CIS 1100

Loops, Command Line Args & File Reading! (Lecture)

Python
Spring 2025
University of Pennsylvania

For Loops

We can use `for` to go over each item in a sequence:

```python
for character in "Hello!":
    print(character)
```

The loop will:

- Run's the code in the loop once for each item in the sequence
- The first time we run the "body" (code *in* the loop), our loop variable (`character` in this example) will be the first item of the sequence
- after we finish the body once, we repeat it for the next item in the sequence. Keep going until there are no more items in the sequence

Review: For Loops

This can be used to repeat code!

Instead of :

```
print("Hello!!!!")
print("Hello!!!!")
print("Hello!!!!")
...
print("Hello!!!!")
```

We can do something better:

```
for i in range(0, 10):
    print("Hello!!!!")
    # note how we aren't required to use i
```

For Loop Practice

FIGURE THESE OUT BY READING THE CODE, DON'T JUST RUN IT

Consider the following loop, what is the final value of `nums`? **(S7)**

```
nums = []
for i in range(0, 4):
    nums.append(i * len(nums))
```

What is the value of `skills` after this code is run? **(S8)**

```
skills = [18, 88, 20, 82, 91, 78, 15]
for i in range(len(skills) - 1):
    if skills[i] >= 20:
        skills[i] = skills[i] + 7
    else:
        skills[i] += 2
```

For your consideration: what are the two different ways we're modifying lists here?

Review: For Loops w/ `enumerate()`

We can use `enumerate()` to get the indices and items of the sequence paired together:

```python
nums = [3, 2, 5]
for index, item in enumerate(nums):
    print(f"Index {i}: {item}")
```

prints:

```
Index 0: 3
Index 1: 2
Index 2: 5
```

Each iteration of the loop gives us two values to work with, so we choose two variable names.

Enumeration Practice

We want to write some code to find the index of the *longest* string in a list. Fill in the loop body. **(C12)**

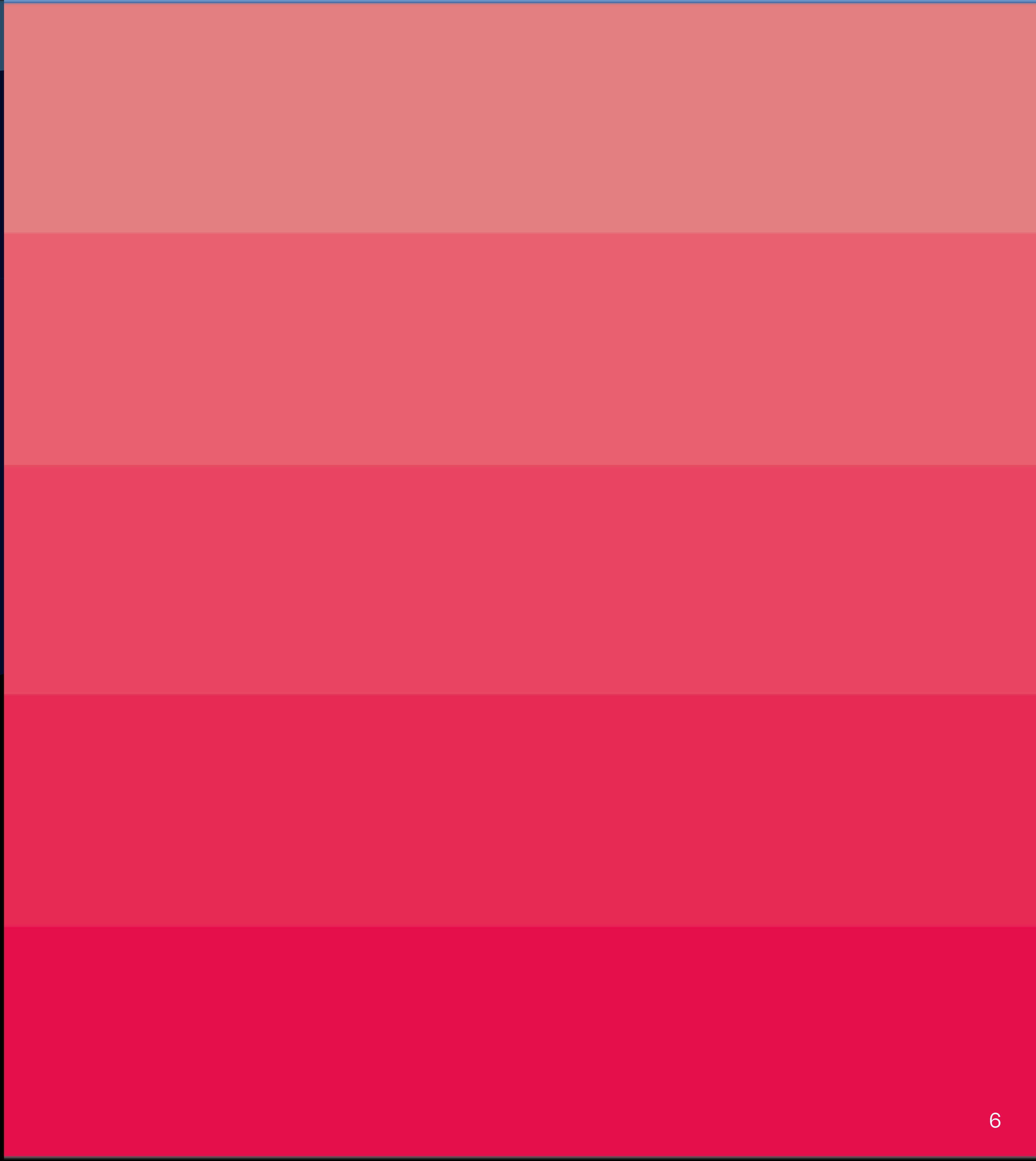
```
strings = ["My", "Anti", "Aircraft", "Friend"]

index = 0
longest_str = strings[0]

for i, string in enumerate(strings):
    # TODO: Fill out this loop




print(f"The longest string is {longest_str} at index {index}")
```

Applying Loops to PennDraw

We can apply loops to pen-draw!

Consider `gradient.py`

```
import penndraw

penndraw.set_pen_color(233, 15, 75)
penndraw.filled_rectangle(0.5, 0.1, 0.5, 0.1)

penndraw.set_pen_color(233, 43, 88)
penndraw.filled_rectangle(0.5, 0.30, 0.5, 0.1)

penndraw.set_pen_color(233, 71, 101)
penndraw.filled_rectangle(0.5, 0.5, 0.5, 0.1)

penndraw.set_pen_color(233, 99, 114)
penndraw.filled_rectangle(0.5, 0.7, 0.5, 0.1)

penndraw.set_pen_color(233, 127, 127)
penndraw.filled_rectangle(0.5, 0.9, 0.5, 0.1)

penndraw.run()
```



Looping in PennDraw

```
penndraw.set_pen_color(233, 15, 75)
penndraw.filled_rectangle(0.5, 0.1, 0.5, 0.1)

penndraw.set_pen_color(233, 43, 88)
penndraw.filled_rectangle(0.5, 0.30, 0.5, 0.1)

penndraw.set_pen_color(233, 71, 101)
penndraw.filled_rectangle(0.5, 0.5, 0.5, 0.1)

penndraw.set_pen_color(233, 99, 114)
penndraw.filled_rectangle(0.5, 0.7, 0.5, 0.1)

penndraw.set_pen_color(233, 127, 127)
penndraw.filled_rectangle(0.5, 0.9, 0.5, 0.1)
```

The "green" value of the color increases by `28` with each repeated chunk, so we could represent it with a formula in terms of i, the iteration number:

$$\text{green} = 15 + 28i$$

Looping in PennDraw

```
penndraw.set_pen_color(233, 15, 75)
penndraw.filled_rectangle(0.5, 0.1, 0.5, 0.1)

penndraw.set_pen_color(233, 43, 88)
penndraw.filled_rectangle(0.5, 0.30, 0.5, 0.1)

penndraw.set_pen_color(233, 71, 101)
penndraw.filled_rectangle(0.5, 0.5, 0.5, 0.1)

penndraw.set_pen_color(233, 99, 114)
penndraw.filled_rectangle(0.5, 0.7, 0.5, 0.1)

penndraw.set_pen_color(233, 127, 127)
penndraw.filled_rectangle(0.5, 0.9, 0.5, 0.1)
```

(L13) Give similar formulae in terms of i for the "blue" value of the color and the y-position of the rectangle.

Looping in PennDraw

```
penndraw.set_pen_color(233, 15, 75)
penndraw.filled_rectangle(0.5, 0.1, 0.5, 0.1)

penndraw.set_pen_color(233, 43, 88)
penndraw.filled_rectangle(0.5, 0.30, 0.5, 0.1)

penndraw.set_pen_color(233, 71, 101)
penndraw.filled_rectangle(0.5, 0.5, 0.5, 0.1)

penndraw.set_pen_color(233, 99, 114)
penndraw.filled_rectangle(0.5, 0.7, 0.5, 0.1)

penndraw.set_pen_color(233, 127, 127)
penndraw.filled_rectangle(0.5, 0.9, 0.5, 0.1)
```

How can we write this to use a loop instead? **(C14)**

```
for i in range(0, 5):
    # TODO
```

Command Line Args

When we run a program we usually type something like

```
python my_program.py
```

We can then send additional information to the program via `input()`, but we can also specify some information when we run the program through "Command Line Arguments".

This means we could type something like

```
python greeting.py Harry
```

to specify some information at the same time as we start the program.

`sys.argv`

We can `import sys` and use `sys.argv` to get command line args

Consider the file `args.py`

```
import sys
print(sys.argv)
```

Run with:

```
python args.py Joel Ra mir ez
```

prints: `['args.py', 'Joel', 'Ra', 'mir', 'ez']`

Note: argv has EVERYTHING after `python` in the commnad. Also note that it is a list of strings, if we want other types we have to explicitly convert.

Command Line Args

Consider we have the following program called `greeting_argv.py`.

```python
import sys

print("Hello, " + sys.argv[1] + "!")
```

And we run it with the command: `python greeting_argv.py Harry`

then we will get `Hello, Harry!`

Command Line Args

Consider we have the following program called `greeting_argv.py`.

```python
import sys

print("Hello, " + sys.argv[1] + "!")
```

And we run it with the command: `python greeting_argv.py Harry`

then we will get `Hello, Harry!`

What if we just did `python greeting_argv.py`?

```
Traceback (most recent call last):
  File "/some_path/greeting_argv.py", line 3, in <module>
    print("Hello, " + sys.argv[1] + "!")
IndexError: list index out of range
```

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.

When we run the program, the computer reads the specified `.py` file

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()
```

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.

print_first_three_lines.py

```
import sys

my_file = open(sys.argv[1], "r")
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.

DEMO: `python first_three_lines.py hello.txt`

`strip()`

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

Whitespace characters are characteres that just add
"spacing" but don't display like typical chraracters.

Whitespace characters: tab (`'\t'`), space (`' '`), newline (`'\n'`)

`readline()` returns a line from a file, with the newline character
(`'\n'`) at the end. We can remove this newline if we call `strip()`:

```
line = my_file.readline().strip()
```

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!

Practice:

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

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

Please write some code that can read a file like this and print out all the lines but the first.

You should use `readline()` and assume that the file can have any number instead of `3`.

(C16)

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

Reminder:

- There is another check-in due before lecture as always.
  - Friday's check-in will have an "exit-ticket" for you to submit questions and metrics about the course.
- I have Office Hours later this afternoon and on Monday mornings
- Joel has OH on Mondays and Wednesdays @ 4pm
- HW01 is due tonight (2/5) at 11:59pm
- Expect HW02 to be released early tomorrow

Enumeration Practice Answers

```
strings = ["My", "Anti", "Aircraft", "Friend"]

index = 0
longest_str = strings[0]

for i, string in enumerate(strings):
    if len(string) > len(longest_str):
        index = i
        longest_str = string

print(f"The longest string is {longest_str} at index {index}")
```