Question 1. [5 MARKS]

Read over the definition of this Python function:

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

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

1. a(5)
   0
2. a([])
   0
3. a([1, 3, 5])
   3
4. a([0, [1, 3, 5], 7])
   3
5. a([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.

```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
```
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(...) have 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 two_whether. Then complete the implementation of two_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 two_whether(t: Tree) -> bool:
    """Return whether at least one 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(3, [Tree(6), Tree(7)])
    >>> tn1 = Tree(1, [tn2, tn3])
    >>> two_whether(tn1)
    True
    >>> two_whether(tn3)
    False
    ""
    return t.value == 2 or any([two_whether(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 x is an integer, then x % 2 == 0 is
True if and only if x 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 its sum with the current top element 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(12)
>>> s.push(4)
>>> # now s.push(5) 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)