Previous Next

8.4.6 Information hiding and getters

MO2.2

Another concept of object-oriented programming is *information hiding*, which refers to keeping the data attribute of an object hidden from the users of the object. Python however does not enforce information hiding. For example, consider an IVP object `myIVPobject`. It is allowable in Python to access any of the data members, such as demonstrated in the following code:

```
print(f"The final time is {myIVPobject._tF}")
```

This code does not seem too worrying as the object's value of `_tF` is just being printed. However, if the implementation of the IVP class were changed such that the final time were kept in a different variable, e.g. `_tF_`, then the above print statement would no longer work (and would cause an error).

Rather, to (somewhat) implement information hiding in Python, the information in the data attributes should only be accessed through a class method. This leads to the implementation of so-called *getter* methods, as can be seen in the code below:

```
class IVP():
    def __init__(self, uI, tI, tF, p, f):
        """
        Args:
            uI (float list): initial condition of
state.
            tI (float): initial time.
            tF (float): final time.
            p (dictionary): set of fixed parameters.
            f (function): takes as input u,t,p and
returns du/dt
        """

        self._uI = uI[:]
        self._tI = tI
        self._tF = tF
        self._p  = copy.deepcopy(p)
        self._f  = f
        self._M  = len(uI)

    def evalf(self, u, t):
        """
        Args:
            u (float list): current solution.
            t (float): current time.

        Returns:
            float list: f(u,t,p).
        """

        return self._f(u, t, self._p)

############ getter methods ############
```

Discussions

All posts sorted by recent activity

Reason for using getters/setters is break
Tacetman
3

Private attributes I tried out the following
JavierM0401
3

Why not use the property decorat ...
__kiwi__
3

```
def get_tI(self):
    """
    Returns:
        float: initial time.
    """
    return self._tI

def get_tF(self):
    """
    Returns:
        float: final time.
    """
    return self._tF

def get_uI(self):
    """
    Returns:
        float list: initial state
    """
        return self._uI[:]

    def get_p(self, name):
        """
        Arg:
            name (key): a key which should be in the
object's parameter
            dictionary

        Returns:
            value of parameter key given by name
        """
        return self._p[name]
```

Then, the value of data attributes is accessed through the getter. For example, the previous print statement using a getter would be:

```
print(f"The final time is {myIVPobject.get_tF()}")
```

The getter methods can also be used so that the dictionary `_p` does not need to be passed by the `evalf` method to the `_f` function. Instead, the object can be passed and then the getter method `get_p` can be used. Here's the code for the modified `evalf` method in which the object reference (i.e. `self`) is passed to `_f`. In this new implementation, we pass `self` as the first argument:

```
def evalf(self, u, t):
    """
    Args:
        u (float list): current solution.
        t (float): current time.

    Returns:
        float list: _f(self,u,t).
    """
    return self._f(self, u, t)
```

This then impacts the IVP user's implementation of their own

This then impacts the IVP user's implementation of their own `evalf` function. For example, here's the modified `coffee_evalf` function:

Previous Next

properties and setters for those properties.

Video on information hiding and getters

Start of transcript