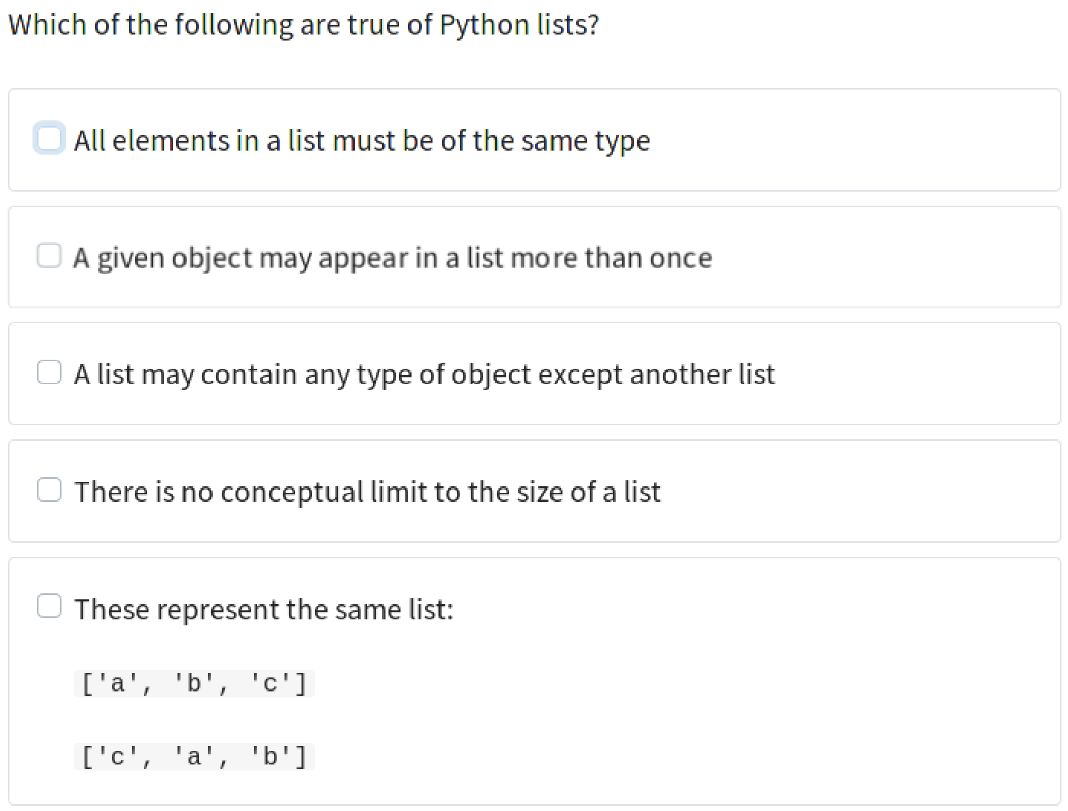
Quiz – Lists and Tuples - Q1



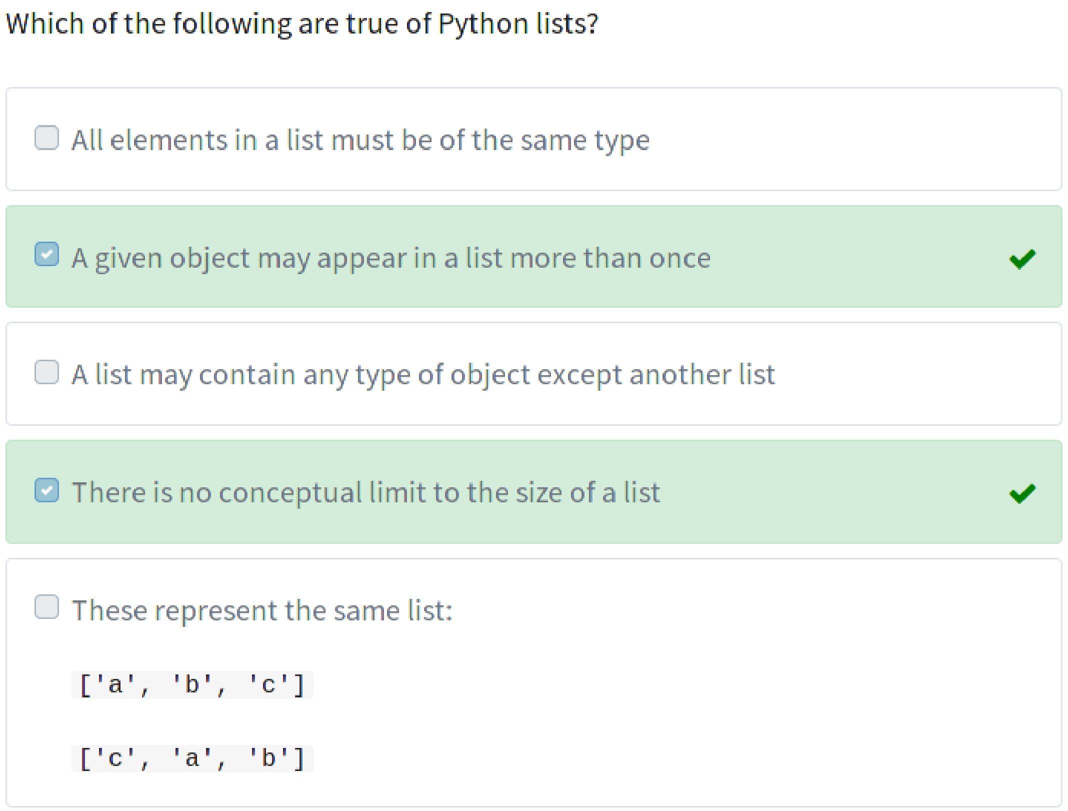
Which of the following are true of Python lists?

- ☐ All elements in a list must be of the same type
- ☐ A given object may appear in a list more than once
- ☐ A list may contain any type of object except another list
- ☐ There is no conceptual limit to the size of a list
- ☐ These represent the same list:

  `['a', 'b', 'c']`

  `['c', 'a', 'b']`

Quiz – Lists and Tuples - A1



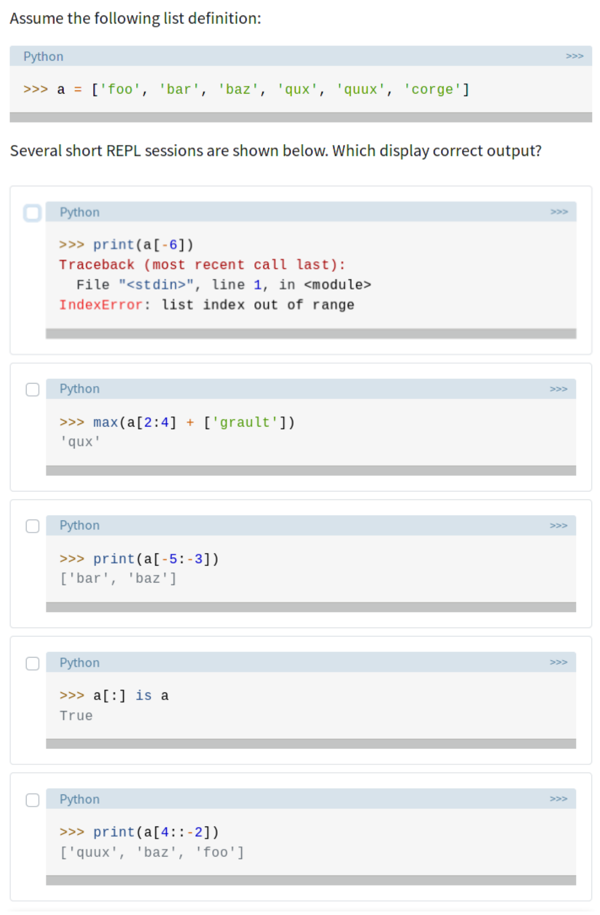
Which of the following are true of Python lists?

- ☐ All elements in a list must be of the same type
- [✓ A given object may appear in a list more than once] ✓
- ☐ A list may contain any type of object except another list
- [✓ There is no conceptual limit to the size of a list] ✓
- ☐ These represent the same list:

```
['a', 'b', 'c']

['c', 'a', 'b']
```

Quiz – Lists and Tuples - Q2



Assume the following list definition:

Python

```
>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
```

Several short REPL sessions are shown below. Which display correct output?

☐ Python

```
>>> print(a[-6])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

☐ Python

```
>>> max(a[2:4] + ['grault'])
'qux'
```

☐ Python

```
>>> print(a[-5:-3])
['bar', 'baz']
```

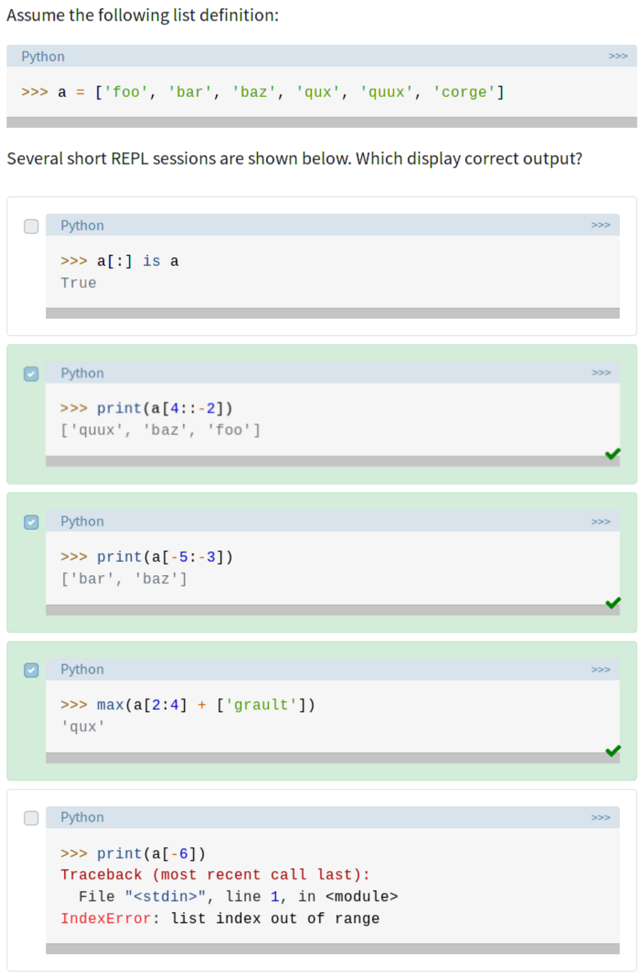
☐ Python

```
>>> a[:] is a
True
```

☐ Python

```
>>> print(a[4::-2])
['quux', 'baz', 'foo']
```

Quiz – Lists and Tuples – A2



Assume the following list definition:

Python

```
>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
```

Several short REPL sessions are shown below. Which display correct output?

[ ] Python

```
>>> a[:] is a
True
```

[✓] Python

```
>>> print(a[4::-2])
['quux', 'baz', 'foo']
```

✓

[✓] Python

```
>>> print(a[-5:-3])
['bar', 'baz']
```

✓

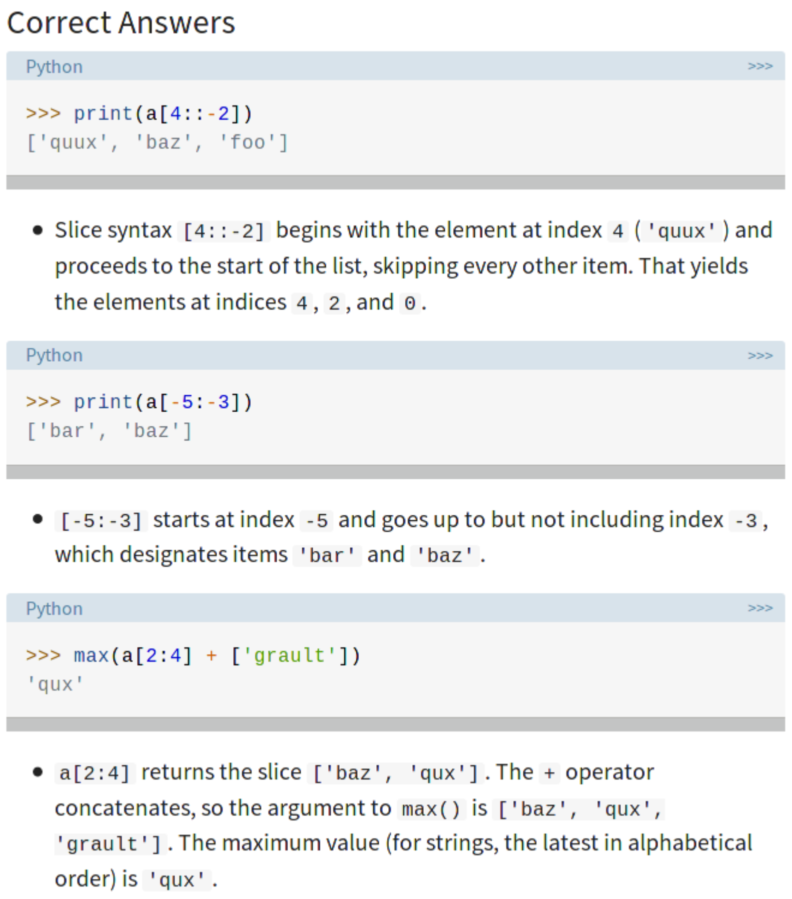
[✓] Python

```
>>> max(a[2:4] + ['grault'])
'qux'
```

✓

[ ] Python

```
>>> print(a[-6])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```



Correct Answers

Python

```
>>> print(a[4::-2])
['quux', 'baz', 'foo']
```

- Slice syntax `[4::-2]` begins with the element at index `4` (`'quux'`) and proceeds to the start of the list, skipping every other item. That yields the elements at indices `4`, `2`, and `0`.

Python

```
>>> print(a[-5:-3])
['bar', 'baz']
```

- `[-5:-3]` starts at index `-5` and goes up to but not including index `-3`, which designates items `'bar'` and `'baz'`.

Python

```
>>> max(a[2:4] + ['grault'])
'qux'
```

- `a[2:4]` returns the slice `['baz', 'qux']`. The `+` operator concatenates, so the argument to `max()` is `['baz', 'qux', 'grault']`. The maximum value (for strings, the latest in alphabetical order) is `'qux'`.

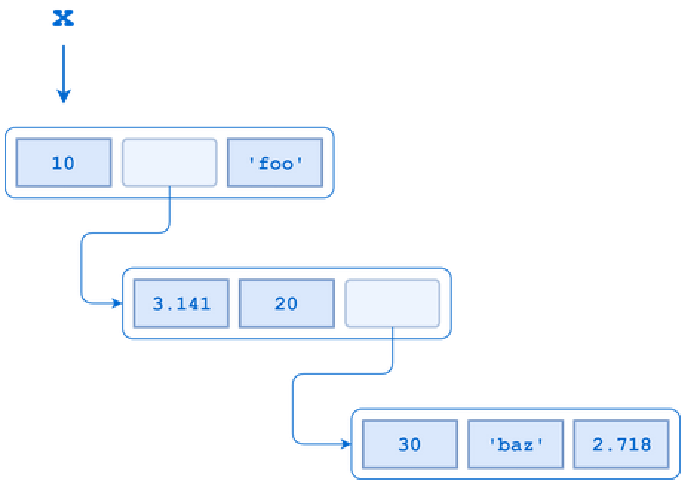
Quiz – Lists and Tuples - Q3

Consider the following nested list definition:

Python

```python
x = [10, [3.141, 20, [30, 'baz', 2.718]], 'foo']
```

A schematic for this list is shown below:



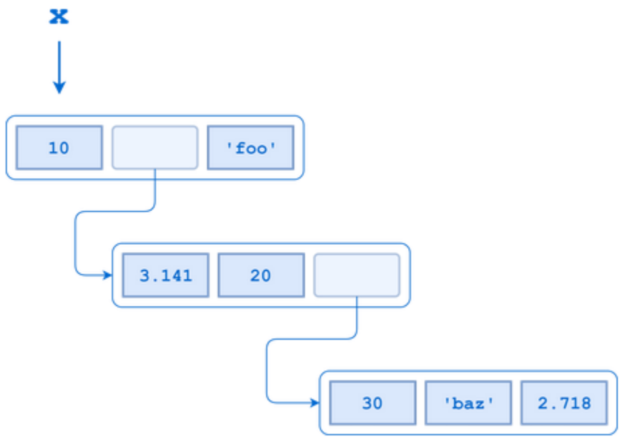
What is the expression that returns the `'z'` in `'baz'`?

Quiz – Lists and Tuples - A3

Consider the following nested list definition:

```python
x = [10, [3.141, 20, [30, 'baz', 2.718]], 'foo']
```

A schematic for this list is shown below:



What is the expression that returns the `'z'` in `'baz'`?

x[1][2][1][2] ✓

Each of the four indices in the answer can be specified as a positive or negative number:

Expression	**Selects**

`x[1]`
`x[-2]`
	The second element of `x`:
	`[3.141, 20, [30, 'baz', 2.718]]`

`x[1][2]`
`x[1][-1]`
	The third element of that sublist:
	`[30, 'baz', 2.718]`

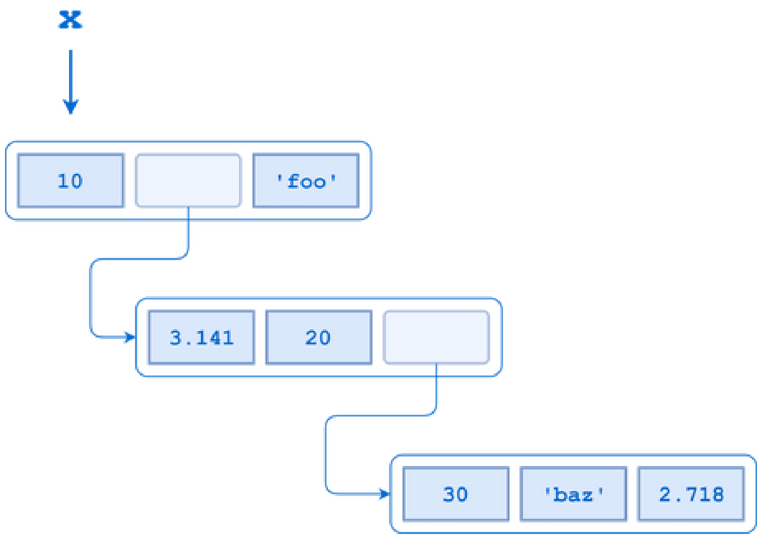
`x[1][2][1]`
`x[1][2][-2]`
	The second element of that sublist: `'baz'`

`x[1][2][1][2]`
`x[1][2][1][-1]`
	The third character of `'baz'`: `'z'`

Quiz – Lists and Tuples - Q4

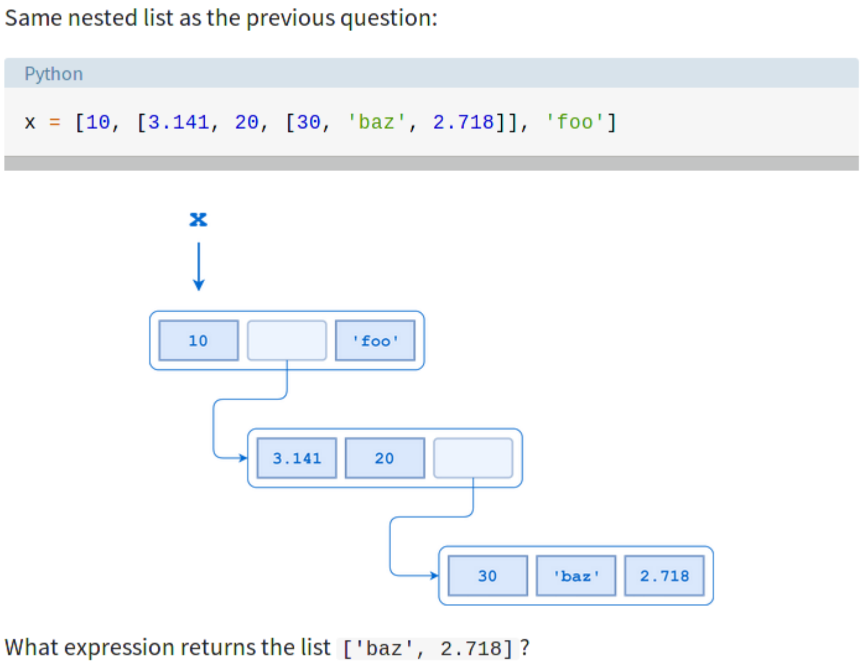
Same nested list as the previous question:

```
Python
x = [10, [3.141, 20, [30, 'baz', 2.718]], 'foo']
```



What expression returns the list `['baz', 2.718]`?

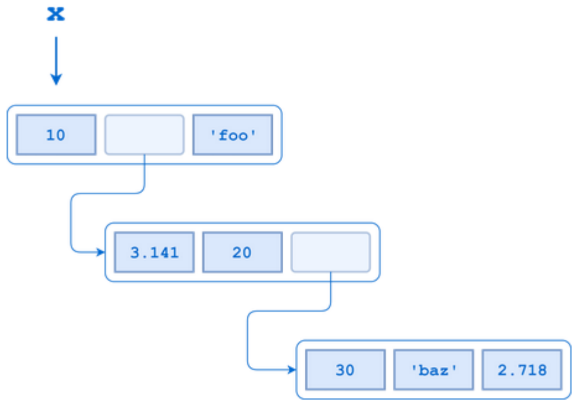
Quiz – Lists and Tuples - A4



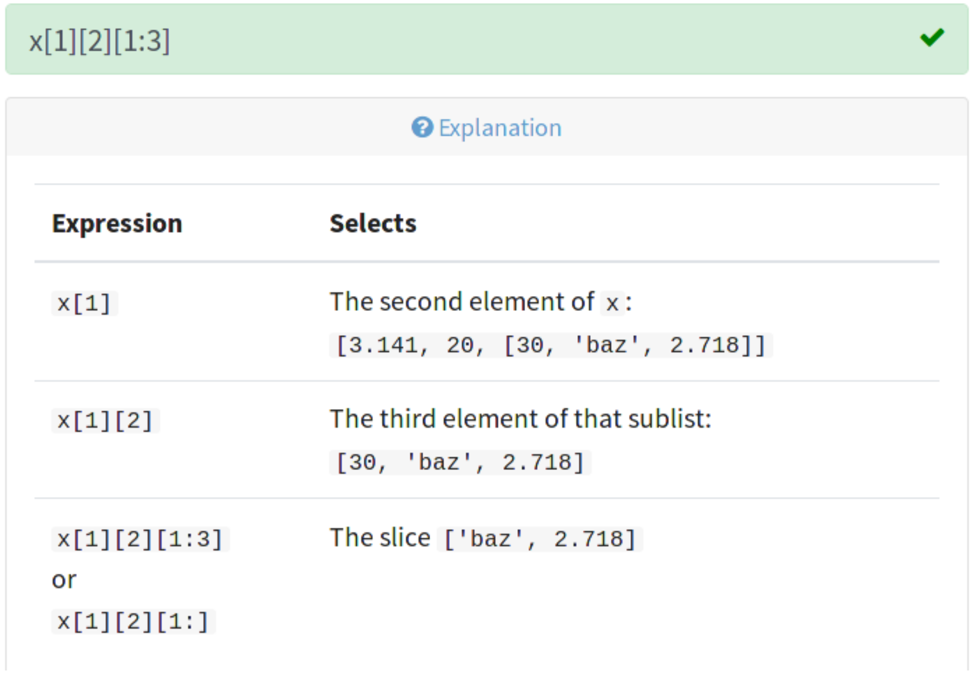
Same nested list as the previous question:

Python

```python
x = [10, [3.141, 20, [30, 'baz', 2.718]], 'foo']
```



What expression returns the list `['baz', 2.718]` ?

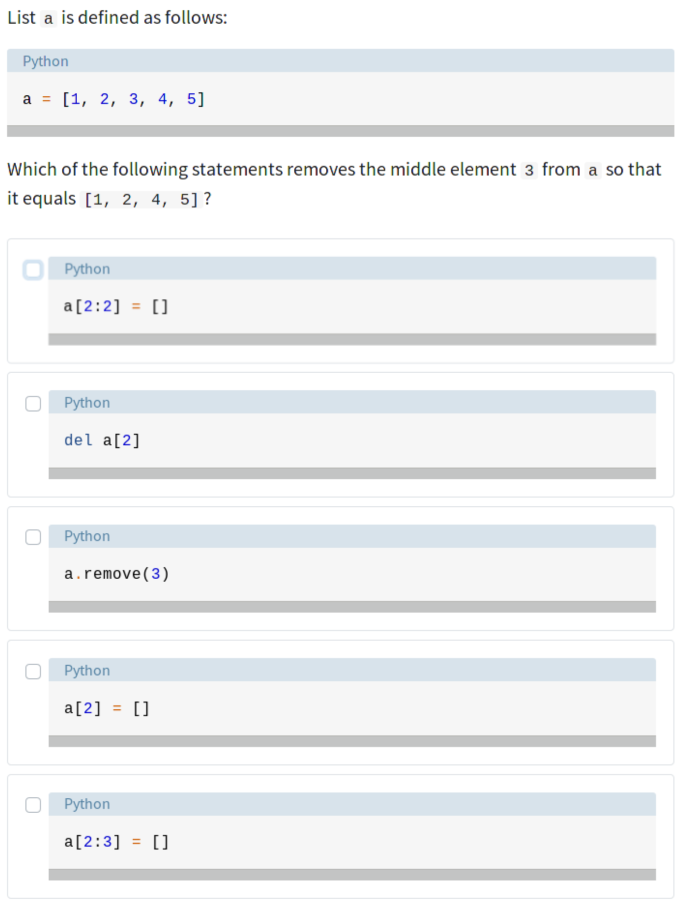


x[1][2][1:3] ✓

Explanation

Expression	Selects
`x[1]`	The second element of `x`: `[3.141, 20, [30, 'baz', 2.718]]`
`x[1][2]`	The third element of that sublist: `[30, 'baz', 2.718]`
`x[1][2][1:3]` or `x[1][2][1:]`	The slice `['baz', 2.718]`

Quiz – Lists and Tuples - Q5



List `a` is defined as follows:

Python
```python
a = [1, 2, 3, 4, 5]
```

Which of the following statements removes the middle element `3` from `a` so that it equals `[1, 2, 4, 5]`?

☐ Python
```python
a[2:2] = []
```

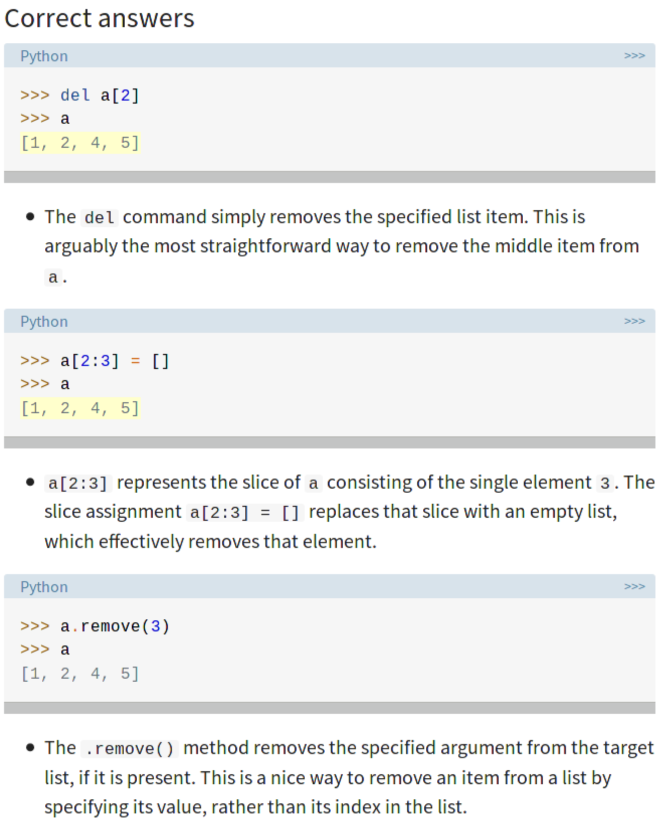
☐ Python
```python
del a[2]
```

☐ Python
```python
a.remove(3)
```

☐ Python
```python
a[2] = []
```

☐ Python
```python
a[2:3] = []
```

Quiz – Lists and Tuples - A5



Correct answers

```python
>>> del a[2]
>>> a
[1, 2, 4, 5]
```

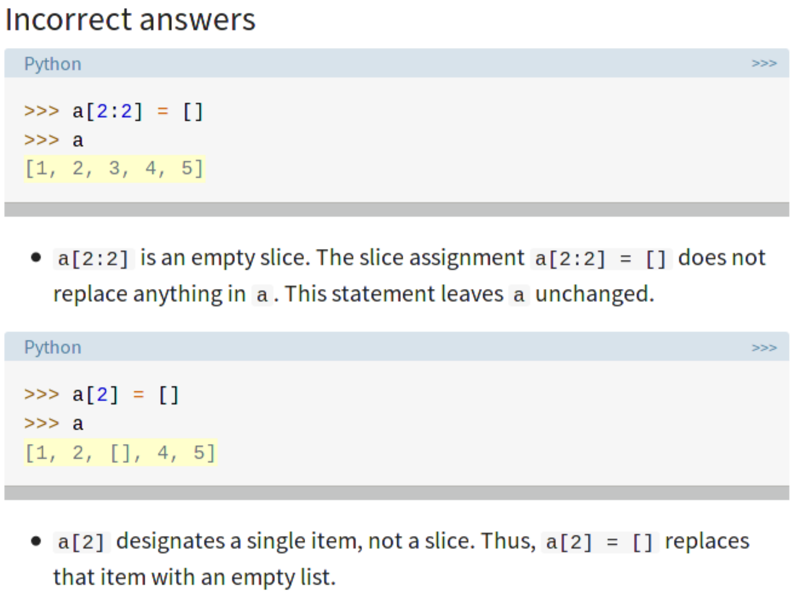
- The `del` command simply removes the specified list item. This is arguably the most straightforward way to remove the middle item from `a`.

```python
>>> a[2:3] = []
>>> a
[1, 2, 4, 5]
```

- `a[2:3]` represents the slice of `a` consisting of the single element `3`. The slice assignment `a[2:3] = []` replaces that slice with an empty list, which effectively removes that element.

```python
>>> a.remove(3)
>>> a
[1, 2, 4, 5]
```

- The `.remove()` method removes the specified argument from the target list, if it is present. This is a nice way to remove an item from a list by specifying its value, rather than its index in the list.



Incorrect answers

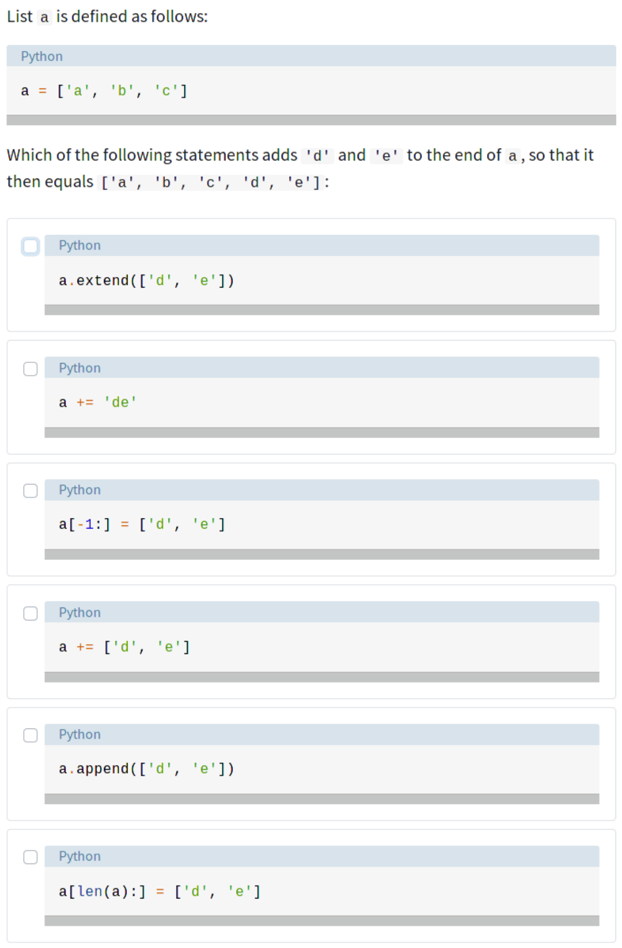
```python
>>> a[2:2] = []
>>> a
[1, 2, 3, 4, 5]
```

- `a[2:2]` is an empty slice. The slice assignment `a[2:2] = []` does not replace anything in `a`. This statement leaves `a` unchanged.

```python
>>> a[2] = []
>>> a
[1, 2, [], 4, 5]
```

- `a[2]` designates a single item, not a slice. Thus, `a[2] = []` replaces that item with an empty list.

Quiz – Lists and Tuples - Q6



List `a` is defined as follows:

Python
```python
a = ['a', 'b', 'c']
```

Which of the following statements adds `'d'` and `'e'` to the end of `a`, so that it then equals `['a', 'b', 'c', 'd', 'e']`:

☐ Python
```python
a.extend(['d', 'e'])
```

☐ Python
```python
a += 'de'
```

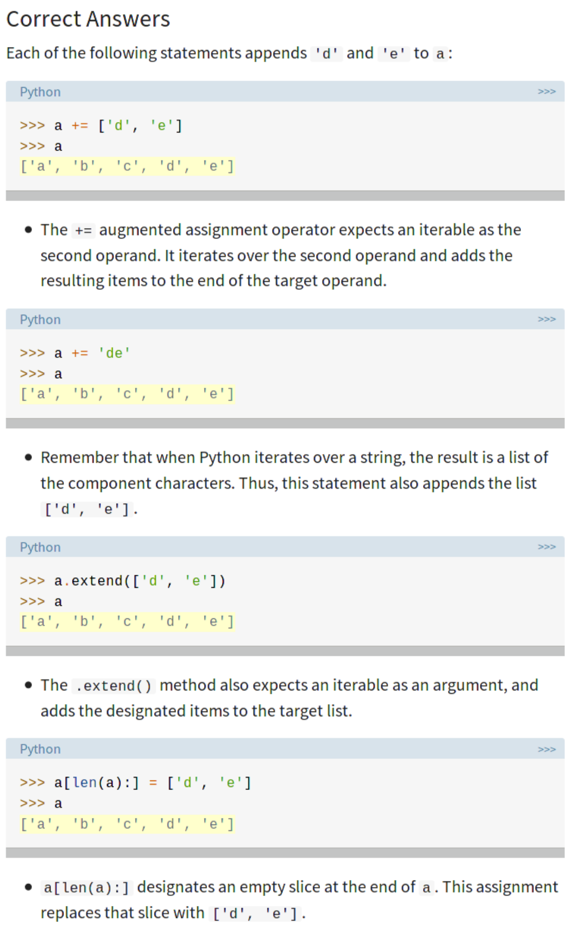
☐ Python
```python
a[-1:] = ['d', 'e']
```

☐ Python
```python
a += ['d', 'e']
```

☐ Python
```python
a.append(['d', 'e'])
```

☐ Python
```python
a[len(a):] = ['d', 'e']
```

Quiz – Lists and Tuples - A6



Correct Answers

Each of the following statements appends `'d'` and `'e'` to `a`:

```
>>> a += ['d', 'e']
>>> a
['a', 'b', 'c', 'd', 'e']
```

- The `+=` augmented assignment operator expects an iterable as the second operand. It iterates over the second operand and adds the resulting items to the end of the target operand.

```
>>> a += 'de'
>>> a
['a', 'b', 'c', 'd', 'e']
```

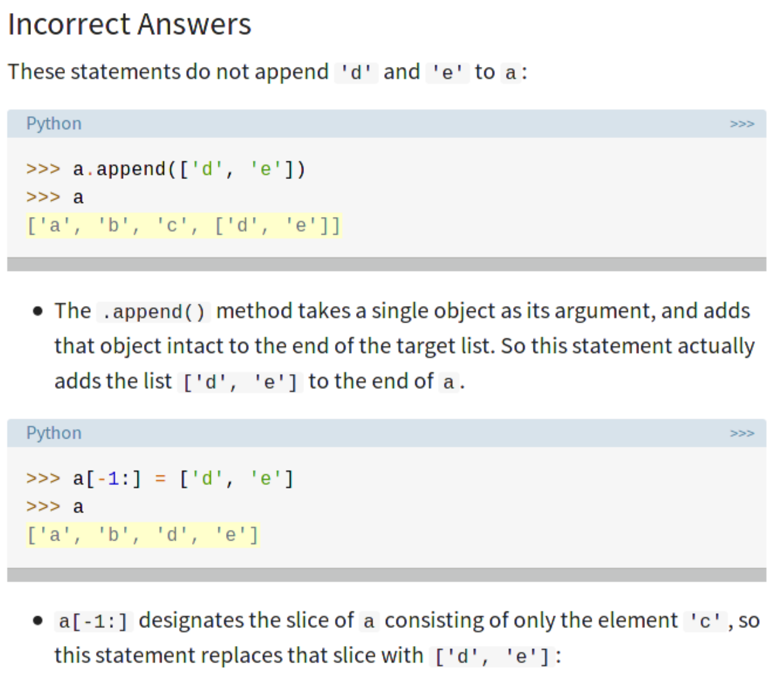
- Remember that when Python iterates over a string, the result is a list of the component characters. Thus, this statement also appends the list `['d', 'e']`.

```
>>> a.extend(['d', 'e'])
>>> a
['a', 'b', 'c', 'd', 'e']
```

- The `.extend()` method also expects an iterable as an argument, and adds the designated items to the target list.

```
>>> a[len(a):] = ['d', 'e']
>>> a
['a', 'b', 'c', 'd', 'e']
```

- `a[len(a):]` designates an empty slice at the end of `a`. This assignment replaces that slice with `['d', 'e']`.



Incorrect Answers

These statements do not append `'d'` and `'e'` to `a`:

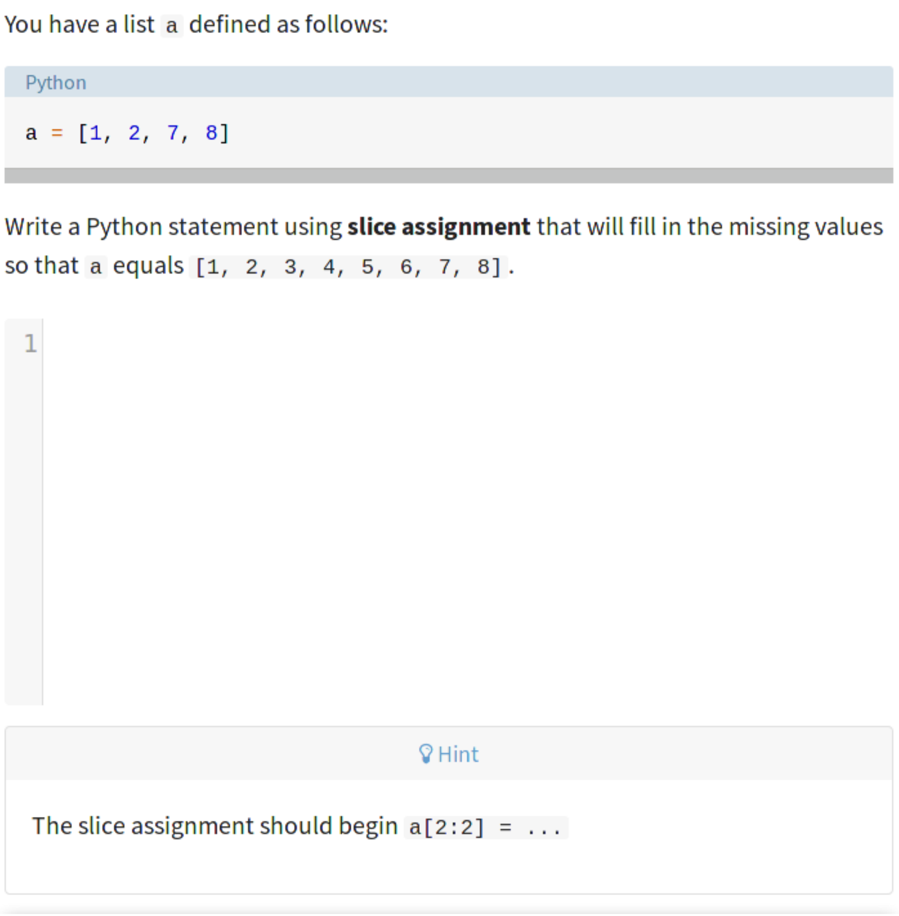
```
>>> a.append(['d', 'e'])
>>> a
['a', 'b', 'c', ['d', 'e']]
```

- The `.append()` method takes a single object as its argument, and adds that object intact to the end of the target list. So this statement actually adds the list `['d', 'e']` to the end of `a`.

```
>>> a[-1:] = ['d', 'e']
>>> a
['a', 'b', 'd', 'e']
```

- `a[-1:]` designates the slice of `a` consisting of only the element `'c'`, so this statement replaces that slice with `['d', 'e']`:

Quiz – Lists and Tuples - Q7



You have a list `a` defined as follows:

Python

```python
a = [1, 2, 7, 8]
```

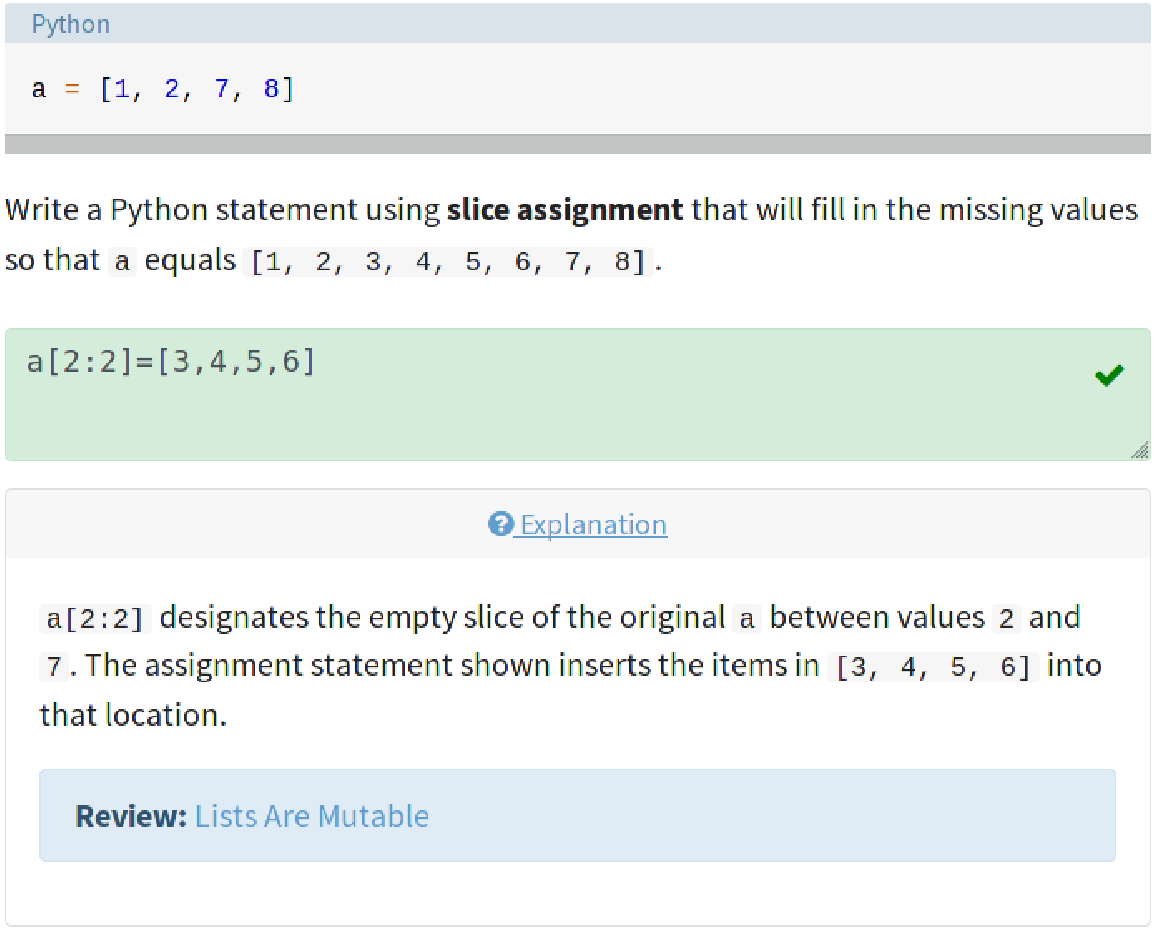
Write a Python statement using **slice assignment** that will fill in the missing values so that `a` equals `[1, 2, 3, 4, 5, 6, 7, 8]`.

```
1
```

Hint

The slice assignment should begin `a[2:2] = ...`

Quiz – Lists and Tuples - A7



Python

```python
a = [1, 2, 7, 8]
```

Write a Python statement using **slice assignment** that will fill in the missing values so that `a` equals `[1, 2, 3, 4, 5, 6, 7, 8]`.

```
a[2:2]=[3,4,5,6]
```

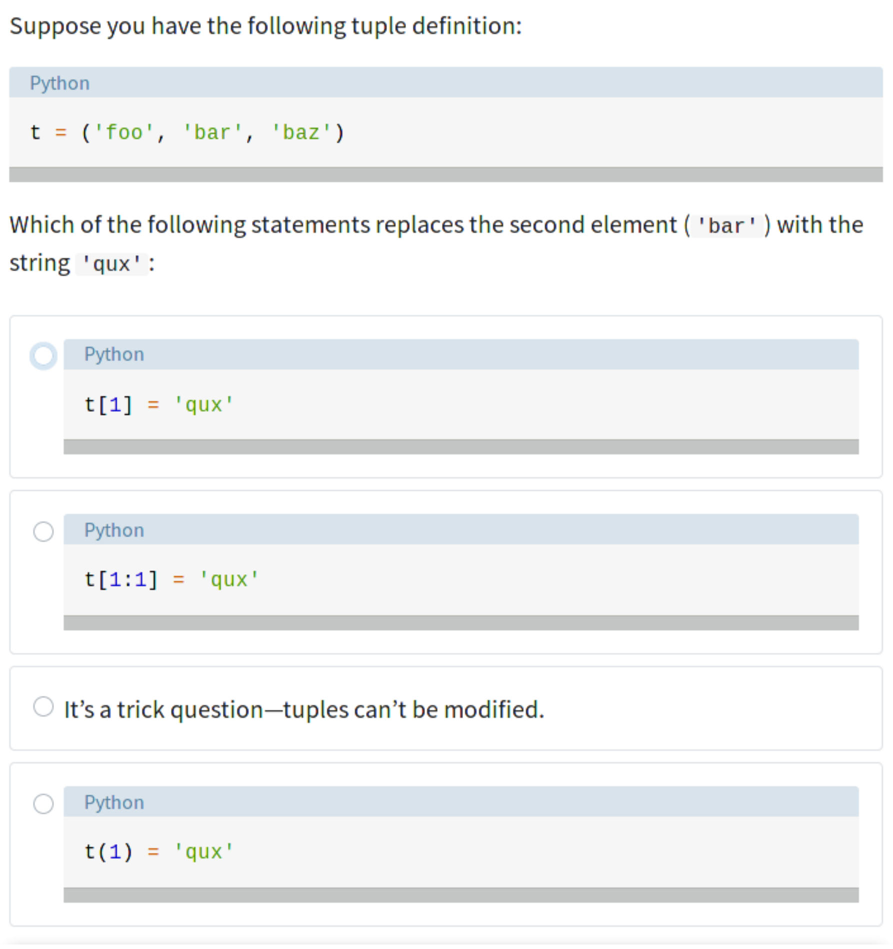
✓

Explanation

`a[2:2]` designates the empty slice of the original `a` between values `2` and `7`. The assignment statement shown inserts the items in `[3, 4, 5, 6]` into that location.

Review: Lists Are Mutable

Quiz – Lists and Tuples - Q8



Suppose you have the following tuple definition:

Python

```python
t = ('foo', 'bar', 'baz')
```

Which of the following statements replaces the second element (`'bar'`) with the string `'qux'`:

- Python

  ```python
  t[1] = 'qux'
  ```

- Python

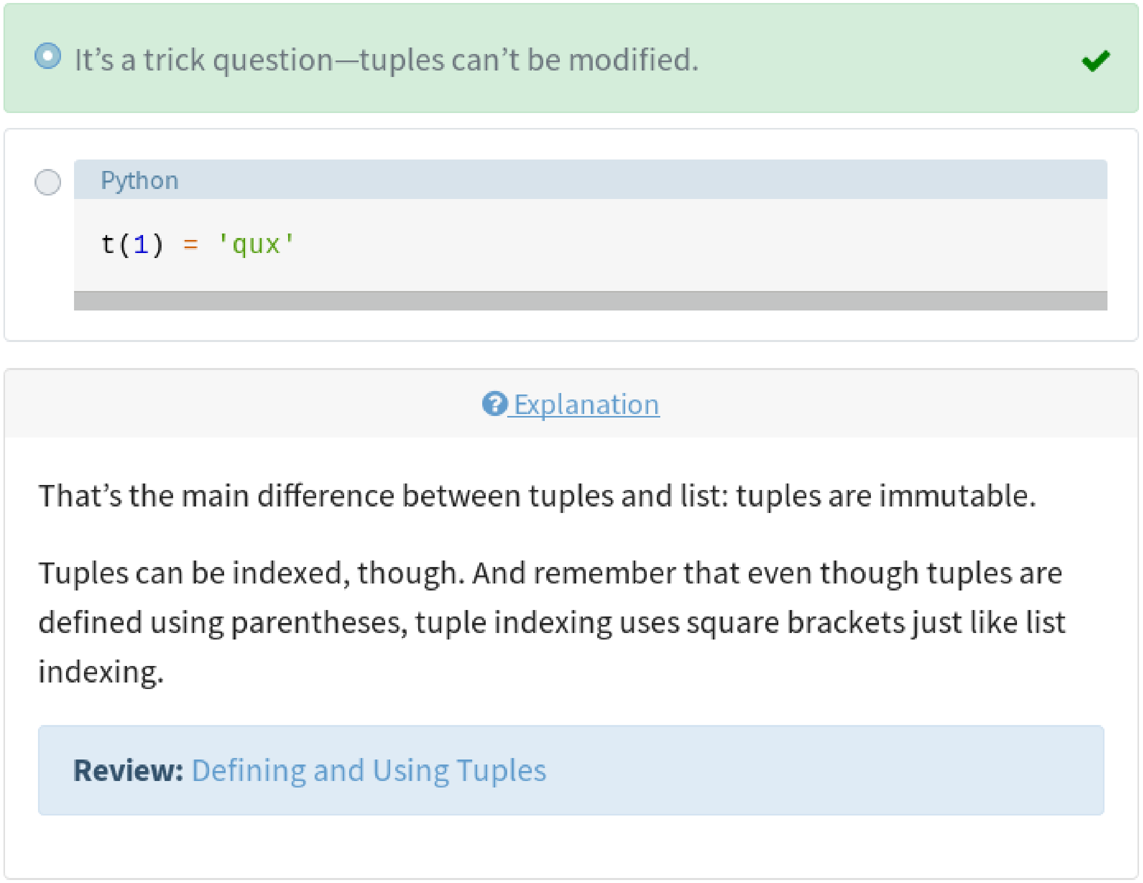
  ```python
  t[1:1] = 'qux'
  ```

- It’s a trick question—tuples can’t be modified.

- Python

  ```python
  t(1) = 'qux'
  ```

Quiz – Lists and Tuples - A8



[✓ It's a trick question—tuples can't be modified.] ✓

Python

```python
t(1) = 'qux'
```

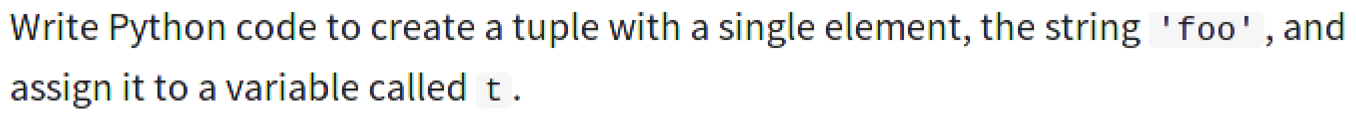
Explanation

That's the main difference between tuples and list: tuples are immutable.

Tuples can be indexed, though. And remember that even though tuples are defined using parentheses, tuple indexing uses square brackets just like list indexing.

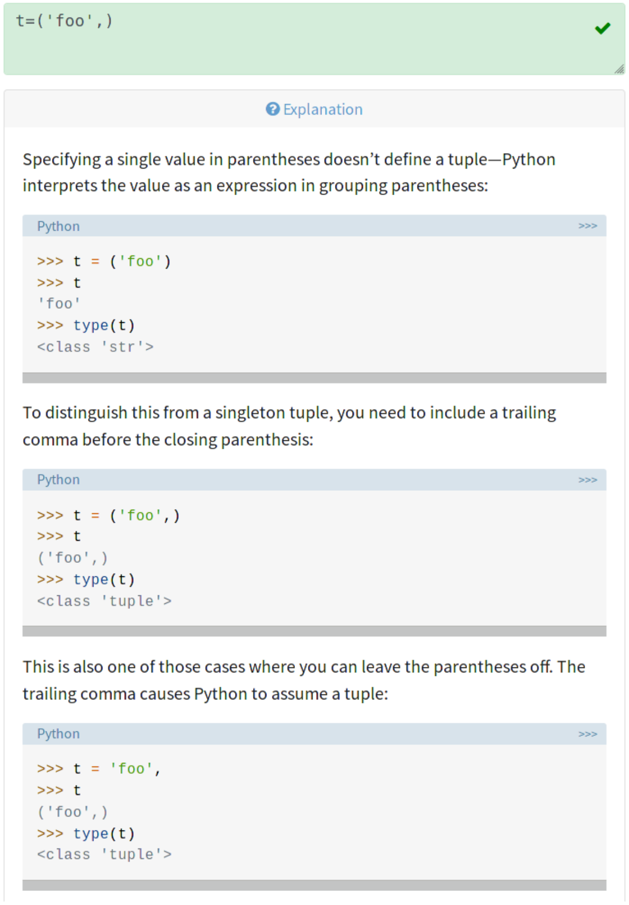
Review: Defining and Using Tuples

Quiz – Lists and Tuples - Q9



Write Python code to create a tuple with a single element, the string `'foo'`, and assign it to a variable called `t`.

Quiz – Lists and Tuples - A9



```
t=('foo',)
```

✓

Explanation

Specifying a single value in parentheses doesn’t define a tuple—Python interprets the value as an expression in grouping parentheses:

Python

```
>>> t = ('foo')
>>> t
'foo'
>>> type(t)
<class 'str'>
```

To distinguish this from a singleton tuple, you need to include a trailing comma before the closing parenthesis:

Python

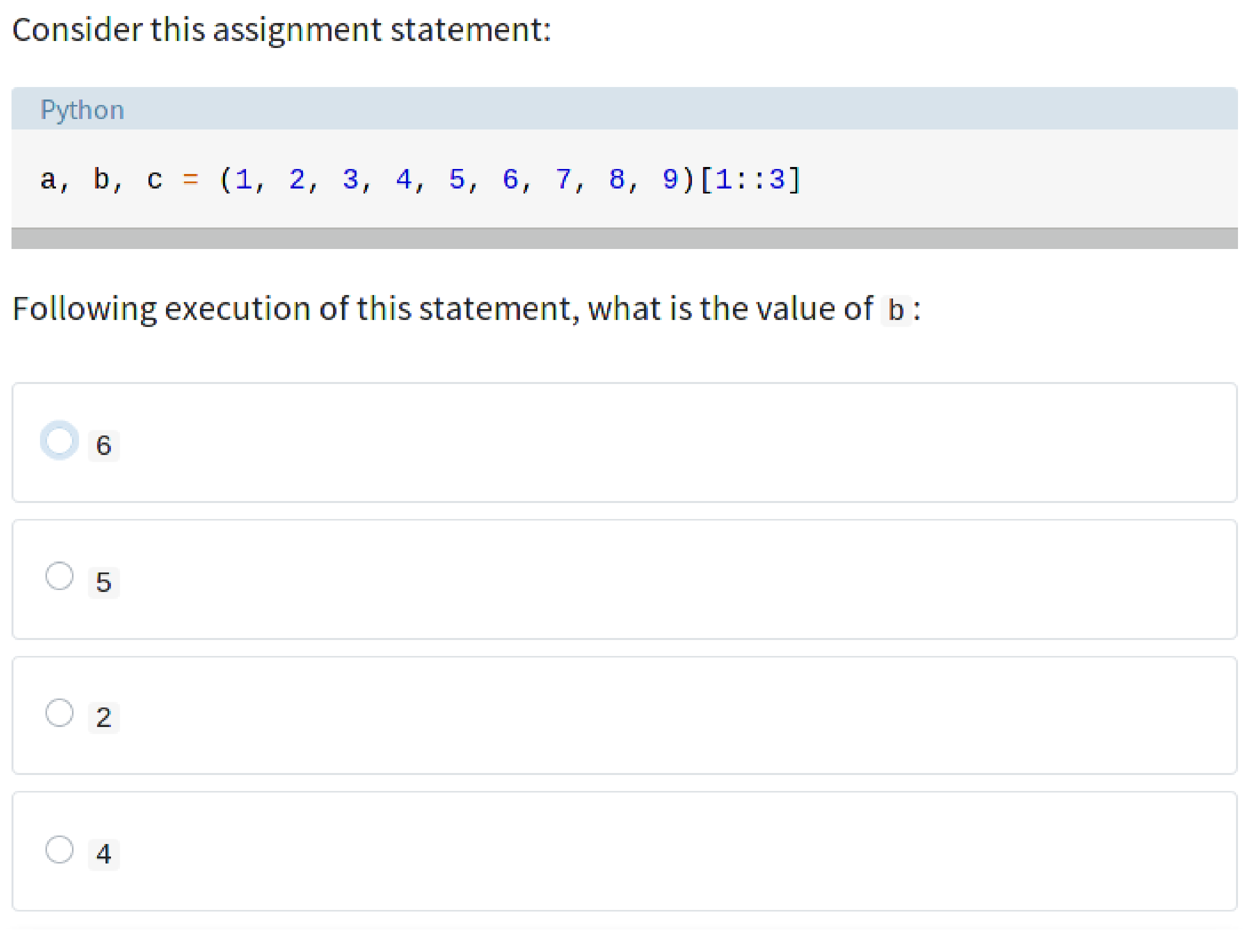
```
>>> t = ('foo',)
>>> t
('foo',)
>>> type(t)
<class 'tuple'>
```

This is also one of those cases where you can leave the parentheses off. The trailing comma causes Python to assume a tuple:

Python

```
>>> t = 'foo',
>>> t
('foo',)
>>> type(t)
<class 'tuple'>
```

Quiz – Lists and Tuples - Q10



Consider this assignment statement:

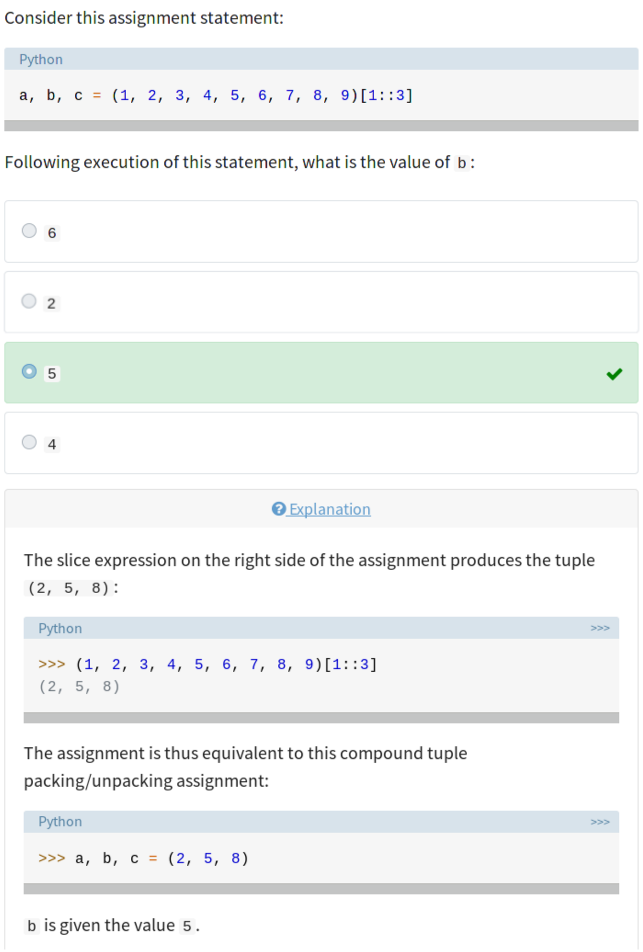
Python

```python
a, b, c = (1, 2, 3, 4, 5, 6, 7, 8, 9)[1::3]
```

Following execution of this statement, what is the value of `b`:

- ○ 6
- ○ 5
- ○ 2
- ○ 4

Quiz – Lists and Tuples - A10



Consider this assignment statement:

Python
```
a, b, c = (1, 2, 3, 4, 5, 6, 7, 8, 9)[1::3]
```

Following execution of this statement, what is the value of `b`:

- ( ) 6
- ( ) 2
- [✓ 5] ✓
- ( ) 4

Explanation

The slice expression on the right side of the assignment produces the tuple `(2, 5, 8)`:

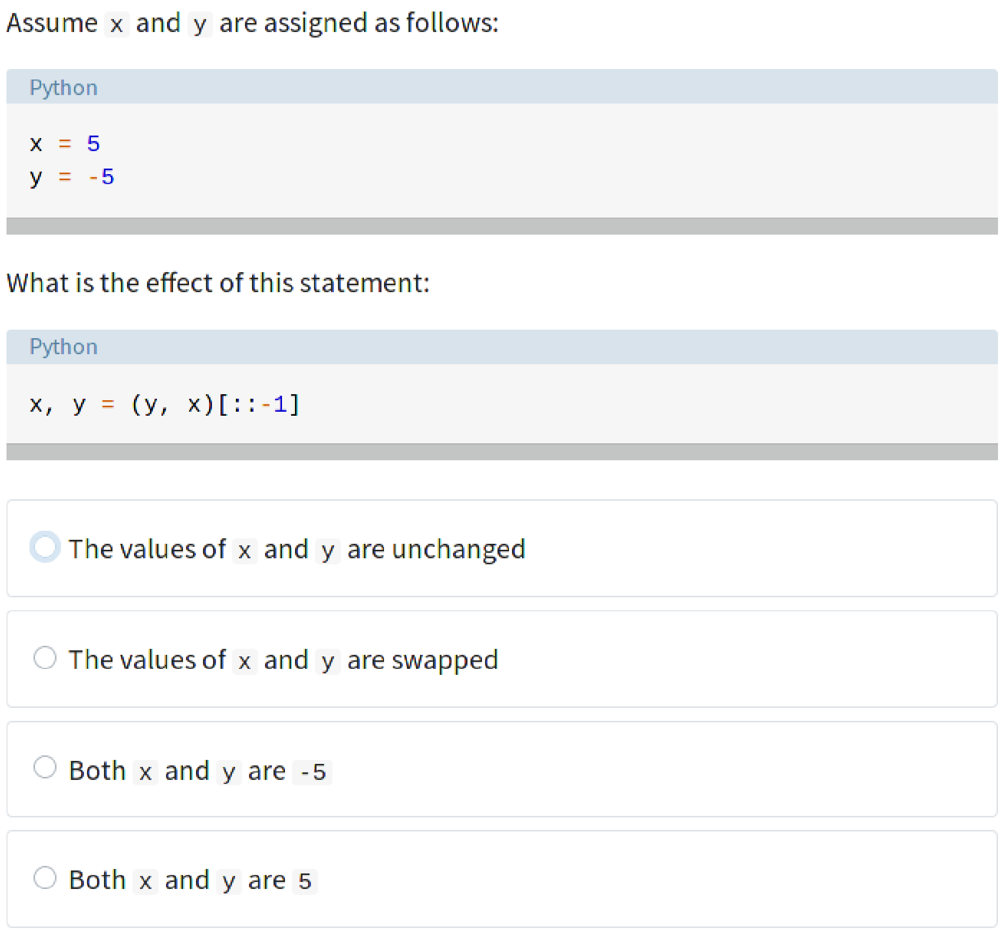
Python >>>
```
>>> (1, 2, 3, 4, 5, 6, 7, 8, 9)[1::3]
(2, 5, 8)
```

The assignment is thus equivalent to this compound tuple packing/unpacking assignment:

Python >>>
```
>>> a, b, c = (2, 5, 8)
```

`b` is given the value `5`.

Quiz – Lists and Tuples - Q11



Assume `x` and `y` are assigned as follows:

Python

```python
x = 5
y = -5
```

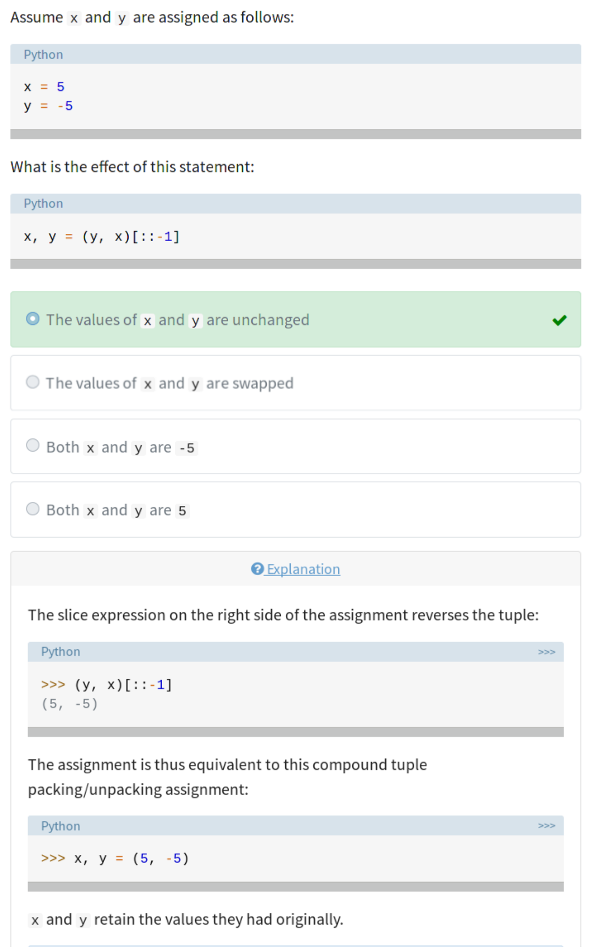
What is the effect of this statement:

Python

```python
x, y = (y, x)[::-1]
```

- ○ The values of `x` and `y` are unchanged
- ○ The values of `x` and `y` are swapped
- ○ Both `x` and `y` are `-5`
- ○ Both `x` and `y` are `5`

Quiz – Lists and Tuples - A11



Assume `x` and `y` are assigned as follows:

Python

```python
x = 5
y = -5
```

What is the effect of this statement:

Python

```python
x, y = (y, x)[::-1]
```

[✓ The values of `x` and `y` are unchanged] ✓

The values of `x` and `y` are swapped

Both `x` and `y` are `-5`

Both `x` and `y` are `5`

Explanation

The slice expression on the right side of the assignment reverses the tuple:

Python >>>

```python
>>> (y, x)[::-1]
(5, -5)
```

The assignment is thus equivalent to this compound tuple packing/unpacking assignment:

Python >>>

```python
>>> x, y = (5, -5)
```

`x` and `y` retain the values they had originally.