1_python_basics_solutions

June 1, 2020

1 Python Basics

Python is a multipurpose programming language that is widely used for scientific computation and data analysis. Python is becomming more popular in the workplace as well as academia so it is a good idea to learn it to enhance your skillset.

To download the the materials click on this link: https://github.com/drjonshiach/Python-materials/archive/master.zip

These materials are designed to be covered in the following order:

1. Python basics
2. Arrays
3. If statements
4. Loops
5. Functions
6. Plotting
7. Example Programs

1.1 Contents

1.2 Jupyter notebooks

The guidance material that you are using now will to teach you how to program in Python using **Jupyter Notebooks**. These are interactive pages which allow you to edit and run Python code within the document without having to use another software package. A Jupyter notebook file has the extension `.ipynb` and can be accessed and edited in a number of differents ways.

1.2.1 Installing Python on your own machine using Anaconda

Python is an open source which means it is free to download and use. The easiest way to install Python onto your own machine is to install Anaconda which is a suite of scientific programming tools which is available for most operating systems and is free to download and use. Acaconda includes two programes which we can use to write Python programes: **Jupyter Notebook** and **Spyder**. These materials use Jupyter notebooks to teach Python but Spyder is also useful for writing longer Python programs.

To use these materials using Jupyter Notebook installed using Anaconda do the following (**note that Anaconda is already installed on PCs in the Faculty of Science and Engineering and on selected PCs in the library**):

1. If you are using a machine on campus go to step 2 else go to https://www.anaconda.com/distribution/, download the appropriate version for your machine and install it following the onscreen prompts.
2. Locate and load **Anaconda Navigator**.
3. Click on the **Launch** button underneath **Jupyter Notebook**.
4. A web broswer window will open showing you the file structure of your machine. Navigate to the directory you use to store Jupyter Notebooks and click on the filename to open it.

1.2.2 CoCalc

You can also use to use Jupyter Notebooks online using CoCalc where you can upload, edit and run notebooks in the cloud. The advantage of using an online platform is that no installation is required and you can run your notebooks from any computer or mobile device with an internet connection. In addition to Jupyter Notebook, CoCalc also has some other software used in scientific computing such as LaTeX, R and Sage.

To use these materials using Jupyter Notebook on CoCalc do the following:

1. Go to https://cocalc.com/ and sign up for an account.
2. Sign in to your account an click on **Create New Project**, give it a suitable name (e.g., `Jupyter notebooks`) and click **Create Project**.
3. Click on **Create or Upload Files...** and upload the Jupyter Notebook files.
4. Once the files have been uploaded click on an individual filename to open it.

1.2.3 Google Colab

Another online platform which you can use Jupyter Notebooks is Google Colab. The use of Google Colab requires you to have a Google account.

To use these materials using Google Colab do the following:

1. If you already have a Google account go to step 2 else go to `https://myaccount.google.com/` and click on **Create a Google account** and follow the onscreen instructions.
2. Go to `https://colab.research.google.com`;
3. Sign in with your Google account details;
4. Upload a Jupyter notebook by clicking in **File** and **Upload Notebook**.

1.2.4 Examples and exercises

These materials have a number of examples and exercises for you to try out. The examples are designed to demonstrate how the various Python commands work and the exercises give you an opportunity to put into practice what you have learned. The solutions to the exercises can be found on Moodle. In programming, there can be many different ways of achieving the same result, don't worry if your solutions do not exactly match the ones provided.

1.2.5 Using Jupyter notebooks

Jupyter notebooks consist of a number of **cells** which can either be a **text cell** like this one which contains text or a **code cell** which contains Python code than can be executed. For example, enter the following commands into the code cell below.

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

[1]:
```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

[1]:
```
86400
```

To execute the code in the above cell, select it with a click and then use the keyboard shortcut **ctrl + enter** or click on the **Run** button. To edit the code, just click the on the cell and start editing.

Here we have computed the number of seconds in a day, stored it as a variable and returned the result. Variables that we have defined in one cell can be used in later code cells. For example, enter the following command in the code cell below and executre it.

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

[2]:
```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

[2]:
```
604800
```

Note that the code above uses the previously defined variable `seconds_in_a_day` so the code cell where this is defined needs to have been executed prior to this one.

1.3 Basic arithmetic operations

We will begin with using Python to perform basic arithmetic operations since these form the fundmentals of computer programming (think of your computer as a very powerful calculator). The arithmetic operators used to perform the basic operations are shown in the table below.

Operator	Name	Python code
+	addition	`x + y`

Operator	Name	Python code
$-$	subtration	`x - y`
$\times$	multiplication	`x * y`
$\div$	division	`x / y`
x^y	exponentiation	`x ^ y`
mod	modulo (remainder)	`x % y`
$\lfloor x \div y \rfloor$	floor division	`x // y`

Example 1 The commands below perform different arithmetic operations. Enter them into the code cells below and execute each one.

```
2 + 3
```

```
[3]: 2 + 3
```

```
[3]: 5
```

```
2 - 3
```

```
[4]: 2 - 3
```

```
[4]: -1
```

```
2 * 3
```

```
[5]: 2 * 3
```

```
[5]: 6
```

```
2 / 3
```

```
[6]: 2 / 3
```

```
[6]: 0.6666666666666666
```

```
2 ** 3
```

```
[7]: 2 ** 3
```

```
[7]: 8
```

```
10 % 3
```

```
[8]: 11 % 3
```

```
[8]: 2
```

```
11 // 3
```

[9]:
```
11 // 3
```

[9]: 3

1.3.1 Order of precendence of operations

Python follows the standard rules for order of operations, i.e., BODMAS: Brackets > Orders (powers) > Division, Multiplication > Addition, Subtraction. Brackets should be used to override this where nececessary.

Example 2 The command below calculates the value of the expression $\frac{1}{2+3}$. Enter it into the code cell below and execute it.

```
1/(2 + 3)
```

[10]:
```
1/(2 + 3)
```

[10]: 0.2

Omitting the brackets from this command results in the command below. Enter this into the code cell below and execute it.

```
1/2 + 3
```

[11]:
```
1 / 2 + 3
```

[11]: 3.5

This has calculated the value of $\frac{1}{2} + 3$.

1.4 Exercise 1

1. Use Python commands to evaluate:

(a) $2 - (3 + 6)$;

[12]:
```
2 - (3 + 6)
```

[12]: -7

(b) $2(5 - 8(3 + 6))$;

[13]:
```
2*(5 - 8*(3 + 6))
```

[13]: -134

(c) $2(2-2(3-6+5(4-7)))$;

[14]:
```
2*(2 - 2*(3 - 6 + 5*(4 - 7)))
```

[14]:
```
76
```

(d) $\dfrac{2(5-4(3+8))}{3(4-(3-5))}$;

[15]:
```
2 * (5 - 4 * (3 + 8)) / (3 * (4 - (3 - 5)))
```

[15]:
```
-4.333333333333333
```

(e) $\dfrac{2(4^5)}{81-5^2}$.

[16]:
```
2*4**5/(81 - 5**2)
```

[16]:
```
36.57142857142857
```

2. Use a Python command to calculate the remainder of 14151 divided by 571.

[17]:
```
14151 % 571
```

[17]:
```
447
```

3. Use a Python command to calculate the number of times 1111 can be divided by 14.

[18]:
```
1111 // 14
```

[18]:
```
79
```

1.5 Mathematical functions

To calculate common mathematical functions such as square roots, logarithms, trigonometric functions etc. (i.e., similar to the function buttons on a scientific calculator) we need to import the `math` library. A **library** is a collection of Python programs which once imported, we can use in our calculations. To import the `math` library execute the following code cell.

[19]:
```
from math import *
```

This command will import all mathematical functions from the `math` library allowing us to use them in our calculations. Some of the most common `math` functions are listed in the table below.

Function	Name	Python code
$\sqrt{x}$	square root	`sqrt(x)`
$\sin(x)$	sine *	`sin(x)`

Function	Name	Python code
$\cos(x)$	cosine	`cos(x)`
$\tan(x)$	tangent	`tan(X)`
$\sin^{-1}(x)$	arcsin (or inverse sin) **	`asin(x)`
e^x	exponential	`exp(x)`
$\ln(x)$	natural logarithm	`log(x)`
$\log_a(x)$	log to the base a	`log(x, a)`
$x!$	factorial	`factorial(x)`
$\|x\|$	modulus (absolute value)	`abs(x)`
π	constant π	`pi`
e	constant e	`e`
$\text{round}(x)$	round to nearest integer	`round(x)`
$\lfloor x \rfloor$	round to integer below	`floor(x)`
$\lceil x \rceil$	round to integer above	`ceil(x)`

* Python assumes all angles are in radians.

** The inverse functions for the other trigonometric ratios are calculated similarly.

If we are only only a small number of functions from a library we can import individual functions by listing them after the `import` command, e.g.,

```
from math import sqrt, sin
```

which will only import the `sqrt` and `sin` functions.

Example 3 The commands below make use of `math` library functions. Enter them into the code cells below and execute each one (make sure you have executed the code cell above which imports the `math` library first).

```
sqrt(9)
```

[20]:
```
sqrt(9)
```

[20]:
```
3.0
```

```
cos(pi/4)
```

[21]:
```
cos(pi/4)
```

[21]:
```
0.7071067811865476
```

```
exp(1)
```

[22]:
```
exp(1)
```

[22]:
```
2.718281828459045
```

```
factorial(6)
```

[23]:
```
factorial(6)
```

[23]:
```
720
```

```
round(1.6)
```

[24]:
```
round(1.6)
```

[24]:
```
2
```

```
floor(1.6)
```

[25]:
```
floor(1.6)
```

[25]:
```
1
```

1.6 Exercise 2

4. Use `math` library functions to evaluate:

(a) $\sqrt{4+6^5}$;

[26]:
```
sqrt(4 + 6**5)
```

[26]:
```
88.20430828479978
```

(b) $\cos(0.8)$;

[27]:
```
cos(0.8)
```

[27]:
```
0.6967067093471654
```

(c) $\tan^{-1}(-0.4)$;

[28]:
```
atan(-0.4)
```

[28]:
```
-0.3805063771123649
```

(d) $\ln(87.95)$;

[29]:
```
log(87.95)
```

[29]:
```
4.476768471183568
```

(e) $\log_{10}(725.345)$.

[30]:
```
log(725.345, 10)
```

[30]: 2.860544621685442

1.7 Variables

Variables are used to store information which can be retrieved elsewhere in a computer program. To define a variable in Python we use the equals sign `=`. For example

```
a = value of a
```

You can also define multiple variables in one line using

```
a, b, c = value of a, value of b, value of c
```

which does the same as

```
a = value of a
b = value of b
c = value of c
```

Once a variable has been defined the information stored in it can be used in other commands in a program.

Example 4 The commands below define the 3 variables `x`, `y` and `z` and calculates their sum. Enter them into the code cell below and execute.

```
x = 1
y = 2.5
z = -3

x + y + z
```

[31]:
```
x = 1
y = 2.5
z = -3

x + y + z
```

[31]: 0.5

The commands below perform the same calculates as above but demonstrate how multiple variables can be defined in the same line. Enter them into the code cell below and execute.

```
x, y, z = 1, 2.5, -3

x + y + z
```

[32]:
```
x, y, z = 1, 2.5, -3

x + y + z
```

[32]: `0.5`

1.7.1 Variable names

A Python variable can have a short name (e.g., `x` and `y`) but sometimes it is advisable to use longer descriptive name so that your program is easier to understand (e.g., `distance`, `seconds_in_a_minute`). Variable names must adhere to the following rules:

- A variable name cannot start with a number and must start with a letter or the underscore character
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (`age`, `Age` and `AGE` are three different variables)

Example 5 The command below attempts to define a variable that violates Python's variable name rules. Enter it into the command cell and execute it.

```
1st_variable = 2
```

[33]:
```
first_variable = 2
```

Python has returned an error because the first character of this variable is a number. To overcome this we need to use a different variable name, edit the code cell above with the following command and execute it.

```
first_variable = 2
```

1.7.2 Types of variables

Python uses the following types of variables:

- **integers** - whole numbers, e.g., `1`, `10`, `-20`.
- **floating point numbers** - real numbers expressed using an integer part and fractional part separated by a decimal point, e.g., `1.6`, `3.1416`.
- **floating point exponential numbers** - numbers expressed in standard form, e.g., `2e4` has the value 2×10^4.
- **complex numbers** - numbers of the form `a+bj` where `a` and `b` are real numbers and `j` is the imaginary number, e.g., `2+3j` (note Python uses `j` to represent the imaginary number and not i which is used in mathematics). Complex numbers are not common in programming and best avoided by treating their real and imaginary parts separately.
- **booleans** - values which are either `True` or `False`.
- **strings** - seqeuences of letters, spaces and symbols, e.g., `hello world`.

When defining a variable, Python will automatically use the appropriate variable type depending on the value assigned.

Example 6 The commands below define variables using the different types of numbers available. Enter them into the code cells below and execute each one.

```
a = 2 # integer number
a
```

[34]:
```
a = 2 # integer number
a
```

[34]:
```
2
```

```
b = 1.5 # floating point number
b
```

[35]:
```
b = 1.5 # floating point number
b
```

[35]:
```
1.5
```

```
c = 5e9 # floating point exponential number
c
```

[36]:
```
c = 5e9 # floating point exponential number
c
```

[36]:
```
5000000000.0
```

```
d = 2 + 3j # complex number
d
```

[37]:
```
d = 2 + 3j # complex number
d
```

[37]:
```
(2+3j)
```

```
e = True
e
```

[38]:
```
e = True # boolean value
e
```

[38]:
```
True
```

```
f = "this is a string" # character string
f
```

[39]:
```
f = "this is a string" # character string
f
```

[39]:
```
'this is a string'
```

1.7.3 Determining the type of a variable

We can determine what type a variable is using the `type` command.

```
type(variable name)
```

Example 7 The command below output the type of the variable `a` defined in example 6. Enter it into the code cell below and execute.

```
type(a)
```

[40]:
```
type(a)
```

[40]:
```
int
```

The result `int` is returned which is an abbreviation of *integer*. Edit the code cell to check the other variables `b` through `f` from example 6.

1.7.4 Converting variable types

We can convert some variables to a different type using the following commands.

Conversion	Python command
floating point number to an integer	`int(x)`
integer number to a floating point number	`float(x)`
integer to a string	`str(x)`

Example 8 The following commands change the type of the number entered into it. Enter them into the code cells below and execute them.

```
int(2.5) # convert 2.5 to an integer
```

[41]:
```
int(2.5) # convert 2.5 to an integer
```

[41]:
```
2
```

```
float(3) # convert 3 to a floating point number
```

[42]:
```
float(3) # convert 3 to a floating point number
```

[42]:
```
3.0
```

```
str(4) # convert the integer 4 to a character
```

[43]:
```
str(4) # convert the integer 4 to a character
```

[43]:
```
'4'
```

Note that the `int(2.5)` command only returns the integer part, it does not round to the nearest integer.

1.8 Exercise 3

5. Write a program that uses variables to convert a temperature in degrees Centigrade C to degrees Fahrenheit F using the formula

$$F = \frac{9}{5}C + 32.$$

What is the equivalent temperature in Fahrenheit of 100°C?

[44]:
```
C = 100
F = 9/5*C + 32
F
```

[44]:
```
212.0
```

6. Write a program that calculates the length of the hypotenuse of a right-angled triangle given the lengths of the two other sides are 2 and 3.

[45]:
```
side1, side2 = 2, 3
hyp = sqrt(side1**2 + side2**2)
hyp
```

[45]:
```
3.605551275463989
```

7. Write a program that calculates an angle of a right-angled triangle in degrees given the lengths of the adjacent and opposite sides are 4 and 5.

[46]:
```
adj, opp = 4, 5
angle = atan(opp/adj)*180/pi
angle
```

[46]:
```
51.34019174590991
```

1.9 Formatting your code

It is good programming practice to format your code so that is can be easily read. There are a number things which you can do to help with this.

1.9.1 Spaces

In a Python program spaces are ignored, however it is common practice to use spaces either side of the arithmetic operators so that it is more readable. It is also advisable to seperate blocks of code with a blank line.

Example 9 The commands below perform the same operations. Enter them into the code cells below and execute them to check they return the same value.

```
1*2+3*4+5/6
```

[47]:
```
1*2+3*4+5/6
```

[47]:
```
14.833333333333334
```

```
1*2 + 3*4 + 5/6
```

[48]:
```
1*2 + 3*4 + 5/6
```

[48]:
```
14.833333333333334
```

1.9.2 Comments

A **comment** in a program is text that is ignored by Python when the code is executed and are useful to helping people understand the program. Comments in Python can be used in two ways:

```
# this is a comment
```

Here any text on the same line to the right of `#` is ignored. These are useful for short comments. For longer comments that span multiple lines we can use `"""` to start and end a comment.

```
"""this is a comment
that spans multiple
lines"""
```

Example 10 The program in the code cell below makes use of comments. Enter is into the code cell below and execute it to check that it runs. Note how the lines of the program are spaced out to improve the readability of the code. Blank lines are also ignored by Python and can be used to improve the readability of a program.

```
"""This program calculates the
sum of two numbers"""

x = 4 # first number
y = 7 # second number

# Calculate the sum of the two numbers
x + y
```

[49]:
```
"""This program calculates the
sum of two numbers"""

x = 4 # first number
y = 7 # second number

# Calculate the sum of the two numbers
x + y
```

[49]: 11

1.9.3 Splitting lines of code

To split a line of code we use the \ symbol. This is useful when a single line of code uses a lot of horizontal space.

Example 11 The program in the code cell below uses the \ symbol to split one of the lines of code over two lines. Enter it into the codel cell below and execute it to check that it runs.

```
# Define the variables
x, y, z = 2, 3, 4

# The code below has been split over two lines (rather unnecessarily)
sum_xyz = x + y \
+ z

# Output the sum of x, y and z
sum_xyz
```

[50]:
```
# Define the variables
x, y, z = 2, 3, 4

# The code below has been split over two lines (rather unnecessarily)
sum_xyz = x + y \
+ z

# Output the sum of x, y and z
sum_xyz
```

[50]: 9

1.10 Printing output

To output text or the value of a variable within a Python program we can use the `print()` command.

```
print("some text")
print(x)
```

Example 12 The commands in the code cells below show how the `print` command is used to print character strings and numbers. Enter then into the code cells below and execute them to see the results.

```
print("Hello world")
```

[51]:
```
print("hello world")
```

```
hello world
```

```
x = 2
print(x)
```

[52]:
```
x = 2
print(x)
```

```
2
```

```
a, b = 2, 3
print(a, b)
```

[53]:
```
a, b = 2, 3
print(a, b)
```

```
2 3
```

1.10.1 Printing text and numbers

Sometimes it is desirable to be able to output text alongside numbers. This can be done using the following

```
print("some text {} some more text {} even more text".format(x, y))
```

The two sets of curly braces `{}` will be replaced with the values of `x` and `y` respectively.

Example 13 The command in the code cell below uses a `print` statement to combine the printing of text and numbers. Enter them into the cell cell below and execute it to see the output (if Python returns an error make sure you have executed the first Section 1.2.3 in this notebook).

```
print("There are {} seconds in a day and {} seconds in a week." \
      .format(seconds_in_a_day, seconds_in_a_week))
```

[54]:
```
print("There are {} seconds in a day and {} seconds in a week." \
      .format(seconds_in_a_day, seconds_in_a_week))
```

```
There are 86400 seconds in a day and 604800 seconds in a week.
```

1.10.2 Formatted output

If we want more control over the output of variables we can use **formatted output**.

```
print("{:a.bf}".format(x))
```

This will output the value of the floating point number `x` using `a` character spaces (including the decimal point) using `b` decimal places.

The `f` in the code above specifies the type of variable we are ouputted. The specifiers for the main variable types are given in the table below.

Variable type	Python code
integer number	`d`
floating point number	`f`
exponential number	`e`
character string	`s`

Example 14 The commands below show the use of formatted output. Enter them into the code cells below and execute them to see the result.

```
print("Then, shalt thou count to {:10d}, no more, no less.".format(3))
```

[55]:
```
print("Then, shalt thou count to {:10d}, no more, no less.".format(3))
```

```
Then, shalt thou count to          3, no more, no less.
```

Here the integer 3 was printed using 10 character spaces so there are 9 empty spaces to the left of the 3.

```
from math import pi

print("The value of pi to 4 decimal places is {:6.4f}.".format(pi))
```

[56]:
```
from math import pi

print("The value of pi to 4 decimal places is {:6.4f}.".format(pi))
```

```
The value of pi to 4 decimal places is 3.1416.
```

Here 6 character spaces were used to display π, 1 space for the 3, 1 space for the decimal point and 4 spaces for the decimal places.

```
print("The speed of light is {:0.2e} m/s.".format(2.9979e8))
```

[57]:
```
print("The speed of light is {:0.2e} m/s.".format(2.9979e8))
```

```
The speed of light is 3.00e+08 m/s.
```

Here the number 2.9979×10^8 was printed to 2 decimal places using the minimum number of spaces required due to the 0 after the colon.

```
name = "Andrew Wiles"
year = 1995

print("Fermat's last theorem was proven by {:20s} in {}.".format(name, year))
```

[58]:
```
name = "Andrew Wiles"
year = 1995

print("Fermat's last theorem was proven by {:20s} in {}.".format(name, year))
```

```
Fermat's last theorem was proven by Andrew Wiles         in 1995.
```

Here the character string `"Andrew Wiles"` was printed using 20 character spaces. Note that the unused spaces appear after the string.

Note that often it takes a bit of trial and error to get the right spacings in your program. Don't be afraid to experiment with your code.

1.10.3 Printing multiple lines

It is possible to print multiple lines with a single print statement using the command `\n` which moves to the next line.

Example 15 The command below uses a single `print` statement to print text over multiple lines. Enter it into the code cell below and execute it to see the result.

```
print("This text \nis printed\n\non multiple lines \n\n\nusing a single print command.")
```

[59]:
```
print("This text \nis printed\n\non multiple lines \n\n\nusing a single print␣
↪command.")
```

```
This text
is printed

on multiple lines


using a single print command.
```

1.10.4 Suppressing the carriage return

By default Python moves to the next line on completion of a print command. To suppress this we can use the following

```
print("some text", end="")
```

Example 16 The commands in the code cell below shows how the use of `end=""` suppresses the carriage return. Enter them into the code cell below and execute it to see the result.

```
print("This text ", end="")
print("is printed on the ", end="")
print("some line using multiple print statements")
```

[60]:
```
print("This text ", end="")
print("is printed on the ", end="")
print("some line using multiple print statements")
```

```
This text is printed on the some line using multiple print statements
```

1.10.5 Printing long lines of text

The \ command can come in useful to split up a `print` comand when you are printing long lines of text.

Example 17 The command below shows have we can use the \ symbol to print long lines of text using a single `print` command. Enter it into the code cell below execute it to see the result.

```
print("some text, " \
      "some more text in the same print command.")
```

[61]:
```
print("some text, " \
      "some more text in the same print command.")
```

```
some text, some more text in the same print command.
```

1.11 Exercise 4

8. Output the value of e to 20 decimal places.

[62]:
```
from math import e

print("{:0.20f}".format(e))
```

```
2.71828182845904509080
```

9. Write a program that uses the formula below to calculate the monthly repayments and total value of a mortgage of £100,000 taken out over 20 years at a fixed annual interest rate of 5%.

$$C = \frac{rP}{1-(1+r)^{-N}}$$

C is the monthly repayment amount, r is the monthly interest rate, P is the amount borrowed, N is the number of monthly repayments. Use `print` commands to output the loan amount,

the duration of the mortgage, the annual interest rate, the monthly repayments and total value of the mortgage.

[63]:
```
P = 100000   # initial loan amount
N = 20*12    # number of months
r = 5/12/100 # monthly interest rate

C = r*P/(1 - (1 + r)**-N)

print("The monthly repayments for mortgage of £{:1.2f} taken at a fixed " \
      "interest rate\nof {:1.2f}% over {} months are £{:1.2f} "\
      "resulting in a total cost of £{:1.2f}."\
      .format(P, r*100*12, N, C, N*C), end="")
```

```
The monthly repayments for mortgage of £100000.00 taken at a fixed interest rate
of 5.00% over 240 months are £659.96 resulting in a total cost of £158389.38.
```

10. Write a Python program that calculates the number of years, days and minutes that are in 1 billion seconds. Output the answers in a single meaningful sentence. Hint: The `x // y` and `x % y` commands will come in useful here.

[64]:
```
x = int(1e9)

# Conversion quantities
seconds = x
seconds_in_a_minute = 60
seconds_in_an_hour = 60*seconds_in_a_minute
seconds_in_a_day = 24*seconds_in_an_hour
seconds_in_a_year = 365*seconds_in_a_day

# Calculate number of years and the seconds left over
years = seconds // seconds_in_a_year
seconds = seconds % seconds_in_a_year

# Calculate the number of days and the seconds left over
days = seconds // seconds_in_a_day
seconds = seconds % seconds_in_a_day

# Calculate the number of hours and the seconds left over
hours = seconds // seconds_in_an_hour
seconds = seconds % seconds_in_an_hour

# Calculate the number of minutes and the seconds left over
minutes = seconds // seconds_in_a_minute
seconds = seconds % seconds_in_a_minute

# Output the results
print("{} seconds is the same as {} years, {} days, {} hours, {} minutes and "\
```

```
      "{} seconds.".format(x, years, days, hours, minutes, seconds))
```

```
1000000000 seconds is the same as 31 years, 259 days, 1 hours, 46 minutes and 40
seconds.
```

11. Use suitable `print()` and `math` library commands to produce the following table where each value is printed using 8 character spaces and 4 decimal places.

```
   x    |  sqrt(x) |  exp(x)  |   ln(x)  |  cos(x)
-------------------------------------------------------
 1.0000 |   1.0000 |   2.7183 |   0.0000 |   0.5403
 2.0000 |   1.4142 |   7.3891 |   0.6931 |  -0.4161
 5.0000 |   2.2361 | 148.4132 |   1.6094 |   0.2837
```

[65]:
```
from math import *

print("    x    |  sqrt(x) |  exp(x)  |   ln(x)  |  cos(x)")
print("-------------------------------------------------------")

x = 1
print("{:8.4f} | {:8.4f} | {:8.4f} | {:8.4f} | {:8.4f}"\
      .format(x, sqrt(x), exp(x), log(x), cos(x)))
x = 2
print("{:8.4f} | {:8.4f} | {:8.4f} | {:8.4f} | {:8.4f}"\
      .format(x, sqrt(x), exp(x), log(x), cos(x)))
x = 5
print("{:8.4f} | {:8.4f} | {:8.4f} | {:8.4f} | {:8.4f}"\
      .format(x, sqrt(x), exp(x), log(x), cos(x)))
```

```
   x    |  sqrt(x) |  exp(x)  |   ln(x)  |  cos(x)
-------------------------------------------------------
 1.0000 |   1.0000 |   2.7183 |   0.0000 |   0.5403
 2.0000 |   1.4142 |   7.3891 |   0.6931 |  -0.4161
 5.0000 |   2.2361 | 148.4132 |   1.6094 |   0.2837
```