Lecture 17 - Object Homework

Our “van” system has a heater, cooler and batteries - and we can run it. Our next step is to take advantage of a little bit of free energy. Remember that before the voltage on the solar panels is 0.4 volts higher than the voltage on the batteries the solar charger can not charge. On cold days we can take advantage of this early morning ramp up.

By adding a electric “heating” pad to the battery compartment and a new switch we can take this power and run it through the heating pad and pre-warm the battery.

One important detail is that we turn off the heating pad before the voltage is too high - if we leave it on then we will burn out the heating pad. So we can run it from about 6 volts to 12.4 - the point where we could possibly take solar and convert it to charge the battery. In reality the we need 0.4 volts more than the battery. The heating pad specification says it can take up to 15.5 volts.

The panels themselves can produce up to 19.8 volts - the charge regulator cuts this off at the 12.8 maximum for the batteries.

Homework 6 - Add a new “class” to the set of classes that run the battery management system to use a heating pad.

Classes and data structures

Often classes are sued to encapsulate the interface to some sort of data that we access.

Let’s talk about a “tree” and how we could build a “tree” class in Python.

First - why use a tree? The answer is speed - but how will a tree effect our code.

What is a “tree”?

How we use Tuples to represent a “tree”.

What are the operations on a “tree”?

```
class BinaryTree:

    def __init__(self, data):

        self.left = None
        self.right = None
        self.data = data

    # Insert into BinaryTree
```

```
        def insert(self, data):

            if self.data:
                if data < self.data:
                    if self.left is None:
                        self.left = BinaryTree(data)
                    else:
                        self.left.insert(data)
                elif data > self.data:
                    if self.right is None:
                        self.right = BinaryTree(data)
                    else:
                        self.right.insert(data)
            else:
                self.data = data

        # Print the Tree Out
        def PrintTree(self):
            if self.left:
                self.left.PrintTree()
            print( self.data),
            if self.right:
                self.right.PrintTree()

        # Inorder traversal
        # Left -> Root -> Right
        def inorderTraversal(self, root):
            res = []
            if root:
                res = self.inorderTraversal(root.left)
                res.append(root.data)
                res = res + self.inorderTraversal(root.right)
            return res




    if __name__ == "__main__":
        bt = BinaryTree(27)
        bt.insert(14)
        bt.insert(35)
        bt.insert(10)
        bt.insert(19)
        bt.insert(31)
        bt.insert(42)

        print(bt.inorderTraversal(bt))
```

Now we can use it something like a dictionary, but having the ability to spit out the data in a sorted order or reverse sorted order is useful. Also the ability to insert at a low cost (and delete at a low cost) is also useful.