5. Which of the following would involve coercion when evaluated in Python?
   **(a)** `4.0 + 3` **(b)** `3.2 * 4.0`

6. Which of the following expressions use explicit type conversion?
   **(a)** `4.0 + float(3)` **(b)** `3.2 * 4.0` **(c)** `3.2 + int(4.0)`

ANSWERS: 1. (b), 2. (a) 23 (b) 25 (c) 50 (d) 18, 3. (a) 3 (b) 256, 4. (a), 5. (a), 6. (a, c)

# COMPUTATIONAL PROBLEM SOLVING

## 2.5 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, consideration 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.

Juffin/iStockphoto

### 2.5.1 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.

### 2.5.2 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.

### 2.5.3 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

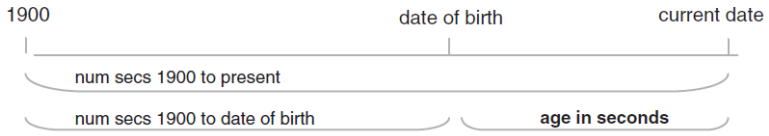
The Python Standard Library module `datetime` will be used to obtain the current date. (See the Python 3 Programmers' Reference.) We consider how the calculations can be approximated without greatly affecting the accuracy of the results.

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
avg_numsecs_month = avgnumsecs_year // 12
```

(one extra day for leap year)

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 - 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
```

### Overall Program Steps

The overall steps in this program design are in Figure 2-23.

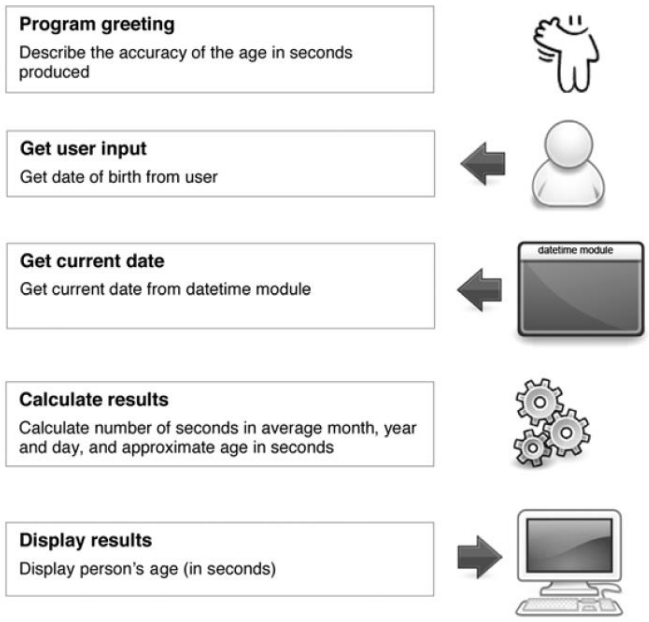



**FIGURE 2-23** Overall Steps of the Age in Seconds Program

### 2.5.4 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 in Figure 2-24.

```
# 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.date.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)
```

**FIGURE 2-24** First Stage of Age in Seconds Program

### 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 in Figure 2-25.

```
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)
```

**FIGURE 2-25** Second Stage of Age in Seconds Program

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 in Figure 2-26.

```
# 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')
```

**FIGURE 2-26** Final Stage of Age in Seconds Program

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 given in Figure 2-27.

| Date of Birth | Expected Results | Actual Results | Inaccuracy | Evaluation |
|---|---|---|---|---|
| January 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 |
| January 4, 2000 | 364,090,570 ± 86,400 | 1,209,617 | 99.64 % | failed |
| December 31, 2009 | 48,821,332 ± 86,400 | -1,123,203 | 102.12 % | failed |
| (day before current date) | 86,400 ± 86,400 | 86,400 | 0 % | passed |

**FIGURE 2-27** Results of First Execution of Test Plan

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
>>>
```

**FIGURE 2-28** Example Output of Final Stage Testing

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
-59961031015
>>> numsecs_1900_today
-59960426182
>>>
```

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 =  ((year_birth - 1900) * avg_numsecs_year) + \
                    ((month_birth - 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 re-run the test plan, we get the following results shown in Figure 2-29.

| Date of Birth | Expected Results | Actual Results | Inaccuracy |
|---|---|---|---|
| January 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 % |
| January 4, 2000 | 364,090,570 ± 86,400 | 364,208,400 | < .009 % |
| December 31, 2009 | 48,821,332 ± 86,400 | 48,929,400 | < .05 % |
| (day before current date) | 86,400 ± 86,400 | 86,400 | 0 % |

**FIGURE 2-29** Results of Second Execution of Test Plan

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.

## CHAPTER SUMMARY

### General Topics

Numeric and String Literals
Limitations of Floating-Point Representation
Arithmetic Overflow and Underflow
Character Representation Schemes (Unicode/ASCII)
Control Characters
String Formatting Implicit and Explicit Line Joining/Variables and Variable Use/ Keyboard Input/Identifier Naming/ Keywords
Arithmetic Operators/Expressions/Infix Notation
Operator Precedence and Associativity
Data Types/Static vs. Dynamic Typing
Mixed-Type Expressions/Coercion and Type Conversion

### Python-Specific Programming Topics

Numeric Literal and String Literal Values in Python
Built-in `format` Function in Python
Variable Assignment and Storage in Python
Immutable Values in Python
Identifier Naming and Keywords in Python
Arithmetic Operators in Python
Operator Precedence and Associativity in Python
Built-in `int()` and `float()` Type Conversion Functions in Python

## CHAPTER EXERCISES

### Section 2.1

1. Based on the information in Figure 2-1, how many novels can be stored in one terabyte of storage?
2. Give the following values in the exponential notation of Python, such that there is only one significant digit to the left of the decimal point.
   **(a)** `4580.5034` **(b)** `0.00000046004` **(c)** `5000402.000000000006`
3. Which of the floating-point values in question 2 would exceed the representation of the precision of floating points typically supported in Python, as mentioned in the chapter?
4. Regarding the built-in `format` function in Python,
   **(a)** Use the `format` function to display the floating-point value in a variable named `result` with three decimal digits of precision.
   **(b)** Give a modified version of the format function in (a) so that commas are included in the displayed results.
5. Give the string of binary digits that represents, in ASCII code,
   **(a)** The string `'Hi!'`
   **(b)** The literal string `'I am 24'`
6. Give a call to `print` that is provided one string that displays the following address on three separate lines.

```
John Doe
123 Main Street
Anytown, Maryland 21009
```

7. Use the `print` function in Python to output `It's raining today.`

### Section 2.2

**8.** Regarding variable assignment,

**(a)** What is the value of variables `num1` and `num2` after the following instructions are executed?

```
num = 0
k = 5
num1 = num + k * 2
num2 = num + k * 2
```

**(b)** Are the values `id(num1)` and `id(num2)` equal after the last statement is executed?

**9.** Regarding the `input` function in Python,

**(a)** Give an instruction that prompts the user for their last name and stores it in a variable named `last_name`.

**(b)** Give an instruction that prompts the user for their age and stores it as an integer value named `age`.

**(c)** Give an instruction that prompts the user for their temperature and stores it as a float named `current_temperature`.

**10.** Regarding keywords and other predefined identifiers in Python, give the result for each of the following,

**(a)** `'int' in dir(__builtins__)`

**(b)** `'import' in dir(__builtins__)`

### Section 2.3

**11.** Which of the following operator symbols can be used as both a unary operator and a binary operator?

+, −, *, /

**12.** What is the exact result of each of the following when evaluated?

**(a)** `12 / 6.0`

**(b)** `21 // 10`

**(c)** `25 // 10.0`

**13.** If variable `n` contains an initial value of `1`, what is the largest value that will be assigned to `n` after the following assignment statement is executed an arbitrary number of times?

```
n = (n + 1) % 100
```

**14.** Which of the following arithmetic expressions could potentially result in arithmetic overflow, where `n` and `k` are each assigned integer values?

**(a)** `n * k` **(b)** `n ** k` **(c)** `n / k` **(d)** `n + k`

### Section 2.4

**15.** Evaluate the following expressions in Python.

**(a)** `10 - (5 * 4)`

**(b)** `40 % 6`

**(c)** `- (10 / 3) + 2`

**16.** Give all the possible evaluated results for the following arithmetic expression (assuming no rules of operator precedence).

```
2 * 4 + 25 - 5
```

**17.** Parenthesize all of the subexpressions in the following expressions following operator precedence in Python.

**(a)** `var1 * 8 - var2 + 32 / var3`

**(b)** `var1 - 6 ** 4 * var2 ** 3`

**18.** Evaluate each of the expressions in question 17 above for `var1 = 10`, `var2 = 30`, and `var3 = 2`.

**19.** For each of the following expressions, indicate where operator associativity of Python is used to resolve ambiguity in the evaluation of each expression.

**(a)** `var1 * var2 * var3 - var4`

**(b)** `var1 * var2 / var3`

**(c)** `var1 ** var2 ** var3`

**20.** Using the built-in type conversion function `float()`, alter the following arithmetic expressions so that each is evaluated using floating-point accuracy. Assume that `var1`, `var2`, and `var3` are assigned integer values. Use the minimum number of calls to function `float()` needed to produce the results.

**(a)** `var1 + var2 * var3`

**(b)** `var1 // var2 + var3`

**(c)** `var1 // var2 / var3`

## PYTHON PROGRAMMING EXERCISES

**P1.** Write a Python program that prompts the user for two integer values and displays the result of the first number divided by the second, with exactly two decimal places displayed.

**P2.** Write a Python program that prompts the user for two floating-point values and displays the result of the first number divided by the second, with exactly six decimal places displayed.

**P3.** Write a Python program that prompts the user for two floating-point values and displays the result of the first number divided by the second, with exactly six decimal places displayed in scientific notation.

**P4.** Write a Python program that prompts the user to enter an upper or lower case letter and displays the corresponding Unicode encoding.

**P5.** Write a Python program that allows the user to enter two integer values, and displays the results when each of the following arithmetic operators are applied. For example, if the user enters the values 7 and 5, the output would be,

```
7 + 5 = 12
7 - 5 = 2
7 * 5 = 35
7 / 5 = 1.40
7 // 5 = 1
7 % 5 = 2
7 ** 5 = 16,807
```

All floating-point results should be displayed with two decimal places of accuracy. In addition, all values should be displayed with commas where appropriate.

## PROGRAM MODIFICATION PROBLEMS

**M1.** Modify the Restaurant Tab Calculation program of section 2.2.5 so that, instead of the restaurant tax being hard coded in the program, the tax rate is entered by the user.

**M2.** Modify the Restaurant Tab Calculation program of section 2.2.5 so that, in addition to displaying the total of the items ordered, it also displays the total amount spent on drinks and dessert, as well as the percentage of the total cost of the meal (before tax) that these items comprise. Display the monetary amount rounded to two decimal places.

**M3.** Modify the Your Place in the Universe program in section 2.3.3 for international users, so that the user enters their weight in kilograms, and not in pounds.

**M4.** Modify the Temperature Conversion program in section 2.4.6 to convert from Celsius to Fahrenheit instead. The formula for the conversion is $f = (c * 9/5) + 32$.

**M5.** Modify the Age in Seconds program so that it displays the estimated age in number of days, hours, and minutes.

**M6.** Modify the Age in Seconds program so that it determines the difference in age in seconds of two friends.

## PROGRAM DEVELOPMENT PROBLEMS

**D1.** Losing Your Head over Chess

The game of chess is generally believed to have been invented in India in the sixth century for a ruling king by one of his subjects. The king was supposedly very delighted with the game and asked the subject what he wanted in return. The subject, being clever, asked for one grain of wheat on the first square, two grains of wheat on the second square, four grains of wheat on the third square, and so forth, doubling the amount on each next square. The king thought that this was a modest reward for such an invention. However, the total amount of wheat would have been more than 1,000 times the current world production.

Develop and test a Python program that calculates how much wheat this would be in pounds, using the fact that a grain of wheat weighs approximately 1/7,000 of a pound.

**D2.** All That Talking

Develop and test a Python program that determines how much time it would take to download all instances of every word ever spoken. Assume the size of this information as given in Figure 2-1. The download speed is to be entered by the user in million of bits per second (mbps). To find your actual connection speed, go to the following website (from Intel Corporation) or similar site,

www.intel.com/content/www/us/en/gamers/broadband-speed-test.html

Because connection speeds can vary, run this connection speed test three times. Take the average of three results, and use that as the connection speed to enter into your program. Finally, determine what is an appropriate unit of time to express your program results in: minutes? hours? days? other?

**D3.** Pictures on the Go

Develop and test a Python program that determines how many images can be stored on a given size USB (flash) drive. The size of the USB drive is to be entered by the user in gigabytes (GB). The number of images that can be stored must be calculated for GIF, JPEG, PNG, and TIFF image file formats. The program output should be formatted as given below.

```
Enter USB size (GB): 4

xxxxx images in GIF format can be stored
xxxxx images in JPEG format can be stored
xxxxx images in PNG format can be stored
xxxxx images in TIFF format can be stored
```

The ultimate file size of a given image depends not only on the image format used, but also on the image itself. In addition, formats such as JPEG allow the user to select the degree of compression for the image quality desired. For this program, we assume the image compression ratios given below. Also assume that all the images have a resolution of 800 $\times$ 600 pixels.

Thus, for example, a 800 $\times$ 600 resolution image with 16-bit (2 bytes) color depth would have a total number of bytes of 800 $\times$ 600 $\times$ 2 = 960,000. For a compression rate of 25:1, the total number of bytes needed to store the image would be 960000/25 = 38400.

Finally, assume that a GB (gigabyte) equals 1,000,000,000 bytes, as given in Figure 2.1.

| Format | Full Name | Color Depth | | Compression | |
|---|---|---|---|---|---|
| GIF | Graphics Interchange Format | 256 colors | 8 bits | lossless | 5:1 |
| JPEG | Joint Photographic Experts Group | 16 million colors | 24 bits | lossy | 25:1 |
| PNG | Portable Network Graphics | 16 million colors | 24 bits | lossless | 8:1 |
| TIFF | Tagged Image File Format | 280 trillion colors | 48 bits | lossless | n/a |

Note that a "lossless" compression is one in which no information is lost. A "lossy" compression does lose some of the original information.

**D4.** Life Signs

Develop and test a program that prompts the user for their age and determines approximately how many breaths and how many heartbeats the person has had in their life. The average respiration (breath) rate of people changes during different stages of development. Use the breath rates given below for use in your program:

| | Breaths per Minute |
|---|---|
| Infant | 30–60 |
| 1–4 years | 20–30 |
| 5–14 years | 15–25 |
| adults | 12–20 |

For heart rate, use an average of 67.5 beats per second.