Computer Science CS134 (Fall 2020)

Daniel Aalberts, Duane Bailey, & Molly Feldman

Laboratory 2

Computing the Age of the Moon

Objective. To develop a non-trivial script.

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. This method is due to the late Princeton mathematician, John Conway.

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

Let's see how Conway's algorithm works. The algorithm involves keeping a running sum, `rsum`. (We'll consider the date of the Apollo 11 moon landing, July 20, 1969, as a running parenthetical example.)

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. (For the moon landing, `rsum = 7+20+30`, or 57.)

2. If the two leading digits of the year, the "century digits," are 20, subtract 8 from `rsum`, otherwise subtract 4 from `rsum`. (The example's year is 1969, so `rsum` becomes 57−4, or 53.)

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. Take your time, and carefully check your work! (For our example, `yy` is 69. 69 is more than 57 by 12, so 69 must be 7 below the next multiple. Since −7 has a smaller magnitude than 12, `dist` is −7.)

4. Notice that the magnitude of `dist`, `|dist|` is always one digit [1]. Next we insert a ten's digit to `dist` that is the value of `|dist|` mod 3. 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 2020 will always be 11. (For the 1969 moon landing, dist becomes −17 because 7 has a remainder of 1 when divided by 3.)

5. Add `dist` to `rsum` and compute the remainder, when divided by 30. (The moon landing has an `rsum` remainder of 6, because `53+(−17)` is 36.) That's the "age" of the moon!

◇

[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(i)` function that computes the absolute value of an `int`, `i`.

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

- The Apollo 11 Moon Landing occurred on July 20, 1969. Thus: $(7 + 20 + 30 - 4 - 17) \text{mod } 30 \equiv 6$
- Moon Unit Zappa was born September 28, 1967. Think: $(9 + 28 + 30 - 4 - 9) \text{mod } 30 \equiv 24$
- Big Bang physicist, Sheldon "Moon Pie" Cooper, turned 21 on February 26, 2001.
  He observes: $(2 + 26 + 30 - 8 + 11) \text{mod } 30 \equiv 1$
- Tally, Duane's moon-colored dog, was born on August 28, 2017.
  She calculates: $(8 + 28 + 30 - 8 - 22) \text{mod } 30 \equiv 6$

This week's tasks. Clone the starter repository for this week's lab into your `cs134` directory as you did last week. Open your terminal application and navigate to your `cs134` directory with:

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

Your CS username replaces 22xyz3. The `clone` command will follow this pattern each week.

1. 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`.
2. Use this function, `main`, to prompt for a date and report the result of the `moonAge` function:

```
def main():
    """A method that prompts for a date and prints the moon's age."""
    month = int(input("Month? "))
    day = int(input("Day? "))
    year = int(input("Year (yyyy)? "))
    age = moonAge(month,day,year)
    print("On {}/{}/{}, the moon's age is {}.".format(month, day, year, age))
```

When your script is complete, you should be able to type:[2]

```
$ python3 phase.py
Month? 9
Day? 18
Year (yyyy)? 2020
On 9/18/2020 the moon's age is 0.
```

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

```
$ python3
>>> from phase import moonAge
>>> moonAge(9,18,2020)
0
```

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

[2] The symbols like $ and >>> are prompts. Do not type the prompts when using our examples on your machine.

Submitting your work. When you're finished with `phase.py`, add, commit, and push your files to the server as you did in Lab 1. Remember that you must certify that your work is your own, by adding your name to the `honorcode.txt` file, submitting it along with your work.

You can verify you've committed all the important files with

```
git status
```

which should report

```
nothing to commit, working tree clean
```

You can verify you've *pushed* all your work by going to `evolene` in a browser, logging in, and checking on the files associated with your `lab02` repository.

Comments on Grading. 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 lab we will focus on the correct functionality of `moonAge`. Stylistically, we expect to see code that uses meaningful names, has informative comments, formatted nicely, using appropriate Python commands. There is some subjectivity to what makes good style, but the overarching goal is to make your logic-as-code as easy to follow as possible. The copy of `GradeSheet.txt` (in the repository) shows you more precisely what we're looking for.

Extra credit—One way to dig deeper. Report a textual description of the moon's age:

moon 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

For example:

```
$ python3 phase.py
Month? 9
Day? 18
Year (yyyy)? 2020
On 9/18/2020 the moon's age is 0, a new moon.
```

This can be accomplished with a new function, `age2Str`, that converts an integer to a string. This function would be called from `main`. We expect that it would make use of several `if` statements. We will give the most credit for demonstrating an approach that reduces the average number of `if` conditions `age2Str` would execute for any particular moon age.

⋆