Python 2: More Abstract Data Types & Comprehensions

IN608: Intermediate Application Development Concepts

Kaiako: Tom Clark & Grayson Orr

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

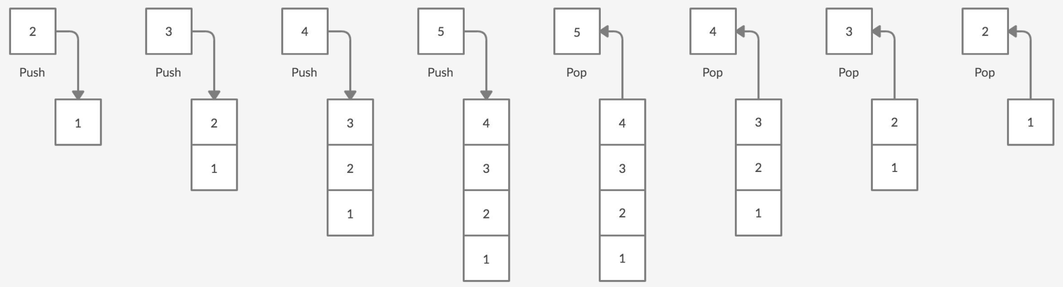
Today's Content

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

More Abstract Types

Stack

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

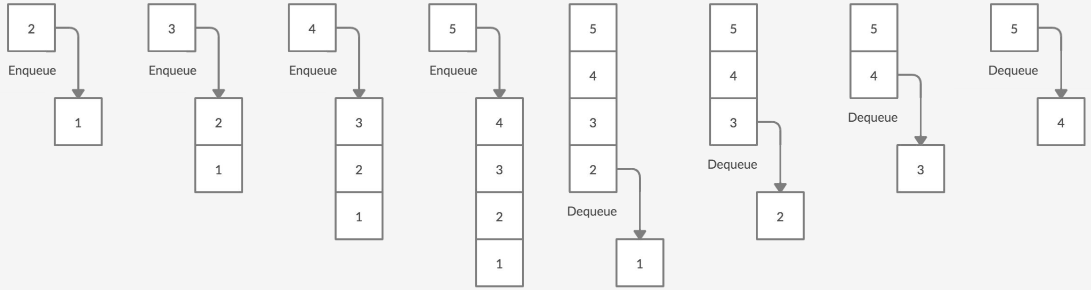
```python
stack = []
print(stack) # []
stack.append('apple')
print(stack) # ['apple']
stack.append('banana')
print(stack) # ['apple', 'banana']
stack.append('cherry')
print(stack) # ['apple', 'banana', 'cherry']
stack.pop()
print(stack) # ['apple', 'banana']
stack.pop()
print(stack) # ['apple']
stack.pop()
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:
  - Linked list
  - Double-ended queue



List as a Queue

- 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

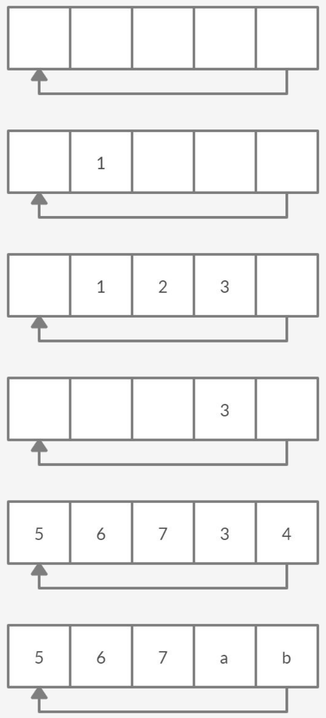
```python
from collections import deque
queue = deque([])
print(queue) # deque([])
queue.append('apple')
print(queue) # deque([‘apple’])
queue.append('banana')
print(queue) # deque([‘apple’, ‘banana’])
queue.append('cherry')
print(queue) # deque([‘apple’, ‘banana’, ‘cherry’])
queue.popleft()
print(queue)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

Queue

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

    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()
```

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



Comprehensions

Comprehensions

- Succinct way of creating a list, set or dictionary
- A comprehension consists of the following elements:
  - Expression (optional)
  - Variable
  - Input sequence
  - Predicate (optional)

```
[expression for variable input sequence predicate]
```

List Comprehension

- Resource: https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions

```
string = '123 Hi 456'
numbers = []
for s in string:
    if s.isdigit():
        numbers.append(int(s))
print(numbers) # [1, 2, 3, 4, 5, 6]

string = '123 Hi 456'
numbers = [int(s) for s in string if s.isdigit()]
print(numbers) # [1, 2, 3, 4, 5, 6]
```

Set Comprehension

```python
class Cat:
    def __init__(self, breed, is_active):
        self.breed = breed
        self.is_active = is_active

def main():
    cats = [
        Cat('Birman', True),
        Cat('Birman', True),
        Cat('Maine Coon', False),
        Cat('Persian', False),
        Cat('Ragdoll', False),
        Cat('Siamese', True)
    ]
    active_cats = {c.breed for c in cats if c.is_active}
    print(active_cats)

if __name__ == '__main__':
    main() # {‘Birman’, ‘Siamese’}
```

Dictionary Comprehension

```
fruit_price = {'apple': 0.89, 'banana': 0.75, 'orange': 0.60, 'pineapple': 3.50}
double_fruit_price = {}
for (k, v) in fruit_price.items():
    double_fruit_price[k] = v * 2
print(double_fruit_price) # {'apple': 1.78, 'banana': 1.5, 'orange': 1.2, 'pineapple': 7.0}

fruit_price = {'apple': 0.89, 'banana': 0.75, 'orange': 0.60, 'pineapple': 3.50}
double_fruit_price = {k: v * 2 for (k, v) in fruit_price.items()}
print(double_fruit_price) # {'apple': 1.78, 'banana': 1.5, 'orange': 1.2, 'pineapple': 7.0}
```