Write recursive evaluate method

first...

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

```
    '''Binary Tree node.'''

    def __init__(self, data, left=None, right=None):
        ''' (BTNode, object, BTNode, BTNode) -> NoneType

        Create BTNode (self) with data and children left and right.
        '''
        self.data, self.left, self.right = data, left, right
```

next...

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

```
def evaluate(b):
    ''' (BTNode) -> float

    Evaluate the expression rooted at b.  If b is a leaf,
    return its float data.  Otherwise, evaluate b.left and
    b.right and combine them with b.data.

    Assume:  -- b is a non-empty binary tree
             -- interior nodes contain data in {'+', '-', '*', '/'}
             -- interior nodes always have two children
             -- leaves contain float data

    >>> b = BTNode(3.0)
    >>> evaluate(b)
    3.0
    >>> b = BTNode('*', BTNode(3.0), BTNode(4.0))
    >>> evaluate(b)
    12.0
    '''
```

1. One of the examples in **evaluate** docstring is simple enough not to require recursion (a base case). 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. Another docstring examples is a typical one which can benefit from recursion. Write code that returns the correct value for this case. **Hint:** it may be helpful to use the built-int **eval** function, which takes a string Python expression and evaluates it.

Now implement the body of **evaluate**