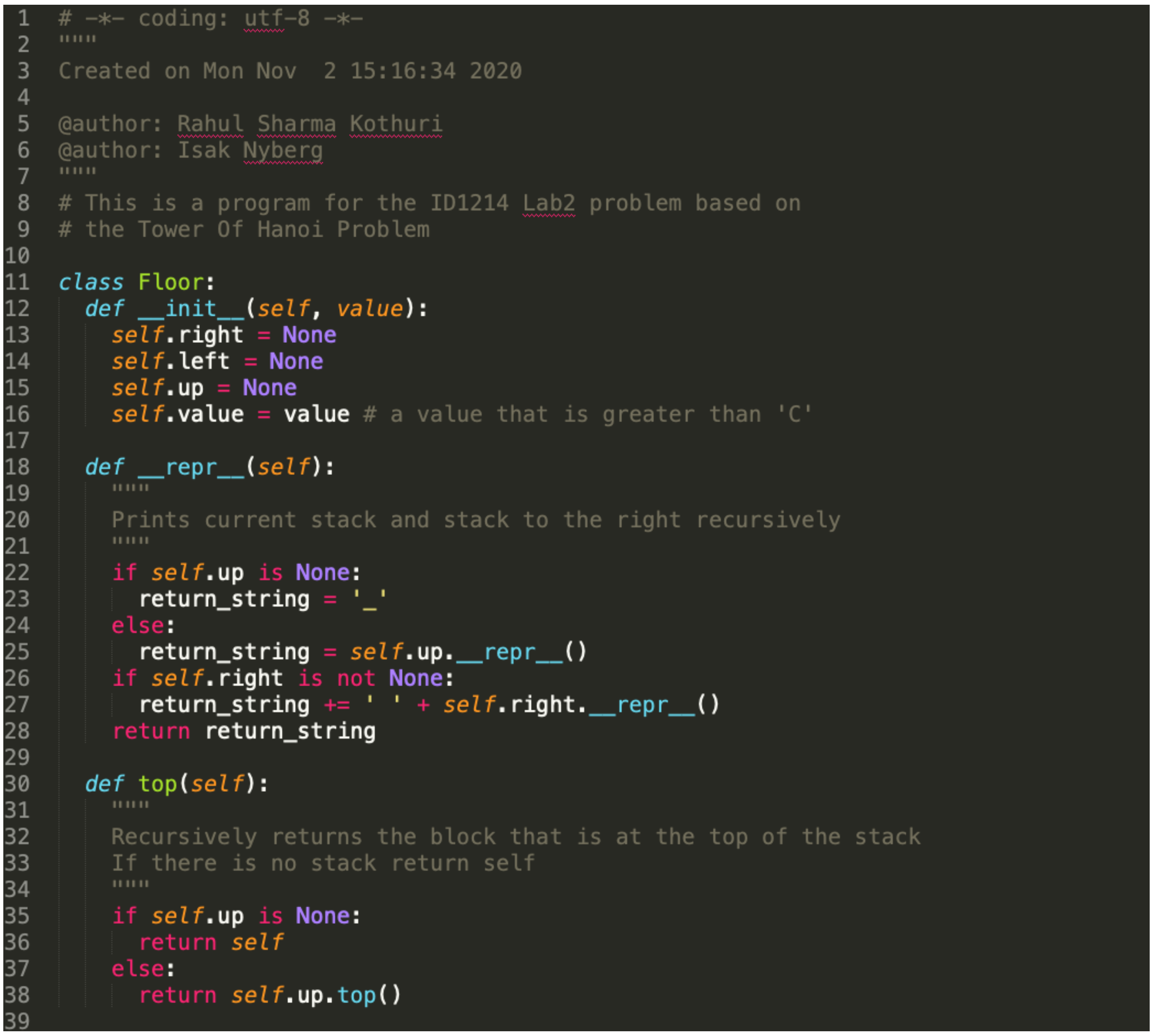
Isak Nyberg
Rahul Sharma Kothuri

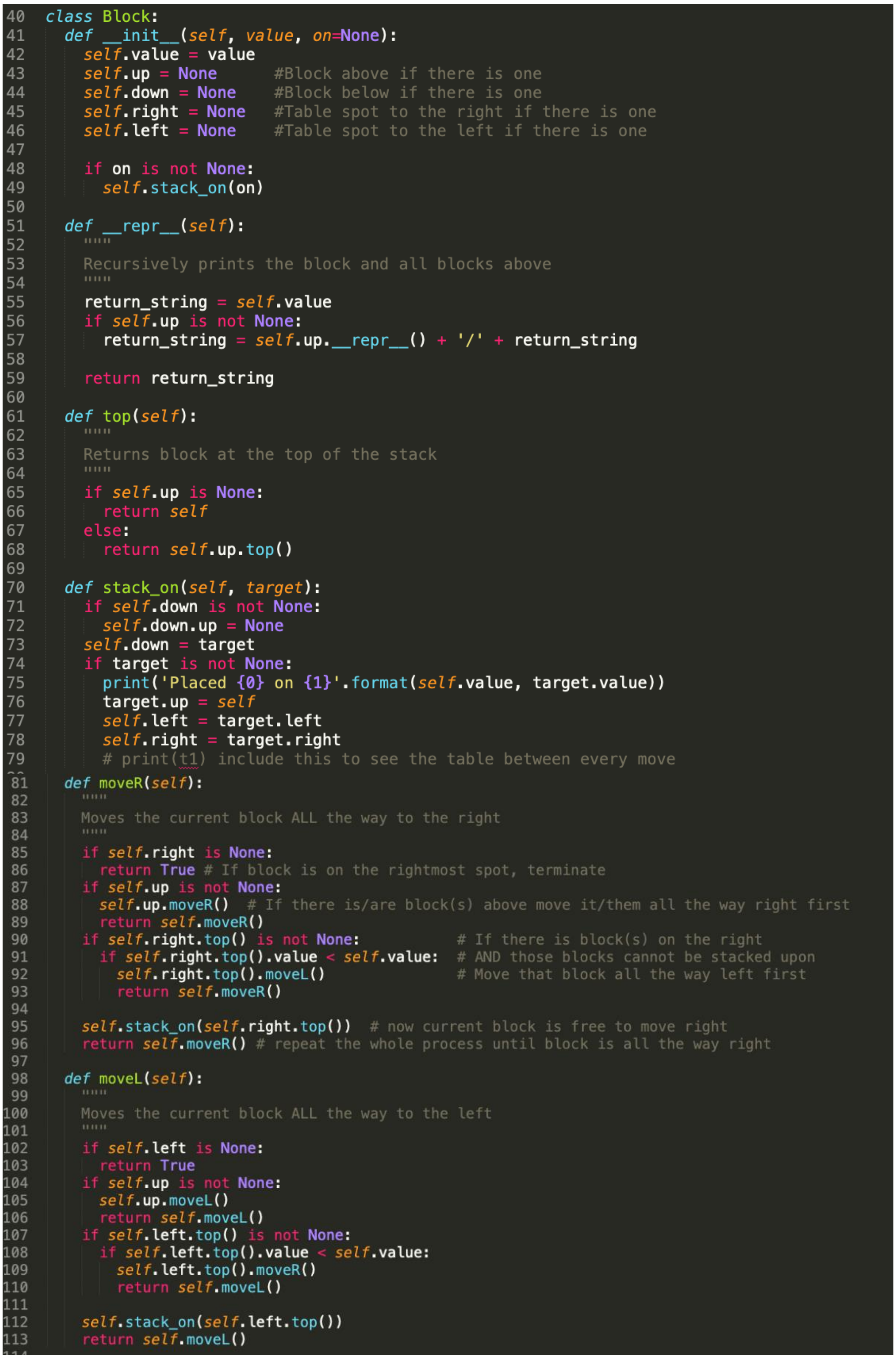
Laboration 2

Code



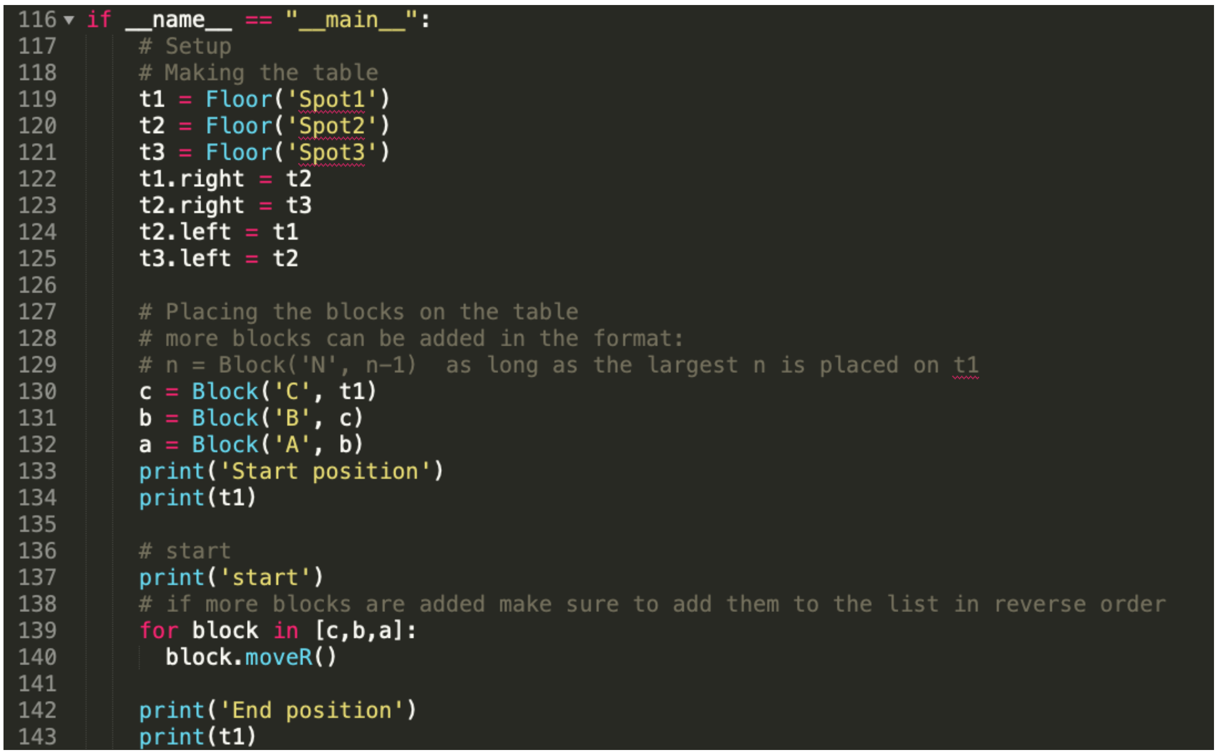
```python
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  2 15:16:34 2020

@author: Rahul Sharma Kothuri
@author: Isak Nyberg
"""
# This is a program for the ID1214 Lab2 problem based on
# the Tower Of Hanoi Problem

class Floor:
  def __init__(self, value):
    self.right = None
    self.left = None
    self.up = None
    self.value = value # a value that is greater than 'C'

  def __repr__(self):
    """
    Prints current stack and stack to the right recursively
    """
    if self.up is None:
      return_string = '_'
    else:
      return_string = self.up.__repr__()
    if self.right is not None:
      return_string += ' ' + self.right.__repr__()
    return return_string

  def top(self):
    """
    Recursively returns the block that is at the top of the stack
    If there is no stack return self
    """
    if self.up is None:
      return self
    else:
      return self.up.top()

```

Isak Nyberg
Rahul Sharma Kothuri



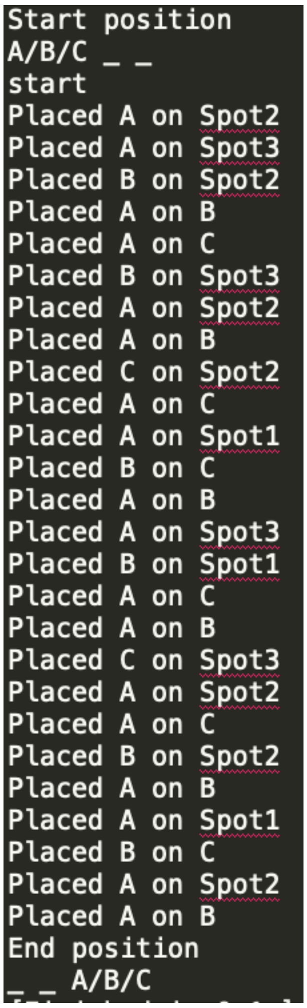
```python
class Block:
    def __init__(self, value, on=None):
        self.value = value
        self.up = None      #Block above if there is one
        self.down = None    #Block below if there is one
        self.right = None   #Table spot to the right if there is one
        self.left = None    #Table spot to the left if there is one

        if on is not None:
            self.stack_on(on)

    def __repr__(self):
        """
        Recursively prints the block and all blocks above
        """
        return_string = self.value
        if self.up is not None:
            return_string = self.up.__repr__() + '/' + return_string

        return return_string

    def top(self):
        """
        Returns block at the top of the stack
        """
        if self.up is None:
            return self
        else:
            return self.up.top()

    def stack_on(self, target):
        if self.down is not None:
            self.down.up = None
        self.down = target
        if target is not None:
            print('Placed {0} on {1}'.format(self.value, target.value))
            target.up = self
            self.left = target.left
            self.right = target.right
            # print(t1) include this to see the table between every move
    def moveR(self):
        """
        Moves the current block ALL the way to the right
        """
        if self.right is None:
            return True # If block is on the rightmost spot, terminate
        if self.up is not None:
            self.up.moveR()  # If there is/are block(s) above move it/them all the way right first
            return self.moveR()
        if self.right.top() is not None:            # If there is block(s) on the right
            if self.right.top().value < self.value:  # AND those blocks cannot be stacked upon
                self.right.top().moveL()            # Move that block all the way left first
                return self.moveR()

        self.stack_on(self.right.top())  # now current block is free to move right
        return self.moveR() # repeat the whole process until block is all the way right

    def moveL(self):
        """
        Moves the current block ALL the way to the left
        """
        if self.left is None:
            return True
        if self.up is not None:
            self.up.moveL()
            return self.moveL()
        if self.left.top() is not None:
            if self.left.top().value < self.value:
                self.left.top().moveR()
                return self.moveL()

        self.stack_on(self.left.top())
        return self.moveL()
```

Isak Nyberg
Rahul Sharma Kothuri



```python
if __name__ == "__main__":
    # Setup
    # Making the table
    t1 = Floor('Spot1')
    t2 = Floor('Spot2')
    t3 = Floor('Spot3')
    t1.right = t2
    t2.right = t3
    t2.left = t1
    t3.left = t2

    # Placing the blocks on the table
    # more blocks can be added in the format:
    # n = Block('N', n-1)  as long as the largest n is placed on t1
    c = Block('C', t1)
    b = Block('B', c)
    a = Block('A', b)
    print('Start position')
    print(t1)

    # start
    print('start')
    # if more blocks are added make sure to add them to the list in reverse order
    for block in [c,b,a]:
        block.moveR()

    print('End position')
    print(t1)
```

Result:



```
Start position
A/B/C _ _
start
Placed A on Spot2
Placed A on Spot3
Placed B on Spot2
Placed A on B
Placed A on C
Placed B on Spot3
Placed A on Spot2
Placed A on B
Placed C on Spot2
Placed A on C
Placed A on Spot1
Placed B on C
Placed A on B
Placed A on Spot3
Placed B on Spot1
Placed A on C
Placed A on B
Placed C on Spot3
Placed A on Spot2
Placed A on C
Placed B on Spot2
Placed A on B
Placed A on Spot1
Placed B on C
Placed A on Spot2
Placed A on B
End position
_ _ A/B/C
```

Isak Nyberg
Rahul Sharma Kothuri

Problem space graph

Blue is block A
Green is block B
Red is block C