ECE 498 - Python Homework 9 Fall 2020

Goals: More f-strings, file paths, exception handling, the `dbm` and `pickle` modules, more programming practice.

For this homework, submit your work in a "`.py`" file. Note that some points may be deducted if your code could obviously be written in a "more Pythonic" way.

1) Write a function with prototype "`def fstring1(a,b,c,d,e):`" whose body simply returns a single f-string such that `print(fstring1(13,12,11,10,9))` followed by `print(fstring1(666,666,666,666,666))` generates exactly

```
val1=  13, val2=0012, val3=   b, val4=0x000A, val5=9. Done!
val1= 666, val2=0666, val3= 29a, val4=0x029A, val5=666. Done!
```

Pay close attention to the number of spaces, leading zeros, capitalization when in hexidecimal, etc

2) Write a function with prototype "`def fstring2(a,b,c,d,e):`" whose body simply returns a single f-string such that `print(fstring2(13,12,11,10,9))` followed by `print(fstring2(666,666,666,666,666))` generates exactly

```
val1=  13.0, val2=0012.0, val3=  1.1e+01, val4=001.0e+01 val5=9.0E+00. Done!
val1= 666.0, val2=0666.0, val3=  6.7e+02, val4=006.7e+02 val5=6.7E+02. Done!
```

Pay close attention to the number of spaces, leading zeros, capitalization (of "E"), etc.

3) Write a function with prototype "`def fstring3(a,b,c,d):`" whose body simply returns a single f-string such that `print(fstring3("Ho","Ho", "Ho", "way to go!"))` followed by `print(fstring3("Eeny", "meeny", "miny", "moe"))` generates exactly

```
val1=Ho      , val2=   Ho   , val3=      Ho, val4=way to go!. Done!
val1=Eeny    , val2= meeny  , val3=    miny, val4=moe. Done!
```

Pay close attention to the field widths and justification

4) Write test code for the above functions that tests using the given strings.

5) Modify the "`walk()`" function given in the text (and class notes) so that rather than printing the file names/path it becomes a generator function that yields a tuple containing both the filename/path and its size (as given by `os.path.getsize()`)

6) Write a function with prototype "`def listfiles(path):`" that uses the above "`walk()`" generator to print a list of files and their sizes, **sorted by filenames**. The function body is limited to two lines and the print must use a single f-string such that the results have the filename left-justified in a 25-character field and the file size is right justified in a 10-character field. E.g.,

```
./beale.py                     6914
./helloworld.py                  42
```

7) Write a function with prototype "`def inputint():`" that will simply prompt the user to enter an `integer` and will then return it. Use exception handling in a way that if the user enters something other than an `int`, it will tell them and the process will be repeated until they enter an `int`. Example:

```
Type in an int: No
Value must be an integer
Type in an int: ??????
Value must be an integer
Type in an int: 42          # Now returns 42
```

8) Add exception-handling code to the following function so that it prints "`infinity`" when division is by zero and it exits gracefully when the iterator is used up. Do not modify **any** of the given code, only add exception handling (and tabbing if necessary).

```
def needsexception():
      x = iter([5,2,0,7])
      while True:
            print(1 / next(x))
```

FYI, the output should be

```
0.2
0.5
infinity
0.14285714285714285
```

9) Write a function with prototype "`def profound():`" that will prompt the user to type something profound. It will then record the date and time using the "`datetime`" module (see what you did for homework 2) and then append the date, time and profound line to a file called "`profound.txt`". Do only one line per function call. Use a single write and f-string such that the file contents look like:

```
2020-10-27 11:20:22 -- Has eighteen letters does
2020-10-27 11:20:36 -- something profound
```

10) Write a function with prototype "`def bestwords():`" that will prompt the user to enter a line with some of the best words (separated by spaces). These will be converted to lower case and stored in a data base file (using the `dbm` and `pickle` modules) whose keys are a tuple of the current year, month and day (using the `datetime` module) and the "values" are a **`set`** of words collected on that day. Each new word should be checked against all previously-entered words (even those on other days) and it is not added if it already exists in the database. (Hint: I simply subtracted all previous sets of words from the current set, then added that new set to the one for the current day (if it already existed).

11) Write a function with prototype "`def printbestwords():`" that will simply open the database of best words and print the keys and values, **sorted** by the keys. Don't forget to close the database on exit. Example output:

```
(2020, 10, 26) {'infantroopen', 'susbesdig'}
(2020, 10, 27) {'deligitimatize'}
(2020, 10, 28) {'transpants', 'resaption'}
```