CIS 210 Fall 2020
Week 5 Lab

- Lab Prep Exercises / Demo – Intro to Python string formatting and formatted strings (f-strings)
- Section Exercises
- More on string formatting
Preparing for Lab 5: Python string formatting

Python 3.6 and higher has formatted strings (f-strings)!
https://docs.python.org/3/whatsnew/3.6.html#whatsnew36-pep498
https://realpython.com/python-f-strings/

Be aware that formatted strings improve on prior, but still used enough that you will almost certainly run into them, ways to format strings in Python.

Before there were formatted strings, there was the Python string format method (text p. 152-156):
https://docs.python.org/3.4/library/string.html

Another way to format strings is deprecated in Python, but it is similar to the string formatting syntax used in the C programming language (and in text 2d. ed.):
https://realpython.com/python-f-strings/

This is a short introduction to Python formatted strings (f-strings), with some examples.

Formatted strings are strings, where the formatted string reflects specified formatting that has been applied to the original, or “format”, string.

Without any special syntax inside the quotes, formatted strings look like strings, with a leading f to indicate that this is a formatted string:

```python
>>> f'hello, world'
'hello, world'
```

Formatted strings, or f-strings, use {} inside a string to indicate where values should be inserted:

```python
>>> x = 'world'
>>> print(f'Hello, {x}.')
Hello, world.
```

Notice that we can do this using literals instead of variables:

```python
>>> f'hello, {'w' + 'o' + 'r' + 'l' + 'd'}}'
'hello, world'
```

```python
>>> f'hello, {99/9}'
'hello, 11.0'
```

Note that the values in the {} are PART of the string into which we are inserting. Note that the values are simply Python expressions, and can be string literals, integers, variables, etc.

```python
>>> print(f'$\{est_tax(20_000, 1)}$')  # number coercion/str concat not needed!
$1870.0$
```
Use \{[commands]\} to provide additional information for formatted strings:
- \{fillchar\}[align][width][,][decimal places][type]
- Fillchar – what char to put in open places – used with align
- Align (right >, left < or center ^)
- Width – how many total spots to use
- Comma – use comma for thousands separator
- Decimal places – how many digits after the decimal (better than the round function)
- Presentation types – d for integers, f for floats, b for binary, c for characters, % for percentages
- More fields are available if you look
- All fields are optional if you don’t need them

For example:

```python
>>> print(f'a = {1.2:.3f}, b = {2:0>6}, c = {0.3:.1%}')
a = 1.200, b = 000002, c = 30.0%
```

Another example:

```python
>>> print(f'a = {1870.0:.2f}, b = {2:^10}, c = {3:>10}, d= {0.359:.1%}')
a = 1870.00, b =     2     , c =          3, d= 35.9%
```

Print with commas example:

```python
>>> print(f'{123456789:,}')
123,456,789
```

Format/print in binary:
```python
>>> f'{44:b}'
'101100'
>>> print(f'{44:b}')
101100
```

Now a character:
```python
>>> print(f'{101:c}')
e
```
Section Exercises – Python formatted strings – complete intro to f-strings (above) before trying these:

(a) Write a Python function to generate letterhead for a given name. The function should take in a single string and print it out surrounded by your letterhead. A simple example is here, but be creative.

*************************
* CIS 210 *
*************************

(b) Write a Python function that returns a number that has been “stringified” into money format – meaning it has a dollar sign, uses thousands comma separator, and is rounded to 2 digits. moneyformat should take in a number and return a string.

For example:
   >>> moneyformat(5)
   $5.00
   >>> moneyformat(1420.8823423)
   $1,420.88

(3) Given two lists of strings: one containing class titles and the second containing the corresponding class location, write a Python function that prints a 2-column table with neatly aligned columns. How wide should the columns be? For example,

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIS 210</td>
<td>WIL 100</td>
</tr>
<tr>
<td>CIS 211</td>
<td>LLCS 101</td>
</tr>
</tbody>
</table>

(4) Write a function that prints a temperature conversion table. The table should include temperatures from 32 to 212 degrees Fahrenheit and their Celsius equivalents, presented in two columns with appropriate headings. Each column should be 10 characters wide, and each temperature should have 3 digits to the right of the decimal point. Recall that the Fahrenheit to Celsius conversion equation is 
\[(F - 32) \times \frac{5}{9} = C.\] (Will the parentheses be necessary in the Python code?)
Just FYI - Examples using string format method (prior to Python 3.6):

Python docs for string formatting:
https://docs.python.org/3/library/string.html#formatstrings
https://docs.python.org/3/library/string.html#formatspec

String format uses {} inside a string to indicate where values should be inserted:

```python
>>> print('Hello, { }.format('world'))
Hello, world.
```

Notice that we can do this using variables instead of string literals:

```python
>>> name = 'world'
>>> print('Hello, { }.format(name))
Hello, world.
```

Arguments to the format method supply the values that should be inserted in the {} placeholders.
Note that these placeholders are PART of the string into which we are inserting. Note that the arguments are simply Python expressions, and can be string literals, integers, variables, etc.

```python
>>> print('Hello, { }.format(w + o + r + l + d))
Hello, world.
```

Another example:

```python
>>> my_string = "a = {:.3f}, b = {:0>6}, c = {:.1%}"
>>> my_string = my_string.format(1.2, 2, 0.3)
>>> print(my_string)
a = 1.200, b = 000002, c = 30.0%
```

Print with commas example:

```python
>>> print(123456789)
123,456,789
```

Format/print in binary:

```python
>>> print(44)
101100
```

Old style formatting (see text ch. 5.2.3) – deprecated in Python but still in use elsewhere

```python
>>> print('Hello, %s.' % ('world'))
Hello, world.
```