

Comp151: lists and files



Reading Assignment



- Read chapter 4 in the book after these slides
-

Getting some data into lists



- So far we have written the data for lists into the program
- How else could we get that data with what we know so far?

Getting some data into lists



- So far we have written the data for lists into the program
- How else could we get that data?
 - Well we could ask the user for it.
- But usually we want to just get the data for the user how might we do that (even if we don't know so far?)

Getting some data into lists



- So far we have written the data for lists into the program
- How else could we get that data?
 - Well we could ask the user for it.
- But usually we want to just get the data for the user how might we do that (even if we don't know so far?)
 - The two I usually here are
 - Get it from a file
 - Get it from the internet
 - The file part is really easy with a text file
 - The internet part is harder (if only because so many sites are trying to charge for it) and we will put it off for now.

Files



- We can split files into two types for our purposes
 - Text files, which contain just printable characters in the zeros and ones
 - (even if some of those are used for formatting like in html)
 - Binary file which contain any arbitrary data,
 - Often the zeros and ones cannot be converted to text in strings
- Now let me have a lucky volunteer or few give me some examples of each type of file

Opening a text file



- To open a file easily in python use built in function open
 - Inside of open
 - First put the file name as a string
 - Then put either
 - 'r' (open for reading)
 - 'w' (open for writing – will overwrite the whole file)
 - 'a' (open for appending – will write new stuff to the end of the file)
- Here give standard warning:
 - Never open a .py file, for your own sanity
- `my_file = open("fun.txt", "r")`

Reading in a text file



- To read all the lines of a file into a list of strings use the `readlines()` function
 - Use the `file.readlines()` function
 - Now `all_lines` is a list of strings
 - Each line is one string in the list.
 - Draw it on the board if needed
- Lets try it
 - Grab the `games.txt` file (from the class website?)
- `my_file = open("fun.txt", "r")`
- `all_lines = my_file.readlines()`

Looping through a list



- If we want to do something for every item (element) in a list
 - We call it “looping through the list” or **iterating** over a list
 - This is a very common thing in programming and python has a couple of easy constructs to do this.
- Basic looping through the list
 - for <new variable> in <list variable>:
 - do something to each item in the list here
- Notice that the part where we do something is indented

Example



- Let's continue the example from earlier to do something for each line in the file
- For this first pass, lets just print each line
- Lets be sure to git commit after this
- `my_file = open("fun.txt", "r")`
- `all_lines = my_file.readlines()`
- for line in all_lines:
 - `print(line)`

Code Blocks



- A **code block**, is used to determine which code belongs together as a “chunk”
 - Determines what code is part of a for loop and what code is after the loop
 - Code blocks in python are determined by indentation
 - All code indented to the same level is part of the same code block
 - Till we reach another indent level
- `my_file = open("fun.txt", "r")`
- `all_lines = my_file.readlines()`
- `for line in all_lines:`
 - `print(line)`
 - `print("will show after each line")`
- `print("this will print once after")`
- `#yellow highlighted is part of the`
`#code block in scope of the for`
`#will be run for every item in list`

with as



- Alternative way to get files
 - Keywords **with** and **as**
 - Creates a codeblock based around a new variable
 - When the code block for with ends
 - Resources released
 - Here File will be closed
- Especially useful for reading/writing files and using lots of files.

```
with open("fun.txt", "r") as my_file:  
    all_lines = my_file.readlines()  
    for line in all_lines:  
        print(line)  
        print("will show after each line")  
    print("this will print once after")
```

Splitting a string



- Recall string objects can do work for us
- One of the things strings can do for us is to split itself into a list of smaller strings
 - Splitting at a particular character
 - Eg:
 - `<string>.split(<char to split>)`
- `demo = "this that these those"`
- `#lets split on the space char`
- `words = demo.split(" ")`
- `#words is`
`["this", "that", "these", "those"]`
- `print(words[1])`
- `#which word gets printed here?`

Now let's extend our example



- As an exercise for the students
- Lets extend our earlier example
- After reading in the file as a list of strings, one for each line and looping through each line, lets split the lines print just the games first
- Then extend it and print first the game, then the total money earned
- So the output would look like:?
-

Scope and Indenting



- Indenting to create a code block
 - Helps determine **scope** of variables
 - **Scope**: where a variable is usable and visible.
 - If you create a variable in a block, it is available till the end of the block
 - Variables created at the leftmost indent level are available till the end of the file
- ```
file_handle = open('silly.txt', 'r')
```
- ```
all_lines = file_handle.readlines()
```
- ```
for line in all_lines:
```

  - ```
course_and_name = line.split(':')
```
 - ```
course = course_and_name[0]
```
  - ```
name = course_and_name[1]
```
 - ```
print(course)
```
  - ```
print(name)
```
- ```
print("those are your CS
required")
```

## Indenting and your book



- Your book has a slightly different approach to the same material on indenting and code blocks, be sure to read it as well (chapter 4) so that you get both perspectives.

# Lists of numbers



- Sometimes we just want to do something multiple times
  - For that we can use a list of numbers.
  - The `range` function will produce a list of numbers\*
  - By default list from zero up to but not including the cutoff
- `numbers = range(10)`
- # numbers is `[0,1,2,3,4,5,6,7,8,9]`

\* it used to be an actual list – now almost a list

# Using number lists



- Number lists created using range usually used to run loops a fixed number of times (one for each number in the list)
  - Can start from any number
  - for number in range(10):
    - print("gets printed 10 times")
  - 
  -
- ```
for num in range(10, 20):  
    print(f"counting from 10-19 currently at {num}")
```

Counting by other than 1



- Can 'skip count'
-
-
-
- Or even count backwards

```
for num in range(10, 100, 10):  
    print(f"skip counting currently at {num}")
```

-
-
-

```
for num in range(10, 1, -1):  
    print(num)  
print("blastoff")
```

tuples



- Tuples are sort of like immutable/read-only lists.
 - Like strings, once you create a tuple you can't change it.
 - But otherwise, for not mutating operations, can do the same things on tuples and lists
 - Because tuples are read-only, often do have different types in a tuple
 - Create literal tuple with ()
- `student = ("john", 3.2, 92)`
- `student[1]` #will be 3.2

slices



- If you want a sub-sequence of any python sequence type use a slice
 - Use board to discuss slicing indexes vs locations
 - The slice copies part of the sequence (string/list/tuple) into a new one.
 - Use `sequence[X:Y]` to make a copy of the subsequence from X to Y
- `courses = ["comp151", "comp152", "comp199", "comp206", "comp250"]`
- `first_year_courses = courses[0:3]`
- `# first_year_courses has contents ["comp151", "comp152", "comp199"]`

Slices II



- But programmers are 'lazy'
 - So a shortcut available if you want to go all the way to one end of the original with a slice
 - Omit the first slice index will start from the beginning
 - Omit the second slice index will go to the end
 - So what do we have here:
 - Also lets try with strings
- `courses = ["comp151", "comp152", "comp199", "comp206", "comp250"]`
- `Question1 = courses[:3]`
- `Question2 = courses[2:]`
- `Question3 = courses[:]`

len



- Sometimes we need to know how many things are in a sequence(string/tuple/list)
 - Python has one function to do all that
 - `len(sequence)` will tell use how many 'things' are in the sequence/linear collection
- `courses = ["comp151", "comp152", "comp199", "comp206", "comp250"]`
- `how_many = len(courses)`
- `#how_many` has value 5
- `desc = "A first year seminar"`
- `num_chars = len(desc)`
- `#num_chars` has value 20 (spaces count as characters)

Reading Assignment



- At this point read chapter 4 to page 64 in your book,
- Now lets add the podcast to the assignment
 - Also please listen to episode 78 (Aug 22 2026) of “the programming podcast”
 - There are links to youtube and spotify on the page when you scroll to the Aug 22 episode
 - <https://www.dthompsondev.com/podcast>
 - We will discuss in class in one week.
- Let’s look at Project2