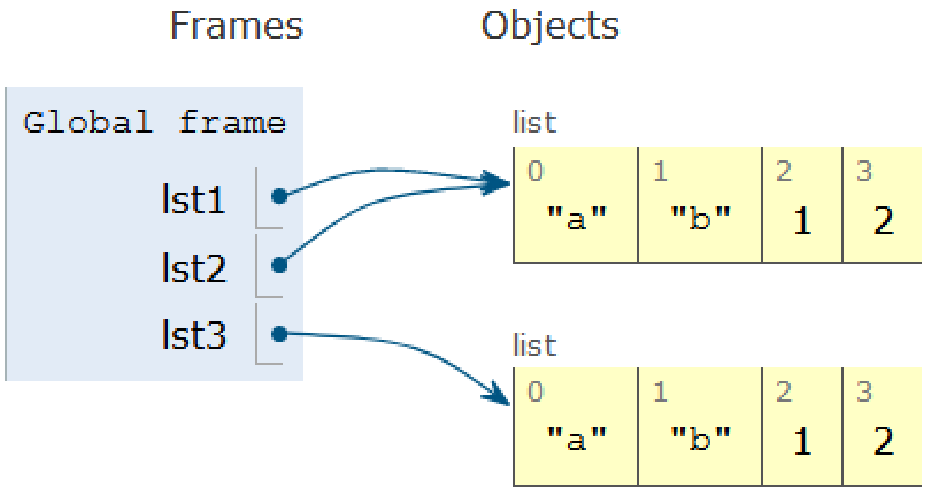
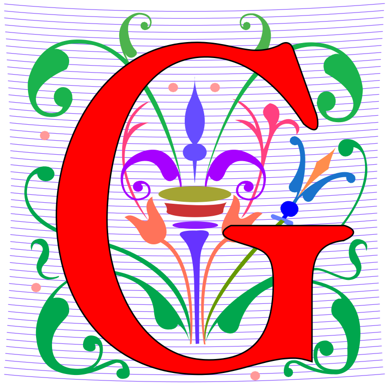
Compsci 101
List and String Operations, For loop

Susan Rodger

February 2, 2023



G is for …



- **Google**
  - How to find the answer to everything
- **Global Variable**
  - Accessible everywhere, typically do not do
- **GIGO**
  - Garbage In, Garbage Out
- **Git**
  - Working Together or Solo

Sir Tim Berners-Lee

- Invented World Wide Web
  - Turing award 2016
- HTTP vs. TCP/IP
  - Just protocols?

This Photo by Unknown Author is licensed under CC BY-SA

"The Web as I envisaged it, we have not seen it yet. The future is still so much bigger than the past."

"We need diversity of thought in the world to face the new challenges."

Did you sign up for compsci@duke.edu mailing list?

- **Mailing list to get the CompSci weekly newsletter**
  - Events, research and job opportunities
- **To add yourself:**
  - Go to lists.duke.edu
  - Authenticate and then add compsci@duke.edu

Announcements

- **Assignment 1 Faces**
  - Program due Tonight (has one grace day)
  - Also REFLECT Form due same time
  - Remember, no consulting hours on Friday
- **APT-2 out today, due Feb 9**
  - Some good practice for the exam
- **Lab 3 Friday**
  - Do prelab 3 before attending!
- **Exam 1 on Tuesday, Feb 7**

PFTD

- **Immutable Types**
- **Objects and what that means**
- **Lists continued**
- **String methods and more**
- **For Loops**
- **Exam 1**

Immutable built-in Types

- **In python string, int, float, boolean - Immutable**
  - Once created cannot change
  - These are still objects in Python3!!
- **Assignment makes a copy**
  - b = a
  - b gets a copy of a
- **Let's look at an example**
  - Example with integers

```
val = 0
bee = val
val = val + 20
```

Immutable built-in Types

- **In python string, int, float, boolean - Immutable**
  - Once created cannot change
  - These are still objects in Python3!!
- **Assignment makes a copy**
  - b = a
  - b gets a copy of a
- **Let's look at an example**
  - Example with integers

```
val = 0
bee = val
val = val + 20
```

```
val  is  0
```

Immutable built-in Types

- **In python string, int, float, boolean - Immutable**
  - Once created cannot change
  - These are still objects in Python3!!
- **Assignment makes a copy**
  - b = a
  - b gets a copy of a
- **Let's look at an example**
  - Example with integers

```
val = 0
bee = val
val = val + 20
```

```
val   is   0
bee   is   0
```

bee is a copy of val

Immutable built-in Types

- **In python string, int, float, boolean - Immutable**
  - Once created cannot change
  - These are still objects in Python3!!
- **Assignment makes a copy**
  - b = a
  - b gets a copy of a
- **Let's look at an example**
  - Example with integers

```
val = 0
bee = val
val = val + 20
```

```
val  is  20
bee  is  0
```

val changing, doesn't affect bee

Immutable built-in Types

- **In python string, int, float, boolean - Immutable**
  - Once created cannot change
  - These are still objects in Python3!!
- **Assignment makes a copy**
  - b = a
  - b gets a copy of a
- **Here is another example!**
  - With strings!

```
val = "apple"
bee = val
val = val + "sauce"
```

Immutable built-in Types

- **In python string, int, float, boolean - Immutable**
  - Once created cannot change
  - These are still objects in Python3!!
- **Assignment makes a copy**
  - b = a
  - b gets a copy of a
- **Here is another example!**
  - With strings!

```
val = "apple"
bee = val
val = val + "sauce"
```

```
val  is "apple"
```

Immutable built-in Types

- **In python string, int, float, boolean - Immutable**
  - Once created cannot change
  - These are still objects in Python3!!
- **Assignment makes a copy**
  - b = a
  - b gets a copy of a
- **Here is another example!**
  - With strings!

```
val = "apple"
bee = val
val = val + "sauce"
```

```
val   is "apple"
bee   is "apple"
```

bee is a copy of val

Immutable built-in Types

- **In python string, int, float, boolean - Immutable**
  - Once created cannot change
  - These are still objects in Python3!!
- **Assignment makes a copy**
  - b = a
  - b gets a copy of a
- **Here is another example!**
  - With strings!

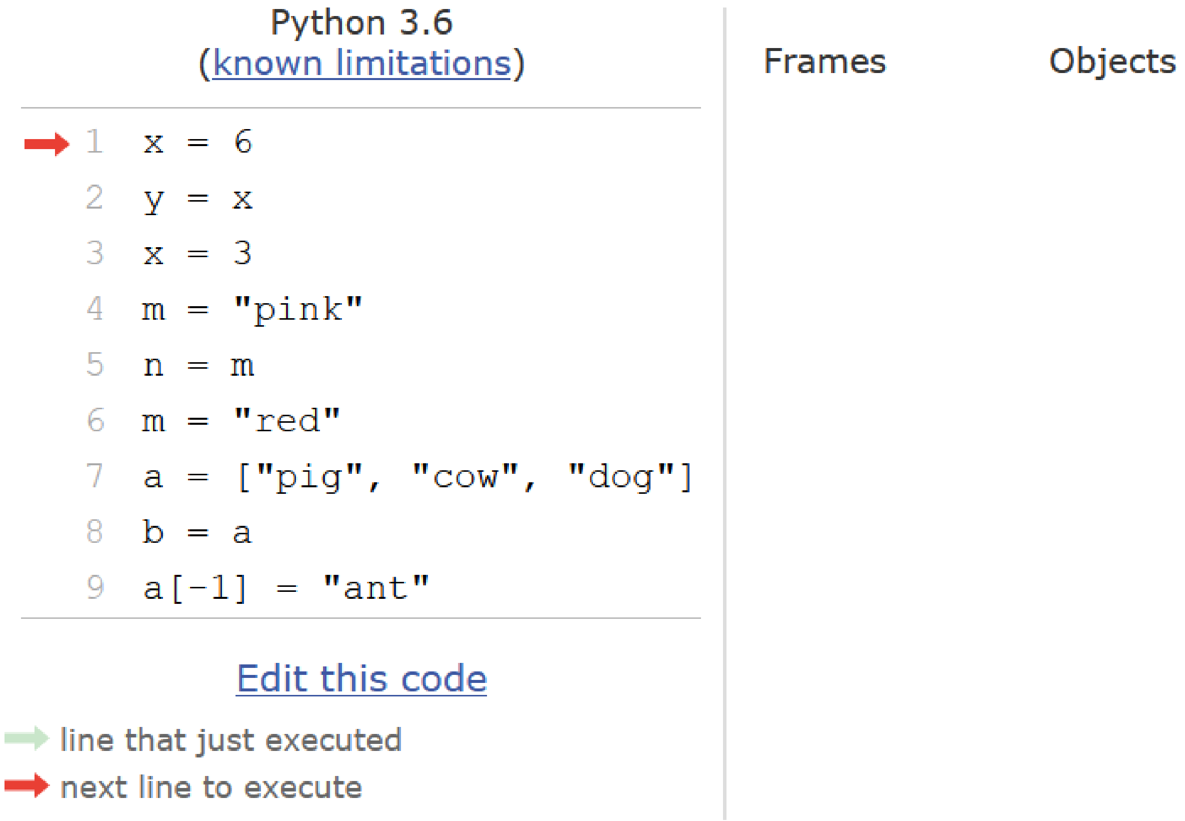
```
val = "apple"
bee = val
val = val + "sauce"
```

```
val  is "applesauce"
bee  is "apple"
```

val changing, doesn't affect bee

Let's see how the memory works in Python Tutor

Compare assign with integers, strings and lists



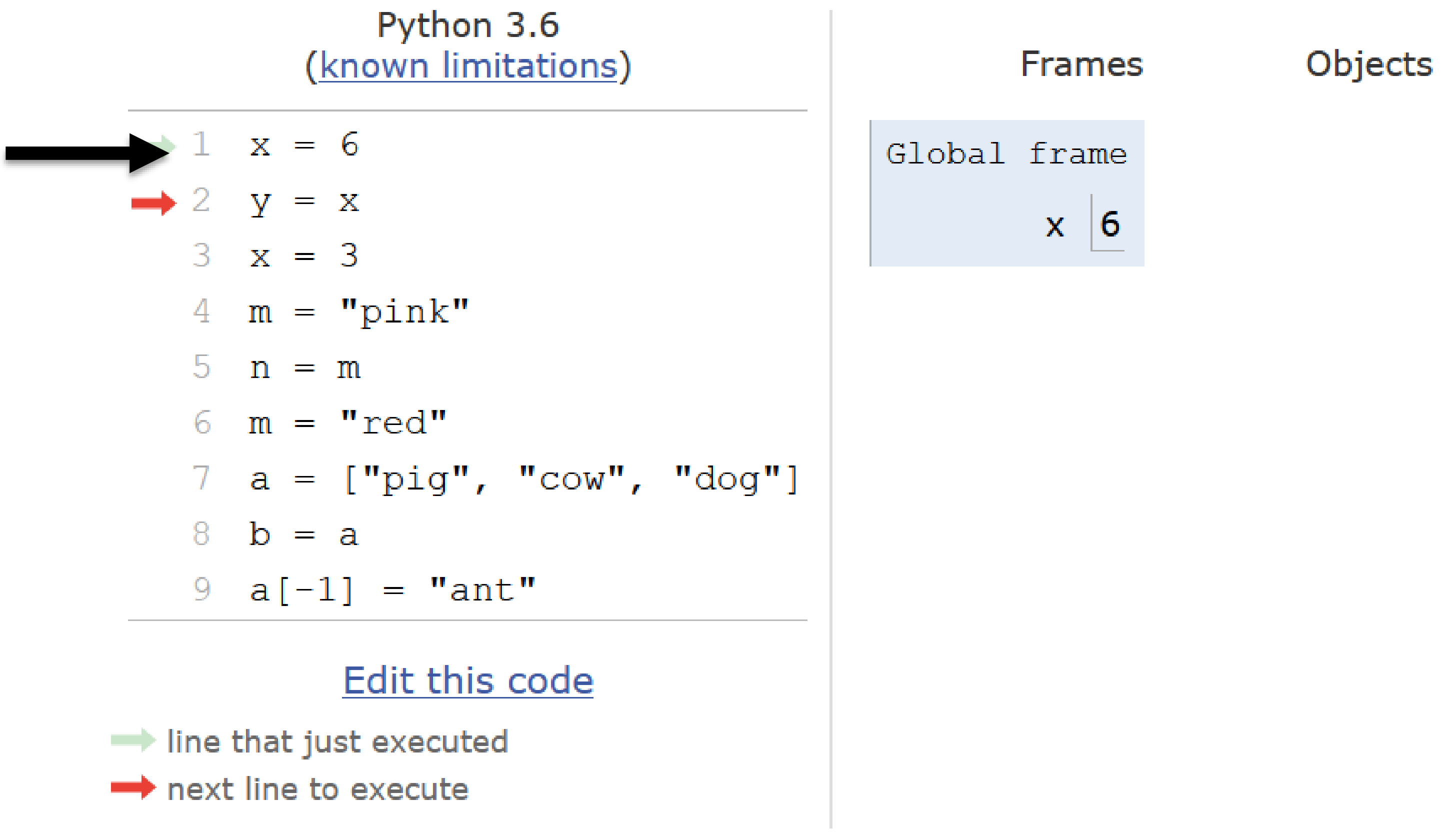
Python 3.6
(known limitations)

```
x = 6
y = x
x = 3
m = "pink"
n = m
m = "red"
a = ["pig", "cow", "dog"]
b = a
a[-1] = "ant"
```

Edit this code

➡ line that just executed
➡ next line to execute

Compare assign with integers, strings and lists



Python 3.6
(known limitations)

```
x = 6
y = x
x = 3
m = "pink"
n = m
m = "red"
a = ["pig", "cow", "dog"]
b = a
a[-1] = "ant"
```

Edit this code

➡ line that just executed

➡ next line to execute

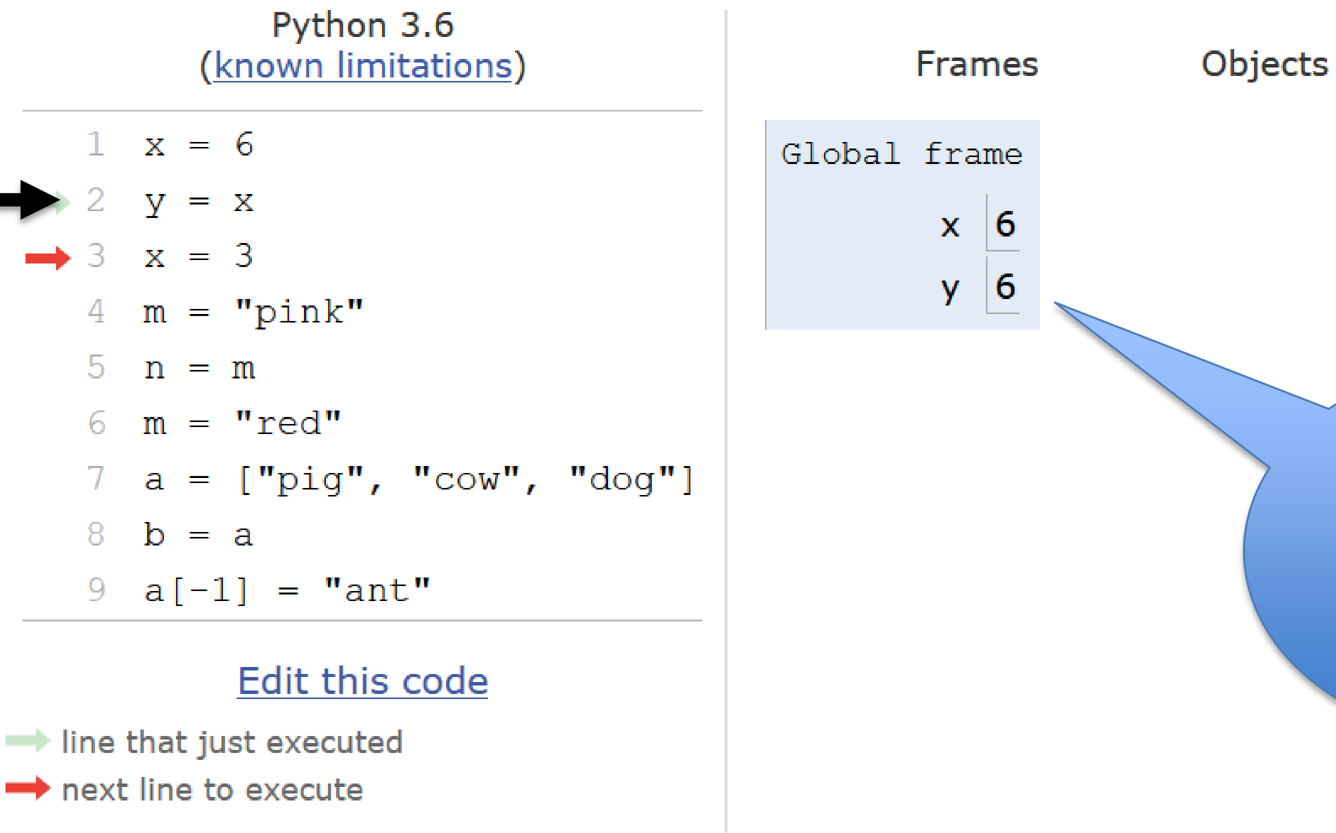
Frames

Objects

Global frame

x	6

Compare assign with integers, strings and lists



Python 3.6
(known limitations)

```
x = 6
y = x
x = 3
m = "pink"
n = m
m = "red"
a = ["pig", "cow", "dog"]
b = a
a[-1] = "ant"
```

Edit this code

➡ line that just executed

➡ next line to execute

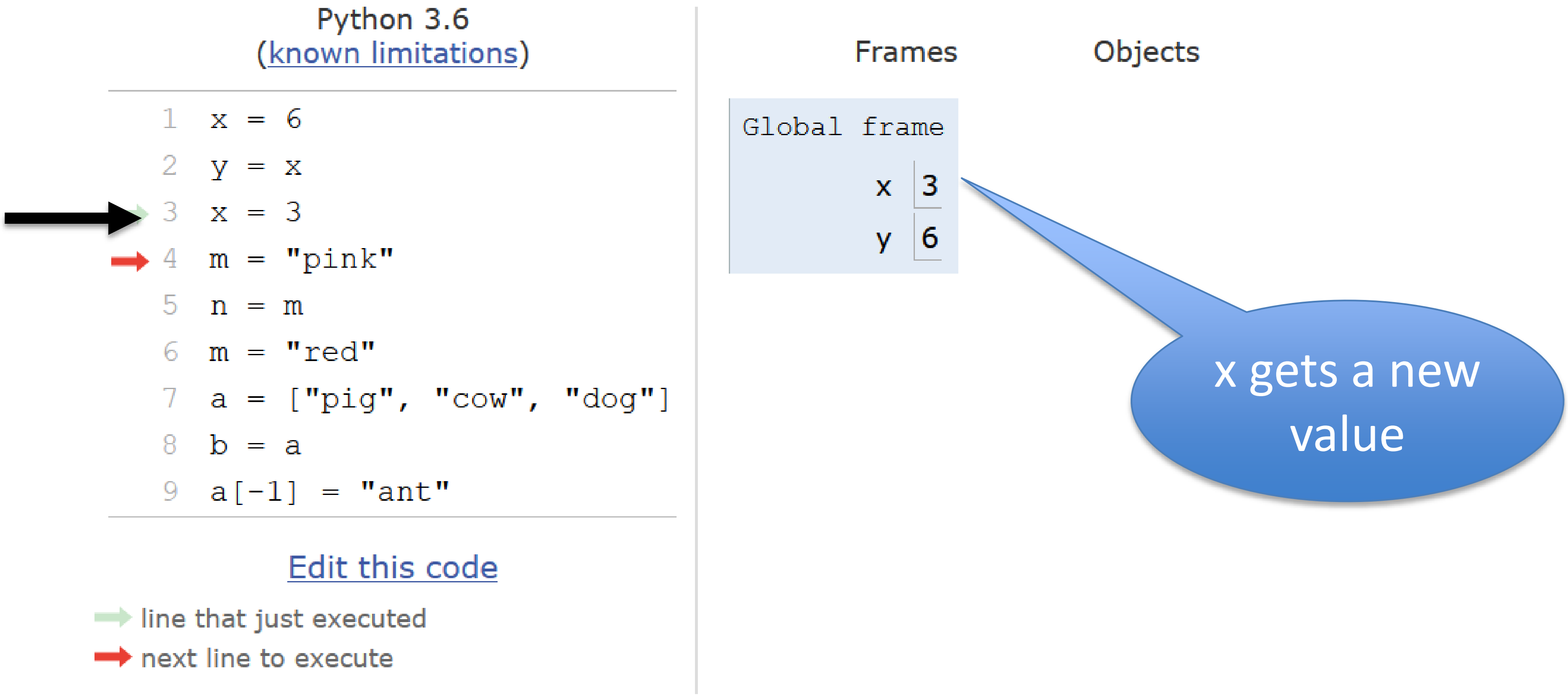
Frames Objects

Global frame

x	6
y	6

y gets a copy of the value of x

Compare assign with integers, strings and lists



Python 3.6
(known limitations)

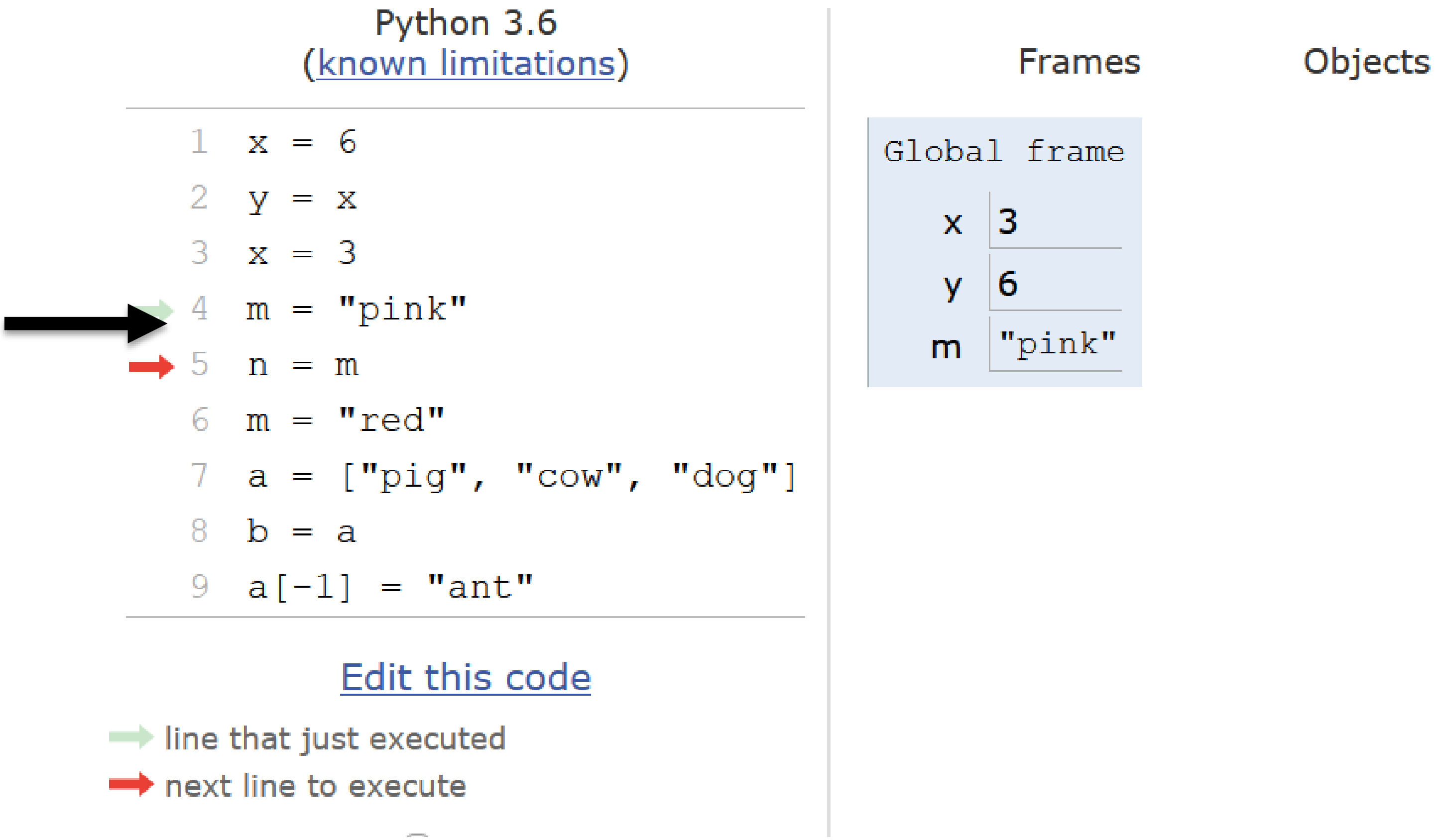
```
x = 6
y = x
x = 3
m = "pink"
n = m
m = "red"
a = ["pig", "cow", "dog"]
b = a
a[-1] = "ant"
```

Edit this code

→ line that just executed
→ next line to execute

x gets a new value

Compare assign with integers, strings and lists



```
x = 6
y = x
x = 3
m = "pink"
n = m
m = "red"
a = ["pig", "cow", "dog"]
b = a
a[-1] = "ant"
```

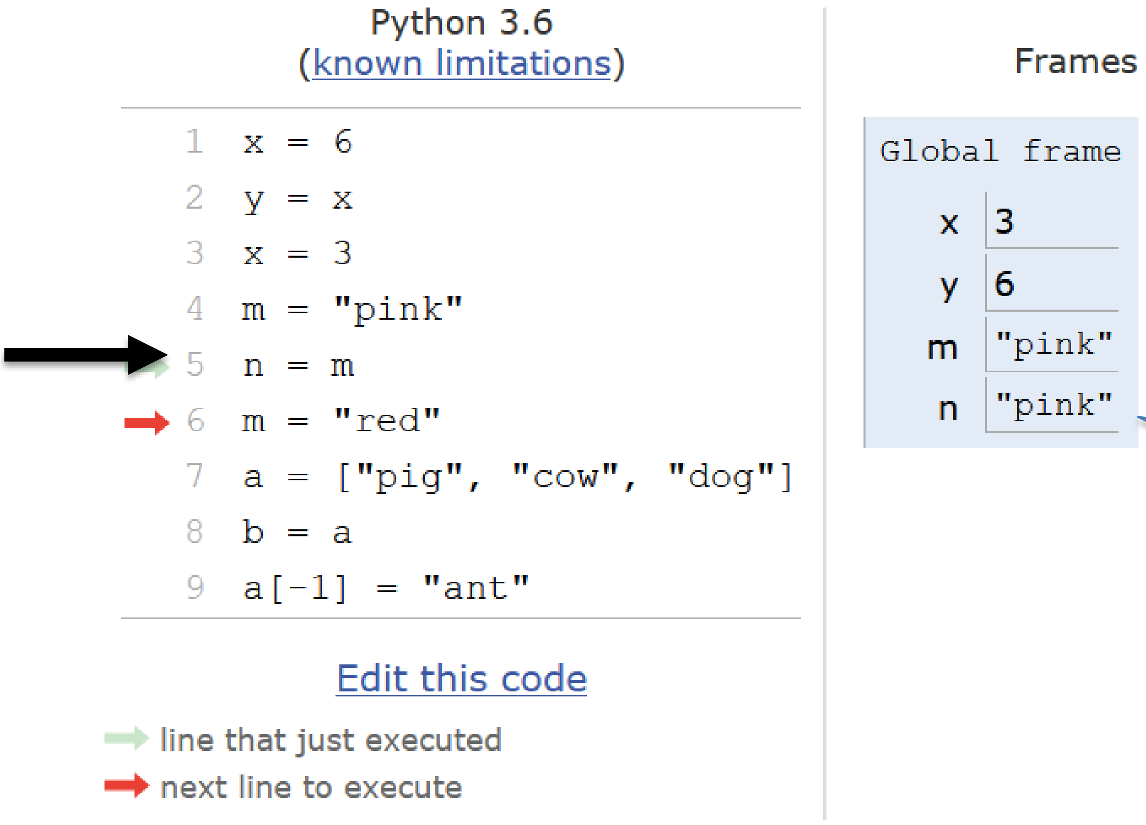
➡ line that just executed

➡ next line to execute

Global frame

x	3
y	6
m	"pink"

Compare assign with integers, strings and lists



Python 3.6
(known limitations)

```
x = 6
y = x
x = 3
m = "pink"
n = m
m = "red"
a = ["pig", "cow", "dog"]
b = a
a[-1] = "ant"
```

Edit this code

→ line that just executed
→ next line to execute

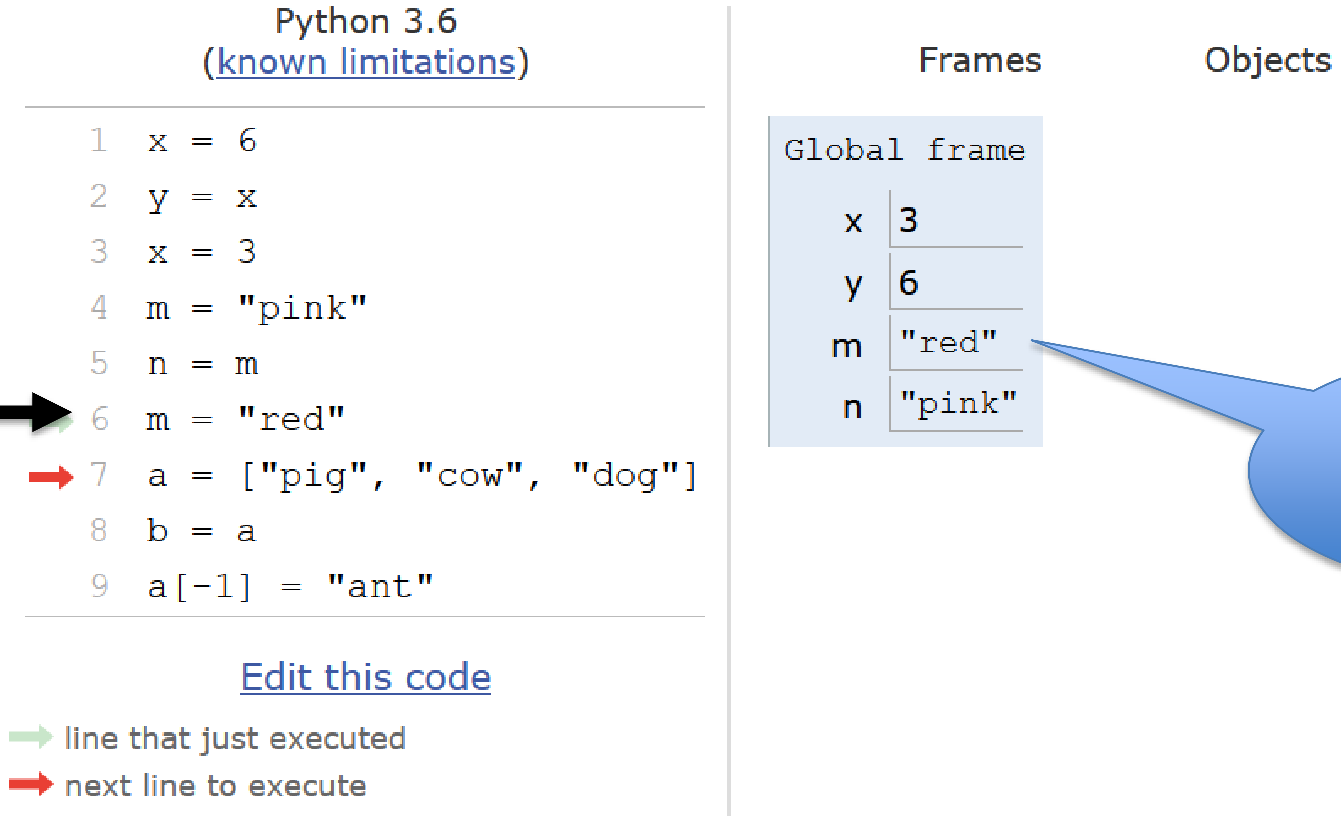
Frames Objects

Global frame

x	3
y	6
m	"pink"
n	"pink"

n gets a copy of the value of m

Compare assign with integers, strings and lists



Python 3.6
(known limitations)

```
x = 6
y = x
x = 3
m = "pink"
n = m
m = "red"
a = ["pig", "cow", "dog"]
b = a
a[-1] = "ant"
```

Edit this code

➡ line that just executed
➡ next line to execute

Frames Objects

Global frame

x	3
y	6
m	"red"
n	"pink"

m gets a new value

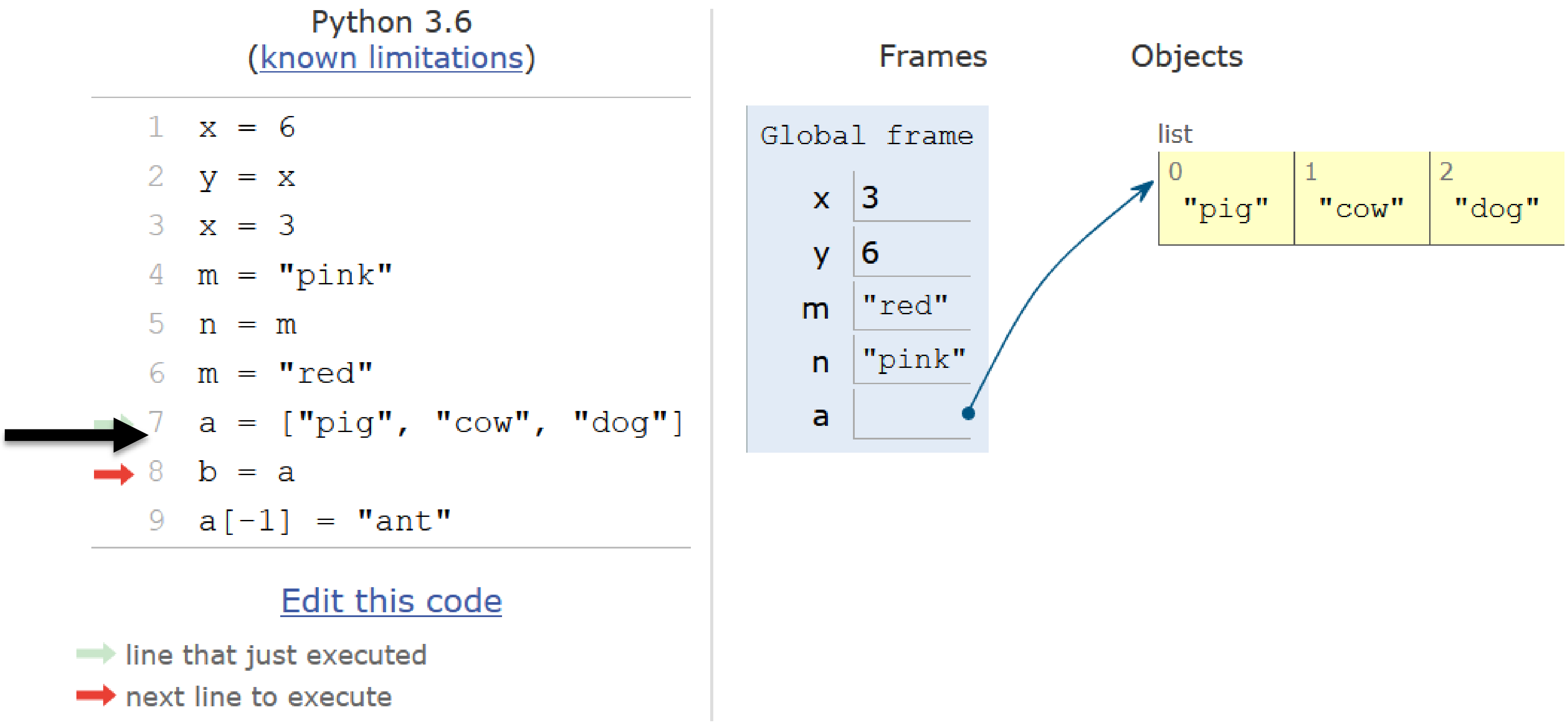
What about lists?

What happens when a and b are list variables?

b = a

It is a copy! Of what?

Compare assign with integers, strings and lists



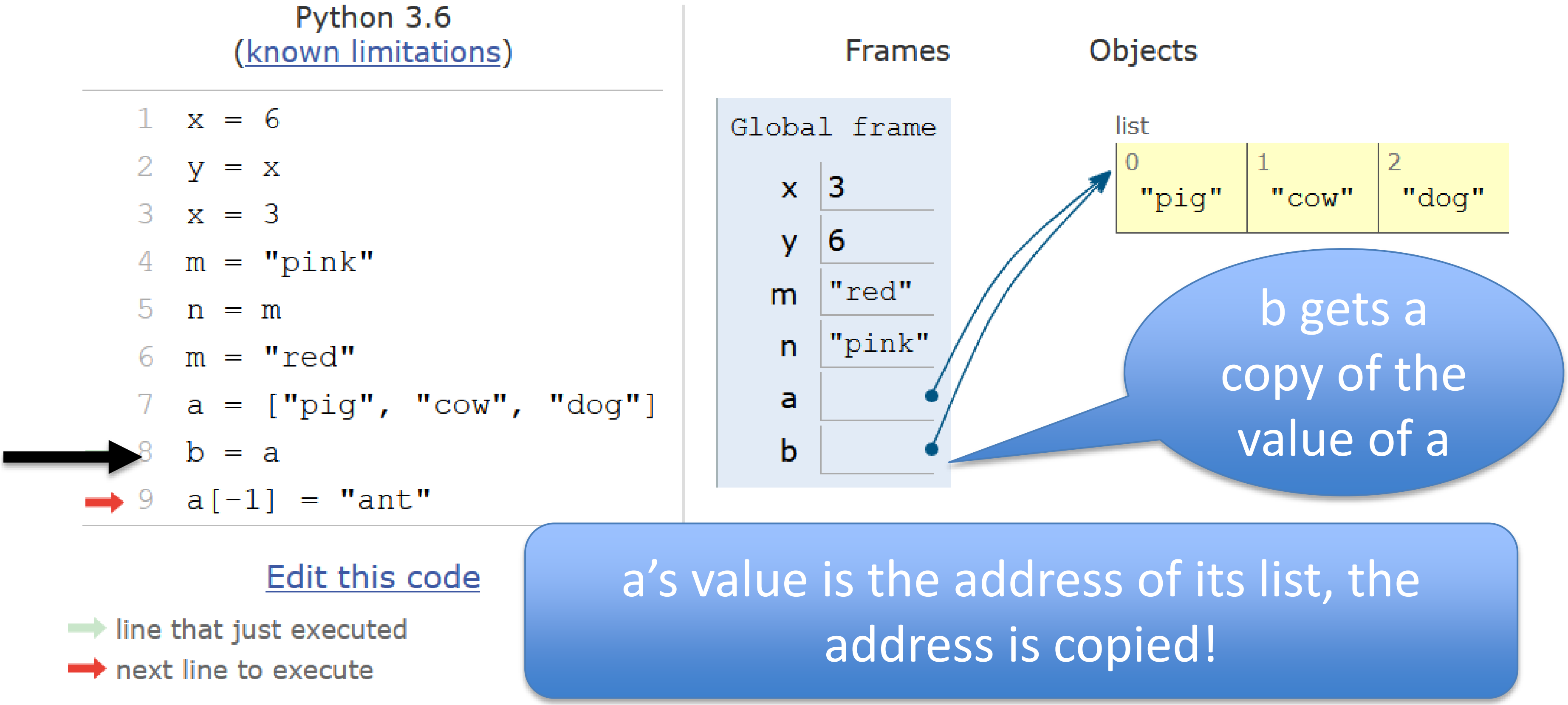
Python 3.6
(known limitations)

```
x = 6
y = x
x = 3
m = "pink"
n = m
m = "red"
a = ["pig", "cow", "dog"]
b = a
a[-1] = "ant"
```

Edit this code

line that just executed
next line to execute

Compare assign with integers, strings and lists



Python 3.6
(known limitations)

```
x = 6
y = x
x = 3
m = "pink"
n = m
m = "red"
a = ["pig", "cow", "dog"]
b = a
a[-1] = "ant"
```

Edit this code

Global frame

x	3
y	6
m	"red"
n	"pink"
a	●
b	●

list

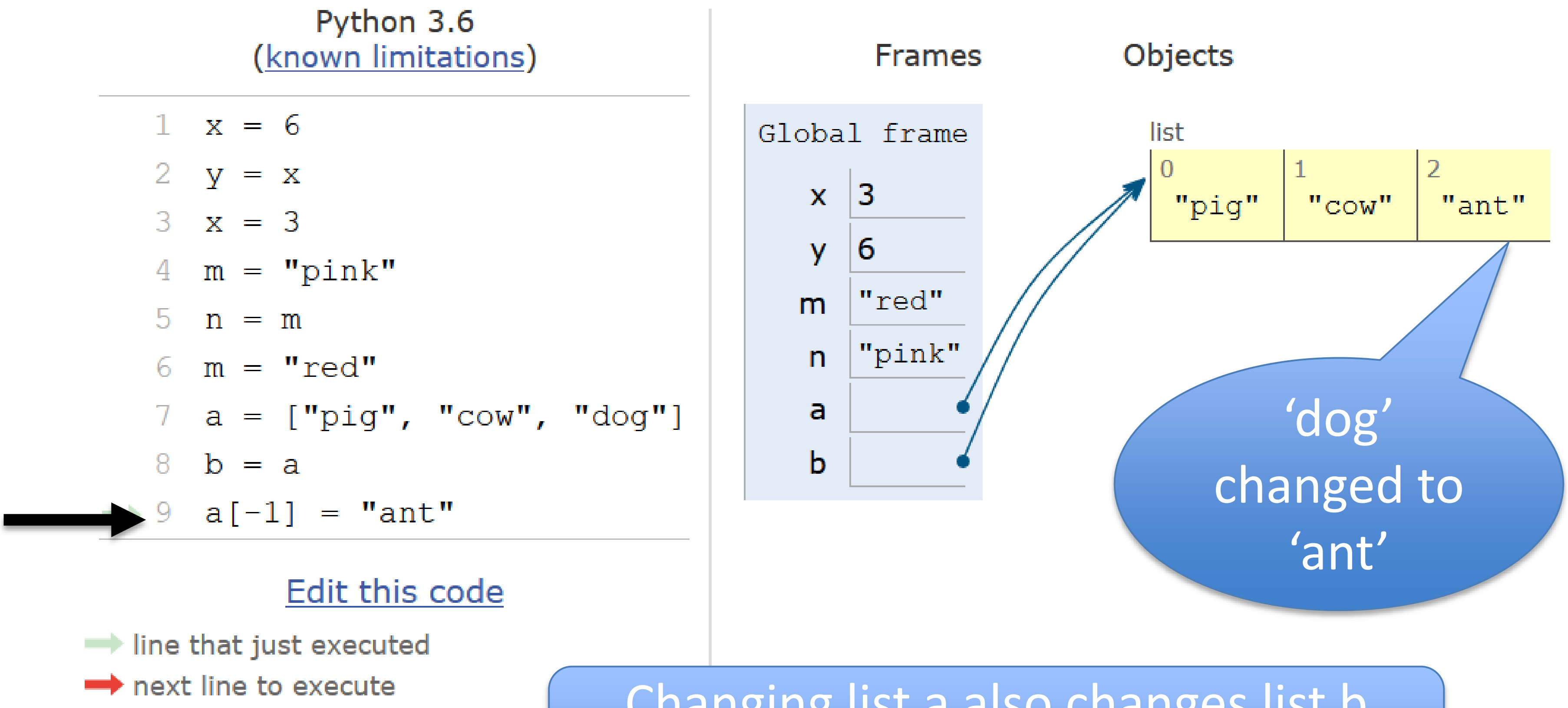
0	1	2
"pig"	"cow"	"dog"

b gets a copy of the value of a

a's value is the address of its list, the address is copied!

a and b refer to the same list!

Compare assign with integers, strings and lists



Python 3.6
(known limitations)

```
x = 6
y = x
x = 3
m = "pink"
n = m
m = "red"
a = ["pig", "cow", "dog"]
b = a
a[-1] = "ant"
```

Edit this code

Global frame

x	3
y	6
m	"red"
n	"pink"
a	●
b	●

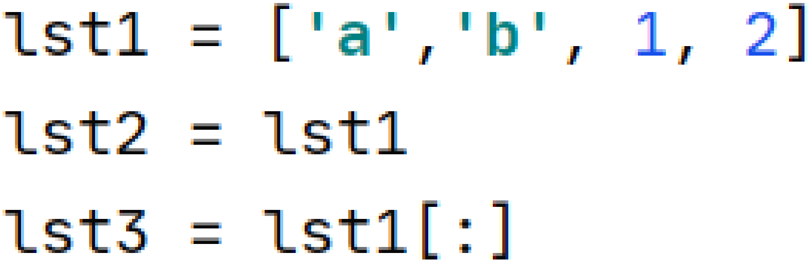
list

0	1	2
"pig"	"cow"	"ant"

‘dog’ changed to ‘ant’

Changing list a also changes list b
As they are the same list!

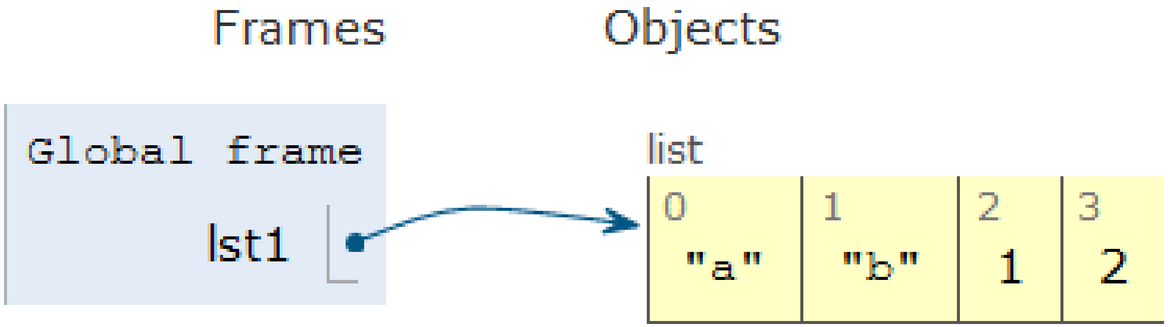
List Cloning (or copying)



```
lst1 = ['a','b', 1, 2]
lst2 = lst1
lst3 = lst1[:]
```

List Cloning (or copying)

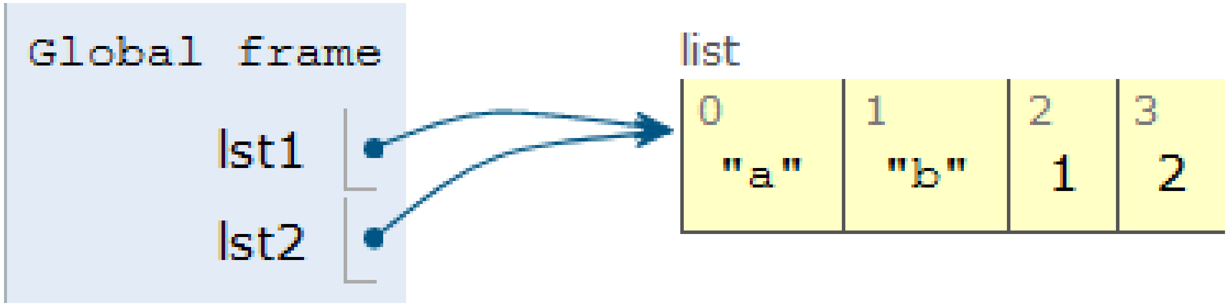
```
lst1 = ['a','b', 1, 2]
lst2 = lst1
lst3 = lst1[:]
```



0	1	2	3
"a"	"b"	1	2

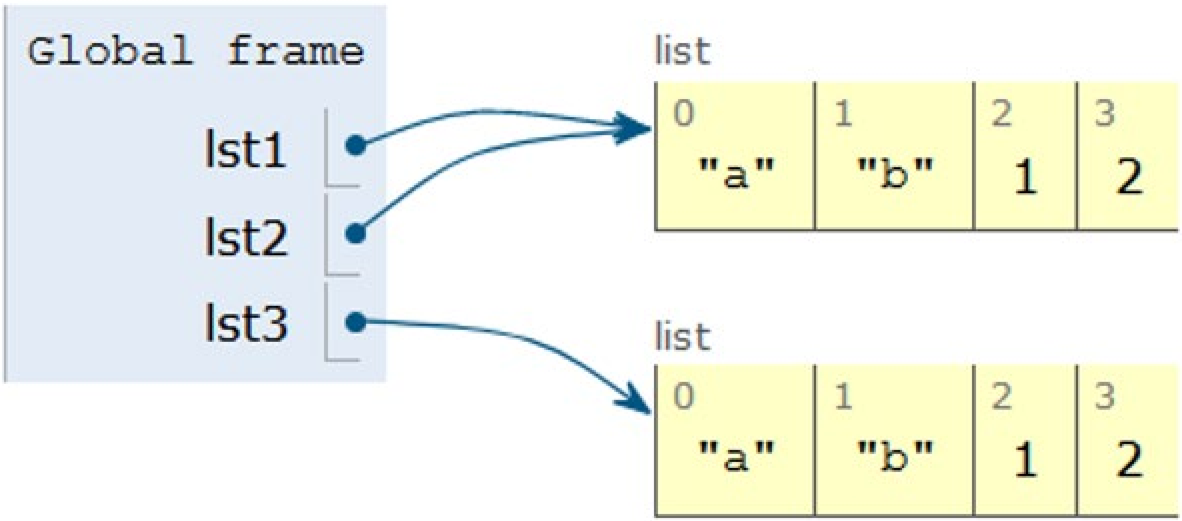
List Cloning (or copying)

```
lst1 = ['a','b', 1, 2]
lst2 = lst1
lst3 = lst1[:]
```



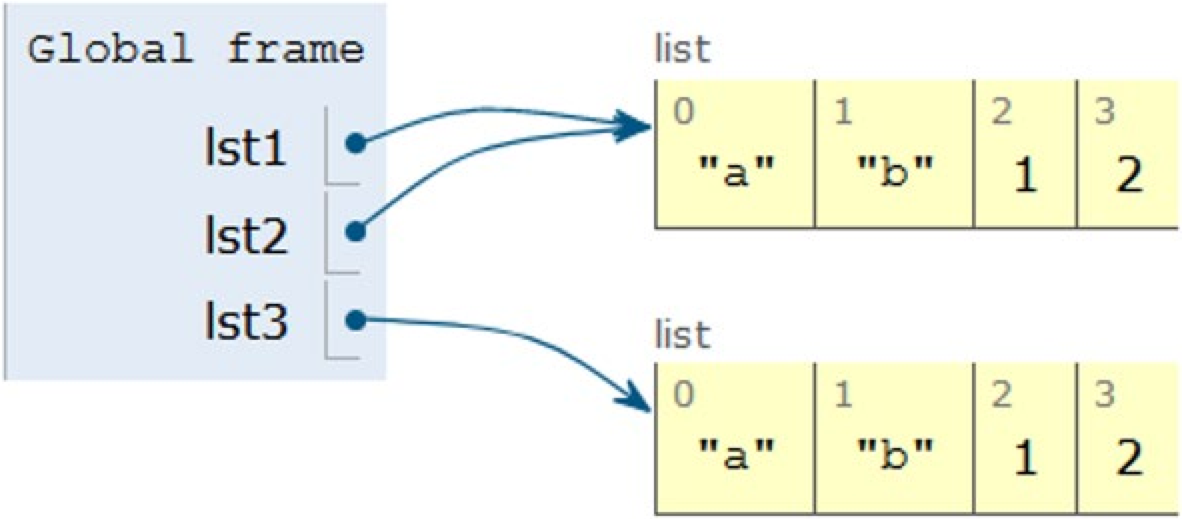
List Cloning (or copying)

```
lst1 = ['a','b', 1, 2]
lst2 = lst1
lst3 = lst1[:]
```



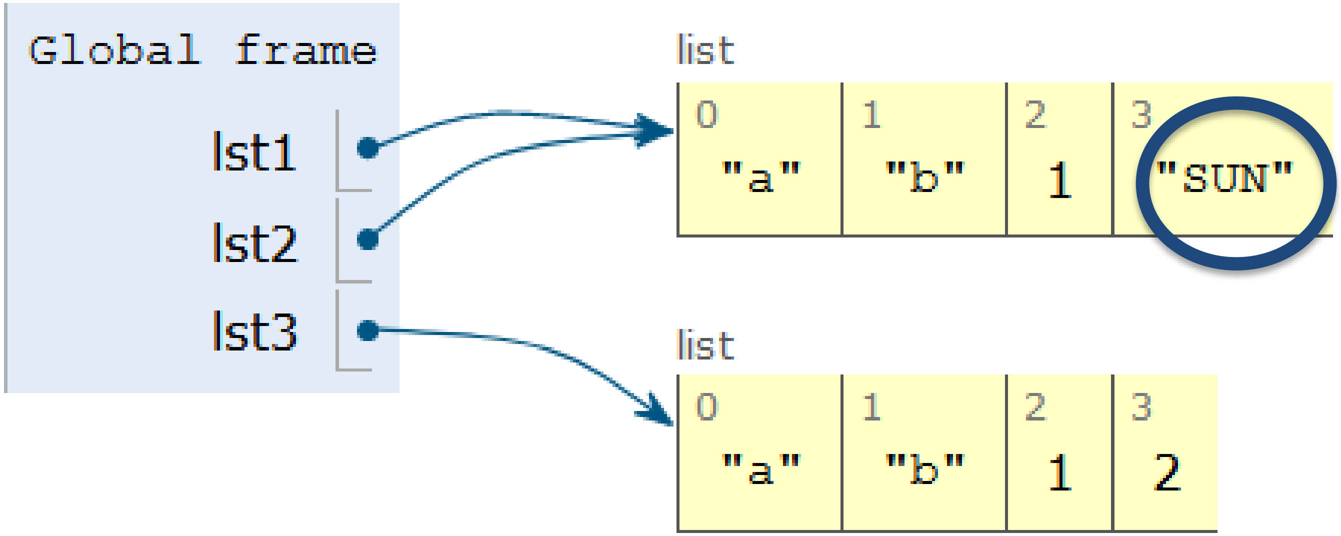
List Cloning (or copying)

```
lst1 = ['a','b', 1, 2]
lst2 = lst1
lst3 = lst1[:]
lst1[-1] = "SUN"
```



List Cloning (or copying)

```
lst1 = ['a','b', 1, 2]
lst2 = lst1
lst3 = lst1[:]
lst1[-1] = "SUN"
```



Global frame

lst1

lst2

lst3

list

0	1	2	3
"a"	"b"	1	"SUN"

list

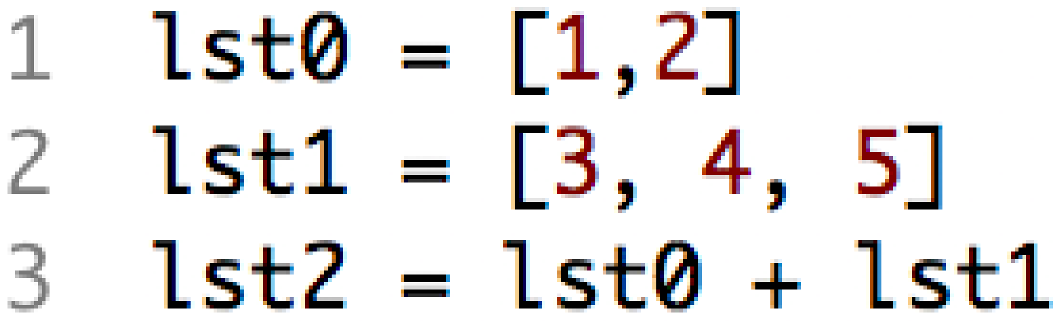
0	1	2	3
"a"	"b"	1	2

WOTO-1 Cloning
http://bit.ly/101s23-0202-1

List Concatenation Steps

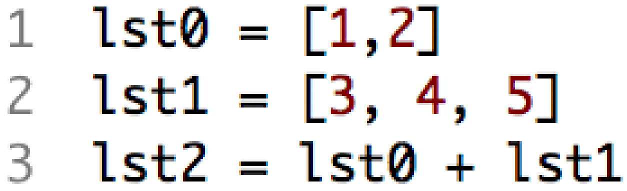
1. **Calculate the *length* of the new list**
2. ***Create* list of that length**
3. ***Copy* values from first list**
4. ***Copy* values from second list**
5. ***Assign the variable to the new list***

Brand new list!



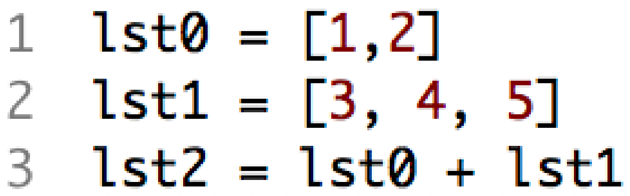
```python
lst0 = [1,2]
lst1 = [3, 4, 5]
lst2 = lst0 + lst1
```

Concatenation: length, create, copy, copy, assign

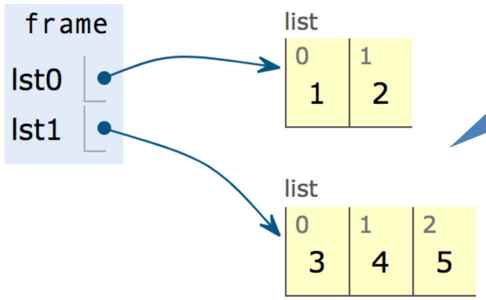


```
lst0 = [1,2]
lst1 = [3, 4, 5]
lst2 = lst0 + lst1
```

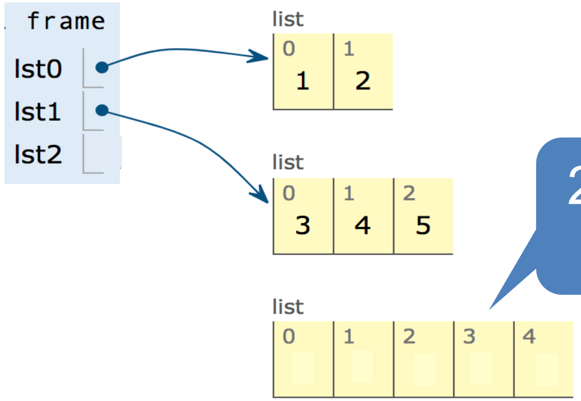
Concatenation:
length, create, copy, copy, assign



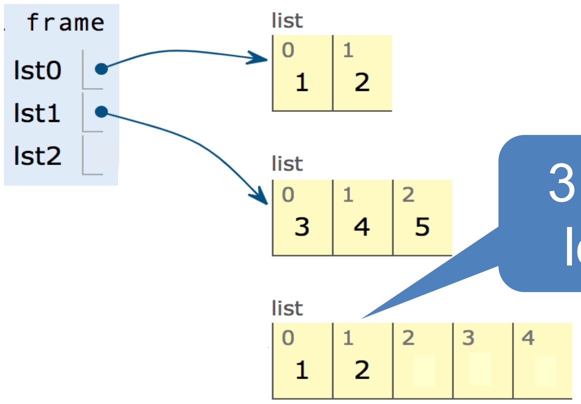
```
lst0 = [1,2]
lst1 = [3, 4, 5]
lst2 = lst0 + lst1
```



1. Calculate length

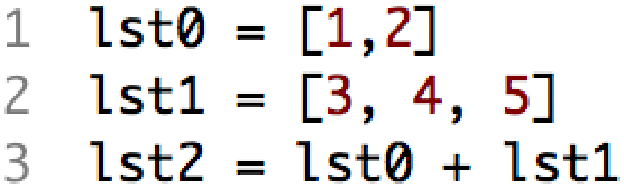


2. Create new list

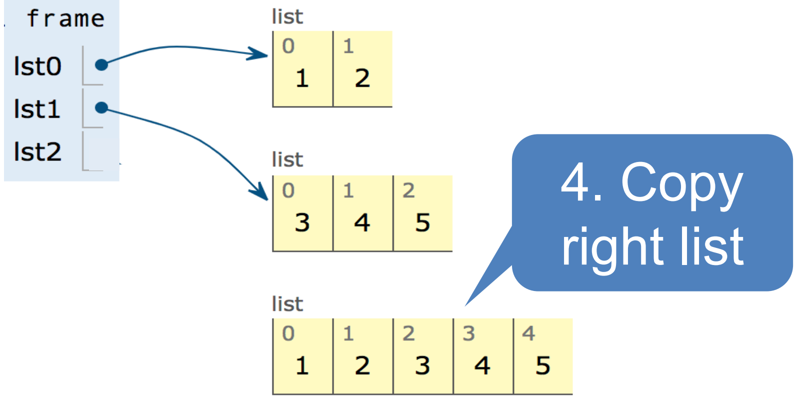


3. Copy left list

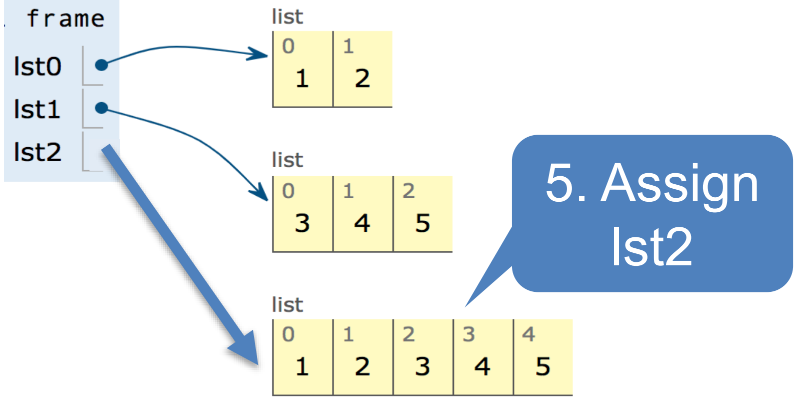
Concatenation:
length, create, copy, copy, assign



```
lst0 = [1,2]
lst1 = [3, 4, 5]
lst2 = lst0 + lst1
```

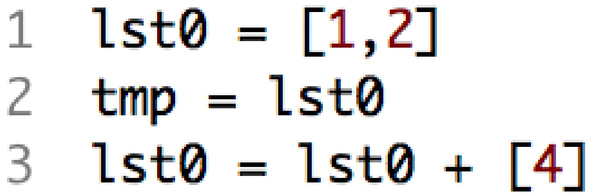


4. Copy right list



5. Assign lst2

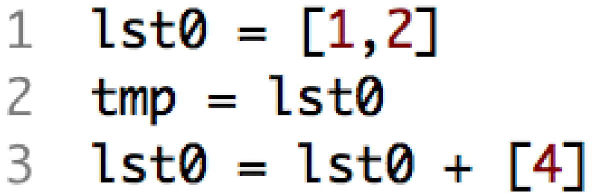
Concatenation: Makes new List



```python
lst0 = [1,2]
tmp = lst0
lst0 = lst0 + [4]
```

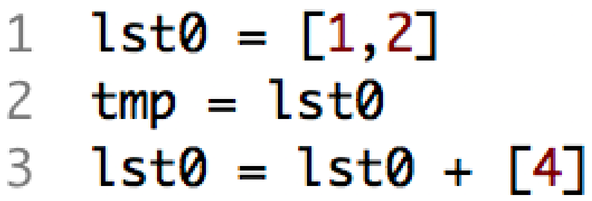
What will Python Tutor Display? How many lists will there be?

Concatenation: Makes new List

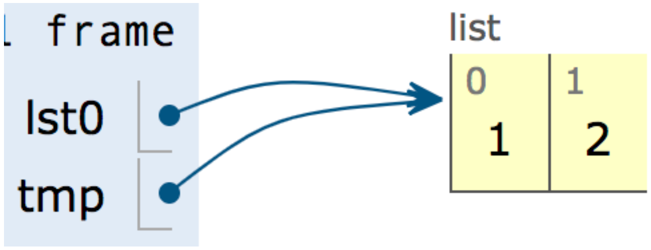


```
lst0 = [1,2]
tmp = lst0
lst0 = lst0 + [4]
```

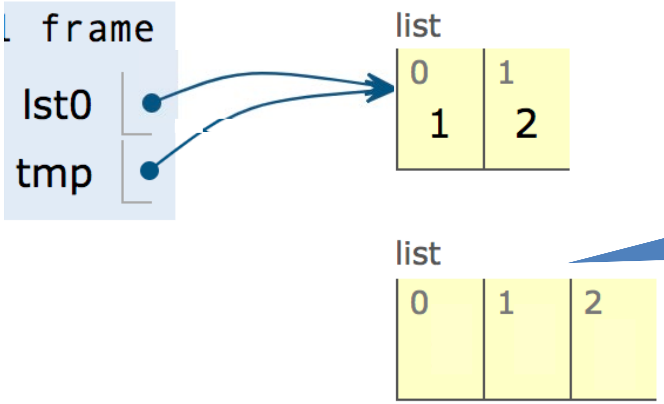
Concatenation: Makes new List



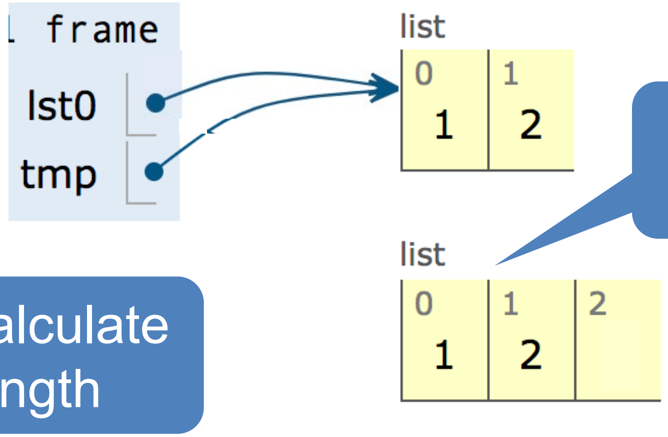
```
lst0 = [1,2]
tmp = lst0
lst0 = lst0 + [4]
```



1. Calculate length

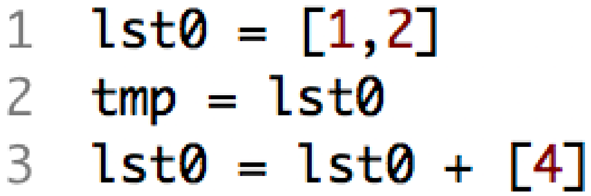


2. Create new list

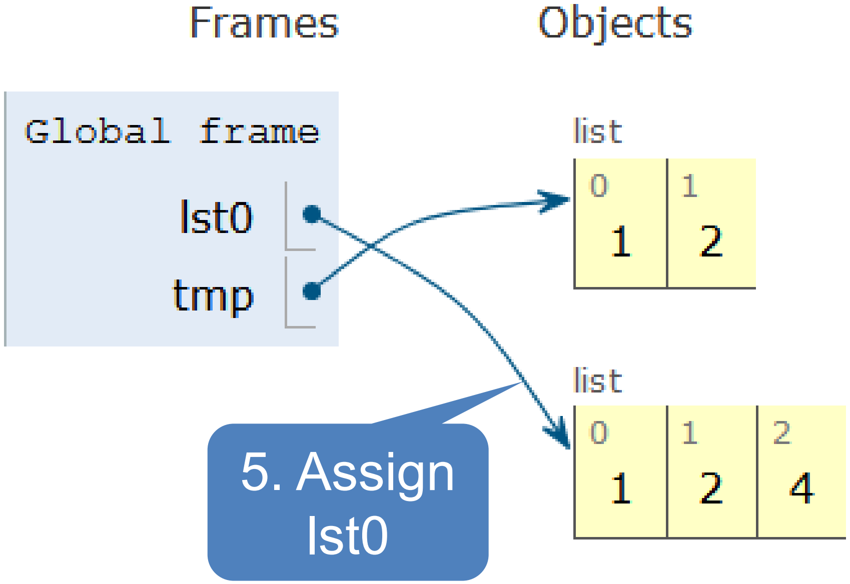


3. Copy left list

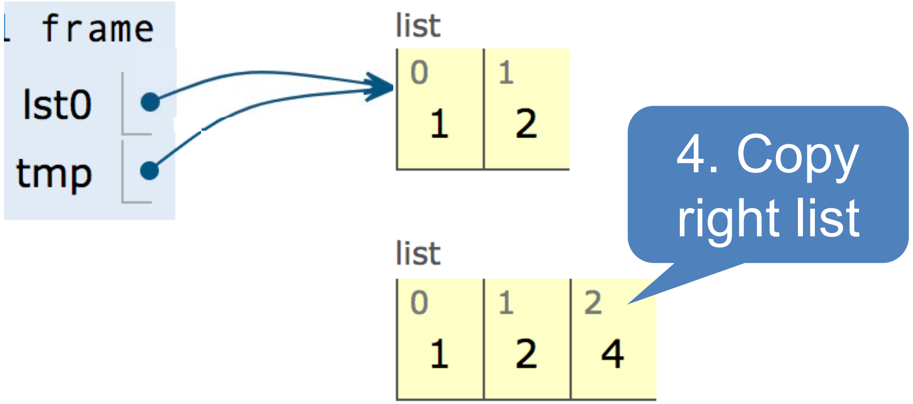
Concatenation: Makes new List



```
lst0 = [1,2]
tmp = lst0
lst0 = lst0 + [4]
```



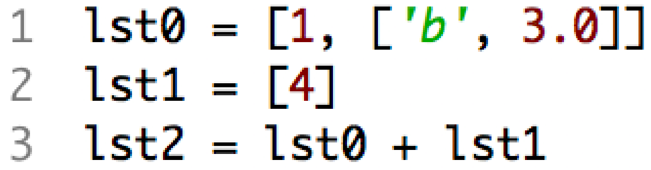
5. Assign lst0



4. Copy right list

Concatenation:
length, create, copy, copy, assign

- **How is the inner list copied?**

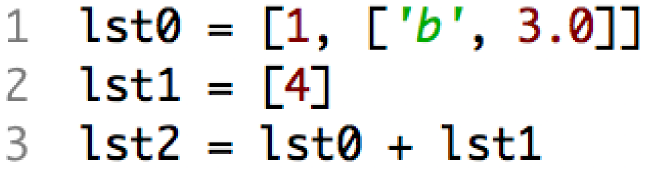


```python
lst0 = [1, ['b', 3.0]]
lst1 = [4]
lst2 = lst0 + lst1
```

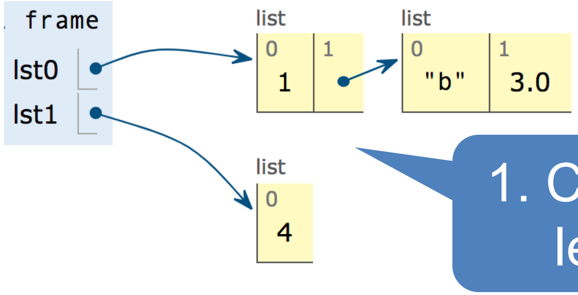
What will Python Tutor Display? How many copies of [‘b’, 3.0] will be present?

Concatenation: length, create, copy, copy, assign

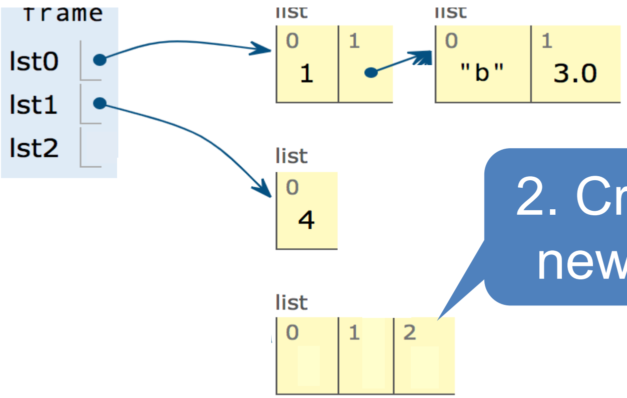
- **How is the inner list copied?**



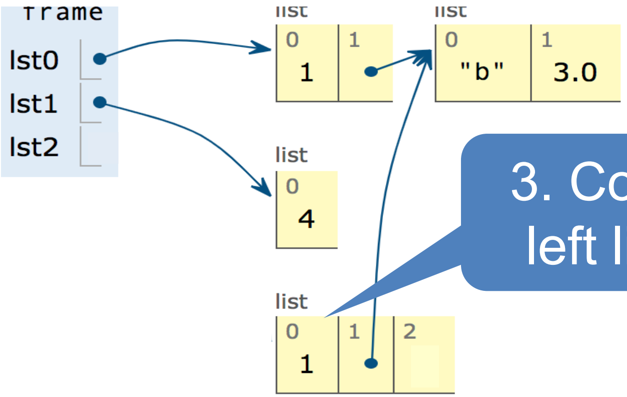
```
lst0 = [1, ['b', 3.0]]
lst1 = [4]
lst2 = lst0 + lst1
```



1. Calculate length



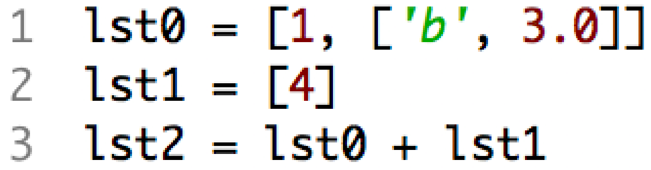
2. Create new list



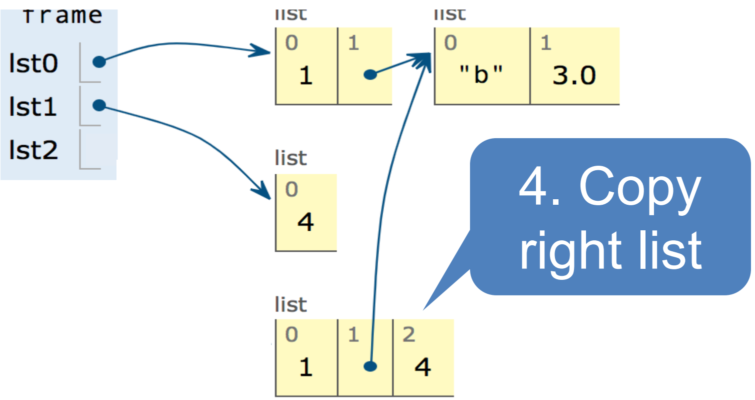
3. Copy left list

Concatenation:
length, create, copy, copy

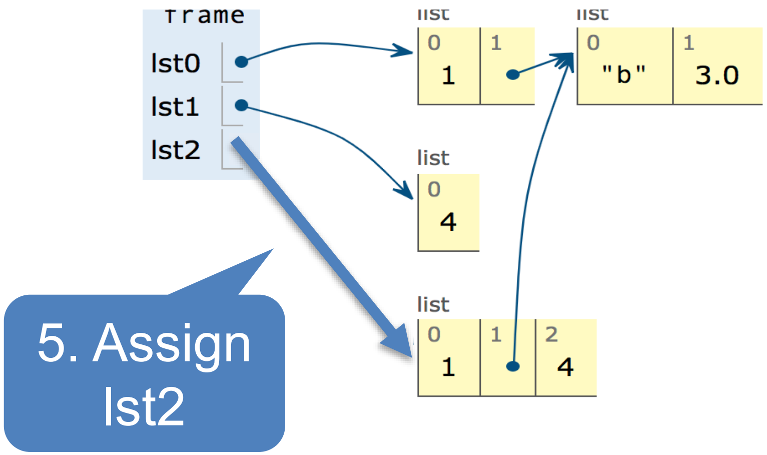
- **How is the inner list copied?**



```
lst0 = [1, ['b', 3.0]]
lst1 = [4]
lst2 = lst0 + lst1
```



4. Copy right list



5. Assign lst2

This is a shallow copy!
Don’t copy inner lists

List Mutation: .append(…)

- **`.append()` – list function that adds element to end of list**
  - Mutates list to left of "."
  - "." – call function to the right of the dot on the thing to the left of the dot (`LEFT.RIGHT`)

```
x = [6, 2, 4]
x.append(3)
x.append( [5,2] )
```

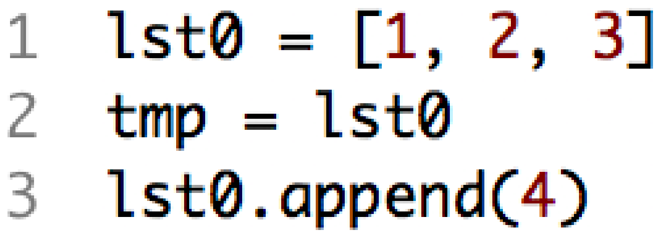
List Mutation: .append(…)

- **`.append()` – list function that adds element to end of list**
  - Mutates list to left of "."
  - "." – call function to the right of the dot on the thing to the left of the dot (`LEFT.RIGHT`)

Same list!

x = [6, 2, 4]	x is [6, 2, 4]
x.append(3)	x is [6, 2, 4, 3]
x.append([5,2])	x is [6, 2, 4, 3, [5, 2]]

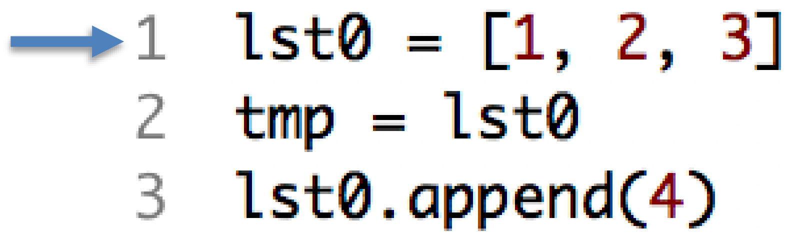
List Mutation: .append(…)



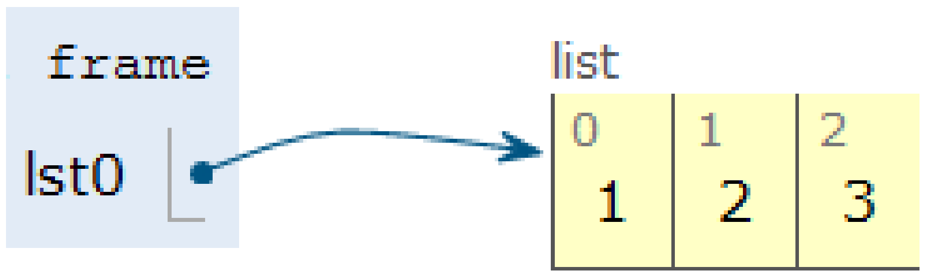
```python
lst0 = [1, 2, 3]
tmp = lst0
lst0.append(4)
```

What will Python Tutor Display? One or two lists?

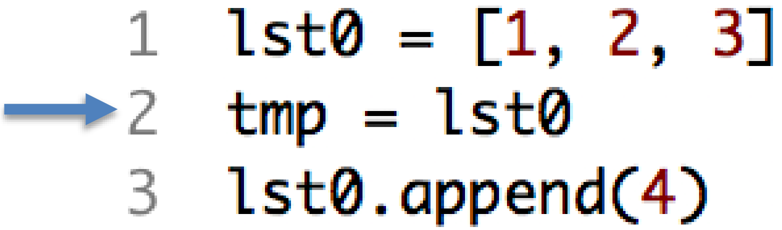
List Mutation: .append(...)



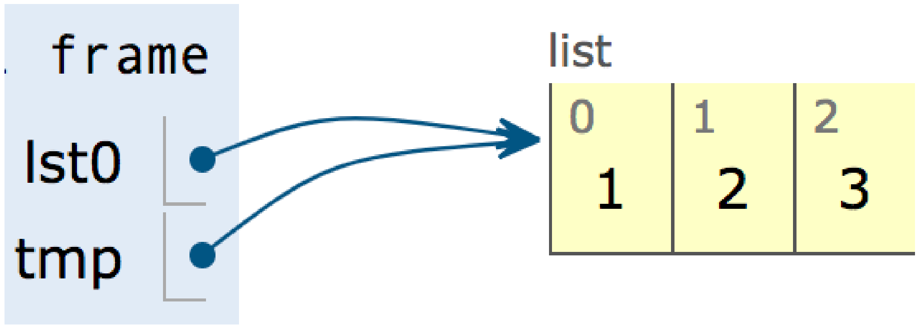
```python
lst0 = [1, 2, 3]
tmp = lst0
lst0.append(4)
```



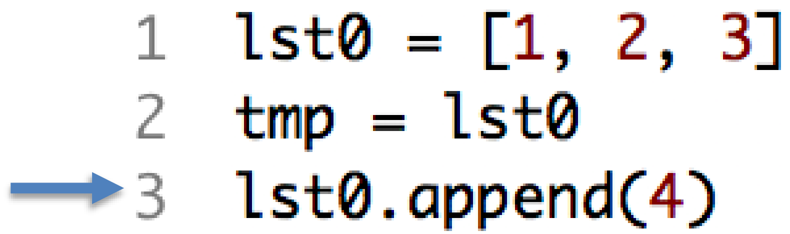
List Mutation: .append(...)



```
lst0 = [1, 2, 3]
tmp = lst0
lst0.append(4)
```

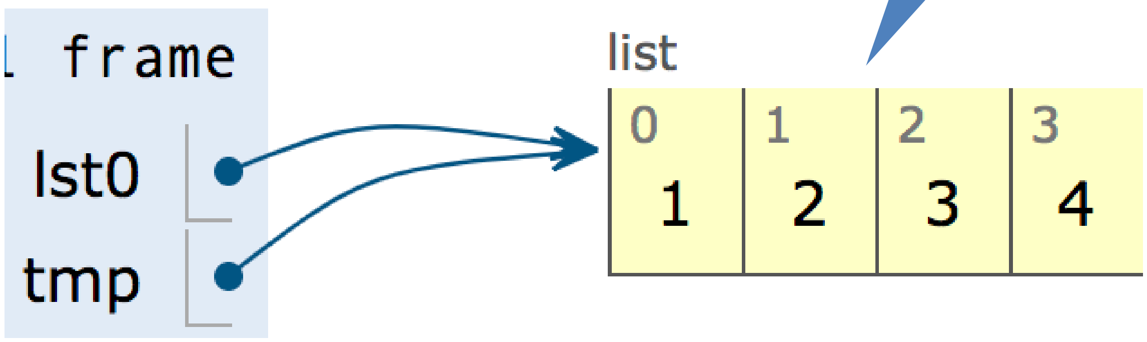


List Mutation: .append(...)

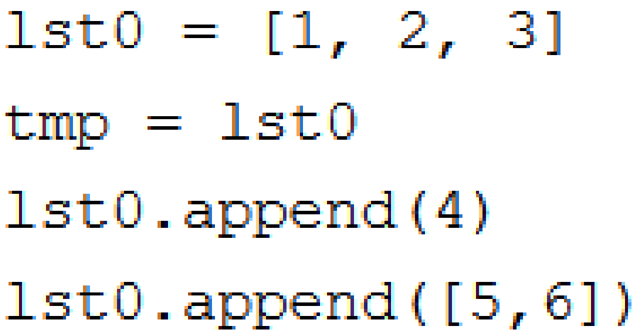


```
lst0 = [1, 2, 3]
tmp = lst0
lst0.append(4)
```

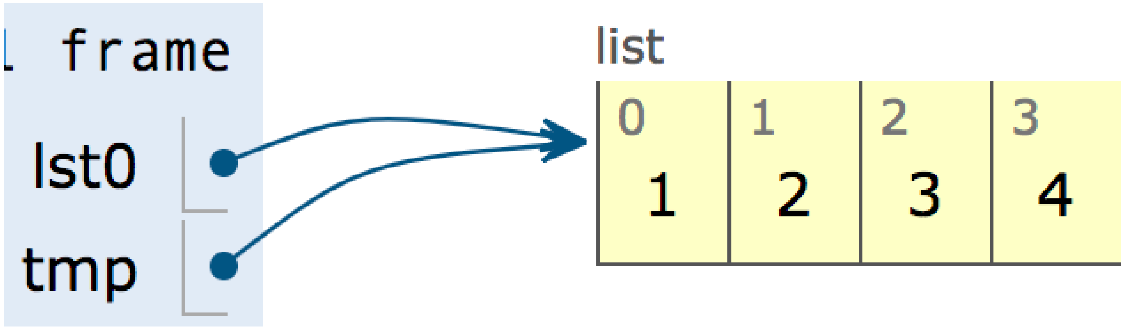
Same list!
No new list



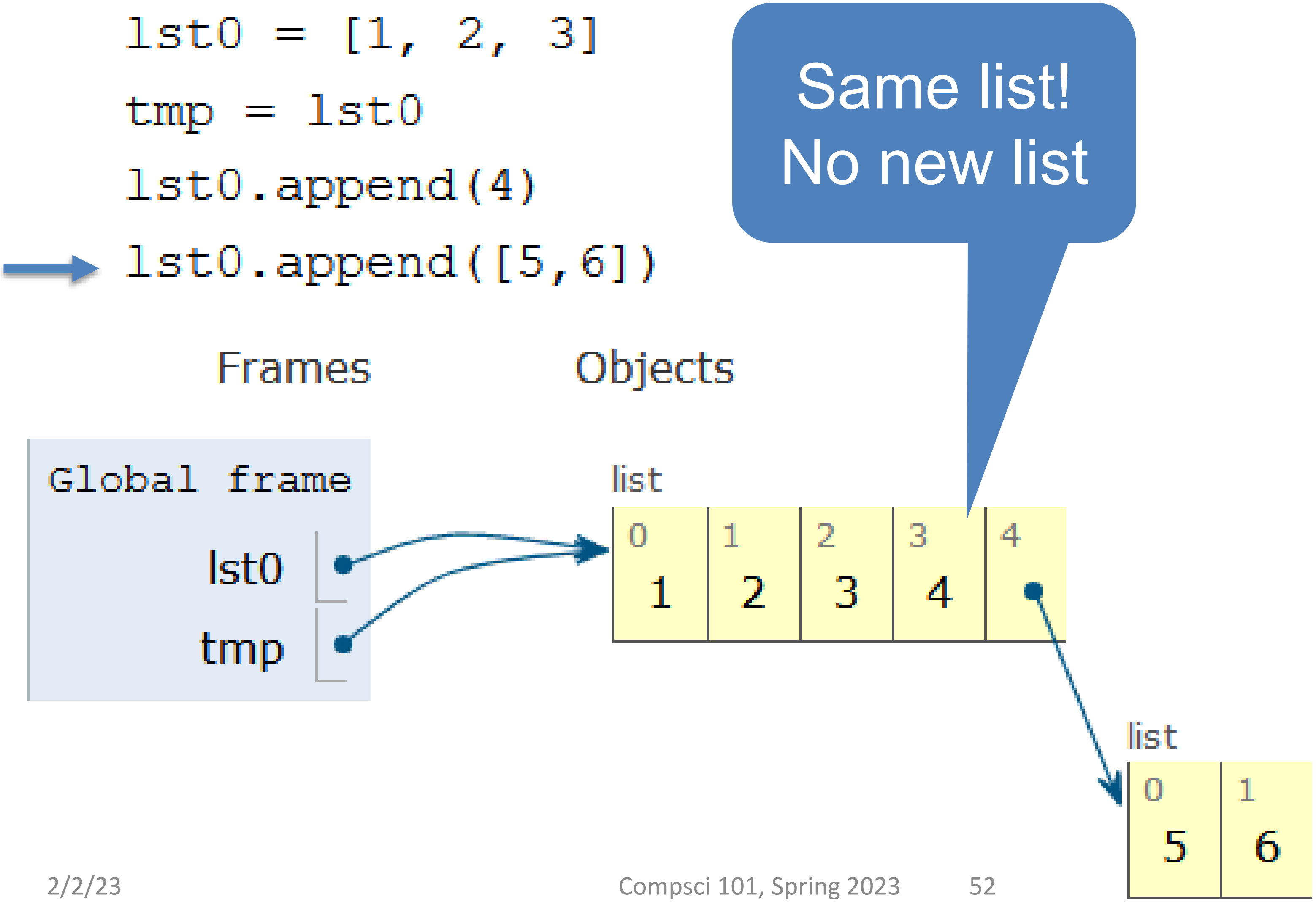
List Mutation: .append(...)



```
lst0 = [1, 2, 3]
tmp = lst0
lst0.append(4)
lst0.append([5,6])
```



List Mutation: .append(...)

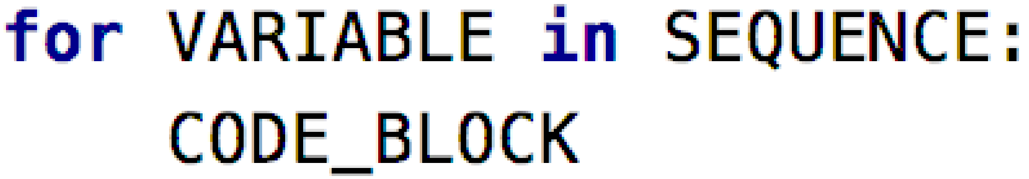


```
lst0 = [1, 2, 3]
tmp = lst0
lst0.append(4)
lst0.append([5,6])
```

Same list!
No new list

WOTO-2 – Mutable and Append
http://bit.ly/101s23-0202-2

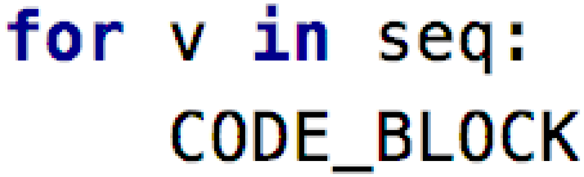
Anatomy of a `for` loop



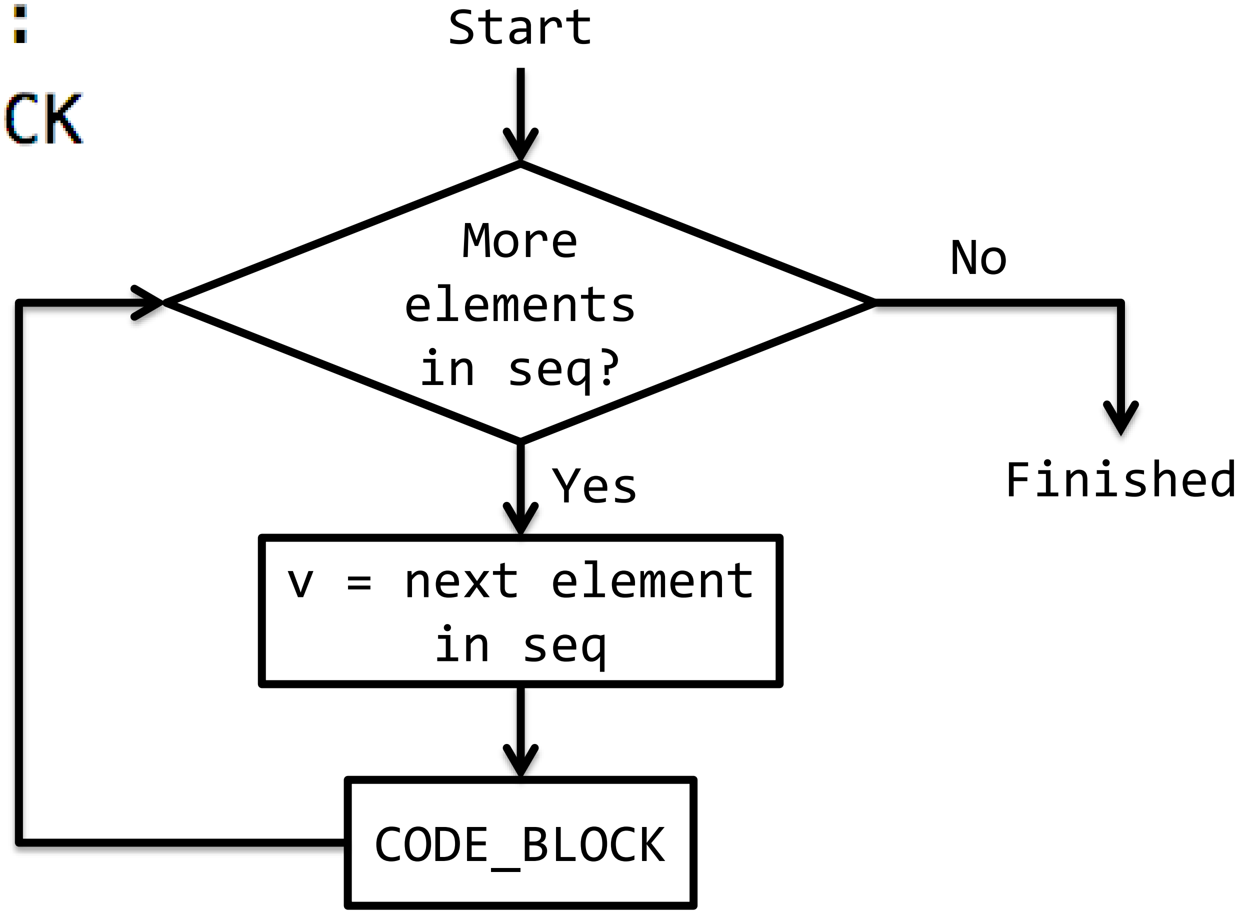
```
for VARIABLE in SEQUENCE:
    CODE_BLOCK
```

- **Think of as:**
  - “For each element in the SEQUENCE put it in the VARIABLE and execute the CODE_BLOCK.”
  - Also called: *Iterate* over the sequence
- **What type(s) are sequences?**
  - Strings, Lists
- **Will VARIABLE likely be in CODE_BLOCK?**

Anatomy of a for loop



```
for v in seq:
    CODE_BLOCK
```



Start

More elements in seq?

No

Finished

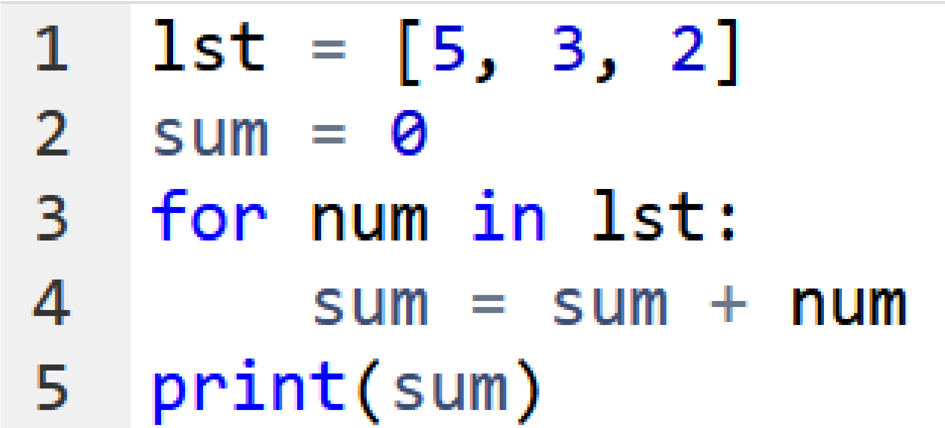
Yes

v = next element in seq

CODE_BLOCK

Example for loop with a list

- What does this for loop do?



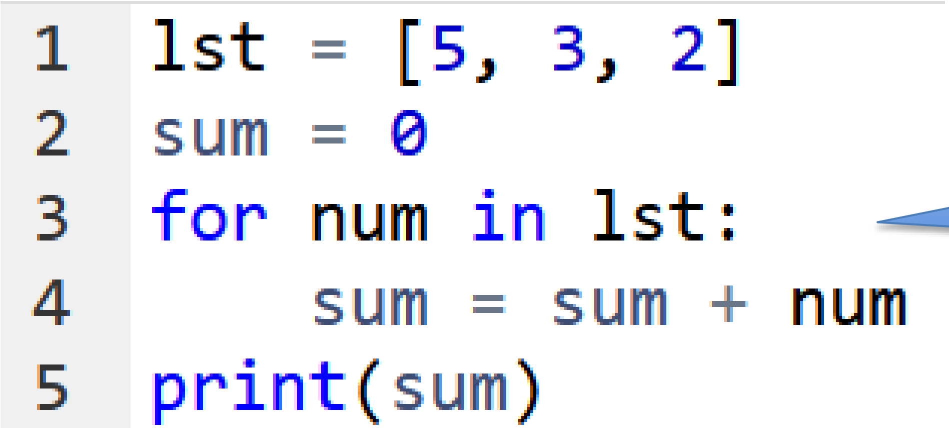
```python
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```

- What is first value of **num**?

- What is final value of **num**?

Example for loop with a list

- What does this for loop do?



```
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```

Adds the numbers in the list

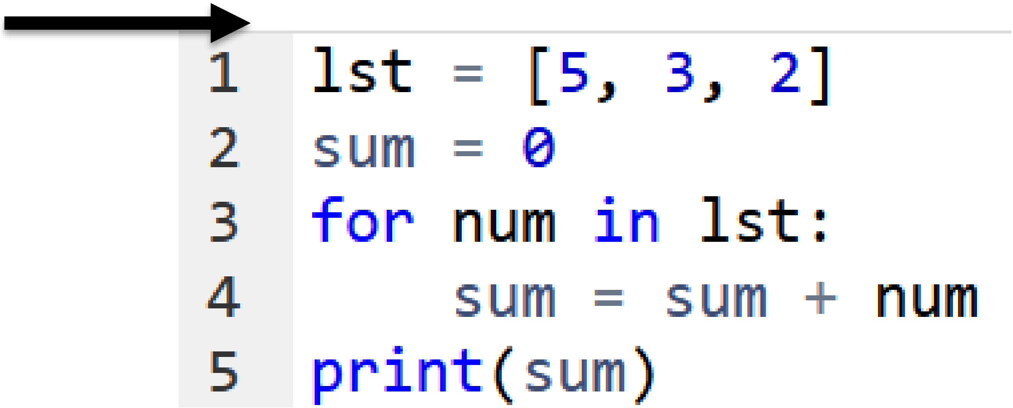
- What is first value of **num**?

  5

- What is final value of **num**?

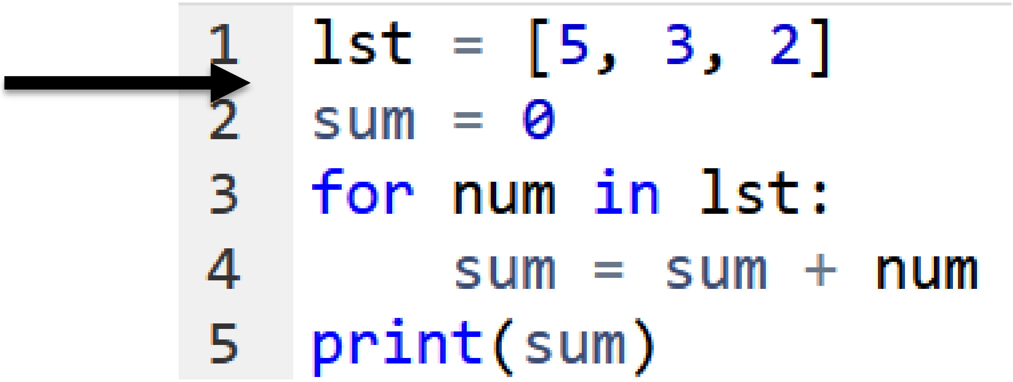
  2

Trace through for loop

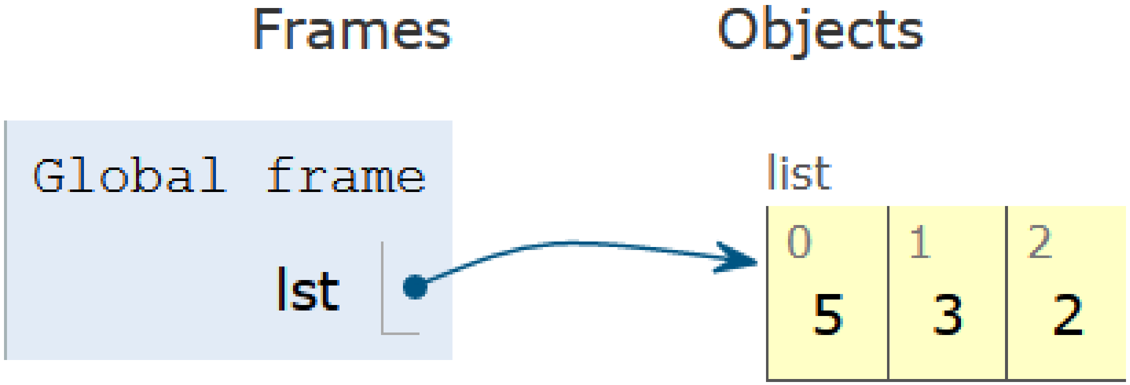


```python
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```

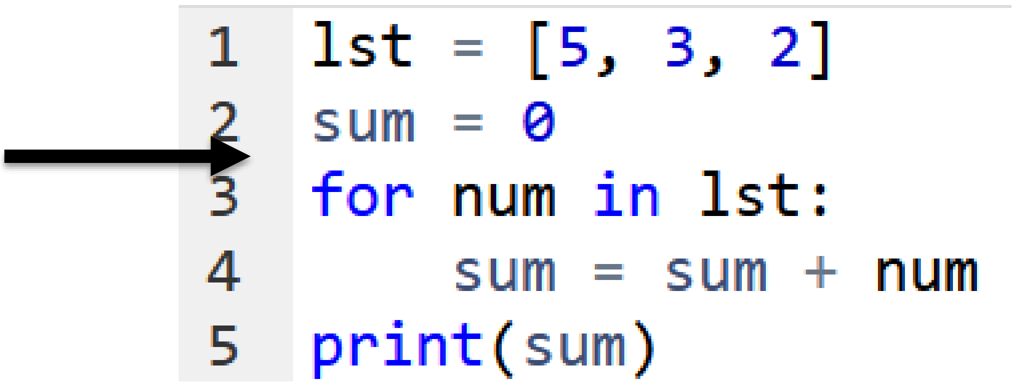
Trace through for loop



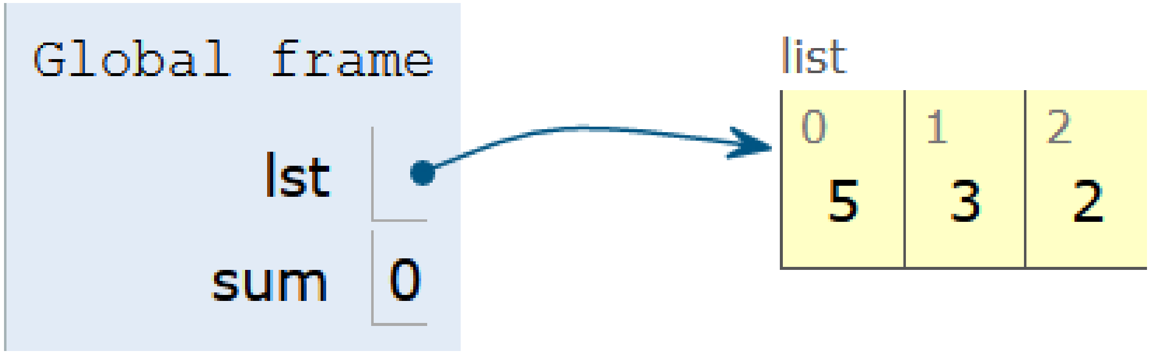
```python
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```



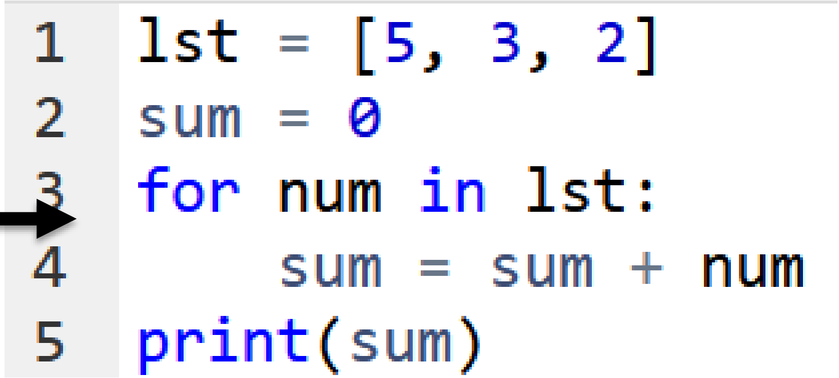
Trace through for loop



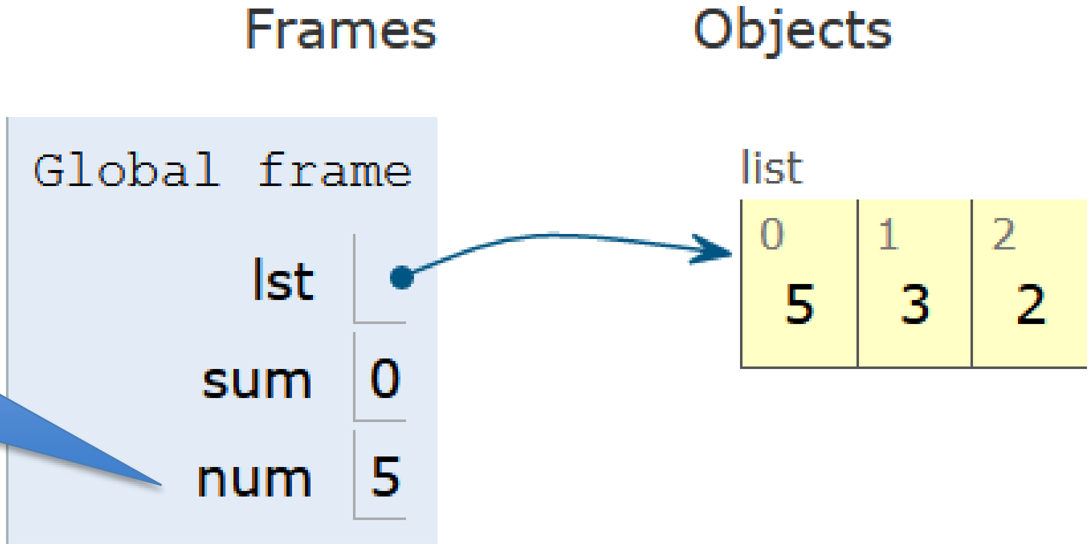
```python
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```



Trace through for loop

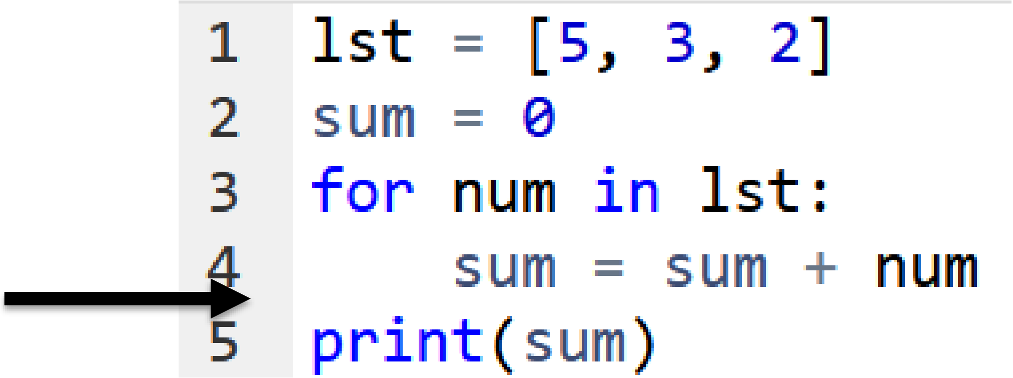


```
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```

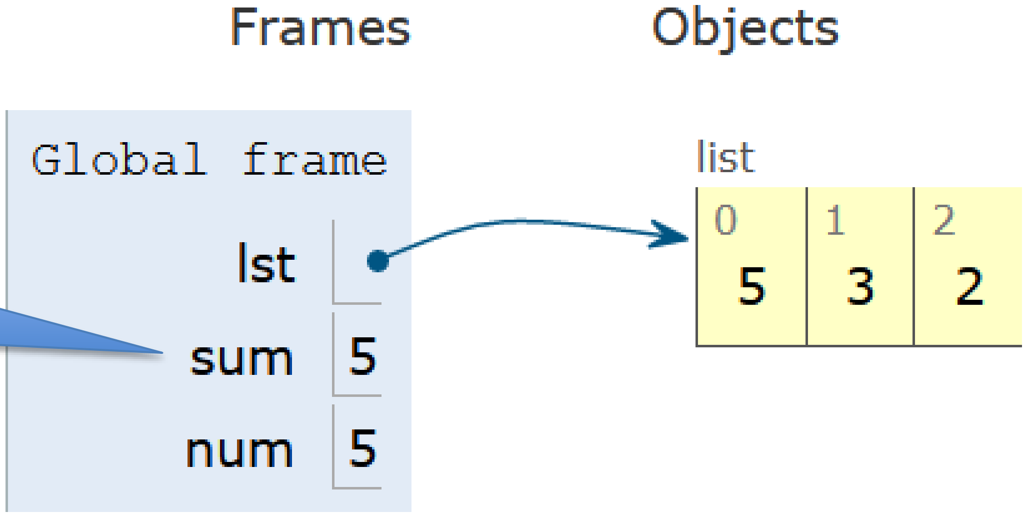


num gets first value in list

Trace through for loop

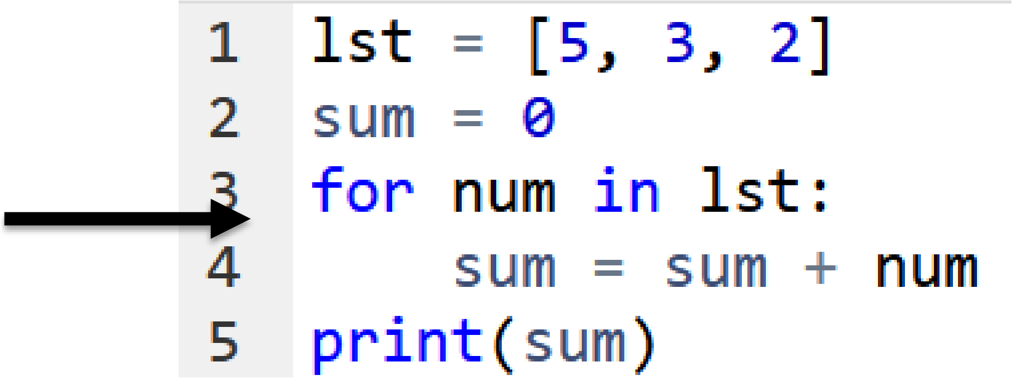


```
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```

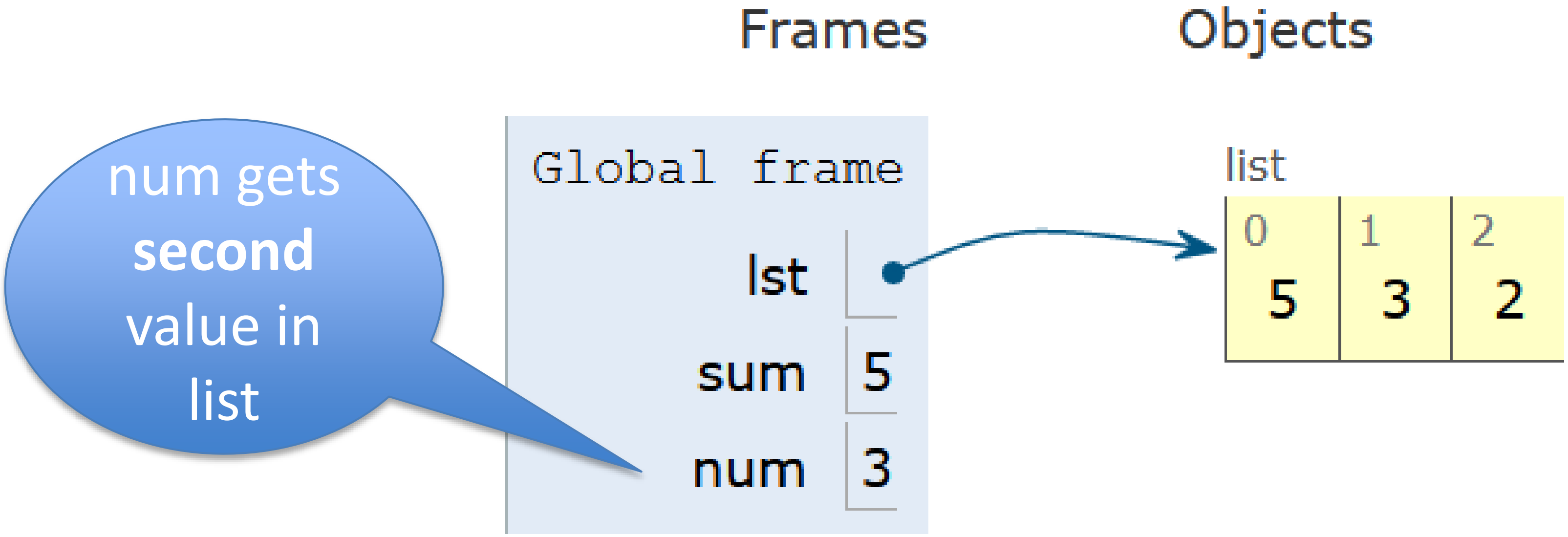


Add num to sum

Trace through for loop

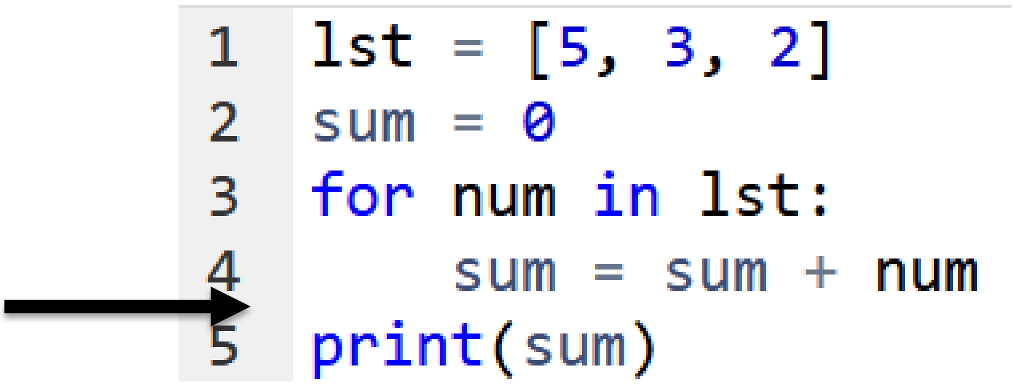


```python
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```

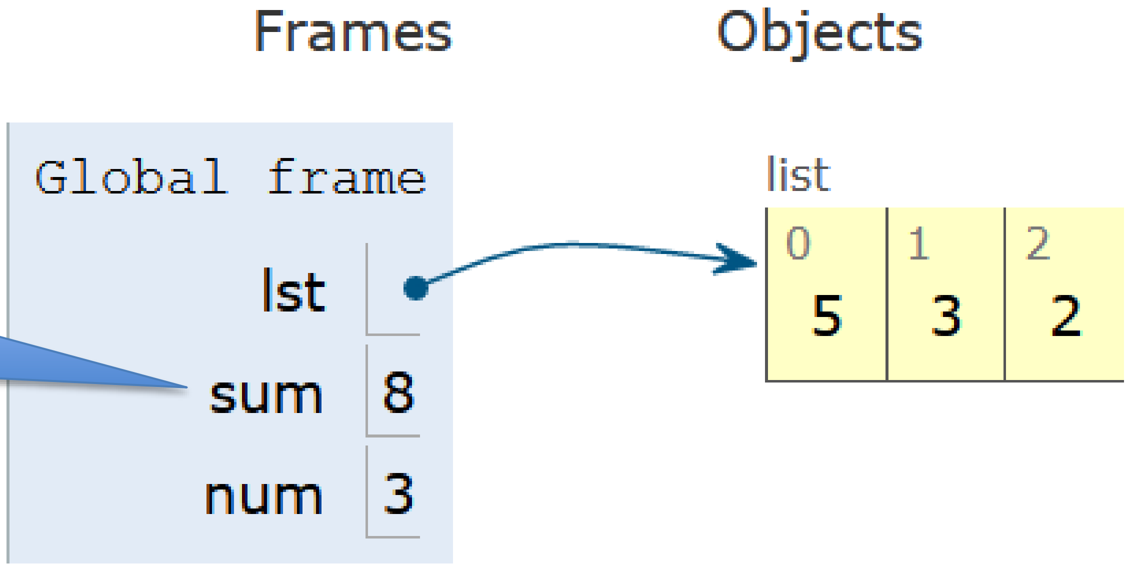


num gets **second** value in list

Trace through for loop

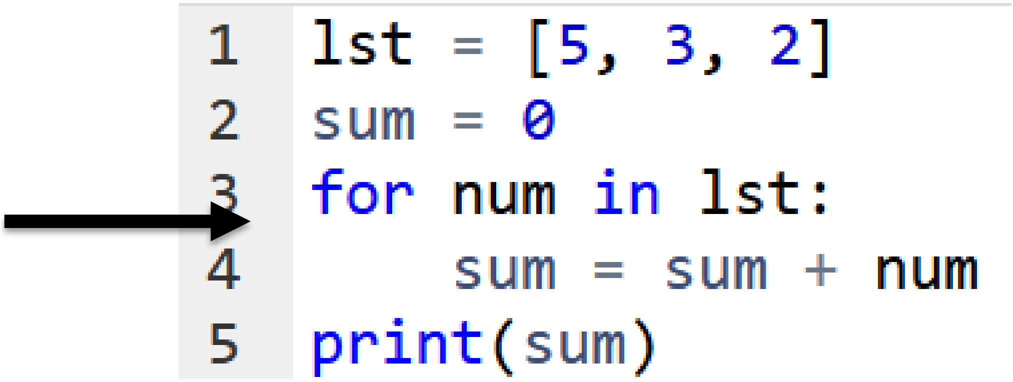


```python
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```



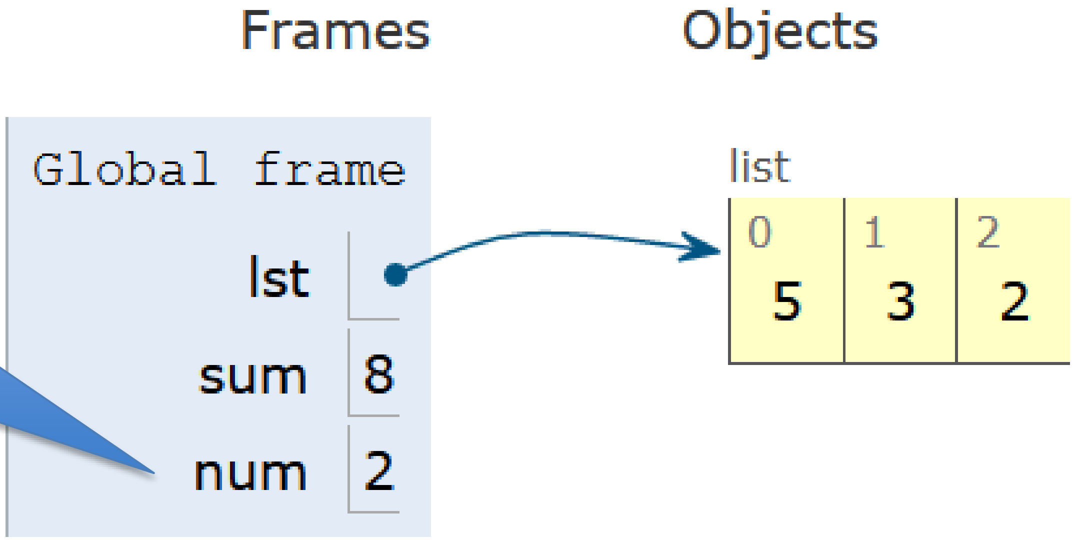
Add num to sum

Trace through for loop

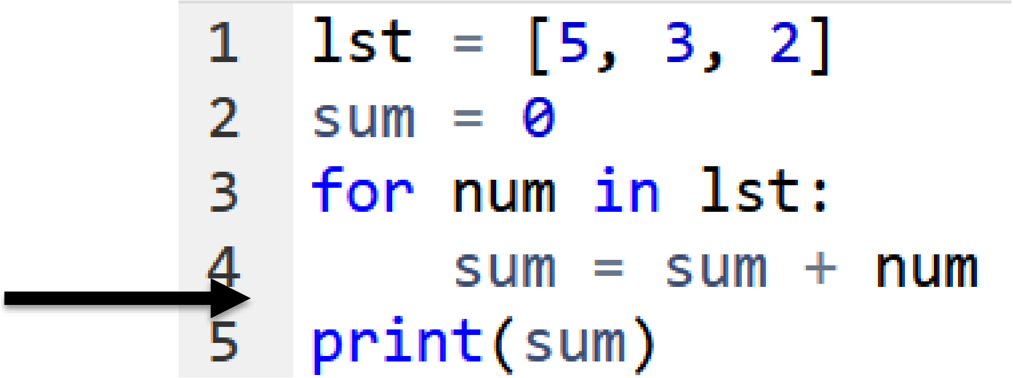


```
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```

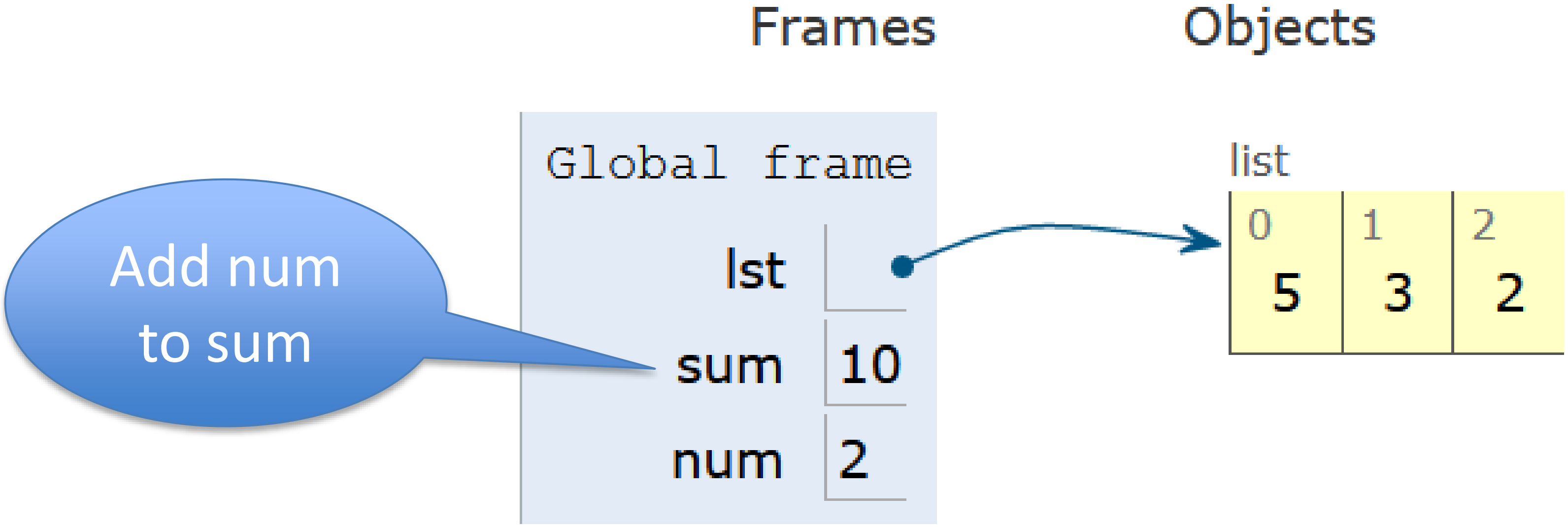
num gets **third** value in list



Trace through for loop

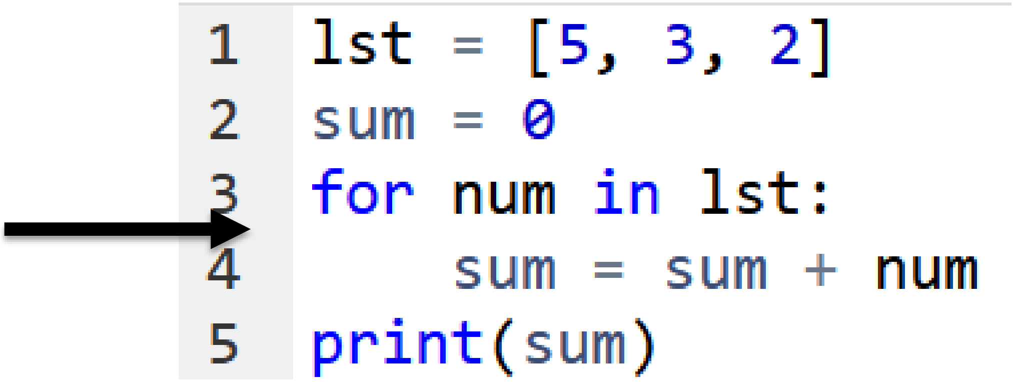


```python
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```

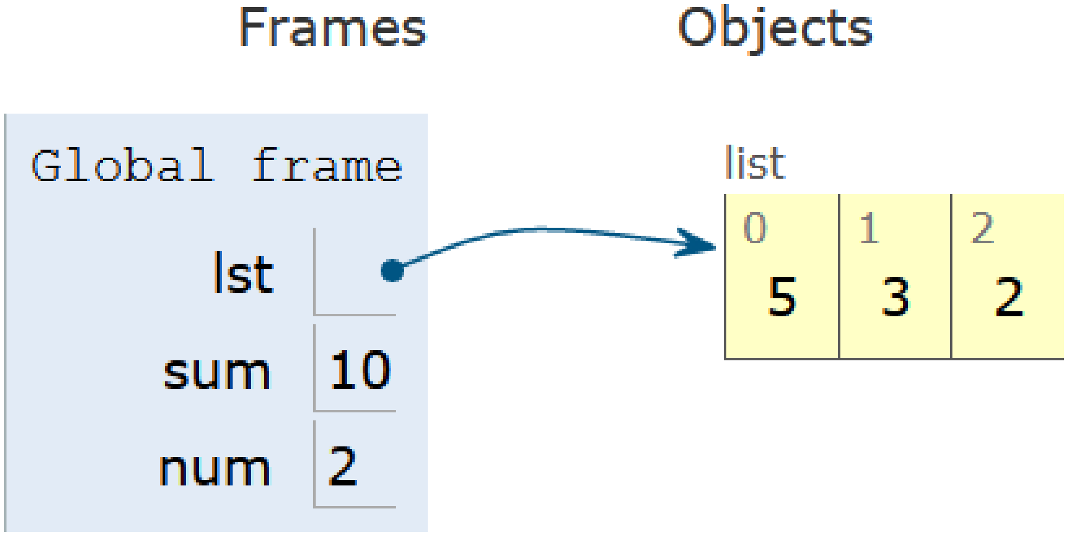


Add num to sum

Trace through for loop



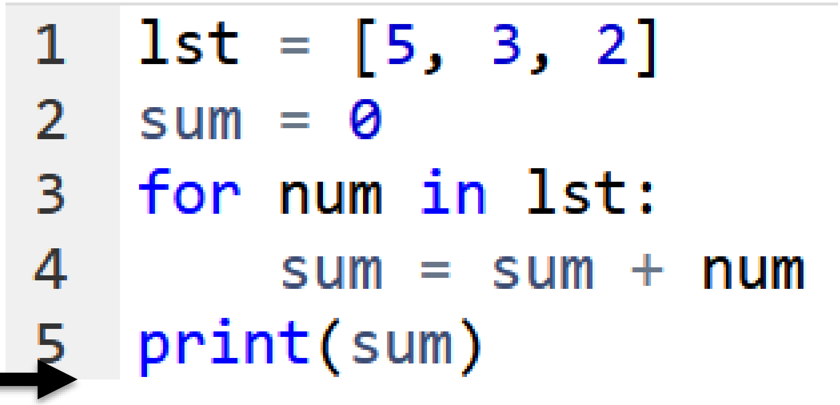
```
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```



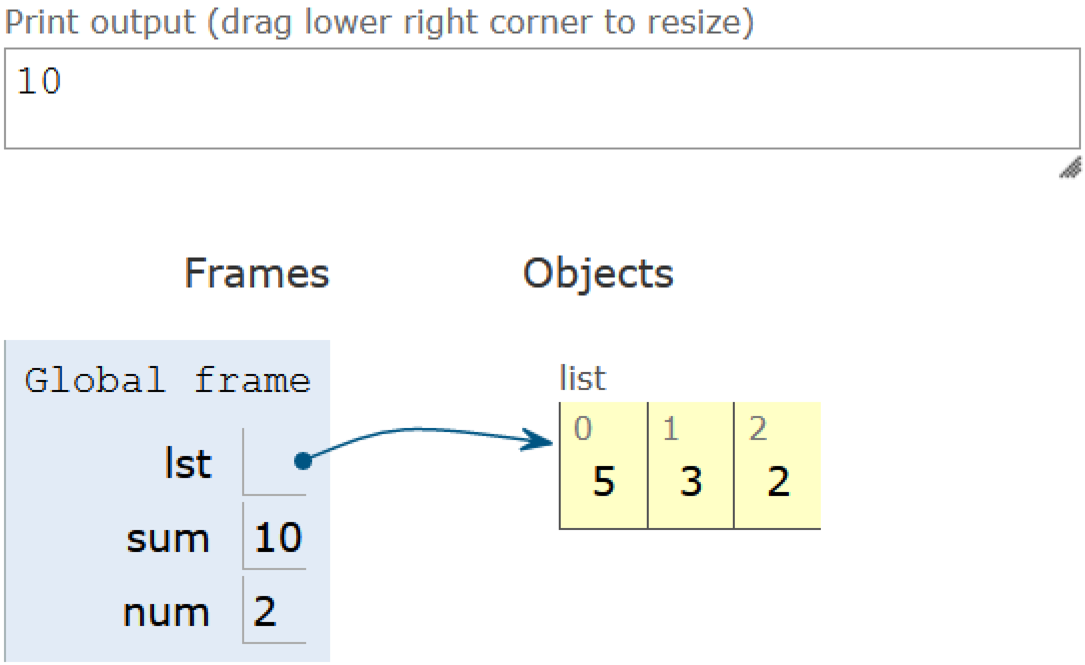
No more values in lst

The for loop is done!

Trace through for loop



```
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```



Print output (drag lower right corner to resize)

```
10
```

Frames

Objects

Global frame

lst	●
sum	10
num	2

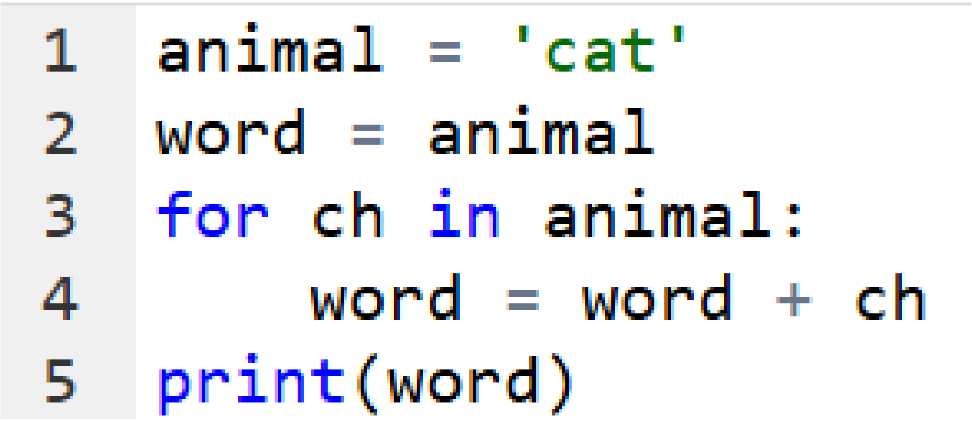
list

0	1	2
5	3	2

Print result

Example for loop with a string

- What does this for loop do?



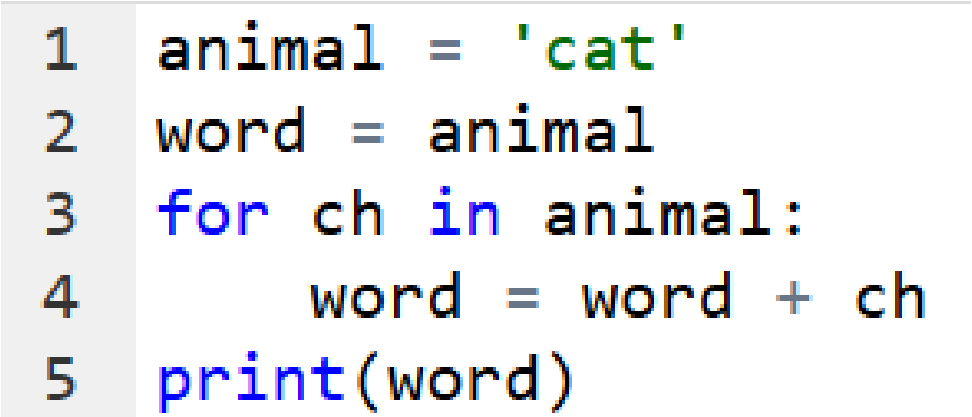
```python
animal = 'cat'
word = animal
for ch in animal:
    word = word + ch
print(word)
```

- What is first value of **ch**?

- What is final value of **ch**?

Example for loop with a string

- What does this for loop do?



```python
animal = 'cat'
word = animal
for ch in animal:
    word = word + ch
print(word)
```

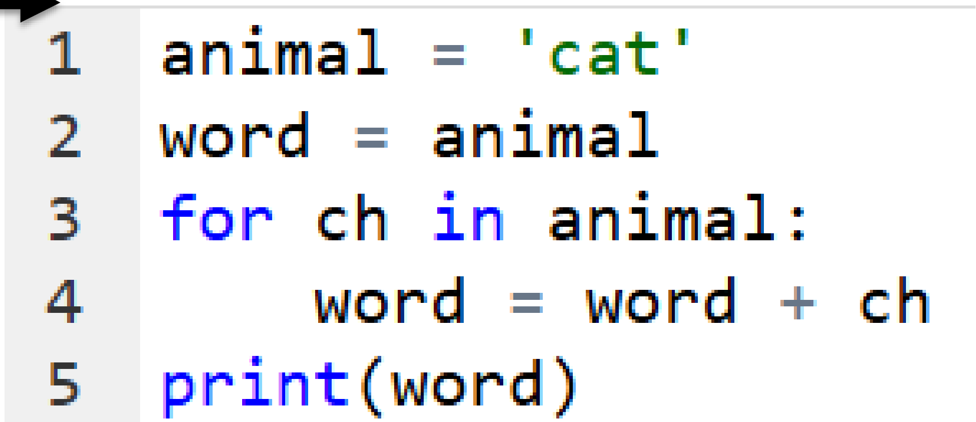
- What is first value of **ch**?

  ‘c’

- What is final value of **ch**?

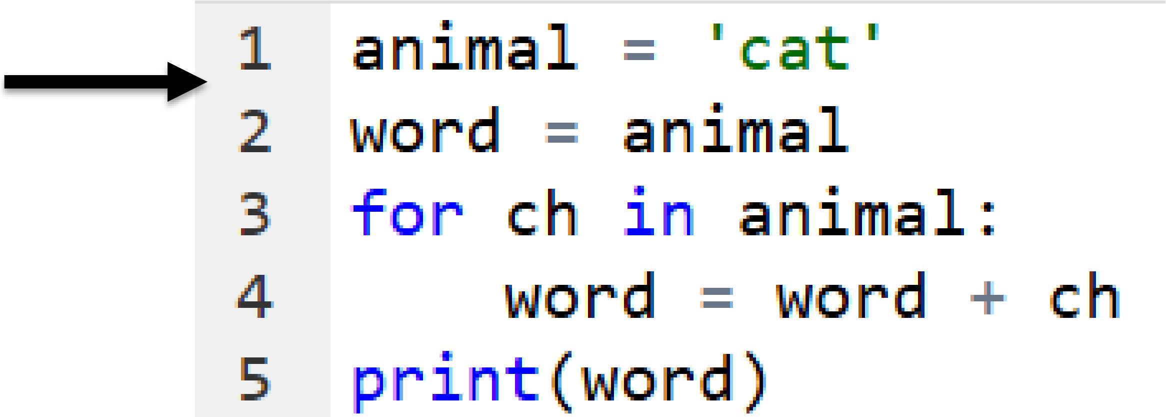
  ‘t’

Trace through for loop

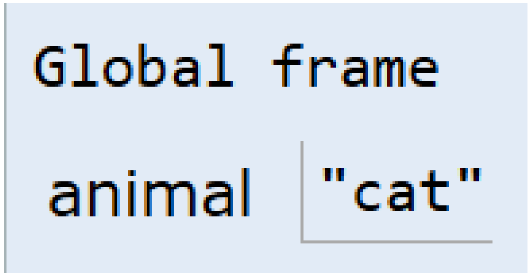


```
animal = 'cat'
word = animal
for ch in animal:
    word = word + ch
print(word)
```

Trace through for loop

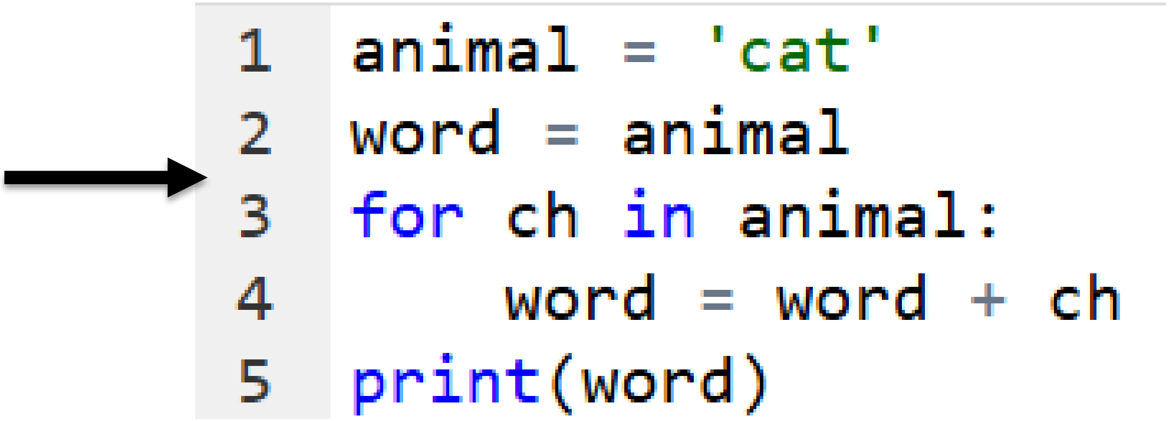


```python
animal = 'cat'
word = animal
for ch in animal:
    word = word + ch
print(word)
```

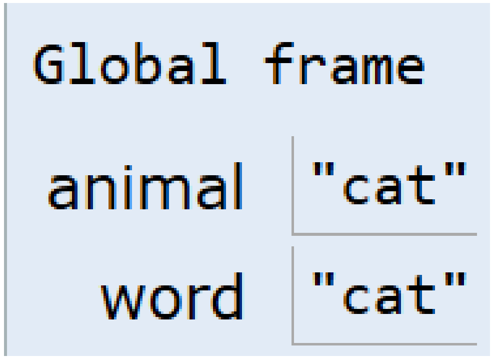


Global frame	
animal	"cat"

Trace through for loop



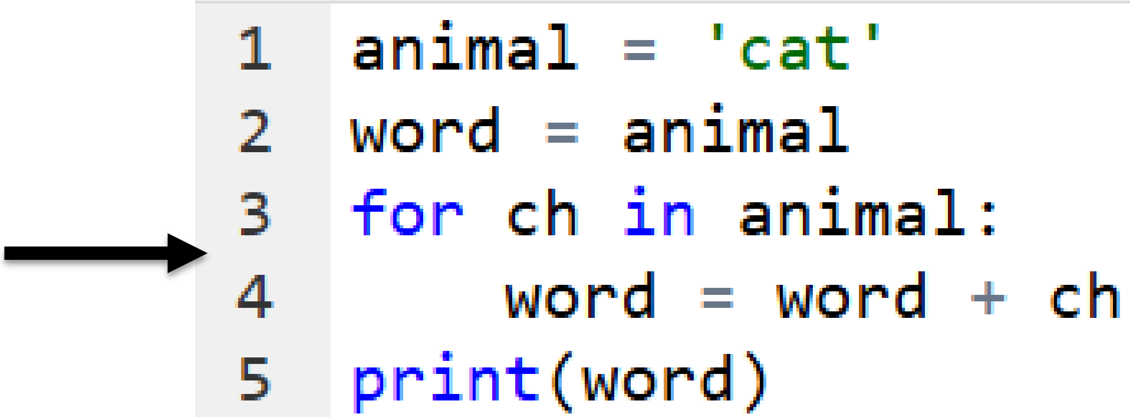
```python
animal = 'cat'
word = animal
for ch in animal:
    word = word + ch
print(word)
```



Global frame

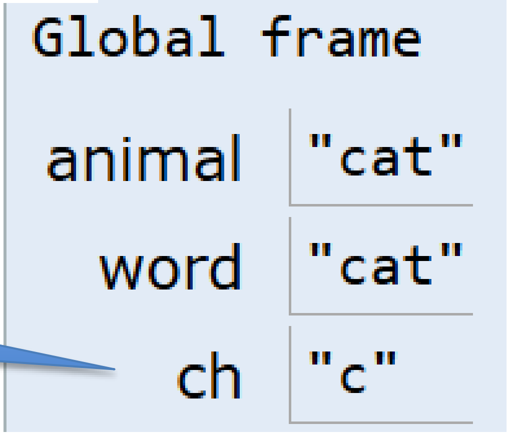
animal	"cat"
word	"cat"

Trace through for loop



```
animal = 'cat'
word = animal
for ch in animal:
    word = word + ch
print(word)
```

Iterate over copy of word: ‘c’ ‘a’ ‘t’

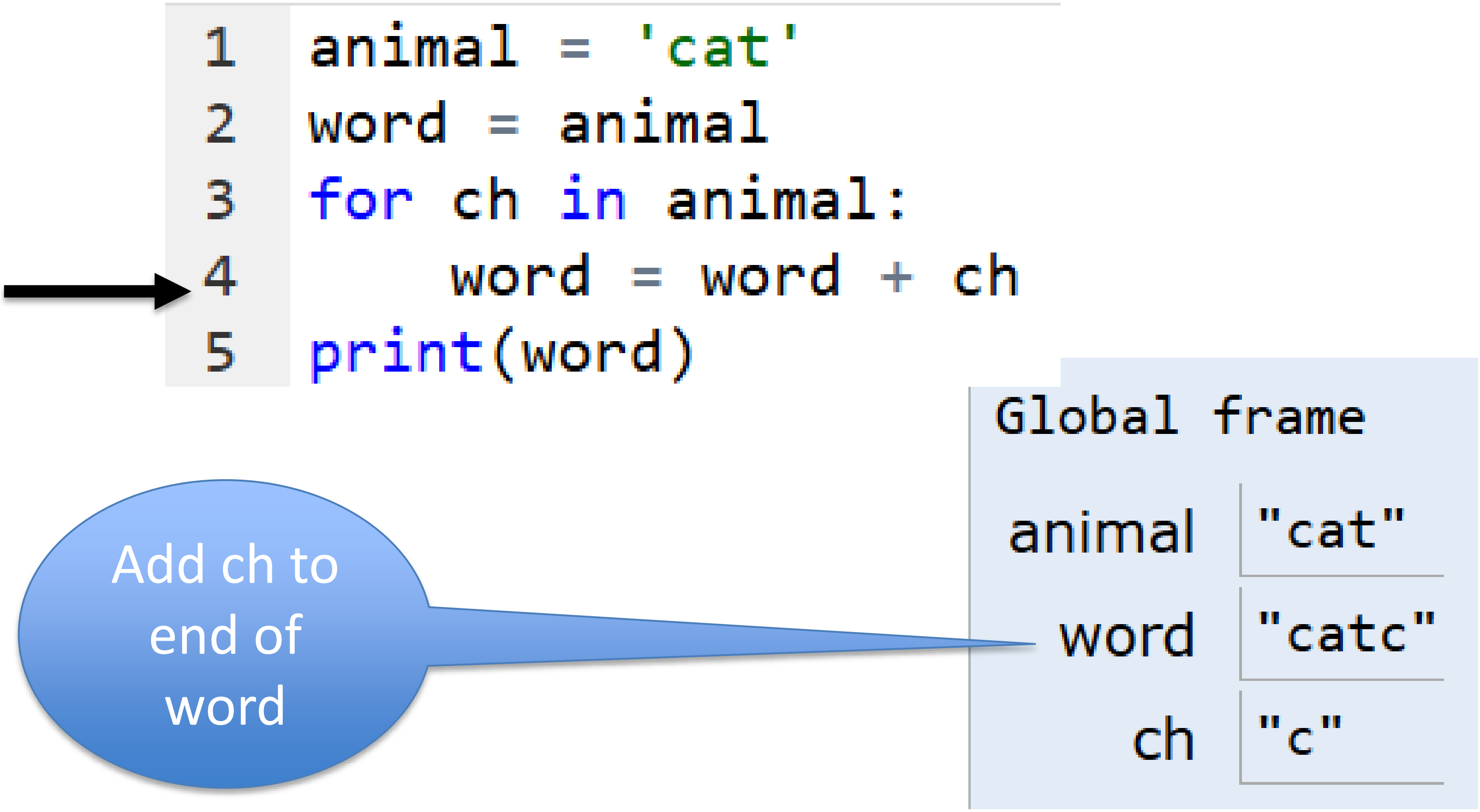


Global frame

animal	"cat"
word	"cat"
ch	"c"

ch gets first character in word

Trace through for loop



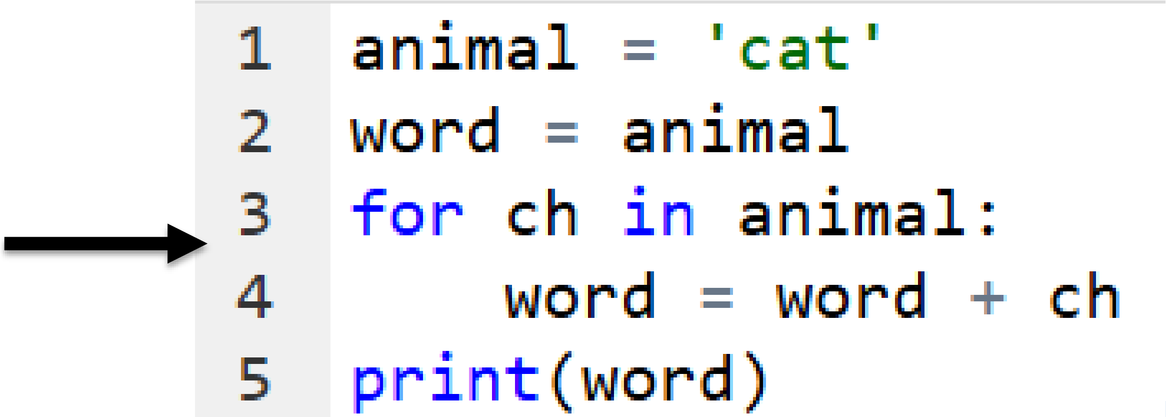
```python
animal = 'cat'
word = animal
for ch in animal:
    word = word + ch
print(word)
```

Global frame

animal	"cat"
word	"catc"
ch	"c"

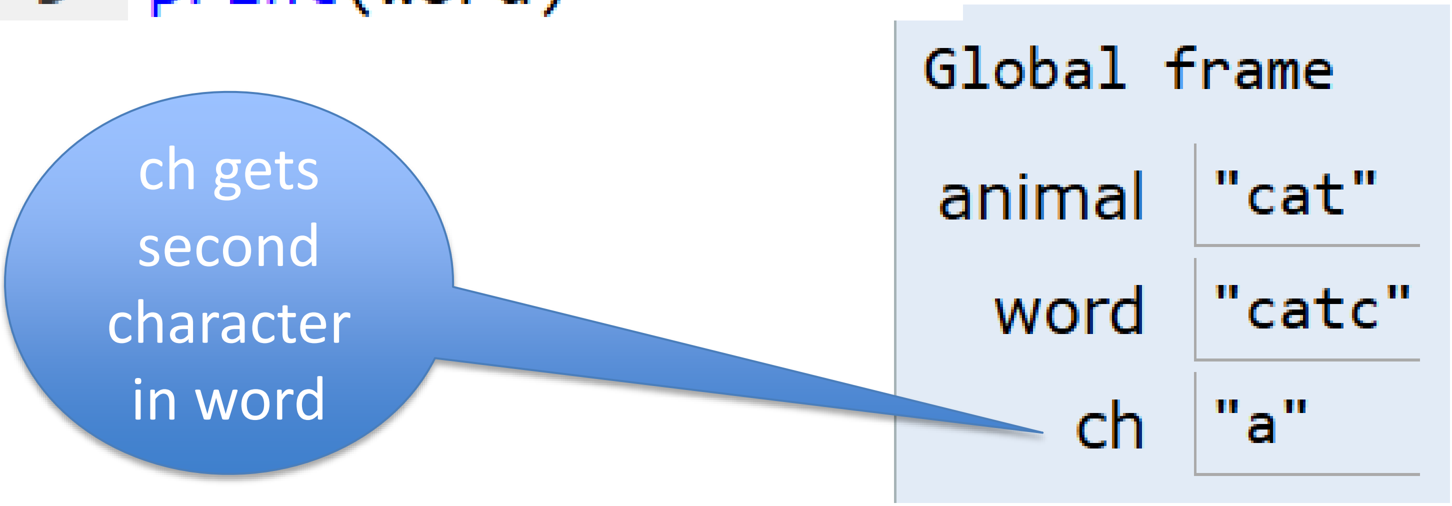
Add ch to end of word

Trace through for loop



```
animal = 'cat'
word = animal
for ch in animal:
    word = word + ch
print(word)
```

Iterate over what is left in copy of word: ‘a’ ‘t’



ch gets second character in word

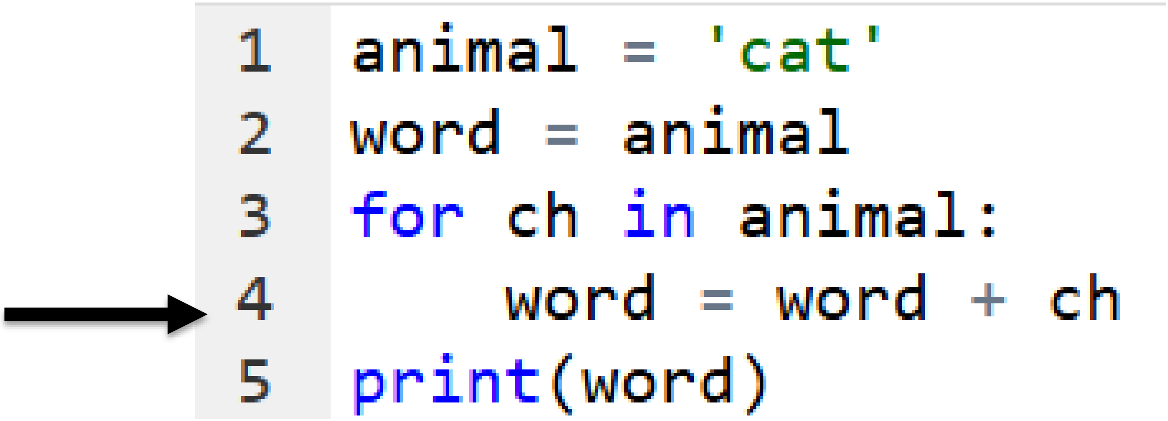
Global frame

animal "cat"

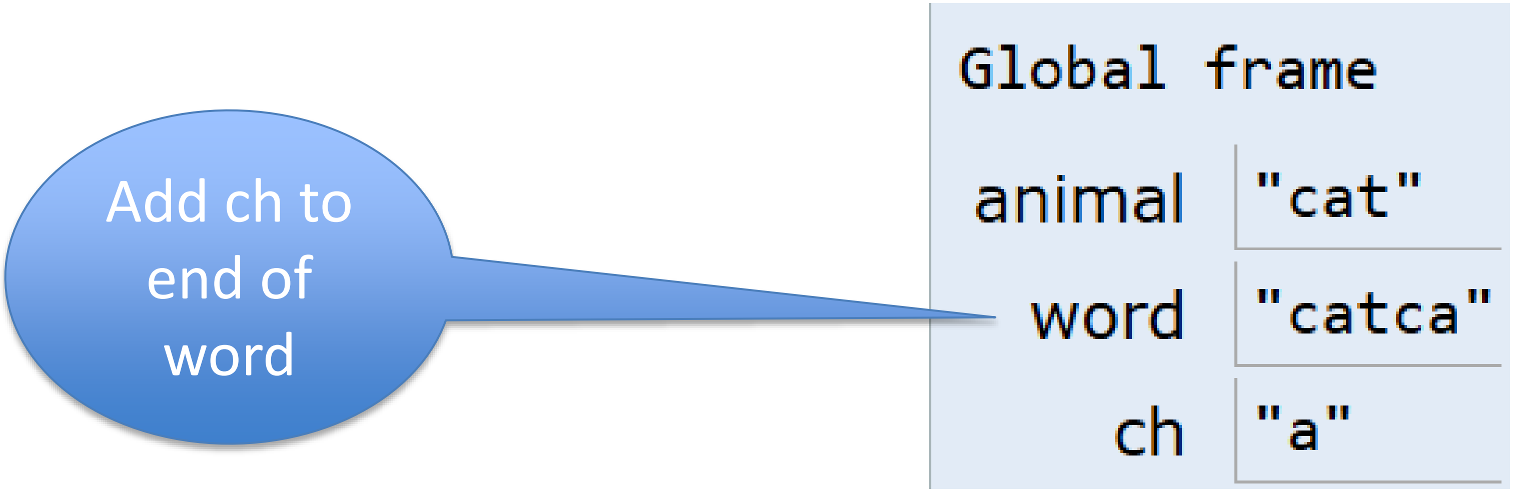
word "catc"

ch "a"

Trace through for loop



```python
animal = 'cat'
word = animal
for ch in animal:
    word = word + ch
print(word)
```

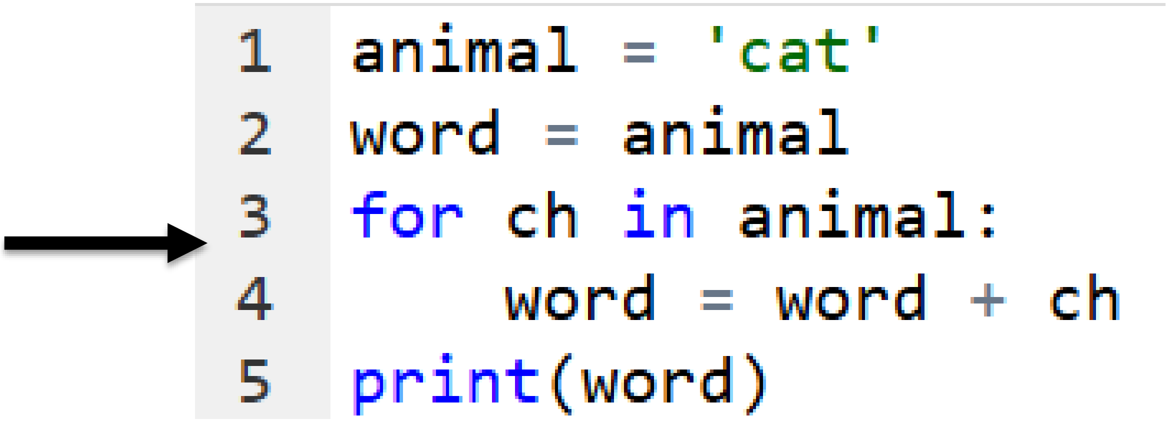


Add ch to end of word

Global frame

animal	"cat"
word	"catca"
ch	"a"

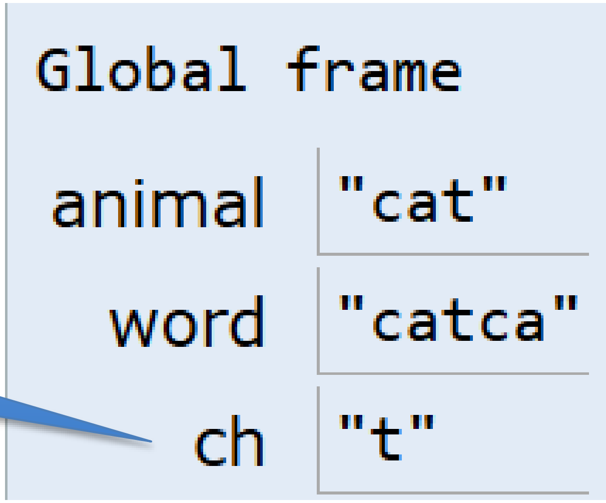
Trace through for loop



```python
animal = 'cat'
word = animal
for ch in animal:
    word = word + ch
print(word)
```

Iterate over what is left in copy of word: ‘t’

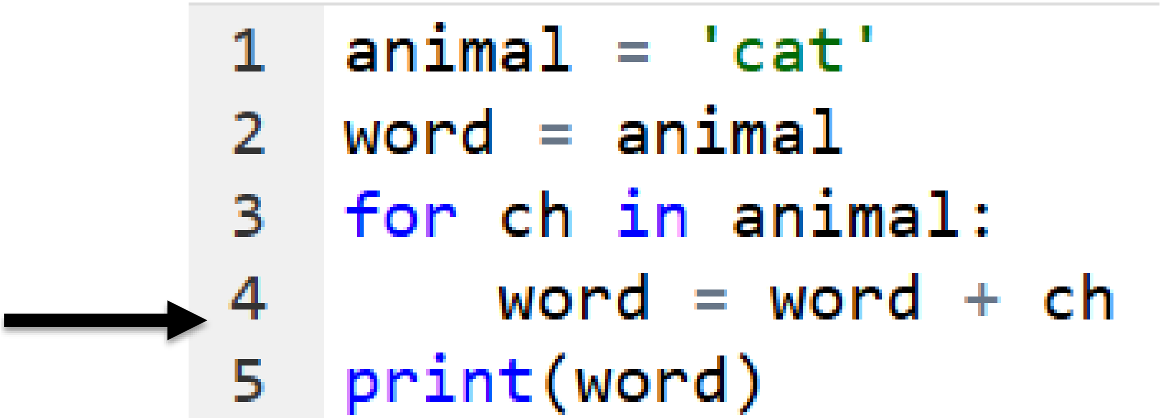
ch gets third character in word



Global frame

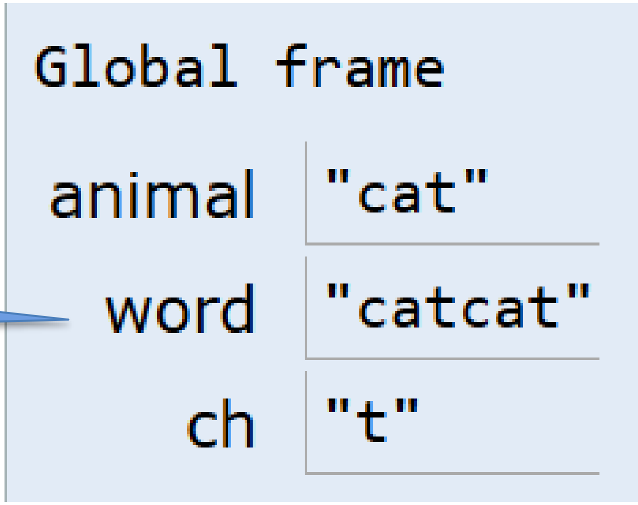
animal	"cat"
word	"catca"
ch	"t"

Trace through for loop



```python
animal = 'cat'
word = animal
for ch in animal:
    word = word + ch
print(word)
```

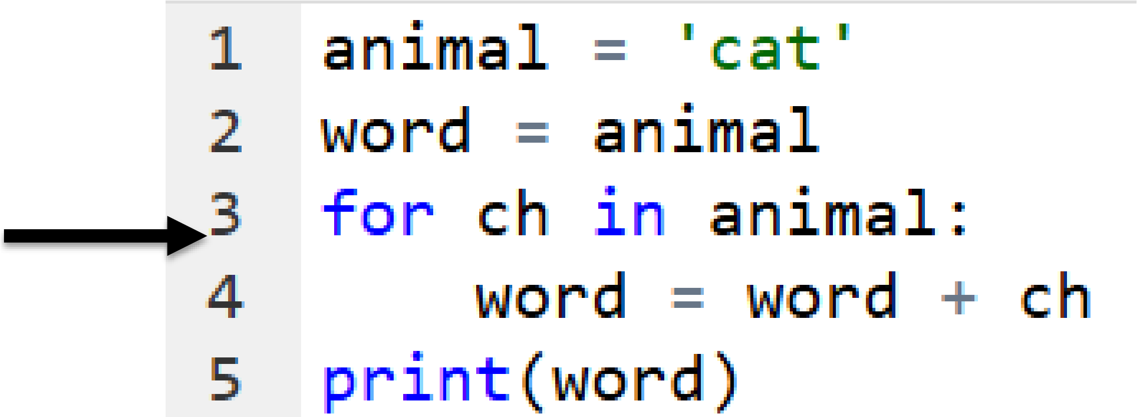
Add ch to end of word



Global frame

animal	"cat"
word	"catcat"
ch	"t"

Trace through for loop

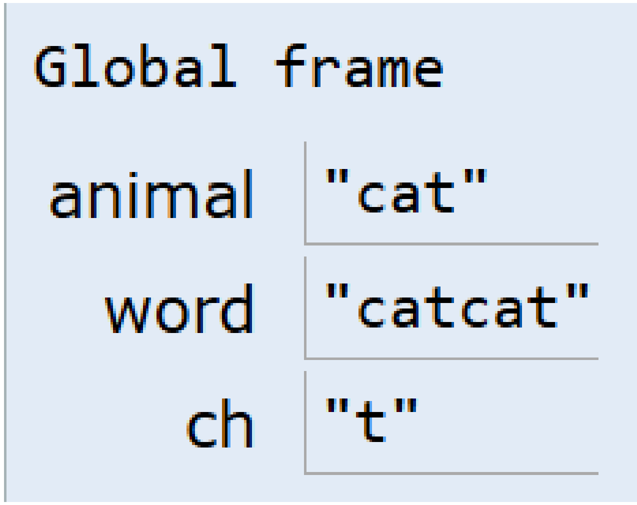


```
animal = 'cat'
word = animal
for ch in animal:
    word = word + ch
print(word)
```

Iterate over what is left in copy of word:

No more characters in word to process

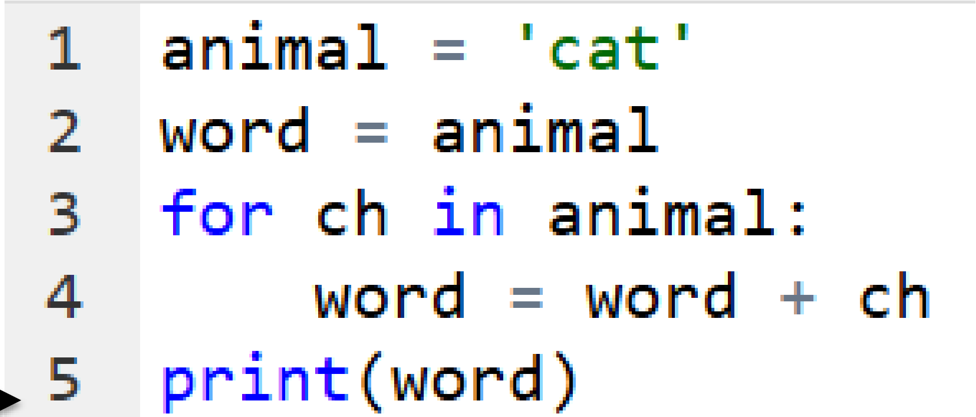
The for loop is done!



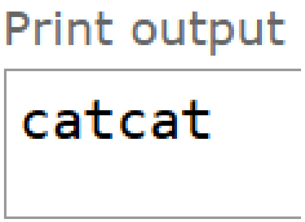
Global frame

animal	"cat"
word	"catcat"
ch	"t"

Trace through for loop

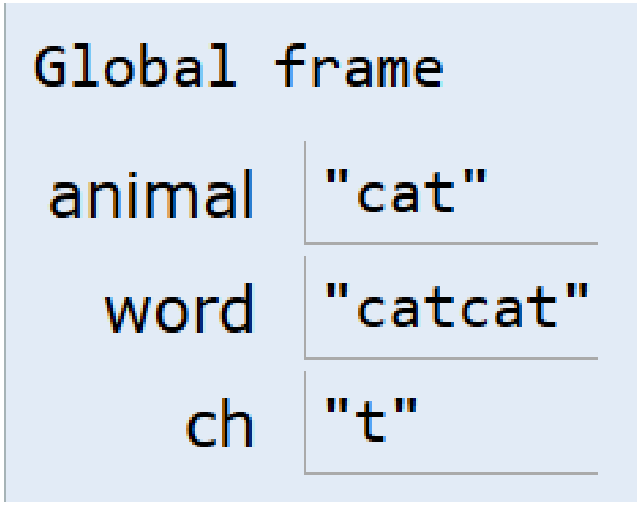


```python
animal = 'cat'
word = animal
for ch in animal:
    word = word + ch
print(word)
```



Print output

```
catcat
```



Global frame

animal	"cat"
word	"catcat"
ch	"t"

Execute code after for loop

String’s split(…)

- **Strings have functions too!**
- **`TYPE_STRING.FUNCTION(PARAMETERS)`**
  - “.” means apply function to what is on the left

`'one fish two fish'.split()` **returns a list**

  - What did it divide the string by?
    - When no parameter, default whitespace

`'one fish, two fish'.split(',')`

String’s split(…)

- **Strings have functions too!**
- **`TYPE_STRING.FUNCTION(PARAMETERS)`**
  - “.” means apply function to what is on the left

`'one fish two fish'.split()` returns a list

```
['one', 'fish', 'two', 'fish']
```

- What did it divide the string by?
  - When no parameter, default whitespace

```
'one fish, two fish'.split(',')
['one fish', ' two fish’]
```

String’s join(…)

- **`TYPE_STRING.join(SEQ_OF_STRINGS)`**
  - Opposite of .split()
  - Creates string from sequence’s items separated by the string to the left of `join`

```
' '.join(['one','fish','two','fish’])


'+'.join(['one','fish','two','fish’])


‘ish'.join([‘f',’w',’d’,’end’])
```

String’s join(…)

- **TYPE_STRING.join(SEQ_OF_STRINGS)**
  - Opposite of .split()
  - Creates string from sequence’s items separated by the string to the left of `join`

```
' '.join(['one','fish','two','fish’])
   'one fish two fish’
'+'.join(['one','fish','two','fish’])
   ‘one+fish+two+fish’
‘ish'.join([‘f',’w',’d’,’end’])
   ‘fishwishdishend’
```

More Methods

String

`.find(s)`	index of first occurrence of s
`.rfind(s)`	index of last occurrence of s (from Right)

`.upper()/`
`.lower()`
	uppercase/lowercase version of string

`.strip()`	remove leading/trailing whitespace
`.count(s)`	number of times see s in string
`.startswith(s)`	bool of whether the string begins with s
`.endswith(s)`	bool of whether the string ends with s

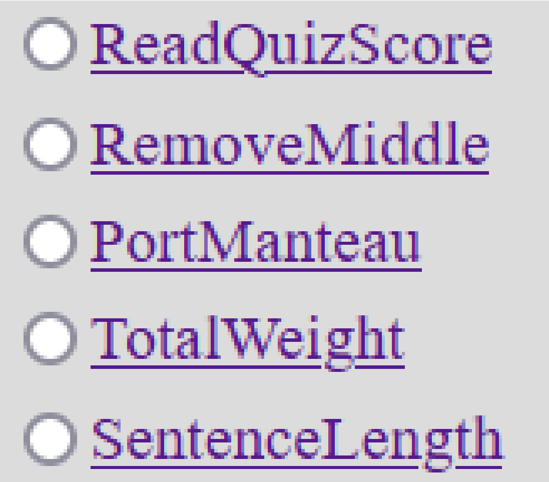
List

`sum(lst)`	sum of the elements in lst
`max(lst)`	maximum value of lst
`min(lst)`	minimum value of lst
`.append(elm)`	Mutates the list by adding elm to the end of the list
`.count(elm)`	Number of times see elm in the list

WOTO-3 – Split and Join
http://bit.ly/101s23-0202-3

APT2 out today – Due Feb 9
Do early - practice for exam

- **5 problems**
  - Write code on paper first - good practice!
  - Then type in and debug



○ ReadQuizScore
○ RemoveMiddle
○ PortManteau
○ TotalWeight
○ SentenceLength

One of these uses a loop

Exam 1 – Feb 7, 2023

- **All lecture/reading topics through today**
  - Topics today at simpler level
    - Loop over list, loop over characters in a string

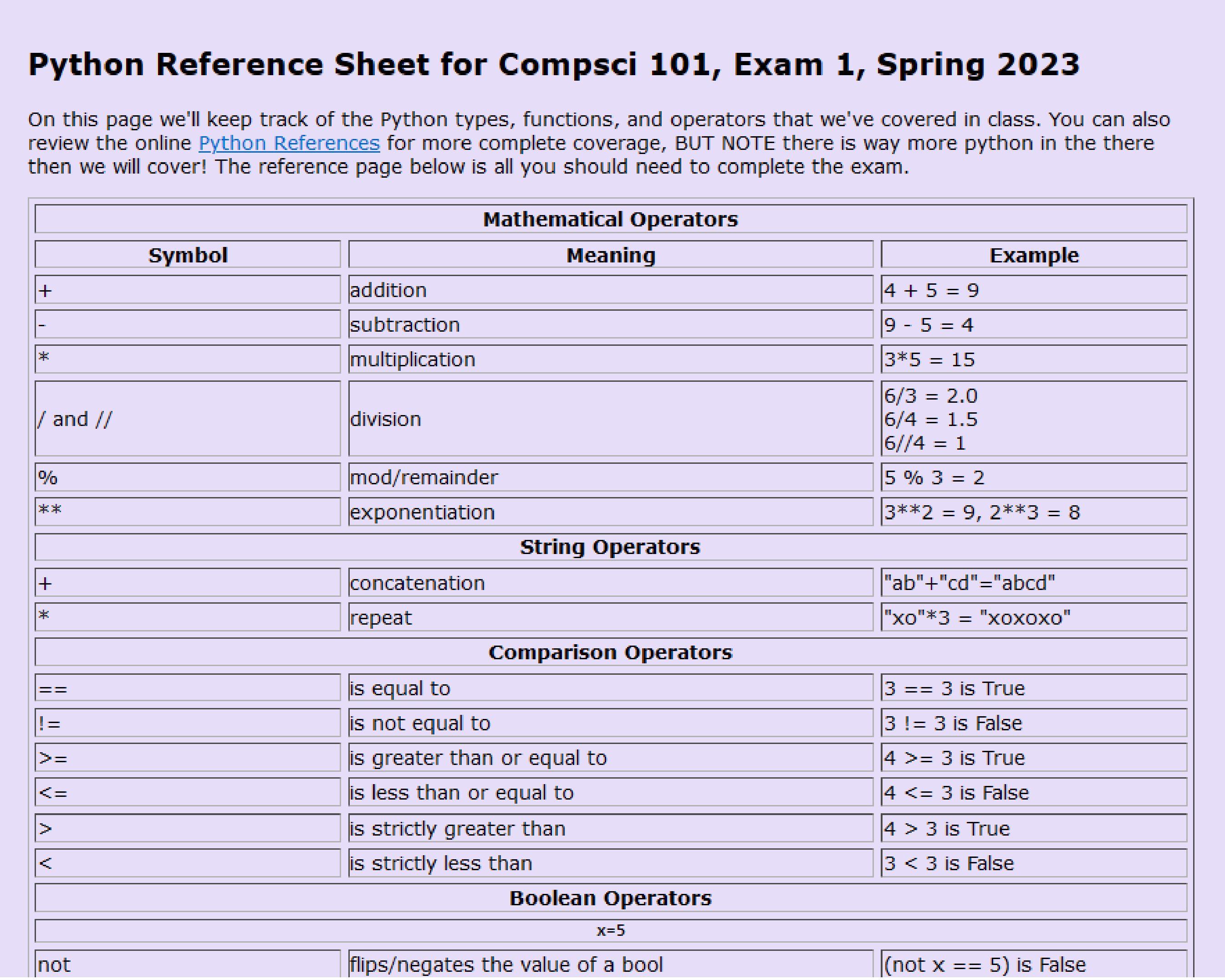
Simple for loop

- **Understand/Study**
  - Reading, lectures
  - Assignment 1, APT-1, (APT-2 helpful, not required)
  - Labs 0-3
  - Very Important! Practice writing code on paper
- **Logistics:**
  - Exam in person, in lecture

Exam 1 – Feb 7, 2023 (cont)

- **What you should be able to do**
  - Read/trace code
  - Determine output of code segment
  - Write small code segments/function
- **Look at old test questions**
  - We will look at some in Lab 3
- **Exam 1 is your own work!**
  - Only bring a pen or a pencil!
  - Do not consult with anyone else.
  - Closed book, no notes, no paper, no calculators
  - See Exam 1 Reference sheet (will be on exam)

Python Reference Sheet, is attached to your exam (see link on calendar page, under 2/7)



Python Reference Sheet for Compsci 101, Exam 1, Spring 2023

On this page we'll keep track of the Python types, functions, and operators that we've covered in class. You can also review the online Python References for more complete coverage, BUT NOTE there is way more python in the there then we will cover! The reference page below is all you should need to complete the exam.

Mathematical Operators		
Symbol	**Meaning**	**Example**
+	addition	4 + 5 = 9
-	subtraction	9 - 5 = 4
*	multiplication	3*5 = 15
/ and //	division	6/3 = 2.0 6/4 = 1.5 6//4 = 1
%	mod/remainder	5 % 3 = 2
**	exponentiation	3**2 = 9, 2**3 = 8
String Operators		
+	concatenation	"ab"+"cd"="abcd"
*	repeat	"xo"*3 = "xoxoxo"
Comparison Operators		
==	is equal to	3 == 3 is True
!=	is not equal to	3 != 3 is False
>=	is greater than or equal to	4 >= 3 is True
<=	is less than or equal to	4 <= 3 is False
>	is strictly greater than	4 > 3 is True
<	is strictly less than	3 < 3 is False
Boolean Operators		
x=5		
not	flips/negates the value of a bool	(not x == 5) is False