user to navigate forward later if needed. After that, retrieves the previous page from the backward history and updates the current_page variable accordingly. Additionally, prints a message indicating that the program is going back to the previous page.
Otherwise, prints a message indicating that there are no previous pages available to navigate back to.

```
19  # Go back
20  while input("Do you want to go back? (yes/no): ").lower() == 'yes':
21      if not backward_history.empty():
22          forward_history.put(current_page)
23          current_page = backward_history.get()
24          print(f"Going back to {current_page}")
25      else:
26          print("No previous page available")
27
```

g. A while loop continues as long as the user inputs 'yes' when prompted to go forward. Then checks if the forward_history stack is not empty, ensuring that there are forward pages available to navigate. And then adds the current page to the backward history before navigating forward, allowing the user to navigate back later if needed. Then, retrieves the next page from the forward history and updates the current_page variable accordingly. Additionally prints a message indicating that the program is going forward to the next page.
Otherwise prints a message indicating that there are no forward pages available to navigate forward to.

```
28  # Go forward
29  while input("Do you want to go forward? (yes/no): ").lower() == 'yes':
30      if not forward_history.empty():
31          backward_history.put(current_page)
32          current_page = forward_history.get()
33          print(f"Going forward to {current_page}")
34      else:
35          print("No forward page available")
```

Exercise:
Create a Python program implementing the concepts of queue that automates a desk manager's system for patient registration and appointment scheduling. The program should allow the desk manager to register patients, remove patients after they meet the doctor, and display the current patient queue.

Instructions:

- Initialize a queue to store patient names. You can use the Queue class from the queue module.
- Implement a menu-driven program with the following options:
  Register Patient: Allows the desk manager to register a new patient by entering their name.
  Remove Patient after Meeting Doctor: Removes the next patient from the queue after they have met the doctor.
  Display Patient Queue: Displays the current queue of patients waiting to meet the doctor.
  Exit: Exits the program.
- Display appropriate messages for each action performed, such as "Patient registered", "Appointment added", "Patient removed", etc.
- Ensure that the program handles cases where the queue is empty or invalid user inputs.
- Test your program thoroughly by registering patients, adding appointments, and removing patients.

Sample Input/Output:

```
Desk Manager - Patient Registration and Appointment System
1. Register Patient
2. Remove Patient after Meeting Doctor
3. Display Patient Queue
4. Exit
Enter your choice (just enter the option number): 1
Enter patient name: t
Patient t registered.

Desk Manager - Patient Registration and Appointment System
1. Register Patient
2. Remove Patient after Meeting Doctor
3. Display Patient Queue
4. Exit
Enter your choice (just enter the option number): 1
Enter patient name: p
Patient p registered.

Desk Manager - Patient Registration and Appointment System
1. Register Patient
2. Remove Patient after Meeting Doctor
3. Display Patient Queue
4. Exit
Enter your choice (just enter the option number): 1
Enter patient name: r
Patient r registered.
```

```
Desk Manager - Patient Registration and Appointment System
1. Register Patient
2. Remove Patient after Meeting Doctor
3. Display Patient Queue
4. Exit
Enter your choice (just enter the option number): 3
Current Patient Queue:
t
p
r


Desk Manager - Patient Registration and Appointment System
1. Register Patient
2. Remove Patient after Meeting Doctor
3. Display Patient Queue
4. Exit
Enter your choice (just enter the option number): 2
Patient t removed after meeting the doctor.
```

```
Desk Manager - Patient Registration and Appointment System
1. Register Patient
2. Remove Patient after Meeting Doctor
3. Display Patient Queue
4. Exit
Enter your choice (just enter the option number): 3
Current Patient Queue:
p
r


Desk Manager - Patient Registration and Appointment System
1. Register Patient
2. Remove Patient after Meeting Doctor
3. Display Patient Queue
4. Exit
Enter your choice (just enter the option number): 4
Exiting program...
```