Question 1.  [5 marks]

Read over the definition of this Python function:

```python
def d(n):
    """Docstring (almost) omitted.""
    return 1 + max([d(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. d(5)
   0
2. d([])
   1
3. d([1, 3, 5])
   1
4. d([0, [1, 3, 5], 7])
   2
5. d([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:
```
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(...) below that make notice(...) 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_all. Then complete the implementation of two_all below. It may be helpful to know that the Python builtin function all(L) returns True if and only if the list L contains only True elements, 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_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 PrefixStack, 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 s2.startswith(s1) is True if and only if s1 is a prefix 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 PrefixStack(Stack):
    """Stack of strings where each is a prefix of its predecessor""

def push(self: 'PrefixStack', s: str) -> None:
"""Place s on top of stack self, provided s is a prefix of the current top element (if there is one). Otherwise raise an Exception and leave stack self as it was

precondition - possibly empty self contains only strings

>>> s = PrefixStack()
>>> s.push("asterisk")
>>> s.push("aster")
>>> # now s.push("asteri") should raise Exception
"""

if not self.is_empty():
    last = self.pop()
    Stack.push(self, last)
    if not last.startswith(s):
        raise Exception('{} not a prefix of {}'.format(s, last))
Stack.push(self, s)