```
#Problem 1
from inspect import stack
print('')
print('Problem 1')
stack1 = []

def push(stack, num):
  stack.append(num)

def display(stack):
  print(stack)


push(stack1, 1)
push(stack1, 2)
push(stack1, 3)
push(stack1, 4)
display(stack1)

#Problem 2
print('')
print('Problem 2')

stack2 = [5,10,15]

def pop(stack):
  lastNum = stack[-1]
  stack.pop(-1)
  print(f'Top element popped: {lastNum}, Updated Stack: {stack}')

pop(stack2)

#Problem 3
print('')
print('Problem 3')

stack3 = [3,6,9]
def peek(stack):
  print(f'Top element is: {stack3[-1]}')

peek(stack3)

#Problem 4
print('')
```

```
print('Problem 4')

stack4 = []
def is_empty(stack):
  if(len(stack) == 0):
    print('Stack is empty.')
  else:
    print('Stack is not empty.')

is_empty(stack4)

#Problem 5
print('')
print('Problem 5')

stack5 = [7, 14, 21]
def stack_size(stack):
  print(len(stack))
stack_size(stack5)

#Problem 6
print('')
print('Problem 6')

stack6 = [1,2,3]
def reverse_stack(stack):
  stack.reverse()
  print(stack)
reverse_stack(stack6)

#Problem 7
print('')
print('Problem 7')

stack7 = "(())"
stack7_1 = "(()"
def is_balanced(stack):
  left = 0
  right = 0
  for letter in stack:

    if (letter == "("):
      left += 1
    elif (letter == ")"):
```

```
        right += 1

    if ((left % 2 == 0) and (right % 2 == 0)):
        print("Parentheses are balanced")
    else:
        print("Parenthese are not balanced")
is_balanced(stack7)
is_balanced(stack7_1)



#Problem 8
print('')
print('Problem 8')

stack8 = [4, 1, 3, 2]

def sort_stack(stack):
    return sorted(stack)

print(sort_stack(stack8))


#Problem 9
print('')
print('Problem 9')

stack9 = [4,5,2,10]


def next_greater_element(nums):
    result = [-1] * len(nums)
    stack = []

    for i in range(len(nums)):
        while stack and nums[i] > nums[stack[-1]]:
            index = stack.pop()
            result[index] = nums[i]
        stack.append(i)

    return result

print(next_greater_element(stack9))
```

```
#Problem 10
print('')
print('Problem 10')

class MinStack:
  def __init__(self):
    self.stack = []

  def push(self, x):
    self.stack.append(x)

  def pop(self):
    if self.stack:
      self.stack.pop()

  def get_min(self):
    if self.stack:
      return min(self.stack)
    return None



min_stack = MinStack()
min_stack.push(1)
min_stack.push(2)
min_stack.push(3)

print(min_stack.get_min())
print(min_stack.stack)
print(min_stack.pop())
print(min_stack.get_min())
```