UPA 2 - Age in Seconds Program

We look at the problem of calculating an individual's age in seconds. It is not feasible to determine a given person's age to the exact second. This would require knowing, to the second, when they were born. It would also involve knowing the time zone they were born in, issues of daylight savings time, con- sideration of leap years, and so forth. Therefore, the problem is to determine an *approximation* of age in seconds. The program will be tested against calculations of age from online resources.

The Problem

The problem is to determine the approximate age of an individual in seconds within 99% accuracy of results from online resources. The program must work for dates of birth from January 1, 1900 to the present.

Problem Analysis

The fundamental computational issue for this problem is the development of an algorithm incorporating approximations for information that is impractical to utilize (time of birth to the second, daylight savings time, etc.), while producing a result that meets the required degree of accuracy.

Program Design

Meeting the Program Requirements

There is no requirement for the form in which the date of birth is to be entered. We will therefore design the program to input the date of birth as integer values. Also, the program will not perform input error checking, since we have not yet covered the programming concepts for this.

Data Description

The program needs to represent two dates, the user's date of birth, and the current date. Since each part of the date must be able to be operated on arithmetically, dates will be represented by three integers. For example, May 15, 1992 would be represented as follows:

```
year = 1992     month = 5     day = 15
```

Hard-coded values will also be utilized for the number of seconds in a minute, number of minutes in an hour, number of hours in a day, and the number of days in a year.

Algorithmic Approach

The Python Standard Library module `datetime` will be used to obtain the current date. (See the Python 3 Programmers' Reference.) We consider how the calculations can be approximated without greatly affecting the accuracy of the results.

We start with the issue of leap years. Since there is a leap year once every four years (with some exceptions), we calculate the average number of seconds in a year over a four-year period that includes a leap year. Since non-leap years have 365 days, and leap years have 366, we need to compute,

`numsecs_day` = (hours per day) * (mins per hour) * (secs per minute)
`numsecs_year` = (days per year) * `numsecs_day`
`avg_numsecs_year` = `(4 * numsecs_year) + numsecs_day) // 4` (one extra day for leap year)
`avg_numsecs_month` = `avgnumsecs_year // 12`

To calculate someone's age in seconds, we use January 1, 1900 as a basis. Thus, we compute two values—the number of seconds from January 1 1900 to the given date of birth, and the number of seconds from January 1 1900 to the current date. Subtracting the former from the latter gives the approximate age,

Note that if we directly determined the number of seconds between the date of birth and current date, the months and days of each would need to be compared to see how many full months and years there were between the two. Using 1900 as a basis avoids these comparisons. Thus, the rest of our algorithm is given below.

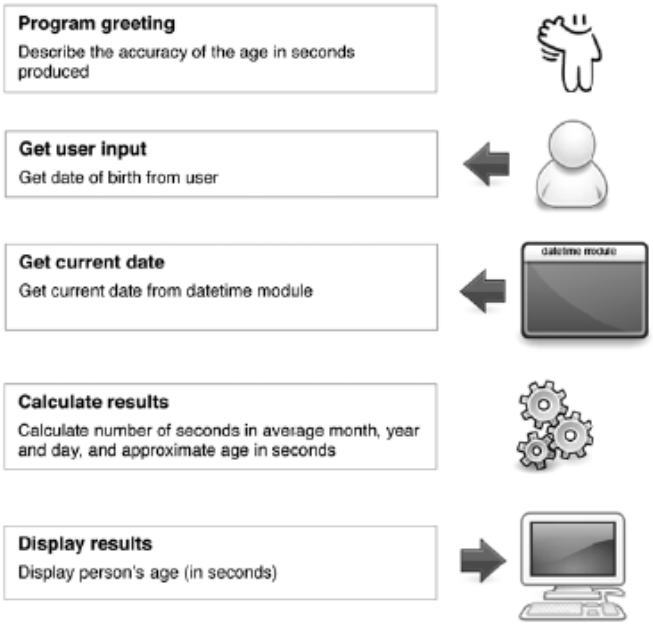
```
numsecs_1900_to_dob = (year_birth - 1900) * avg_numsecs_year +
                      (month_birth -1) * avg_numsecs_month +
                      (day_birth * numsecs_day)

numsecs_1900_to_today = (current_year 2 1900) * avg_numsecs_year +
                        (current_month -1) * avg_numsecs_month +
                        (current_day * numsecs_day)

age_in_secs = num_secs_1900_to_today - numsecs_1900_to_dob
```

Program Stages

The overall steps in this program design are show below:



Program Implementation and Testing

Stage 1—Getting the Date of Birth and Current Date

First, we decide on the variables needed for the program. For date of birth, we use variables `month_ birth`, `day_birth`, and `year_birth`. Similarly, for the current date we use variables `current_ month`, `current_day`, and `current_year`. The first stage of the program assigns each of these values, shown below:

```
# Age in Seconds Program (Stage 1)
# This program will calculate a person's approximate age in seconds

import datetime

# Get month, day, year of birth
month_birth = int(input('Enter month born (1-12): '))
day_birth = int(input ('Enter day born (1-31): '))
year_birth = int(input('Enter year born (4-digit): '))

# Get current month, day, year
current_month = datetime.date.today().month
current_day = datetime.date.today().day
current_year = datetime.cate.today().year

# Test output
print('\nThe date of birth read is: ', month_birth, day_birth,
        year_birth)

print('The current date read is: ', current_month, current_day,
       current_year)
```

Stage 1 Testing

We add test statements that display the values of the assigned variables. This is to ensure that the dates we are starting with are correct; otherwise, the results will certainly not be correct. The test run below indicates that the input is being correctly read.

```
Enter month born (1-12): 4
Enter day born (1-31): 12
Enter year born (4-digit): 1981
The date of birth read is: 4 12 1981
The current date read is: 1 5 2010
```

Stage 2—Approximating the Number of Seconds in a Year/Month/Day

Next we determine the approximate number of seconds in a given year and month, and the exact number of seconds in a day stored in variables `avg_numsecs_year`, `avg_numsecs_month`, and `numsecs_day`, respectively, shown below:

```
1  # Age in Seconds Program (Stage 2)
2  # This program will calculate a person's approximate age in seconds
3
4  import datetime
5
6  ### Get month, day, year of birth
7  ##month_birth = int(input('Enter month born (1-12): '))
8  ##day_birth = int(input ('Enter day born (1-31): '))
9  ##year_birth = int(input('Enter year born (4-digit): '))
10
11 ### Get current month, day, year
12 ##current_month = datetime.date.today().month
13 ##current_day = datetime.date.today().day
14 ##current_year = datetime.date.today().year
15
16 # Determine number of seconds in a day, average month, and average year
17 numsecs_day = 24 * 60 * 60
18 numsecs_year = 365 * numsecs_day
19
20 avg_numsecs_year = ((4 * numsecs_year) + numsecs_day) // 4
21 avg_numsecs_month = avg_numsecs_year // 12
22
23 # Test output
24 print('numsecs_day ', numsecs_day)
25 print('avg_numsecs_month = ', avg_numsecs_month)
26 print('avg_numsecs_year = ', avg_numsecs_year)
```

The lines of code prompting for input are commented out (**lines 6–9** and **11–14**). Since it is easy to comment out (and uncomment) blocks of code in IDLE, we do so; the input values are irrelevant to this part of the program testing.

Stage 2 Testing

Following is the output of this test run. Checking online sources, we find that the number of seconds in a regular year is 31,536,000 and in a leap year is 31,622,400. Thus, our approximation of 31,557,600 as the average number of seconds over four years (including a leap year) is reasonable. The `avg_num_seconds_month` is directly calculated from variable `avg_numsecs_ year`, and `numsecs_day` is found to be correct.

```
numsecs_day 86400
avg_numsecs_month = 2629800
avg_numsecs_year = 31557600
```

Final Stage—Calculating the Number of Seconds from 1900

Finally, we complete the program by calculating the approximate number of seconds from 1900 to both the current date and the provided date of birth. The difference of these two values gives the approximate age in seconds. The complete program is shown below:

```
# Age in Seconds Program
# This program will calculate a person's approximate age in seconds

import datetime

# Program greeting
print('This program computes the approximate age in seconds of an')
print('individual based on a provided date of birth. Only dates of')
print('birth from 1900 and after can be computed\n')

# Get month, day, year of birth
month_birth = int(input('Enter month born (1-12): '))
day_birth = int(input ('Enter day born (1-31): '))
year_birth = int(input('Enter year born (4-digit): '))

# Get month, day, year of birth
current_month = datetime.date.today().month
current_day = datetime.date.today().day
current_year = datetime.date.today().year

# Determine number of seconds in a day, average month, and average year
numsecs_day = 24 * 60 * 60
numsecs_year = 365 * numsecs_day

avg_numsecs_year = ((4 * numsecs_year) + numsecs_day) // 4
avg_numsecs_month = avg_numsecs_year // 12

# Calculate approximate age in seconds
numsecs_1900_dob = (year_birth - 1900 * avg_numsecs_year) + \
                   (month_birth - 1 * avg_numsecs_month) + \
                   (day_birth * numsecs_day)

numsecs_1900_today = (current_year - 1900 * avg_numsecs_year) + \
                     (current_month - 1 * avg_numsecs_month) + \
                     (current_day * numsecs_day)

age_in_secs = numsecs_1900_today - numsecs_1900_dob

# output results
print('\nYou are approximately', age_in_secs, 'seconds old')
```

We develop a set of test cases for this program. We follow the testing strategy of including "average" as well as "extreme" or "special case" test cases in the test plan. The test results are shown below:

Birth Date	Expected Results	Actual Results	Inaccuracy	Evaluation
Jan 1, 1900	3,520,023,010±86,400	1,468,917	99.96%	failed
April 12, 1981	955,156,351±86,400	518,433	99.94%	failed
Jan. 4, 2000	364,090,570±86,400	1,209,617	99.64	failed
Dec. 31, 2009	48,821,332±86,400	-1,123,203	102.12	failed
Yesterday	86,400±86,400	86,400	0 %	passed

The "correct" age in seconds for each was obtained from an online source. January 1, 1900 was included in the test plan since it is the earliest date ("extreme case") that the program is required to work for. April 12, 1981 was included as an average case in the 1900s, and January 4, 2000 as an average case in the 2000s. December 31, 2009 was included since it is the last day of the last month of the year. Finally, a test case for a birthday on the day before the current date was included as a special case. (See sample program execution in Figure 2-28). Since these values are continuously changing by the second, we consider any result within one day's worth of seconds (± 84,000) to be an exact result.

```
This program computes the approximate age in seconds of an
individual based on a provided date of birth. Only ages for
dates of birth from 1900 and after can be computed

Enter month born (1-12): 4
Enter day born (1-31): 12
Enter year born: (4-digit)1981

You are approximately 518433 seconds old
>>>
```

The program results are obviously incorrect, since the result is approximately equal to the average number of seconds in a month (determined above). The only correct result is for the day before the current date. The inaccuracy of each result was calculated as follows for April 12, 1981,

((abs(expected_results – actual_results) – 86,400) / expected_results) * 100 = ((917,110,352 – 518,433) - 86400) / 917,110,352) * 100 = 99.93 %

Either our algorithmic approach is flawed, or it is not correctly implemented. Since we didn't find any errors in the development of the first and second stages of the program, the problem must

be in the calculation of the approximate age in **lines 29–37**. These lines define three variables: `numsecs_1900_dob`, `numsecs_1900_today`, and `age_in_secs`. We can inspect the values of these variables after execution of the program to see if anything irregular pops out at us.

```
This program computes the approximate age in seconds
of an individual based on a provided date of birth.
Only ages for dates of birth from 1900 and after can
be computed
Enter month born (1-12): 4
Enter day born (1-31): 12
Enter year born: (4-
digit)1981
You are approximately 604833 seconds old
>>>
>>> numsecs_1900_dob
259961031015
>>> numsecs_1900_today
259960426182
>>>
```

Clearly, this is where the problem is, since we are getting negative values for the times between 1900 and date of birth, and from 1900 to today. We "work backwards" and consider how the expressions could give negative results. This would be explained if, for some reason, the second operand of the subtraction were greater than the first. That would happen if the expression were evaluated, for example, as

```
numsecs_1900_dob = (year_birth - (1900 * avg_numsecs_year)) + \
                   (month_birth - (1 * avg_numsecs_month)) + \
                   (day_birth * numsecs_day)
```

rather than the following intended means of evaluation,

```
numsecs_1900_dob 5 ((year_birth - 1900) * avg_numsecs_year) + \
                   ((month_birth 2 1) * avg_numsecs_month) + \
                   (day_birth * numsecs_day)
```

Now we realize! Because we did not use parentheses to explicitly indicate the proper order of operators, by the rules of operator precedence Python evaluated the expression as the first way above, not the second as it should be. This would also explain why the program gave the correct result for a date of birth one day before the current date. Once we make the corrections and rerun the test plan, we get the following results shown below:

Birth Date	Expected Results	Actual Results	Inaccuracy

Jan 1, 1900	3,520,023,010±86,400	3,520,227,600	< .004 %
April 12, 1981	955,156,351±86,400	955,222,200	0 %
Jan. 4, 2000	364,090,570±86,400	364,208,400	< .009 %
Dec. 31, 2009	48,821,332±86,400	48,929,400	< .05%
Yesterday	86,400±86,400	86,400	0 %

These results demonstrate that our approximation of the number of seconds in a year was sufficient to get very good results, well within the 99% degree of accuracy required for this program. We would expect more recent dates of birth to give less accurate results given that there is less time that is approximated. Still, for test case December 31, 2009 the inaccuracy is less than .05 percent. Therefore, we were able to develop a program that gave very accurate results without involving all the program logic that would be needed to consider all the details required to give an exact result.