

Forum:[Hack Forums](#)

Thread: [Extremely noob question](#)

SIM-2362662 - January 10, 2020, 6:13 pm

My code:

Code class A:

```
def A_display(self, msg):  
    print(msg)
```

class B:

```
    classA=A()  
    def reveal(self):  
        self.classA.A_display("Calling Inner class function from Outer class")
```

clB=B()

B.reveal()

Error:

Quote:Traceback (most recent call last):

File "C:/Users/ghioc/Desktop/PythonWorld/CAM4/MAIN.py", line 44, in <module>

B.reveal()

TypeError: reveal() missing 1 required positional argument: 'self'

Why is this happening and how can I avoid this error?

Ri?dle - January 10, 2020, 9:19 pm

You're doing Code B.reveal()when you should be doing

Code B().reveal()or

Code clB = B()

clB.reveal()

Nomad.Null - January 12, 2020, 10:40 am

You forgot to provide your `__init__()` method for the class. Without an `__init__()` method, self is meaningless.

The `__init__()` method is the "instantiation" method for each class, basically the instructions that are run each time a new instance of the class is created. The first argument of the `__init__()` method is always the name of the instance (conventionally "self" but it could be anything. You could make it "potato" if you like.

Here is your fixed code.

Code class A:

```
def __init__(self):  
    pass
```

```
def A_display(self, msg):  
    print(msg)
```

```
class B:
    def __init__(self):
        self.classA = A()

    def reveal(self):
        self.classA.A_display("Calling Inner class function from Outer class")
```

```
b = B()
b.reveal()
And here is the potato code
```

```
Code class A:
    def __init__(potato):
        pass

    def A_display(potato, msg):
        print(msg)
```

```
class B:
    def __init__(potato):
        potato.classA = A()

    def reveal(potato):
        potato.classA.A_display("Calling Inner class function from Outer class")
```

```
b = B()
b.reveal()
```

Ri?dle - January 12, 2020, 11:06 am

(Jan 12, 2020 11:40 AM)True Demon Wrote: [CITE](#) You forgot to provide your `__init__()` method for the class. Without an `__init__()` method, `self` is meaningless.

The `__init__()` method is the "instantiation" method for each class, basically the instructions that are run each time a new instance of the class is created. The first argument of the `__init__()` method is always the name of the instance (conventionally "self" but it could be anything. You could make it "potato" if you like.

Here is your fixed code.

-snip-

This wasn't his problem.

he defined `Code classA=A()` outside of the `__init__` method because he wanted it to be a shared global variable for the classes (probably for memory purposes), he didn't want to initialize a new `Code class A` object every time he initializes a new `Code class B` object. assuming he did it on purpose lol

so his `classA` variable is shared by all instances of class B

Nomad.Null - January 12, 2020, 1:37 pm

I suppose that could be the case, but it's a rather niche use of classes that I've rarely seen in practice. I think we're both making assumptions about his use-case here.

Since he said upfront that it was a "noob" question, I assumed he wasn't terribly familiar with

classes in the first place and was maybe just missing the more traditional solution. If that were the case, it would just be simpler to declare it as a static method like so:

Code class A:

```
@staticmethod
def A_display(msg):
    print(msg)
```

No need for instantiation of the class or anything, that way. `A_display()`

I personally stay away from global class variables because they have a tendency to introduce really weird behavior that can make debugging a PITA. Any time you use global variables in classes, you risk making unintended changes to the class whenever you call the class itself or an instance of it.

A static method is safer, and equally memory efficient in this case. ?

Riddle - January 12, 2020, 4:30 pm

(Jan 12, 2020 02:37 PM)True Demon Wrote: [\[CITE\]](#) I suppose that could be the case, but it's a rather niche use of classes that I've rarely seen in practice. I think we're both making assumptions about his use-case here.

Since he said upfront that it was a "noob" question, I assumed he wasn't terribly familiar with classes in the first place and was maybe just missing the more traditional solution. If that were the case, it would just be simpler to declare it as a static method like so:

Code class A:

```
@staticmethod
def A_display(msg):
    print(msg)
```

No need for instantiation of the class or anything, that way. `A_display()`

I personally stay away from global class variables because they have a tendency to introduce really weird behavior that can make debugging a PITA. Any time you use global variables in classes, you risk making unintended changes to the class whenever you call the class itself or an instance of it.

A static method is safer, and equally memory efficient in this case. ?

Yeah, I just meant the error he got wasn't the result of not having an init method, i'm sure he'll learn something new from your previous post though, you explained it quite well. I doubt he made it a global variable on purpose as well