Computer Science CS134 (Spring 2020)

Shikha Singh & Iris Howley

Laboratory 2

Computing the Age of the Moon

Objective. To construct a script that computes something non-trivial.

This week we'd like you to implement an algorithm to compute the phase of the moon as a python script. Like our day-of-the-week algorithm, this algorithm is simple enough so you could compute the age in your head, but complex enough so that you'd be motivated to have it available as a python function.

Method. We'll be implementing a method for determining the moon's age suitable for mental calculation. Like the day-of-week calculation, this method is due to John Conway.

Let's assume that you have a date, specified as a valid month, day, and year. For purposes of practicality, we'll limit the year to those in the range 1900 to 2099. Let's assume the month is specified as a value between 1 and 12 and the day is a value between 1 and 31. This algorithm computes the moon's *age* in days, a value between 0 and 29; when the age is zero, the moon is new, when it's 15, it's full.

The algorithm involves keeping a running sum, `rsum`, modulo 30.

1. Initialize the sum, `rsum`, to the sum of the day, the month, and 30. The final 30 is helpful to avoid negative modular arithmetic, later.

2. If the two leading digits of the year, the "century digits," are 20, subtract 8 from `rsum`, otherwise subtract 4 from `rsum`. Be sure to treat the digits as numbers, without converting to/from strings!

3. Compute the value of the year within the century, `yy`, which would be a value between 0 and 99. Then, compute the distance of `yy` to the closest multiple of 19—lets call this value `dist`. For example, 2018 yields -1, but 1999 gives +4. (Notice that 2000 gives 0, but the century correction avoids a discontinuity here.) This calculation is challenging to think about. Be careful to check your work!

4. To `dist` "prepend" a ten's digit that is the value of `|dist|` mod 3.[1] Thus 2014, whose distance is -5, will have a final `dist` of -25, 1999 generates 14, and 2000 gives 0. Notice that this part of the computation never changes during the year. The final value of `dist` for 2018 is -11.

5. Add `dist` to `rsum` and compute the remainder, when divided by 30. That's the age of the moon!

Here are some examples you might use to verify your calculations:

- Moon Unit Zappa was born September 28, 1967. Think: $(9 + 28 + 30 - 4 - 9) \bmod 30 \equiv 24$
- Tally, the moon-colored dog, was born on August 28, 2017.
  She calculates: $(8 + 28 + 30 - 8 - 22) \bmod 30 \equiv 6$
- Pixel, the moon-howling pup, was born on May 16, 2018: $(5 + 16 + 30 - 8 - 11) \bmod 30 \equiv 2$

[1] `|x|` stands for the absolute value of x, which is defined as x if $x \geq 0$ and $-x$ if $x < 0$. It is handy that Python has an in-built `abs()` function that computes the absolute value of a number.

This week's tasks. Clone the starter repository for this week's lab into your `cs134` directory as you did last week, by going to `https://evolene.cs.williams.edu` and clicking `Clone` and the the `Copy URL to clipboard` button under the `Clone with HTTPS` text. This copies the URL of the git repository for this lab. Return to the `Terminal` application and type `git clone` followed by the URL of the lab project you just copied (you can paste on Mac by pressing `Command-V`) followed by the name of the lab `lab02`. Your line should look something like this:

```
git clone https://evolene.cs.williams.edu/cs134-s20/lab02/22xyz3.git ~/cs134/lab02/
```

where your CS username replaces 22xyz3.
Your job is to write, in a file called `phase.py`, a function, `moonAge(month,day,year)`, that computes the age of the moon associated with the date specified by `month`, `day`, and `year`.

You can exercise your `moonAge` function, by writing a `main` function that takes in user input, e.g.:

```
def main():
     month = int(input("Month? "))
     day = int(input("Day? "))
     year = int(input("Year (yyyy)? "))
     age = moonAge(month,day,year)
     print("On ",month,"/",day,"/",year," the moon's age is ",age,".", sep = '')
```

When your script is complete, you should be able to type:

```
-> python3 phase.py
Month? 9
Day? 17
Year (yyyy)? 2018
On 9/17/2018 the moon's age is 7.
```

You should also be able to test your code *interactively* with

```
-> python3
>>> from phase import moonAge
>>> moonAge(9,17,2018)
7
>>>
```

Be sure to understand the difference between excercising code in a script and testing functions interactively.

Submitting your work. When you're finished with `phase.py`, stage, commit, and push your work to the server as you did in Lab 1. Remember that you must certify that your work is your own, by typing out the Honor Code statement in the `honorcode.txt` file, committing and pushing it along with your work.

Late days. You are allowed a total of 3 late days over the semester, with at most 2 late days towards any one lab. You must request a late day in advance on the form, here: `http://bit.ly/s20late`.

Working across different machines. If you clone the lab repository on two different machines and are planning to switch between them, you must always start by "pulling" your most recent work from the server to the local machine. You can do this by adding the correct project folder to Atom, toggling the `Git` tab, and clicking on the `Fetch` button at the bottom right of the `Git` tab.

Grading Guidelines. Both functionality and programming style are important when writing code, just as both the content and the writing style are important when writing an essay. In this program, some of the specific functional requirements we will test for include a correct implementation of:

- Steps 1-5 (without the use of `strings`)
- `main` function that asks user input (month, day, year), invokes `moonAge` function and prints the output
- `moonAge(month, day, year)` with 3 parameters that returns the age of the moon as an `int`.

Stylistically, we expect to see programs that exhibit the following: meaningful names used in declarations, informative comments, good and consistent formatting, and good choice of Python commands. There is some subjectivity to what makes good style, but the basic goal is to make your ideas as clear and easy to follow as possible.

Grading scale. Programming labs will be graded on the following scale:

A+: An absolutely fantastic submission of the sort that will only come along a few times during the semester.
A: A submission that exceeds our standard expectations for the assignment. The program must reflect additional work beyond the requirements or get the job done in a particularly elegant way.
A-: A submission that satisfies all the requirements for the assignment – a job well done.
B+: Submission meets the requirements for the assignment, possibly with a few small problems.
B: A submission that has problems serious enough to fall short of the requirements for the assignment.
C: A submission that has extremely serious problems, but nonetheless shows some effort and understanding.
D: A submission that shows little effort and does not represent passing work.

Extra credit—One way to dig deeper. Consider reporting the western description of the moon's age:

age	description
0,1,29	new
2,3,4,5,6	waxing crescent
7,8	first quarter
9,10,11,12,13	waxing gibbous
14,15,16	full
17,18,19,20,21	waning gibbous
22,23,	third quarter
24,25,26,27,28	waning crescent

Your program should report the moon's age together with its description, for example:

```
-> python3 phase.py
Month? 9
Day? 18
Year (yyyy)? 2018
On 9/18/2018 the moon's age is 8, a first quarter moon.
```

It is possible, though not necessary for this problem, to determine the moon's description by using a small number of strings and without using any `if` statements! You may also find that using `lists` are helpful, though we haven't yet covered them in class.

⋆