[301] Lists

Tyler Caraza-Harter

Learning Objectives Today

List syntax

- creation, indexing, for loop

Chapter 10 of Think Python

Comparison to strings

- similarity: len, slicing, concatenation, in, multiply
- differences: flexible types, mutability

Modifying lists

- update, append, pop, sort

Switching between strings and lists

- split, join

Today's Outline

List Syntax

Similarities with Strings

Difference 1: Flexibility of Types

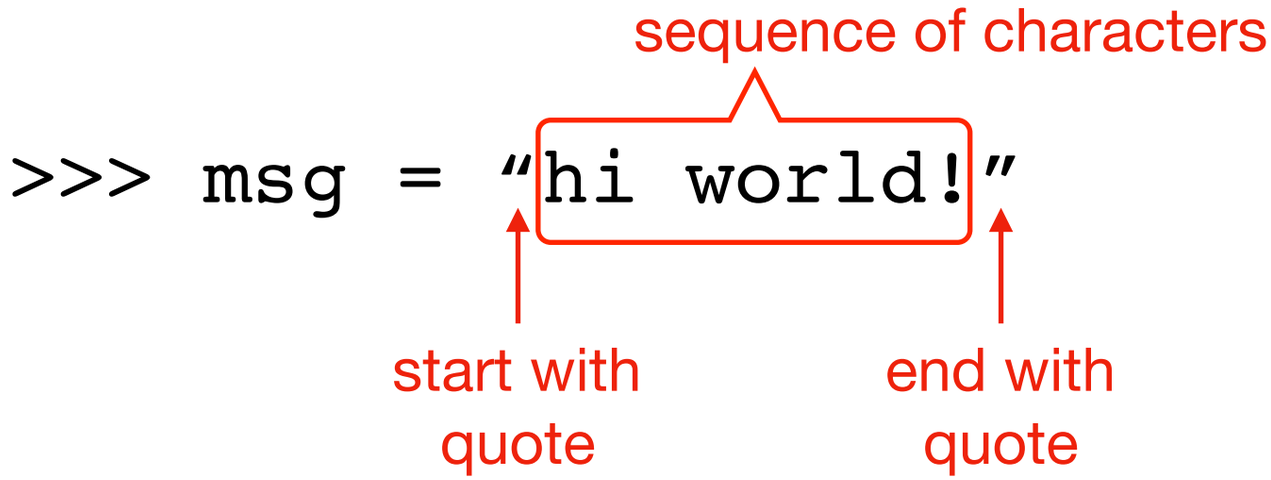
Difference 2: Mutability

Transforming between Strings and Lists

A string is a **sequence** of characters

```
>>> msg = “hi world!”
```

A string is a **sequence** of characters



A string is a **sequence** of characters

```
>>> msg = “hi world!”
```

Things we can do with sequences

- index
- slice
- for loop

A string is a **sequence** of characters

```
>>> msg = “hi world!”
>>> msg[1]
‘i’
```

Things we can do with sequences

- **index**
- slice
- for loop

A string is a **sequence** of characters

```
>>> msg = “hi world!”
>>> msg[1]
‘i’
>>> msg[3]
‘w’
```

Things we can do with sequences

- **index**
- slice
- for loop

A string is a **sequence** of characters

```
>>> msg = “hi world!”
>>> msg[3:]
‘world!’
```

Things we can do with sequences

- index
- **slice**
- for loop

A string is a **sequence** of characters

```
>>> msg = “hi world!”
>>> msg[3:]
‘world!’
>>> msg[3:-1]
‘world’
```

Things we can do with sequences

- index
- **slice**
- for loop

A string is a **sequence** of characters

```
>>> msg = “hi world!”
>>> for c in msg:
...   print(c)
```

Things we can do with sequences

- index
- slice
- **for loop**

A string is a **sequence** of characters

```
>>> msg = “hi world!”
>>> for c in msg:
...     print(c)
...
h
i

w
o
r
l
d
!
```

Things we can do with sequences

- index
- slice
- **for loop**

A string is a **sequence** of characters

```
>>> msg = “hi world!”
```

What if we want a sequence, of something other than characters?

Use a Python list, with any items we want!

A list is a **sequence** of values (could be integers, or anything else)

```
>>> msg = “hi world!”
>>> nums = [22, 11, 33]
```

What if we want a sequence, of something other than characters?

Use a Python list, with any items we want!

A list is a **sequence** of values (could be integers, or anything else)

```
>>> msg = “hi world!”
>>> nums = [22, 11, 33]
```

square bracket instead of quote

sequence of values, comma separated

square bracket instead of quote

What if we want a sequence, of something other than characters?

Use a Python list, with any items we want!

A list is a **sequence** of values (could be integers, or anything else)

```
>>> msg = “hi world!”
>>> nums = [22, 11, 33]
```

Things we can do with sequences

- index
- slice
- for loop

A list is a **sequence** of values (could be integers, or anything else)

```
>>> msg = “hi world!”
>>> nums = [22, 11, 33]
>>> nums[0]
22
```

Things we can do with sequences

- **index**
- slice
- for loop

A list is a **sequence** of values (could be integers, or anything else)

```
>>> msg = “hi world!”
>>> nums = [22, 11, 33]
>>> nums[0]
22
>>> nums[-1]
33
```

Things we can do with sequences

- **index**
- slice
- for loop

A list is a **sequence** of values (could be integers, or anything else)

```
>>> msg = “hi world!”
>>> nums = [22, 11, 33]
>>> nums[1:]
[11, 33]
```

Things we can do with sequences

- index
- **slice**
- for loop

A list is a **sequence** of values (could be integers, or anything else)

```
>>> msg = “hi world!”
>>> nums = [22, 11, 33]
>>> nums[1:]
[11, 33]
>>> nums[3:]
[ ]
```

Things we can do with sequences

- index
- **slice**
- for loop

A list is a **sequence** of values (could be integers, or anything else)

```
>>> msg = “hi world!”
>>> nums = [22, 11, 33]
>>> for x in nums:
...     print(x)
```

Things we can do with sequences

- index
- slice
- **for loop**

A list is a **sequence** of values (could be integers, or anything else)

```
>>> msg = “hi world!”
>>> nums = [22, 11, 33]
>>> for x in nums:
...    print(x)
...
22
11
33
```

Things we can do with sequences

- index
- slice
- **for loop**

Demo: Finding a Sum

Goal: write a function to add a list of numbers

Input:

- Python list containing floats

Output:

- Sum of the numbers

Example:

```
>>> nums = [1, 2, 3]
>>> add_nums(nums)
6
>>> add_nums([20, 30])
50
```

Demo: Finding a Sum

Goal: write a function to add a list of numbers

Input:

- Python list containing floats

Output:

- Sum of the numbers

Example:

```
>>> nums = [1, 2, 3]
>>> add_nums(nums)
6
>>> add_nums([20, 30])
50
```

Note: I did it the hard way as an example, but these are handy:
min(lst), max(lst), sum(lst), len(lst)

Today's Outline

List Syntax

Similarities with Strings

Difference 1: Flexibility of Types

Difference 2: Mutability

Transforming between Strings and Lists

Things we can do with strings and lists

1. len

2. slicing

3. concatenation

4. in

5 multiply by an int

1. len(sequence)

string

```
>>> msg = “321go”
```

list

```
>>> items = [99,11,77,55]
```

1. len(sequence)

string

```
>>> msg = “321go”
>>> len(msg)
5
```

list

```
>>> items = [99,11,77,55]
>>> len(items)
4
```

2. slicing

string

```
>>> msg = “321go”
>>> msg[3:]
‘go’
```

list

```
>>> items = [99,11,77,55]
>>> items[1:3]
[11,77]
```

3. concatenation

string

```
>>> msg = “321go”
>>> msg + “!!!”
‘321go!!!’
```

list

```
>>> items = [99,11,77,55]
>>> items + [1,2,3]
[99,11,77,55,1,2,3]
```

4. in

string

```
>>> msg = “321go”
>>> ‘g’ in msg
True
```

list

```
>>> items = [99,11,77,55]
>>> 11 in items
True
```

4. in

string

```
>>> msg = “321go”
>>> ‘g’ in msg
True
>>> ‘z’ in msg
False
```

list

```
>>> items = [99,11,77,55]
>>> 11 in items
True
>>> 10 in items
False
```

5. multiply by int

string

```
>>> msg = “321go”
>>> msg * 2
‘321go321go’
```

list

```
>>> items = [99,11,77,55]
>>> items * 2
[99,11,77,55,99,11,77,55]
```

Today's Outline

List Syntax

Similarities with Strings

Difference 1: Flexibility of Types

Difference 2: Mutability

Transforming between Strings and Lists

Items can be any types

string, bool, int, float

even other lists!

Items can be any types

string, bool, int, float

even other lists!

Code example (run in terminal):

```
l = [True, False, 3, "hey", [1, 2]]
for item in l:
  print(type(l))
```

Items can be any types

string, bool, int, float

even other lists!

Code example (run in terminal):

```
l = [True, False, 3, "hey", [1, 2]]
for item in l:
  print(type(l))
```

What to type if we want to get 2 (last item of last item)?

Today's Outline

List Syntax

Similarities with Strings

Difference 1: Flexibility of Types

Difference 2: Mutability

Transforming between Strings and Lists

Mutability

Definition

- a type is **mutable** if values can be changed
- a type is **immutable** if values cannot be changed

Mutability

Definition

- a type is **mutable** if values can be changed
- a type is **immutable** if values cannot be changed

```
s = “hello”
s[0] = “j”
```

Mutability

Definition

- a type is **mutable** if values can be changed
- a type is **immutable** if values cannot be changed

```
s = “hello”
s[0] = “j”
```

fails! because strings are immutable

Mutability

Definition

- a type is **mutable** if values can be changed
- a type is **immutable** if values cannot be changed

```
s = “hello”
s[0] = “j”
```

fails! because strings are immutable

Mutability

Definition

- a type is **mutable** if values can be changed
- a type is **immutable** if values cannot be changed

```
s = “hello”
s[0] = “j”
```

fails! because strings are immutable

```
s += “oooo”
```

Mutability

Definition

- a type is **mutable** if values can be changed
- a type is **immutable** if values cannot be changed

```
s = “hello”
s[0] = “j”
```

fails! because strings are immutable

```
s += “oooo”
```

this works! because we aren’t changing the string “hello”. We’re reassigning a new string “hellooooo” to the variable s

Mutability

Definition

- a type is **mutable** if values can be changed
- a type is **immutable** if values cannot be changed

```
s = “hello”
s[0] = “j”
```

fails! because strings are immutable

```
s += “oooo”
```

this works! because we aren’t changing the string “hello”. We’re reassigning a new string “hellooooo” to the variable s

```
nums = [3,2,1]

nums[0] = 300
```

Mutability

Definition

- a type is **mutable** if values can be changed
- a type is **immutable** if values cannot be changed

```
s = “hello”
s[0] = “j”
```

fails! because strings are immutable

```
s += “oooo”
```

this works! because we aren’t changing the string “hello”. We’re reassigning a new string “hellooooo” to the variable s

```
nums = [3,2,1]

nums[0] = 300
# nums is [300,2,1]
```

Mutability

Definition

- a type is **mutable** if values can be changed
- a type is **immutable** if values cannot be changed

```
s = “hello”
s[0] = “j”
```

fails! because strings are immutable

```
s += “oooo”
```

this works! because we aren’t changing the string “hello”. We’re reassigning a new string “hellooooo” to the variable s

```
nums = [3,2,1]

nums[0] = 300
# nums is [300,2,1]
nums += [9,8]
```

Mutability

Definition

- a type is **mutable** if values can be changed
- a type is **immutable** if values cannot be changed

```
s = “hello”
s[0] = “j”
```

fails! because strings are immutable

```
s += “oooo”
```

this works! because we aren’t changing the string “hello”. We’re reassigning a new string “hellooooo” to the variable s

```
nums = [3,2,1]

nums[0] = 300
# nums is [300,2,1]
nums += [9,8]
# nums is [300,2,1,9,8]
```

Mutability

Definition

- a type is **mutable** if values can be changed
- a type is **immutable** if values cannot be changed

```
s = “hello”
s[0] = “j”
```

fails! because strings are immutable

```
s += “oooo”

nums = [3,2,1]

nums[0] = 300
# nums is [300,2,1]
nums += [9,8]
# nums is [300,2,1,9,8]
```

this works! because we aren’t changing the string “hello”. We’re reassigning a new string “hellooooo” to the variable s

both work, because lists are mutable

Ways to mutate a list

Common Modifications

- L[index] = new_value
- L.append(new_value)
- L.pop(index)
- L.sort()

Example code:

```
L = [3,2,1]
L.append(0)
L[1] = -1
L.sort()
L.pop(0)
```

Demo these in interactive mode

Demo: Finding a Median

Goal: write a function to find the median of a list of numbers

Input:

- Python list containing floats

Output:

- The median

Example:

```
>>> nums = [1,5,2,9,8]
>>> median(nums)
5
>>> median([1, 20, 30, 100])
25
```

Today's Outline

List Syntax

Similarities with Strings

Difference 1: Flexibility of Types

Difference 2: Mutability

Transforming between Strings and Lists

split method

Turns a string into a list

- operates on a string
- takes a separator
- returns a list

```
>>> S = “this is a test”
>>> L = S.split(“ “)
>>> L
[“this”, “is”, “a”, “test”]
```

join method

Turns a list into a string

- operates on a separator
- takes a list
- returns a string

```
>>> L = [“i”, “don’t”, “know”]
>>> sep = “...“
>>> sep.join(L)
i...don't...know
```

Demo: Censoring Profanity

Goal: write a function to replace curse words with start

Input:

- A profane string

Output:

- A sanitized string

Example:

>>> censor(“OMG this class is so fun”)
‘*** this class is so fun’
>>> censor(“the midterm was darn tough”)
‘the ******* was **** tough’

Demo: Censoring Profanity

Goal: write a function to replace curse words with start

Input:

- A profane string

Output:

- A sanitized string

Example:

```
>>> censor(“OMG this class is so fun”)
‘*** this class is so fun’
>>> censor(“the midterm was darn tough”)
‘the ******* was **** tough’
```

replaces offensive words like “darn” and “midterm” with stars