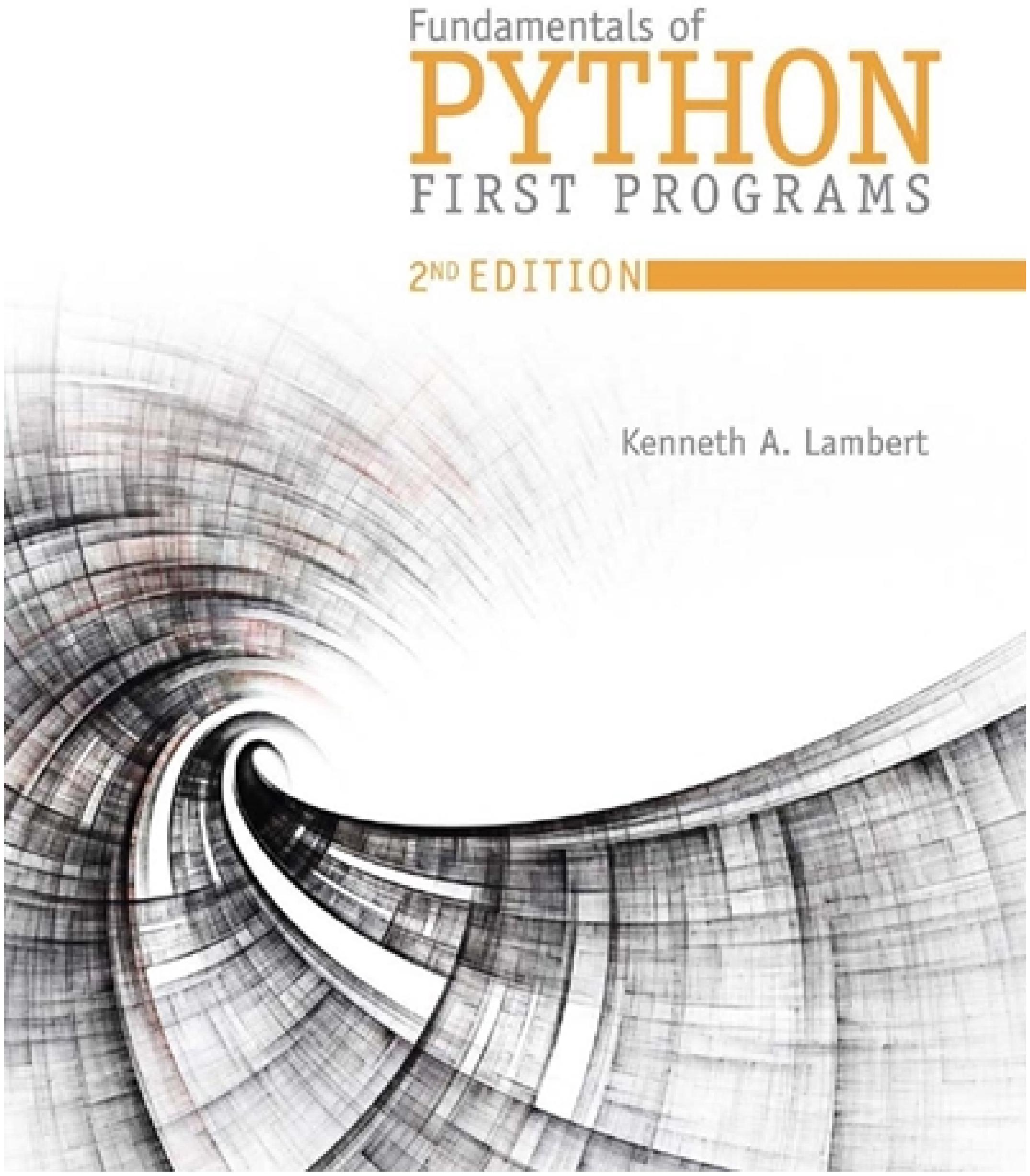
# Solutions for Fundamentals of Python First Programs 2nd Edition by Lambert



**Solutions**

# *Answers to End of Section Exercises for Chapter 2*

## *Exercises 2.1*

1. Analysis describes what a system does in terms of its input, outputs, and functions from a user's perspective. Design describes how a system accomplishes its tasks. Coding produces the software for the system. Testing examines whether or not the software does what it is supposed to do.
2. Analysis and design provide detailed blueprints for coding a system. Without these blueprints, it may be difficult to determine whether the system will do what it is supposed to do, and it may be difficult to minimize errors in the code and structure it in a way that eases maintenance.

## *Exercises 2.3*

1. a. **`"dogcat"`**
   b. **`"the dog chases the cat"`**
   c. **`"dogdogdogdog"`**
2. 
```
"Ken Lambert\nComputer Science\nWashington and Lee"
"""Ken Lambert
Computer Science
Washington and Lee"""
```
3. An apostrophe can be included in a string that is enclosed within double quotes.
4. When the Python interpreter encounters a newline character while printing a string, the cursor moves to the next line of output before the rest of the characters are displayed.
5. a, b, and c.
6. Program documentation can inform the reader of the author and purpose of a program and can also describe the strategy or method used in a particular line of code.

## *Exercises 2.4*

1. a. **`int`**
   b. **`float`**
   c. **`float`**
   d. **`int`**
   e. **`str`**
2. **`int`** is the type of integers or whole numbers, whereas **`float`** is the type of numbers that include a whole part (digits to the left of a decimal point) and a fractional part (digits to the right of the decimal point).
3. a. 3.5576e2
   b. 7.832e-3
   c. 4.3212e0
4. 36 and 38

## *Exercises 2.5*

1. a. 14
   b. 30
   c. 64
   d. 0
   e. 0.6666666666666663
   f. 1
2. a. 5.0
   b. 4
3. Pass the number to the **`round`** function.
4. Pass the number as an argument the **`str`** function and use the result returned with the string and the **`+`** operator.
5. **`x = x + 1`**

## *Exercises 2.6*

1. A function is a piece of code or an algorithm that can be called by name. A function's arguments are data that its caller passes to the function for its use when it is called.
2. 
```
import  math
print(math.pow(8, 2), math.pow(5, 4))
```
3. First, import the module. Then, call the **`dir`** function with the module's name as an argument.
4. First, import the module. Then, run the **`help`** function with the function's name, qualified by the module name, as an argument. Example: **`help(math.sqrt)`**

## *Answers to Review Questions for Chapter 2*

1. c
2. d
3. c, d
4. b
5. c
6. b
7. c
8. c
9. **`from math import sqrt, log`**
10. The **`dir`** function returns a list of the named resources (functions and variables) in its argument, which is a module. The **`help`** function displays all of the documentation for its argument, which can be a module or other resource.

# Chapter 2

# Software Development, Data Types, and Expressions

## At a Glance

## Instructor's Manual Table of Contents

- Overview
- Objectives
- Teaching Tips
- Quick Quizzes
- Class Discussion Topics
- Additional Projects
- Additional Resources
- Key Terms

## Lecture Notes

## Overview

Chapter 2 covers topics related to software development, data types, and expressions. Students first learn about the basic phases of the software development process. Next, they learn how to use strings, integer, and floating point numbers in Python. Simple as well as mixed-mode arithmetic expressions are covered. Students will also learn how to initialize and use variables with appropriate names. Students learn about modules, how to import functions from library modules, and how to call functions with arguments and use the returned values appropriately. Finally, they learn how to construct a simple Python program that performs inputs, calculations, and outputs, as well as how to use docstrings to document Python programs.

## Objectives

After completing this chapter, students will be able to:

- Describe the basic phases of software development: analysis, design, coding, and testing
- Use strings for the terminal input and output of text
- Use integers and floating point numbers in arithmetic operations
- Construct arithmetic expressions
- Initialize and use variables with appropriate names
- Import functions from library modules
- Call functions with arguments and use returned values appropriately
- Construct a simple Python program that performs inputs, calculations, and outputs
- Use docstrings to document Python programs

## Teaching Tips

### The Software Development Process

1. Explain that there are several approaches to *software development*, the *waterfall model* being one of them.

2. Use Figure 2.1 to describe the role of each of the phases of the waterfall model.

3. Stress that modern software development is usually *incremental* and *iterative*. Introduce the term *prototype*.

4. Note that programs rarely work as hoped the first time they are run. Use Figure 2.2 to stress that it is important to perform extensive and careful testing in all of the development phases.

5. Use Figure 2.3 to explain that the cost of developing software is not spread equally over the phases.

| | |
|---|---|
| ***Teaching Tip*** | For more information on the software development process, visit http://www.selectbs.com/analysis-and-design/what-is-a-software-development-process. |

## Strings, Assignment, and Comments

1. Use this section to introduce the use of strings for the output of text and the documentation of Python programs.

### Data Types

1. Introduce the terms *data type*, *literal*, and *numeric data type*, emphasizing the differences and relationship between them. Table 2.2 lists some Python data types.

### String Literals

1. Explain the term string literal and use real examples to illustrate how to create string literals in Python.

2. Introduce the term *empty string*, and be sure to distinguish the empty string from a string containing only white spaces.

| | |
|---|---|
| ***Teaching Tip*** | For more information about string literals, visit https://docs.python.org/2.0/ref/strings.html |

3. Show students how to create strings that span multiple-lines, and introduce them to the *newline character*, \n.

### Escape Sequences

1. Define the term *escape sequence*, and use Table 2.3 to introduce some of the most useful escape sequences in Python.

## String Concatenation

1. Using one or more examples, demonstrate how to perform string concatenation in Python using the + operator.

2. Explain that the * operator allows you to build a string by repeating another string a given number of times.

## Variables and the Assignment Statement

1. Remind students the meaning of the term *variable*.

2. Explain the naming rules that apply to variables in Python. Provide several examples to illustrate each of these rules.

3. Introduce the term *symbolic constant*, and note that programmers usually use all uppercase letters to name symbolic constants.

4. Use one or more examples to show how to write an *assignment statement* in Python. Introduce the term *variable reference* and explain the difference between *defining* or *initializing* a variable and variable references.

5. Explain the purposes variables have in a program. Be sure to explain the term *abstraction* in this context.

## Program Comments and Docstrings

1. Explain the purpose and importance of *program comments*.

2. Provide examples of how to include *docstrings* and *end-of-line* comments in Python. Stress that using these types of comments appropriately is important.

3. Review the guidelines for creating good documentation of code. You can use the list on Page 46 of the text as a guide.

| | |
|---|---|
| ***Teaching Tip*** | The pydoc module can be used to display information about a Python object, including its docstring. For more information, visit http://epydoc.sourceforge.net/docstrings.html. |

## Quick Quiz 1

1. In which phase of the waterfall model does the programmer determine how the program will do its task?
   a. analysis
   b. design
   c. implementation
   d. integration
   Answer: B

2. What is another name for a rough draft or skeletal version of a program?
   Answer: prototype

3. True or False: In programming, a data type consists of a set of values and a set of operations that can be performed on those values.
   Answer: True

4. The newline character \n is called which of the following?
   a. escape sequence
   b. backspace
   c. horizontal tab
   d. print function
   Answer: A

5. Variables receive their initial values and can be reset to new values with which of the following?
   a. variable reference
   b. program comment
   c. assignment statement
   d. escape sequence
   Answer: C

## Numeric Data Types and Character Sets

1. Explain that the following section provides a brief overview of numeric data types and their cousins, character sets.

### Integers

1. Explain how integer literals are written in Python.

### Floating-Point Numbers

1. Explain that real numbers have infinite precision, and point out that this is not possible on a computer due to memory limits.

2. Use Table 2.4 to introduce the use of *floating-point* numbers in Python. Explain that these numbers can be written using *decimal notation* or *scientific notation*.

| | |
|---|---|
| ***Teaching Tip*** | Complex numbers are also supported in Python. For more information, visit https://docs.python.org/3.7/library/stdtypes.html |

### Character Sets

1. Explain that in Python, character literals look just like string literals and are of the string type. Point out that character literals belong to several different *character sets*, among them the *ASCII set* and the *Unicode set*.

2. Provide a brief overview of the ASCII character set, using Table 2.5 as a guide.

| | |
|---|---|
| ***Teaching Tip*** | You can find more information about Python's Unicode support at https://docs.python.org/3/howto/unicode.html |

3. Use a few examples to show how to convert characters to and from ASCII using the `ord` and `chr` functions.

## Expressions

1. Explain that *expressions* provide an easy way to perform operations on data values to produce other values.

2. Note that when entered at the Python shell prompt, an expression's operands are evaluated first. The operator is then applied to these values to compute the final value of the expression.

### Arithmetic Expressions

1. Introduce the term *arithmetic expression*. Use Table 2.6 to describe the arithmetic operators available in Python.

2. Explain the difference between binary operators and unary operators. Give examples of each type of operator.

3. Briefly list the *precedence rules* that apply to arithmetic operators, using the list on Page 50 as a guide. Be sure to explain the meaning of the terms *left associative* and *right associative*, and to point out that you can use parentheses to change the order of evaluation in an arithmetic expression.

4. Use Table 2.7 to show a few examples of how arithmetic expressions are evaluated. Introduce the term *semantic error* and clearly explain the difference between a semantic error and a syntax error.

5. Stress that when both operands of an expression are of the same numeric type, the resulting value is of that type; when each operand is of a different type, the resulting value is of the more general type.

6. Explain how the backslash character \ can be used to break an expression onto multiple lines.

### Mixed-Mode Arithmetic and Type Conversions

1. Use a few examples to show how *mixed-mode arithmetic* can be problematic. Show how to use a *type conversion function* (see Table 2.8) to solve this problem.

2. Stress that the `int` function converts a `float` to an `int` by truncation, not by rounding. Show how to use the `round` function in cases when rounding is desirable.

3. Use one or more examples to show that type conversion also occurs in the construction of strings from numbers and other strings. Explain how to use the `str` function to solve this problem.

4. Explain that Python is a *strongly typed programming language*.

## Using Functions and Modules

1. Explain that Python includes many useful functions, which are organized in libraries of code called *modules*.

| | |
|---|---|
| ***Teaching Tip*** | Python's global module index is available at https://docs.python.org/3/py-modindex.html . |

## Calling Functions: Arguments and Return Values

1. Introduce the terms *function*, *argument/parameter* (*optional* and *required*), and *returning a value*.

2. Define the concept of a *default behavior* of a function, and explain how this may be changed by calling the function using optional arguments.

3. Use an example to show how to obtain more information about a function by using `help`.

## The `math` Module

1. Show how to use functions in the `math` module, both by importing the whole module and by importing individual resources.

2. Point out that if you are going to use only a couple of a module's resources frequently, you can avoid the use of the qualifier with each reference by importing the individual resources.

## The Main Module

1. Explain that, like any module, the *main module* can be imported. Show how this is equivalent to importing a Python script as a module.

2. Use the example provided in the book to show how to import the main module created in the case study of this chapter.

## Program Format and Structure

1. Provide some guidance on how a typical Python program should look. Use the bullet points on Page 57 as a guide.

## Running a Script from a Terminal Command Prompt

1. Use Figures 2.5 through 2.7 to show how to run a script from a terminal command prompt.

2. Note that Python installations enable you to launch Python scripts by double-clicking the files from the OS's file browser. Explain what the fly-by-window problem is, and how to solve it: add an input statement at the end of the script that pauses until the user presses the ENTER or RETURN key.

## Quick Quiz 2

1. True or False: In the 1960s, the original ASCII set encoded each keyboard character and several control characters using the integers from 0 through 255.
   Answer: False

2. True or False: The precedence rules you learned in algebra apply during the evaluation of arithmetic expressions in Python.
   Answer: True

3. Which of the following provides an easy way to perform operations on data values to produce other data values?
   a. strings
   b. expressions
   c. conversions
   d. modules
   Answer: b

4. What is a module?
   Answer: Python includes many useful functions, which are organized in libraries of code called modules.

5. Which of the following is a chunk of code that can be called by name to perform a task?
   a. parameter
   b. argument
   c. function
   d. script
   Answer: C

6. True or False: Arguments are also known as literals.
   Answer: False

## Class Discussion Topics

1. Are your students familiar with other software development models? If so, ask them to discuss their similarities and differences with the waterfall model. Which one do they like best?

2. Have your students used functions in other languages?

## Additional Projects

1. Ask students to do some research and find out what other escape sequences are available in Python. They should compile a list of each of these with a short description of their functions.

2. Python provides other useful modules besides the `math` module. Ask students to do some research and find one or two other modules that they think will be useful in future projects. They should provide a brief description of each module, and include a list of two to five functions that belong to each module.

## Additional Resources

1. String Literals:
   https://docs.python.org/2.0/ref/strings.html

2. Python Docstrings:
   http://epydoc.sourceforge.net/docstrings.html

3. Unicode HOWTO:
   https://docs.python.org/3.7/howto/unicode.html

4. Global Module Index:
   https://docs.python.org/3/py-modindex.html#cap-g

5. Module tutorial:
   http://docs.python.org/tutorial/modules.html

## Key Terms

- See Glossary for definitions of Key Terms