G11-Term2-Test3-Solutions

January 20, 2020

1 Computer Science G11

1.1 Term 2 Test 3

Date Mon Jan 20 2020

1.2 NAME:

2 1 (KtiCa) (40%) Implement a stack class. Make sure to satisfy the following requirements and definitions.

In computer science, a stack is an abstract data type (for the sake of this problem, think of this as a class) that serves as a collection of elements, with two principal operations: `push`, which adds an element to the collection, and. `pop`, which removes the most recently added element that was not yet removed.

The interface of your code should be as follows

```
s = Stack() #creates an empty stack
s.pop()     # returns None
s.push('hello')
s.push(3)
s.len()     # returns 2, i.e, the number of items in the stack
s.pop()     # returns 3 removing it from the stack
s.len()     # returns 1
s.push([5,7,9,'foo'])
s.len()     # retutns 5
s.pop()     # returns 'foo'
s.pop()     # returns 9
```

3 2 (KtiCa) (40%) Implement a *joining* of stacks, that is, an operator that takes two stacks and returns a new stack containing the union of the both stacks' elements.

The interface of your code should be as follows

```
A = Stack([0,2,4,6])
B = Stack([1,3,5,7])
C = A*B
C.len()   # 8
C.pop()   # 7
C.pop()   # 5
```

4 3 (KtiCA) (20%) We may say that two stacks are equal when they store the same elements. Implement the comparision of stacks.

The interface of your code should satisfy

```
s = Stack()
A = Stack([0,'hello',s]  )
B = Stack()
B.push(0)
B.push('hello')
B.push(s)
A == B      # True
B.pop()
s.push(3)
B.push(s)
A == B      # False
```

5 Solutions

[105]:
```
#Notice the following
print( type([]))
print( type( list() ) )
print( type([]) == list)

a=[ 1,2,3]
b =list(a[:-1])
x=a.pop()
x,a,b,a+b
```

```
<class 'list'>
<class 'list'>
True
```

[105]: (3, [1, 2], [1, 2], [1, 2, 1, 2])

[133]:
```
import copy

class Stack:
    def __init__(self,l=None):
        self.s =  []
```

```
        if type(l) == list:
            #this would be the most general solutions which works no matter the␣
↪elements in l
            #self.s = copy.deepcopy(l) #In a single statement, without a loop.␣
↪Otherwise, iterate over l
            # The following satisfies the requirements but wouldn't work if l␣
↪itself contains a list
            for v in l:
                if type(v) == Stack:
                    self.s.append(Stack(v.s))
                else:
                    self.s.append(v)
        elif not l == None: # Careful: In python None is an object. Hence␣
↪list([None]) = [None]. NOT what we want!
            self.s = list([l])
    def push(self,x):
        # x=copy.deepcopy(x) #to make it work in the general case, no matter␣
↪what object we're pushing. See bottom.
        self.s.append(x)
    def pop(self):
        #return self.s.pop()  #this is the shortest way. Alternatively, use␣
↪these three steps
        x = self.s[-1]
        self.s = list( self.s[:-1])
        return x
    def len(self):
        return len(self.s)  #if unsure whether this would work, you can count␣
↪manually iterating over l as follows
        #l = 0
        #for v in self.s:
        #    l +=1
        return l
    def __mul__(self,B):
        return Stack(self.s+B.s)
    def __eq__(self,B):
        if isinstance(B,self.__class__):
            l = self.len()
            if B.len() != l : return False
            for i in range(l):
                if self.s[i] != B.s[i]:
                    return False
            return True
        return False
    def __str__(self):
        msg ='<'
        for e in self.s:
```

```
            msg += str(e)+' '
        return msg+'>'
```

[134]:
```
s = Stack()
A = Stack([0,'hello',s] )
B = Stack()
print('lengths A:',A.len(),A)

B.push(0)
B.push('hello')
B.push(s)
print('lengths B:',B.len(),B)

print('A',A,'B',B,'A==B',A == B)      # True
B.pop()
s.push(3)
B.push(s)
print('A',A,'B',B,'A==B',A == B)      # False
```

```
lengths A: 3 <0 hello <> >
lengths B: 3 <0 hello <> >
A <0 hello <> > B <0 hello <> > A==B True
A <0 hello <> > B <0 hello <3 > > A==B False
```

[135]:
```
t=[]
C = Stack([0,'hello',t])
D= Stack([0,'hello'])
t.append(666)
D.push(t)
print('C',C,'D',D,'C==D',C==D)  # Give True, but we would want it to be FAlse
# The problem is that `t` in C is not a copy of list `t` but a _refference_ to␣
 ↪it.
# Hence, when modifying t (as in t.append(666)) C also notices that change.
# In order for this to work not matter the elements we create a stack with, we␣
 ↪need to use
# a deepcopy in the constructor and the push
#
# Example to show that push also needs a deepcopy mechanism
r=[55]
D.push(r)
print('D before changing r',D)
r.append(-1)
print('D After',D)
```

```
C <0 hello [666] > D <0 hello [666] > C==D True
D before changing r <0 hello [666] [55] >
D After <0 hello [666] [55, -1] >
```

[ ]: