Data Structures in Python Chapter 3

- Stack Concept and ADT
- Stack Example - Matching
- Stack Example - Postfix
- **Queue**
- Deque
- Deque Profiling
- Circular Queue
- Linked list
- Unordered List
- Ordered List and Iterator

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204 Handong Global University

Agenda

- Introduction
- Queue Abstract Data Type (ADT)
- Implementing a queue using a list

Can you think of other examples of queues?

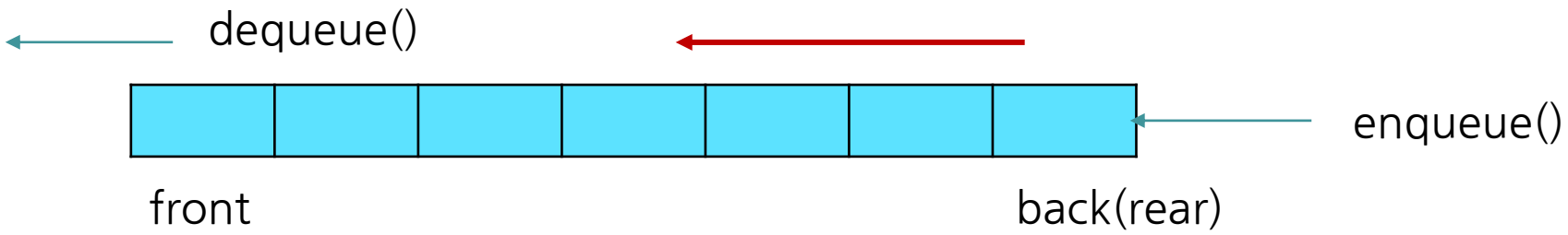
Introduction - What is a Queue?

- Queues are appropriate for many real-world situations.
  - Example: A line to buy a movie ticket
  - Computer applications, e.g., a request to print a document

- A queue is an **ordered collection** of items where the addition of new items happens at **one end** (the back of the queue) and the removal of existing items always takes place at the other end (the front of the queue).
  - New items enter at the back(or rear) of the queue.
  - Items leave from the front of the queue.
  - **First-in, first-out (FIFO)** property:
    - The first item inserted into a queue is the first item to leave.

Introduction - What is a Queue?

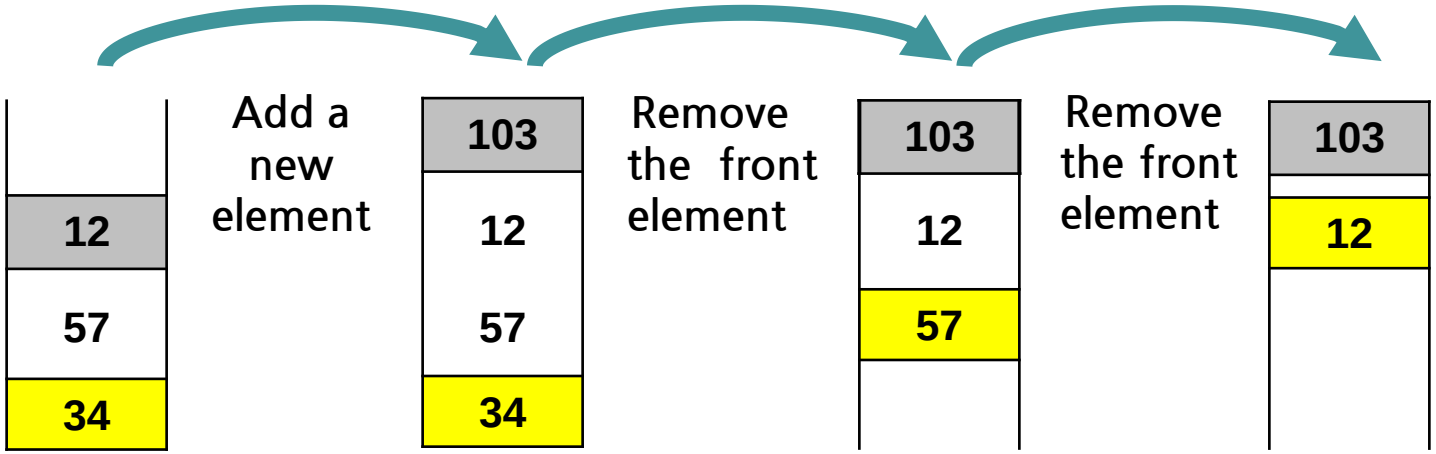
- Queues implement the FIFO (first-in first-out) policy:
  - For example: the printer / job queue!



Introduction - Queue Example

- Add only to the back of a Queue.
- Remove only from the front of the Queue.
  - Note: The last item placed on the queue will be the last item

First In - First Out (FIFO)



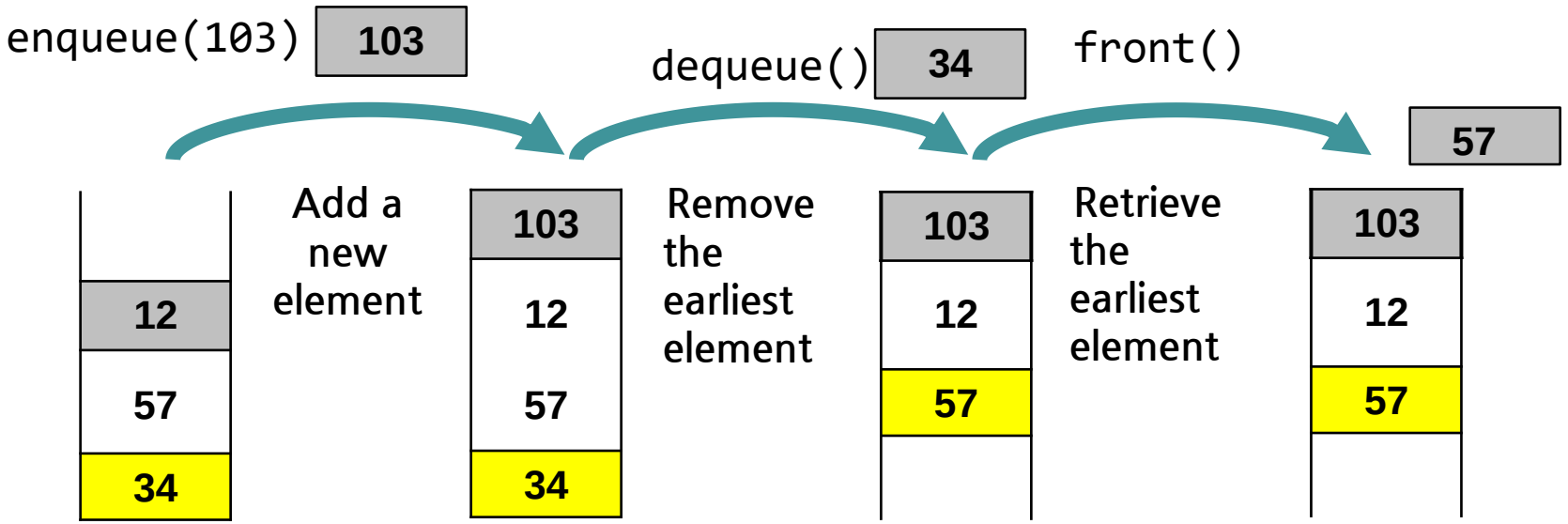
Introduction - ADT Queue Operations

- What are the operations which can be used with a Queue Abstract Data?
  - Create an empty queue:
  - Determine whether a queue is empty:
    - is_empty()
  - Count how many items in the queue:
    - size
  - Add a new item to the queue:
    - enqueue
  - Remove from the queue the item that was added earliest:
    - dequeue
  - Retrieve from the queue the item that was added earliest:
    - front
  - Retrieve from the queue the item that was added latest:
    - back

Introduction - ADT Queue Operations

- What are the operations which can be used with a Queue Abstract Data?

First In - First Out (FIFO)



The Queue Abstract Data Type

- Queue() creates a new queue that is empty.
  - It needs no parameters and returns an empty queue.
- enqueue(item) adds a new item to the back of the queue.
  - It needs the item and returns nothing.
  - The queue is modified.
- dequeue() removes the front item from the queue.
  - It needs no parameters and returns the item.
  - The queue is modified.

Queue(), enqueue(item) and **dequeue()** are critical operations in order to manipulate the elements of the queue.

The Queue Abstract Data Type

- is_empty() tests to see whether the queue is empty
  - It needs no parameters and returns a Boolean value. The queue is not modified.
- size() returns the number of items in the queue.
  - It needs no parameters and returns an integer.
- front()/back() returns the earliest or latest item from the queue, respectively
  - The queue is not modified.

is_empty(), size(), front() and back() are useful to allow the users to retrieve the properties of the queue but they are not necessary.

The Stack Abstract Data Type - Code Example - Application

- What is the output of the following code snippet?

Code	Queue Implementation	Output
`s = Queue()`	`s = []`	
`print(s.is_empty())`	`s = []`	`True`
`s.enqueue(4)`	`s = [4]`	
`s.enqueue('hat')`	`s = [4,'hat']`	
`print(s.front())`	`s = [4,'hat']`	`4`
`s.enqueue(True)`	`s = [4,'hat',True]`	
`print(s.size())`	`s = [4,'hat',True]`	`3`
`print(s.is_empty())`	`s = [4,'hat',True]`	`False`
`s.enqueue(8.4)`	`s = [4,'hat',True,8.4]`	
`s.dequeue()`	`s = ['hat',True,8.4]`	`4`
`s.dequeue()`	`s = [True, 8.4]`	`hat`
`print(s.size())`	`s = [True, 8.4]`	`2`

The Stack Abstract Data Type - Exercise 1

- What is the output of the following code snippet?

Code

```
s = Queue()
print(s.is_empty())
s.enqueue(4)
s.enqueue('hat')
print(s.front())
print(s.size())
print(s.is_empty())
s.dequeue()
s.enqueue(3)
s.dequeue()
print(s.size())
```

Queue Implementation

```
s =
s =
s =
s =
s =
s =
s =
s =
s =
s =
s =
```

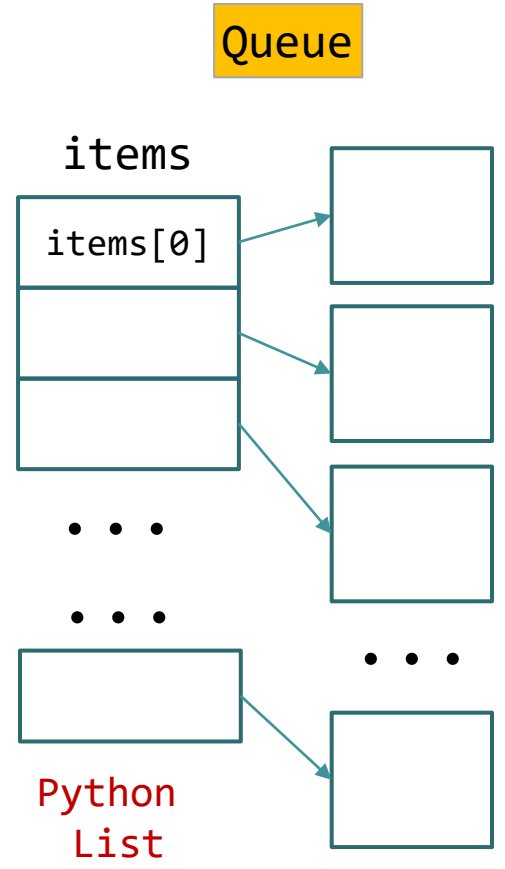
front back

Output

The Queue Implementation - Code Example - Application

- We use the Python list object to implement the queue.

```
class Queue:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def size(self):
        return len(self.items)
    ...
```



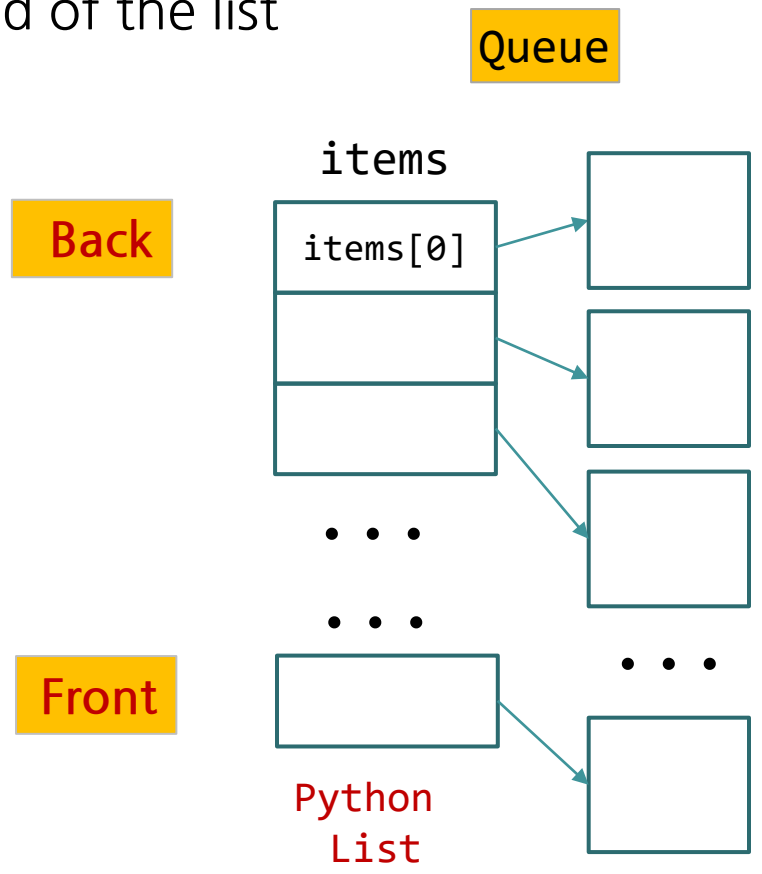
The Queue Implementation - Code Example - Application

- We use the Python list object to implement the queue.
- **Version 1**
  - The addition of new items takes place at the beginning of the list
  - The removal of existing items takes place at the end of the list

```
class Queue:
    ...
    def enqueue(self, item):
        self.items.insert(0, item)

    def dequeue(self):
        return self.items.pop()
    ...
```

Big-O?
enqueue()/search(): O(n)
dequeue()/front(): O(1)



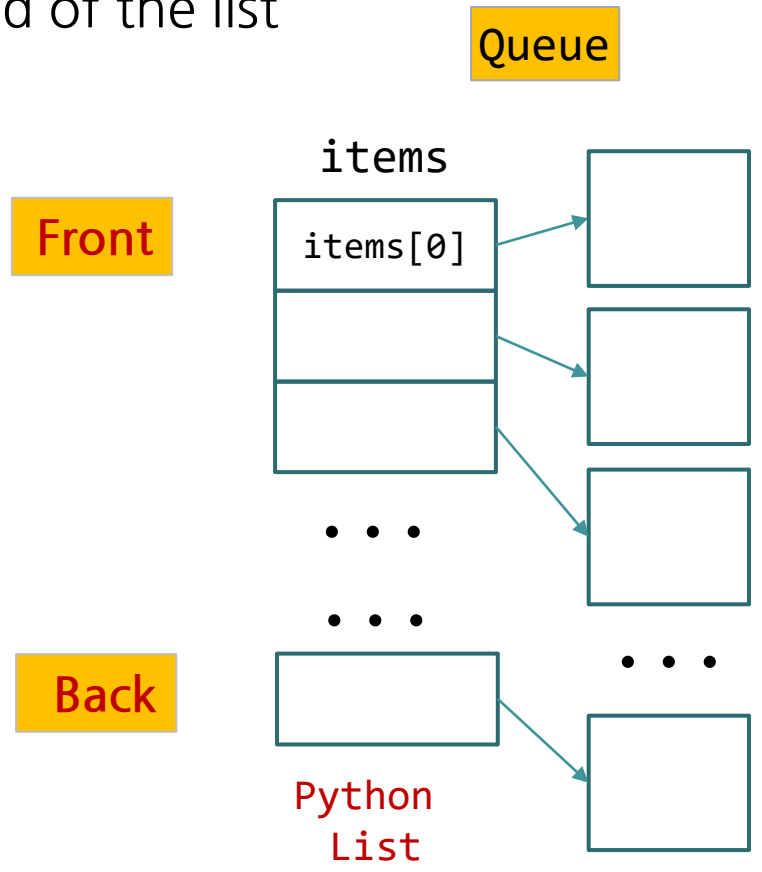
The Queue Implementation - Code Example - Application

- We use the Python list object to implement the queue
- **Version 2**
  - The addition of new items takes place at the beginning of the list
  - The removal of existing items takes place at the end of the list

```
class Queue:
    ...
    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        return self.items.pop(0)
    ...
```

Big-O?
enqueue()/front()/back(): O(1)
dequeue()/search(): O(n)



The Queue Implementation - Code Example - Application

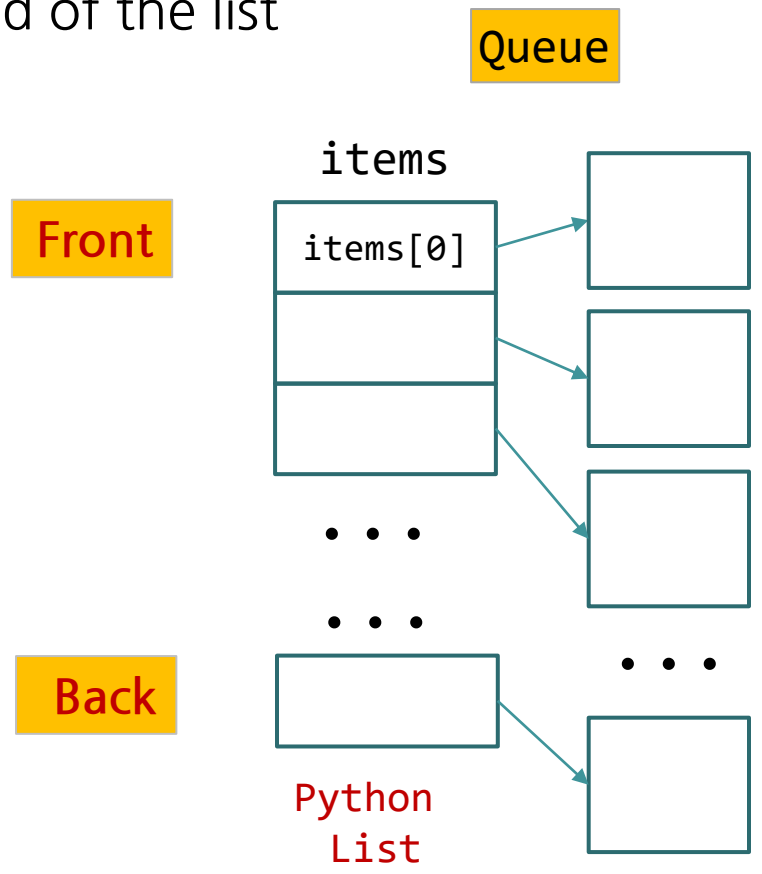
- We use the Python list object to implement the queue
- Version 2
  - The addition of new items takes place at the beginning of the list
  - The removal of existing items takes place at the end of the list

```
class Queue:
    ...
    def back(self):
        self.items[-1]

    def dequeue(self):
        return self.items[0]

    def is_empty(self):
        return self.items == 0
```

Big-O?
enqueue()/front()/back(): O(1)
dequeue()/search(): O(n)



The Queue Implementation - Exercise 2

- What is the output of the following code snippet?

Code:

```
from dspy import Queue
try:
    q = Queue()
    q.enqueue(2)
    q.enqueue(4)
    q.enqueue(6)
    while not q.is_empty():
        print(q.dequeue())
except IndexError:
    print ('empty queue')
```

Output:

The Queue Implementation - Exercise 2 solution

- What is the output of the following code snippet?

Code:

```
from dspy import Queue
try:
    q = Queue()
    q.enqueue(2)
    q.enqueue(4)
    q.enqueue(6)
    while not q.is_empty():
        print(q.dequeue())
except IndexError:
    print ('empty queue')
```

Output: 2 4 6

The Queue from queue in Python

- Python has a built-in module called queue that serves a class called Queue.
- The Queue class methods:
  - put(data) - adds or pushes the data to the queue
  - get() - removes the first element from the queue and returns it
  - empty() - returns whether the stack is empty or not
  - qsize() - returns the length of the queue.

The Queue from queue in Python

- Python has a built-in module called queue that serves a class called Queue.

Code:

```
from queue import Queue

queue_object = Queue()
queue_object.put(1)
queue_object.put(2)
queue_object.put(3)

print(queue_object.get())
print(queue_object.get())

print("Size", queue_object.qsize())
print(queue_object.get())
print(queue_object.empty())
```

Comparisons between Queue & Stack

- Behavior:
  - The behavior of a **stack** is like a **Last-In-First-Out (LIFO)** system.
  - The behavior of a **queue** is like a **First-In-First-Out (FIFO)** system.
- Implementation with Python list:
  - The list methods make it very easy to use a list as a stack
    - To add an item to the top of the stack, using **append()**
    - To retrieve an item from the top of the stack, using **pop()** without an explicit index.
- It is **not** efficient to use a list as a queue.
  - To add or remove an item from the end of list are fast, using **append()** and **pop().**
  - To add or remove an item at the beginning of list are slow (because all of the other elements have to be shifted by one).

Comparisons between Queue & Stack

- Big O:
  - Stack
    - **push(): O(1)**
    - **pop(): O(1)**
    - peek(): O(1)
    - search(): O(n)
  - Queue (best scenario)
    - **enqueue(): O(n)**
    - **dequeue(): O(1)**
    - front(): O(1)
    - back(): O(1)
    - search(): O(n)

Summary

- The definition of the queue operations gives the ADT queue first-in, first-out (FIFO) behavior.
- Python list supports a simple implementations of queue.
- Python provides its own queue objects:
  - but does not follow the conventional terminology:
  - https://docs.python.org/3/library/queue.html