Try to find a bug in this small Python code. You will be surprised!

Timur Bakibayev Follow
Dec 20, 2020 · 5 min read ★

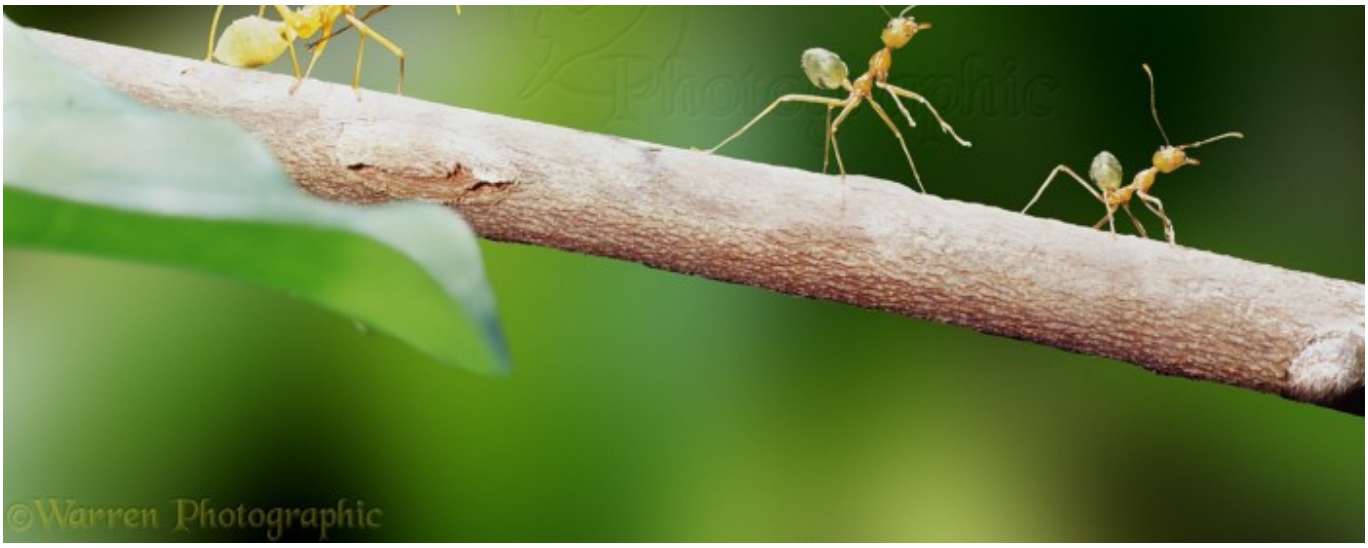
Especially, if you switch from another programming language

Introduction

In my bigger project, I had a bug that was hard to find. It was absolutely not obvious, and I have lost quite some time investigating it.

So I thought that can be interesting to investigate this bug together.

We will build a simple tree, and there will already be a small bug harming it.



Source: https://www.warrenphotographic.co.uk/00725-green-tree-ants-with-mimic-bug

Let's do it!

Building a Tree

Building a tree is simple and straightforward. We only need to define a class Node, and then we connect them using a list.

Here is an implementation:

```
class Node:
    children = []

    def __init__(self, name, parent=None):
        self.name = name
        self.parent = parent
        if parent is not None:
            parent.children.append(self)

    def __str__(self):
        return self.name
```

So, when we create a new Node, we may set a parent node, and the new node will be inserted in the list of parent's children. This way, we will easily be able to print out the complete tree with this simple function:

```
def printer(root, level=0):
    print("  "*level + "├", root.name)
    for node in root.children:
        printer(node, level+1)
```

Here we use a “level” parameter for indentation. We are writing in Python, so, for us, indentation matters most.

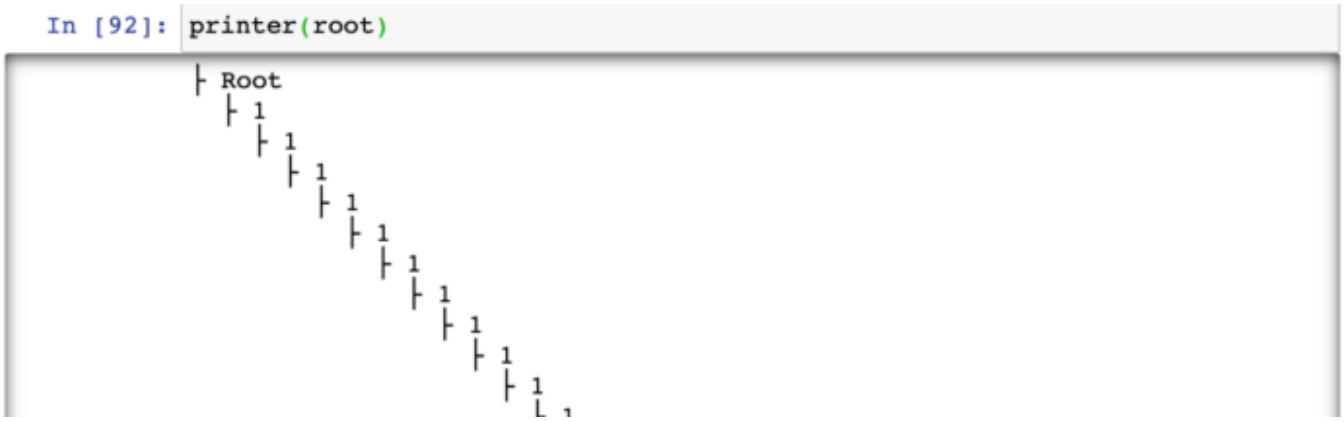
Ok. Now, having everything ready, let’s plant a tree:

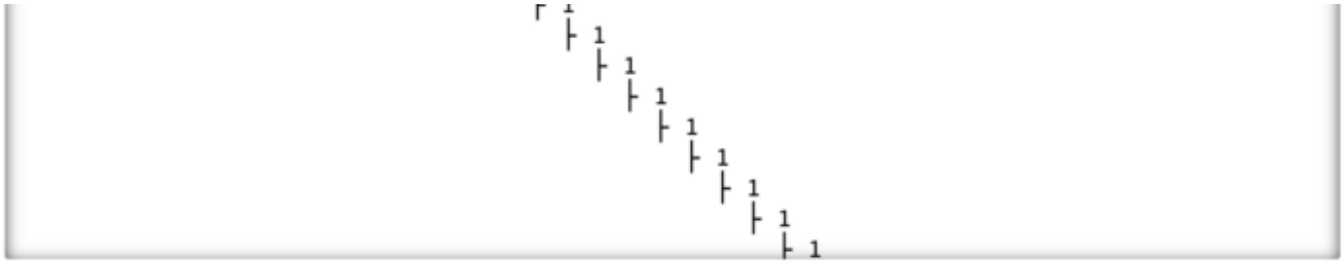
```
root = Node("Root")
node1 = Node("1",root)
node11 = Node("1-1", node1)
node12 = Node("1-2", node1)
node13 = Node("1-3", node1)
node14 = Node("1-4", node1)
node15 = Node("1-5", node1)
node2 = Node("2",root)
node21 = Node("2-1", node2)
node22 = Node("2-2", node2)
node23 = Node("2-3", node2)
node24 = Node("2-4", node2)
node25 = Node("2-5", node2)
```

Nice! Let’s now print it out with our Printer function above:

```
printer(root)
```

That’s it. We have a problem:





You may see that this tree never ends

Investigation

First, it would be much more interesting for you to analyze the code above, and find out the problem on your own.

Our printer is a recursive function that outputs the name of a given node with some indentation, then calls itself for its children, indented a little more. There is no problem with this function, and it works 100% correctly.

Is there a problem with creating the Nodes?

No, we create the first node as a root without a parent (how sad), and then we create two children for the root, following by 5 children for each of them.

So, the problem is somewhere else. And it is only the Class definition left.

Let's take one child and see what is in:

```
print(node25)
#output: 2—5
```

That's fine.

```
print(node25.parent)
#output: 2
```

That's also fine. But what about the parent of the parent? Hmm…

```
print(node25.parent.parent)
#output: root
```

That is also ok. Let's check the children.

```
print(node25.children)
#output: [<__main__.Node object at 0x7fb443dce750>, <__main__.Node
object at 0x7fb443dcec10>, <__main__.Node object at 0x7fb443dced50>,
<__main__.Node object at 0x7fb443dcec90>, <__main__.Node object at
0x7fb443dced90>, <__main__.Node object at 0x7fb443dcea10>,
<__main__.Node object at 0x7fb443ad4290>, <__main__.Node object at
0x7fb443ad40d0>, <__main__.Node object at 0x7fb443ad4210>,
<__main__.Node object at 0x7fb443ad4150>, <__main__.Node object at
0x7fb443dcef50>, <__main__.Node object at 0x7fb443ad4110>]
```

What?! How many children are there?

```
print(len(node25.children))
#output: 12
```

Well… we can see two problems here.

The first problem is that our list looks weird. Of course, this is more important :)

To fix that, simply change "__str__" to "__repr__". This is strange though. When we print out a single Node, it uses the "__str__" function, and in a list, it uses the "__repr__". Why, Python?

Ok, and the second problem (the main one) is that Node25 has way too many children. Let's print them out again, now with the "__repr__" function:

```
print(node25.children)
#output: [1, 1–1, 1–2, 1–3, 1–4, 1–5, 2, 2–1, 2–2, 2–3, 2–4, 2–5]
```

Wow, all nodes except the Root are the children of Node25!

Ok, what about the other node?

```
print(node21.children)
#output: [1, 1–1, 1–2, 1–3, 1–4, 1–5, 2, 2–1, 2–2, 2–3, 2–4, 2–5]
```

Hm…

```
print(root.children)
#output: [1, 1–1, 1–2, 1–3, 1–4, 1–5, 2, 2–1, 2–2, 2–3, 2–4, 2–5]
```

Now the problem is clear — they all share their children.

And here is the reason:

The Core of the Problem

And the core of the problem is here:

```
class Node:
    children = [] # <----- THIS IS THE BUG!

    def __init__(self, name, parent=None):
        self.name = name
        self.parent = parent
        if parent is not None:
            parent.children.append(self)

    def __repr__(self):
        return self.name
```

The thing is, everything declared in a Class this way, are singletons. Yep. Believe me :)

If you used to write in C++ or Java, that is quite a surprise, isn't it?

And what is the correct implementation here?

```
class Node:
    def __init__(self, name, parent=None):
        self.name = name
        self.parent = parent
        self.children = [] # <--- move it here
        if parent is not None:
            parent.children.append(self)

    def __repr__(self):
        return self.name
```

If you want to be sure that an object or a variable belongs to an object, always use "self."!

Now, let's check again:

```
print(root.children)
#out: [1, 2]
```

And the printer:

That's way better! Thanks for reading!

By the way, I have made the exact same mistake in my other articles, including the implementation of Tetris:

But from now on, I will be very careful :)

Python Bug Programming Object Oriented Class