DATA STRUCTURE

Dr. Rania Baashirah

r.baashirah@ubt.edu.sa

Mobile: +966-559009202

CHAPTER 3: Linear Data Structure

Introduction

- Linear structures are data collection whose items are ordered depending: how they are added, or removed.
- They can be thought of as having two ends.
  - “left” and “right” ||||||||| [||||||||||||||| =
  - “front” “rear"
  - "bottom" and "top"
- They are different from they way they are added and removed.

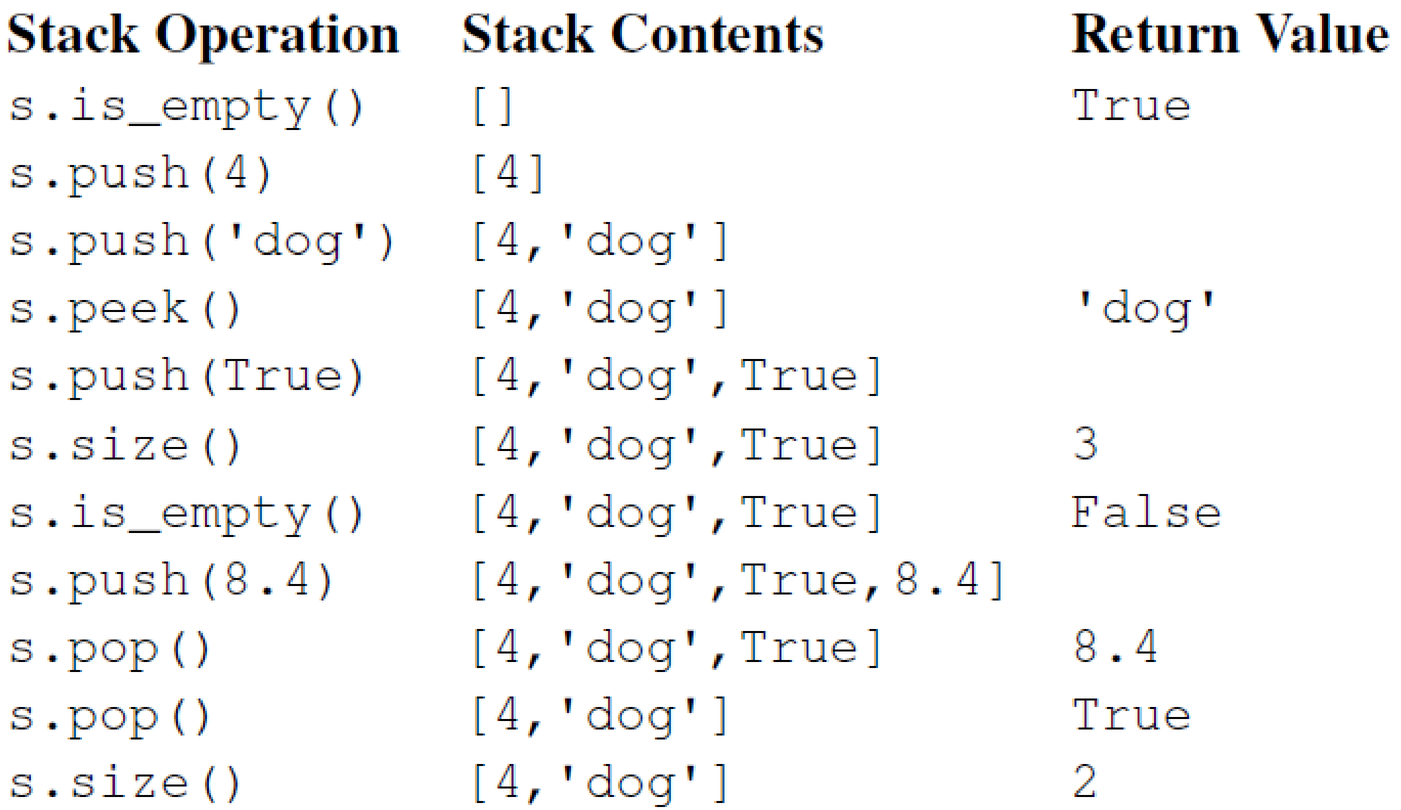
Stack

Stack of Books

Stack of Plates

Stack in python

- It is called push down stack
- In python stack is nothing but a list and can hold any data type.
- The only different that are not allowed to remove data randomly.
- Ex. Trays at café, books, browser



Stack Operation	Stack Contents	Return Value
`s.is_empty()`	`[]`	`True`
`s.push(4)`	`[4]`	
`s.push('dog')`	`[4,'dog']`	
`s.peek()`	`[4,'dog']`	`'dog'`
`s.push(True)`	`[4,'dog',True]`	
`s.size()`	`[4,'dog',True]`	`3`
`s.is_empty()`	`[4,'dog',True]`	`False`
`s.push(8.4)`	`[4,'dog',True,8.4]`	
`s.pop()`	`[4,'dog',True]`	`8.4`
`s.pop()`	`[4,'dog']`	`True`
`s.size()`	`[4,'dog']`	`2`

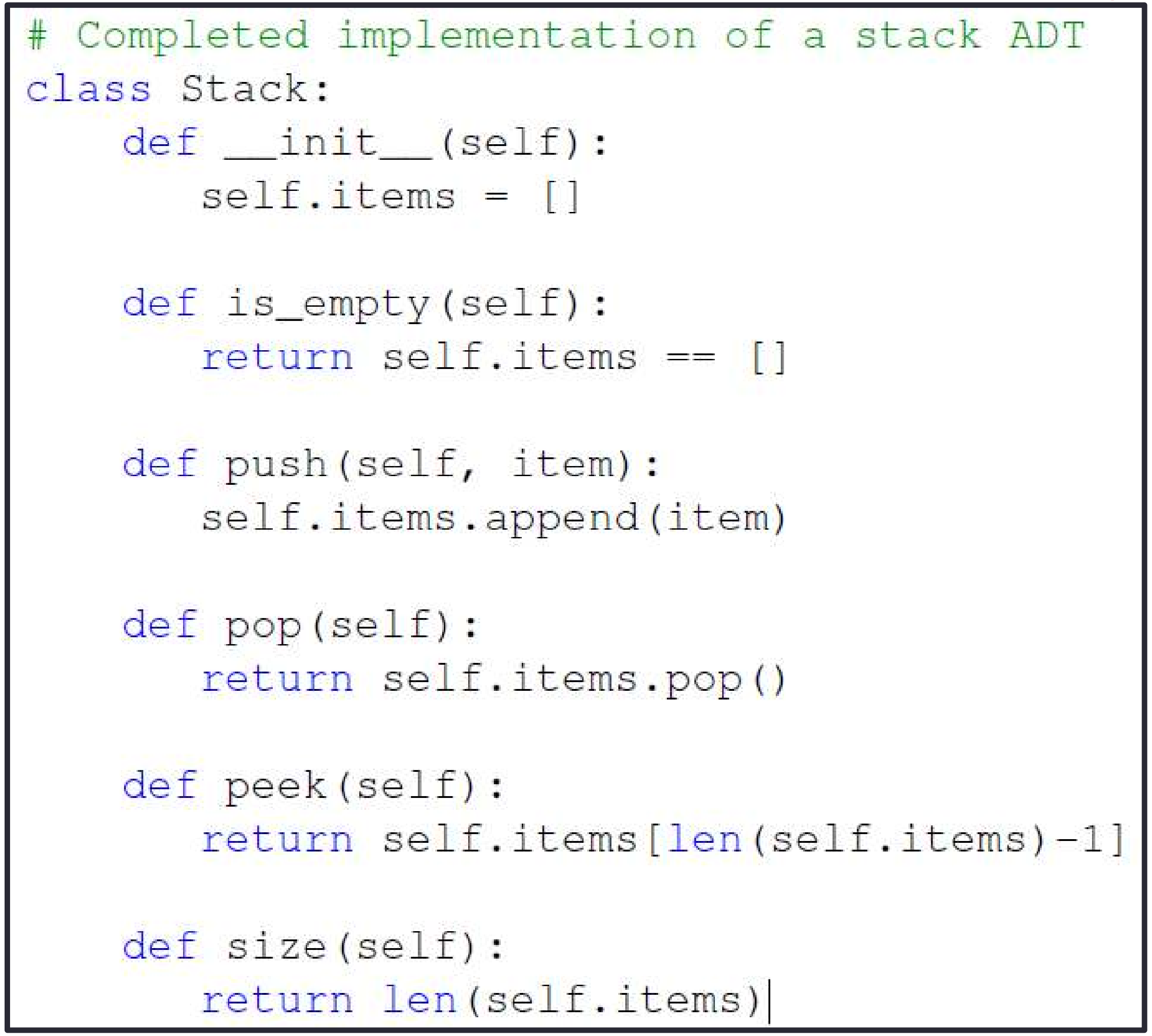
Concrete Data Structure and Abstract Data Structure (ADT)

- **Concrete Data Structures**
  - Arrays, records, and linked lists are concrete data types.
  - They are provided by the computer language.
  - They are stored at specific addresses in memory.
- **Abstract Data Structures/Types (ADTs)**
  - Offer a higher-level view of our interaction with data, and consists of:
    - Data.
    - Operations on this data.
  - They are defined by set of operations: what can be done, what result.

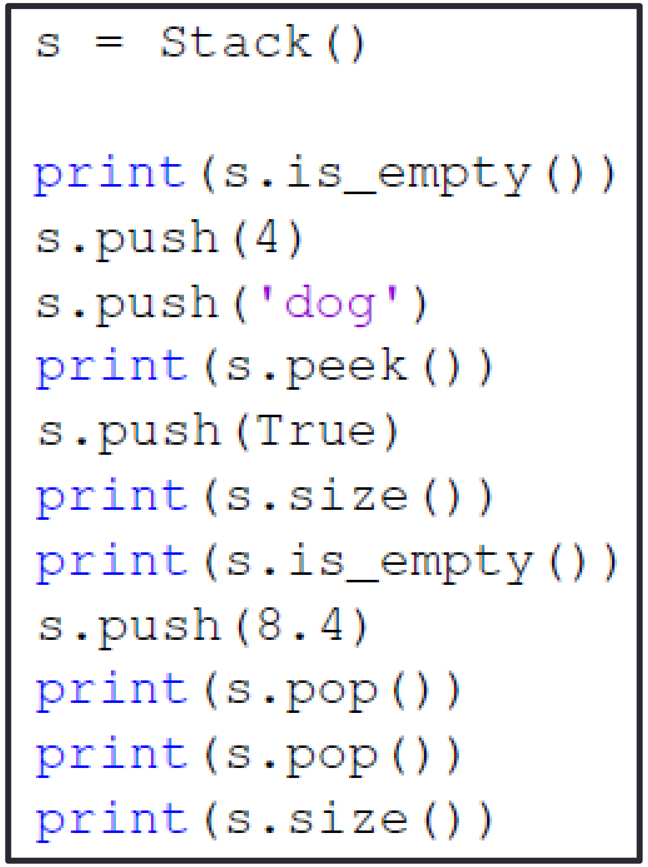
The Stack ADT

- A stack is a list where a new item is added at the "top" and the removed item is taken also from the "top".
- Stacks use **L**ast-**I**n **F**irst-**O**ut (**LIFO**) Algorithm.
- A stack **S** has associated the following operations:
  - Push(x) – Add x to top of stack **S**.
  - Pop() – Remove top item from stack S ***and return*** it.
  - Peek() – Return top of element of stack **S** ***without removing*** it.
  - IsEmpty() – Return true if stack **S** is empty.
  - IsFull() – Return true if stack **S** is full.
  - size() – Return the size of stack **S**.

Stack Implementation in Python



```python
# Completed implementation of a stack ADT
class Stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)

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

    def peek(self):
        return self.items[len(self.items)-1]

    def size(self):
        return len(self.items)
```

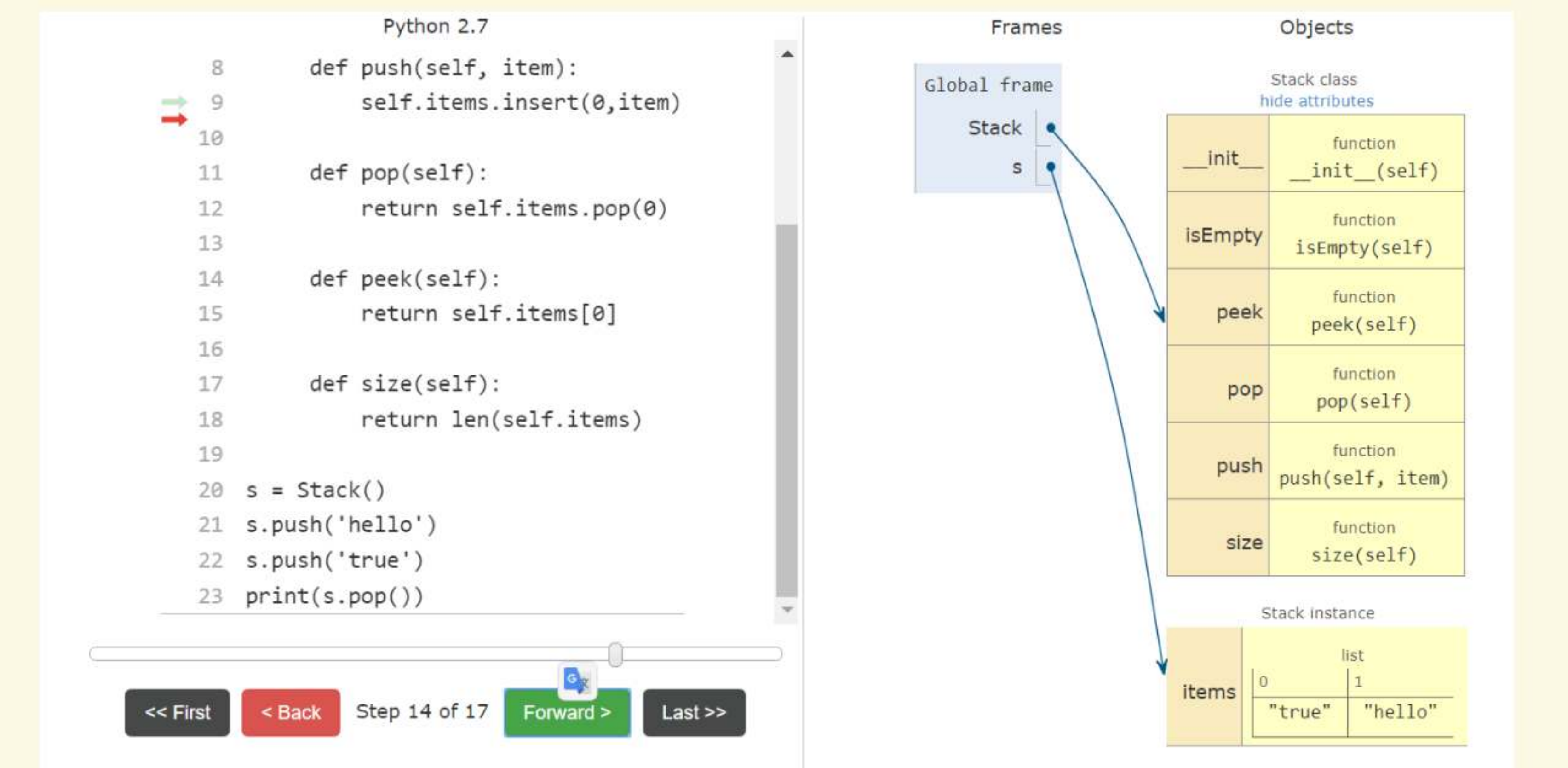


```python
s = Stack()

print(s.is_empty())
s.push(4)
s.push('dog')
print(s.peek())
s.push(True)
print(s.size())
print(s.is_empty())
s.push(8.4)
print(s.pop())
print(s.pop())
print(s.size())
```

Question: Is it a problem to make the stack inserting and removing the first element of the list ?

Implementing both Algorithms in python

- http://interactivepython.org/courselib/static/pythonds/BasicDS/ImplementingaStackinPython.html
- Every source code in this book is available online with a nice Demo.



```
    def push(self, item):
        self.items.insert(0,item)

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

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

    def size(self):
        return len(self.items)

s = Stack()
s.push('hello')
s.push('true')
print(s.pop())
```

Global frame

Stack	●
s	●

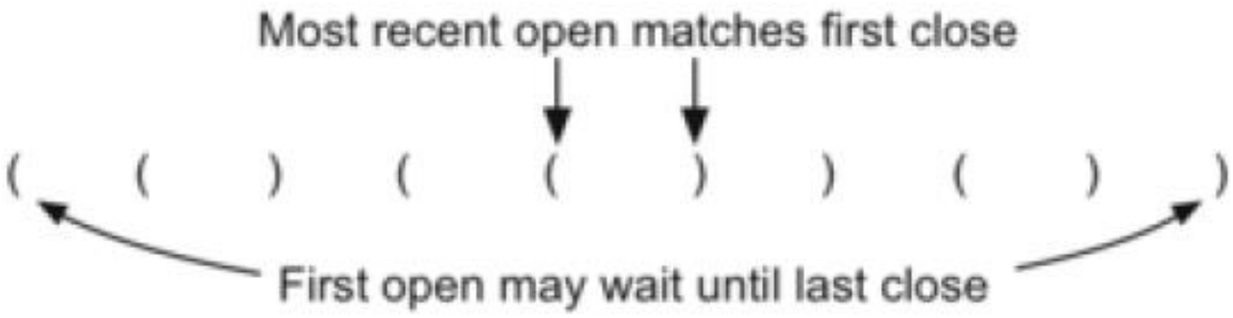
Stack class
hide attributes

__init__	function __init__(self)
isEmpty	function isEmpty(self)
peek	function peek(self)
pop	function pop(self)
push	function push(self, item)
size	function size(self)

Stack instance

	list
items	0 "true" | 1 "hello"

Balance Parentheses Problem


Most recent open matches first close
(( ) (( )) ( ))
First open may wait until last close

(( ) ( ) ( ) ( ))

(((( ))))

(( ) ((( )) ( )))

- In systematic way how can you decide wither a given string of parentheses is balanced or not ?!
- Answer: using stack ☺
    - Move from left to right; whenever it is "(" then push whenever it is ")" then pop; at the end *you should end up with an empty stack*.
        - Algorithm is given next slide.

Balance Parentheses Problem

Try Online

```
Function isBalance(e : Expression of parentheses)
   let s = new stack
   let balanced = True
       while (e has more symbols and balanced)
          symbol = take next symbol of e
          if symbol is "("
              s.push("(")
          else
              if s.isEmpty()
                 balanced = False
              else
                 s.pop()

   if balanced and s.isEmpty()
       return True
   else
       return False
```

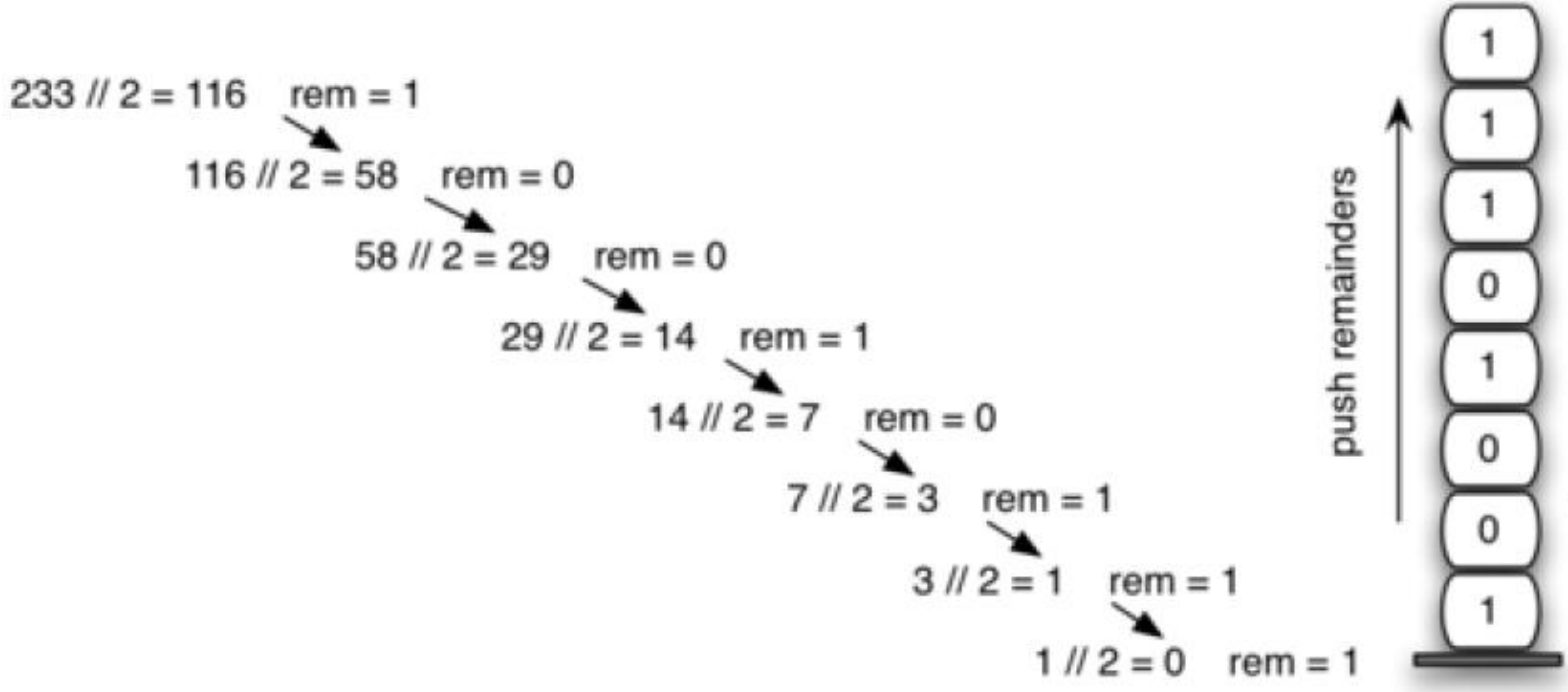
(( ) ( ) ( ) ( ))

(((( ))))

(( ) ((( )) ( )))

Converting Decimal To Binary

Try Online



233 // 2 = 116 rem = 1

116 // 2 = 58 rem = 0

58 // 2 = 29 rem = 0

29 // 2 = 14 rem = 1

14 // 2 = 7 rem = 0

7 // 2 = 3 rem = 1

3 // 2 = 1 rem = 1

1 // 2 = 0 rem = 1

Complexity

- Append()
- Pop()
- Have constant complexity of O(1)

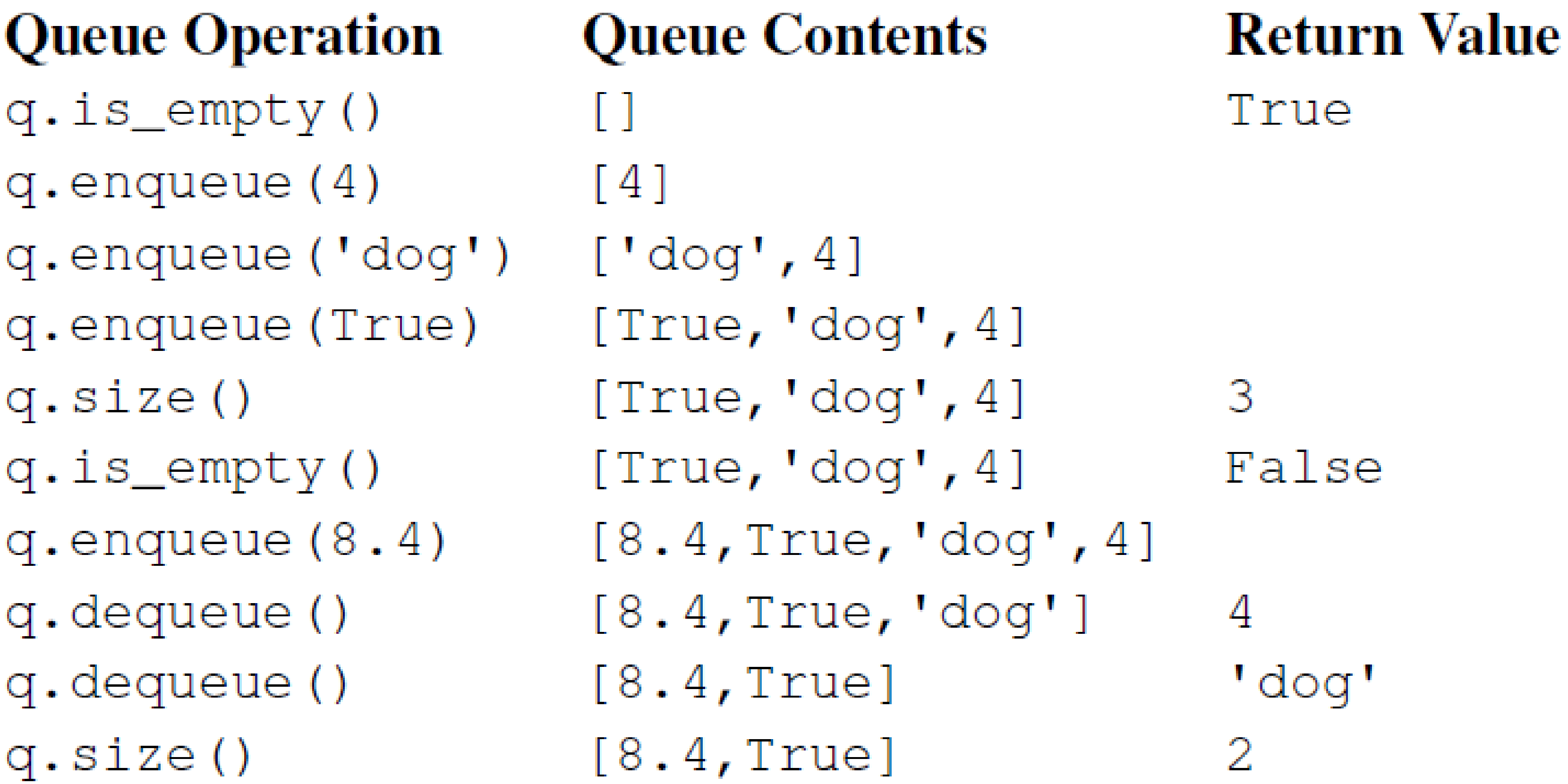
- Append(0)
- Pop(0)
- Require O(n) since we specify a position

Queue

- Queue is nothing but a linear data structure with an extra special rule, which is "***First Come First Serve***"

Queue Operations Example

- Like Stack in python, Queue is also list and can hold any data type.



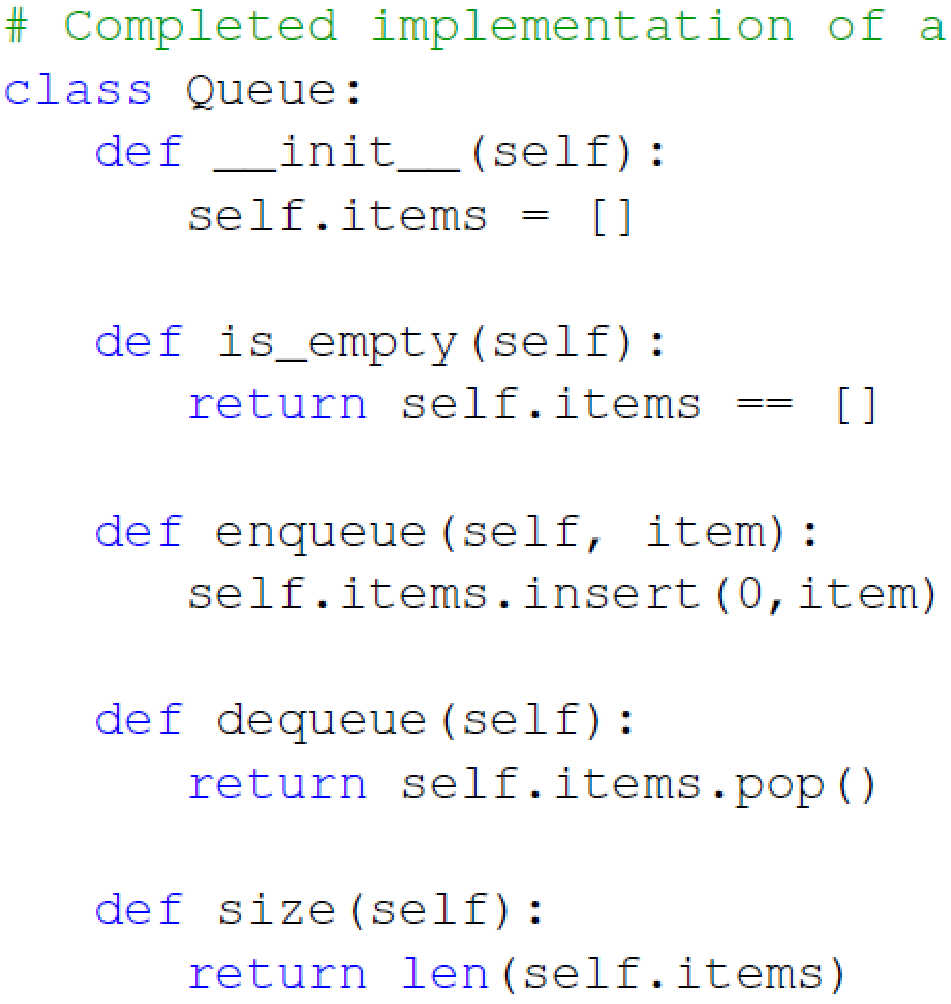
Queue Operation	Queue Contents	Return Value
`q.is_empty()`	`[]`	`True`
`q.enqueue(4)`	`[4]`	
`q.enqueue('dog')`	`['dog',4]`	
`q.enqueue(True)`	`[True,'dog',4]`	
`q.size()`	`[True,'dog',4]`	`3`
`q.is_empty()`	`[True,'dog',4]`	`False`
`q.enqueue(8.4)`	`[8.4,True,'dog',4]`	
`q.dequeue()`	`[8.4,True,'dog']`	`4`
`q.dequeue()`	`[8.4,True]`	`'dog'`
`q.size()`	`[8.4,True]`	`2`

The Queue ADT

- Adding the new item at the "top" and the removed item is taken from the front.
- A queue is a **F**irst-**I**n **F**irst-**O**ut (**FIFO**) abstract data type.
- A queue **Q** has associated the following operations:
    - EnQueue(x) – Add x to back of queue Q.
    - DeQueue() – Remove front of queue Q, and return it.
    - Peek() – Return front of queue Q, without removing it.
    - IsEmpty() – Return true if queue Q is empty.
    - IsFull() – Return true if queue Q is full.
    - size() – Return the size of queue Q.

Queue Implementation in python

Try Online

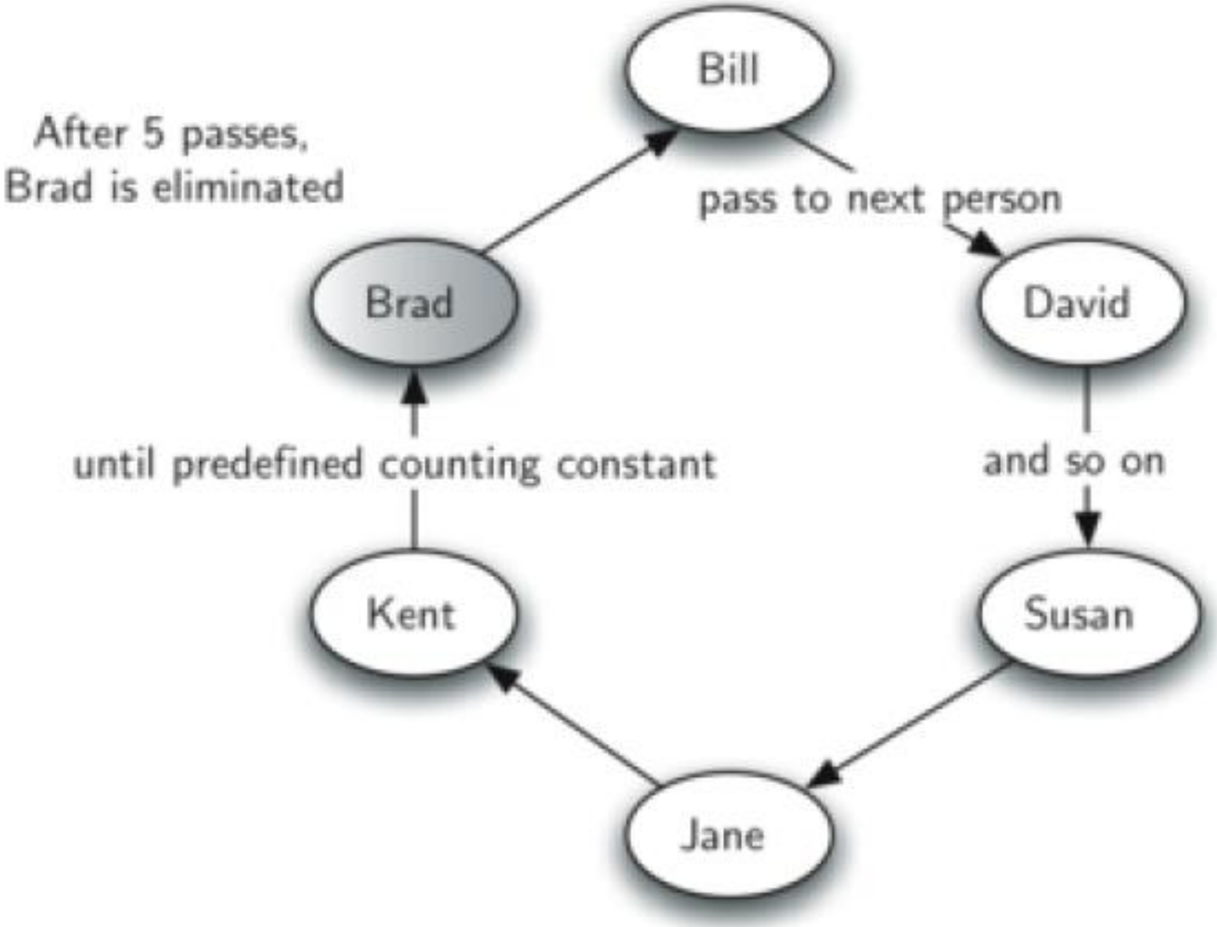


```python
# Completed implementation of a queue ADT
class Queue:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

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

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

    def size(self):
        return len(self.items)
```

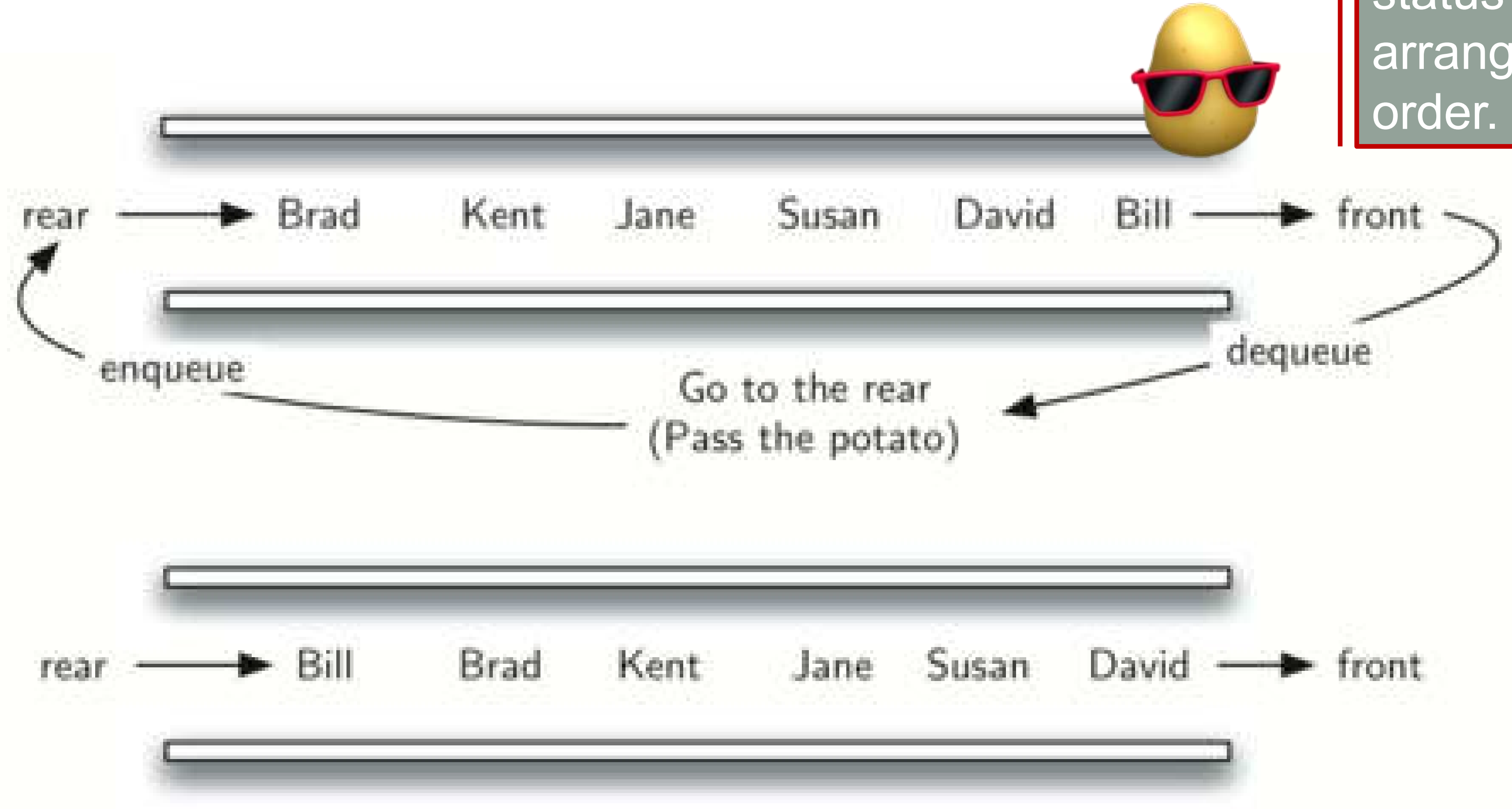
Simulation: Hot Potato

- Hot potato plate starts from Bill and passes to his next kid.
- After number of passes n, the kid with holding plate is out!
- And then continue.
- Last kid left is the winner.

```
Bill, David, Susan, Jane, Kent, Brad
```



How algorithm works



Notice that the initial status makes kids to arranged in reversed order.

Python Simulation of Hot Potatoe

Try Online

```
from queue1 import Queue

def hotPotato(namelist, n):
    q = Queue()
    for name in namelist:
        q.enqueue(name)

    while q.size() > 1:
        for i in range(n):
            q.enqueue(q.dequeue())

        q.dequeue()

    return q.dequeue()
# -------- Test ------------
winner = hotPotato(["Bill","David","Susan","Jane","Kent","Brad"], 5)
print(winner)
```

Complexity

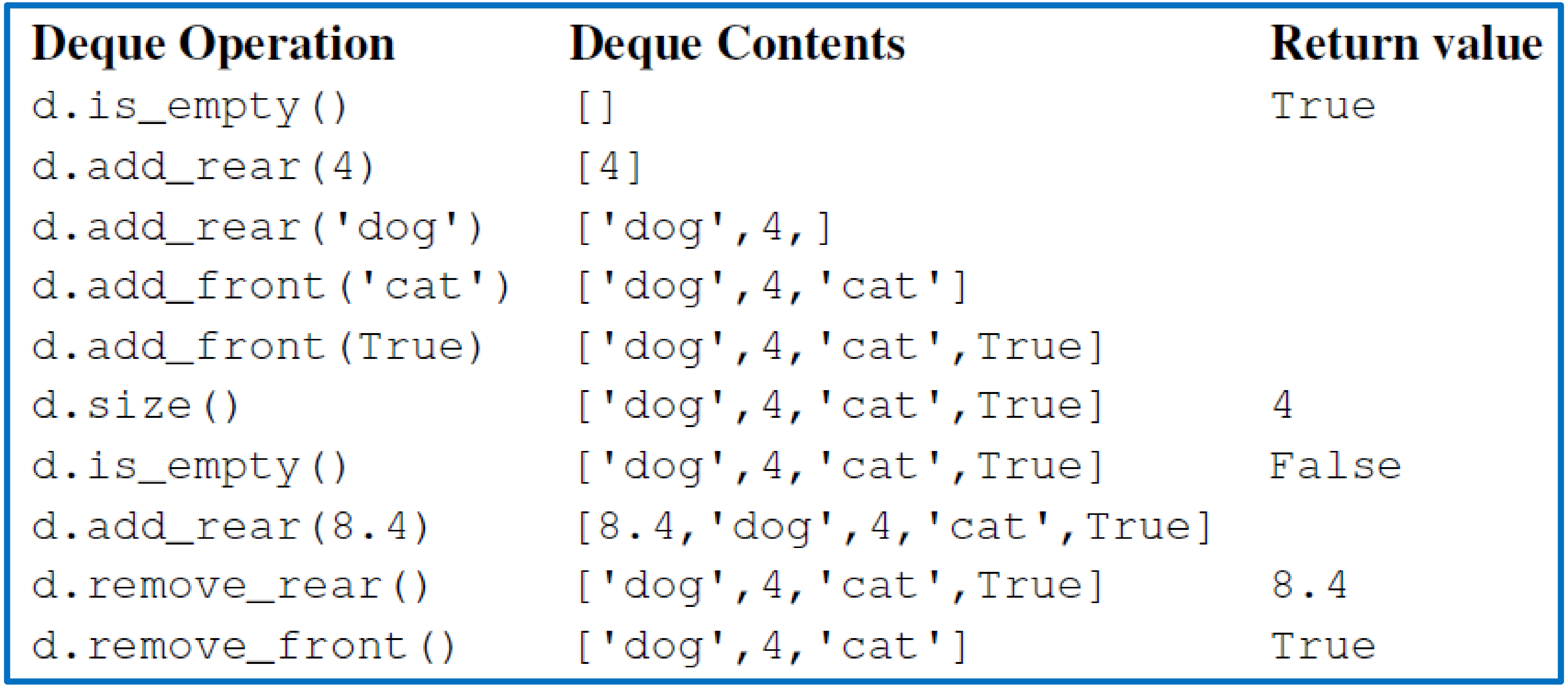
Try Online

- For a normal queue as a list
- queue(x) = O(n)
- dequeue() = O(1)

- For a circular implementation
- queue(x) = O(1)
- dequeue() = O(1)

DEques (Double Ended Queue)

- **DEque** is a queue with flexibility of add and remove from both sides.



Deque Operation	Deque Contents	Return value
`d.is_empty()`	`[]`	`True`
`d.add_rear(4)`	`[4]`	
`d.add_rear('dog')`	`['dog',4,]`	
`d.add_front('cat')`	`['dog',4,'cat']`	
`d.add_front(True)`	`['dog',4,'cat',True]`	
`d.size()`	`['dog',4,'cat',True]`	`4`
`d.is_empty()`	`['dog',4,'cat',True]`	`False`
`d.add_rear(8.4)`	`[8.4,'dog',4,'cat',True]`	
`d.remove_rear()`	`['dog',4,'cat',True]`	`8.4`
`d.remove_front()`	`['dog',4,'cat']`	`True`

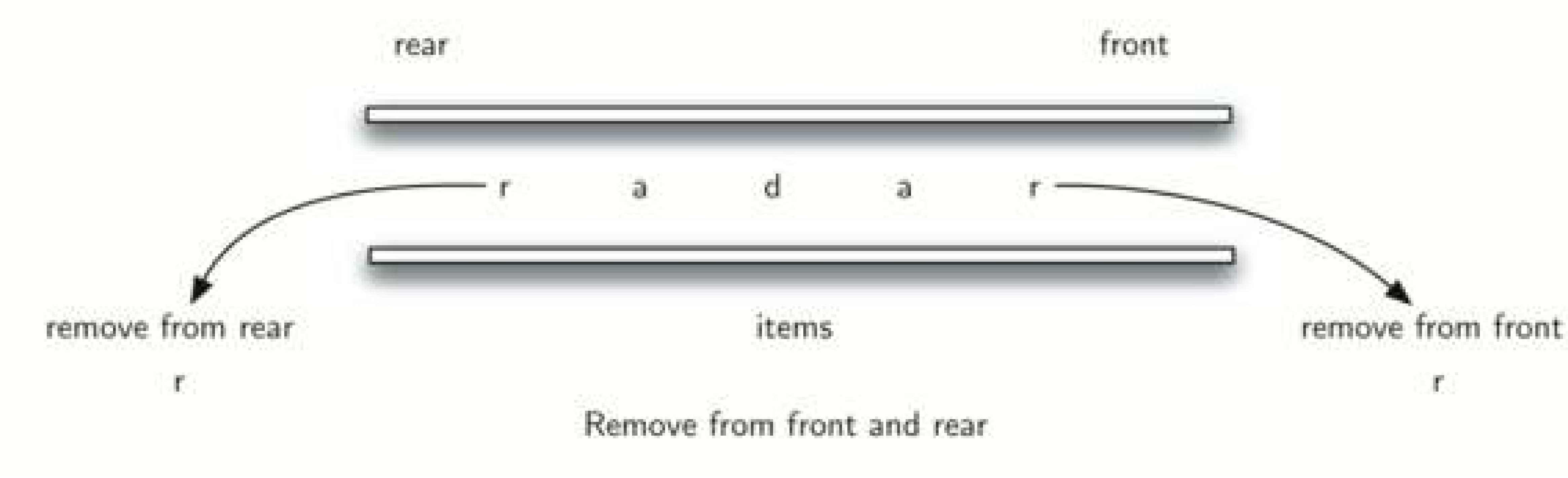
The DEques ADT

Try Online

- A queue **Q** has associated the following operations:
  - **add_front(x)** adds x to the front.
  - **add_rear(x)** adds x to the rear.
  - **remove_front()** removes the front item ***and returns it***.
  - **remove_rear()** removes the rear item ***and returns it***.
  - **is_empty()** return Ture if empty; False owtherwise.
  - **size()** returns the number of items.

Palindrome Checker

- Palindrome word is the English word than can be read from both sides such as 'madam'; 'radar'; 'toot'; and so on.



Remove from front and rear

Palindrome Checker Complexity

O(n)