CHAPTER 1

The Python Shell

LEARNING GOAL: You can use Python as a scientific calculator.

1.1 IN THIS CHAPTER YOU WILL LEARN

- How to use the Python shell as a scientific calculator
- How to calculate the ΔG of ATP hydrolysis
- How to calculate the distance between two points
- How to create your own Python module

1.2 STORY: CALCULATING THE ΔG OF ATP HYDROLYSIS

1.2.1 Problem Description

$$\text{ATP} \rightarrow \text{ADP} + \text{P}_i$$

The hydrolysis of one phosphodiester bond from ATP results in a standard Gibbs energy (ΔG^0) of –30.5 kJ/mol. According to biochemistry textbooks, the real ΔG value depends on the concentration of the compounds. And these concentrations can differ quite a lot among tissues (see Table 1.1, according to Berg et al.[*]).

[*] Jeremy M. Berg, John L. Tymoczko, and Lubert Stryer, *Biochemistry*, 5th ed. (New York: W. H. Freeman, 2002).

TABLE 1.1 Compound Concentration in Different Tissues.

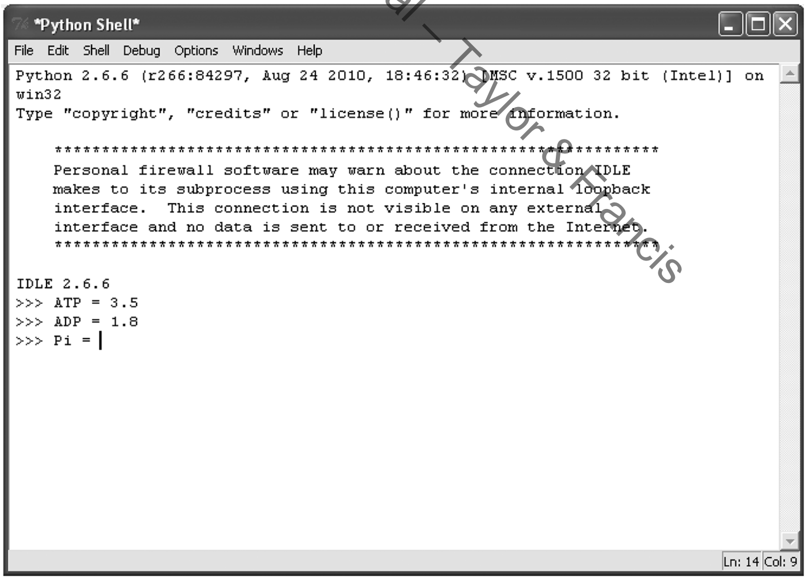
Tissue	**[ATP] [mM]**	**[ADP] [mM]**	**$[P_i]$[mM]**
Liver	3.5	1.8	5.0
Muscle	8.0	0.9	8.0
Brain	2.6	0.7	2.7

How can the real ΔG value for ATP hydrolysis be calculated? The Gibbs energy as a function of the concentrations of the compounds can be written as

$$\Delta G = \Delta G^0 + RT * \ln ([ADP] * [P_i] / [ATP])$$

You can insert values from the table into this equation with many tools (e.g., a pocket calculator, the Windows calculator application, or your mobile phone). In this book, you are going to learn a much more efficient and powerful tool for calculations and data management: the Python programming language.

Using Python, you can do the calculation for *liver tissue* in the interactive Python interpreter (see Figure 1.1). The prompt >>> indicates the



```
Python 2.6.6 (r266:84297, Aug 24 2010, 18:46:32) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 2.6.6
>>> ATP = 3.5
>>> ADP = 1.8
>>> Pi = 
```

FIGURE 1.1 The Python shell. *Note:* To start it you have to type "python" at the prompt of the UNIX terminal shell (in UNIX/Linux or Mac OS X) or start 'Python (command line)' from the program menu (in Windows).

place where you can enter commands, and it appears when you start a Python interactive session (see Section 1.3.1). Python commands must be typed just at the right side of the prompt.

1.2.2 Example Python Session

```
>>> ATP = 3.5
>>> ADP = 1.8
>>> Pi = 5.0
>>> R = 0.00831
>>> T = 298
>>> deltaG0 = -30.5
>>>
>>> import math
>>> deltaG0 + R * T * math.log(ADP * Pi / ATP)
-28.161154161098693
```

Source: Adapted from code published by A.Via/K.Rother under the Python License.

1.3 WHAT DO THE COMMANDS MEAN?

In programming, most of what you do can be roughly summarized in five points: organize data, use other programs, calculate things, and read and write data. The previous example contains three. First, it organizes the parameters for the ΔG formula by storing them in variables. Variables are containers that help you not to write the same numbers repeatedly. Second, it uses an external program to calculate the logarithm: the `math.log(x)` function calculates the logarithm of `x` and is accessed through the `import` statement, which makes available extra Python functions by connecting a program to other *modules* (`math` in the example) where such functions are stored. Modules are programming units collecting variables, functions, and other useful objects. They are always stored in files. See Box 1.1 for more on the `import` statement and Python modules.

Finally, the example in Section 1.2.2 calculates the ΔG value. Simple arithmetical calculations work very similar to a pocket calculator. The second part of this book is dedicated to other ways in which you can manipulate your data. The first thing you can try to do yourself is to start the calculation in the previous section.

BOX 1.1 THE `import` STATEMENT AND THE CONCEPT OF MODULES

When you write

```
>>> import math
```

you are connecting to the `math` module. What exactly is `math`? `math` is a file on your computer; its actual name is `math.py`. The `.py` extension stands for Python, and the file contains Python instructions, i.e., definitions of variables and functions and instructions (for calculating things). The `math.py` file, in particular, contains instructions for the definition and calculation of mathematical functions (e.g., `sqrt()`, `log()`, etc.).

In Python, text files that contain Python instructions are called *modules*. The `import` instruction is needed to access an external module and read its content. This way, all the definitions present in a module will become available when you import it: effectively, the code is shared and can be used in many different programs.

How can you know which mathematical functions are defined in the `math` module? Either you can browse the Internet and find and open the file `math.py` and read its contents, or you can use the instruction

```
>>> import math
>>> dir(math)
```

You can use the `dir(math)` instruction only after having imported the `math` module, otherwise the `dir()` function does not know what its argument is. As a result, you will see a complete list of variables and functions present in the `math` module:

```
['__doc__', '__name__', '__package__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copy-
sign', 'cos', 'cosh', 'degrees', 'e', 'exp', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'hypot',
'isinf', 'isnan', 'ldexp', 'log', 'log10', 'log1p',
'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt',
'tan', 'tanh', 'trunc']
```

You can get a short explanation of each function by typing, for instance,

```
>>> help(math.sqrt)
```

1.3.1 How to Run the Example on Your Computer

The Python programming language is a program that needs to be started before you can use it. On Linux (Ubuntu) and Mac OS X, Python is already installed and can be started from a text console by typing "python" at the command line prompt. See Appendix D to learn how to run a program from a text terminal. On Windows, you need to install Python first and then start a Python shell window in 'Start' → 'All programs' → 'Python' → 'IDLE or Python (command line)'. First, download Python 2.7 from www.python.org. Install it, then start 'Python (command line)' from the program menu. For more details, see Box 1.2. When you see the >>> sign in a text window on your screen, you have succeeded and are ready to write program code (see Figure 1.1). The Python shell can be exited by typing Ctrl+D.

BOX 1.2 HOW TO INSTALL PYTHON

On Linux and Mac OS X, Python is already installed. In rare cases where it is not, you can get the most recent version from the package manager or by typing at a command terminal:

```
sudo apt-get install python
```

On Windows you need to download the Python Windows installer from www.python.org. Make sure you download version 2.7 of Python. Versions 3.0 and above are at the time of writing experimental and not compatible with this book. The programming language can be installed like most programs by clicking and accepting the defaults.

To check whether your installation was successful, you should start Python. There are two ways to run Python code:

1. Using the interactive mode (Python shell). On Linux and Mac OS X, you type "python" from a text console and press Enter. On Windows you choose 'Start' → 'Programs' → 'Python 2.7' → 'Python (command line)'. The Python shell will start in a separate window. Alternatively, you can open a text console by entering "cmd" in the 'Start' → 'Execute' dialog, then change the directory to C:\Python27 and type "python" there. When you see the prompt >>>, your installation is successful.
2. Writing code into a script file having the `.py` extension (e.g., `my_script.py`) and executing the script by typing at the UNIX/Linux shell prompt:

```
python my_script.py
```

See also Section 2.3.1, "How to Execute the Program."

The Python Shell

The interactive mode is ideal for learning and for testing pieces of code. Each single instruction is written and directly executed. You can write instructions after the >>> sign and confirm them by pressing Enter. Each instruction is executed immediately.

```
>>> ATP = 3.5
>>> ATP
3.5
```

You can use the interactive mode for numerical calculations:

```
>>> 3 * 4
12
>>> 12.5 / 0.5
25.0
>>> (12.5 / 0.5) * 100
2500.0
>>> 3 ** 4
81
>>> 3 ** (4 + 2)
729
```

A disadvantage of the Python shell is that when you exit the session (by typing Ctrl+D), your code gets lost. Therefore, you can only save the code you have written by copying and pasting the instructions to a text editor. Text editors are described in Box 2.2 and Box D.2. To save your code, writing Python instructions to files directly is more convenient. See Example 1.1 or Chapter 2.

If something goes wrong, Python returns error messages, the content of which depends on the type of error. For example, if you mistype an instruction and write, for example,

```
>>> imprt math
```

instead of

```
>>> import math
```

you will get a message saying "`SyntaxError: invalid syntax`" plus some additional information to help you correct the error(s). The errors you can encounter and how you can manage them are described in Chapter 12. Making errors is normal in programming.

1.3.2 Variables

In Section 1.2.2, a number of variables are initially defined. That is, the values to be used in the calculation are put into named containers.

For example, when writing

```
>>> ATP = 3.5
```

the computer will remember the number `3.5` under the name `ATP`, so when you write later

```
>>> ATP
```

the computer will print the value `3.5`.

In the same way, all numbers used (`1.8`, `5.0`, `0.00831`, `298`, and `–30.5`) are recorded each in its own variable (`ADP`, `Pi`, `R`, `T`, and `deltaG0`, respectively). Note that none of the numbers have a unit. Like when using a pocket calculator, you need to take care to convert them properly. This is why for the gas constant *R* (8.31 J/kmol) the value

```
>>> R = 0.00831
```

is used, so that it fits to the unit of ΔG^0 (kJ/kmol). As with a pocket calculator, you are responsible for converting numbers to appropriate units.

Each kind of object can be stored in a variable. In other words, you can "label" a piece of data with a name and, instead of writing the whole data every time you need it, you can just use the name of the variable. The more complex and the more frequently used the data are (e.g., the nucleotide sequence of a whole gene), the more convenient it is to use a variable name in its place.

So, if you want to use the Gibbs energy value for ATP hydrolysis

$$\Delta G^0 = -30.5 \text{ kJ/mol}$$

several times, it would be better to put it into a variable and use the variable name instead of the whole number.

The operator used to assign an object to a variable name is the equal sign =:

```
>>> deltag = -30.5
```

Python distinguishes between integer and floating-point numbers:

```
>>> a = 3
>>> b = 3.0
```

In Python jargon, we say that the two variables `a` and `b` have different data *types*. The variable `a` is an *integer*; `b` is a *float*. Their difference can be seen when you divide these numbers by another integer:

```
>>> a / 2
1
>>> b / 2
1.5
```

You can enforce conversion of an integer to a float number by dividing the integer by a float:

```
>>> a / 2.0
1.5
```

You can assign numbers, text, and many other kinds of data to variables. More generally, you can refer to the data as Python *objects*. In the following example, you assign a floating-point number object to a variable:

```
>>> deltag = -30.5
```

If you assign a new value to an existing variable name, the second value will overwrite the first. In other words, by setting

```
>>> deltag = -28.16
```

`deltag` is now `-28.16` and no longer `-30.5`. In later chapters, you will encounter more types of data.

There are some *rules* in the choice of variable names:

- Some words cannot be used for variable names because they have a meaning in Python. For instance, `import` cannot be used as a variable name. For a complete list of reserved words, see Box 1.3.

- The first character of a variable name cannot be a number.

- Variable names are case sensitive. Thus, `var` and `Var` are different names.
- Most special characters, i.e., all of `$ % @ / \ . , [ ] ( ) { } #` are not allowed.

BOX 1.3 RESERVED WORDS IN PYTHON

Python reserved words cannot be used for variables because they have a meaning in Python. Here are some examples: `and`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `exec`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `not`, `or`, `pass`, `print`, `raise`, `return`, `try`, `while`.

Q & A: DOES IT MATTER WHETHER I USE UPPERCASE OR LOWERCASE FOR VARIABLE NAMES?

Try the following code:

```
>>> ATP = 3.5
>>> atp = 8.0
>>> ATP
```

The result of the last command is 3.5, not 8.0. As a general rule in Python, it makes a difference whether uppercase or lowercase is used for naming variables.

Q & A: WHAT HAPPENS WHEN I USE A VARIABLE FOR THE FIRST TIME?

In some programming languages, you need to list all variables that you want to use and explicitly reserve memory for them. In Python, you don't have to do that. The Python interpreter treats everything as *objects*. This means every time you use a new variable name, Python recognizes the nature of the data (integer, float, text, etc.) and reserves sufficient memory for it. Python also automatically associates a list of *instruments* to the variable type. For example, the numerical variables `a` and `b` defined previously "know" that you can add, subtract, and multiply them and perform all numerical operations displayed in Table 1.2.

TABLE 1.2 Arithmetical Operations in Python.

Operator	Meaning
`a + b`	addition
`a - b`	subtraction
`a * b`	multiplication
`a/b`	division
`a ** b`	power (a^b)
`a % b`	modulo: the remainder of the division a / b
`a // b`	floor division, rounds down
`a * (b + c)`	parentheses, b + c will be done before the multiplication

1.3.3 Importing Modules

After defining variables, the next command in the Python session in Section 1.2.2 imports a module with mathematical functions. In Python, `import` is a command that activates installed extra libraries or single variables and functions. `math` is the name of a library module that is automatically installed with Python. It is activated by

```
>>> import math
```

In this chapter, the `log` function from the `math` module is being used to calculate a logarithm. For a complete list of available functions in `math`, see http://docs.python.org/2/library/math.html or type

```
>>> dir(math)
```

in the Python shell.

Every module can contain functions and variables. Modules are used to reuse code and to divide big programs into smaller parts and therefore organize them better. Every time you need, for example, a constant like the gas constant R, you can fetch it from its module without redefining it. Modules collected in the Python Standard Library are basically extra functions that somebody else wrote and optimized for you.

Python makes available hundreds of modules, i.e., sets of functions that become available through the `import` command. Moreover, you can create your own modules by writing Python instructions to a text file and saving the file with the `.py` extension (see Example 1.2). Modules will be discussed in more detail in Part III of the book.

To employ the logarithm function from the `math` module, we used the notation `math.log`. The *dot* between the module and function name has a very special role in Python. The dot is a "linker" between objects. We say that the object on the right of the dot is an attribute of the object on the left. So,

```
>>> math.log
```

means that the `log` object (a function) is an attribute of the `math` object (a module). In other words, `log` is a part of the `math` module, and if you want to use it after importing the module, you have to refer to it using the dot syntax. This is true for everything in Python. Whenever an object A contains another object B, the syntax to use it is `A.B`. If B contains C, and A contains B, you can write: `A.B.C`.

Objects can also be imported selectively from modules. In other words, you may want to import a single object or a few objects instead of the whole content of a module. To import only the logarithm function instead of the entire `math` module, you can write

```
>>> from math import log
```

To use the imported function now, instead of writing `math.log`, you need to directly write `log`. The question of which variable and function names are available at a given moment is best explained by the concept of Python namespaces (see Box 1.4).

BOX 1.4 NAMESPACES

The collection of object *names* (of variables, functions, etc.) defined in a module is called the *namespace* of that module. Each module has its own namespace. For instance, the namespace of the `math` module contains the names `pi`, `sqrt`, `cos`, and many others. The namespace of the `random` module contains none of the former but contains the names `randomint` and `random` instead. Even the Python shell has its own namespace, containing, for example, `print`.

The same name (e.g., `pi`) in two different modules may indicate two distinct objects, and the dot syntax makes it possible to avoid confusion between the namespaces of the two modules. What actually happens when the command `import` is executed? It happens that the code written in the imported module is entirely read and interpreted and its namespace is imported as well but kept separated from the namespace of the importing module. So, if you write

```
>>> import math
>>> sqrt(16)
```

```
Traceback (most recent call last):
       File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
>>>
```

the name `sqrt` will not be recognized as an attribute of the `math` module unless you use the dot syntax

```
>>> math.sqrt(16)
4.0
>>>
```

But if you use

```
>>> from math import sqrt
```

you are actually merging the `math` namespace with the Python shell namespace. So, now, you can directly use

```
>>> sqrt(16)
4.0
>>>
```

You have to be careful when you merge the namespaces of two modules and be aware of how you are using variable names. In fact, if you import everything from the `math` module using the following instruction:

```
>>> from math import *
```

you will have

```
>>> pi
3.141592653589793
```

but by typing

```
>>> pi = 100
```

you are actually overwriting the `pi` variable imported from the `math` module, and `pi` will no longer have the π value. This may generate unexpected results in your calculations.

Q & A: WHY DO I HAVE TO IMPORT THE `math` LIBRARY WHEN IT IS INSTALLED ANYWAY?

In Python, there are about 100 different libraries in addition to `math`. Together, they have several thousand functions. Searching through all functions would make it easy to get lost even for experienced programmers. This is why they have been grouped into modules. Thus, you can add extra components to a Python program only if you need them.

1.3.4 Calculations

In the final part of the ΔG example, the calculation is done. The translation of the formula in Section 1.2.2 contains an addition (+), two multiplications (*), a division (/), and the natural logarithm (`math.log(...)`). The parentheses after the `log` are obligatory. Python also supports subtraction (–), power (`**`), floor division (//, rounding down), and modulo (`%`, resulting in the remainder of a division).

```
>>> deltaG0 + R * T * math.log(ADP * Pi / ATP)
```

Upon pressing Enter, you will see the result displayed immediately:

```
-28.161154161098693
```

Standard Arithmetical Operations

Most calculations will probably be simpler than calculating ΔG values. Arithmetical operations can be done right away from the command prompt

```
>>> a = 3
>>> b = 4
>>> a + b
7
```

Or you can leave the variables away and write numbers directly:

```
>>> 3 + 4
7
```

Table 1.2 gives an overview of the available arithmetical operations in Python.

Q & A: DO I NEED TO WRITE NUMBERS WITH DECIMAL PLACES?

There are two things to note: First, when you perform a calculation with integer numbers, the result is also an integer number. Second, when you calculate with floating-point numbers, the result will also be a floating-point number. For instance, if you execute the division

```
>>> 4 / 3
1
```

The result is `1` as an integer number, because it gets rounded down automatically. However, the result changes when you add one decimal place:

```
>>> 4.0 / 3.0
1.3333333333333333
```

The result of the second division is given with a precision of 16 decimal places. When you put together integer and floating-point numbers in a calculation, the result will also be a float.

Q & A: WHY DO WE USE VARIABLES AT ALL? WOULDN'T THE ΔG EXAMPLE BE SIMPLER IF WE JUST PUT THE NUMBERS INTO THE FORMULA DIRECTLY?

Yes and no. Yes, because it is fewer lines to write. No, because your code becomes much harder to read and not reusable. Consider the line for calculating the ΔG value:

```
>>> -30.5 + 0.000831 * 298 * math.log(1.8 * 5.0 / 3.5)
-30.26611541610987
```

How long would it take you to figure out that this result is actually wrong, although the calculation is mathematically correct? The problem becomes easier to spot if you have

```
>>> R = 0.000831
```

whereas it should be

```
>>> R = 0.00831
```

In the first line, one decimal place was forgotten while converting the units. This is a very common programming error. Often errors have nothing to do with the program itself but with misconceptions about the data. Ideas on how you can spot such problems more easily are explained in Chapter 12 and Chapter 15.

Mathematical Functions

When you issue the command

```
>>> import math
```

a set of mathematical functions from the `math` module is made available in the current Python interactive session. The most important functions from `math` are listed in Table 1.3.

TABLE 1.3 Some Important Functions Defined in the `math` Module.

Function	Meaning
`log(x)`	natural logarithm of x ($ln\ x$)
`log10(x)`	decadic logarithm of x ($log\ x$)
`exp(x)`	natural exponent of x (e^x)
`sqrt(x)`	square root of x
`sin(x), cos(x)`	sine and cosine of x (x given in radians)
`asin(x), acos(x)`	arcsin and arccos of x (result in radians)

When you are using functions in Python, the parentheses are mandatory:

```
>>> math.sqrt(49)
7.0
```

`math` also defines the constants `math.pi` ($\pi = 3.14159$) and `math.e` ($e = 2.71828$). They can be used just as any variable. For example, to calculate the volume of a 50 ml Falcon tube (a plastic cylinder used for centrifugation) that is 115 mm long and 30 mm wide, you can use `math.pi`:

```
>>> diameter = 30.0
>>> radius = diameter / 2.0
>>> length = 115.0
>>> math.pi * radius ** 2 * length / 1000.0
81.2887099116359
```

Source: Adapted from code published by A.Via/K.Rother under the Python License.

1.4 EXAMPLES

Example 1.1 How to Calculate the Distance between Two Points

A point in the three-dimensional space is defined by its Cartesian coordinates (x, y, z). The distance d between two points p_1 and p_2, the coordinates of which are (x_1, y_1, z_1) and (x_2, y_2, z_2), respectively, is given by the following equation:

$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

The coordinates of the two points can be stored in six variables: `x1`, `y1`, `z1` and `x2`, `y2`, `z2`, respectively. You need two methods from the math module (`pow()` and `sqrt()`). In the following script, we actually import all functions (`*`) from the `math` module. The `pow(i, j)` method has two arguments: the number `i` you want to raise to the power of `j`, and `j`.

```
>>> from math import *
>>> x1, y1, z1 = 0.1, 0.0, -0.7
>>> x2, y2, z2 = 0.5, -1.0, 2.7
>>> dx = x1 - x2
>>> dy = y1 - y2
>>> dz = z1 - z2
>>> dsquare = pow(dx, 2) + pow(dy, 2) + pow(dz, 2)
>>> d = sqrt(dsquare)
>>> d
3.5665109000254018
```

Example 1.2 How to Create Your Own Modules

Technically, a Python module is a text file ending with `.py` (see Box 1.1). You can place variables and Python code, functions, etc., there. A short Python module can be written and used quickly. For instance, you could outsource the ATP constant to a module in four steps:

1. Create a new text file with a text editor.
2. Give it a name ending with `.py` (e.g., `hydrolysis.py`).
3. Add some code. For example, you could add the ATP constant

   ```
   ATP = -30.5
   ```

4. Finally, import the module from the Python shell:

   ```
   >>> import hydrolysis
   ```

   or

   ```
   >>> from hydrolysis import ATP
   ```

For the import to work, you need to store the module file in the same directory where you started the Python shell (on Linux and Mac) or in the Python library (C:/Python7/lib/site-packages/ on Windows). You may also save your modules to another directory (you may want to have a special directory where you collect all your modules) and add the directory path to a special Python variable (called `sys.path`, i.e., the variable `path` belonging to the module `sys`). Later in the book we will explain how to do it.

1.5 TESTING YOURSELF

Exercise 1.1 Calculate the ΔG Value for All Three Tissues

In which tissue does ATP hydrolysis set the most energy free? Use the code provided earlier to answer the question. (See Table 1.1.)

Exercise 1.2 Convert the Values to kcal

Calculate the three ΔG values for all three tissues to kcal/mol. The conversion factor is 1 kcal/mol = 4.184 kJ/mol.

Exercise 1.3 pH Calculation

In a solution you have a proton concentration of 0.003162 mM. What is the pH of the solution?

Exercise 1.4 Exponential Growth

Given optimal growth conditions, a single *E. coli* bacterium can divide within 20 minutes. If the conditions stay optimal, how many bacteria are there after 6 hours?

Exercise 1.5 Calculate the Volume of a Bacterial Cell

The average length of an *E. coli* cell is given as 2.0 μm, and its diameter as 0.5 μm. What would the volume of one bacterial cell be if it were a perfect cylinder? Use Python to do the calculation. Use variables for the parameters.