LISTS AS SEQUENCES

WARM UP: WHAT'S ALL THIS, THEN?

```
sentence = "ships      at  a distance have every man's wish on board"
count = 0
last_character = "" # this is the empty string as a placeholder
for character in sentence:
    if last_character != " " and character == " ":
        count += 1
    last_character = character
if last_character != " ":
    count += 1
print(count)
```

Describe to a partner: what does this do and how does it work?

WARM UP: WHAT'S ALL THIS, THEN?

```
sentence = "ships      at  a distance have every man's wish on board"
count = 0
last_character = "" # this is the empty string as a placeholder
for character in sentence:
    if last_character != " " and character == " ":
        count += 1
    last_character = character
if last_character != " ":
    count += 1
print(count)
```

Set up our input variable (`sentence`) and our other utility variables (`count` and `last_character`).

WARM UP: WHAT'S ALL THIS, THEN?

```
sentence = "ships      at  a distance have every man's wish on board"
count = 0
last_character = "" # this is the empty string as a placeholder
for character in sentence:
    if last_character != " " and character == " ":
        count += 1
    last_character = character
if last_character != " ":
    count += 1
print(count)
```

Visit each `character` in the sentence one by one.

WARM UP: WHAT'S ALL THIS, THEN?

```
sentence = "ships      at  a distance have every man's wish on board"
count = 0
last_character = "" # this is the empty string as a placeholder
for character in sentence:
    if last_character != " " and character == " ":
        count += 1
    last_character = character
if last_character != " ":
    count += 1
print(count)
```

If we are going from a letter character (checking `last_character`) to a space character (checking `character`), we've found a new word. **Why?**

WARM UP: WHAT'S ALL THIS, THEN?

```
sentence = "ships      at  a distance have every man's wish on board"
count = 0
last_character = "" # this is the empty string as a placeholder
for character in sentence:
    if last_character != " " and character == " ":
        count += 1
    last_character = character
if last_character != " ":
    count += 1
print(count)
```

Next iteration of the loop, the current `character` will become the previous one, so we save it for next time in `last_character`.

WARM UP: WHAT'S ALL THIS, THEN?

```
sentence = "ships      at  a distance have every man's wish on board"
count = 0
last_character = "" # this is the empty string as a placeholder
for character in sentence:
    if last_character != " " and character == " ":
        count += 1
    last_character = character
if last_character != " ":
    count += 1
print(count)
```

Slightly fussy bit at the end to make sure the last word gets counted since we won't have crossed from non-space to space at the end of the string.

CALLING FUNCTIONS ON STRINGS

THE PATTERN

Python has a bunch of useful functions that you can call on a string `s` to analyze or modify them. To use them, we usually follow a pattern like this:

```
result = s.function()
```

- `function()` is the **function call**, where we'd replace `function` with the name of the actual one like `strip` or `split` or `upper` or ...
- Use the `.` syntax to call the function on a particular string
- Functions outside of PennDraw often **return** (produce) a value
  - that value could have any type: another `str`, a `bool`, etc.
  - save that new value inside of a variable if you want to use it later.

Some Examples!

STRIP()

We can use the `.strip()` function on a string to remove any leading or trailing whitespace.

Whitespace characters are characters that just add "spacing" but don't display like typical chraracters: tab (`'\t'`), space (`' '`), newline (`'\n'`)

```
line = "      look at all   that wasted space!                "
stripped_line = line.strip()    # note: we're producing a new string
print(len(line))                # 🖨️➡️ 61
print(len(stripped_line))       # 🖨️➡️ 32
```

Returns a new string; original one is unmodified.

ISUPPER() AND UPPER()

`.isupper()` determines whether all of a string's letter characters as all uppercase and returns a `bool`.
`.upper()` creates a new string where all of the letter characters are uppercase; it returns a new `str`.

```
course = "CIS 1100"
result = course.isupper()
print(result)                    # 🖨️ ➡️ True

greeting = "heya"
better_greeting = greeting.upper()
print(better_greeting)           # 🖨️ ➡️ HEYA
```

ACTIVITY C1

Get as far as you can. There are a lot of steps here. Remember `strip()`, `isupper()`, and `upper()`

A user signs up for your social network and they enter their username, which is available to you in the variable `username`. Do the following:

- If the username has any whitespace surrounding it, get rid of that first.
- Then, if any of the first three characters are uppercase letters, print `"good start"`
- Finally, print a version of the username where ALL letter characters become uppercase.

`" YES1no "` should print "good start" and then `"YES1NO"`

SOLUTION

`" YES1no "` should print "good start" and then `"YES1NO"`

```
username = "   YES1no  "
username = username.strip()     # remove whitespace and save
first_three = username[0:3]     # look at only the first 3 chars
any_upper = False
for char in first_three:
    if char.isupper():
        any_upper = True
if any_upper:
    print("good start")
username = username.upper()     # convert to uppercase and save
print(username)                 # done!
```

FROM STRINGS TO LISTS

SPLIT()

What if we want to get all the "words" that make-up a string?

The `split` function returns a list of strings containing all the words that have whitespace between them.

```
line = "I am 2 late"
tokens = line.split()
print(line) # ["I", "am", "2", "late"]
```

Note how all the elements are still strings!

LET'S REMEMBER SOME STUFF ABOUT LISTS

- Lists are **sequences**, or ordered collections of data
  - Denoted by square brackets, e.g. `[1, 2, 4, 8, "yes"]`
  - Can contain data of mixed/any type
- Supports **indexing**, **slicing**, and **length** with the same syntax.
- Like a string, you can add stuff to the end of a list...
  - `+` concatenates two *lists* together and produces a new list
  - `l.append(e)` puts `e` at the end of the existing list `l`

What review questions do you have about list behavior that we can walk through?

STRINGS AS SENTENCES AS LISTS OF WORDS

`s.split()` takes the string `s` and returns a list of all of the space-separated "words" in it.

Quick practice (**L1**): write down a short line from a song/movie/book as a string.

```
s = "He made Thebes in oranges and lemons and limes"
```

Write a few short lines of Python to print out the **number of words** in your quote and then **just the third word on its own.**

Check your work with a partner. Also, find out what their quoted line is from (for fun).

STRINGS AS SENTENCES AS LISTS OF WORDS

`s.split()` takes the string `s` and returns a list of all of the space-separated "words" in it.

Quick practice (**L1**): write down a short line from a song/movie/book as a string.

```
s = "He made Thebes in oranges and lemons and limes"
words = s.split()
num_words = len(words)
third_word = words[2]
```

FILES

FILES

On computers we have things called **files**.
Files are where we store information that the computer can still access even after the computer turns off and on again.

We have already use files before: our programs are stored in `.py` files.

For now, we can assume that the contents of files are all characters. For text files like these, we think of files as being made of a "sequence" of lines of text.

```
Is there anybody     # first line
                     # second line
out there?           # third line
```

OPEN() AND CLOSE()

To read a file, we need to create a file "object" associated with that file. We can create a variable holding a file object with the `open()` call.

```
# opens the file "filename.txt" with "r" (Reading) enabled
example_file = open("filename.txt", "r")
```

When we are completely done with a file, we need to close it

```
example_file.close()
```

...but what do we do in between the opening and closing?

READLINE()

Once we have an open file object, we can use `readline()` to read a line from the file. TRY IT!

```
# on website as first_three_lines.py
my_file = open("beep.boop", "r")    # open beep.boop for reading
for i in range(3):
    line = my_file.readline()
    print(line)
my_file.close()
```

The next time we call `readline()` we get the next line of the file. These File objects remembers our position in the file.

PRACTICE (C2)

Assume we have a file named `beep.boop` with the layout:

```
this file has 3 lines after this
line 0
line 1
line 2
```

Write some code that can read a file like this and print out all the lines but the first. *Write your solution so that it'd work if file had any number, not just* `3`.

- You should probably use: `open(filename, "r")`, `file.readline()`, `file.close()`, `string.strip()`, `string.split()`