ECE 498 - Python Homework 8 Fall 2020

Goals: More generator functions, zip function, f-strings, introduction to lambda functions 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.

0) Import your "`juliandate`" code.

1) Write a generator function with prototype "`def gendates(month, day, year):`" that will generate successive dates beginning with the given month, day, year. It operates by getting the Julian day number for the given date (by calling your imported getJDN routine), and in an infinite loop, calling JDN2date to convert it back, while adding one to the Julian day number each time through the loop.

2) Write a generator function with prototype "`def gendow(month, day, year):`" that will generate successive days of the week (as strings) beginning with that of the given month, day, year. For example, given month, day, year equal to 10, 21, 2020 it will return "Wednesday", "Thursday", "Friday", etc. It operates by calling your `dow()` function only once and then cycling through days of the week from there.

3) Write a function with prototype "`def gendatedows(month, day, year):`" that will return a generator function that combines `gendates` and `gendow` returning tuples of `month,day,year,dow`. This is a one line function body containing "`yield from`" and a "`zip`" of the two functions.

4) Write simple test code that uses "`gendatedows()`" to create a list of 10 tuples of month,day,year,dow beginning with the date (12,26,2020). (A one-liner if you do it right…) Then print the result. It should start: `[(12,26,2020, 'Saturday'), (12,27,2020, 'Sunday'), ...`

5) Rewrite the `easter()` function (see code accompanying this lab) to be a "generator function" with prototype "`def geneaster(year):`" You pass in the year on creation and it will generate Easter dates beginning with that year. Modify what is returned to also include the year, using the order "year, month, day."

6) Write test code that will use `geneaster()` to create an Easter generator starting at the year 1900, and will then use the generator to find and print 10 Easter dates beginning with that year. (Do not create new generators for each year.) After that use "`next()`" to generate and print the date for one more year.

7) Write a generator function with prototype "`def limitgeneaster(year, limit):`" that will generate Easter dates beginning with the given year, but will stop after generating "`limit`" dates. This should be a one-line function body that uses `geneaster()`. You can model your code after the given "`limiter()`" function (accompanying this assignment), but do not use that function.

8) Repeat problem 6) but this time use `limitgeneaster()` as your generator, creating a generator that only creates at most 10 dates. Use "`next()`" after printing the 10 dates and see the result, then comment it out, if it is a problem.

9) Write a function with prototype "`def datedict(year, limit):`" that will use a generator created with `limitgeneaster()` to create and return a dictionary who keys are tuples of `(month, day)` and whose values are a list of years having Easter on that month and day. E.g, with "`year`" and "`limit`" of 1900 and 200, respectively, one key-value pair in the dictionary is (3, 25) [1951, 2035, 2046].

10) Write brief test code that creates a date dictionary using `datedict()` and the prints the results sorted my month-day. Each line should use a single print and f-string such that the printing starts out like the following:

```
Years with Easter on: 3/23: [1913, 2008]
Years with Easter on: 3/24: [1940]
Years with Easter on: 3/25: [1951, 2035, 2046]
Years with Easter on: 3/26: [1967, 1978, 1989, 2062, 2073, 2084]
```

11) Redo the previous test code, sorting by the number of years in the list of years for a given month-day. Your only change should be the addition of a sorting key consisting of a "`lambda`" function. Your output might start as follows:

```
Years with Easter on: 3/24: [1940]
Years with Easter on: 3/23: [1913, 2008]
Years with Easter on: 4/25: [1943, 2038]
Years with Easter on: 4/24: [2011, 2095]
```

For maximum credit on this problem make one very small change that causes all month-day entries having the same number of years to appear in order. i.e., the above would actually start with

```
Years with Easter on: 3/24: [1940]
Years with Easter on: 3/23: [1913, 2008]
Years with Easter on: 4/24: [2011, 2095]
Years with Easter on: 4/25: [1943, 2038]
```

Notes on next page

Notes:

Functions with this lab:

Easter function:

```
def easter(year):
  _,A = divmod(year, 19)
  B,C = divmod(year, 100)
  D,E = divmod(B, 4)
  F,_ = divmod(B+8, 25)
  G,_ = divmod(B-F+1, 3)
  _,H = divmod(19*A + B - D - G + 15, 30)
  I,K = divmod(C, 4)
  _,L = divmod(32 + 2*E + 2*I - H - K, 7)
  M,_ = divmod(A + 11*H + 22*L, 451)
  N,P = divmod(H + L - 7*M + 114, 31)

  return N, P+1
```

limiter function:

```
def limiter(genr, limit):
  return (v for _,v in zip(range(limit), genr))
```