Python 2: More Abstract Data Types

IN608: Intermediate Application Development Concepts

Last Session's Content

- Abstract data types
  - List
  - Tuple
  - Set
  - Dictionary
- OOP recap
  - Access modifiers
  - Encapsulation
  - Abstraction
  - Single inheritance
  - Multiple inheritance
  - Multi-level inheritance
  - Polymorphism

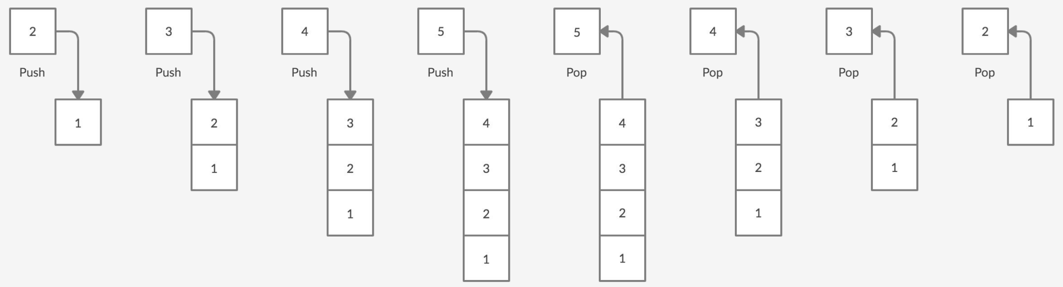
Today’s Content

- More abstract data types
  - List as a stack
  - Stack class
  - List as a queue
  - Queue class
  - Circular queue

More Abstract Types

Stack

- Last in, first out (LIFO)
- Two primary operations:
  - push
  - pop
- Additional operations:
  - peek
  - isEmpty
  - isFull
- Implementations:
  - Array
  - Singly linked list



List as a Stack

- List methods:
  - `append(`*x*`)` - add *x* to the end of the list
  - `pop()` - remove & return the last item in the list
- Resource: https://docs.python.org/3/tutorial/datastructures.html#using-lists-as-stacks

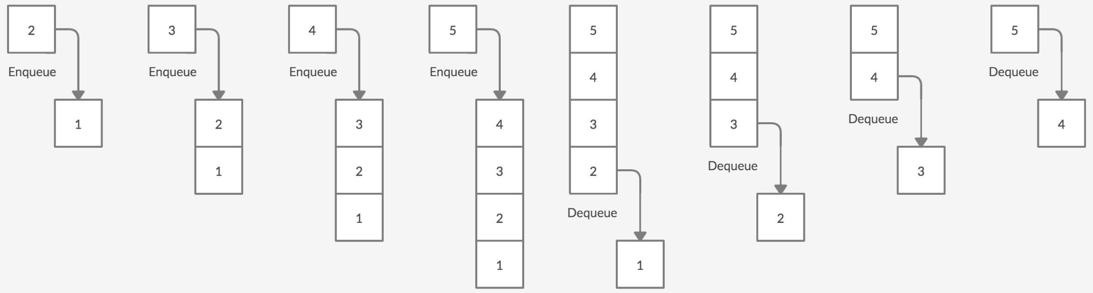
```
stack = []
print(stack) # []
stack.append('apple')
print(stack) # ['apple']
stack.append('banana')
print(stack) # ['apple', 'banana']
stack.append('cherry')
print(stack) # ['apple', 'banana', 'cherry']
stack.pop()
print(stack) # ['apple', 'banana']
stack.pop()
print(stack) # ['apple']
stack.pop()
print(stack) # []
```

Stack Class

```python
class Stack:
    def __init__(self):
        self.stack = []

    def push(self, item):
        pass

    def pop(self):
        pass

    def peek(self):
        pass

    def is_empty(self):
        pass

def main():
    stack = Stack()

if __name__ == '__main__':
    main()
```

Queue

- First in, first out (FIFO)
- Two primary operations:
  - enqueue
  - dequeue
- Additional operations:
  - peek
  - isEmpty
  - isFull
- Implementations:
  - Singly or doubly linked list
  - Double-ended queue



List as a Queue

- collections module
- Deque methods:
  - `append(x)` - add *x* to the right side of the deque
  - `popleft()`- remove & return an item from the left side of the deque
- Lists are not efficient for this purpose
- Resources:
  - https://docs.python.org/3/tutorial/datastructures.html#using-lists-as-queues
  - https://docs.python.org/3/library/collections.html#collections.deque

```python
from collections import deque
queue = deque([])
print(queue) # deque([])
queue.append('apple')
print(queue) # deque([‘apple’])
queue.append('banana')
print(queue) # deque([‘apple’, ‘banana’])
queue.append('cherry')
print(queue) # deque([‘apple’, ‘banana’, ‘cherry’])
queue.popleft()
print(queue) # deque([‘banana’, ‘cherry’])
queue.popleft()
print(queue) # deque([‘cherry’])
queue.popleft()
print(queue) # deque([])
```

Queue

```python
from collections import deque

class Queue:
    def __init__(self):
        self.queue = deque([])

    def enqueue(self, item):
        pass

    def dequeue(self):
        pass

    def peek(self):
        pass

    def is_empty(self):
        pass

def main():
    queue = Queue()

if __name__ == '__main__':
    main()
```

Programming Activity
(30 Minutes)

Programming Activity

- Checkout to master - `git checkout master`
- Create a new branch called 02-practical - `git checkout -b 02-practical`
- Copy 02-practical.ipynb from the course materials repository into your practicals repository
- Open up the Anaconda Prompt (it should be install on all lab computers) & `cd` to your practicals repository
- Run the following command: `jupyter notebook`

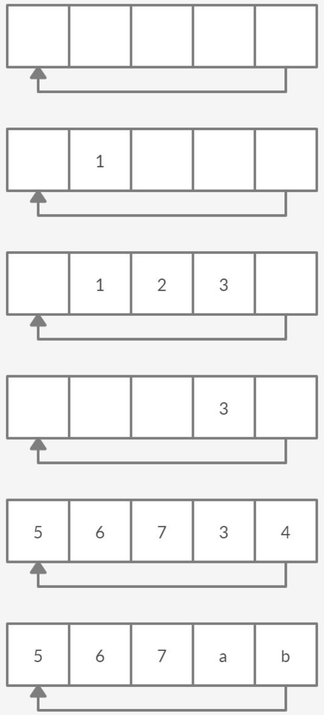
Programming Activity

- Please open 02-practical.ipynb
- Please **ONLY** answer questions 1-2
- We will go through the solutions after 30 minutes

Solutions

Circular Queue

- Alternative names:
  - Circular buffer
  - Cyclic buffer
  - Ring buffer
- How does it work?
- Implementation using four pointers:
  - Buffer start in memory
  - Buffer end in memory
  - Start of valid data, i.e., index or pointer
  - End of valid data, i.e., index or pointer
- You can prevent overwriting the data & return an error or raise an exception



1
1 2 3
3
5 6 7 3 4
5 6 7 a b