MONASH
INFORMATION
TECHNOLOGY

FIT9133 Semester 2 2019
Programming Foundations in Python

Week 7:
Classes and Abstract Data Types

Chunyang Chen

Module 3 Synopsis

- Module 3 is aimed to introduce you with:
  - Concepts of decomposition
    - Functions and methods
    - Modules in Python
  - Concepts of classes and methods
    - Implementation
    - Object instantiation
  - Abstract Data Types: Stack, Queue

Module 3 Learning Objectives

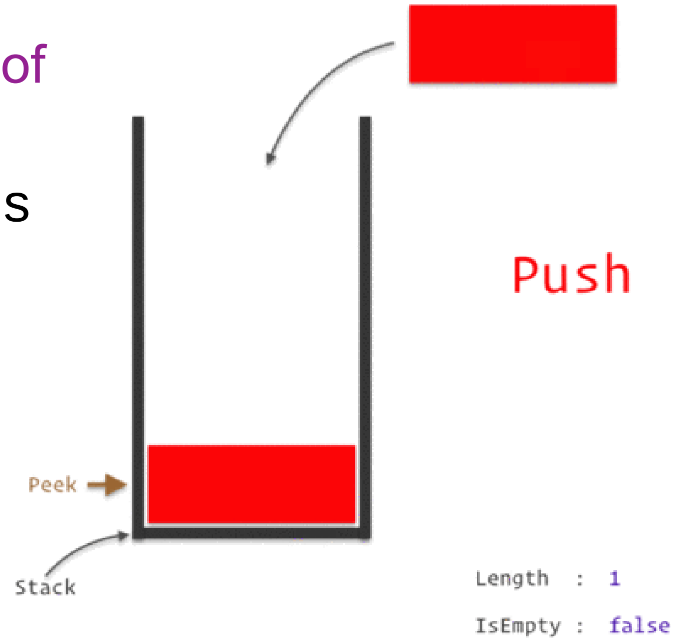
- Upon completing this module, you should be able to:
  - Identify how to decompose a computational program into manageable units of functions and/or classes
  - Recognize how data is represented and manipulated by three common Abstract Data Types (ADTs): Stack, and Queue
  - Define and implement your own abstract data type with the essential associated methods

MONASH University

Abstract Data Type: Stack

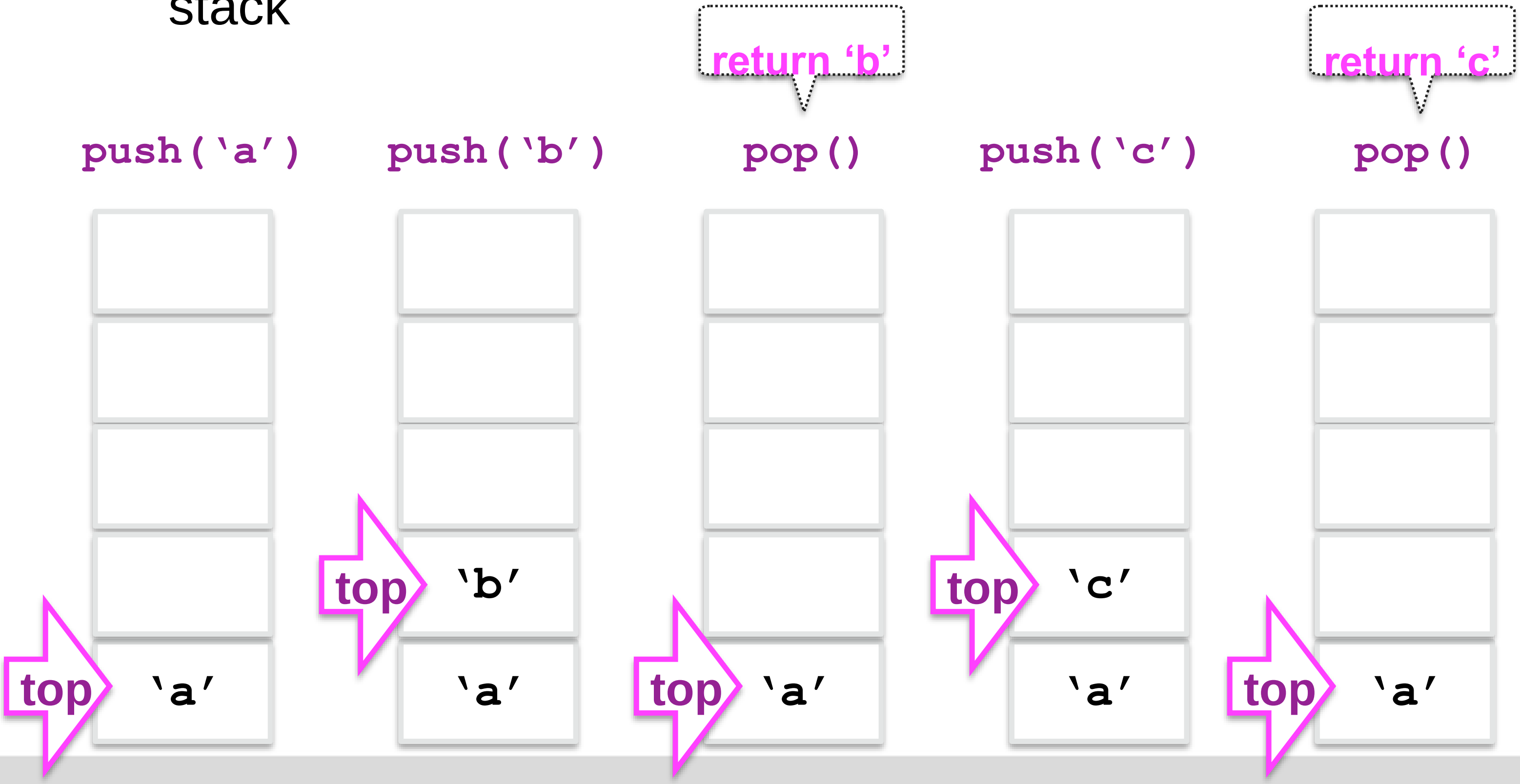
Stack

- Stack:
  - An ordered collection where data items are accessed based on **LIFO (Last-In-First-Out)**
  - Adding new items and removing existing items happened at the “top” of the stack
  - Useful for *reversing* the order of items within a collection



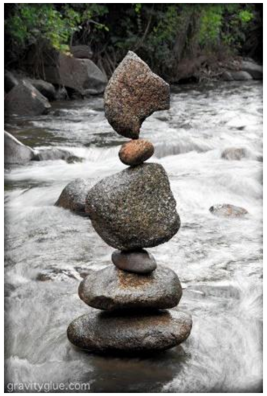
Stack

- Stack operations:
  - **push(item)**: add a new item onto the top of the stack
  - **pop()**: remove an existing item from the top of the stack
  - **peek()**: look at the top item of the stack; without modifying the stack



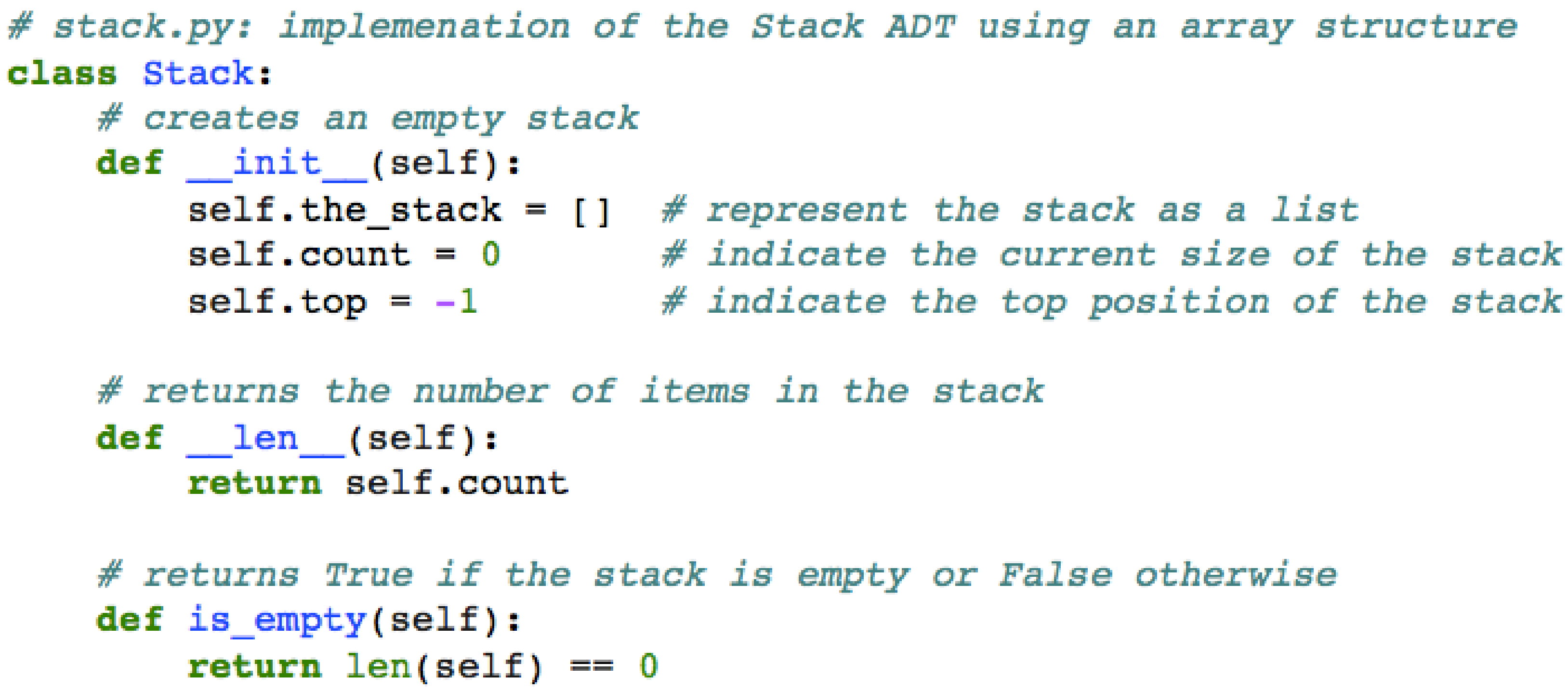
Practical questions solved by Stack

- Stack in real-world:



- Stack for practical questions:
  - Check for balanced parentheses, braces e.g., “(()())”
  - Reverse a string without using recursion
  - Undo changes in MS Word (addition/deletion of records)
  - …

Stack ADT: Implementation

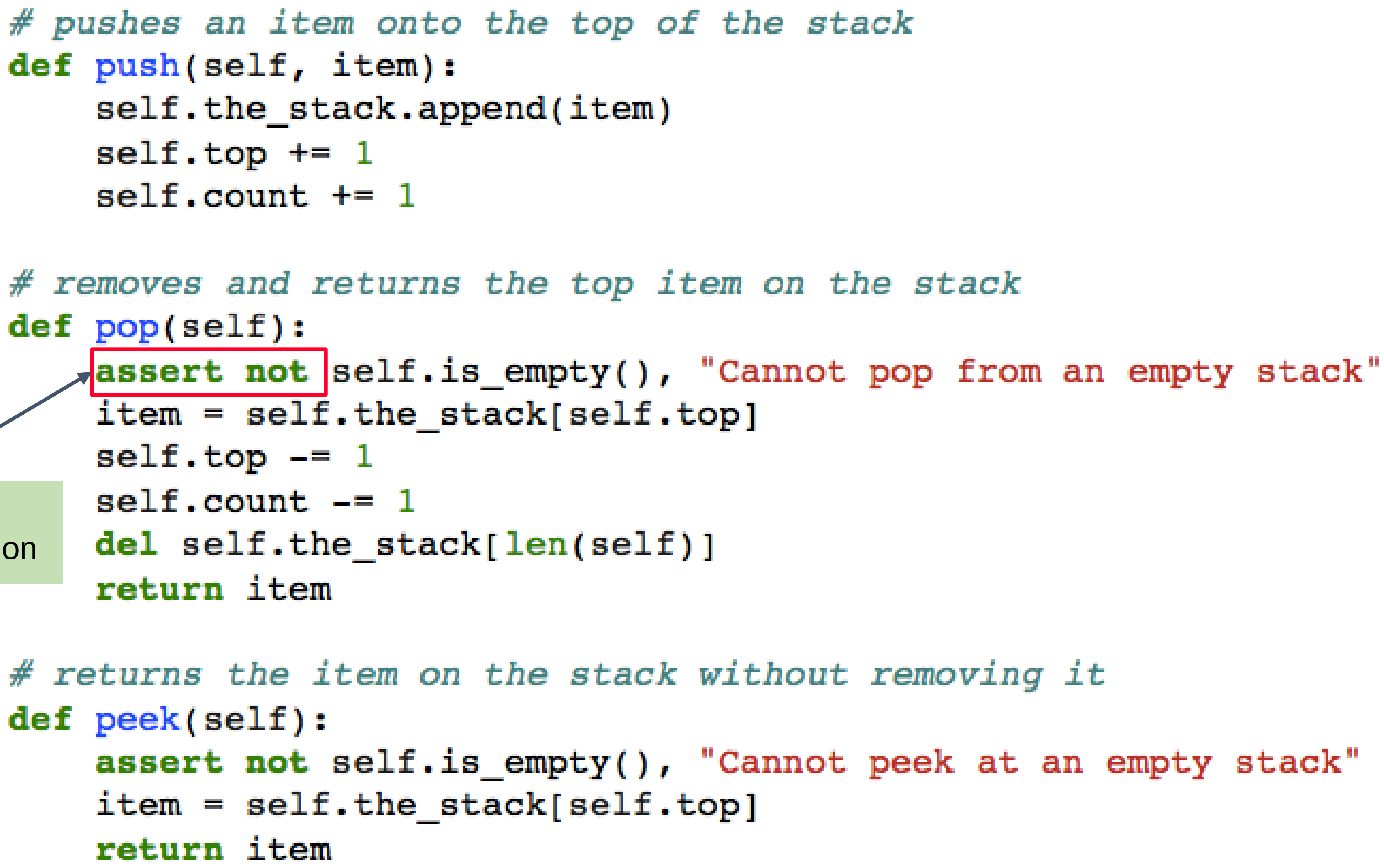


```
# stack.py: implemenation of the Stack ADT using an array structure
class Stack:
    # creates an empty stack
    def __init__(self):
        self.the_stack = []  # represent the stack as a list
        self.count = 0       # indicate the current size of the stack
        self.top = -1        # indicate the top position of the stack

    # returns the number of items in the stack
    def __len__(self):
        return self.count

    # returns True if the stack is empty or False otherwise
    def is_empty(self):
        return len(self) == 0
```

Stack ADT: Implementation



```
# pushes an item onto the top of the stack
def push(self, item):
    self.the_stack.append(item)
    self.top += 1
    self.count += 1

# removes and returns the top item on the stack
def pop(self):
    assert not self.is_empty(), "Cannot pop from an empty stack"
    item = self.the_stack[self.top]
    self.top -= 1
    self.count -= 1
    del self.the_stack[len(self)]
    return item

# returns the item on the stack without removing it
def peek(self):
    assert not self.is_empty(), "Cannot peek at an empty stack"
    item = self.the_stack[self.top]
    return item
```

Statement for testing assumption

Review Question 1 J49QZ1

What would be the output of the given program?

```
from stack import Stack

int_stack = Stack()
int_stack.push(1)
int_stack.push(2)
print(int_stack.pop(), end=“,”)
int_stack.push(3)
print(int_stack.pop(), end=“,”)
print(int_stack.pop())
```

A. 2, 1, 3

B. 1, 2, 3

C. 2, 3, 1

D. 3, 2, 1

Review Question 2

J49QZ1

What would be the output of the given program?

```
from stack import Stack

int_stack = Stack()
int_stack.push(1)
int_stack.push(2)
print(int_stack.pop(), end=“,”)
print(int_stack.pop(), end=“,”)
print(int_stack.pop())
int_stack.push(3)
```

A. 2, 1, 3

B. 1, 2,

C. 2, 1,

D. Some error occurred

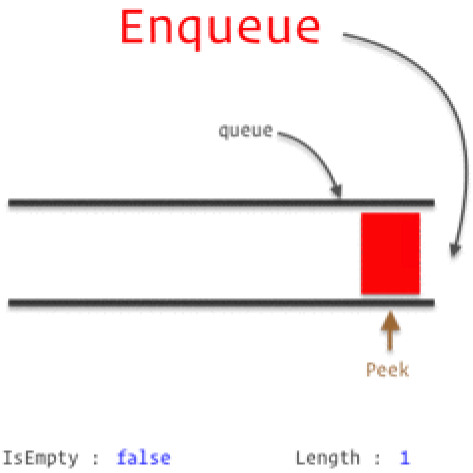
MONASH University

Abstract Data Type: Queue

Queue ADT

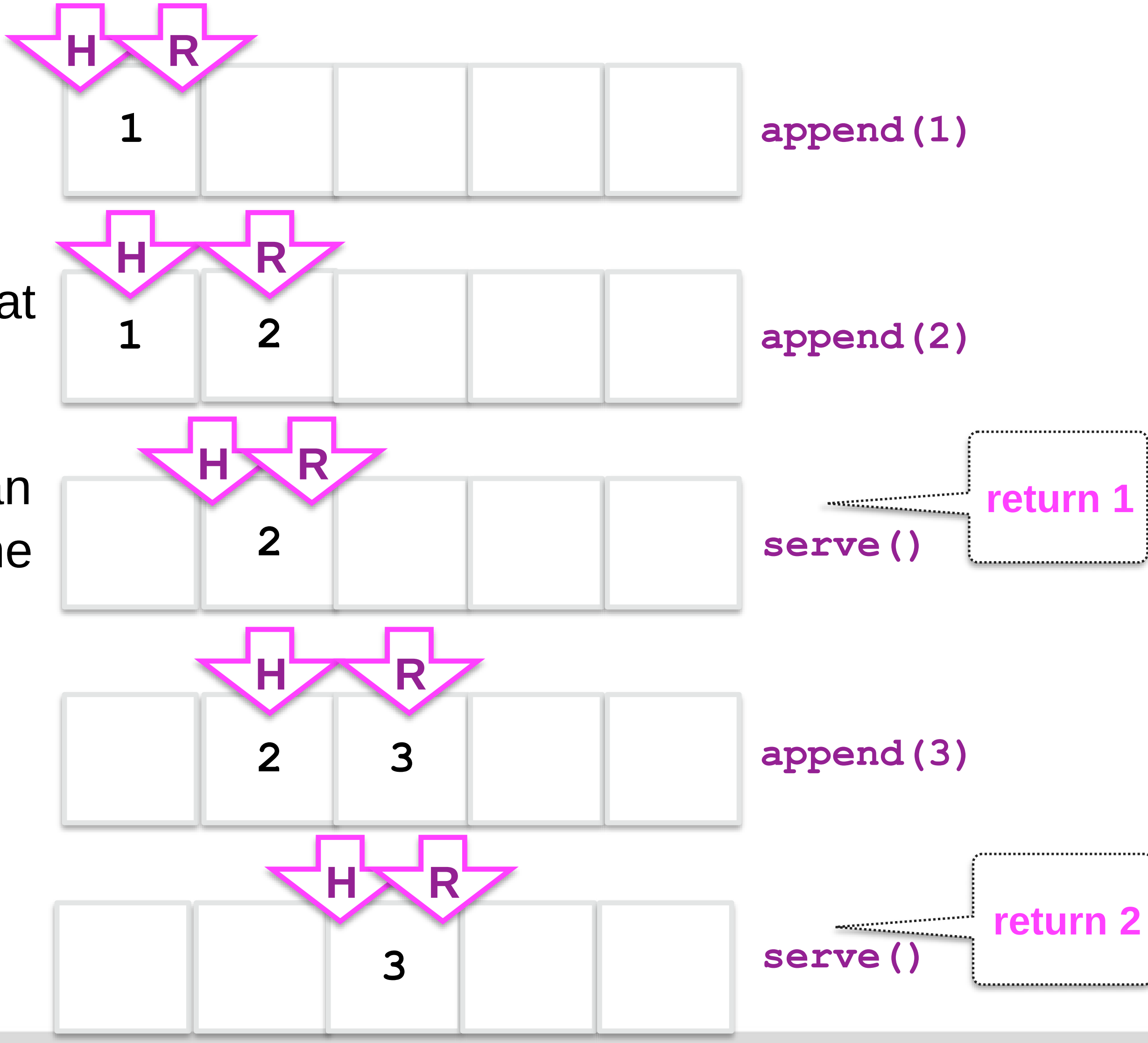
- Queue ADT:
  - An ordered collection where data items are accessed based on **FIFO (First-In-First-Out)**
  - Adding new items at the "rear" of the queue (i.e. enqueue)
  - Removing existing items at the "head" of the queue (i.e. dequeue)
  - Useful for demonstrating a *queuing* system (strictly no jumping the queue)

Queue of dogs for Toilet ...

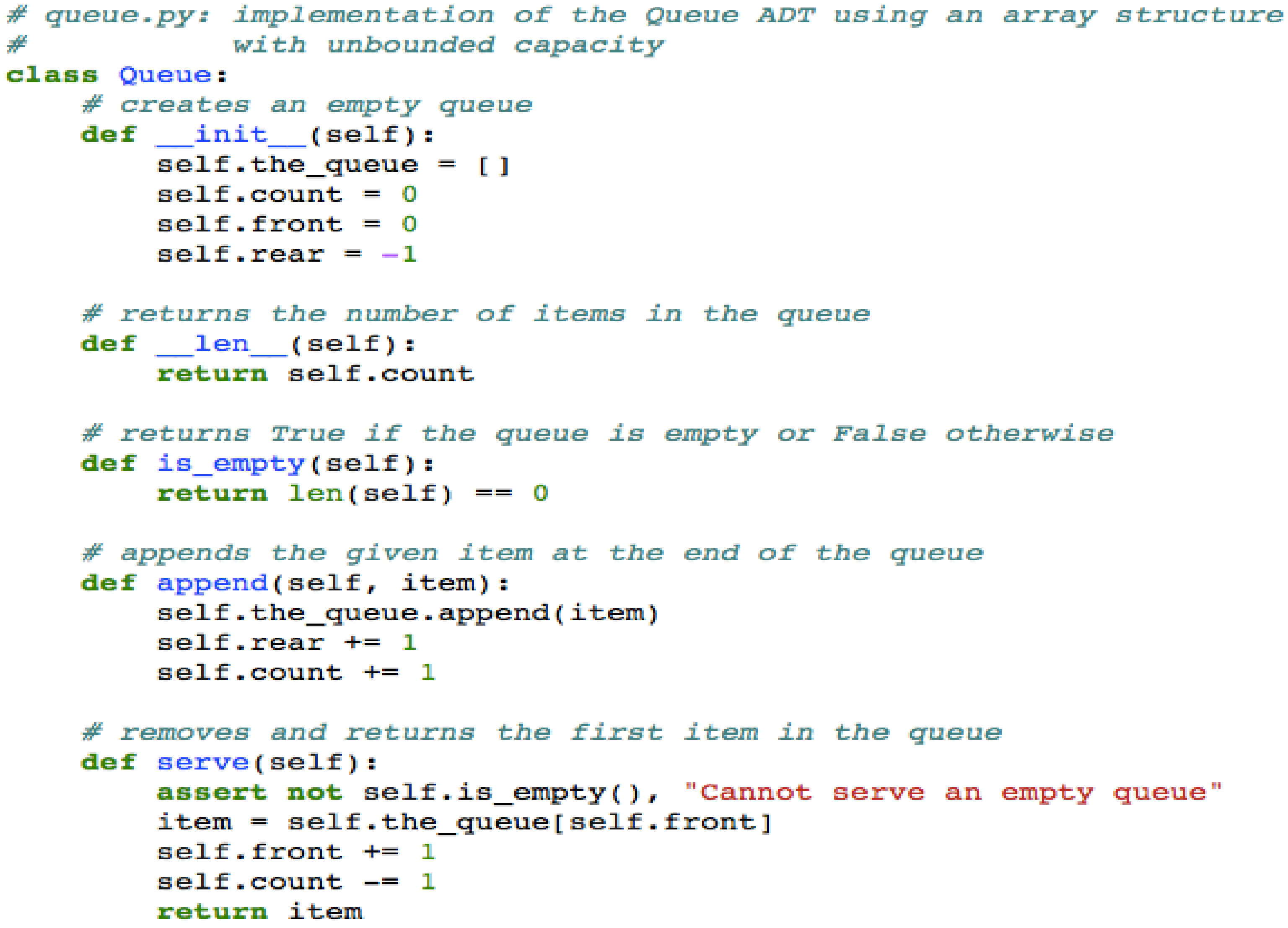


Queue ADT: Operations

- Queue operations:
  - **append(item)**: append a new item at the rear (end) of the queue
  - **serve()**: remove an existing item from the front of the queue



Queue ADT: Implementation



```python
# queue.py: implementation of the Queue ADT using an array structure
#           with unbounded capacity
class Queue:
    # creates an empty queue
    def __init__(self):
        self.the_queue = [ ]
        self.count = 0
        self.front = 0
        self.rear = -1

    # returns the number of items in the queue
    def __len__(self):
        return self.count

    # returns True if the queue is empty or False otherwise
    def is_empty(self):
        return len(self) == 0

    # appends the given item at the end of the queue
    def append(self, item):
        self.the_queue.append(item)
        self.rear += 1
        self.count += 1

    # removes and returns the first item in the queue
    def serve(self):
        assert not self.is_empty(), "Cannot serve an empty queue"
        item = self.the_queue[self.front]
        self.front += 1
        self.count -= 1
        return item
```

Review Question 3

J49QZ1

Assuming the queue is with the length of 3, and it is implemented as a *circular* structure. What would be the output of the given program?

```
from queue import Queue

int_queue = Queue()
int_queue.append(3)
int_queue.append(1)
int_queue.append(4)
print(int_queue.serve())
int_queue.append(2)
print(int_queue.serve())
```

A. 3, 2

B. 1, 3

C. 3, 1

D. Some error occurred

Recall First Three Modules

Summary

1. Programming concepts & basic grammar in Python
   - Variable, operator, control structure (selection, iteration)
   - Naming convention, indentation, comment/document

2. Data structure:
   - Atomic types: int, float, bool
   - Collective types: list, tuple, set, dictionary, stack, queue

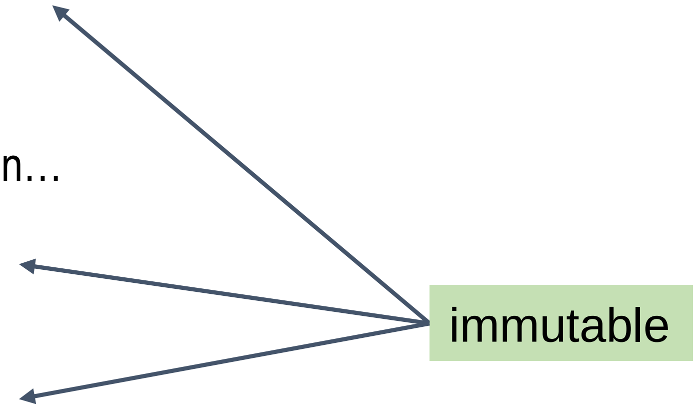
3. Program decomposition:
   - Function
   - OOP & class

Programming concepts & basic grammar in Python

- Variables (semantic name)
- Operator (arithmetic, relational, logical)
- Control structure
  - Selection: if-condition
  - Iteration: for-loop, while-loop,
    - Continue & break
- Indentation (space, tab)
- Comments (inline, block)
- Input/Output (I/0)
  - Standard I/O: input(), print()
  - File I/O: read, write

Data Structure

- String manipulation
  - “”, lower(), split(), strip(), find(),count(), len(), in …
- List
  - [], append(), pop(), index(), sort(), list comprehension…
- Tuple
- Set
  - Unique items
  - Operations: union, intersection, difference, symmetric difference
- Dictionary
  - Key-value pairs with fast retrieval
  - in, keys(), items()



immutable

Decomposition

- Decomposition
  - a process of breaking a complex problem into simpler, independent, manageable units
  - Reusable, clarity, maintainable
- Function
  - Definition: def, name, parameter, return, comments
  - Invoking/calling: parameters
- Object oriented programming
  - Everything in Python is an object
- Class
  - Implementation: data attributes, procedures → methods
  - Instance
  - Variable scoping

Week 7 Summary

- We have discussed:
  - Functions and Python modules
  - Classes and Methods
  - Abstract Data Types (Stack, Queue, List)

- Next week:
  - Midsemester test will be held on **16th September**

Reminder: Please come to the lab/consultation for assignment 1 **interview**.