Question 1. [5 MARKS]

Read over the definition of this Python function:

```
def c(n):
    """Docstring (almost) omitted."""
    return 1 + max([c(i) for i in n] + [0]) if isinstance(n, list) else 0
```

Work out what each function call produces, and write it in the space provided.

1. `c(5)`

   0

2. `c([])`

   1

3. `c([1, 3, 5])`

   1

4. `c([0, [1, 3, 5], 7])`

   2

5. `c([0, [1, 3, 5, [7, [9]]], 11])`

   4

Question 2. [5 MARKS]

Read over the declarations of the three **Exception** classes, the definition of **raiser**, and the supplied code for **notice** below. Then complete the code for **notice**, using only **except** blocks, and perhaps an **else** block.

```
class E1(Exception):
    pass

class E2(E1):
    pass

class E3(E2):
    pass

def raiser(n: int) -> None:
    """Raise exceptions based magnitude of n"""
    if n < 2:
        raise E3
    elif n < 4:
        raise E2
```

```
    elif n < 6:
        raise E1
    else:
        b = 1 / n

def notice(n: int) -> str:
    """Return messages appropriate to raiser(n).

    >>> notice(15)
    'ok'
    >>> notice("CSC148")
    'purple alert!'
    >>> notice (1)
    'red alert!'
    >>> notice(3)
    'orange alert!'
    >>> notice(5)
    'yellow alert!'
    """
    try:
        raiser(n)
    # Write some "except" blocks and perhaps an "else" block
    # below that make notice(...)
    # have the behaviour shown the the docstring above

    except E3:
        return 'red alert!'
    except E2:
        return 'orange alert!'
    except E1:
        return 'yellow alert!'
    except Exception:
        return 'purple alert!'
    else:
        return 'ok'
```

Question 3. [5 MARKS]

Read over the declaration of the class **Tree** and the docstring of the function **two_all**. Then complete the implementation of **two_all**. You may find the builtin Python function **all(L)** useful, which returns True if all elements of list **L** are True.

```
class Tree:
    """Bare-bones Tree ADT"""
```

```
    def __init__(self: 'Tree',
                 value: object =None, children: list =None):
        """Create a node with value and any number of children"""

        self.value = value
        if not children:
            self.children = []
        else:
            self.children = children[:] # quick-n-dirty copy of list


def two_all(t: Tree) -> bool:
    """Return whether every value in tree t is 2

    precondition - t is a non-empty tree with number values

    >>> tn2 = Tree(2, [Tree(4), Tree(4.5), Tree(2), Tree(5.75)])
    >>> tn3 = Tree(2, [Tree(2), Tree(2)])
    >>> tn1 = Tree(1, [tn2, tn3])
    >>> two_all(tn1)
    False
    >>> two_all(tn3)
    True
    """

    return t.value == 2 and all([two_all(c) for c in t.children])
```

Question 4. [5 MARKS]

Complete the implementation of **push** in the class **ParityStack**, a subclass of **Stack**. Notice that you may use **push**, **pop**, and **is_empty**, the public operations of **Stack**, but you may not assume anything about **Stack**'s underlying implementation. You may find it useful to know that if **n1** is an integer, then **n1** % 2 == 0 if and only if **n1** is even.

```
from csc148stack import Stack
"""
Stack operations:
    pop(): remove and return top item
    push(item): store item on top of stack
    is_empty(): return whether stack is empty.
"""

class ParityStack(Stack):
    """Stack of integers where consecutive elements sum to even"""
```

```
    def push(self: 'ParityStack', n: int) -> None:
        """Add n to top of stack self provided n's sum with its
        predecessor is even.  Otherwise raise an Exception and
        leave stack self as it was before.

        precondition - possibly empty self contains only integers

        >>> s = ParityStack()
        >>> s. push(11)
        >>> s.push(3)
        >>> # now s.push(4) should raise Exception
        """

        if not self.is_empty():
            last = self.pop()
            Stack.push(self, last)
            if not (last + n) % 2 == 0:
                raise Exception('{} + {} is not even'.format(n, last))
        Stack.push(self, n)
```