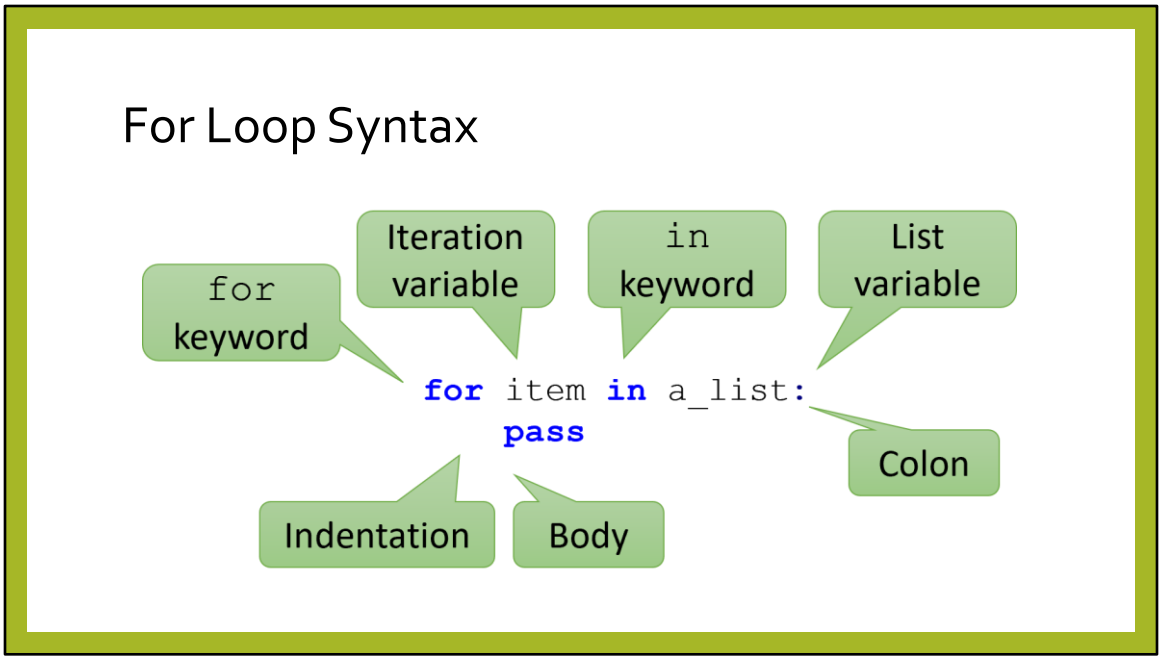
FOR LOOPS

The Python Bakery

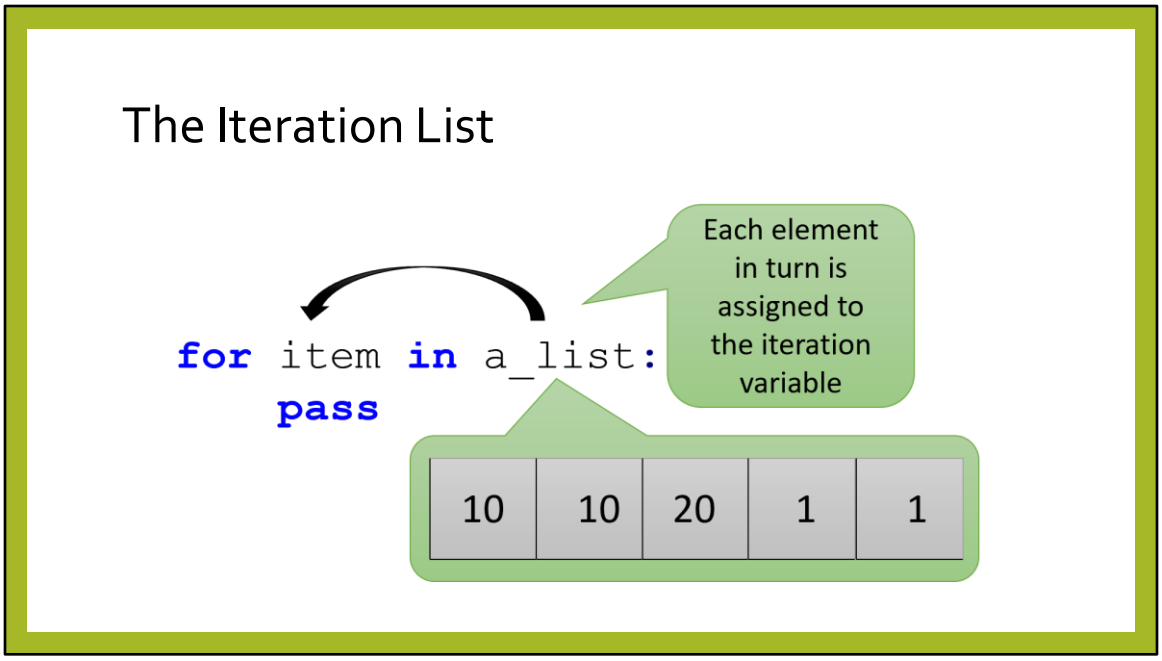
Doing Things Multiple Times

```
costs = [100, 25, 25, 50, 10]

# Without FOR loops
print(costs[0])
print(costs[1])
print(costs[2])
print(costs[3])
print(costs[4])

# With FOR loops!
for a_cost in costs:
    print(a_cost)
```

The program shown here prints out the list of numbers, each number on its own line, in two different ways.
In the first way, we had to write a print statement 5 times, once for each number.
In the second way, however, we only had to write a pair of statements by using a for loop.
The for loop is an amazingly useful tool that let us perform an action on each element of a list.



Here's how we write a FOR loop.
First, we write the word for.
Then we make a new variable named the iteration variable.
We'll come back to this later.
Next, we write the word in.
Then, we put the list variable or value that we want to iterate over.
After that, we need to put a colon (:).
Finally, we put statements inside the body of the loop, indented with 4 spaces.



A crucial element of any for loop is the iteration list.
This is the data that we want to iterate over.
When the loop executes, each item of the list will be assigned to the iteration variable in turn.
It doesn't matter if the list has no items, ten items, or a thousand items; Python will still process each one in turn.

The Iteration Variable

```
for a_temperature in temperatures:
    pass

for a_book in books:
    pass

for a_name in names:
    pass

for a_fruit in fruits:
    pass
```

One of the hardest things to understand about for loops is the iteration variable.
The iteration variable represents the generalized version of each item in the list.
If you have a list of temperatures, it is "a temperature".
If you have a list of books, it is "a book".
By performing operations on this generalized version of a list item, you can work on the entire list while only having to think about one element at a time.
The iteration variable is created by the for loop when it executes.
The type of the iteration variable is the same type as the element type of the list.

The Body

```
prices = [10, 20, 15]

for a_price in prices:
    adjusted = a_price * .9
    print(adjusted)
# Body ends
```

Just like an if statement, we can put statements inside of a for loop.
Each statement is indented with four spaces, and each statement will be executed one-by-one, from top to bottom.
The first unindented line marks the end of the body, just like with functions and if statements.
However, unlike those other bodies, these statements will be repeated again and again, once for each element of the list.

The Flow of a For Loop

```
for a_price in prices:
adjusted = a_price * .9
print(adjusted)
```

Previously, we said that a program was like a river, running from the top to the bottom.
If statements made the stream split into two directions.
For loops also make the stream split, but one of the new streams will move back up.
This is the crucial idea, and its why we call it a loop.
The program will LOOP until each element of the list has been assigned to the iteration variable.

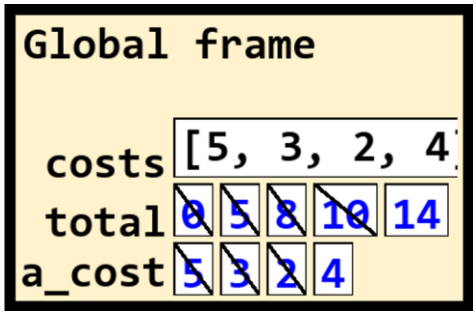
The Scope of a For Loop

```
numbers = [1, 5, 2]
message = "The number is"
for number in numbers:
    print(message, number)
print("The last number was", number)
```

Unlike functions, for loops do not have their own scope.
Variables defined outside of a for loop can be used inside the loop body.
The iteration variable can be used after the loop has finished.
Nothing special has to be done with regards to scope with loops.

Tracing a For Loop

```
costs = [5, 3, 2, 4]
total = 0
for a_cost in costs:
    total += a_cost
print(total)
```



Let's look at a loop in practice.
Notice how the value of the iteration variable changes each time we go through the loop again, and the cursor jumps back up to the start of the loop.
This behavior is crucial!
In fact, the fourth line is executed 4 times, once for each element of the list. Notice how we cross out the previous value of the iteration variable as we complete each element, and the total variable gets updated each time too.
We strongly recommend tracing the code yourself using Thonny or some other debugger that let's you step through the code line by line, to see for yourself how the control flow changes over time.