Lecture 10: Mutability

CS 61A - Summer 2024

Charlotte Le

Announcements

Contents

1) Objects

Objects

```
>>> from datetime import date
>>> date
<class 'datetime.date'>
>>> today = date(2024, 7, 3)
>>> today
datetime.date(2024, 7, 3)
>>> freedom = date(2024, 8, 8)
>>> str(freedom - today)
'36 days, 0:00:00'
>>> today.year
2024
>>> today.month
7
```

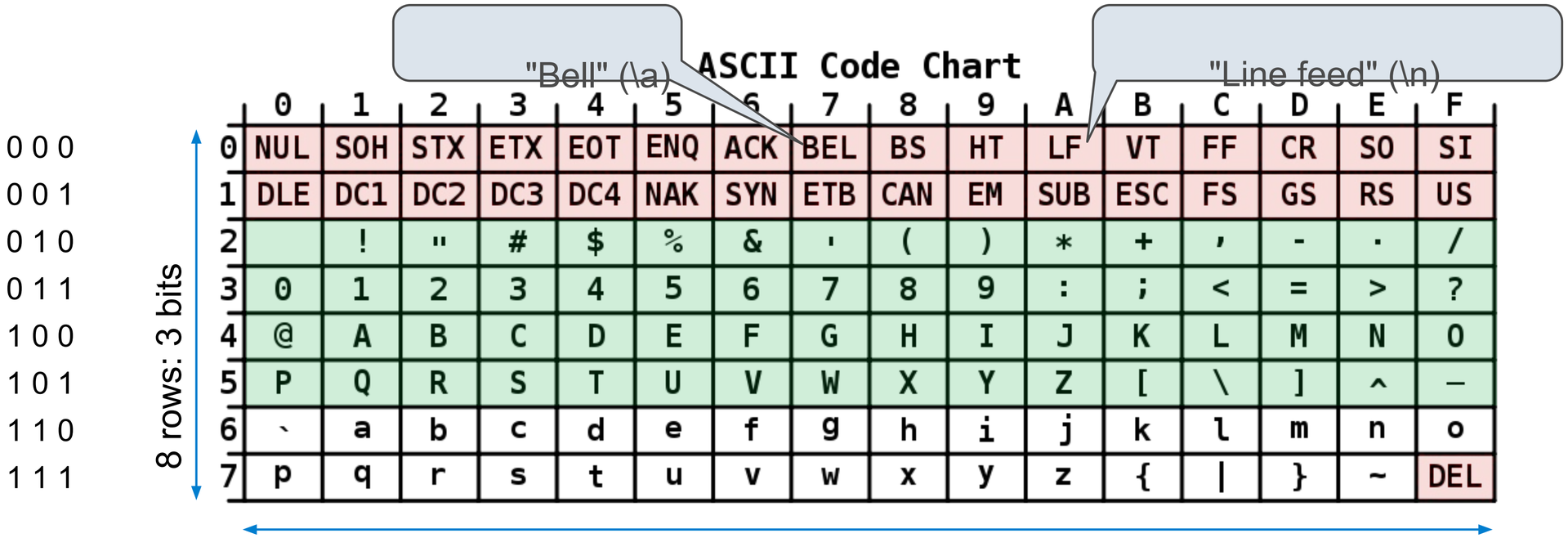
```
>>> s = 'Hello'
>>> s.upper()
'HELLO'
>>> s.lower()
'hello'
>>> s.swapcase()
'hELLO'
```

Objects

- Objects represent information
- They consist of data and behavior bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; **classes** are first-class values in Python
- Object-oriented programming:
    - A metaphor for organizing large programs
    - Special syntax that can improve the composition of programs
- In Python, every value is an object
    - All **objects** have **attributes**
    - A lot of data manipulation happens through object **methods**
    - Functions do one thing; objects do many related things

Representing Strings: the ASCII Standard

American Standard Code for Information Interchange



- Layout was chosen to support sorting by character code
- Rows indexed 2-5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission

(Demo)

Representing Strings: the Unicode Standard

- 137,994 characters in Unicode 12.1
- 150 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

LATIN CAPITAL LETTER A

DIE FACE-6

EIGHTH NOTE

'⚅' '♪'

(Demo)

2) Mutation Operators

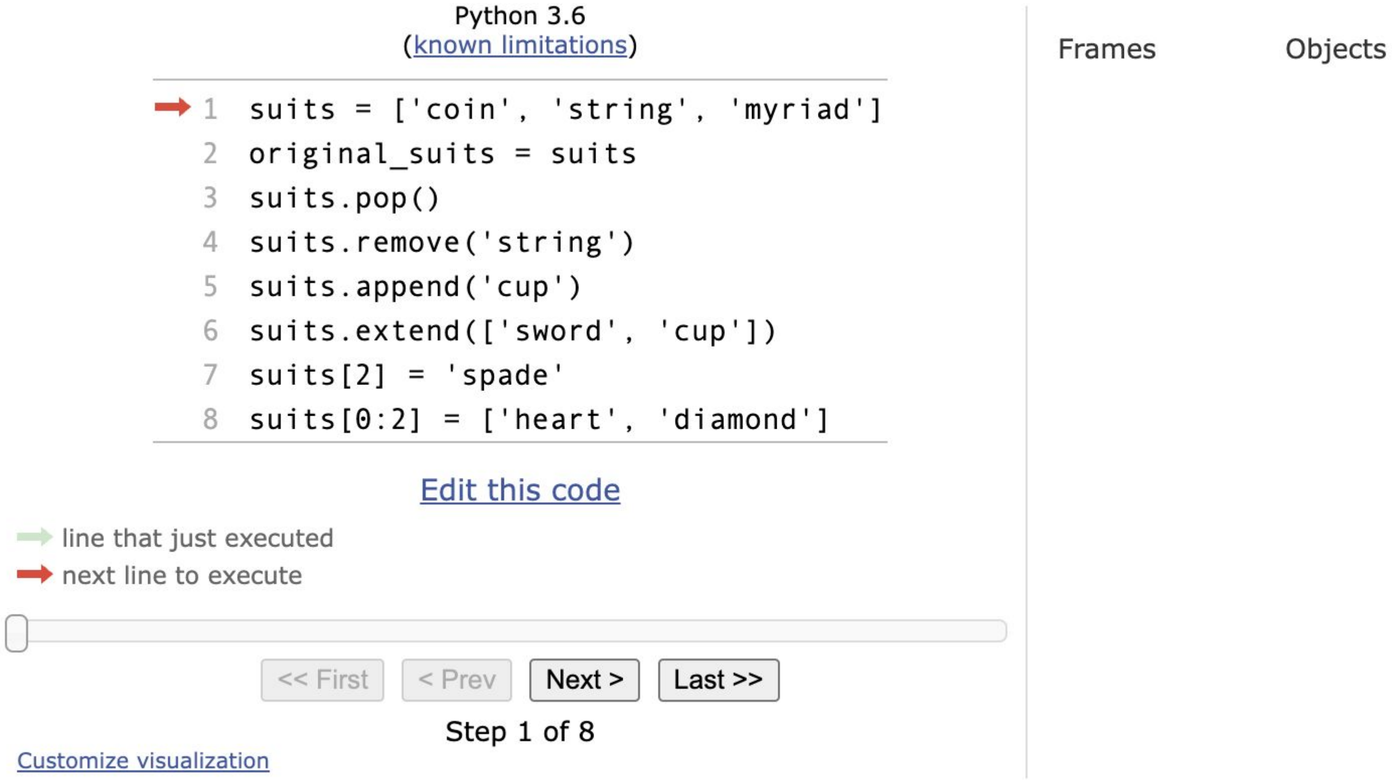
List Methods

- `append(el)`
  - Add `el` to the end of the list
  - Return `None`
- `extend(lst)`
  - Extend the list by concatenating it with `lst`
  - Return `None`.
- `insert(i, el)`
  - Insert `el` at index `i`. This does not replace any existing elements, but only adds the new element `el`.
  - Return `None`.
- `remove(el)`
  - Remove the first occurrence of `el` in list. Errors if `el` is not in the list.
  - Return None otherwise.
- `pop(i)`
  - Remove and return the element at index `i`

Card Example

```
>>> suits = ['coin', 'string', 'myriad']
>>> original_suits = suits
>>> suits.pop()
'myriad'
>>> suits.remove('string')
>>> suits
['coin']
>>> suits.append('cup')
>>> suits.extend(['sword', 'cup'])
>>> suits
['coin', 'cup', 'sword', 'cup']
>>> suits[2] = 'spade'
>>> suits
['coin', 'cup', 'spade', 'cup']
>>> suits[0:2] = ['heart', 'diamond']
>>> suits
['heart', 'diamond', 'spade', 'cup']
>>> original_suits
['heart', 'diamond', 'spade', 'cup']
```

Card Example



Card Example

Python 3.6
(known limitations)

```
1  suits = ['coin', 'string', 'myriad']
2  original_suits = suits
3  suits.pop()
4  suits.remove('string')
5  suits.append('cup')
6  suits.extend(['sword', 'cup'])
7  suits[2] = 'spade'
8  suits[0:2] = ['heart', 'diamond']
```

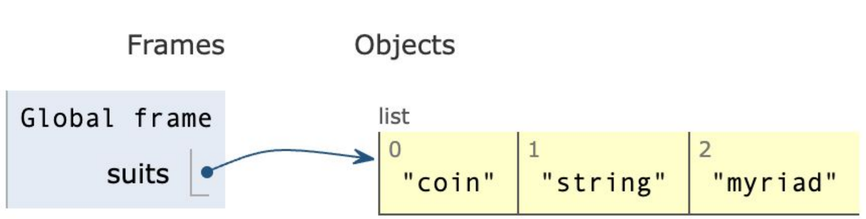
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 2 of 8

Customize visualization



Card Example

Python 3.6
(known limitations)

```
1  suits = ['coin', 'string', 'myriad']
2  original_suits = suits
3  suits.pop()
4  suits.remove('string')
5  suits.append('cup')
6  suits.extend(['sword', 'cup'])
7  suits[2] = 'spade'
8  suits[0:2] = ['heart', 'diamond']
```

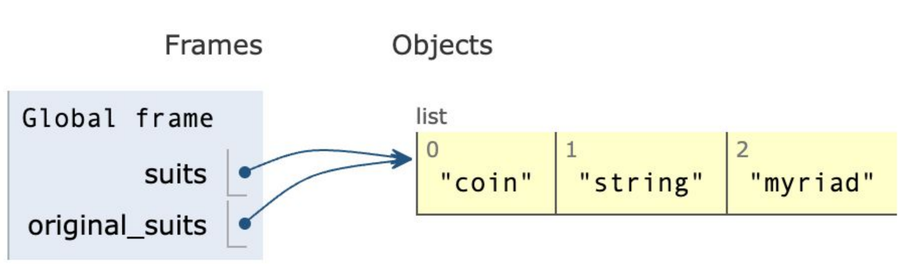
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 3 of 8

Customize visualization



Card Example

Python 3.6
(known limitations)

```
1  suits = ['coin', 'string', 'myriad']
2  original_suits = suits
3  suits.pop()
4  suits.remove('string')
5  suits.append('cup')
6  suits.extend(['sword', 'cup'])
7  suits[2] = 'spade'
8  suits[0:2] = ['heart', 'diamond']
```

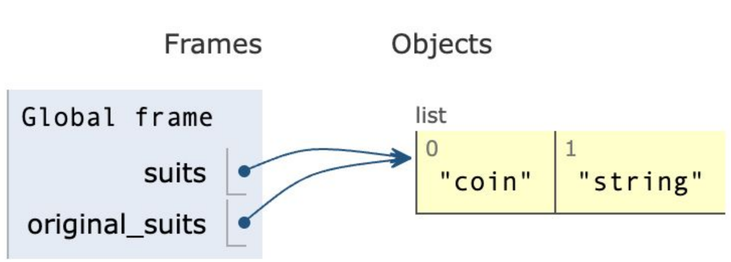
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 4 of 8

Customize visualization



Card Example

Python 3.6
(known limitations)

```
1  suits = ['coin', 'string', 'myriad']
2  original_suits = suits
3  suits.pop()
4  suits.remove('string')
5  suits.append('cup')
6  suits.extend(['sword', 'cup'])
7  suits[2] = 'spade'
8  suits[0:2] = ['heart', 'diamond']
```

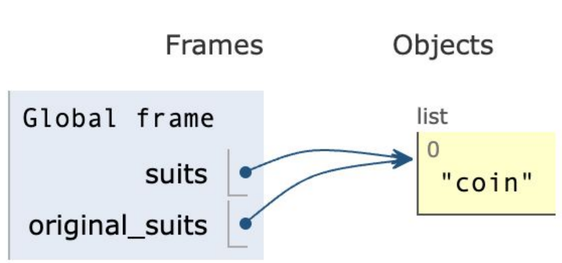
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 5 of 8

Customize visualization



Card Example

Python 3.6
(known limitations)

```
1  suits = ['coin', 'string', 'myriad']
2  original_suits = suits
3  suits.pop()
4  suits.remove('string')
5  suits.append('cup')
6  suits.extend(['sword', 'cup'])
7  suits[2] = 'spade'
8  suits[0:2] = ['heart', 'diamond']
```

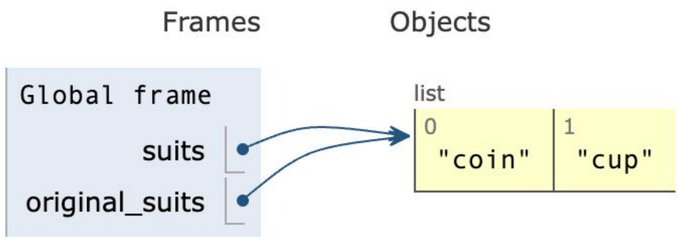
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 6 of 8

Customize visualization



Card Example

Python 3.6
(known limitations)

```
1  suits = ['coin', 'string', 'myriad']
2  original_suits = suits
3  suits.pop()
4  suits.remove('string')
5  suits.append('cup')
6  suits.extend(['sword', 'cup'])
7  suits[2] = 'spade'
8  suits[0:2] = ['heart', 'diamond']
```

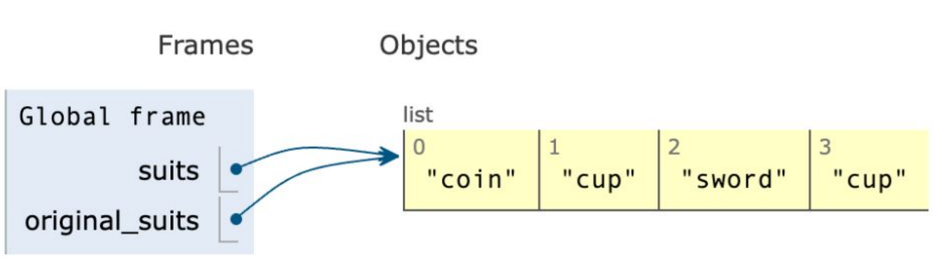
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 7 of 8

Customize visualization



Card Example

Python 3.6
(known limitations)

```
1  suits = ['coin', 'string', 'myriad']
2  original_suits = suits
3  suits.pop()
4  suits.remove('string')
5  suits.append('cup')
6  suits.extend(['sword', 'cup'])
7  suits[2] = 'spade'
8  suits[0:2] = ['heart', 'diamond']
```

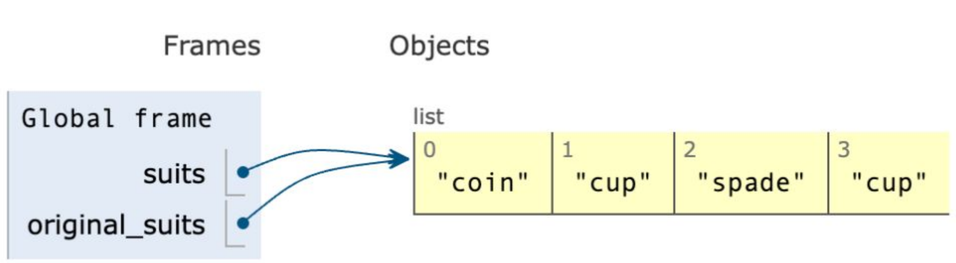
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 8 of 8

Customize visualization



Card Example

Python 3.6
(known limitations)

```
1  suits = ['coin', 'string', 'myriad']
2  original_suits = suits
3  suits.pop()
4  suits.remove('string')
5  suits.append('cup')
6  suits.extend(['sword', 'cup'])
7  suits[2] = 'spade'
8  suits[0:2] = ['heart', 'diamond']
```

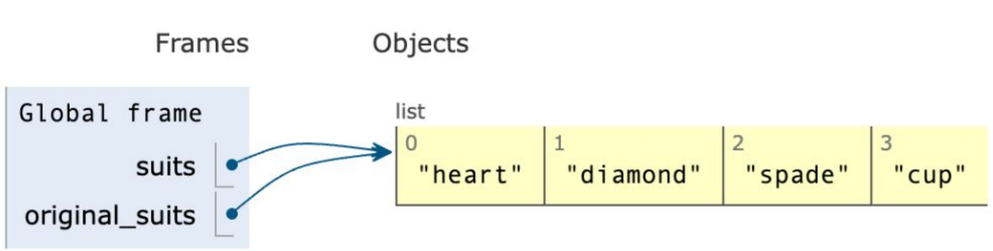
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Done running (8 steps)

Customize visualization

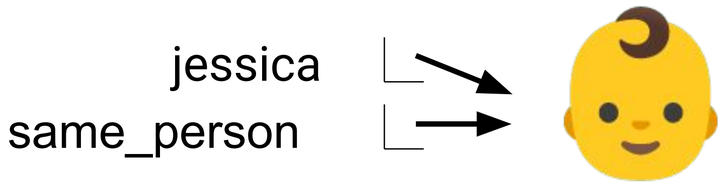


Break

3) Mutable Objects & Scope

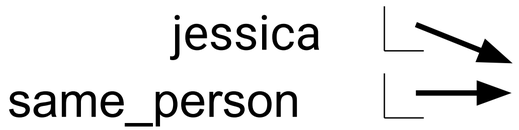
Some Objects Can Change

- First example in the course of an object changing state
- The same object can change in value throughout the course of computation
- All names that refer to the same object are affected by a mutation
- Only objects of mutable types can change
  - **Mutable: lists & dictionaries**
  - **Immutable: strings, tuples, numeric types, etc.**



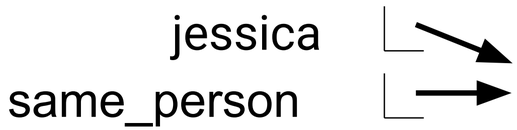
Some Objects Can Change

- First example in the course of an object changing state
- The same object can change in value throughout the course of computation
- All names that refer to the same object are affected by a mutation
- Only objects of mutable types can change
  - **Mutable: lists & dictionaries**
  - **Immutable: strings, tuples, numeric types, etc.**



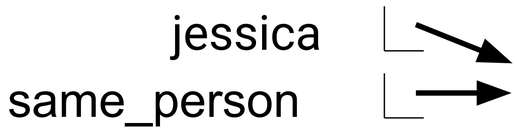
Some Objects Can Change

- First example in the course of an object changing state
- The same object can change in value throughout the course of computation
- All names that refer to the same object are affected by a mutation
- Only objects of mutable types can change
  - **Mutable: lists & dictionaries**
  - **Immutable: strings, tuples, numeric types, etc.**



Some Objects Can Change

- First example in the course of an object changing state
- The same object can change in value throughout the course of computation
- All names that refer to the same object are affected by a mutation
- Only objects of mutable types can change
  - **Mutable: lists & dictionaries**
  - **Immutable: strings, tuples, numeric types, etc.**



Numerals Example

Python 3.6
(known limitations)

```
1  numerals = {'I': 1, 'V': 5, 'X': 10}
2  print(numerals['X'])
3  numerals['X'] = 11
4  print(numerals['X'])
5  numerals['L'] = 50
6  numerals.pop('X')
```

Edit this code

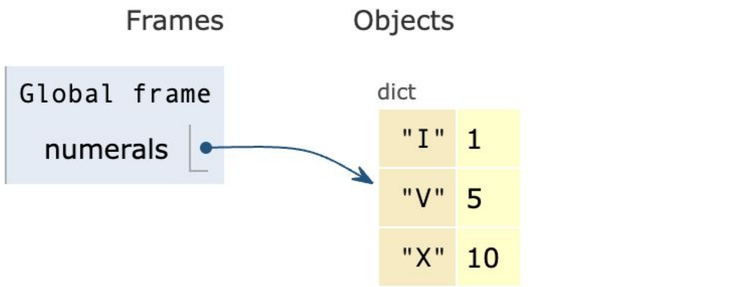
line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 2 of 6

Customize visualization

Print output (drag lower right corner to resize)



Numerals Example

Python 3.6
(known limitations)

```
1  numerals = {'I': 1, 'V': 5, 'X': 10}
2  print(numerals['X'])
3  numerals['X'] = 11
4  print(numerals['X'])
5  numerals['L'] = 50
6  numerals.pop('X')
```

Edit this code

line that just executed
next line to execute

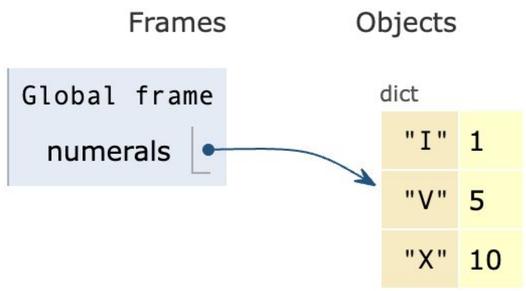
<< First | < Prev | Next > | Last >>

Step 3 of 6

Customize visualization

Print output (drag lower right corner to resize)

```
10
```



Numerals Example

Python 3.6
(known limitations)

```
1  numerals = {'I': 1, 'V': 5, 'X': 10}
2  print(numerals['X'])
3  numerals['X'] = 11
4  print(numerals['X'])
5  numerals['L'] = 50
6  numerals.pop('X')
```

Edit this code

line that just executed
next line to execute

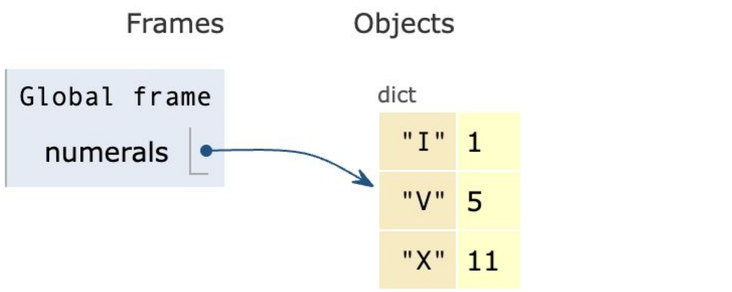
<< First | < Prev | Next > | Last >>

Step 4 of 6

Customize visualization

Print output (drag lower right corner to resize)

```
10
```



Numerals Example

Python 3.6
(known limitations)

```
1  numerals = {'I': 1, 'V': 5, 'X': 10}
2  print(numerals['X'])
3  numerals['X'] = 11
4  print(numerals['X'])
5  numerals['L'] = 50
6  numerals.pop('X')
```

Edit this code

line that just executed
next line to execute

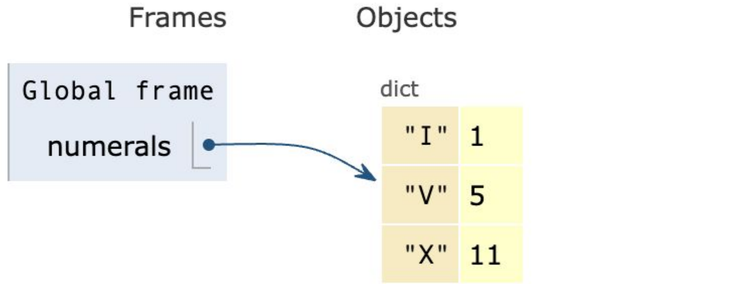
<< First | < Prev | Next > | Last >>

Step 5 of 6

Customize visualization

Print output (drag lower right corner to resize)

```
10
11
```



Numerals Example

Python 3.6
(known limitations)

```
1  numerals = {'I': 1, 'V': 5, 'X': 10}
2  print(numerals['X'])
3  numerals['X'] = 11
4  print(numerals['X'])
5  numerals['L'] = 50
6  numerals.pop('X')
```

Edit this code

line that just executed
next line to execute

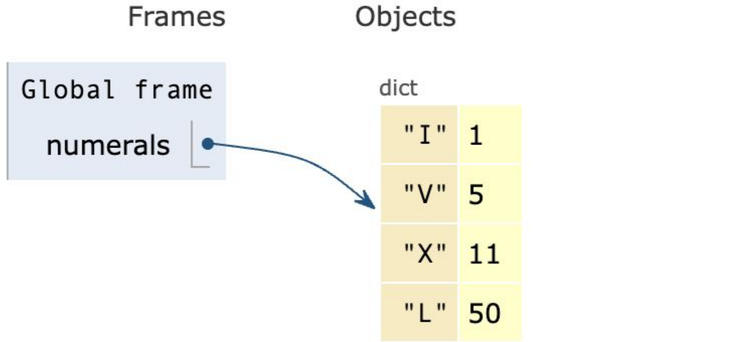
<< First | < Prev | Next > | Last >>

Step 6 of 6

Customize visualization

Print output (drag lower right corner to resize)

```
10
11
```



Numerals Example

Python 3.6
(known limitations)

```
1  numerals = {'I': 1, 'V': 5, 'X': 10}
2  print(numerals['X'])
3  numerals['X'] = 11
4  print(numerals['X'])
5  numerals['L'] = 50
6  numerals.pop('X')
```

Edit this code

line that just executed
next line to execute

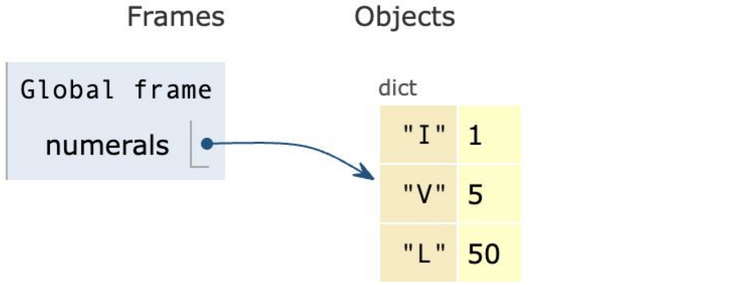
<< First | < Prev | Next > | Last >>

Done running (6 steps)

Customize visualization

Print output (drag lower right corner to resize)

```
10
11
```

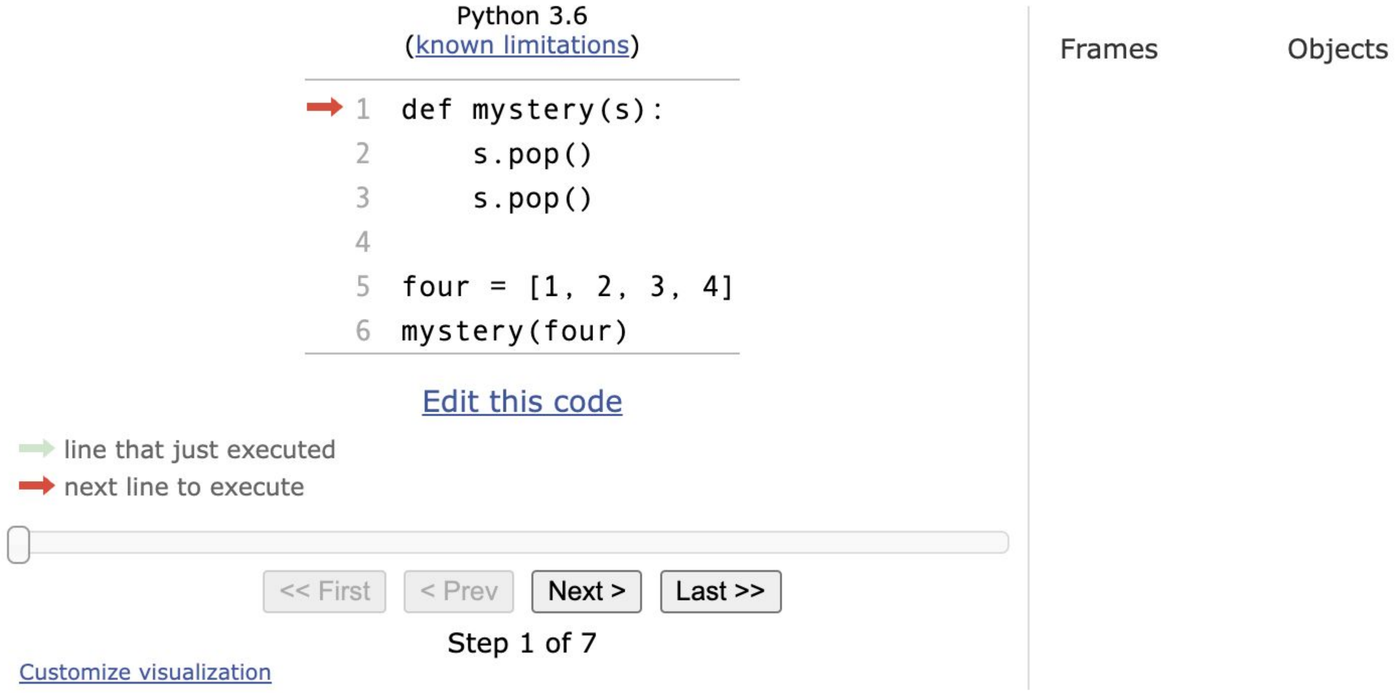


Mutation Can Happen Within a Function Call

- A function can change the value of any object in its scope

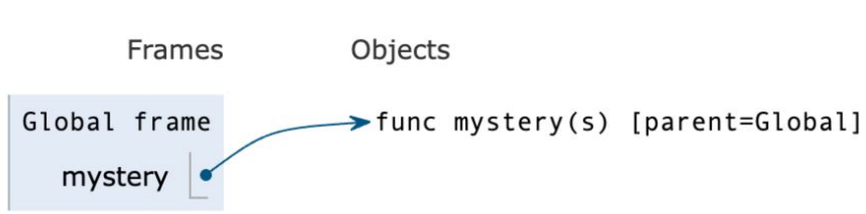
Mystery List Example

- A function can change the value of any object in its scope



Mystery List Example

- A function can change the value of any object in its scope



Mystery List Example

- A function can change the value of any object in its scope

Python 3.6
(known limitations)

```
1  def mystery(s):
2      s.pop()
3      s.pop()
4
5  four = [1, 2, 3, 4]
6  mystery(four)
```

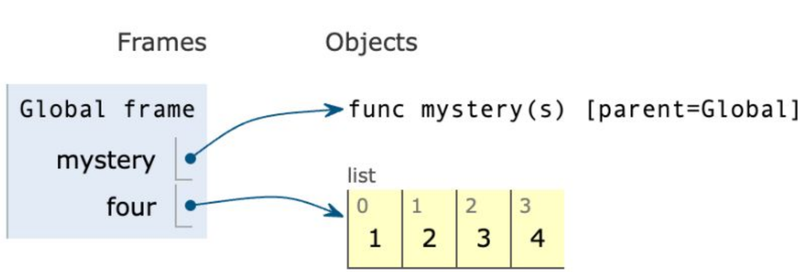
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

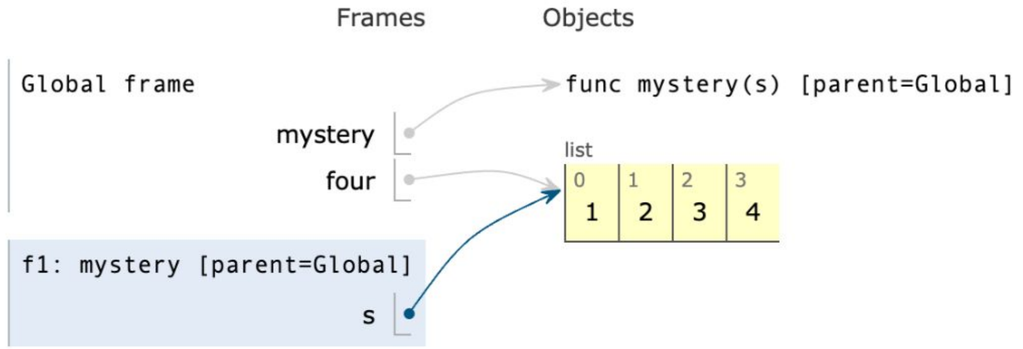
Step 3 of 7

Customize visualization



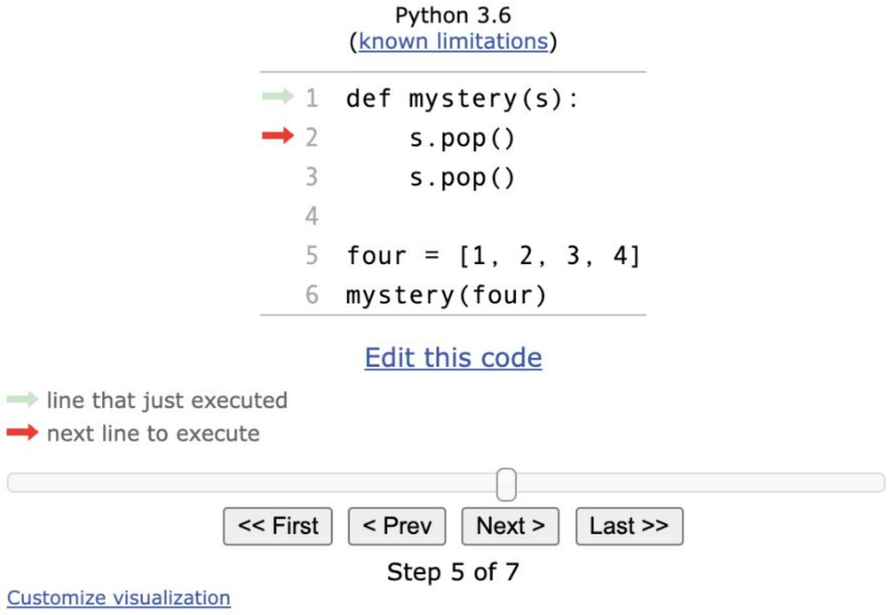
Mystery List Example

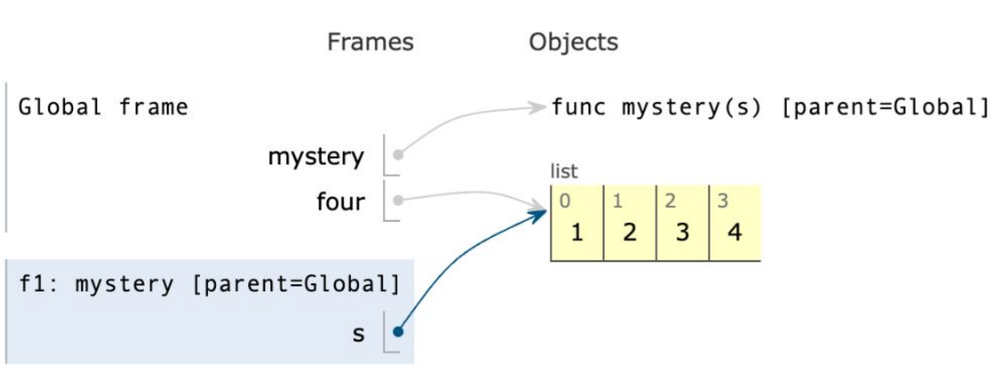
- A function can change the value of any object in its scope



Mystery List Example

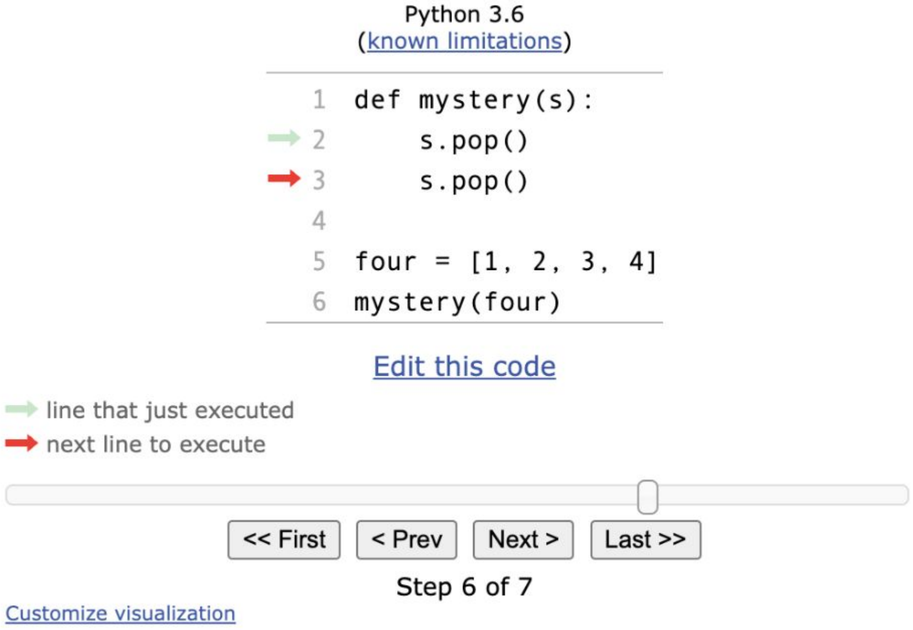
- A function can change the value of any object in its scope

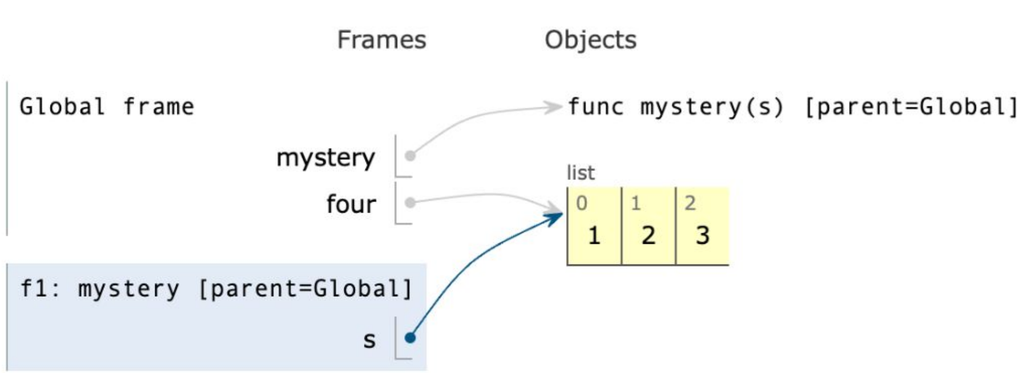




Mystery List Example

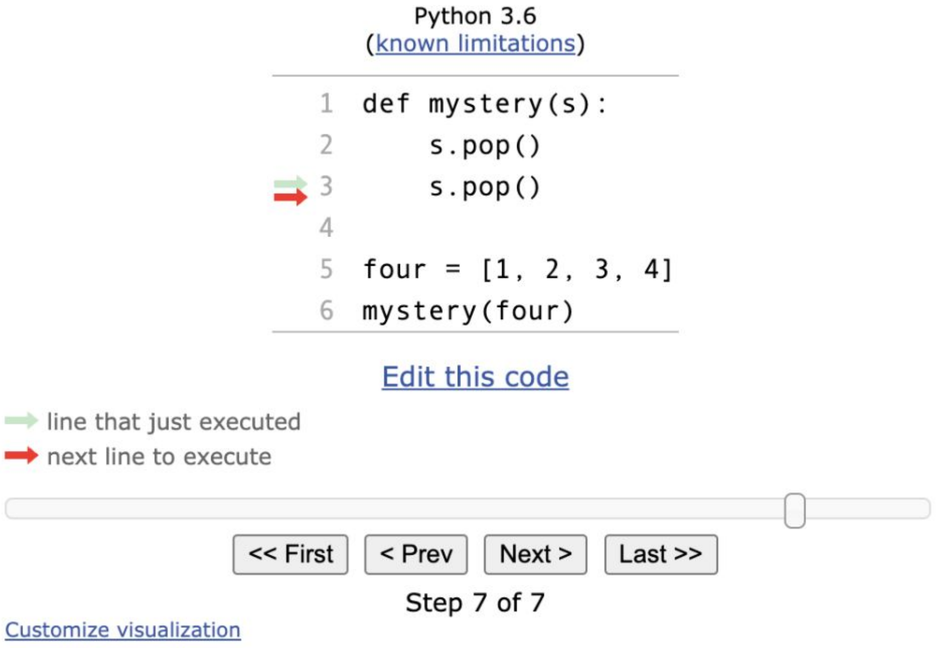
- A function can change the value of any object in its scope

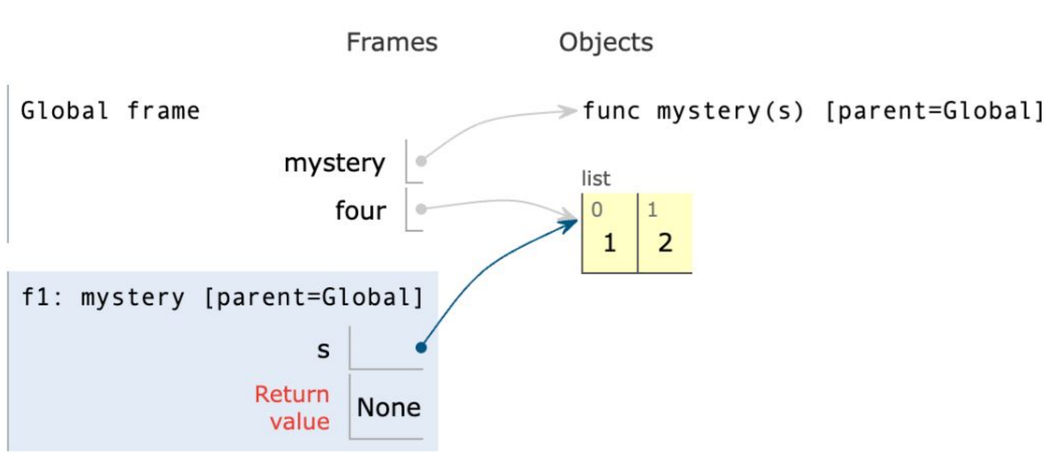




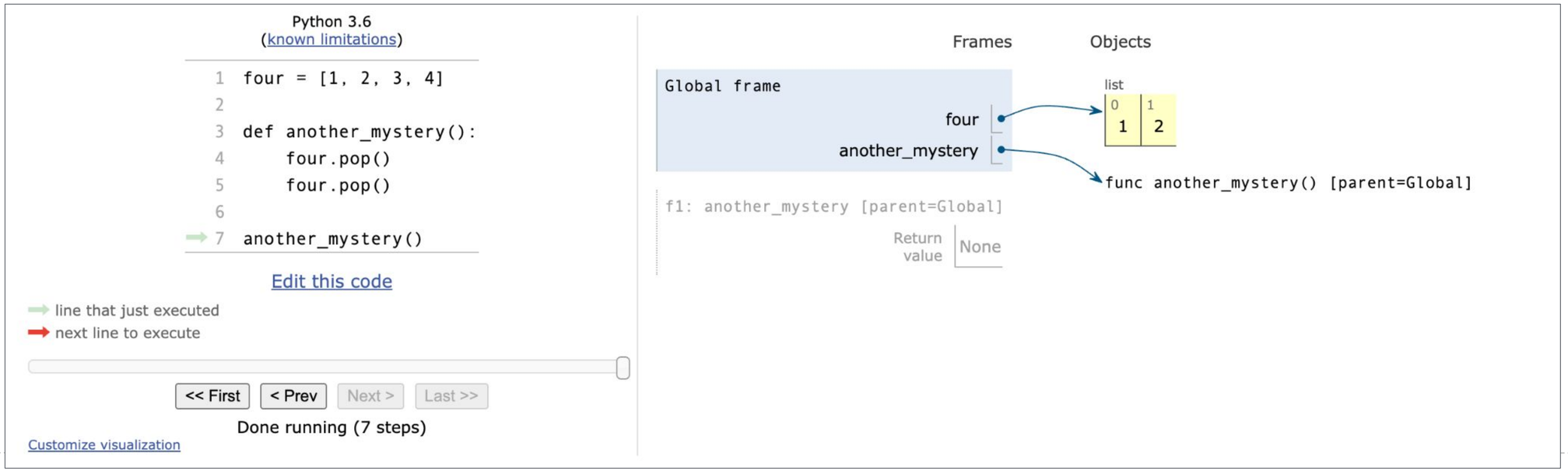
Mystery List Example

- A function can change the value of any object in its scope





Equivalent Example



4) Tuples

Introduction to Tuples

```
>>> (3, 4, 5, 6)
(3, 4, 5, 6)
>>> 3, 4, 5, 6
(3, 4, 5, 6)
>>> ()
()
>>> tuple()
()
>>> tuple([3, 4, 5])
(3, 4, 5)
>>> 2,
(2,)
>>> (2,)
(2,)
>>> 2
2
>>> (3, 4) + (5, 6)
(3, 4, 5, 6)
>>> 5 in (3, 4, 5)
True
```

Tuples Can Be Used as Dictionary Keys

```
>>> {(1, 2): 3}
{(1, 2): 3}
>>> {[1, 2]: 3}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>> {(1, [2]): 3}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Tuples Are Immutable Sequences

- Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

- The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

- An immutable sequence may still change if it *contains* a mutable value as an element

```
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
```

```
>>> s = ([1, 2], 3)
>>> s[0][0] = 4
>>> s
([4, 2], 3)
```

Break

5) Mutation

Sameness & Change

- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```

```
>>> a = [10]
>>> b = [10]
>>> a == b
True
>>> b.append(20)
>>> a
[10]
>>> b
[10, 20]
>>> a == b
False
```

Identity Operators

Identity

<exp0> **is** <exp1>

evaluates to True if both <exp0> and <exp1> evaluate to the same object

Equality

<exp0> **==** <exp1>

evaluates to True if both <exp0> and <exp1> evaluate to equal values

Identical objects are always equal values

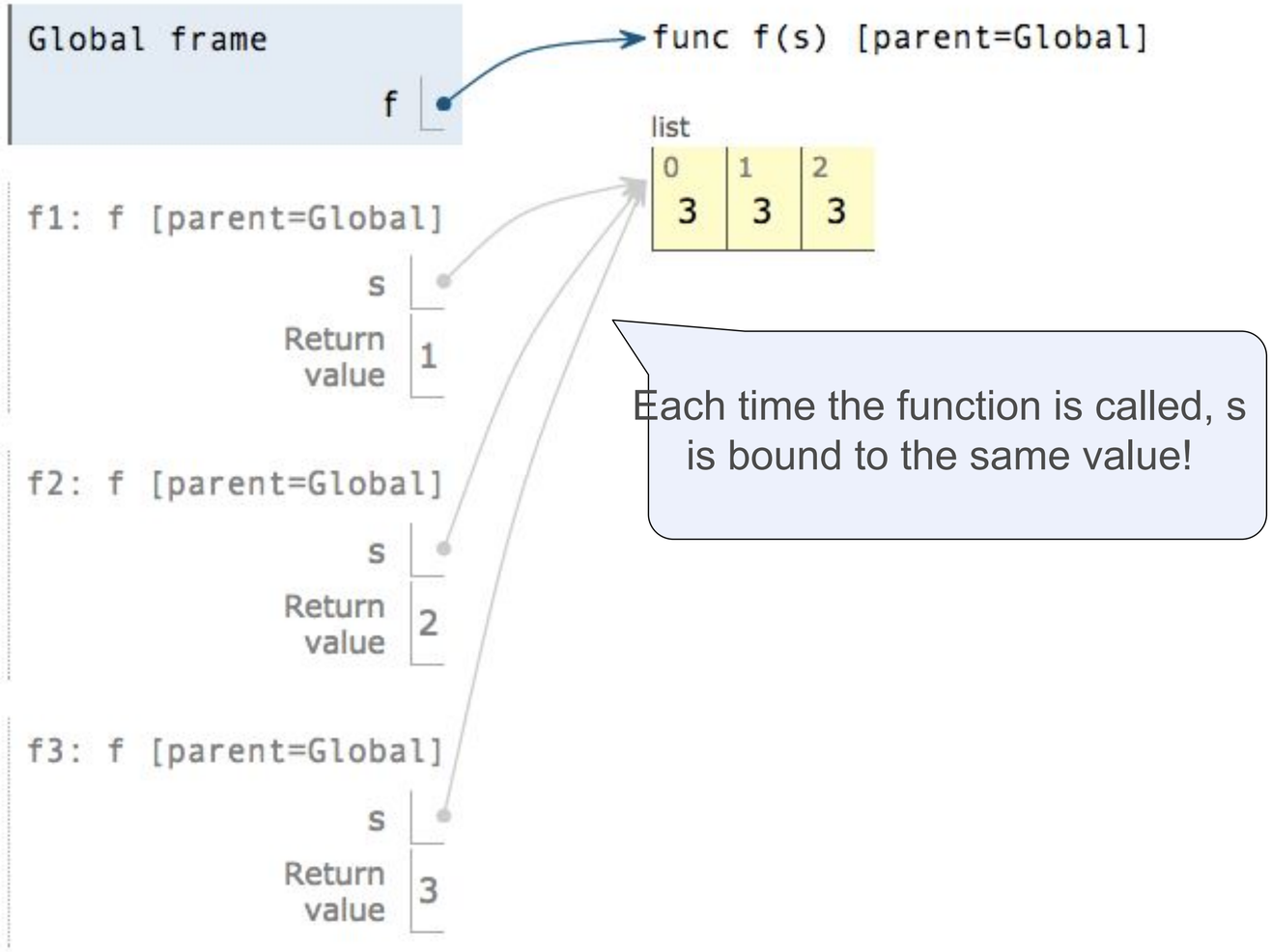
Identity Operators

```
>>> [10] == [10]
True
>>> a = [10]
>>> b = [10]
>>> a == b
True
>>> a is b
False
>>> a.extend([20, 30])
>>> a
[10, 20, 30]
>>> b
[10]
>>> c = b
>>> c is b
True
>>> c.pop()
10
>>> c
[]
>>> b
[]
>>> a
[10, 20, 30]
```

Mutable Default Arguments Are Dangerous

- A default argument value is part of a function value, not generated by a call

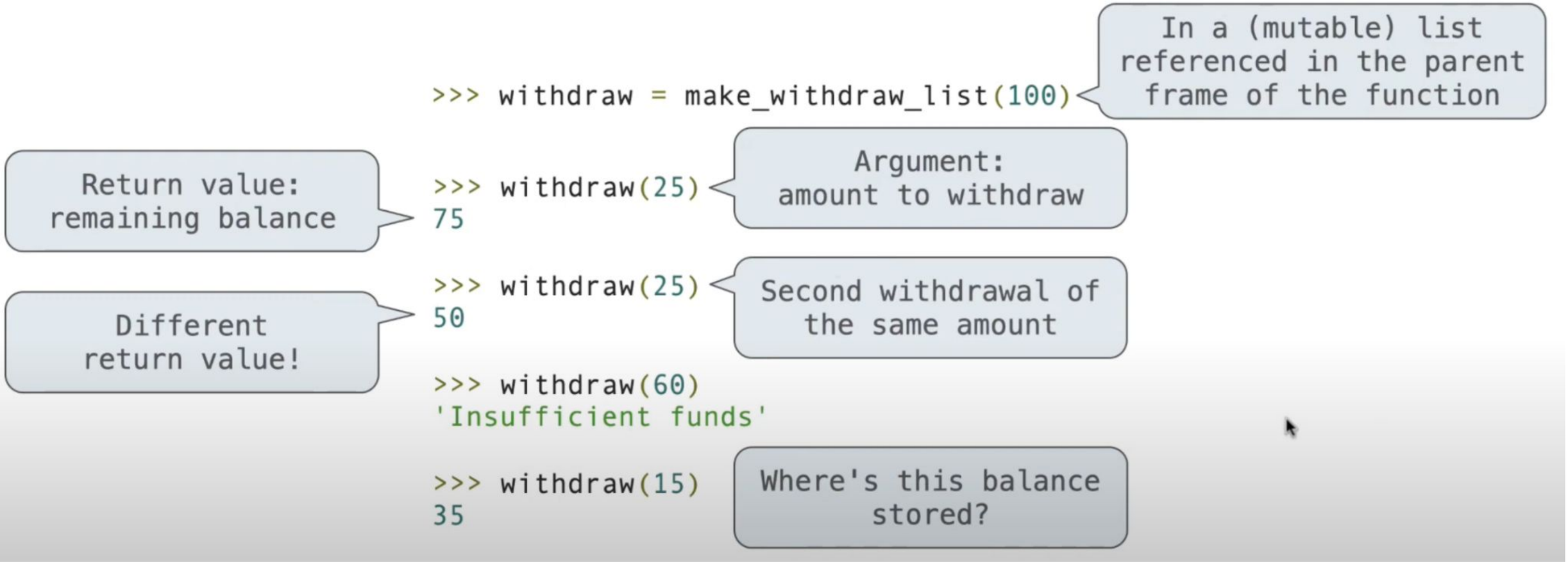
```
>>> def f(s=[]):
...     s.append(3)
...     return len(s)
...
>>> f()
1
>>> f()
2
>>> f()
3
```



6) Mutable Functions

Functions with Behavior that Changes

Let's model a bank account that has a balance of $100



Mutable Values & Persistent Local State