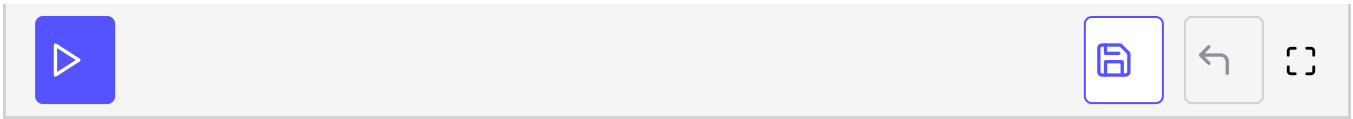
List comprehensions

Let's look at list comprehensions

List comprehensions in Python are very handy. They can also be a little hard to understand when and why you would use them. List comprehensions tend to be harder to read than just using a simple **for** loop as well. You may want to review the looping chapter before you continue.

If you are ready, then we'll spend some time looking at how to construct list comprehensions and learn how they can be used. A list comprehension is basically a one line **for** loop that produces a Python **list** data structure. Here's a simple example:

```
x = [i for i in range(5)]
print(x)
```



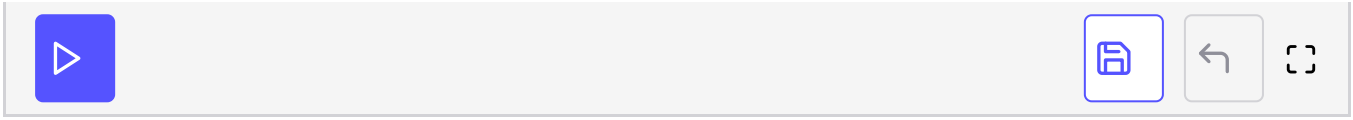
Let's break this down a bit. Python comes with a range function that can return a list of numbers. By default, it returns integers starting at 0 and going up to but not including the number you pass it. So in this case, it returns a list containing the integers 0-4. This can be useful if you need to create a list very quickly. For example, say you're parsing a file and looking for something in particular. You could use a list comprehension as a kind of filter:

```
if [i for i in line if "SOME TERM" in i]:
    # do something
```

I have used code similar to this to look through a file quickly to parse out specific lines or sections of the file. When you throw functions into the mix, you can start doing some really cool stuff. Say you want to apply a function to every element in a list, such as when you need to cast a bunch of strings into integers:

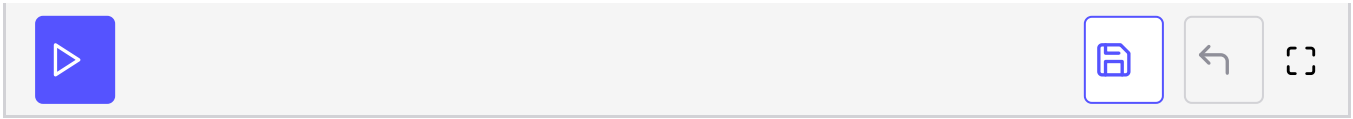
integers:

```
x = ['1', '2', '3', '4', '5']
y = [int(i) for i in x]
print(y) # [1, 2, 3, 4, 5]
```



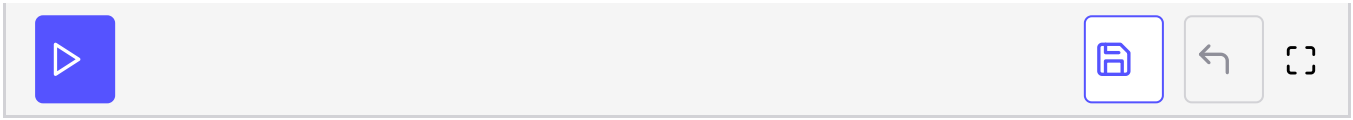
This sort of thing comes up more often than you'd think. I have also had to loop over a list of strings and call a string method, such as strip on them because they had all kinds of leading or ending white space:

```
myStringList = [
  '    Hello how are you?',
  'I\'m good    ',
  '    I\'m good too   ']
myStrings = [s.strip() for s in myStringList]
print(myStrings)
```



There are also occasions where one needs to create a nested list comprehension. One reason to do that is to flatten multiple lists into one. This example comes from the Python documentation:

```
vec = [[1,2,3], [4,5,6], [7,8,9]]
flatVec = [num for elem in vec for num in elem]
print(flatVec) # [1, 2, 3, 4, 5, 6, 7, 8, 9]
```



The documentation shows several other interesting examples for nested list comprehensions as well. I highly recommend taking a look at it! At this point, you should now be capable of using list comprehensions in your own code and use them well. Just use your imagination and you'll start seeing lots of good places where you too can use them.

Now we're ready to move on to Python's dictionary comprehensions!