Mutability

Announcements

Midterm Exam Next Week
(Sorry!)

But you’ll gonna do well!

Mutating Lists: Example functions of the list class

•`append()` adds a single element to a list:

```
s = [2, 3]
t = [5, 6]
s.append(4)
s.append(t)
t = 0
```

Try in PythonTutor.

•`extend()` adds all the elements in one list to another list:

```
s = [2, 3]
t = [5, 6]
s.extend(4) # 🚫 Error: 4 is not an iterable!
s.extend(t)
t = 0
```

Try in PythonTutor. (After deleting the bad line)

Mutating Lists -- More Functions!

- `list += [x, y, z] # just like extend.`
- You need to be careful with this one! It modifies the list.
- `pop()` removes and returns the last element:

```
s = [2, 3]
t = [5, 6]
t = s.pop()
```

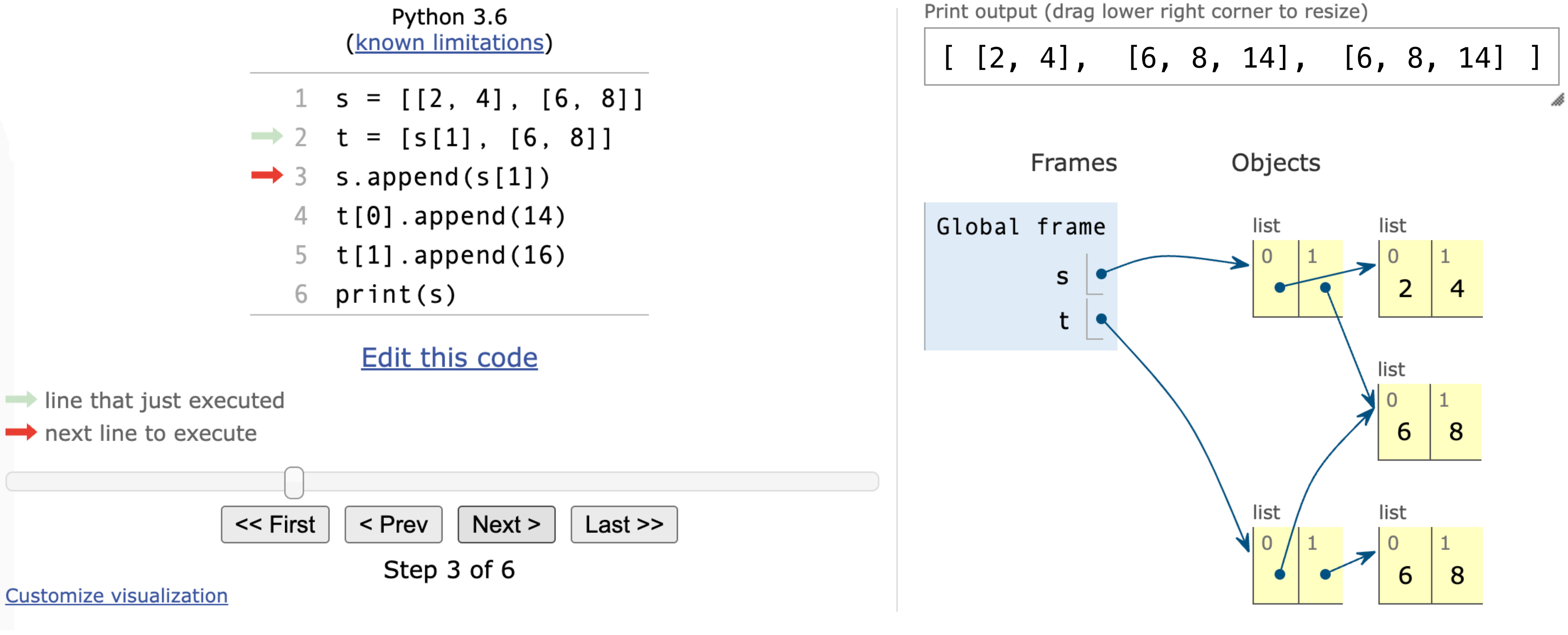
Try in PythonTutor.

- `remove()` removes the first element equal to the argument:

```
s = [6, 2, 4, 8, 4]
s.remove(4)
```

Try in PythonTutor.

Nested Lists



Python 3.6
(known limitations)

```
s = [[2, 4], [6, 8]]
t = [s[1], [6, 8]]
s.append(s[1])
t[0].append(14)
t[1].append(16)
print(s)
```

Edit this code

Print output (drag lower right corner to resize)

```
[ [2, 4],  [6, 8, 14],  [6, 8, 14] ]
```

https://pythontutor.com/cp/composingprograms.html#code=s%20%3D%20%5B%5B2,%204%5D,%20%5B6,%208%5D%5D%0At%20%3D%20%5Bs%5B1%5D,%20%5B6,%208%5D%5D%0As.append%28s%5B1%5D%29%0At%5B0%5D.append%2814%29%0At%5B1%5D.append%2816%29%0Aprint%28s%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Building Lists Using Append

```
def sums(n, m):
    """Return lists that sum to n containing positive numbers up to m that
    have no adjacent repeats, for n > 0 and m > 0.

    >>> sums(5, 1)
    []
    >>> sums(5, 2)
    [[2, 1, 2]]
    >>> sums(5, 3)
    [[1, 3, 1], [2, 1, 2], [2, 3], [3, 2]]
    >>> sums(5, 5)
    [[1, 3, 1], [1, 4], [2, 1, 2], [2, 3], [3, 2], [4, 1], [5]]
    >>> sums(6, 3)
    [[1, 2, 1, 2], [1, 2, 3], [1, 3, 2], [2, 1, 2, 1], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
    """
    result = []
    for k in range(1, ___min(m + 1, n)___): # k is the first number of a list
        for rest in ___sums(n-k, m)___:
            if rest[0] != k:
                result.append(___[k] + rest___)  # build a list out of k and rest
    if n <= m:
        result.append([n])
    return result
```

https://pythontutor.com/cp/composingprograms.html#code=result%20%3D%20%5B%5B1%5D,%20%5B2%5D,%20%5B3%5D%5D%0Afor%20k%20in%20range%281,%204%29%3A%0A%20%20%20%20for%20s%20in%20result%3A%0A%20%20%20%20%20%20%20%20if%20s%5B-1%5D%20!
%3D%20k%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20s.append%28k%29%0A%20%20%20%20%20%20%20%20%20%20%20%20result.append%28s%29%0Aprint%28result%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

7

Mutation and Identity

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- This view is no longer valid in the presence of 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

(Demo)

https://pythontutor.com/cp/composingprograms.html#code=s%20%3D%20%5B3,%205,%207%5D%0At%20%3D%20%5B9,%2011%5D%0As.append%28t%29%0As.extend%28t%29%0At%5B1%5D%20%3D%2013%0Aprint%28s%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

What is the meaning of is?

- is in Python means two items have the exact same identity
- Thus, a is b implies a == b
- Why? Each object has a function id() which returns its "address"
- The id is essentially an internal "locator" for that data in memory.
- Think of two houses which have the exact same floor plan, look the same, etc. The are "the same house" but each have a unique address. (And thus are different houses)
- Think this is tricky? cool? amazing?
  - Take CS61C (Architecture) and CS164 (Programming Languages)

Arrays vs Lists

Numpy Arrays Represent Fixed-Length Sequences of Numbers

```
import numpy as np
a = np.array([3, 4, 5, 6])
b = a + 1
```

vs

```
s = [3, 4, 5, 6]
t = [x + 1 for x in s]
```

Numpy array advantages:

- Much faster repeated arithmetic
- More concise expressions
- Handles 2+ dimensions (matrix, etc.)

Numpy disadvantages:

- Fixed size: appending makes a new array
- Fixed type: [3, 4] and [[3, 4], [5, 6]] but not [3, [4, 5]]

(Speed Test Demo)

Guidance:

- Repeated calculations over long lists of numbers should use array operations
- Collecting results as they are generated should use a list
- **We don’t use numpy in C88C**