TypeError: method() takes 1 positional argument but 2 were given

If I have a class ...

```
class MyClass:

    def method(arg):
        print(arg)
```

... which I use to create an object ...

```
my_object = MyClass()
```

... on which I call `method("foo")` like so ...

```
>>> my_object.method("foo")
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: method() takes exactly 1 positional argument (2 given)
```

... why does Python tell me I gave it two arguments, when I only gave one?

python python-3.x arguments self

asked May 29 '14 at 23:27
Zero Piraeus
24.3k ● 15 ● 78 ● 112

4 Answers

In Python, this:

```
my_object.method("foo")
```

... is syntactic sugar, which the interpreter translates behind the scenes into:

```
MyClass.method(my_object, "foo")
```

... which, as you can see, does indeed have two arguments - it's just that the first one is implicit, from the point of view of the caller.

This is because most methods do some work with the object they're called on, so there needs to be some way for that object to be referred to inside the method. By convention, this first argument is called `self` inside the method definition:

```
class MyNewClass:

    def method(self, arg):
        print(self)
        print(arg)
```

If you call `method("foo")` on an instance of `MyNewClass`, it works as expected:

```
>>> my_new_object = MyNewClass()
>>> my_new_object.method("foo")
<__main__.MyNewClass object at 0x29045d0>
foo
```

Occasionally (but not often), you really *don't* care about the object that your method is bound to, and in that circumstance, you can decorate the method with the builtin `staticmethod()` function to say so:

```
class MyOtherClass:

    @staticmethod
    def method(arg):
        print(arg)
```

... in which case you don't need to add a `self` argument to the method definition, and it still works:

```
>>> my_other_object = MyOtherClass()
>>> my_other_object.method("foo")
foo
```

answered May 29 '14 at 23:27
Zero Piraeus
24.3k ● 15 ● 78 ● 112

10 In short: Adding `self` as first argument to the method solves the problem. – amoebe Jun 10 at 11:15

Something else to consider when this type of error is encountered:

I was running into this error message and found this post helpful. Turns out in my case I had overridden an **init**() where there was object inheritance.

The inherited example is rather long, so I'll skip to a more simple example that doesn't use inheritance:

```
class MyBadInitClass:
    def __init__(self, name):
        self.name = name

    def name_foo(self, arg):
        print(self)
        print(arg)
        print("My name is", self.name)


class MyNewClass:
    def new_foo(self, arg):
        print(self)
        print(arg)


my_new_object = MyNewClass()
my_new_object.new_foo("NewFoo")
my_bad_init_object = MyBadInitClass(name="Test Name")
my_bad_init_object.name_foo("name foo")
```


Result is:

```
<__main__.MyNewClass object at 0x033C48D0>
NewFoo
Traceback (most recent call last):
  File "C:/Users/Orange/PycharmProjects/Chapter9/bad_init_example.py", line 41, in
<module>
    my_bad_init_object = MyBadInitClass(name="Test Name")
TypeError: object() takes no parameters
```

PyCharm didn't catch this typo. Nor did Notepad++ (other editors/IDE's might).

Granted, this is a "takes no parameters" TypeError, it isn't much different than "got two" when expecting one, in terms of object initialization in Python.

Addressing the topic: An overloading initializer will be used if syntactically correct, but if not it will be ignored and the built-in used instead. The object won't expect/handle this and the error is thrown.

In the case of the sytax error: The fix is simple, just edit the custom init statement:

```
def __init__(self, name):
    self.name = name
```

answered Jan 4 '16 at 4:33

Jonru2016

81 ● 1 ● 1

It occurs when you don't specify the no of parameters the *init*() or any other method looking for

For example

```
class Dog:
    def __init__(self):
        print("IN INIT METHOD")

    def __unicode__(self,):
        print("IN UNICODE METHOD")

    def __str__(self):
        print("IN STR METHOD")

obj=Dog("JIMMY",1,2,3,"WOOF")
```

When you run the above programme ,it gives you an error like that **TypeError: __init__() takes 1 positional argument but 6 were given**

how we can get rid of this thing?

just pass the parameters ,what *init*() method looking for

```
class Dog:
    def __init__(self, dogname, dob_d, dob_m, dob_y, dogSpeakText):
        self.name_of_dog = dogname
        self.date_of_birth = dob_d
        self.month_of_birth = dob_m
        self.year_of_birth = dob_y
        self.sound_it_make = dogSpeakText

    def __unicode__(self, ):
        print("IN UNICODE METHOD")
```

```
    def __str__(self):
        print("IN STR METHOD")


obj = Dog("JIMMY", 1, 2, 3, "WOOF")
print(id(obj))
```

edited Apr 15 at 20:38

Al Sweigart
3,154 ● 1 ● 23 ● 37

answered Feb 3 at 3:45

TRINADH KOYA
617 ● 7 ● 11

Also, as in my case, make sure you actually still have a parameter in the function. I overlooked that.

I had

```
class MyClass():
    def foo(foo_string):
        return get_bar(foo_string)
```

When I went and changed the method and removed the parameter, I forgot to update my other files that used the same function.

TLDR; Refactor your code.

answered 8 hours ago

ThinkDigital
13 ● 2
