37 1/2 +4 E.C. +1 Class Bump 42 1/2 / 40

Denison CS-181/DA-210 Quiz 1

- February 19, 2021
- Each questions is worth 10 points, and the quiz is a total of 40 points.

Instructions

1. This is a **50 minute test**. It must be turned in, at the latest, at the end of class period. For in-person students, that means giving the instructor the written test with solutions. For remote or virtual-remote students, that means uploading the `.md` text file with answers following each of the questions. Upload is to the Notebowl entry for this quiz. Remote and remote-virtual can also choose to print the `.pdf` version of the test, handwrite their answers, and upload pictures of each of the completed pages.
2. **No electronic resources**
   - No notebooks; no web pages; no documentation pages; no PDFs; no online tools; no Jupyter lab; no Jupyter notebook, no Python execution.
   - Exception: Atom editor to enter answers in a markdown file for remote or remote-virtual students without access to printer
3. You are permitted a **handwritten** 3 by 5 index card, one-sided, with any notes you choose to include.
4. No docstrings are required for functions you write on a quiz.
5. After the quiz is complete/turned in, students that opted in for study groups should upload a picture to the Notebowl submission site.

Q1: Python Features

A. List Comprehension Semantics

What is the value of `L3` from the following list comprehension?

```
L1 = ["hello", "goodbye", "hi"]
L2 = [2, 1, 2]

L3 = [a * b for a, b in zip(L1, L2) if len(a) <= 5]
```

3 1 2

```
# Write answer here
```

L3=["hellohello", "hihi"] 3/3

B. Writing a List Comprehension

Write a function

```
failsThreshold(data, threshold)
```

that **uses a list comprehension** to create and return a new list whose values are those elements in `data` not as great as the specified `threshold` value. The original list (`data`) should not be modified. For example, if `data = [6,4,-8,7]` and `threshold = 6` then you return `[4, -8]`. For partial credit, a `for` loop solution may be used.

```
# Write answer here
```

```
def failsThreshold(data, threshold):
    return [x for x in data if x < threshold]
```

3 / 3

C: Writing a Lambda

The formula for the volume of a cylinder is $\pi \cdot r^2 \cdot h$

where r is the radius and h is the height of the cylinder. Assuming the `math` module has been imported, and that π is available as the module constant, `math.pi` , write a lambda expression to compute the volume of a cylinder and assign this lambda to Python variable `volume` . Partial credit will be given for a `def` -based function.

```
# Write answer here
```

```
volume = lambda r,h: (math.Pi)*(r**2)*(h)
```

3 / 3

Q2: Regular Expressions

A: Finding Matches

In the following, you are given a regular expression pattern and a target. You are to write down a **list of the strings matched** by the regular expression pattern against the target. Think of this as your manually emulating what a `findall()` would do for the pattern and target pair.

```
pattern = r'ab{1,3}c?'
target = """
  abc
  ac
  aab
  abbb
  bbc
  abbbbbc
"""
```

ab (1 to 2 times) c

+ 1/2 (5)

```
# Write answer here
```

["abC", "ab", "ab", "ab"]

more b's consumed

```
pattern = r'ab{1,3}c?'
target = """
  abc
  ac
  aab
  abbb
  bbc
  abbbbbc
"""
re.findall(pattern, target)

['abc', 'ab', 'abbb', 'abbb']
```

solution

B: Pattern Differentiation

Given the table below, constructed as a single string, find a single regular expression that matches all the items in the first column (the entirety of the match, but stopping before the trailing spaces) but none of those in the second column. A disjunction of the literals in the first column will not be awarded any points. Nor will a solution that uses the line-and-column nature of the way the string was constructed. (I.e. if the string were all in one line, your solution should still match correctly.)

```
target = """
 Match       | No Match
-------------|---------
 affgfking   | fgok
 rafgkahe    | a fgk
 bafghk      | affgm
 baffgkit    | afffhk
"""
```

5 / 5

No boundaries needed b/c of \w+ at \w+

```
#----------------------------
pattern = r"\w+fg\w*k\w*"
#----------------------------
```

- : can pick up first fg in lines 1 and 4
- : literals to "anchor" the other restrictions
- : this could be more restrictive, like [fk]?
- : consume rest of "word"

```
# Write answer here
```

Pattern = r"\w+ | b\w+ | \w+ki\w+"

focus on fg in the prefix or suffix

Your solution works, but it is using disjunction instead of really doing pattern difference.

Q3: Processing a Delimited Data File

Suppose you are given a file that uses a variation of the delimitted file format we have used so far this semester. In particular, a file formatted this way has:

1. Exactly one comment line as the first line in the file. This line begins with a hash `#` and may contain unstructured information. This line, while present in the file, is to be skipped over in file processing.
2. Exactly one header line as the second line in the file. This line contains the names of each of the column headers, and these column names are all separated by the three character sequence of a space, a vertical bar (`'|'`), and then another space.
3. All the remaining lines are data-carrying lines with fields that correspond to the header names, and are also separated by the three character sequence of a space, a vertical bar (`'|'`), and then another space.

The following is an example:

```
# Country Area and Population Data Set
country | area | population | order
BR | 8.516 | 200.40 | 1
RU | 17.100 | 143.50 | 2
IN | 3.286 | 1252.00 | 3
CH | 9.597 | 1357.00 | 4
SA | 1.221 | 52.98 | 5
```

Further, suppose that you may have **any number of columns** and **any number of rows**.

If we do not need to be concerned with any data type conversions (so you are not required to convert the character numbers in the above example to float values), then we can use a variation of the file processing seen in prior homeworks and in class.

Write a function

```
read_vbsv_LoL(path)
```

where `vbsv` is acronym for vertical-bar-separated-value, and `path` is a path to a file formatted as described above. Your function should return **both** a list of column names **and** a list of row lists representation of the data (LoL). Header names and field values should not have any leading or trailing whitespace.

You need not concern yourself with how `path` is constructed before invocation of your function, and may assume that it refers to an actual file that is formatted properly.

```
# Write answer here
```

On other side

~~def read_VBSV_LoL(Path):~~
~~header=[]~~
~~data=[]~~
~~fObj=open(Path)~~
~~header=fObj.readline().strip().split("|")~~
~~for line in file:~~
~~data.append(line.strip().split("|"))~~
~~fObj.close()~~
~~return header, data~~

def read_VBSV_LoL(Path): 9½|10

header=[]
data=[]
fObj=open(Path)
for item in fObj.readline().strip().split(" "):
if item != "|":
header.append(item) ✓ OK

LoL=[]
for line in file:
field=line.strip().split(" "):
for item in field:
if item != "|":
data.append(item)
LoL.append(data)
✓ fObj.close()
return header, data

Q4: Using Representations

Sometimes we want, after the fact, to convert a set of values in a column of a LoL representation or a DoL representation from strings to an appropriate data type. For instance, the `read_vbsv_LoL()` function results in all values being strings. But we would want column indices 1 and 2 to be converted to `float` values and column 3 to be converted to `int` values.

Since `float` and `int` are functions, they can be passed to a function. Such passing of a function to another function will used in both parts of this question.

Using the LoL Representation

Write a function

```
convert_LoL_column(LoLdata, column_index, cfun)
```

that, **in place**, converts all values in the column specified by `column_index` using the conversion function passed as `cfun`, assuming `LoLdata` is a list of row lists representation, and the values in the given column are strings, and thus legal for a function that converts from a string to a data type. Your function does not return anything.

You need not perform an invocation, but the following illustrates how we might get an LoL representation and then invoke the function you are writing:

```
columns, LoL = read_vbsv_LoL(path)
convert_LoL_column(LoL, 2, float)
convert_LoL_column(LoL, 3, int)
```

5/5

```
# Write answer here
```

```
def convert_LoL_column(LoLdata, column_index, cfun):
    for item in LoLdata:
        item[column_index] = cfun(item[column_index])
```

My solution:

```
def convert_LoL_column(LoLdata, column_index, cfun):
    for row in LoLdata:
        row[column_index] = cfun(row[column_index])
        return
```

LoLdata is a list of lists, so row is, itself a list representing one row, and column_index, in that row, is the one to convert.

Using the DoL Representation

Write a function

```
convert_DoL_column(DoLdata, column_name, cfun)
```

that, **in place**, converts all values in the column specified by `column_name` (a string with the column name) using the function passed as `cfun`, assuming `DoLdata` is a dictionary of column lists representation, and the values in the given column are strings, and thus legal for a function that converts from a string to a data type. Your function does not return anything.

You need not perform an invocation, but the following illustrates how we might get an DoL representation and then invoke this function:

```
DoL = read_vbsv_DoL(path)
convert_DoL_column(DoL, 'population', float)
convert_DoL_column(DoL, 'order', int)
```

4 1/2 / 5

```
# Write answer here
```

```
def convert_DoL_column(DoLdata, column_name, cfun):
    for i in range(len(DoLdata[column_name])):
        DoLdata[column_name][i] = cfun(DoLdata[column_name][i])
```

Two solutions Explicit for over rows in column

```
#-------------------------------------------------------------------------
def convert_DoL_column(DoLdata, column_name, cfun):
    for index in range(len(DoLdata[column_name])):
        DoLdata[column_name][index] = cfun(DoLdata[column_name][index])
    return
#-------------------------------------------------------------------------

# OR

#-------------------------------------------------------------------------
def convert_DoL_column(DoLdata, column_name, cfun):
    DoLdata[column_name] = [ cfun(val) for val in DoLdata[column_name] ]
    return
#-------------------------------------------------------------------------
```

list comprehension to get converted column.

Marcy