

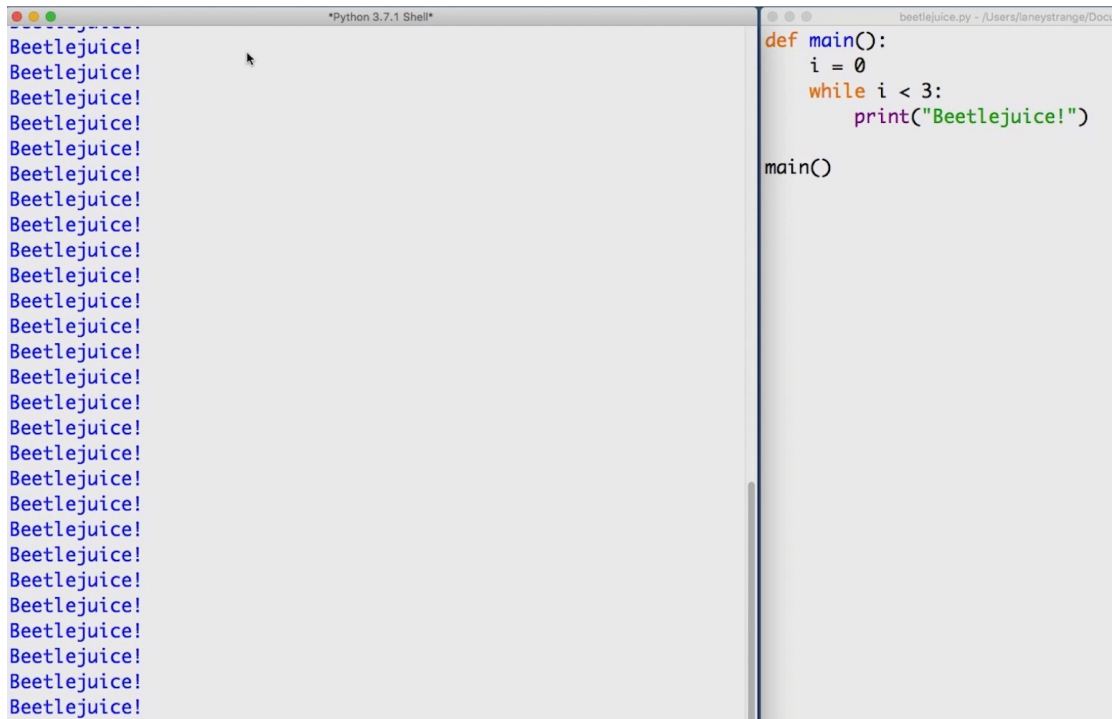
Today we're talking about **debugging** loops, the kinds of problems that might come up when you have a loop in your code. Let's start with this example. I'm gonna use Beetlejuice as an example. If you haven't seen this movie Beetlejuice, go look it up, it's from the 1980s. You can thank me later.

So what happens in the movie is that you want to summon this demon Beetlejuice, so you need to say his name three times. And I'm going to write up just a little bit of code to do that. I'm going to start with my main because I always start with main.

And I'm just going to write a while loop where I'm going to try to summon Beetlejuice by saying his name three times.

```
def main():  
    i = 0  
    while i < 3:  
        print("Beetlejuice!")  
  
main()
```

All right, I save this code. My module is also called Beetlejuice. Now, I'm going to run it. What's going to happen?



```
*Python 3.7.1 Shell*
```

```
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!  
Beetlejuice!
```

```
beetlejuice.py - /Users/aneystange/Docu  
def main():  
    i = 0  
    while i < 3:  
        print("Beetlejuice!")  
  
main()
```

I wanted it to run three times, but it didn't do exactly what I hoped, right?

It's still going. We know that computers don't get bored. We could be going here all day. We could come back, go make a sandwich, come back, and it's still running, right? That's what happens. So I hit `Ctrl+C` to stop it. And I know there's a problem in my code.

It doesn't work the way I wanted it to, so now I have to go and fix it. This is what our theme is for the day. And in a lot of these videos we talk about testing our code and fixing buggy code. The reason we do that is because literally no one writes perfect code the first time.

This is what we do in computer science. We write something, we try it out, we test it. If it doesn't work the way we wanted, we try it again. This example that we just saw is called an infinite loop. If you haven't already written an infinite loop by accident, you will at some point, because we all do.

It's just no big deal. It's just science, we all do it. That takes us to step one. We don't panic. We saw this infinite loop come up. I didn't panic, you didn't panic, everything is fine. Step two is to debug our loop. **Debugging** means we try to figure out what went wrong so that we can fix it.

Easiest way to debug a loop is just to use simple print statements. And because we know all about loops, we know that a loop has to have a way to end, it has to have a way for either the condition to become false. Or we have to have a break statement, one or the other.

Something like that, some way to make the flow of control get out of the loop. So I'm going to start just by looking at my code. There's no break statement, so nothing is going wrong with that. So it must be something up with the condition. This is my condition:

$$i < 3$$

We need that condition to become false at some point or the loop never ends. That's what's happening. So what I'm going to do is just add a print statement so I can see what's going on. Why does this never become false? Something is happening with my variable i that's making this loop never become false.

So what I'm going to do is just print out what's the value of `i`, a simple `print` statement. And hopefully, this will help me see a little bit of what's going on. So all I did was add this `print` statement to debug.

```
def main():
    i = 0
    while i < 3:
        print("Beetlejuice!")
        print("i =", i)

main()
```

I'm going to run it again. And there it goes, forever and ever and ever.

[illegible]

I'm going to hit Ctrl+C to stop it. And look at that. This is telling me exactly what's happening. `i` is always equal to 0. That's not what we want. We want `i` to keep going up until it hits three and then stop. And so, we've debugged it, we found what the problem is.

Next, we need to fix the problem. `i` never changes. So I am going to set

```
i += 1
```

I'm even going to get rid of my debugging statement because I'm fairly confident this is going to help.

```
def main():
    i = 0
    while i < 3:
        print("Beetlejuice!")
        i += 1

main()
```

And I'm going to go back and run it. And there we go.

Beetlejuice! Beetlejuice! Beetlejuice! Michael Keaton appears, and everything goes really well after that, I'm pretty sure. So this is our way of debugging, simplest way, print statements. We print things that are related to the loop condition. There are other kinds of bugs that can show up, though. And sometimes that can be harder to track down.

So what I always do is start with print statements just like we did here. And if that doesn't help, then what I will do is use the Python debugger. There's one that's built into Idle, and it helps us step through our code just one line at a time to see what's going on.

Now, let's take a look at another piece of code. Let's say that I want to do something like a rocket ship. So I want to do five, four, three, two, one, blast off. I'm going to have my variable called `i`. I want `i` to count down. And then, I'm going to say I'm going to print the value of `i`.

And then, once my loop is over, I'm just going to print blast off like the rocket taking off. So my goal here is to print 5, 4, 3, 2, 1, blastoff. That's what we want to see.

```
def main():
    i = 5
    while i >= 0:
        print(i)
        i -= 1
    print("Blastoff!")
main()
```

Now, let me make some space here. And I'm going to run it.

```
>>>
RESTART: /Users/laneystrange/Documents/Northeastern/Align Online
/functions/beetlejuice.py
5
4
3
2
1
0
Blastoff!
>>>
```

Ln: 402 Col: 4

And you can see, not an infinite loop, but also not quite right. This is what we call an **off by one error**. The loop runs one too many times. So you can see it goes 5, 4, 3, 2, 1, 0. We have this extra thing going on. And what I would first try is print statements.

If that's not helping me, then I'm going to switch in to use the debugger. It's like step two, our print statements aren't helping. So in Idle, I'm going to go over here and click on Debug, and it's going to take me to the debugger. What happens with the debugger is that I can choose **breakpoints**.

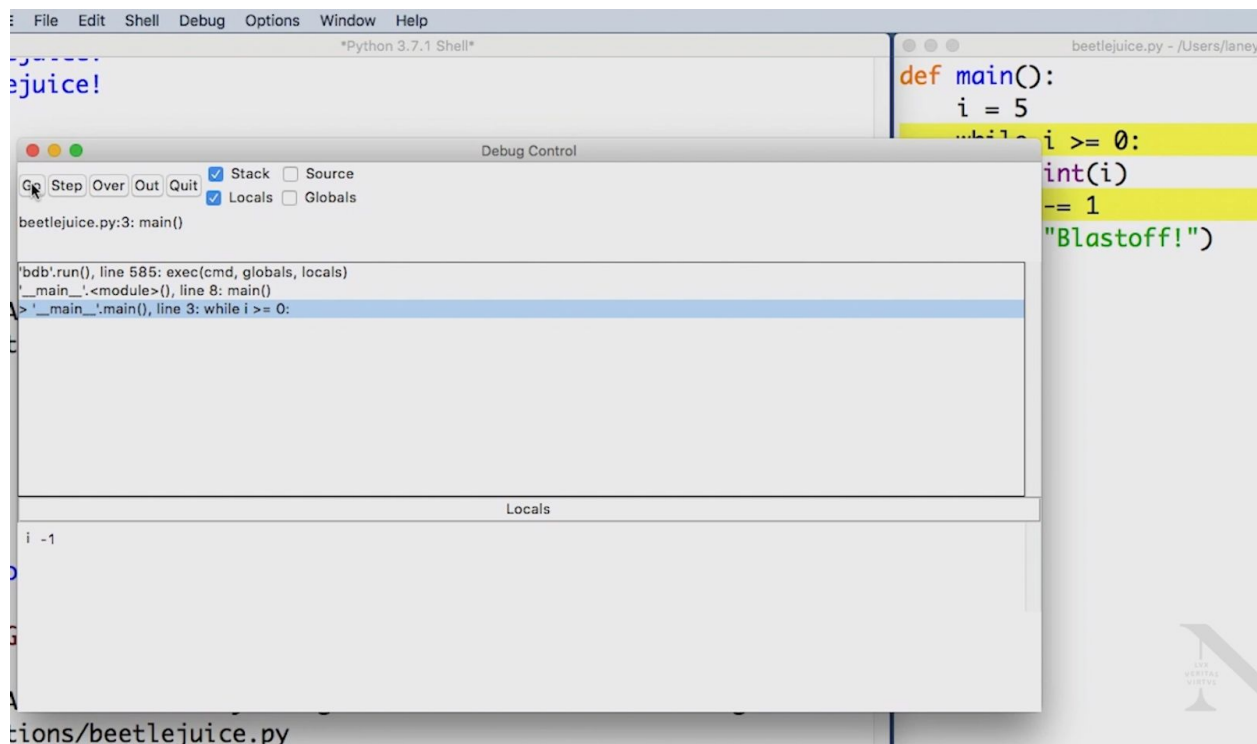
A **breakpoint** is a line in your program where you want to pause execution. And then, you can see all the values of all of your variables at that point. So that might be helpful because

something is off by one somewhere. So I'm going to go in here, and I'm going to set a couple of breakpoints.

This is a problem with the loop. So I'm going to set a breakpoint here where it's checking the condition. All I did was right-click and I choose breakpoint. And I'm going to do another one here where the value of `i` changes. I right-click, I set breakpoint. Now, I have two breakpoints.

So now, when I run my code through the debugger, we can pause at those places and check the values of our variables just to see what's going on. So I'm going to run it just like I normally do, Run Module. And you can see, this is the debugger that pops out.

It says Debug Control in it. And it's got all these variables down here in the bottom screen called locals. We're going to ignore those until we see something familiar. This is all stuff that Python needs. And I'm going to hit the button that says Go, and it's going to run the code until it hits a breakpoint, and then it's going to pause.



So I'm going to hit Go, and it stops at a breakpoint. You can see where it stops because it says line 3, right there in the blue highlight. So `i` is 5 at the beginning, good news. I'm going to hit Go again. And now, it's going to pause again when we do `i` minus equal 1.

That line hasn't happened yet, so we hit Go again. `i` goes down to 4. We hit Go again, `i` goes down to 3. We hit Go again, `i` goes down to 2. `i` goes down to 1. `i` goes down to 0. But then, look what happens. `i` goes down to negative 1.

That's the problem. Why would `i` ever be negative 1? We want it to go 5, 4, 3, 2, 1, blastoff. The very lowest value, the part where we break the loop, `i` should be 0. What's the doing being negative 1, right? So the debugger helped us sort of figure that out.

We can see in this locals, `i` takes on this value negative 1, which it should never have done. But now the debugger's told me where the problem is, I still have to go back and fix it. So I'm going to get rid of the debugger, just going to x out of it.

And all this told me was that I have just a problem in my condition. I have a greater than or equal to. It should just be greater than. Now, it's going to stop when `i` one above where I had it before. So let's do that again. Now, we're going to do Run > Run Module.

5, 4, 3, 2, 1, blastoff. Excellent, that's what we wanted. Okay, so let's recap this video on debugging when you're trying to solve a problem with a loop. First, we don't panic, we know this. But the first thing to try if something is going wrong is you just use print statements.

Print some values that are related to the loop condition, try that first. If that doesn't help, then you switch to the debugger. And then, you're going to step through your code to look at particular values, at particular lines. And the last thing is to not worry, buggy loops happen to everybody.

We all do it, it's not big deal, and these tools should help you figure it out. That's all that we have for today. Thanks for watching Align online.