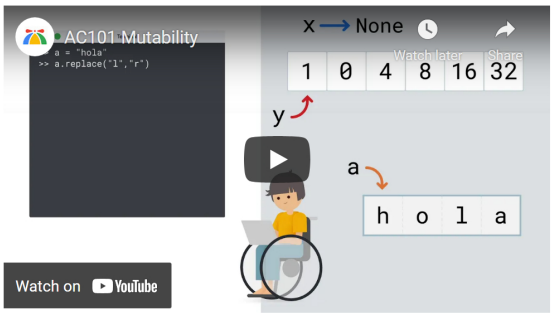
9.2. Mutability

Some Python collection types - strings and lists so far - are able to change and some are not. If a type is able to change, then it is said to be mutable. If the type is not able to change then it is said to be immutable. This will be expanded below.



Activity: 1 -- Video: (mutabilityvid)

9.2.1. Lists are Mutable

Unlike strings, lists are **mutable**. This means we can change an item in a list by accessing it directly as part of the assignment statement. Using the indexing operator (square brackets) on the left side of an assignment, we can update one of the list items.

Save & Run 5/13/2021, 1:46:59 PM - 2 of 2 Show in CodeLens

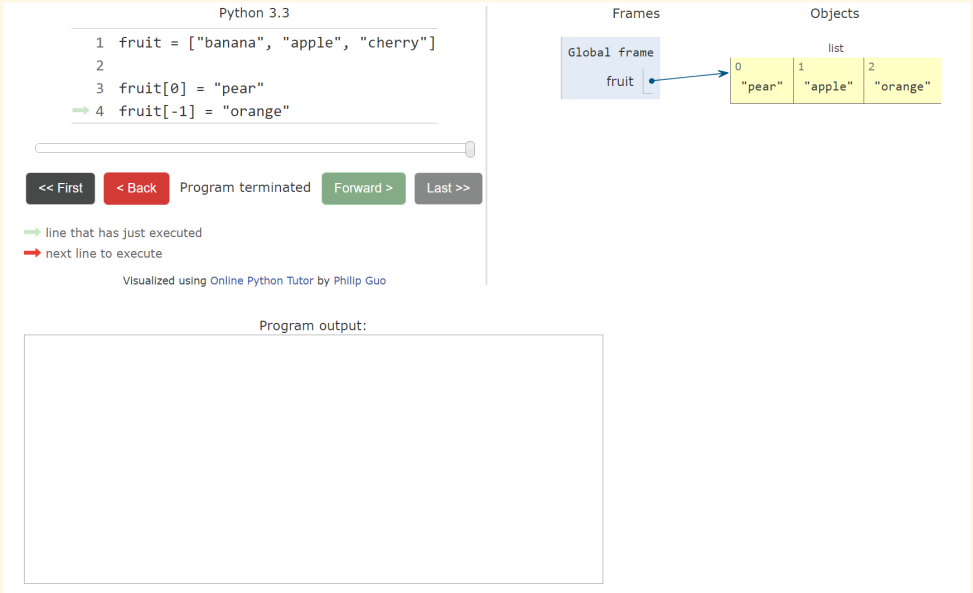
```
1 fruit = ["banana", "apple", "cherry"]
2 print(fruit)
3
4 fruit[0] = "pear"
5 fruit[-1] = "orange"
6 print(fruit)
7
```

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

Activity: 2 -- ActiveCode (ac8_1_4)

An assignment to an element of a list is called **item assignment**. Item assignment does not work for strings. Recall that strings are immutable.

Here is the same example in codelens so that you can step through the statements and see the changes to the list elements.



Activity: 3 -- CodeLens: (clens8_1_1)

By combining assignment with the slice operator we can update several elements at once.

Save & Run 5/13/2021, 1:47:06 PM - 2 of 2 Show in CodeLens

```
alist = ['a', 'b', 'c', 'd', 'e', 'f']
alist[1:3] = ['x', 'y']
print(alist)

```

```
['a', 'x', 'y', 'd', 'e', 'f']
```

Activity: 4 -- ActiveCode (ac8_1_5)

We can also remove elements from a list by assigning the empty list to them.

Save & Run 5/13/2021, 1:47:08 PM - 2 of 2 Show in CodeLens

```
alist = ['a', 'b', 'c', 'd', 'e', 'f']
alist[1:3] = []
print(alist)

```

```
['a', 'd', 'e', 'f']
```

Activity: 5 -- ActiveCode (ac8_1_6)

We can even insert elements into a list by squeezing them into an empty slice at the desired location.

Save & Run 5/13/2021, 1:47:10 PM - 2 of 2 Show in CodeLens

```
alist = ['a', 'd', 'f']
alist[1:1] = ['b', 'c']
print(alist)
alist[4:4] = ['e']
print(alist)

```

```
['a', 'b', 'c', 'd', 'f']
['a', 'b', 'c', 'd', 'e', 'f']
```

Activity: 6 -- ActiveCode (ac8_1_7)

9.2.2. Strings are Immutable

One final thing that makes strings different from some other Python collection types is that you are not allowed to modify the individual characters in the collection. It is tempting to use the `[]` operator on the left side of an assignment, with the intention of changing a character in a string. For example, in the following code, we would like to change the first letter of `greeting`.

Save & Run 5/13/2021, 1:47:12 PM - 2 of 2 Show in CodeLens

```
greeting = "Hello, world!"
```

```
greeting[0] = 'J'             # ERROR!
print(greeting)

```

Activity: 7 -- ActiveCode (ac8_1_1)

Error

```
TypeError: 'str' does not support item assignment on line 2
```

Description

Type errors most often occur when an expression tries to combine two objects with types that should not be combined. Like raising a string to a power

To Fix

To fix a type error you will most likely need to trace through your code and make sure the variables have the types you expect them to have. It may be helpful to print out each variable along the way to be sure its value is what you think it should be.

Instead of producing the output `Jello, world!`, this code produces the runtime error `TypeError: 'str' object does not support item assignment`.

Strings are **immutable**, which means you cannot change an existing string. The best you can do is create a new string that is a variation on the original.

Save & Run 5/13/2021, 1:47:16 PM - 2 of 2 Show in CodeLens

```
greeting = "Hello, world!"
newGreeting = 'J' + greeting[1:]
print(newGreeting)
print(greeting)           # same as it was

```

```
Jello, world!
Hello, world!
```

Activity: 8 -- ActiveCode (ac8_1_2)

The solution here is to concatenate a new first letter onto a slice of `greeting`. This operation has no effect on the original string.

While it's possible to make up new variable names each time we make changes to existing values, it could become difficult to keep track of them all.

Save & Run 5/13/2021, 2:47:37 PM - 4 of 4 Show in CodeLens

```
phrase = "many moons"
phrase_expanded = phrase + " and many stars"
phrase_larger = phrase_expanded + " litter"
phrase_complete = "M" + phrase_larger[1:] + " the night sky."
excited_phrase_complete = phrase_complete[:-1] + "!"

```

Activity: 9 -- ActiveCode (ac8_1_3)

The more that you change the string, the more difficult it is to come up with a new variable to use. It's perfectly acceptable to re-assign the value to the same variable name in this case.

9.2.3. Tuples are Immutable

As with strings, if we try to use item assignment to modify one of the elements of a tuple, we get an error. In fact, that's the key difference between lists and tuples: tuples are like immutable lists. None of the operations on lists that mutate them are available for tuples. Once a tuple is created, it can't be changed.

```
julia[0] = 'X'  # TypeError: 'tuple' object does not support item assignment
```

Check your understanding

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

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

- A. [4,2,True,8,6,5]
- B. [4,2,True,6,5]
- C. Error, it is illegal to assign

Check me | Compare me

✔ Yes, the value True is placed in the list at index 2. It replaces 8.

Activity: 10 -- Multiple Choice (question8_1_1)

seqmut-1-2: What is printed by the following statements:

```
s = "Ball"
s[0] = "C"
print(s)
```

- A. Ball
- B. Call
- C. Error

Check me | Compare me

✔ Yes, strings are immutable.

Activity: 11 -- Multiple Choice (question8_1_2)

You have attempted 12 of 11 activities on this page

9.1. Introduction: Transforming Sequences">
9.3. List Element Deletion">
9.1. Introduction: Transforming Sequences">
✔ Completed. Well Done!
9.3. List Element Deletion">Next Section - 9.3. List Element Deletion

© Copyright 2017 bradleymiller. Created using Runestone 4.1.17.
| Back to top