Lists: Many Items, Plus Indexing and Append

Python for Absolute Beginners · Lesson 12

What you'll learn

- What a list is, and why one variable sometimes needs to hold many values
- How to make a list and read items by position
- How to count a list's items
- How to add an item with append and change an item by position
- How to loop through a whole list

One variable, many values

- So far every variable has held one value: one name, one number, one string
- But real programs deal in groups — a class of students, a shopping list
- A list stores many values in a single variable, in order
- Items sit inside square brackets, separated by commas
- Items can mix kinds, but usually hold one kind of thing, to keep code readable

Make a list

```
fruits = ["apple", "banana", "cherry"]
print(fruits)

['apple', 'banana', 'cherry']
```

Python prints the whole list back, brackets and all, so you can see exactly what is inside.

Reading items by position

- Like string characters, list items are counted from 0
- The first item is position 0, the second is position 1, and so on
- Asking for a position past the end gives an error — there is no fourth fruit in a three-item list
- A handy shortcut: position -1 means the last item, -2 the second-to-last

Read by position

```
fruits = ["apple", "banana", "cherry"]
print(fruits[0])
print(fruits[1])
print(fruits[2])

apple
banana
cherry
```

The same bracket notation you use to make the list, now with a position inside.

The last item shortcut

```
fruits = ["apple", "banana", "cherry"]
print(fruits[-1])

cherry
```

Minus 1 is a handy shortcut for the last item.

Counting with len

```
fruits = ["apple", "banana", "cherry"]
print(len(fruits))

3
```

len() counts a list's items, exactly as it counts a string's characters.

Changing a list: append and replace

- Lists are changeable — unlike strings, you can alter them after making them
- append() sticks a new item on the end
- list[1] = … replaces whatever was at position 1
- The bracket notation on the left of = means “store”
- The order of the other items does not change — only position 1 does

Append and replace

```
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")
print(fruits)
fruits[1] = "blueberry"
print(fruits)

['apple', 'banana', 'cherry', 'orange']
['apple', 'blueberry', 'cherry', 'orange']
```

Read the second output carefully: only position 1 changed.

Looping through a list

- Lists and loops belong together: a for loop walks a list one item at a time
- The loop variable takes each value in turn
- You do not need to know in advance how many items there are
- Append an item later, and the same loop prints one more line with no other change
- Code that works for three items works for three hundred

Loop through a list

```
            fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print("I like " + fruit)

I like apple
I like banana
I like cherry
```

The loop adapts to the list, however many items it holds.

Hands-on exercise

- Make a list of four of your favorite foods; print the whole list, then just the first item
- Print how many items your list has with len()
- Append a fifth food, print the list again, and confirm it grew
- Replace the second item with something else and print the list
- Write a loop that prints “I like …” for every item in your list

Key takeaways

- A list stores many values in order, in one variable, in square brackets.
- Positions start at 0; -1 is the last item; len() counts items.
- Lists are changeable: append() adds to the end, list[i] = … replaces one item.
- A for loop over a list handles every item, however many there are.

Take the quiz

Take the Lesson 12 quiz — 5 quick questions to check what you learned.

Next lesson

Dictionaries: Labeled Storage