An appropriate complete solution to the exercise is then

```
# Convert from Fahrenheit degrees to Celsius degrees:
F = 69.8
C = (5.0/9)*(F - 32)
print C

'''
Sample run:
python f2c.py
21.0
'''
```

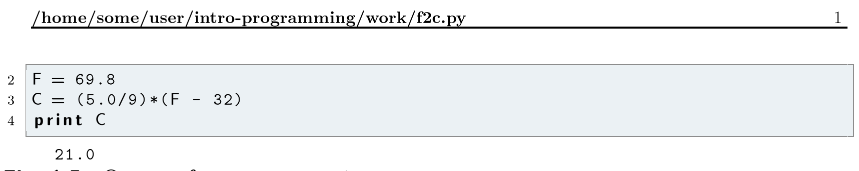
Another way of documenting the output from your own program is to use the `pyreport` program, which formats the code nicely and inserts the result of all output from the program in the resulting report. Applying `pyreport` to the `f2c.py` program is very simple:

```
Terminal
Unix/DOS> pyreport f2c.py
```

The result is a file `f2c.pdf` which you can print. Figure 1.5 displays what the printed file looks like. You can also generate a web page instead of a PDF file[30]:

```
Terminal
Unix/DOS> pyreport -t html f2c.py
```

The result now is a file `f2c.html` which you can load into a web browser. The `pyreport` program can do much more than shown in Figure 1.5. For example, mathematical formulas and graphics can easily be inserted in the resulting document[31].



/home/some/user/intro-programming/work/f2c.py 1

```
F = 69.8
C = (5.0/9)*(F - 32)
print C
```

```
21.0
```

**Fig. 1.5** Output from `pyreport`.

**Exercise 1.1.** *Compute 1+1.*

The first exercise concerns some basic mathematics: Write a Python program for printing the result of 1+1. Name of program file: `1plus1.py`. ◇

[30] The `-t` option specifies the output file type, which here is `html` – the common language in web pages. By default, the output from `pyreport` is PDF. Many other formats and options are possible, just write `pyreport` to see the possibilities.

[31] You can search for “pyreport” on google – the first hit leads you to a description of the program.

**Exercise 1.2.** *Write a "Hello, World!" program.*

Almost all books about programming languages start with a very simple program that prints the text "Hello, World!" to the screen. Make such a program in Python. Name of program file: `hello_world.py`. ◇

**Exercise 1.3.** *Convert from meters to British length units.*

Make a program where you set a length given in meters and then compute and write out the corresponding length measured in inches, in feet, in yards, and in miles. Use that one inch is 2.54 cm, one foot is 12 inches, one yard is 3 feet, and one British mile is 1760 yards. As a verification, a length of 640 meters corresponds to 25196.85 inches, 2099.74 feet, 699.91 yards, or 0.3977 miles. Name of program file: `length_conversion.py`. ◇

**Exercise 1.4.** *Compute the mass of various substances.*

The density of a substance is defined as $\varrho = m/V$, where $m$ is the mass of a volume $V$. Compute and print out the mass of one liter of each of the following substances whose densities in g/cm$^3$ are found in the file `src/files/densities.dat`: iron, air, gasoline, ice, the human body, silver, and platinum: 21.4. Name of program file: `1liter.py`. ◇

**Exercise 1.5.** *Compute the growth of money in a bank.*

Let $p$ be a bank's interest rate in percent per year. An initial amount $A$ has then grown to

$$A\left(1+\frac{p}{100}\right)^n$$

after $n$ years. Make a program for computing how much money 1000 euros have grown to after three years with 5% interest rate. Name of program file: `interest_rate.py`. ◇

**Exercise 1.6.** *Find error(s) in a program.*

Suppose somebody has written a simple one-line program for computing sin(1):

```
x=1; print 'sin(%g)=%g' % (x, sin(x))
```

Type in this program and try to run it. What is the problem? ◇

**Exercise 1.7.** *Type in program text.*

Type the following program in your editor and execute it. If your program does not work, check that you have copied the code correctly.

```
from math import pi

h = 5.0   # height
b = 2.0   # base
r = 1.5   # radius

area_parallelogram = h*b
print 'The area of the parallelogram is %.3f' % area_parallelogram
```

```
area_square = b**2
print 'The area of the square is %g' % area_square

area_circle = pi*r**2
print 'The area of the circle is %8.3f' % area_circle

volume_cone = 1.0/3*pi*r**2*h
print 'The volume of the cone is %.3f' % volume_cone
```

Name of program file: `formulas_shapes.py`. ◇

**Exercise 1.8.** *Type in programs and debug them.*

Type these short programs in your editor and execute them. When they do not work, identify and correct the erroneous statements.

(a)Does $\sin^2(x) + \cos^2(x) = 1$?

```
from math import sin, cos
x = pi/4
1_val = sin^2(x) + cos^2(x)
print 1_VAL
```

Name of program file: `sin2_plus_cos2.py`

(b)Work with the expressions for movement with constant acceleration:

```
v0 = 3 m/s
t = 1 s
a = 2 m/s**2
s = v0*t + 1/2 a*t**2
print s
```

Name of program file: `acceleration.py`

(c)Verify these equations:

$$(a+b)^2 = a^2 + 2ab + b^2$$

$$(a-b)^2 = a^2 - 2ab + b^2$$

```
a = 3,3   b = 5,3
a2 = a**2
b2 = b**2

eq1_sum = a2 + 2ab + b2
eq2_sum = a2 - 2ab + b2

eq1_pow = (a + b)**2
eq2_pow = (a - b)**2

print 'First equation:  %g = %g', % (eq1_sum, eq1_pow)
print 'Second equation: %h = %h', % (eq2_pow, eq2_pow)
```

Name of program file: `a_pm_b_sqr.py` ◇

**Exercise 1.9.** *Evaluate a Gaussian function.*

The bell-shaped Gaussian function,

$$f(x) = \frac{1}{\sqrt{2\pi}\, s} \exp\left[-\frac{1}{2}\left(\frac{x-m}{s}\right)^2\right], \tag{1.6}$$

is one of the most widely used functions in science and technology[32]. The parameters $m$ and $s$ are real numbers, where $s$ must be greater than zero. Make a program for evaluating this function when $m = 0$, $s = 2$, and $x = 1$. Verify the program's result by comparing with hand calculations on a calculator. Name of program file: `Gaussian_function1.py`. ◇

**Exercise 1.10.** *Compute the air resistance on a football.*

The drag force, due to air resistance, on an object can be expressed as

$$F_d = \frac{1}{2} C_D \varrho A V^2, \tag{1.7}$$

where $\varrho$ is the density of the air, $V$ is the velocity of the object, $A$ is the cross-sectional area (normal to the velocity direction), and $C_D$ is the drag coefficient, which depends heavily on the shape of the object and the roughness of the surface.

The gravity force on an object with mass $m$ is $F_g = mg$, where $g = 9.81\mathrm{m\,s}^{-2}$.

Make a program that computes the drag force and the gravity force on an object. Write out the forces with one decimal in units of Newton ($\mathrm{N} = \mathrm{kg\,m/s}^2$). Also print the ratio of the drag force and the gravity force. Define $C_D$, $\varrho$, $A$, $V$, $m$, $g$, $F_d$, and $F_g$ as variables, and put a comment with the corresponding unit.

As a computational example, you can initialize all variables with values relevant for a football kick. The density of air is $\varrho = 1.2$ kg $\mathrm{m}^{-3}$. For any ball, we have obviously that $A = \pi a^2$, where $a$ is the radius of the ball, which can be taken as 11 cm for a football. The mass of the ball is 0.43 kg. $C_D$ can be taken as 0.2.

Use the program to calculate the forces on the ball for a hard kick, $V = 120$ km/h and for a soft kick, $V = 10$ km/h (it is easy to mix inconsistent units, so make sure you compute with $V$ expressed in m/s). Name of program file: `kick.py`. ◇

**Exercise 1.11.** *Define objects in IPython.*

Start `ipython` and give the following command, which will store the interactive session to a file `mysession.log`:

[32] The function is named after Carl Friedrich Gauss, 1777–1855, who was a German mathematician and scientist, now considered as one of the greatest scientists of all time. He contributed to many fields, including number theory, statistics, mathematical analysis, differential geometry, geodesy, electrostatics, astronomy, and optics. Gauss introduced the function (1.6) when he analyzed probabilities related to astronomical data.

```
In [1]: %logstart -r -o mysession.log
```

Thereafter, define an integer, a real number, and a string in IPython. Apply the `type` function to check that each object has the right type. Print the three objects using printf syntax. Finally, type `logoff` to end the recording of the interactive session:

```
In [8]: %logoff
```

Leave IPython and restart it with the `-logplay mysession.log` on the command line. IPython will now re-execute the input statements in the logfile `mysession.log` so that you get back the variables you declared. Print out the variables to demonstrate this fact. ◇

**Exercise 1.12.** *How to cook the perfect egg.*

As an egg cooks, the proteins first denature and then coagulate. When the temperature exceeds a critical point, reactions begin and proceed faster as the temperature increases. In the egg white the proteins start to coagulate for temperatures above 63 C, while in the yolk the proteins start to coagulate for temperatures above 70 C. For a soft boiled egg, the white needs to have been heated long enough to coagulate at a temperature above 63 C, but the yolk should not be heated above 70 C. For a hard boiled egg, the center of the yolk should be allowed to reach 70 C.

The following formula expresses the time $t$ it takes (in seconds) for the center of the yolk to reach the temperature $T_y$ (in Celsius degrees):

$$t = \frac{M^{2/3}c\rho^{1/3}}{K\pi^2(4\pi/3)^{2/3}} \ln\left[0.76\frac{T_o - T_w}{T_y - T_w}\right] . \tag{1.8}$$

Here, $M$, $\rho$, $c$, and $K$ are properties of the egg: $M$ is the mass, $\rho$ is the density, $c$ is the specific heat capacity, and $K$ is thermal conductivity. Relevant values are $M = 47$ g for a small egg and $M = 67$ g for a large egg, $\rho = 1.038\ \mathrm{g\,cm^{-3}}$, $c = 3.7\ \mathrm{J\,g^{-1}\,K^{-1}}$, and $K = 5.4\cdot 10^{-3}\ \mathrm{W\,cm^{-1}\,K^{-1}}$. Furthermore, $T_w$ is the temperature (in C degrees) of the boiling water, and $T_o$ is the original temperature (in C degrees) of the egg before being put in the water. Implement the formula in a program, set $T_w = 100$ C and $T_y = 70$ C, and compute $t$ for a large egg taken from the fridge ($T_o = 4$ C) and from room temperature ($T_o = 20$ C). Name of program file: `egg.py`. ◇

**Exercise 1.13.** *Evaluate a function defined by a sum.*

The piecewise constant function

$$f(t) = \begin{cases} 1, & 0 < t < T/2, \\ 0, & t = T/2, \\ -1, & T/2 < t < T \end{cases} \tag{1.9}$$

can be approximated by the sum

$$S(t;n) = \frac{4}{\pi}\sum_{i=1}^{n}\frac{1}{2i-1}\sin\frac{2(2i-1)\pi t}{T} \tag{1.10}$$

$$= \frac{4}{\pi}\left(\sin\frac{2\pi t}{T} + \frac{1}{3}\sin\frac{6\pi t}{T} + \frac{1}{5}\sin\frac{10\pi t}{T} + \cdots\right) \tag{1.11}$$

It can be shown that $S(t;n) \rightarrow f(t)$ as $n \rightarrow \infty$. Write a program that prints out the value of $S(\alpha T;n)$ for $\alpha = 0.01$, $T = 2\pi$, and $n = 1, 2, 3, 4$. Let `S` ($= S(t;n)$), `t`, `alpha`, and `T` be variables in the program. A new `S`, corresponding to a new $n$, should be computed by adding one term to the previous value of `S`, i.e., by a statement like `S = S + term`. Run the program also for $\alpha = 1/4$. Does the approximation $S(\alpha T;4)$ seem to be better for $\alpha = 1/4$ than for $\alpha = 0.01$? Name of program file: `compare_func_sum.py`.

*Remark.* This program is very tedious to write for large $n$, but with a loop, introduced in the next chapter (see Exercise 2.39), we can just code the generic term parameterized by $i$ in (1.10). This approach yields a short program that can be used to evaluate $S(t;n)$ for any $n$. Exercise 4.20 extends such an implementation with graphics for displaying how well $S(t;n)$ approximates $f(t)$ for some values of $n$. A sum of sine and/or cosine functions, as in (1.10), is called a *Fourier series*. Approximating a function by a Fourier series is a very important technique in science and technology. ◇

**Exercise 1.14.** *Derive the trajectory of a ball.*

The purpose of this exercise is to explain how Equation (1.5) for the trajectory of a ball arises from basic physics. There is no programming in this exercise, just physics and mathematics.

The motion of the ball is governed by Newton's second law:

$$F_x = ma_x \tag{1.12}$$

$$F_y = ma_y \tag{1.13}$$

where $F_x$ and $F_y$ are the sum of forces in the $x$ and $y$ directions, respectively, $a_x$ and $a_y$ are the accelerations of the ball in the $x$ and $y$ directions, and $m$ is the mass of the ball. Let $(x(t), y(t))$ be the position of the ball, i.e., the horisontal and vertical corrdinate of the ball at time $t$. There are well-known relations between acceleration, velocity, and position: the acceleration is the time derivative of the velocity, and the velocity is the time derivative of the position. Therefore we have that

$$a_x = \frac{d^2x}{dt^2}, \tag{1.14}$$

$$a_y = \frac{d^2y}{dt^2}\,. \tag{1.15}$$

If we assume that gravity is the only important force on the ball, $F_x = 0$ and $F_y = -mg$.

Integrate the two components of Newton's second law twice. Use the initial conditions on velocity and position,

$$\frac{d}{dt}x(0) = v_0 \cos\theta, \tag{1.16}$$

$$\frac{d}{dt}y(0) = v_0 \sin\theta, \tag{1.17}$$

$$x(0) = 0, \tag{1.18}$$

$$y(0) = y_0, \tag{1.19}$$

to determine the four integration constants. Write up the final expressions for $x(t)$ and $y(t)$. Show that if $\theta = \pi/2$, i.e., the motion is purely vertical, we get the formula (1.1) for the $y$ position. Also show that if we eliminate $t$, we end up with the relation (1.5) between the $x$ and $y$ coordinates of the ball. You may read more about this type of motion in a physics book, e.g., [6]. ◇

**Exercise 1.15.** *Find errors in the coding of formulas.*

Some versions of our program for calculating the formula (1.2) are listed below. Determine which versions that will not work correctly and explain why in each case.

```
C = 21;    F =  9/5*C + 32;         print F
C = 21.0;  F =  (9/5)*C + 32;       print F
C = 21.0;  F =  9*C/5 + 32;         print F
C = 21.0;  F =  9.*(C/5.0) + 32;    print F
C = 21.0;  F =  9.0*C/5.0 + 32;     print F
C = 21;    F =  9*C/5 + 32;         print F
C = 21.0;  F =  (1/5)*9*C + 32;     print F
C = 21;    F =  (1./5)*9*C + 32;    print F
```

◇

**Exercise 1.16.** *Find errors in Python statements.*

Try the following statements in an interactive Python shell. Explain why some statements fail and correct the errors.

```
1a = 2
a1 = b
x = 2
y = X + 4  # is it 6?
from Math import tan
print tan(pi)
pi = "3.14159'
print tan(pi)
c = 4**3**2**3
_ = ((c-78564)/c + 32))
discount = 12%
AMOUNT = 120.-
amount = 120$
```

```
address = hpl@simula.no
and = duck
class = 'INF1100, gr 2"
continue_ = x > 0
bærtype = """jordbær"""
rev = fox = True
Norwegian = ['a human language']
true = fox is rev in Norwegian
```

Hint: It might be wise to test the values of the expressions on the right-hand side, and the validity of the variable names, seperately before you put the left- and right-hand sides together in statements. The last two statements work, but explaining why goes beyond what is treated in this chapter. ◇

**Exercise 1.17.** *Find errors in the coding of a formula.*
Given a quadratic equation,

$$ax^2 + bx + c = 0,$$

the two roots are

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}\,. \tag{1.20}$$

Why does the following program not work correctly?

```
a = 2; b = 1; c = 2
from math import sqrt
q = sqrt(b*b - 4*a*c)
x1 = (-b + q)/2*a
x2 = (-b - q)/2*a
print x1, x2
```

Hint: Compute all terms in (1.20) with the aid of a calculator, and compare with the corresponding intermediate results computed in the program (you need to add some `print` statements to see the result of `q`, `-b+q`, and `2*a`). ◇