Write recursive contains method

first...

Read over the **_init_** method for class **Tree**:

```
class Tree:
    ''' Represent a Bare-bones Tree ADT'''

    def __init__(self, value=None, children=None):
        ''' (Tree, object, list) -> NoneType

        Create Tree(self) with content value and 0 or more children.
        '''
        self.value = value
        # copy children if not None
        self.children = children.copy() if children else []
```

next...

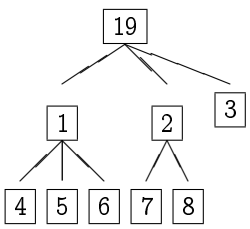
Now, read the header and docstring for the function **contains**, and then answer the questions that follow it.

```
def contains(t, v):
    ''' (Tree, v) -> bool

    Return whether Tree t contains v.

    >>> t = Tree(17)
    >>> contains(t, 17)
    True
    >>> t = descendents_from_list(Tree(19), [1, 2, 3, 4, 5, 6, 7, 8], 3)
    >>> contains(t, 5)
    True
    >>> contains(t, 18)
    False
    '''
```

then...

1. One of the examples in **contains** docstring is simple enough not to require recursion. Write an **if...** expression that checks for this case, and then returns the correct thing. Include an **else...** for when the tree is *less* easy to deal with.

2. Below is a picture of a larger **Tree t**, with several levels. Consider a function call **contains(t, 18)** that passes your tree to the function. Are there smaller trees for which it would be helpful to know whether they contain 18? Which smaller trees are they? Write an example of a function call **contains(t, 18)** on one of these smaller trees. You can access these trees through the variable **t**.



3. Suppose the call in the previous step gives you the correct answer according to the docstring: it returns whether or not the smaller tree contains 18. How will you combine the solutions for all the smaller instances to get a solution for **Tree t** itself? Write code to return the correct thing. Put this code in the **else...** expression that you created in the first step.