



Middlebury

CSCI 146: Intensive Introduction to Computing

Fall 2025

Lecture 6: Sequences II (Methods)

Goals for today

- Use sequence functions on `strings` and `lists`.
- Use sequence, `string` and `list` methods to transform those objects.
- Differentiate between mutable and immutable types.
- Predict how immutable and mutable values referenced by variables will be affected by assignments, operators, functions and methods.
- Practice implementing functions that loop over `strings`.

To follow along: create a `class06.py` script where we will write code together.
(in your `cs146` folder)

Warm-up problem: write a function to calculate the average low temperature from this forecast:

Middlebury, VT ☆ ⋮

Tue 23	69°/58°		Showers	💧 54%	↑ S 8 mph
Wed 24	71°/58°		Cloudy	💧 23%	↓ NNE 7 mph
Thu 25	68°/60°		Rain	💧 86%	↖ SE 8 mph
Fri 26	76°/54°		Partly Cloudy	💧 24%	→ WSW 7 mph
Sat 27	76°/56°		Partly Cloudy	💧 20%	↑ S 7 mph
Sun 28	72°/45°		Mostly Sunny	💧 24%	→ WSW 8 mph
Mon 29	71°/44°		Mostly Sunny	💧 6%	↘ NW 8 mph
Tue 30	69°/46°		Mostly Sunny	💧 5%	↓ NNW 9 mph
Wed 01	67°/42°		Mostly Sunny	💧 8%	↓ N 9 mph
Thu 02	69°/42°		Mostly Sunny	💧 9%	↑ S 8 mph
Fri 03	69°/45°		Mostly Sunny	💧 18%	↗ SSW 8 mph
Sat 04	70°/45°		Partly Cloudy	💧 11%	↗ SSW 8 mph
Sun 05	70°/46°		Partly Cloudy	💧 24%	↗ SSW 8 mph
Mon 06	72°/47°		Partly Cloudy	💧 22%	↑ S 7 mph

Possible solution to average temperature problem????

```
1 def average_temp(temps):
2     """
3     Calculates the average temperature from a list of temperatures.
4
5     Args:
6         temps: list of temperatures
7
8     Returns:
9         Average of items in temps.
10    """
11    n_days = len(temps)
12    avg_temp = temps[0]
13    for i in range(1, n_days):
14        avg_temp += temps[i] / n_days
```

divide by
n-days

$$\text{avg} = \frac{t[0] + t[1] + \dots}{n\text{-days}}$$

$$= \frac{t[0]}{n\text{-days}} + \frac{t[1]}{n\text{-days}} + \dots$$

Where's the bug??

We should be able to write this ourselves but, just in case, maybe **Python** can already do parts of this.



Here is a bug-free version:

```
1 def average_temp(temps):
2     """
3     Calculates the average temperature from a list of temperatures.
4
5     Args:
6         temps: list of temperatures
7
8     Returns:
9         Average of items in temps.
10    """
11    n_days = len(temps)
12    avg_temp = 0
13    for i in range(n_days):
14        avg_temp += temps[i]
15    return avg_temp / n_days
```

But yes, **Python** does know how to do this.

Actually, there are many things already built into **Python**.

- **sum**: calculate the sum of the items.
- **min**: get the minimum value.
- **max**: get the maximum value.
- **sorted**: get a new list in which the items are sorted.

```
>>> list_of_strings = ["this", "is", "a", "list", "of", "strings"]
>>> len(list_of_strings)
6
>>> min(list_of_strings)
"a"
>>> max(list_of_strings)
"this"
>>> temps = [58, 58, 60, 54, 56, 45, 44, 46, 42, 42, 45, 45, 46, 47]
>>> sum(temps) / len(temps)
49.142857142857146
```

Functions versus methods (methods are functions defined for "objects" which we call using `.` on that object).

```
>>> help(sum)
...
>>> s = "Hi CSCI0146" # s is an instance of a str
>>> s.index("C")
3
>>> s.count("0")
1
```

How are we supposed to know what methods are defined??

`dir(type_name)` or `dir(variable_name)`

```
>>> dir(str)
...
>>> s = "Hi CSCI0146"
>>> dir(s)
...
```

Investigate! `index`, `count`, `lower`, `upper`, `capitalize`, `replace`, `swapcase`.

Question 1: what is the value of `t` after this code executes?

```
>>> help(str.replace)
```

```
replace(self, old, new, count=-1, /)
```

Return a copy with all occurrences of substring old replaced by new.

```
>>> s = 'Mississippi'
```

```
>>> t = len(s.replace('ss', 'a'))
```

- A. 11
- B. "ss"
- C.
- D. "Miaaiaaippi"

Question 2: what is the value of `t` after this code executes?

```
>>> help(str.replace)
```

```
replace(self, old, new, count=-1, /)
```

Return a copy with all occurrences of substring old replaced by new.

```
>>> s = 'Mississippi'
```

```
>>> t = s.upper().replace('ss', 'a')
```

- A. "Mississippi"
- B. "Miaiaippi"
- C. "MISSISSIPPI"
- D. "MIAIAIPPI"

What about **lists**?

```
>>> dir(list)
...
>>> help(list.pop)
...
>>> list_of_strings = ["this", "is", "a", "list", "of", "strings"]
>>> list_of_strings.pop()
>>> list_of_strings
```

```
>>> list_of_strings.append("strings")
>>> list_of_strings
['this', 'is', 'a', 'list', 'of', 'strings']
```

Investigate! difference between **list.sort** and **sorted**

Recall the concept of (im)mutability from last class.

```
>>> s = "test"  
>>> s.upper()  
>>> s  
"test"
```

```
>>> a = "hi"  
>>> b = a  
>>> a = "bye"  
>>> b  
"hi"
```

Question 3: what is the value of **a** after this code executes?

```
>>> a = [2, 4, 6, 8]
>>> a.remove(4)
>>> a.pop(2)
```

A. [2, 4]

B. [6, 8]

C. [2, 6]

D. [2, 8]

Handwritten annotations:
Initial list: [2, 4, 6, 8] with indices 0, 1, 2, 3 above the elements.
After `a.remove(4)`: [2, 6, 8] (the 4 is crossed out).
After `a.pop(2)`: [2, 6] (indicated by an arrow from the `2` in the code to the handwritten list).

0 2 21 2 0 0 0 0 0 0 0 0 0 0 0

Question 4: what is the value of **a** after this code executes?

```
>>> a = [2, 4, 6, 8]
>>> a.pop(2)
>>> a.remove(4)
```

A. [2, 4]

B. [6, 8]

C. [2, 6]

D. [2, 8]

Question 5: Assume **value** is defined and not empty. Which of the following is valid only if **value** is a string or a list, but not for both types?

A. `value.append("a")`

B. `value[0]`

C.

```
for c in value:
    print(c)
```

D. All are valid for both types

E. All are only valid for only one of string or list

Question 6: If you execute the code below as a program (with the ), what will be displayed in the shell?

```
1 s = "CS"
2 t = "cs"
3 s.lower()
4 for i in range(len(s)):
5     print(s[i] + t[i])
```

Cc
Ss

A.

cc
ss

B.

Cc
Ss

C.

CSCS
CSCS

D.

cC
sS

Question 7: what is the value of **t** after this code executes?

```
>>> s = "abc"  
>>> t = s  
>>> s = s.upper()
```

A. "abc"

B. "ABC"

C. "s"

D. "S"

Exercise: make a password generator! Define a function **password_gen** with one parameter (**n**: the number of characters in the password to create).

Suggestions:

- Define a string or list with a set of allowed characters.
- Use **random.randint** to sample a character from a string or list of allowed characters.

```
from random import randint

CHARS = "abcdefghijklmnopqrstuvwxyz0123456789_!@#$%^&*"

def password_gen(length):
    """
    Generate a random password

    Args:
        length: number of characters in the password

    Returns:
        Password string
    """
    result = ""
    for i in range(length):
        result = result + CHARS[randint(0, len(CHARS)-1)]
    return result
```

Summary & Reminders

- Use built-in functions and methods when you can! (and use `dir` and `help` to learn what/how to call them).
- Have a look at feedback on Programming Assignment 1 (resubmit until Thursday October 2nd).
- Programming Assignment 2 due Thursday (initial submission). Read the instructions carefully!
- Quiz 3 on Friday (see notes linked on calendar).
- Help hours: see [go/smith](#) (Smith Gakuya, ASI) and [go/cshelp](#).