CS 134:
Strings & Iteration

Announcements & Logistics

- **Homework 3** will be posted to Course Website, due next Monday @ 10p
- **Lab 1** graded feedback will be released today
    - Should be available on Gradescope...
- **Lab 2** due today 10pm / tomorrow 10pm
- Lab 3 (with a **prelab**) will be released on Friday

Do You Have Any Questions?

Last Time

- Looked at more complex decisions in Python
  - Used Boolean expressions with **and**, **or**, **not**
- Chose between many different options in our code
  - `if elif else` chained conditionals

Today's Plan

- Introduce *iteration* using **for loops** to iterate over **sequences**
- Revisit an old type in the context of sequences:
  - the 'string'

Sequences in Python: Strings

- **Sequences** in Python represent **ordered collections of elements**: e.g., strings, lists, ranges, etc.
- **Strings** (type `str`) are ordered sequences of individual characters
  - Example: `word = "Hello"`
  - `'H'` is the first character of word, `'e'` is the second character, and so on
  - Each sequence element has a position, known as its index
  - In CS, we often **zero-index**, so we say that `'H'` is at **index** 0, `'e'` is at **index** 1, and so on
- We can access each character of a string using these **indices**

How Do Indices Work?

- Can access elements of a sequence (such as a string) using its **index**
- Indices in Python are both positive and negative
- Everything outside of these values will cause an `IndexError`.

0	1	2	3	4	5	6	7
"W	i	l	l	i	a	m	s"
-8	-7	-6	-5	-4	-3	-2	-1

Note: Most other languages do not support negative indexing!

Accessing Elements of Sequences

0	1	2	3	4	5	6	7
'W	i	l	l	i	a	m	s'
-8	-7	-6	-5	-4	-3	-2	-1

```
>>> word = "Williams"
>>> word[0] # character at 0th index?
'W'
>>> word[3] # character at 3rd index?
'l'
>>> word[7] # character at 7th index?
's'
>>> word[8] # will this work?

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Sequence Length

- The **`len(seq)`** function returns the length of the sequence **`seq`**
- Even though we zero-index, we still include the total number of elements in the length

```
  0  1  2  3  4  5  6  7
 'W  i  l  l  i  a  m  s'
 -8 -7 -6 -5 -4 -3 -2 -1
```

```
>>> word = "Williams"
>>> len(word) # total number of characters
8
```

```
>>> word[len(word)] # will this work?
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

```
>>> word[len(word)-1] # what about this?
's'
```

Iteration Motivation: Counting Vowels

- **Problem:** Write a function `count_vowels(word)` that takes a **string** `word` as input and returns the number of vowels in the string (an **int**)
- We'll create a function `is_vowel()` to help us:

```
def count_vowels(word):
     '''Returns number of vowels in the word'''
     # Write code here

>>> count_vowels("Williamstown")
4
>>> count_vowels("Ephelia")
4
```

is_vowel(char)

```python
    def is_vowel(ch):
        """ Returns True if ch (str) is a vowel"""
        return ch=='a' or ch=='e' or ch=='i' or ch=='o'
or ch=='u' or ch=='A' or ch=='E' or ch=='I' or ch=='O'
or ch=='U'
```

First Attempt with Conditionals

- **Note**: `val += 1` is shorthand for

  ```
  val = val + 1
  ```

- Any downsides to this approach?
- What if I change `word` to `"Williamstown"`?

```
word = "Williams"
counter = 0
if is_vowel(word[0]):
    counter += 1
if is_vowel(word[1]):
    counter += 1
if is_vowel(word[2]):
    counter += 1
if is_vowel(word[3]):
    counter += 1
if is_vowel(word[4]):
    counter += 1
if is_vowel(word[5]):
    counter += 1
if is_vowel(word[6]):
    counter += 1
if is_vowel(word[7]):
    counter += 1
print(counter)
3
```

First Attempt with Conditionals

- Using conditionals as shown is repetitive and does not generalize to arbitrarily long words
  - shorter word would "index out of bounds"
  - longer word would stop too soon
- We need something else that allows us to "loop" over the characters in an arbitrary input string
  - "For each character word, add 1 if that character is a vowel"

```
word = "Williams"
counter = 0
if is_vowel(word[0]):
    counter += 1
if is_vowel(word[1]):
    counter += 1
if is_vowel(word[2]):
    counter += 1
if is_vowel(word[3]):
    counter += 1
if is_vowel(word[4]):
    counter += 1
if is_vowel(word[5]):
    counter += 1
if is_vowel(word[6]):
    counter += 1
if is_vowel(word[7]):
    counter += 1
print(counter)
3
```

For Loops

Iterating with `for` Loops

- One of the most common ways to traverse or manipulate a sequence is to perform some action **for each element** in the sequence
- This is called **looping** or **iterating** over the elements of a sequence
- Syntax of a for loop:

var is called the loop variable

```
for var in seq:
    # body of loop
    # body of loop
```

seq is any type of sequence (for example, a string or a list)

It doesn't have to be called 'var'! It's a variable name!

Iterating with `for` Loops

- As the loop executes, the loop variable (`char` in this example) takes on the value of successive sequence elements, one by one

```
>>> # small example of for loop
>>> word = "Williams"

>>> for char in word:
...     print(char)

W
i
l
l
i
a
m
s
```

Note. Python for loops are meant *specifically* for iterating over *sequences* and are also called a "for each" loop.

Why might we call it that?

Counting Vowels

- Let us use a for loop to implement `count_vowels()` function
- What do we need to keep track of as we iterate over `word`?

```
def count_vowels(word):
    '''Takes word (str) as argument and returns
    the number of vowels in it (as int)'''

    # Write code here
```

Counting Vowels

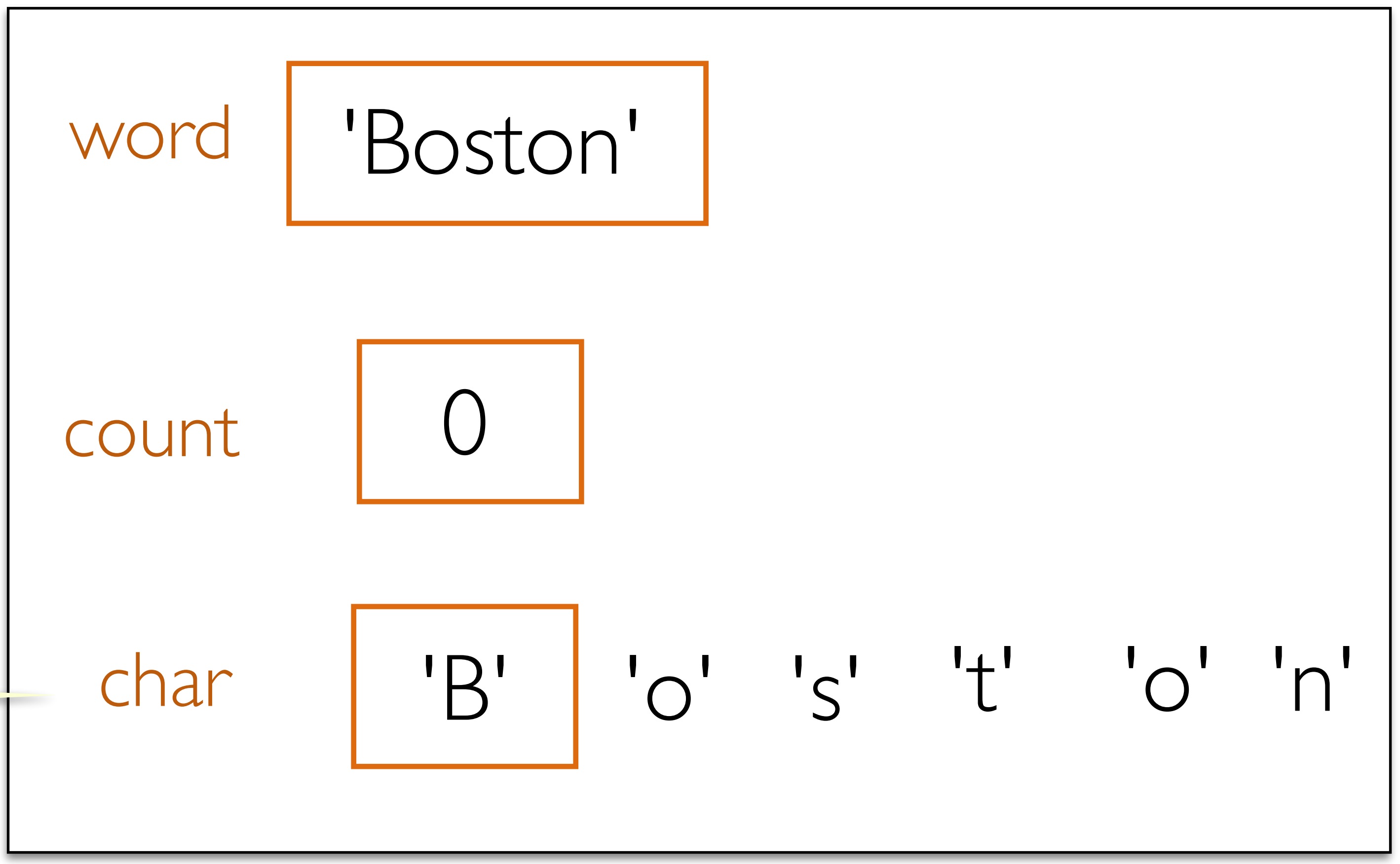
- Notice how `count` "accumulates" values in the loop
- We call `count` an **accumulation variable**

```python
def count_vowels(word):
    '''Takes word (str) as argument and returns
    the number of vowels in it (as int)'''

    count = 0 # initialize accumulator variable(counter)

    # iterate over word one character at a time
    for char in word:
        if is_vowel(char):
            count += 1  # increment accumulator variable
    return count
```

Counting Vowels: Tracing the Loop

```
def count_vowels(word):
    '''Takes word (str) as argument and returns
    the number of vowels in it (as int)'''

    count = 0
    for char in word:
        if is_vowel(char):
            count += 1
    return count
```

```
count_vowels('Boston')
```

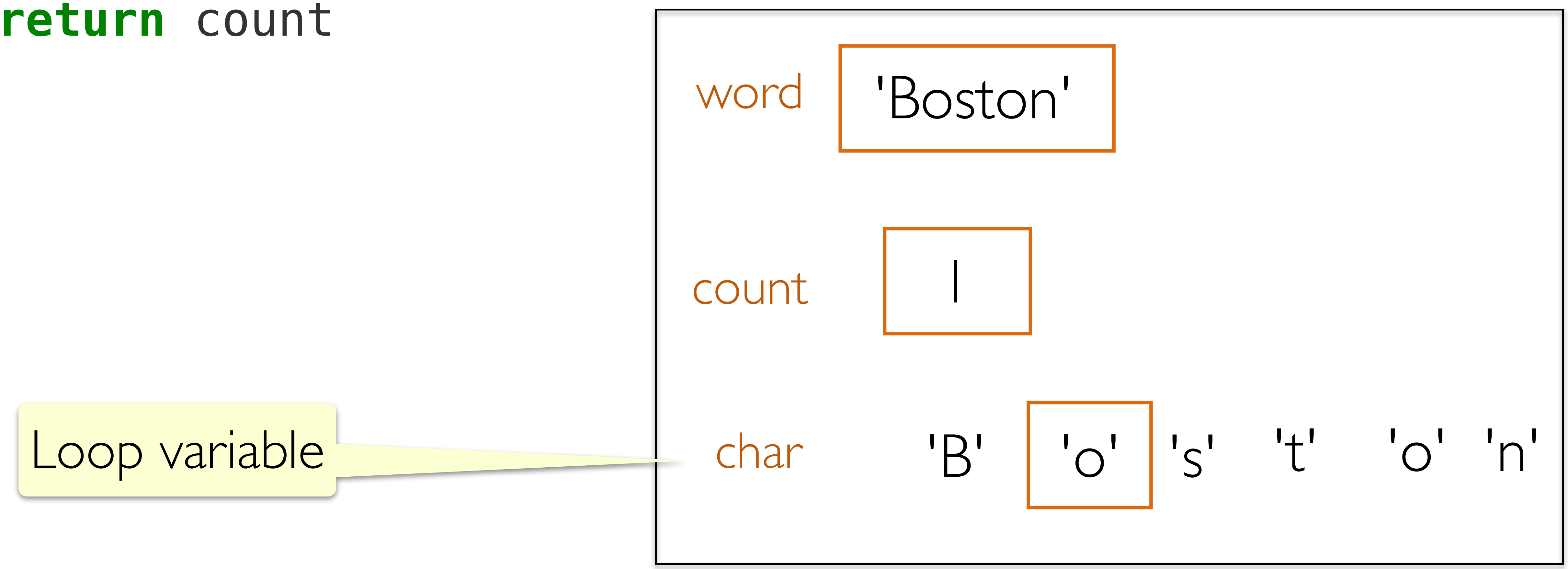


Loop variable

Counting Vowels: Tracing the Loop

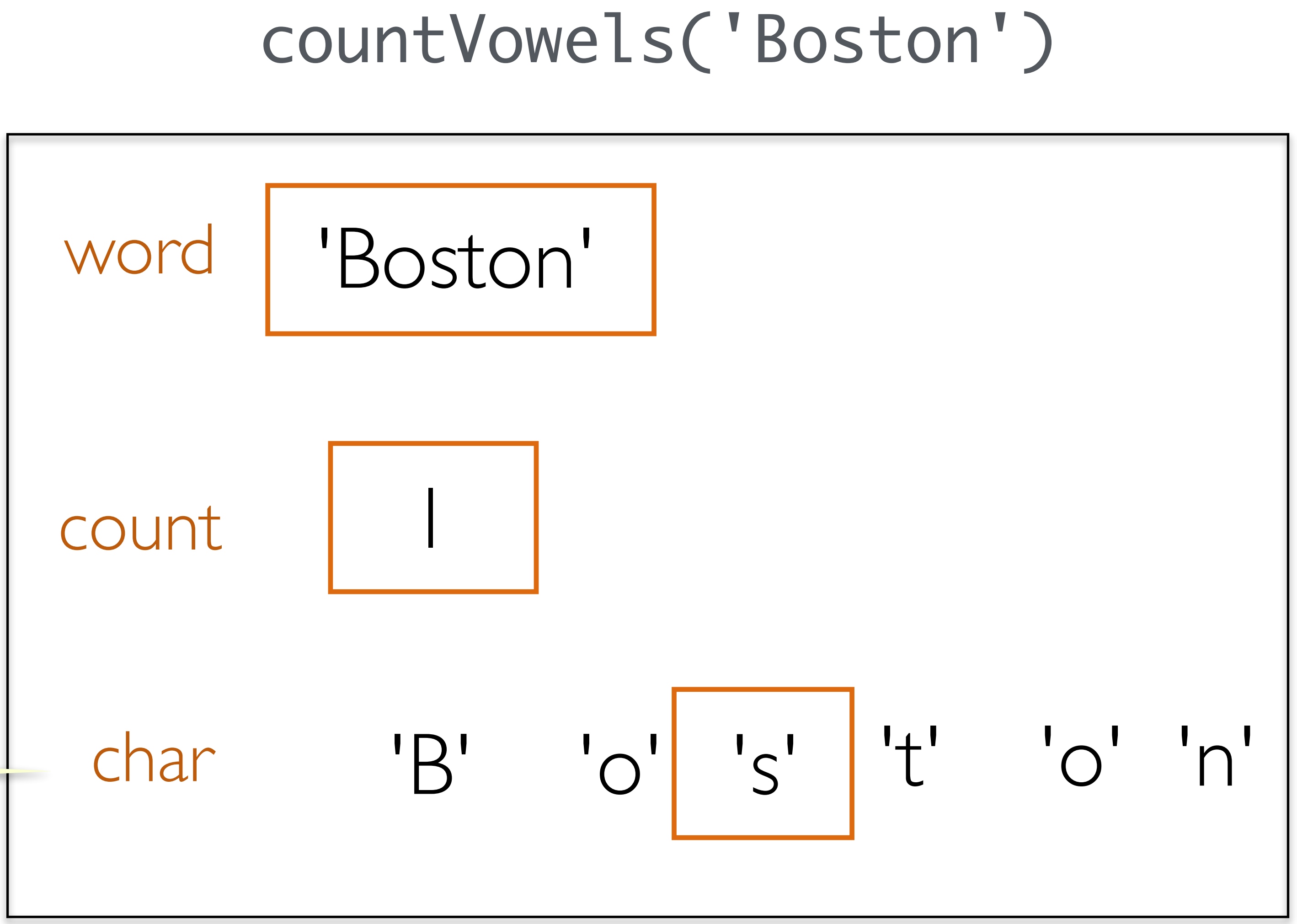
```
def count_vowels(word):
    '''Takes word (str) as argument and returns
    the number of vowels in it (as int)'''

    count = 0
    for char in word:
        if is_vowel(char):
            count += 1
    return count
```

`countVowels('Boston')`



Counting Vowels: Tracing the Loop

```
def count_vowels(word):
    '''Takes word (str) as argument and returns
    the number of vowels in it (as int)'''

    count = 0
    for char in word:
        if is_vowel(char):
            count += 1
    return count
```

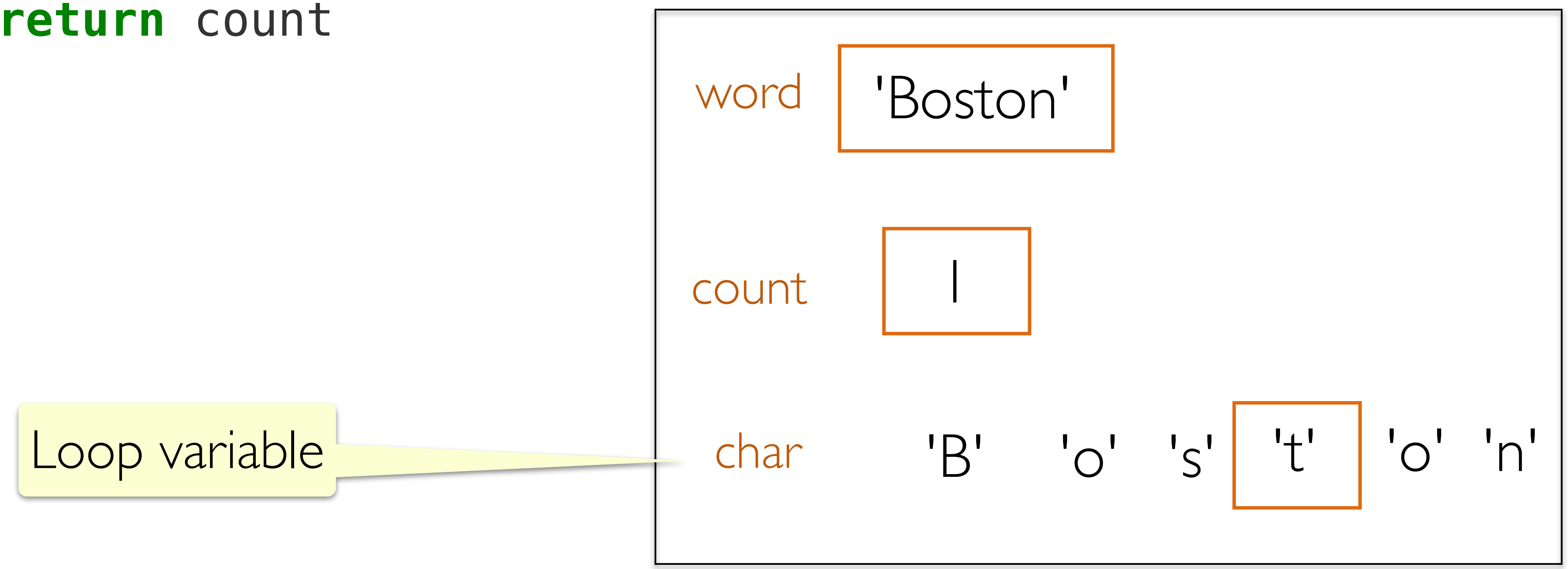


Loop variable

Counting Vowels: Tracing the Loop

```
def count_vowels(word):
    '''Takes word (str) as argument and returns
    the number of vowels in it (as int)'''

    count = 0
    for char in word:
        if is_vowel(char):
            count += 1
    return count
```

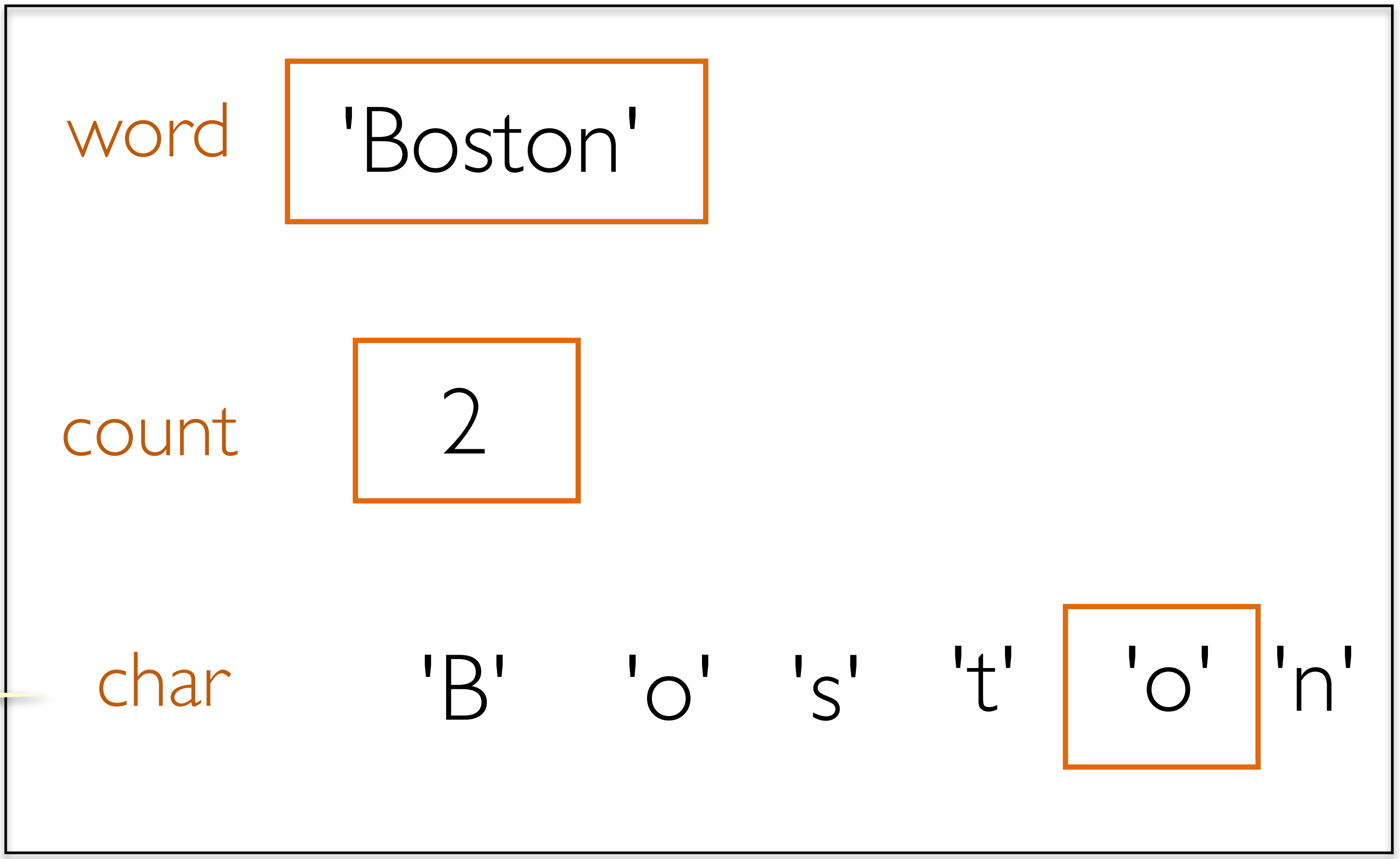
`countVowels('Boston')`



Counting Vowels: Tracing the Loop

```
def count_vowels(word):
    '''Takes word (str) as argument and returns
    the number of vowels in it (as int)'''

    count = 0
    for char in word:
        if is_vowel(char):
            count += 1
    return count
```

```
countVowels('Boston')
```

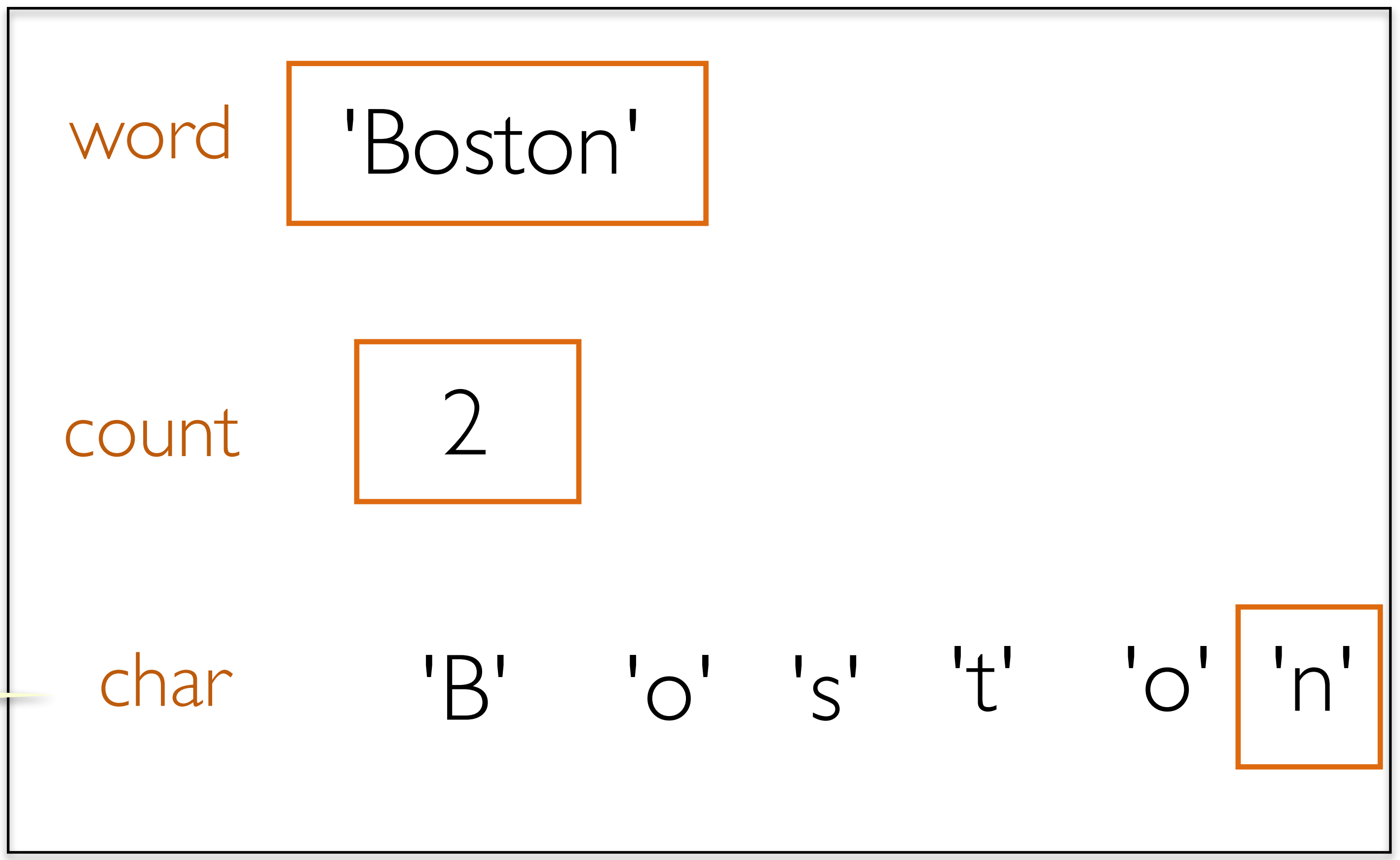


Loop variable

Counting Vowels: Tracing the Loop

```
def count_vowels(word):
    '''Takes word (str) as argument and returns
    the number of vowels in it (as int)'''

    count = 0
    for char in word:
        if is_vowel(char):
            count += 1
    return count
```

`countVowels('Boston')`



Loop variable

Exercise:
Vowel Sequences

Exercise: Vowel Sequences

- Define a function **`vowel_seq(word)`** that takes a string **`word`** and returns a string containing all the vowels in **`word`** in the order they appear

```
>>> vowel_seq("Chicago")
'iao'
>>> vowels_seq("protein")
'oei'
>>> vowel_seq("rhythm")
''
```

What might be other good values to test edge cases?

Exercise: Vowel Sequences

- Accumulation variables don't have to be counters!
- Can accumulate strings as well: initialize to '' instead of zero

```
def vowel_seq(word):
    '''Takes word (str) as input and returns
    the vowel subsequence in given word (str)'''
    vowels = ""  # initialize accumulation var
    for char in word:
        if is_vowel(char): # if vowel
            vowels += char # accumulate characters
    return vowels
```

Sequence Operations

Operation	Result
`seq[i]`	The `i`'th item of `seq`, when starting with 0
`seq[si:ee]`	slice of `seq` from `si` to `ee`
`seq[si:ee:s]`	slice of `seq` from `si` to `ee` with step `s`
`len(seq)`	length of `seq`
`seq1 + seq2`	The concatenation of `seq1` and `seq2`
`x in seq`	True if `x` is contained within `seq`
`x not in seq`	False if `x` is contained within `seq`

The end!