**Kathmandu University**

**Department of Computer Science and Engineering**

**Dhulikhel, Kavre**

****

**A Lab Report #3**

**[ Course title: COMP 307]**

**Submitted by:**

**Bisheshwor Neupane(35)**

**Submitted to:**

**Mr. Dhiraj Shrestha**

**Department of Computer Science and Engineering**

**Submission Date:**

**August 09, 2021**

1. **Simulate the concept of First fit, best fit and worst fit allocation algorithms**

**[Make necessary assumptions]**

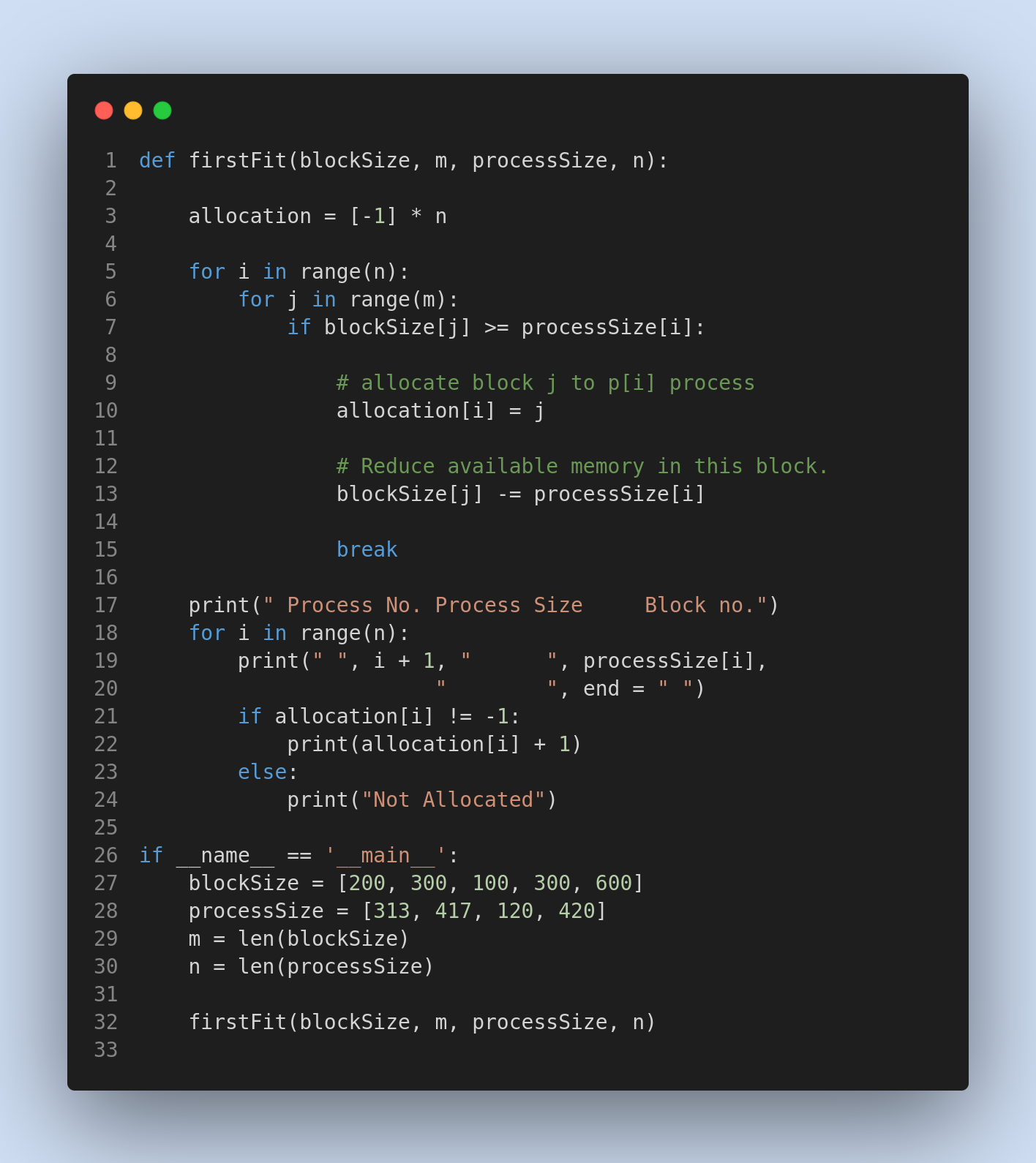
Solution:

**First fit Allocation:**

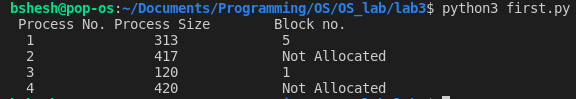
**Algorithm:**

* Input memory blocks and processes with sizes.
* Initialize all memory blocks as free.
* Start by picking each process and check if it can be assigend to current block.
* If size-of-process is less than or equals to size-of-block if yes then assign and check for next process.
* If not then keep checking the further process.

**Code Implementation in Python:**



**Output:**

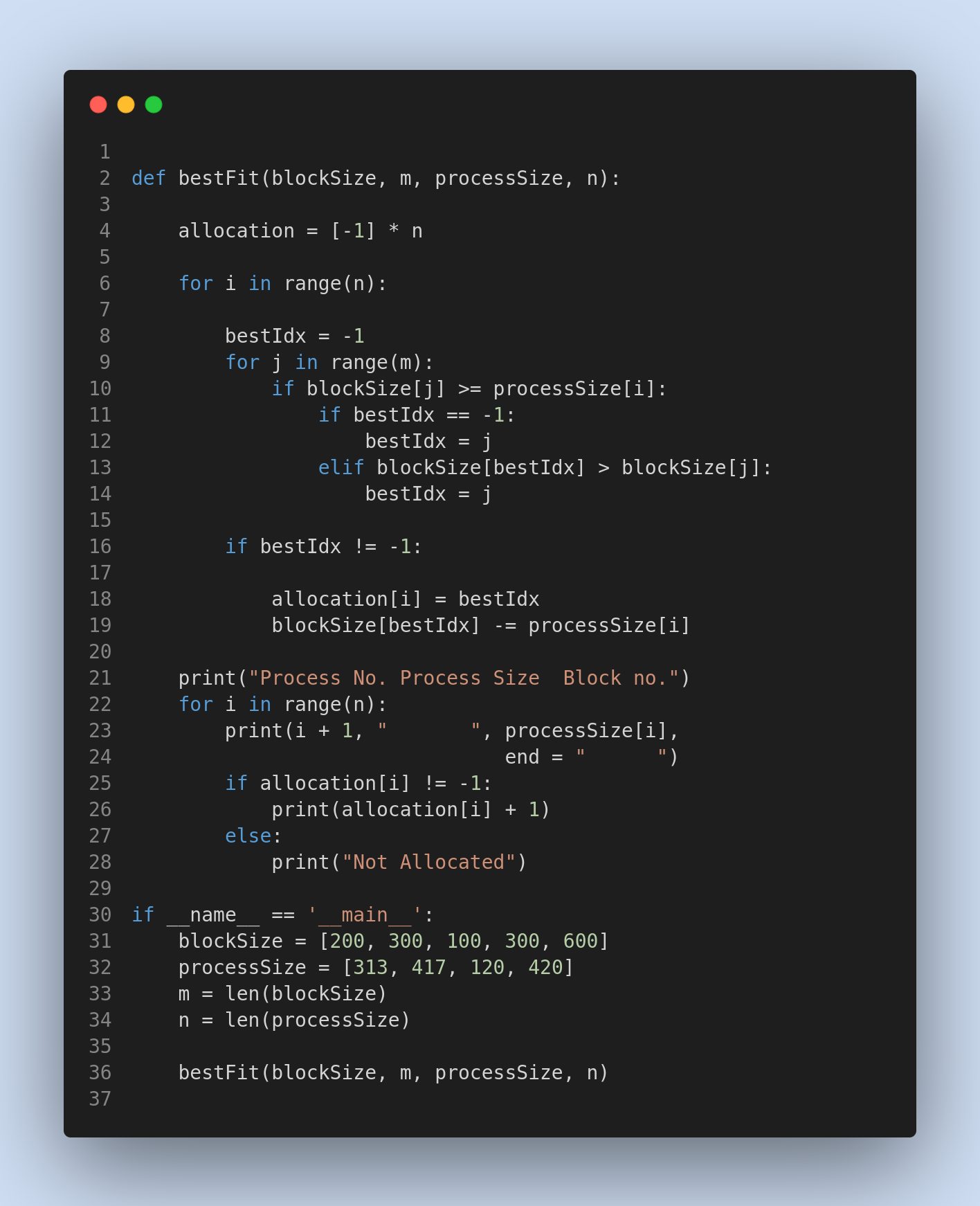


**Best Fit Allocation:**

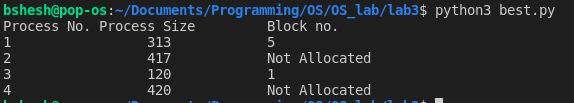
**Algorithm:**

* Input memory blocks and processes with sizes.
* Initialize all memory blocks as free.
* Start by picking each process and find the minimum block size that can be assigned to current process i.e., find min(blockSize[1], blockSIze[2], .... blockSize[n]) > processSize[current], if found then assign it to the current process.
* If not then leave that process and keep checking the further processes.

**Code Implementation in Python:**



**Output:**

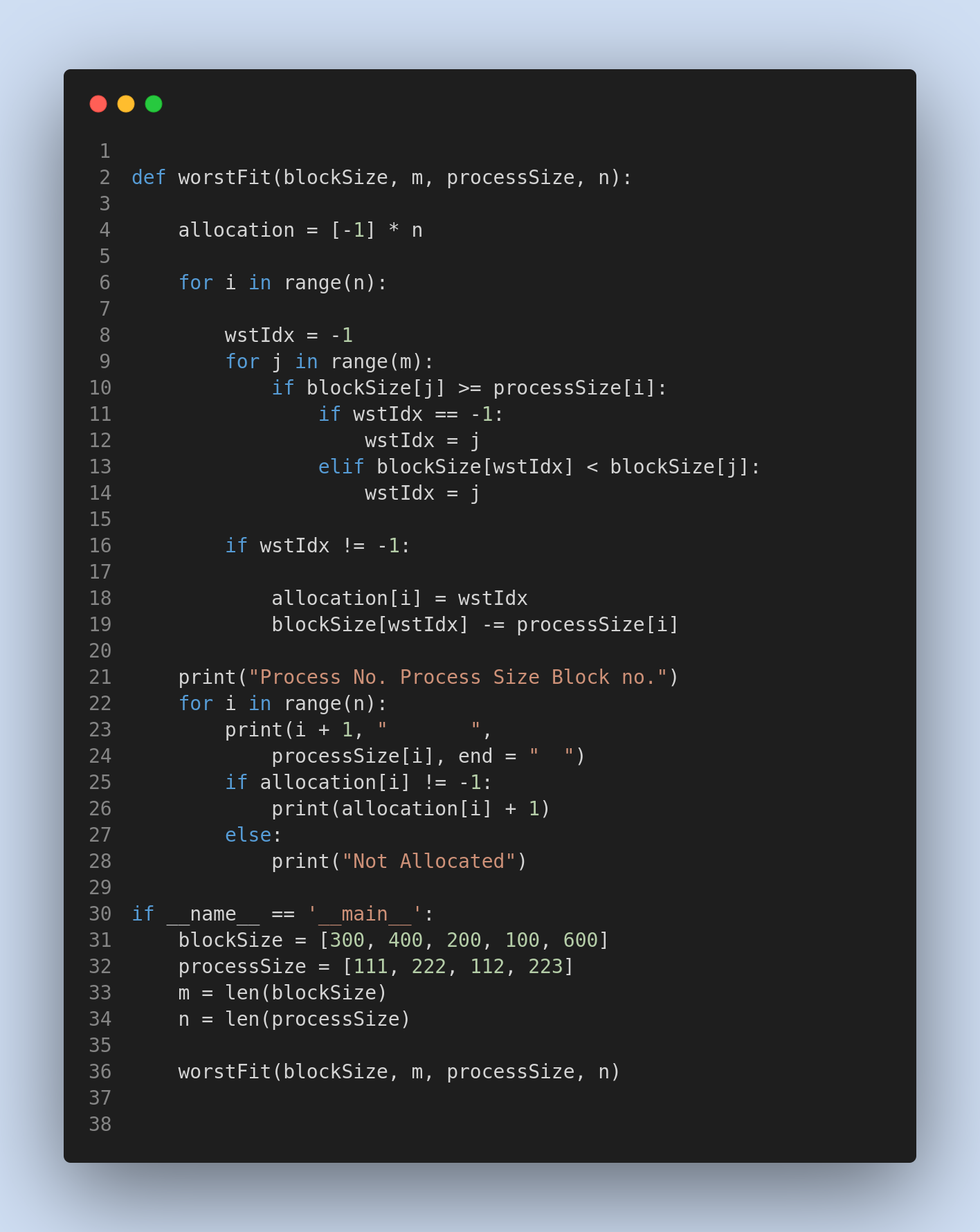


**Worst Fit Allocation:**

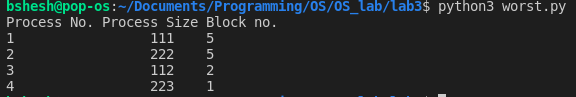
**Algorithm:**

* Input memory blocks and processes with sizes.
* Initialize all memory blocks as free.
* Start by picking each process and find the maximum block size that can be assigned to current process i.e., find max(bockSize[1], blockSize[2],.....blockSize[n]) > processSize[current], if found then assign it to the current process.
* If not then leave that process and keep checking the further processes.

**Code Implementation in Python:**



**Output:**



1. **Simulate the concept of following page replacement algorithms: [Make necessary assumptions]**
   1. **FIFO**
   2. **Optimal**
   3. **LRU**
   4. **Second Chance page replacement algorithm**

Solutions:

1. **FIFO**

Algorithm:

1- Start traversing the pages.

i) If set holds less pages than capacity.

a) Insert page into the set one by one until

the size of set reaches capacity or all

page requests are processed.

b) Simultaneously maintain the pages in the

queue to perform FIFO.

c) Increment page fault

ii) Else

If current page is present in set, do nothing.

Else

a) Remove the first page from the queue

as it was the first to be entered in

the memory

b) Replace the first page in the queue with

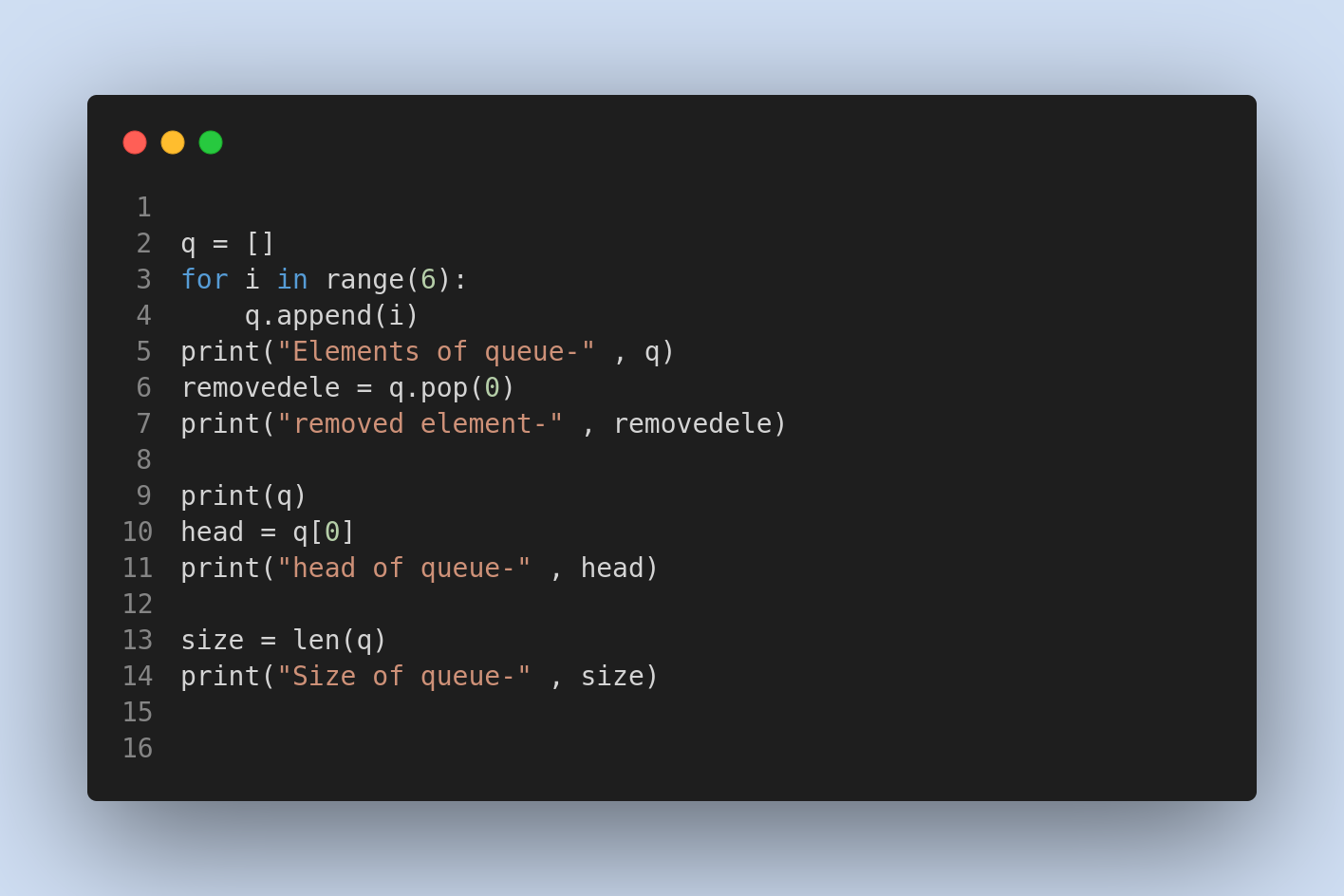
the current page in the string.

c) Store current page in the queue.

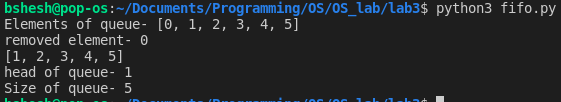
d) Increment page faults.

2. Return page faults.

**Code Implementation in Python:**



**Output:**



1. **Optimal**

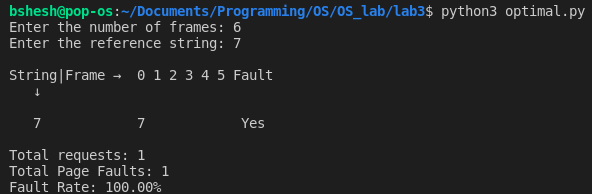
**Algorithm:**

1. If referred page is already present, increment hit count.
2. If not present, find if a page that is never referenced in future. If such a page exists, replace this page with new page. If no such page exists, find a page that is referenced farthest in future. Replace this page with new page.

**Code Implementation in Python:**

****

**Output:**



1. **LRU**

**Algorithm:**

Let **capacity** be the number of pages that

memory can hold. Let **set** be the current

set of pages in memory.

1- Start traversing the pages.

i) **If set holds less pages than capacity.**

a) Insert page into the set one by one until

the size of **set** reaches **capacity** or all

page requests are processed.

b) Simultaneously maintain the recent occurred

index of each page in a map called **indexes**.

c) Increment page fault

ii) **Else**

**If** current page is present in **set**, do nothing.

**Else**

a) Find the page in the set that was least

recently used. We find it using index array.

We basically need to replace the page with

minimum index.

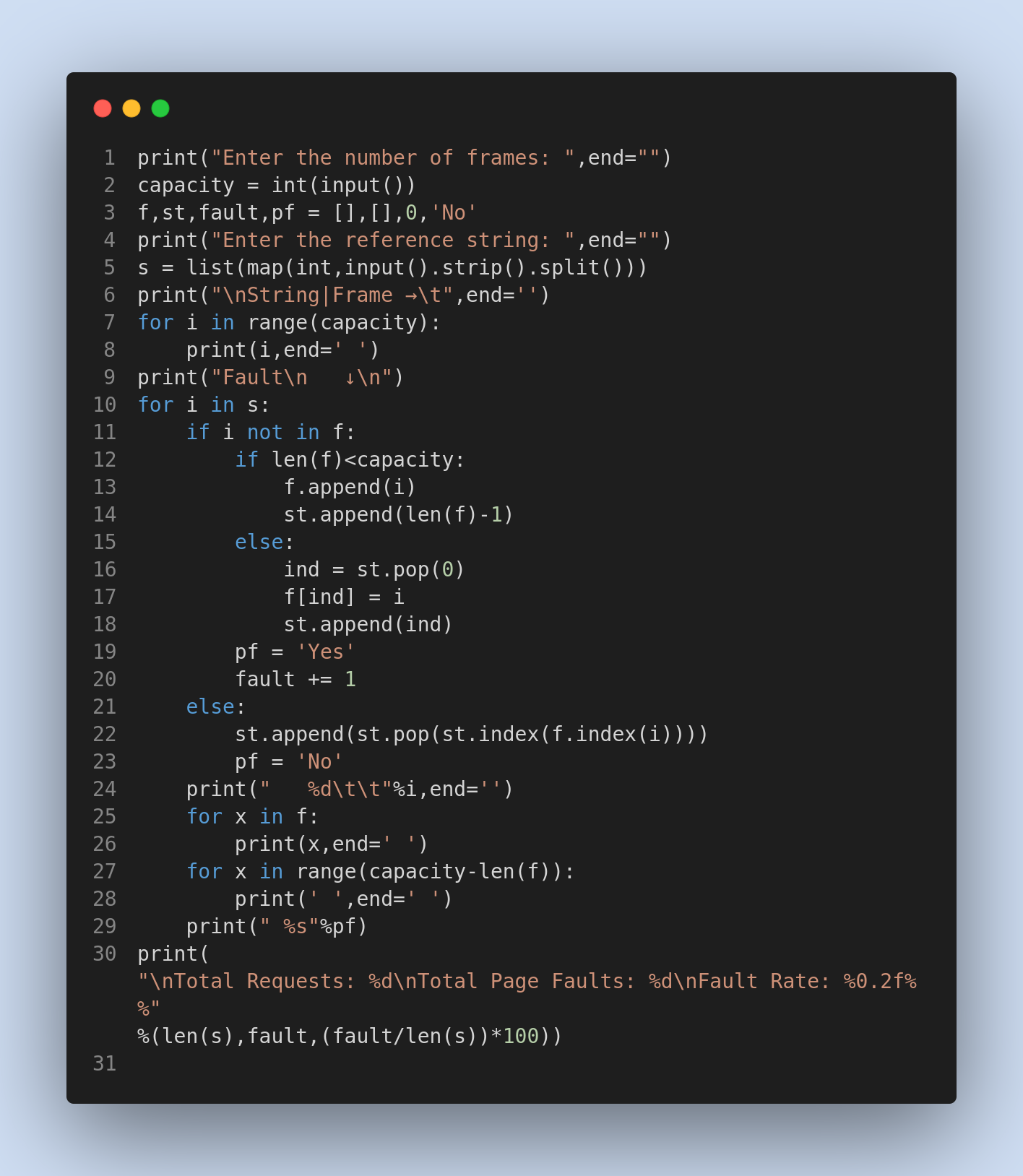
b) Replace the found page with current page.

c) Increment page faults.

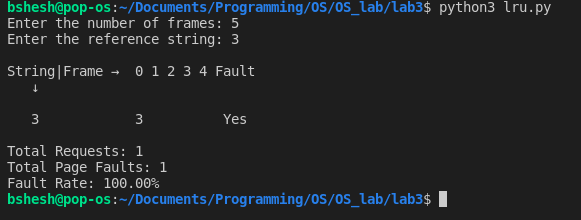
d) Update index of current page.

2. Return page faults.

**Code Implementation in Python:**



**Output:**



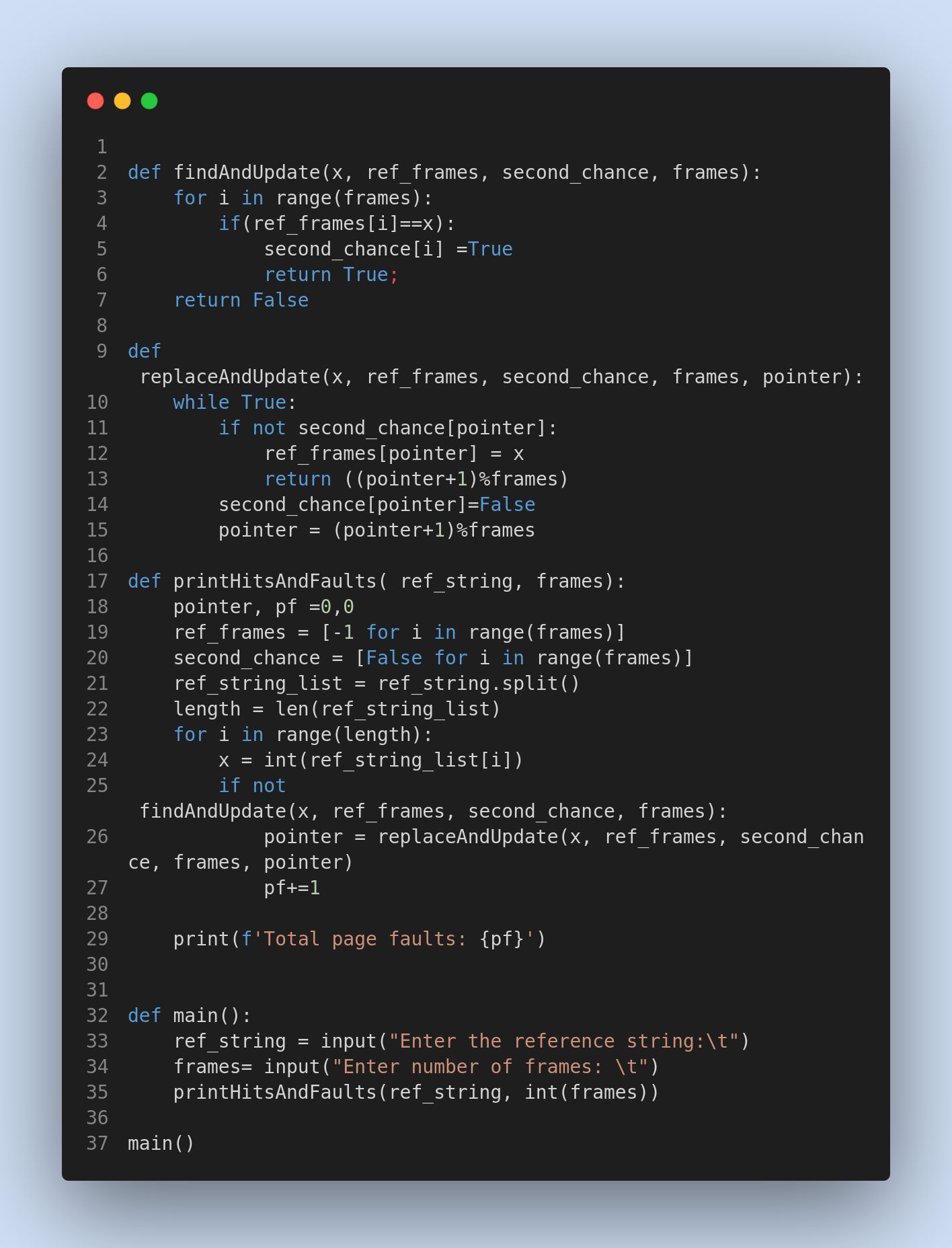
**D.Second Chance page replacement algorithm**

**Algorithm:**

Create an array **frames** to track the pages currently in memory and another Boolean array **second\_chance** to track whether that page has been accessed since it’s last replacement (that is if it deserves a second chance or not) and a variable **pointer** to track the target for replacement. 

1. Start traversing the array **arr**. If the page already exists, simply set its corresponding element in **second\_chance** to true and return.
2. If the page doesn’t exist, check whether the space pointed to by **pointer** is empty (indicating cache isn’t full yet) – if so, we will put the element there and return, else we’ll traverse the array **arr** one by one (cyclically using the value of **pointer**), marking all corresponding **second\_chance** elements as false, till we find a one that’s already false. That is the most suitable page for replacement, so we do so and return.
3. Finally, we report the page fault count.

**Code Implementation in Python:**



**Output:**

