[301] Lists

Tyler Caraza-Harter
Learning Objectives Today

Lists, the mutable sequence that can hold ANYTHING!

Sequence stuff
- indexing, slicing, for loops
- len, in, concatenation, multiplication

Mutating!
- update, append, pop, sort

Switching between strings and lists
- split, join

Chapter 10 of Think Python
Today's Outline

From Strings to Lists

More Sequence Capabilities

Difference 1: Flexibility of Types

Difference 2: Mutability

Transforming between Strings and Lists
A string is a **sequence** of characters

**indexing:** access one value  
**slicing:** extract sub-sequence

for loop: execute for each value

Things we can do with sequences

- indexing
- slicing
- for loop
A string is a **sequence** of characters

```python
>>> msg = "hi world!"
```

Things we can do with sequences

- indexing
- slicing
- for loop
A string is a **sequence** of characters

```python
>>> msg = "hi world!"
>>> msg[1]
```

Things we can do with sequences

- **indexing**
- **slicing**
- **for loop**
A string is a **sequence** of characters

```python
>>> msg = "hi world!"
>>> msg[1]
'i'
```

Things we can do with sequences
- **indexing**
- slicing
- for loop
A string is a **sequence** of characters

```python
>>> msg = "hi world!"
>>> msg[1]
'i'
>>> msg[3]
```

Things we can do with sequences
- **indexing**
- slicing
- for loop
A string is a **sequence** of characters

```python
>>> msg = "hi world!"
>>> msg[1]
'i'
>>> msg[3]
'w'
```

Things we can do with sequences

- **indexing**
- **slicing**
- **for loop**
A string is a sequence of characters

```python
>>> msg = "hi world!"
>>> msg[3:]
```

Things we can do with sequences

- indexing
- **slicing**
- for loop
A string is a **sequence** of characters

```python
>>> msg = "hi world!"
>>> msg[3:]
'world!'
```

Things we can do with sequences
- indexing
- **slicing**
- for loop
A string is a **sequence** of characters

```python
global msg
msg = "hi world!"
print(msg[3:])  # world!
print(msg[3:-1])  # hi wor
```

**Things we can do with sequences**
- indexing
- **slicing**
- for loop
A string is a sequence of characters

```python
>>> msg = "hi world!"
>>> msg[3:]
'world'
>>> msg[3:-1]
'world'
```

Things we can do with sequences
- indexing
- slicing
- for loop
A string is a sequence of characters

```python
>>> msg = "hi world!"
>>> for c in msg:
...    print(c)
```

Things we can do with sequences
- indexing
- slicing
- for loop
A string is a **sequence** of characters

```python
>>> msg = "hi world!"
>>> for c in msg:
...     print(c)
...     ...

h
i

w
o
r
l
d!
```

**Things we can do with sequences**
- indexing
- slicing
- **for loop**
A string is a **sequence** of characters

```python
>>> msg = "hi world!"
```

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

Use a Python **list**, with any items we want!
A string is a **sequence** of characters

```python
>>> 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 anything

```python
>>> 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 **anything**

```python
>>> msg = "hi world!"
>>> nums = [22, 11, 33]
```

**Things we can do with sequences**

- indexing
- slicing
- for loop
A list is a **sequence** of *anything*

```python
>>> msg = "hi world!"
>>> nums = [22, 11, 33]
>>> nums[0]
```

**Things we can do with sequences**
- **indexing**
- slicing
- for loop
A list is a **sequence** of *anything*

```python
>>> msg = "hi world!"
>>> nums = [22, 11, 33]
>>> nums[0]
22
```

**Things we can do with sequences**
- **indexing**
- slicing
- for loop
A list is a **sequence** of anything

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

Things we can do with sequences

- **indexing**
- **slicing**
- **for loop**
A list is a **sequence** of *anything*

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

Things we can do with sequences
- indexing
- slicing
- for loop
A list is a sequence of anything

```python
>>> msg = "hi world!"
>>> nums = [22, 11, 33]
>>> [22, 11, 33][1]
11
```

seeing brackets for both creating lists and indexing often confuses new coders!

Things we can do with sequences
- indexing
- slicing
- for loop
A list is a **sequence** of *anything*

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

Things we can do with sequences

- indexing
- **slicing**
- for loop
A list is a **sequence** of **anything**

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

Things we can do with sequences

- indexing
- **slicing**
- for loop
A list is a **sequence** of **anything**

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

Things we can do with sequences
- indexing
- slicing
- for loop
A list is a sequence of anything

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

Things we can do with sequences

- indexing
- slicing
- for loop
A list is a **sequence** of *anything*

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

**Things we can do with sequences**

- indexing
- slicing
- **for loop**
A list is a **sequence** of **anything**

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

Things we can do with sequences
- indexing
- slicing
- **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:
```python
def add_nums(nums):
    return sum(nums)

>>> nums = [1, 2, 3.5]
>>> add_nums(nums)
6.5
>>> add_nums([20, 30.1])
50.1```
Today's Outline

From Strings to Lists

**More Sequence Capabilities**

Difference 1: Flexibility of Types

Difference 2: Mutability

Transforming between Strings and Lists
Cool stuff we can do with strings and lists

1. indexing
2. slicing
3. for loops
4. len
5. concatenation
6. in
7. multiply by an int
4. len(sequence)

string

```python
>>> msg = "321go"
```

list

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

string

```python
>>> msg = "321go"
>>> len(msg)
5
```

list

```python
>>> items = [99,11,77,55]
>>> len(items)
4
```
5. concatenation

<table>
<thead>
<tr>
<th>string</th>
<th>list</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;&gt; msg = &quot;321go&quot;</td>
<td>&gt;&gt;&gt; items = [99,11,77,55]</td>
</tr>
<tr>
<td>&gt;&gt;&gt; msg + &quot;!!!&quot;</td>
<td>&gt;&gt;&gt; items + [1,2,3]</td>
</tr>
<tr>
<td>‘321go!!!’</td>
<td>[99,11,77,55,1,2,3]</td>
</tr>
</tbody>
</table>
6. in

string

```python
>>> msg = "321go"
>>> 'g' in msg
True
```

list

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

string

```python
>>> msg = "321go"
>>> 'g' in msg
True
>>> 'z' in msg
False
```
7. multiply by int

**string**

```python
>>> msg = "321go"
>>> msg * 2
'321go321go'
```

**list**

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

str methods
find replace upper/lower format etc.

lists

sequence stuff
indexing slicing for loops len concatenation in multiply by an int

flexible types
mutation

now
Today's Outline

From Strings to Lists

More Sequence Capabilities

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!

coding demo:

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

**bonus:** how to extract the last item of the last item?
Today's Outline

From Strings to Lists

More Sequence Capabilities

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

*careful! this is is about values, not variables*
Mutability

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

**careful! this is is about values, not variables**

<table>
<thead>
<tr>
<th></th>
<th>set variable to new value</th>
<th>change existing value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>list</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(mutable)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>str</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(immutable)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- `s = "AB"`  ✔️
- `s += "C"`  ✔️
- `s[0] = "3"` ✗
Mutability

Definition

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

**careful! this is is about values, not variables**

<table>
<thead>
<tr>
<th>list (mutable)</th>
<th>set variable to new value</th>
<th>change existing value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nums = [1,2]</td>
<td>nums += [3] ✔️</td>
<td>nums[0] = 3 ✔️</td>
</tr>
<tr>
<td>str (immutable)</td>
<td>s = &quot;AB&quot; ✔️</td>
<td>s = &quot;201&quot; ✗</td>
</tr>
<tr>
<td></td>
<td>s += &quot;C&quot; ✔️</td>
<td>s[0] = &quot;3&quot; ✗</td>
</tr>
</tbody>
</table>
Ways to mutate a list

Common Modifications
- \( L[\text{index}] = \text{new\_value} \)
- \( L.\text{append} (\text{new\_value}) \)
- \( L.\text{pop} (\text{index}) \)
- \( L.\text{sort} () \)

Example code:

\[
L = [3, 2, 1] \\
L.\text{append}(0) \\
L[1] = -1 \\
L.\text{sort}() \\
L.\text{pop}(0)
\]
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

From Strings to Lists

More Sequence Capabilities

Difference 1: Flexibility of Types

Difference 2: Mutability

Transforming between Strings and Lists
split method

\[ S = "a\ quick\ brown\ fox" \]
\[ L = S.split("\ ") \]

"a quick brown fox" ➔ ["a", "quick", "brown", "fox"]
join method

\[
L = ["M", "SS", "SS", "PP", ""]
S = "I".join(L)
\]

separator

["M", "SS", "SS", "PP", ""] → ????
join method

L = ["M", "SS", "SS", "PP", ""]
S = "I".join(L)

separator

["M", "SS", "SS", "PP", "]
MISSISSIPPI
join method

\[ L = ["M", "SS", "SS", "PP", ","] \]
\[ S = "I".join(L) \]

what if removed?

separator

["M", "SS", "SS", "PP", "]]\n
MISSISSIPPI
join method

\[
L = ["M", "SS", "SS", "PP"] \\
S = "I".join(L)
\]

separator

["M", "SS", "SS", "PP", "]

MISSISSIPP
Demo: Censoring Profanity

Goal: write a function to replace curse words with stars

Input:
- A profane string

Output:
- A sanitized string

Example:

```python
>>> censor("OMG this class is so fun")
'*** this class is so fun'

>>> censor("the midterm is darn soon")
'the ******* was **** tough'
```
Demo: Censoring Profanity

Goal: write a function to replace curse words with stars

Input:
• A profane string

Output:
• A sanitized string

Example:

```python
>>> censor("OMG this class is so fun")
'*** this class is so fun'

>>> censor("the midterm is darn soon")
'the ******* was **** tough'
```

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