How to Avoid the 7 Most Frequent Mistakes in Python Programming

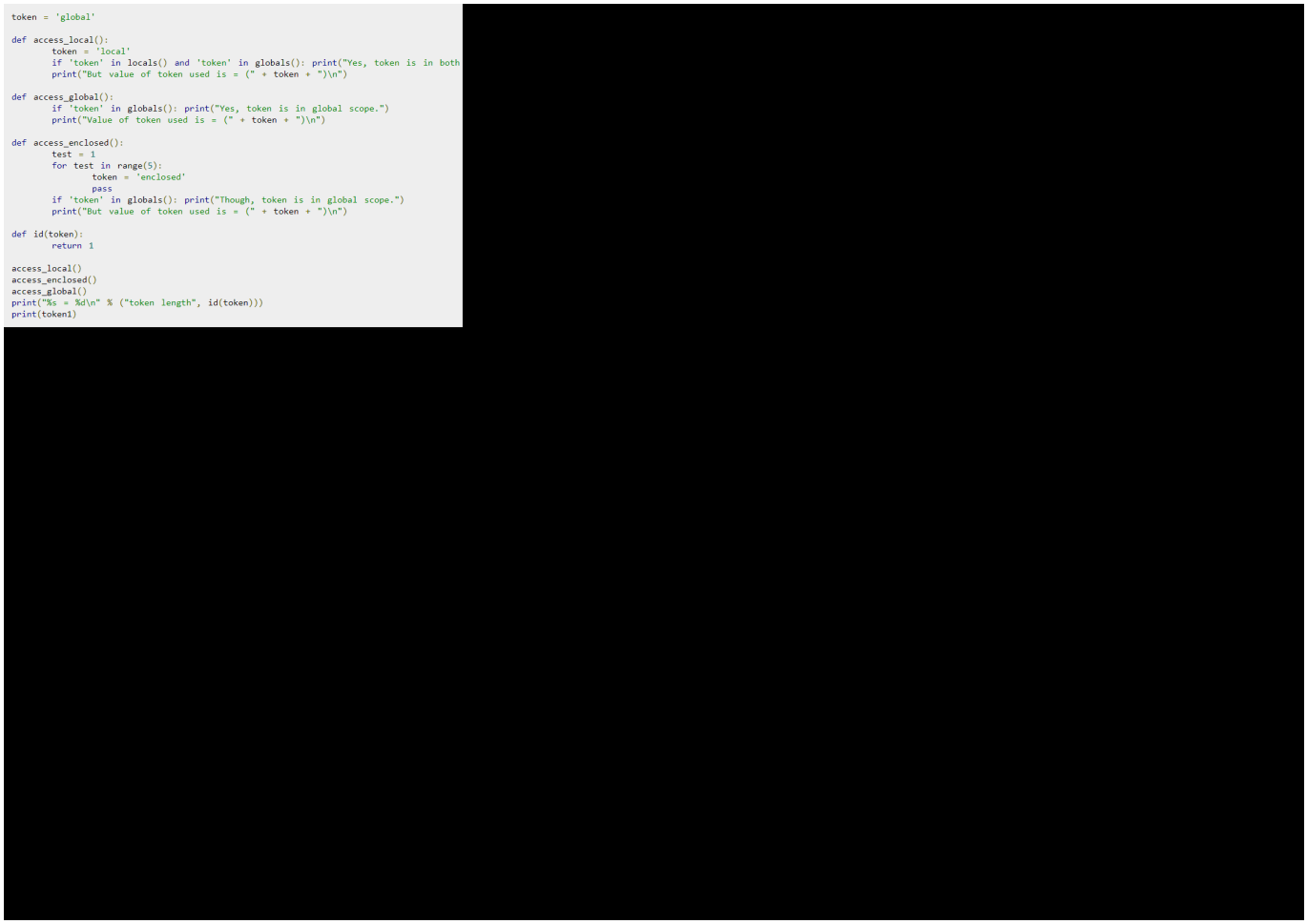
Author : admin

Python is very appealing for ***Rapid Application Development for many reasons***, including **high-level built in data structures, dynamic typing and binding**, or to use as glue to connect different components. **It's simple and easy to learn but new Python developers can fall in the trap of missing certain subtleties.**

Here are 7 common mistakes that are harder to catch but that even more experienced Python developers have fallen for.

1. The misuse of expressions as function argument defaults

Python allows developers to indicate optional function arguments by giving them default values. In most cases, this is a great feature of Python, but it can create some confusion when the default value is mutable. In fact, the common mistake is thinking that the optional argument is set to whatever default value you've set every time the function argument is presented without a value. It can seem a bit complicated, but the answer is that the default value for this function argument is only evaluated at the time you've defined the function, one time only.



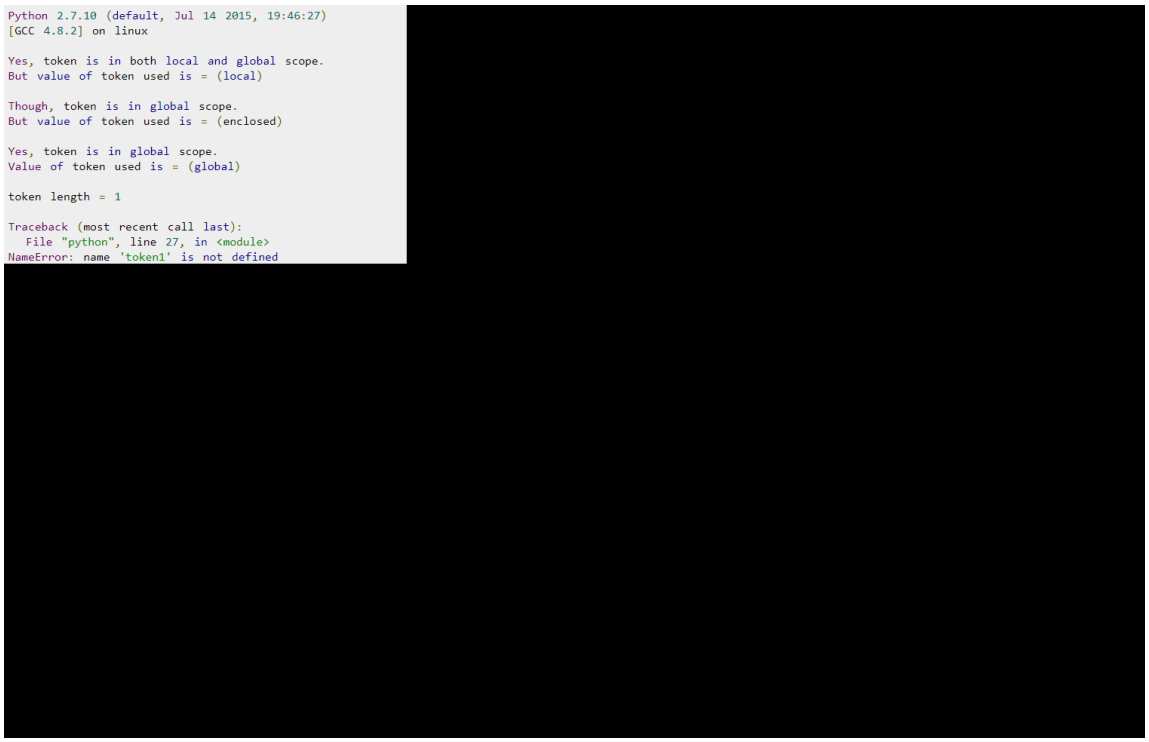
2. Incorrect use of class variables

Python handles class variables internally as dictionaries and they will follow the **Method Resolution Order (MRO)**. If an attribute is not found in one class it will be looked up in base classes so references to one part of the code are actually references to another part, and that can be quite difficult to handle well in Python. For class attributes, I recommend reading up on this aspect of Python independently to be able to handle them.

```
class Employee(object):
      def __init__(self, exp):
            self._exp = exp

      @property
      def exp(self):
            return self._exp

      @exp.setter
      def exp(self, value):
            self._exp = value

      @exp.deleter
      def exp(self):
            del self._exp

emp = Employee(10)
print("default: ", emp.exp)

emp.exp = 20
print("Updated: ", emp.exp)
```

3. Incorrect specifications of parameters for exception blocks

There is a common problem in Python when except statements are provided but they don't take a list of the exceptions specified. The syntax except Exception is used to bind these exception blocks to optional parameters so that there can be further inspections. What happens, however, is that certain exceptions are then not being caught by the except statement, but the exception becomes bound to parameters. The way to get block exceptions in one except statement has to be done by specifying the first parameter as a tuple to contain all the exceptions that you want to catch.



4. Failure to understand the scope rules

The scope resolution on Python is built on the **LEGB** rule as it's commonly known, which means **Local, Enclosing, Global, Built-in.** Although at first glance this seems simple, there are some subtleties about the way it actually works in Python, which creates a more complex Python problem. If you make an assignment to a variable in a scope, Python will assume that variable is local to the scope and will shadow a variable that's similarly named in other scopes. This is a particular problem especially when using lists.

5. Modifying lists during iterations over it

When a developer deletes an item from a list or array while iterating, they stumble upon a **well known Python problem** that's easy to fall into. To address this, *Python* has incorporated many programming paradigms which can really simplify and streamline code when they're used properly. **Simple code is less likely to fall into the trap of deleting a list item while iterating over it.** You can also use list comprehensions to avoid this problem.

6. Name clash with Python standard library

Python has so many library modules which is a bonus of the language, but the problem is that you can inadvertently have a name clash between your module and a module in the standard library. The problem here is that you can accidentally import another library which will import the wrong version. To avoid this, it's important to be aware of the names in the standard library modules and stay away from using them.

```
class Employee(object):
     def __init__(self, name, exp):
          self._name = name
          self._exp = exp
     # Java-style getter/setter
     def getName(self):
          return self._name
     def setName(self, name):
          self._name = name
     def getExp(self):
          return self._exp
     def setExp(self, exp):
          self._exp = exp

emp = Employee('techbeamers', 10)
print("Employee-1: ", emp.getName(), emp.getExp())
emp.setName('Python Programmer')
emp.setExp(20)
print("Employee-2: ", emp.getName(), emp.getExp())
```

7. Problems with binding variables in closures

Python has a **late binding behavior** which looks up the values of variables in closure only when the inner function is called. To address this, you may have to take advantage of default arguments to create anonymous functions that will give you the desired behavior – it’s either elegant or a hack depending on how you look at it, but it’s important to know.

Python is very powerful and flexible and it's a great language for developers, but it's important to be familiar with the nuances of it to optimize it and avoid these errors.

Ellie Coverdale, a technical writer at Essay roo and UK Writings, is involved in tech research and projects to find new advances and share her insights. She shares what she has learned with her readers on the Boom Essays blog.