Nested for Loops

In this lesson, we'll create nested 'for' loops!

We'll cover the following Execution of Nested Loops Using a Nested for Loop The break Keyword The continue Keyword

Execution of Nested Loops

Python lets us easily create loops within loops. There's only one catch: the inner loop will always complete before the outer loop.

For each iteration of the outer loop, the iterator in the inner loop will complete its iterations for the given range, after which the outer loop can move to the next iteration.

Using a Nested for Loop

Let's take an example. Suppose we want to print two elements whose sum is equal to a certain number n.

The simplest way would be to compare every element with the rest of the list. A nested for loop is perfect for this:

```
n = 50
num_list = [10, 4, 23, 6, 18, 27, 47]

for n1 in num_list:
    for n2 in num_list: # Now we have two iterators
        if(n1 + n2 == n):
            print(n1, n2)
```









In the code above, each element is compared with every other element to check if $n_1 + n_2$ is equal to n. This is the power of nested loops!

The break Keyword

Sometimes, we need to exit the loop before it reaches the end. This can happen if we have found what we were looking for and don't need to make any more computations in the loop.

A perfect example is the one we have just covered. At a certain point, n1 is n1 i

That's what the break keyword is for. It can break the loop whenever we want.

Let's add it to the example above:

```
n = 50
num_list = [10, 4, 23, 6, 18, 27, 47]
result = ()
found = False  # This bool will become true once a pair is found

for n1 in num_list:
    for n2 in num_list:
        if(n1 + n2 == n):
            result = (n1, n2)
            found = True  # Set found to True
            break  # Break inner loop if a pair is found

if found:
        break  # Break outer loop if a pair is found

print(result)
```

As we can see, this time result is equal to (23, 27) and (27, 23) is omitted.

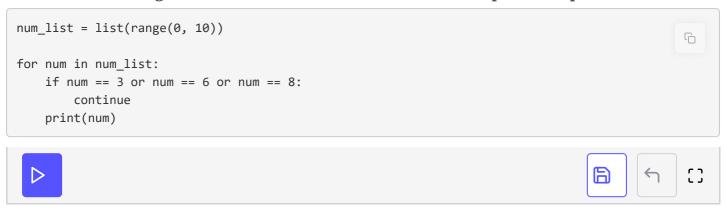
This is because (23, 27) is the first pair which satisfies the condition. We terminate the loop after that using the found bool. Hence, (27, 23) is never computed.

The continue Keyword

When the **continue** keyword is used, the rest of that particular iteration is skipped. The loop *continues* on to the next iteration. We can say that it doesn't break the

loop, but it skips an the code in the current heration and moves to the next one.

We don't need to get into too much detail, so here's a simple example:



The loop goes into the if block when num is 3, 6, or 8. When this happens, continue is executed and the rest of the iteration, including the print() statement, is skipped.

In the next lesson, we'll learn how to make a while loop.