9.8. Append versus Concatenate

The `append` method adds a new item to the end of a list. It is also possible to add a new item to the end of a list by using the concatenation operator. However, you need to be careful.

Consider the following example. The original list has 3 integers. We want to add the word “cat” to the end of the list.

Save & Run 5/13/2021, 2:23:03 PM - 2 of 2 Show in CodeLens

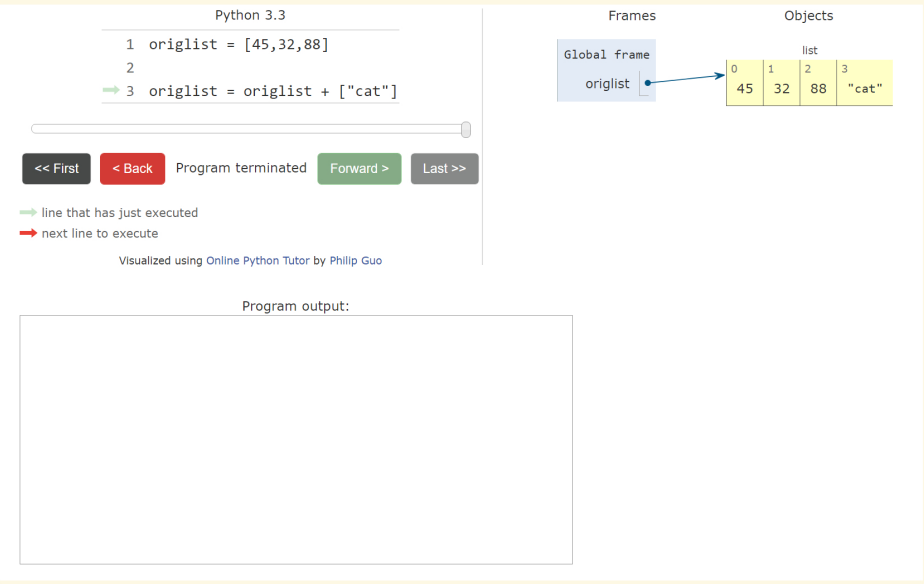
```
1 origlist = [45,32,88]
2
3 origlist.append("cat")
4
```

Activity: 1 -- ActiveCode (clens8_7_1)

Here we have used `append` which simply modifies the list. In order to use concatenation, we need to write an assignment statement that uses the accumulator pattern:

```
origlist = origlist + ["cat"]
```

Note that the word “cat” needs to be placed in a list since the concatenation operator needs two lists to do its work.



Activity: 2 -- CodeLens: (clens8_7_2)

It is also important to realize that with append, the original list is simply modified. On the other hand, with concatenation, an entirely new list is created. This can be seen in the following codelens example where``newlist`` refers to a list which is a copy of the original list, `origlist`, with the new item “cat” added to the end. `origlist` still contains the three values it did before the concatenation. This is why the assignment operation is necessary as part of the accumulator pattern.

Program output:

Activity: 3 -- CodeLens: (clens8_7_3)

This might be difficult to understand since these two lists appear to be the same. In Python, every object has a unique identification tag. Likewise, there is a built-in function that can be called on any object to return its unique id. The function is appropriately called `id` and takes a single parameter, the object that you are interested in knowing about. You can see in the example below that a real id is usually a very large integer value (corresponding to an address in memory). In the textbook though the number will likely be smaller.

```
>>> alist = [4, 5, 6]
>>> id(alist)
4300840544
>>>
```

Save & Run 5/13/2021, 2:23:17 PM - 2 of 2 Show in CodeLens

```
1 origlist = [45,32,88]
2 print("origlist:", origlist)
3 print("the identifier:", id(origlist))             #id of the list before changes
4 newlist = origlist + ['cat']
5 print("newlist:", newlist)
6 print("the identifier:", id(newlist))              #id of the list after concatenta
7 origlist.append('cat')
8 print("origlist:", origlist)
9 print("the identifier:", id(origlist))             #id of the list after append is
10
```

```
origlist: [45, 32, 88]
the identifier: 2
newlist: [45, 32, 88, 'cat']
the identifier: 3
origlist: [45, 32, 88, 'cat']
the identifier: 2
```

Activity: 4 -- ActiveCode (ac8_7_1)

Note how even though `newlist` and `origlist` appear the same, they have different identifiers.

We have previously described x += 1 as a shorthand for x = x + 1. With lists, += is actually a little different. In particular, origlist += ["cat"] appends "cat" to the end of the original list object. If there is another alias for `origlist, this can make a difference, as in the code below. See if you can follow (or, better yet, predict, changes in the reference diagram).

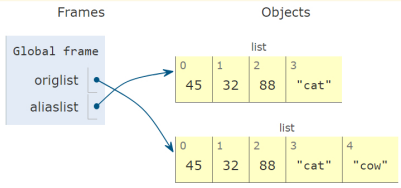
Python 3.3

```
1 origlist = [45,32,88]
2 aliaslist = origlist
3 origlist += ["cat"]
4 origlist = origlist + ["cow"]
```

<< First | < Back | Program terminated | Forward > | Last >>

line that has just executed
next line to execute

Visualized using Online Python Tutor by Philip Guo



Program output:

Activity: 5 -- CodeLens: (clens8_7_2a)

We can use append or concatenate repeatedly to create new objects. If we had a string and wanted to

make a new list, where each element in the list is a character in the string, where do you think you should start? In both cases, you'll need to first create a variable to store the new object.

Save & Run 5/13/2021, 2:23:22 PM - 2 of 2 Show in CodeLens

```
st = "Warmth"
a = []

```

Activity: 6 -- ActiveCode (ac8_72)

Then, character by character, you can add to the empty list. The process looks different if you concatentate as compared to using append.

Save & Run 5/13/2021, 2:23:25 PM - 2 of 2 Show in CodeLens

```
st = "Warmth"
a = []
b = a + [st[0]]
c = b + [st[1]]
d = c + [st[2]]
e = d + [st[3]]
f = e + [st[4]]
g = f + [st[5]]
print(g)

```

```
['W', 'a', 'r', 'm', 't', 'h']
```

Activity: 7 -- ActiveCode (ac8_7_3)

Save & Run 5/13/2021, 2:23:26 PM - 2 of 2 Show in CodeLens

```
st = "Warmth"
a = []
a.append(st[0])
a.append(st[1])
a.append(st[2])
a.append(st[3])
a.append(st[4])
a.append(st[5])
print(a)

```

```
['W', 'a', 'r', 'm', 't', 'h']
```

Activity: 8 -- ActiveCode (ac8_7_4)

This might become tedious though, and difficult if the length of the string is long. Can you think of a better way to do this?

Check your understanding

seqmut-7-1: What is printed by the following statements?

```
alist = [4,2,8,6,5]
alist = alist + 999
print(alist)
```

A. [4,2,8,6,5,999]

B. Error, you cannot concatenate a list with an integer.

Check me Compare me

✔ Yes, in order to perform concatenation you would need to write alist+[999]. You must have two lists.

Activity: 9 -- Multiple Choice (question8_7_1)

You have attempted 10 of 9 activities on this page

9.7. Mutating Methods">

9.9. Non-mutating Methods on Strings">

9.7. Mutating Methods">

✔ Completed. Well Done!

9.9. Non-mutating Methods on Strings">Next Section - 9.9. Non-mutating Methods on Strings

| Back to top