CIS 1100

File Reading, Nested For, and List Comp (Lecture)

Python

Fall 2024

University of Pennsylvania

Review: For Loops w/ enumerate()

We can use `enumerate()` to get both the items and indexes of a sequence:

```
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
```

Enumeration Practice

We want to write some code to find the index of the longets string in a list, finish the code: (C12)

```
strings = ["Ants", "From", "Up", "There"]

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}")
```

Files

On computers we have something 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`

without a loop

The code we had on the previous slide is equivalent to the following code:

```
import sys

my_file = open(sys.argv[1], "r")
line = my_file.readline()
print(line)
line = my_file.readline()
print(line)
line = my_file.readline()
print(line)
my_file.close()
```

Readline and loops

which is also equivalent to:

```
import sys

my_file = open(sys.argv[1], "r")
for i in range(7, 10):
    line = my_file.readline()
    print(line)
my_file.close()
```

This is to show that the actual value of `i` does not matter for these examples. What matters is that the loop will run 3 times. With files we always start from the beginning and each call to `readline()` gets the next line of the file.

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 `\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 (C14):

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`.

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

Note how we kinda have to use loops for this...

Nested For

We can have loops inside of loops

What does this print? (S7)

```
for e in range(1, 4):
    prod = 1
    for x in range (1, e + 1):
        prod = prod * x
    print(prod)
```

List Comprehension Syntax

Recall a `for` loop that copies all characters of a string into a list:

```
new_list = []
for character in "ABCD":
    new_list.append(character)
```

"For each character in the string, place that character in the new list I am creating."

👇👇👇

```
new_list = [character for character in "ABCD"]
```

List Comprehension Syntax

A basic list comprehension can be written like so:

```
[<expression> for variable in sequence]
```

- `for variable in sequence` works exactly like a regular `for` loop
  - Each element in `sequence` gets visited one-by-one and is given the name `variable`
- The value of `<expression>` is appended to the output list for each element in the sequence
  - Usually write `<expression>` in terms of `variable`
- A new list is created!

Recall: Getting Non-Zero Exam Scores

This loop-based version...

```
exam_scores = [100, 0, 89, 93, 78, 67, 0]
non_zeroes = []                 # [] is a list with no contents
for score in exam_scores:       # For each score from the list,
  if score > 0:                 # if that score is not zero,
    non_zeroes.append(score)    # add that score to the end of the new list.
```

...can be rewritten to:

```
exam_scores = [100, 0, 89, 93, 78, 67, 0]
non_zeroes = [score for score in exam_scores if score > 0]
print(non_zeroes)
```

🖨️👇

```
[100, 89, 93, 78, 67]
```

List Comprehension Practice (L11)

Fill out the list comprehension so we have a list with each element of `values` with 10 added to it.

```
values = [0, 5, 10, 23]
values_added_ten = [
    # TODO: fill in this list comprehension
]
```

Write a list comprehension that gets a list of all even length strings from `names`:

```
names = ['bob', "steve", "pete", "me", "abcde"]
even_naes = [ ______________ ]
```

more practice on the next slide

More Practice (L13)

Write the equivalent of this code:

```
strings = ["arriving", "somewhere", "but", "not", "here"]
result = []
for i,string in enumerate(strings):
    new_entry = (' '*i)+string
    result.append(new_entry)
```

but use a list comprehension, e.g.:

```
strings = ["arriving", "somewhere", "but", "not", "here"]
result = [
    # TODO: fill in this list comprehension
]
```

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.
- We highly recommend you look back at some of these lecture examples for the upcoming homework assignment