Data Structures in Python Chapter 3

Agenda

- Deque in Python
  - Abstract Data Type
  - Time Complexity: List vs Deque
  - Building Efficient Queues and Stacks

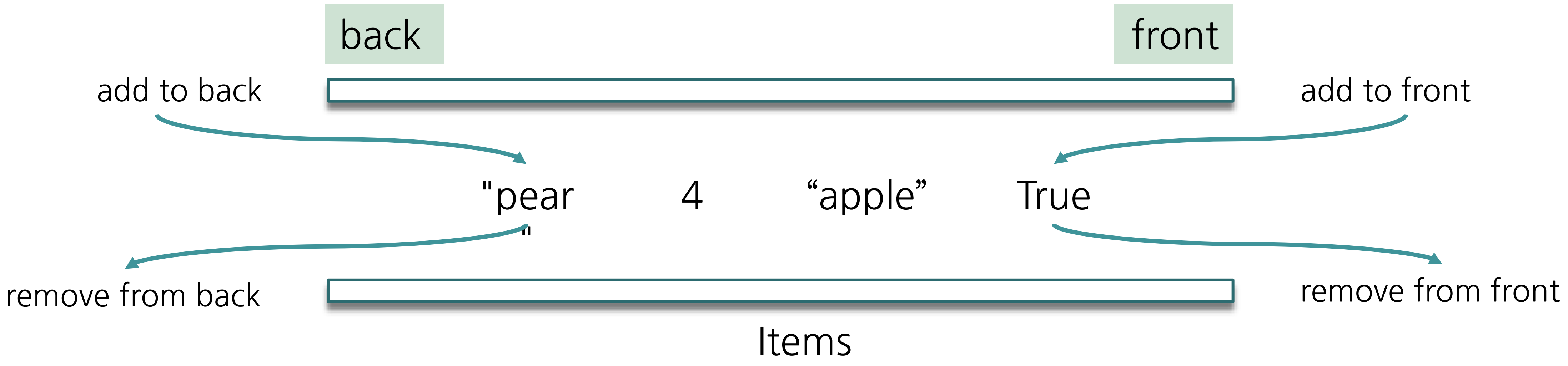
Deque

- **Double Ended Queue (pronounced like 'deck')**
  - A deque is an ordered collection of items where items are added and removed from either end, either front or back(rear).
- The newest item is at one of two ends.
  - It is implemented as a **doubly linked list** internally.

A Deque of Python Data Objects

Deque

- **Double Ended Queue (pronounced like 'deck')**
  - A deque is an ordered collection of items where items are added and removed from either end, either front or back(rear).
- The newest item is at one of two ends.
  - It is implemented as a **doubly linked list** internally.



A Deque of Python Data Objects

Deque Time Complexity

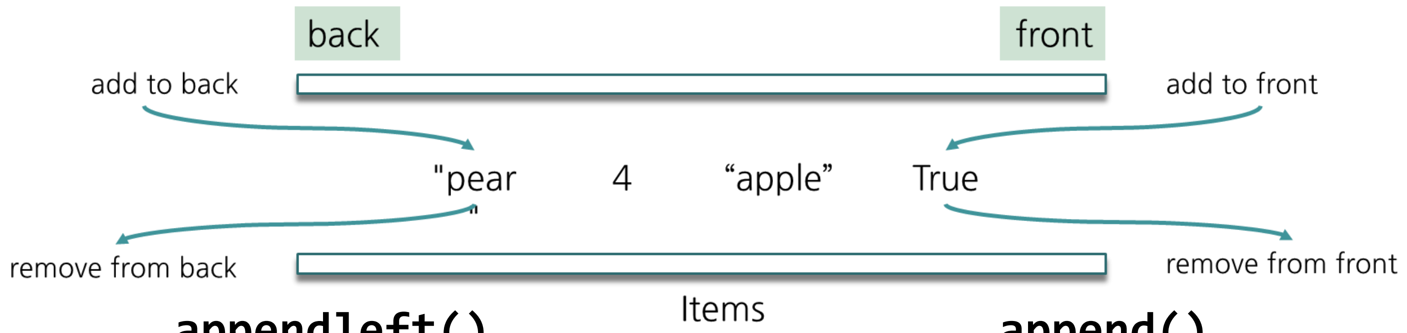
- **Double Ended Queue (pronounced like 'deck')**
  - It is specially designed to provide fast and memory-efficient ways to append and pop item from both ends of the underlying data structure.
  - It is useful for implementing elegant, efficient, and Pythonic queues and stacks,

Operation	deque	list
Pop and append items on the **left end**	$O(1)$	$O(n)$
Pop and append items on the **right end**	$O(1)$	$O(1)$ + reallocation
Insert and delete items in the **middle**	$O(n)$	$O(n)$
Access **arbitrary items** through indexing	$O(n)$	$O(1)$
	a doubly linked list	an array

Time Complexity of Python Deque Implementation

Deque ADT

- Create an empty deque:
- Determine whether a deque is empty:
- Add a new item to the deque:
  - **`append()`** - adds a new item to the right end (front) of the deque.
  - **`appendleft()`** - adds a new item to the left end (rear, back) of the deque.
- Remove a new item from the deque:
  - **`pop()`** - remove an argument from the right end of the deque.
  - **`popleft()`** - remove an argument from the left end of the deque.
- count() - counts the number of occurrences of the value passed by an argument.
- insert(i, a) - inserts the value mentioned in arguments(a) at index(i) specified in arguments.



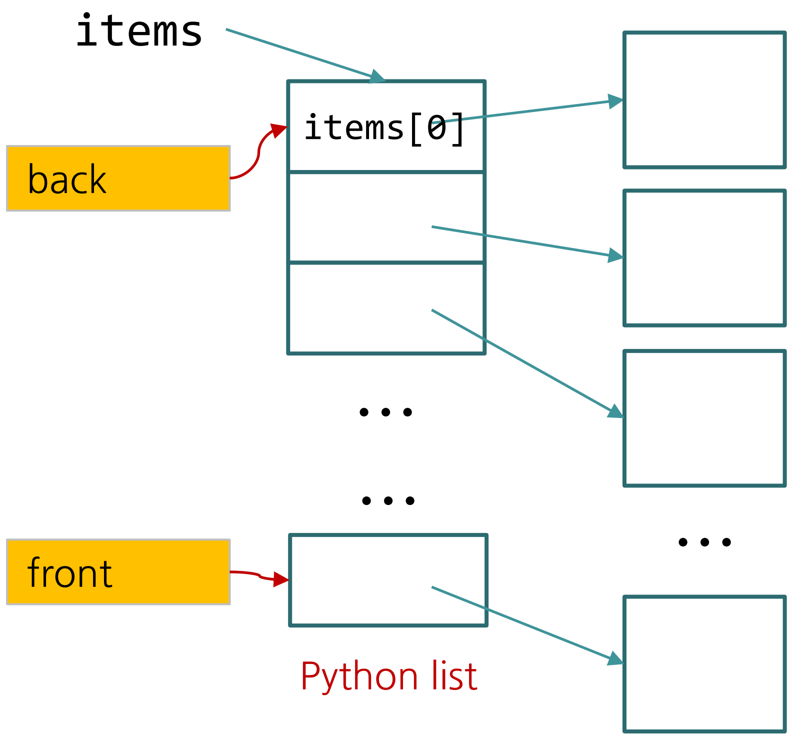
`appendleft()`
`popleft()`

`append()`
`pop()`

Deque - Implementing using Python list

- If we use Python **`list`** class to implement the **deque**, …

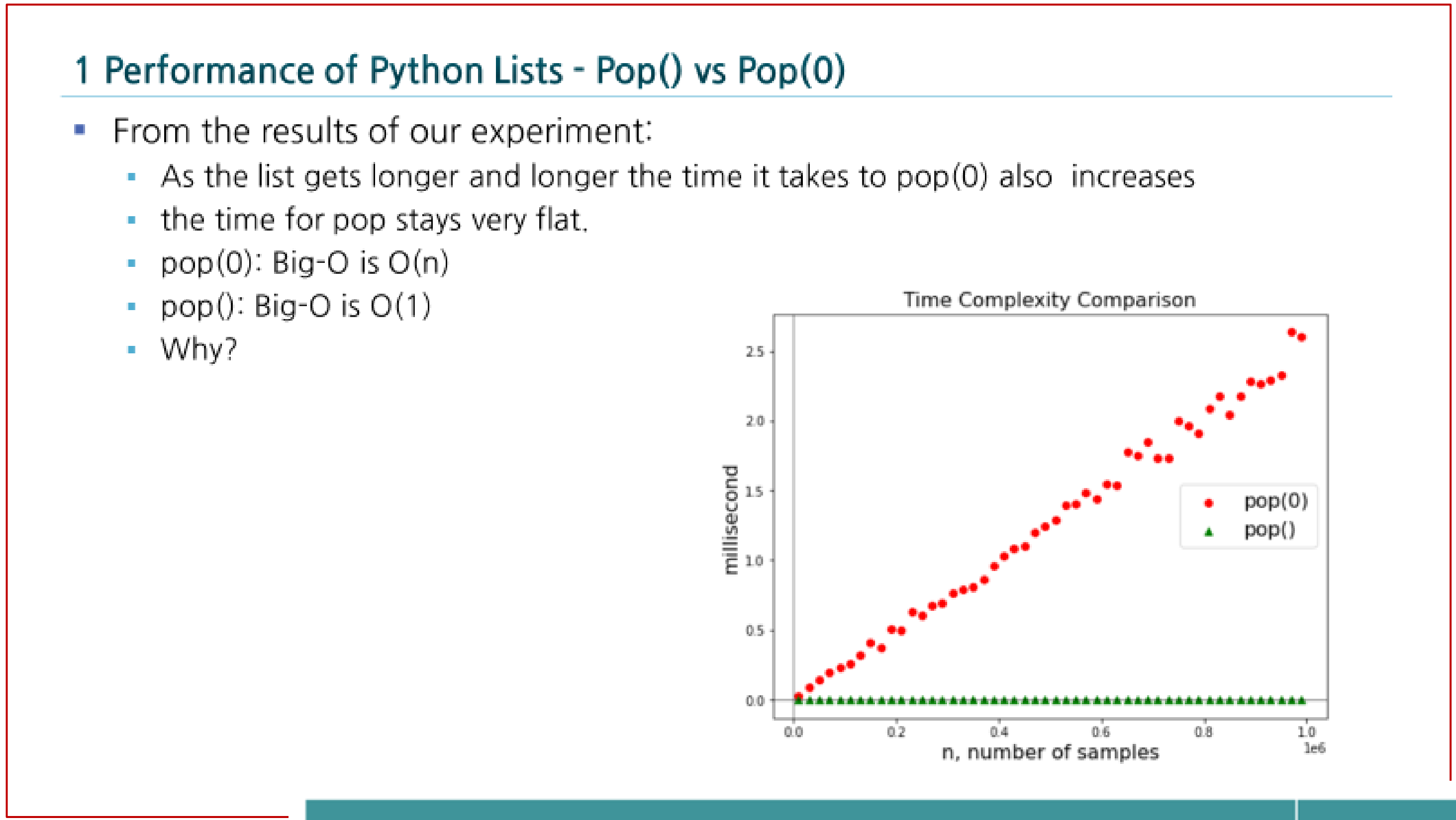
```
class Deque:
    def __init__(self):
        self.items = []
    ...
    def append(self, item):
        self.items.append(item)

    def appendleft(self, item):
        self.items.insert(0,item)

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

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



Big-O?
- append()/pop(): **O(1)**
- appendleft/popleft(): **O(n)**

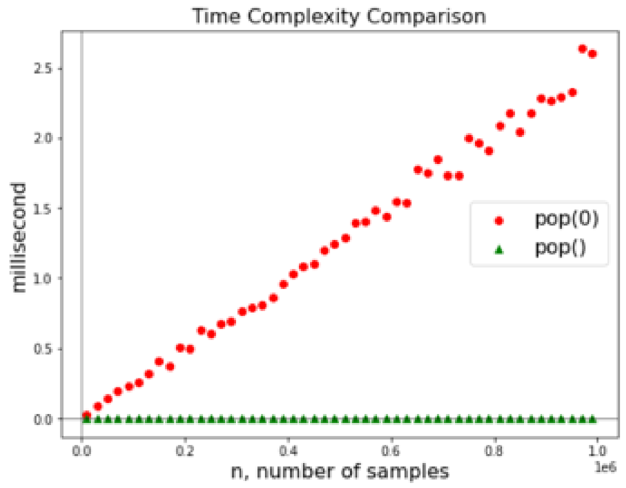
Deque - Implementing using Python list

- If we use Python **`list`** class to implement the **`deque`**, …



1 Performance of Python Lists - Pop() vs Pop(0)

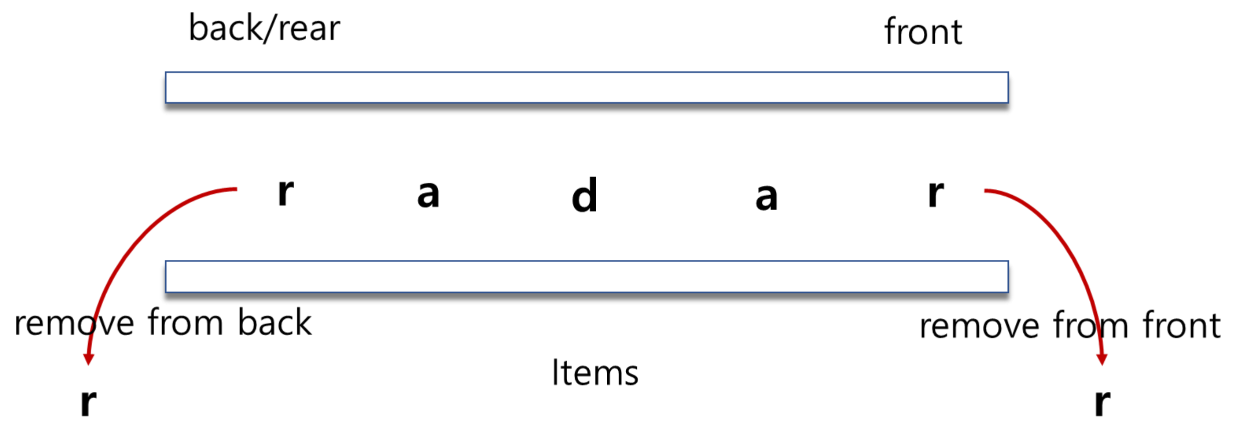
- From the results of our experiment:
  - As the list gets longer and longer the time it takes to pop(0) also increases
  - the time for pop stays very flat.
  - pop(0): Big-O is O(n)
  - pop(): Big-O is O(1)
  - Why?



Operation	deque	list
Pop and append items on the left end	$O(1)$	$O(n)$
Pop and append items on the right end	$O(1)$	$O(1)$ + reallocation

Deque - Palindrome Checker

- A string which reads the same either left to right, or right to left is known as a palindrome.
  - Radar
  - deed
  - a dog, a plan, a canal:pagoda

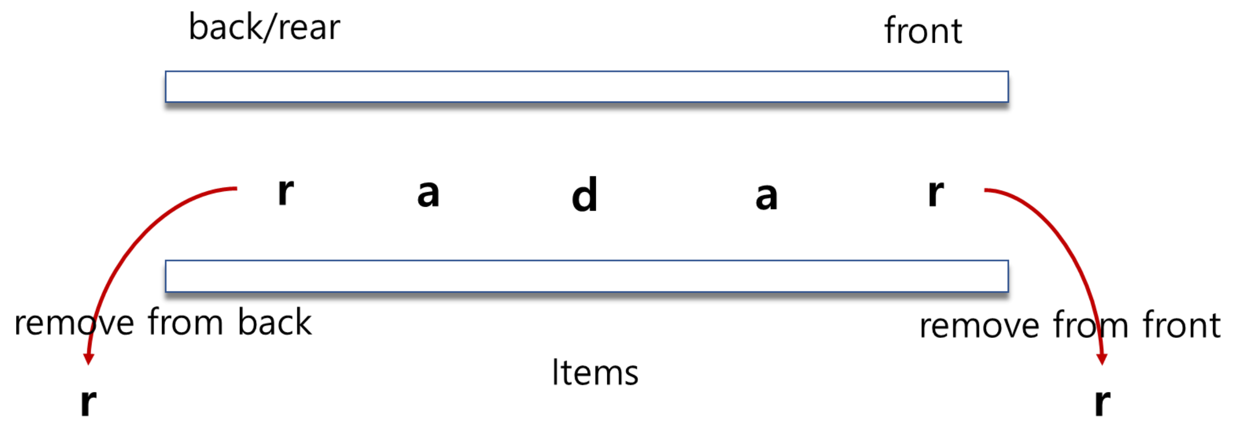


Remove from front and back

Deque - Palindrome Checker

Algorithm:

- Create a **deque** to store the characters of the string.
  - The front of the deque will hold the first character of the string and the back of the deque will hold the last character.
- Remove both of them directly, we can compare them and continue only if they match.
  - If we can keep matching first and the last items, we will eventually either run out of characters or be left with a deque of size 1.
  - In either case, the string must be a palindrome

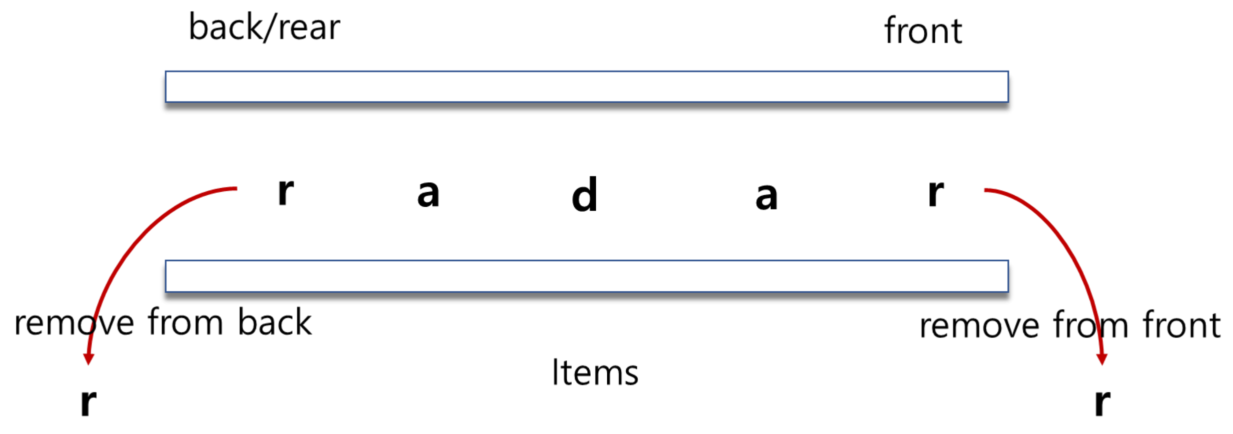


Remove from front and back

Deque - Palindrome Checker

Examples:

- print(pal_checker("hello world"))
  - Queue: h, e, l, l, o, w, o, r, l, d
  - 1st round: compare h and d => FALSE, STOP
- print(pal_checker("radar"))
  - Queue: r, a, d, a, r
  - 1st round: compare r (front) and r (back)
  - 2nd round: compare a (front) and a (back)
  - 3rd round: size() = 1, STOP, return TRUE



Remove from front and back

Deque - Palindrome Checker

Coding:

- Check:
  - The front of the deque (the first character of the string)
  - The back of the deque (the last character of the string)

```
    still_equal = True
    while char_deque.size() > 1 and still_equal:
        first = char_deque.pop()
        last = char_deque.popleft()
        if first != last:
            still_equal = False
    return still_equal
```

- It would be **a good coding exercise** if you rewrite the code such that it accepts the following list of words as palindromes.
  - Radar
  - deed
  - a dog, a plan, a canal:pagoda

Deque - maxlen

- Specify the maximum length of a given deque using the **maxlen** argument when you're instantiating the class.
  - If you supply a value to **maxlen**, then your deque will only store up to maxlen items.
  - In this case, you have a **bounded deque**.
  - Once a bounded deque is full of the specified number of items, adding a new item at either end automatically removes and discards the item at the opposite end:

- Having the option to restrict the maximum number of items allows you to use deques for tracking the latest elements in a given sequence of objects or events. For example, you can
  - **track the last five transactions** in a bank account,
  - **the last ten** open text files in an editor,
  - **the last five pages** in a browser, and more.

Deque - maxlen example

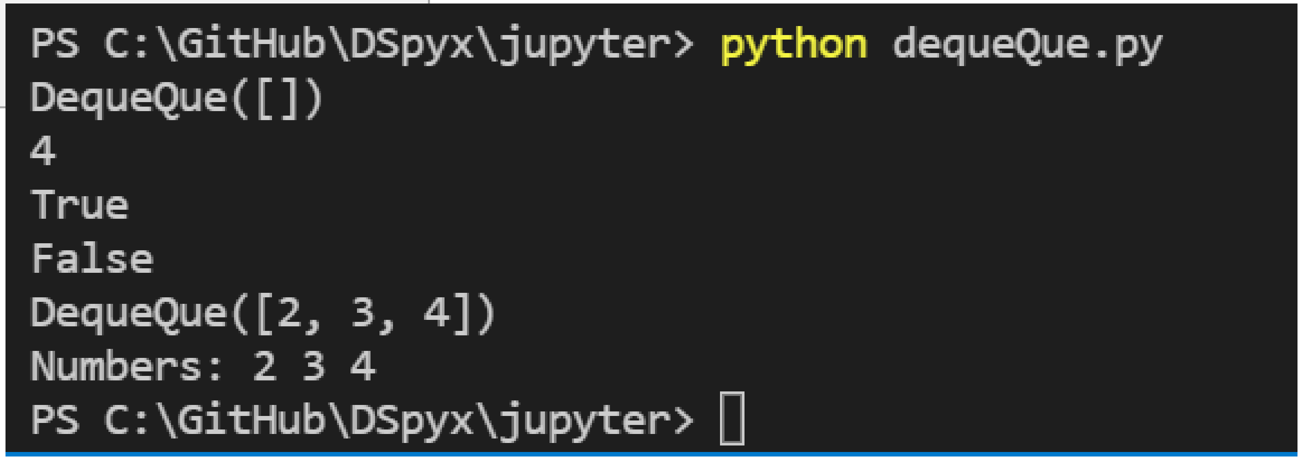
- In this example, pages keeps a list of the last three sites your application visited.
- Once pages is full, adding a new site to an end of the deque automatically discards the site at the opposite end.
- This behavior keeps your list **up to date with the last three sites** you used.

```
from collections import deque

sites = ("google.com", "yahoo.com", "bing.com")
pages = deque(maxlen=3)
print(pages.maxlen)                     # 3
for site in sites:
    pages.appendleft(site)

print(pages)                            # deque(['bing.com', 'yahoo.com', 'google.com'], maxlen=3)
pages.appendleft("handong.edu")
print(pages)                            # deque(['handong.edu', 'bing.com', 'yahoo.com'], maxlen=3)
pages.appendleft("mit.edu")
print(pages)                            # deque(['mit.edu', 'handong.edu', 'bing.com'], maxlen=3)
```

Deque - Exercise

- Define **DequeQue class** using deque such that the test code works as shown:

```
if __name__ == '__main__':
    numbers = DequeQue()         # DequeQue([])
    print(numbers)
    for number in range(1, 5):
        numbers.enqueue(number)
    print(len(numbers))          # 4
    print(2 in numbers)          # True
    print(10 in numbers)         # False
    numbers.dequeue()
    print(numbers)               # Queue([2, 3, 4])
    print('Numbers:', end = ' ')
    for number in numbers:
        print(f"{number}", end = ' ')
```



```
PS C:\GitHub\DSpyx\jupyter> python dequeQue.py
DequeQue([])
4
True
False
DequeQue([2, 3, 4])
Numbers: 2 3 4
PS C:\GitHub\DSpyx\jupyter>
```

Deque - Exercise Hints

- Read about deque in Python document
  - Define a **DequeQue** class in a file called **dequeQue.py** :

```
#%%writefile dequeQue.py
from collections import deque

class DequeQue:
    def __init__(self):
        self.items = deque()
        ...



        ...


        ...
    def __iter__(self):
        yield from self.items

if __name__ == '__main__':
    numbers = DequeQue()       # DequeQue([])
    ...
```

Summary

- The **deque (double ended queue)** in Python was designed to guarantee efficient **append and pop** operations on either end of the sequence in O(1).
- The **deque** in Python is ideal for implementing **queue and stack** data structures.
- Now, you may decide when to use `deque` instead of `list`.