Why not use the property decorator
__kiwi__ 4w

If the goal is to minimize writing to the data attributes and avoid changes to known mutables why not do this? This is just a rewrite of your v1 code using decorators to avoid "get_this" and dunder to increase the "security".

```
from copy import deepcopy

class IVP():
    def __init__(self, uI, tI, tF, p):
        """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 = deepcopy(uI)
        self.__tI = tI
        self.__tF = tF
        self.__p  = deepcopy(p)
        # self.__f  = f  # removed at 8.5

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

        Returns:
            float list: _f(self,u,t).
        """
        raise NotImplementedError("evalf is not implemented for this object")
        # return self.__f(self, u, t)

    @property
    def tI(self):
        """Returns: float: initial time."""
        return self.__tI

    @property
    def tF(self):
        """Returns: float: final time."""
        return self.__tF

    @property
    def uI(self):
        """Returns: float list: initial state"""
        return self.__uI

    @property
    def p(self):
        """Returns: a value from p Dict so use form inst.p(name)"""
        return lambda *k: self.__p[k[0]] if k else "Please provide key name"

    def __len__(self):
        """len is defined as number of states (in _uI)"""
        return len(self.__uI)
```

and the inheriting Coffee class:

```
class coffeeIVP(IVP):
    def evalf(self, Tc, t):
        """Args:
            Tc (float list): current temperature of coffee.
            t (float): current time
        Returns:
            f (float list): returns dTc/dt
        """
        mc = self.p('mc')
        cc = self.p('cc')
        h = self.p('h')
        A = self.p('A')
        Tout = self.p('Tout')
        f = h*A / (mc*cc) * (Tout - Tc[0])
        return [f]
```

Related to 8 Initial Value Problems, Python Classes, and Discretization / 8.4 Introduction to Python Classes / 8.4.6 Information hiding and getters

Showing 1 endorsed response Newest first

Endorsed Staff 4w

Add comment

__kiwi__ 4w

Thanks Andrew. Just for anyone who's interested the following is a little "tutorial" I provide at 6.00.1x.

Getters and Setters are taught in the form they are here because this is a Computer Science course and they are common in other languages like Java. But Python is *a language for consenting adults* and information hiding by default is not pythonic.

So what do we do in Python?

The pythonic way is to make class attributes public **unless there is good reason** to do otherwise. So if class Mine has an attribute deep the way we read or write to it in instance SilverDollar is:

```
class Mine():
    def __init__(self, deep):
        self.deep = deep

SilverDollar = Mine(30)
SilverDollar.deep = 50 # to set x and
SilverDollar.deep  # to get the value of deep
```

No ugly stuff, just `instance_name.variable_name`.

But what if some of our users are idiots so we need to check that deep is a positive number but under 2000 (its a mine not a mountain). Similarly, mines deeper than 1000 are dangerous to miners so we need to hide them from the work & safety people - any mines deeper than 1000 will just return 1000. So, what's the pythonic way? Properties. The class looks like this but the classes user can access the attributes as above preserving the original simple interface:

```
class Mine():
    def __init__(self, deep):
        self.deep = deep

    @property
    def deep(self):
        if self.__deep > 1000:
            return 1000
        return self.__deep

    @deep.setter
    def deep(self, deep):
        if deep > 2000:
            self.__deep = 2000
        elif deep <= 0:
            self.__deep = 0
        else:
            self.__deep = deep

# and try it out by creating an instance
SilverDollar = Mine(-400)
print(SilverDollar.deep)
SilverDollar.deep = 5055
```

Showing 10 results for "state" Clear results

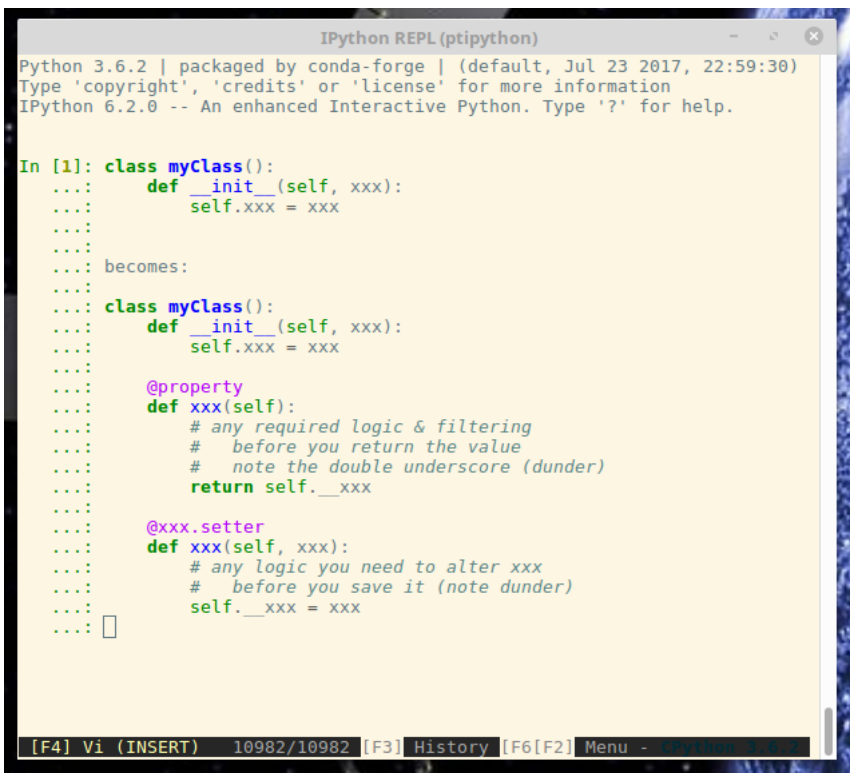
All posts sorted by recent activity

My posts | All posts | Topics | Learners

state

Add a post

So you start out without information hiding. Then you can add it later if needed but without altering your users' interface. Based on - for more information.



Code at pastebin.com

Showing 1 response

wangaj_mit Staff 4w

You're absolutely right that using property decorators and dunder naming would be more idiomatic Python. Our target audience, though, is for those who have just completed 6.00.1x. I don't believe they cover decorators at all, and dunder naming might just be touched upon. In the end, these are somewhat Python-specific concepts and a bit beyond the scope of our prerequisites. But we appreciate the thought!

My posts | All posts | Topics | Learners

21 4w

Pset 3: Errors on Vocareum although all my values are ...
adevrent
4 6d

PSET 3: Assertion Errors in Contour Plot For PSET 3 contou ...
sandipan_dey
5 2w

Stiffness issue? No preview available
JavierM0401
3 2w

Explicit methods convergence No preview available
JavierM0401
5 4w

Leap frog What is leap frog and in which section was it covered?
stp2004
4 4w

I don't understand what is being asked for finger exercise 3.3.1 i ...
pkchong79
11 4w

list comprehensions Thought I would share, th ...
dgarman
2 4w

Why not use the property decorator If the goal is to minimize w ...
__kiwi__
3 4w

Interesting dynamic models I find interesting all the models pre ...
JavierM0401
3 4w

Add a response

edX

About
Affiliates
edX for Business
Open edX
Careers
News

Legal

Terms of Service & Honor Code
Privacy Policy
Accessibility Policy
Trademark Policy
Sitemap
Cookie Policy
Your Privacy Choices

Connect

Idea Hub
Contact Us
Help Center
Security
Media Kit

© 2023 edX LLC. All rights reserved.
深圳市恒宇博科技有限公司 粤ICP备17044299号-2