Computer Science 134C
Introduction to Computer Science, in Python
Lecture #3 (Code reuse with scripts and functions)
February 12,2020

Keywords
argv, condition, function, maximum, method, reuse, parameters, script, dow

We organize computations to be reused.

1. How do we compute the day of the week (Sunday is 0, Monday is 1, etc.) given the time in seconds since the 1960s? (Hint: We need the `time` function from the `time` package. Also the 1960s ended on a Wednesday.)

2. Suppose we wanted to compute the day of the week for an arbitrary date, specified using a month, day, and year?

   We'll limit ourselves to dates between 1900 and 2099, inclusive.[1] The computed remainder tells us the day of the week, where 0 is Saturday.

   If you do this in your head, you need to remember a short table of monthly adjustments. Each entry in the table corresponds to a month, where January is month 1 and December is month 12.

Month	1	2	3	4	5	6	7	8	9	10	11	12
Adjustment	1	4	4	0	2	5	0	3	6	1	4	6

   Notice that 144 is 12^2, 025 is 5^2, 036 is 6^2, and 146 is a bit more than 12^2. If the year is divisible by 4 (it's a leap year) and the date is January or February, you must subtract 1 from the adjustment.

   Now, here's the algorithm:

   1. Write down the date numerically. The date consists of a month between 1 and 12, a day of the month between 1 and 31, and the number of years since 1900. Grace Hopper, computer language pioneer, was born December 9, 1906. That would be represented as year 6. Pixel, the Sentient Snowball, was born on May 16, 2018, year 118.
   2. Compute the sum of the following quantities:
      - the month adjustment from the given table (*e.g.*, 6 for Admiral Hopper)
      - the day of the month
      - the year
      - the whole number of times 4 divides the year (*e.g.*, 29 for Pixel)
   3. Compute the remainder of the sum of step 2, when divided by 7. The remainder gives the day of the week, where Saturday is 0, Sunday is 1, *etc.* Notice that we can compute the remainders *before* we compute the sum. You may also have to compute the remainder after the sum as well, but if you're doing this in your head, this considerably simplifies the arithmetic.

   How might you adjust the algorithm to make the result of 0 indicate Sunday?

⋆

[1] This particular technique is due to John Conway, of Princeton University. Professor Conway answers 10 day of the week problems before gaining access to his computer. His record is well under 15 seconds for 10 correct dates. See "Scientist at Work: John H. Conway; At Home in the Elusive World of Mathematics," *The New York Times*, October 12, 1993.

3. A script, dow.py, to compute the day of the week according to the algorithm on the front of this page:

```
# A script to compute the day of the week.
# (c) 2018 duane a. bailey
month = int(input("Month (1-12): "))
day = int(input("Day (1-31): "))
year = int(input("Year (1900-2099): "))

# this is a *list* containing 12 integers.
adjustments = [ 1,4,4, 0,2,5, 0,3,6, 1,4,6 ]

# the integers in the adjustment list are indexed 0 through 11
# madj is the adjustment based on the particular month
madj = adjustments[month-1]

# it's best to think of the year as a value between 0 and 200
year -= 1900

# this is the main calculation:
sum = madj + day + (year//4) + year

# this is a correction for early in leap years
if (year%4 == 0) and (month <= 2) :
    sum -= 1

# a *list of strings*, indexed between 0 and 6 (remainders, mod 7)
dayName = ["Saturday", "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]
print(dayName[sum%7])
```

Recall: to *execute* this script, we can type:

```
python3 dow.py
```

We can now *reuse* the code in the script, without re-typing the commands.