In the last video, we saw that we could nest conditionals inside of other conditionals. But what if there is exactly one of many actions that can be taken? This will require nesting conditionals for each of the options. Let's take a look at an example where we're printing something for each value between 1 and 5.

```
1  def main():
2      number = int(input("Enter integer between 1 and 5 (inclusive)"))
3
4      if number == 1:
5          print("one")
6      else:
7          if number == 2:
8              print("two")
9          else:
10             if number == 3:
11                 print("three")
12             else:
13                 if number == 4:
14                     print("four")
15                 else:
16                     print("five")
17
18     print("Done")
19
20 main()
21
```

Notice that this program prints exactly 1 of 5 different messages depending upon the user's input. The `if` block contains a single `print` statement, and each `else` block, except for the last, contains another `if` statement. The flow of this program forces the program to check each condition in turn.

The first condition that matches wins. What you should notice is also the way the text is driving to the right with each `if` statement. We can't exactly stop this, since changing the indentation would change the meaning of the program. Now imagine that we want to do this for the numbers between 1 and 100. That would be a crazy amount of indentation.

Fortunately, Python provides a special syntax for this kind of multiway conditional construct using another key word called, **`elif`**. The `elif` syntax exists primarily to avoid this rightward drift of code. Let's see what it looks like. We start with the code that we had previously.

Everywhere there is an `if` inside of an `else` block, I'm going to replace it with an `elif`, fixing the indentation as we go. So the first condition is that number is equal to 1, and then I have an `else` with an `if` inside. I want to change that else to `elif`.

And then fix the indentation. Again, I have an `else` but it's at the wrong indentation, so I fix it. Inside of that `else` is this if number is equal to 3. So I'll change that to `elif`, and then I'll fix the indentation. Again, I have another `else`, which I need to move back.

Inside of that `else` is another `if` statement. So I'm going to change that to `elif`, and then fix

the indentation. Finally, I have the last `else` I need to fix the indentation of the `else`. There is no `if` inside of that, but I do need to fix the indentation of the print.

```
1   def main():
2       number = int(input("Enter integer between 1 and 5 (inclusive)"))
3
4       if number == 1:
5           print("one")
6       elif number == 2:
7           print("two")
8       elif number == 3:
9           print("three")
10      elif number == 4:
11          print("four")
12      elif number == 5:
13          print("five")
14
15      print("Done")
16
17
18  main()
19
```

And there you go. This program is equivalent to the last. You can check that out for yourself by downloading the code samples and running them. Many of you might have said that you could have avoided all that indentation if you had simply just used a series of `if` statements.

Let us take a look at what that looks like.

Left code with `elif`

```
if number == 1:
    print("one")
elif number == 2:
    print("two")
elif number == 3:
    print("three")
elif number == 4:
    print("four")
elif number == 5:
    print("five")

print("Done")
```

Right Code with `if`

```
if number == 1:
    print("one")
if number == 2:
    print("two")
if number == 3:
    print("three")
if number == 4:
    print("four")
if number == 5:
    print("five")

print("Done")
```

Notice that the only difference between these two codes is that, the one on the left uses `elif` and the one on the right uses `if`. Notice that these programs are roughly the equivalent to one another, the results are the same.

Given the number entered by the user, the appropriate word is printed to the screen, and then the message "done" is printed. Despite the similarity between the results, the one on the left is a better program. To understand this, let's look at the execution of these two programs. Supposed a user enters the value one.

The first program tests to if the value is 1. And it is. So it prints the appropriate message, and then skips everything else, and is done. In the second program, the program test to see if the numbers is 1, and it is so it prints the appropriate message.

But then it must also test to see if the number is 2. And it's not. If the number is 3, and it's not, if the number is 4, and it's not, if the number is 5, and it's not, and then it can be done. For a program of this size and complexity, it probably doesn't make a difference.

But there is a difference. And as the complexity of a program increases, the added computation can really affect the efficiency of the program.

Okay, that's all for now, thank you for watching Align online.