

CS 220 / CS319

Lists

Department of Computer Sciences
University of Wisconsin-Madison

Learning Objectives

List creation and sequence operations

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

Key differences between strings and lists

- type flexibility
- mutability

Mutating a list using:

- indexing
- methods: append, extend, pop, and sort

`split(...)` a string into a list

`join(...)` a list into a string

Today's Outline

From Strings to Lists

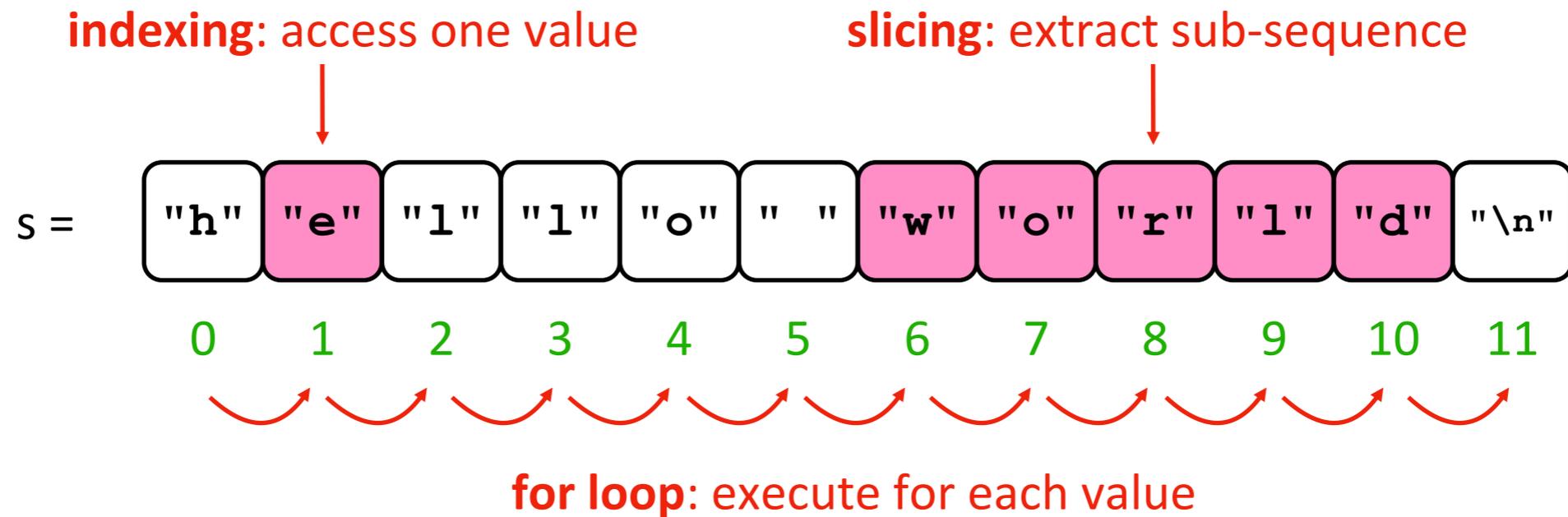
More Sequence Capabilities

Difference 1: Flexibility of Types

Difference 2: Mutability

Transforming between Strings and Lists

A string is a **sequence** of characters



Things we can do with sequences

- indexing
- slicing
- for loop

A string is a **sequence** of characters

```
>>> 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

```
>>> 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

```
>>> 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

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

sequence of characters

start with quote

end with quote

str syntax

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*

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

str syntax
list syntax

↑ 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 *anything*

```
>>> 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*

```
>>> 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*

```
>>> 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*

```
>>> 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:

```
>>> 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~~

any sequence

- 1 indexing
- 2 slicing
- 3 for loops
- 4 len
- 5 concatenation
- 6 in
- 7 multiply by an int

4. len(sequence)

string

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

list

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

5. concatenation

string

```
>>> msg = "321go"  
>>> msg + "!!!"  
'321go!!!'
```

list

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

6. 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
```

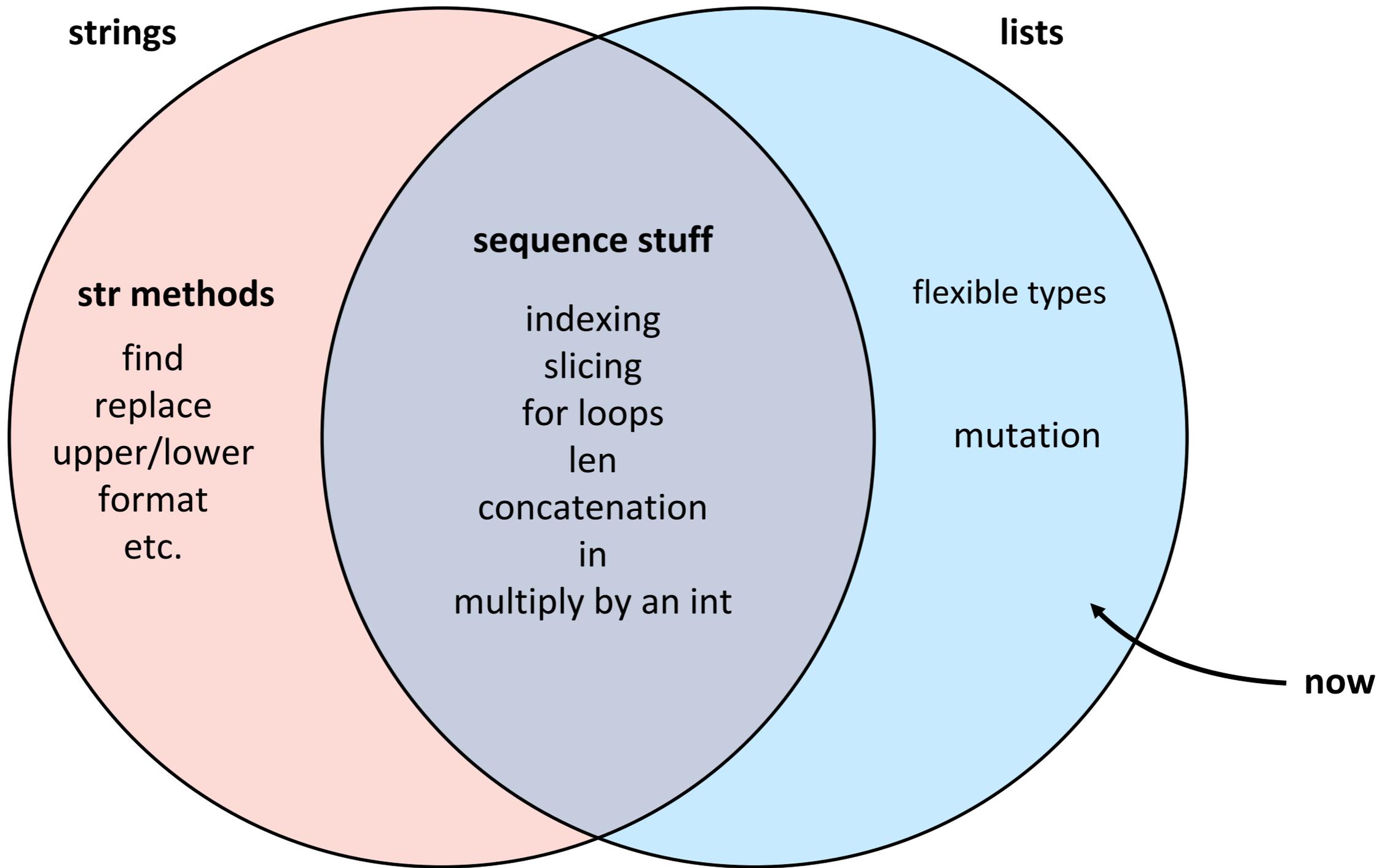
7. multiply by int

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

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!

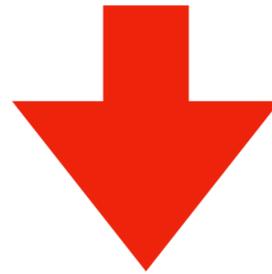
coding demo:

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

bonus: how to extract the last item of the last item?

Example game map with list of lists

```
[  
  [".", ".", ".", ".", ".", "S"],  
  [".", "S", "S", "S", ".", "S"],  
  [".", ".", ".", ".", ".", "S"],  
  [".", ".", ".", ".", ".", "."],  
  [".", ".", ".", ".", "S", "."],  
  [".", ".", ".", ".", "S", "."]  
]
```



```
.....S  
.SSS.S  
.....S  
.....  
....S.  
....S.
```

*rows and columns
of data are useful for
more than games...*

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 about *values*, not *variables*
(variables can ALWAYS be changed)

	set variable to new value	change existing value
list (mutable)	<pre>nums = [1, 2] nums = [3, 4]</pre> ✓	<pre>nums = [2, 2, 9] nums[2] = 0</pre> ✓
str (immutable)	<pre>s = "AB" s = "CD" s += "E"</pre> ✓	<pre>s = "229" s[2] = "0"</pre> ✗

Ways to mutate a list

Common Modifications

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

Example code:

```
L = [3, 2, 1]
L.append(0)
L.extend([9, 8])
L[1] = -1
L.sort()
L.pop(0)
```

Today's Outline

From Strings to Lists

More Sequence Capabilities

Difference 1: Flexibility of Types

Difference 2: Mutability

Transforming between Strings and Lists

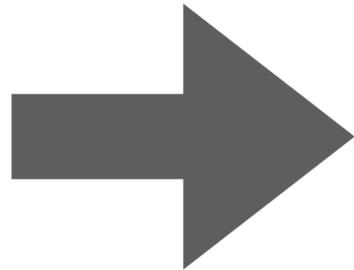
split method

```
S = "a quick brown fox"
```

```
L = S.split(" ")
```

separator

```
"a quick brown fox"
```



```
["a", "quick", "brown", "fox"]
```

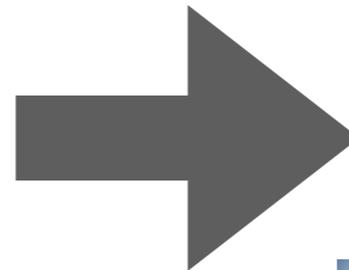
join method

```
L = ["M", "SS", "SS", "PP", ""]
```

```
S = "I".join(L)
```

separator

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



MISSISSIPPI



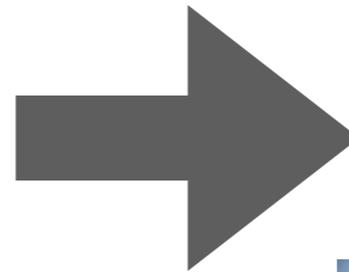
join method

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

separator

what if removed?

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



MISSISSIPPI



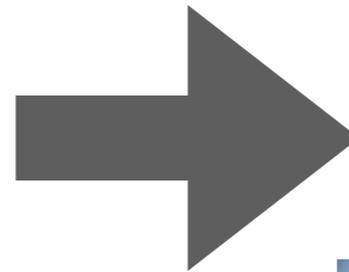
join method

```
L = ["M", "SS", "SS", "PP"]
```

```
S = "I".join(L)
```

separator

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



MISSISSIP



Demo: Censoring Profanity

Goal: write a function to replace curse words with stars

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

Demo: Finding a Median – Next lecture...

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
```

Challenge

1. Command line arguments, as a list

```
import sys  
arg1 = sys.argv[1]  
arg2 = sys.argv[2]
```

2. Random values, from a list

```
import random  
random.choice(["rock", "paper", "scissors"])
```