codecademy

Stacks

Stack *overflow*

Every stack has a size that determines how many nodes it can accommodate. Attempting to push a node in a full stack will result in a stack overflow. The program may crash due to a stack overflow.
A stack is illustrated in the given image. `stackA.push(xg)` will result in a stack overflow since the stack is already full.

The *stack* data structure

A *stack* is a data structure that follows a last in, first out (LIFO) protocol. The latest node added to a stack is the node which is eligible to be removed first. If three nodes (`a` , `b` and, `c`) are added to a stack in this exact same order, the node `c` must be removed first. The only way to remove or return the value of the node `a` is by removing the nodes `c` and `b` .

Main methods of a *stack* data structure

The stack data structure has three main methods: `push()` , `pop()` and `peek()` . The `push()` method adds a node to the top of the stack. The `pop()` method removes a node from the top of the stack. The `peek()` method returns the value of the top node without removing it from the stack.

Stack data structure

A `Stack` is a data structure that supports two basic operations: pushing a new item to the top of the stack and popping a single item from the top of the stack.
In order to implement a stack using a node class, we have to store a node that is currently referencing the top of the stack and update it during the push and pop operations.

codecademy

```
from node import Node

class Stack:
  def __init__(self, limit=1000):
    self.top_item = None
    self.size = 0
    self.limit = limit

  def push(self, value):
    if self.has_space():
      item = Node(value)
      item.set_next_node(self.top_item)
      self.top_item = item
      self.size += 1
    else:
      print("All out of space!")

  def pop(self):
    if self.size > 0:
      item_to_remove = self.top_item
      self.top_item
= item_to_remove.get_next_node()
      self.size -= 1
      return item_to_remove.get_value()
    else:
      print("This stack is totally
empty.")

  def peek(self):
    if self.size > 0:
      return self.top_item.get_value()
    else:
      print("Nothing to see here!")

  def has_space(self):
    return self.limit > self.size

  def is_empty(self):
    return self.size == 0
```