Question 1.  [5 MARKS]

Read over the definition of this Python function:

```python
def c(s):
    """Docstring (almost) omitted."""
    return sum([c(i) for i in s]) if isinstance(s, list) else 1
```

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

1. `c(5)`  
   1

2. `c([])`  
   0

3. `c(['one', 2, 3.5])`  
   3

4. `c(['one', [2, 'three'], 4, [5, 'six']])`  
   6

5. `c(['one', [2, 'three'], 4, [5, [5.5, 42], 'six']])`  
   8

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.

```python
class EX(Exception):
    pass

class EXX(EX):
    pass

class EXXX(EXX):
    pass

def raiser(n: int) -> None:
    """Raise exceptions based on divisibility of n"""
    if n % 12 == 0:
        raise EXXX
    elif n % 6 == 0:
        raise EXX
    elif n % 3 == 0:
        raise EX
```

```python
# Complete the code for notice
notice
```
raise EX
else:
    b = 1 / n

def notice(n: int) -> str:
    """Return message appropriate to raiser(n).
    >>> notice(17)
    'fine'
    >>> notice("compute")
    'whatever!'
    >>> notice(12)
    'oops! oops! oops!'
    >>> notice(6)
    'oops! oops!'
    >>> notice(3)
    'oops!'
    """
    try:
        raiser(n)
        # Write some "except" blocks and perhaps an "else" block
        # below that make notice(...) make the behaviour shown in the docstring above
        except EXXX:
            return 'oops! oops! oops!'
        except EXX:
            return 'oops! oops!'
        except EX:
            return 'oops!'
        except Exception:
            return 'whatever!'
        else:
            return 'fine'

Question 3. [5 MARKS]

Read over the declaration of the class Tree and the docstring of the function initial_a_whether. Then complete the implementation of initial_a_whether below. It may be helpful to know that the Python builtin function any(L) returns True if list L contains at least one True element, and False otherwise.

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 initial_a_whether(t: Tree) -> bool:
    """Return whether at least one value of tree t begins with "a"
    precondition - t is a non-empty tree with non-empty string values"
    
    return t.value[0] == 'a' or any([initial_a_whether(c) for c in t.children])

Question 4. [5 MARKS]

Complete the implementation of push in the class ContainingStack, 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 s1 and s2 are strings, then s1 in s2 is True if and only if s1 is a substring of s2.

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 ContainingStack(Stack):
    """Stack of strings where each element contains its predecessor"""
```python
def push(self: 'ContainingStack', s: str) -> None:
    """Place s on top of stack self provided it contains
    the string currently on top of self (if there is one). Otherwise,
    raise an Exception and leave stack self as it was

    precondition - possibly empty self contains only strings

    >>> s = ContainingStack()
    >>> s.push("solve")
    >>> s.push("absolved")
    >>> # now s.push("abs") should raise Exception
    ""

    if not self.is_empty():
        last = self.pop()
        Stack.push(self, last)
        if not last in s:
            raise Exception('
                {} is not contained in {}'.format(s, last))
    Stack.push(self, s)
```