Chapter 1 Summary

Comments are indicated by a hash sign # (also known as the pound or number sign). Text to the right of the hash sign is ignored. (But, hash loses its special meaning if it is part of a string, i.e., enclosed in quotes.)

print(): is used to produce output. The optional arguments sep and end control what appears between values and how a line is terminated, respectively.

Code may contain syntactic bugs (errors in grammar) or semantic bugs (error in meaning). Generally, Python will only raise, or throw, an exception when the interpreter encounters a syntactic bug.

help(): provides help. It can be used interactively or with a specific value specified as its argument.

Chapter 2 Summary

Literals are data that are entered directly into the code.

Data has type, for example, int (integer), float (real number with finite precision), and str (string, a collection of characters).

type() returns the type of its argument.

A literal float has a decimal point or contains the power-of-ten exponent indicator e, e.g., 1.1e2, 1le1, and 110.0 are equivalent floats.

A literal int does not contain a decimal point (nor the power-of-ten exponent indicator e).

A literal str is enclosed in either a matching pair of double or single quotes. Quotation marks can be repeated three times at the beginning and end of a string, in which case the string can span multiple lines.

\n is used in a string to indicate the newline character.

Characters in a string may be escaped by preceding the character with a backslash. This causes the character to have a different meaning than usual, e.g., a backslash can be placed before a quotation mark in a string to prevent it from indicating the termination of the string; the quotation mark is then treated as part of the string.

An expression produces data. The simplest expression is a literal.

There are numerous arithmetic operators. Binary operators (which require two operands) include:

+, - ⇔ addition and subtraction
/, // ⇔ float and floor division
% ⇔ modulo (or remainder)
* ⇔ multiplication
** ⇔ exponentiation

Float division (/) yields a float regardless of the types of the operands. For all other arithmetic operators, only when both operands are integers does the operation yield an integer. Said another way, when either or both operands are floats, the arithmetic operation yields a float.

Floor division (//) yields a whole number (which may be either an int or a float, depending on the operands). The result is the largest whole number that does not exceed the value that would be obtained with float division.

Modulo (%) yields the remainder (which may be either an int or a float, depending on
the operands) after floor division has been performed.

\[
divmod(a, b): \text{ equivalent to } \Rightarrow a \div b, a \mod b.
\]

Evaluation of expressions containing multiple operations follows the rules of precedence to determine the order of operation. Exponentiation has highest precedence; multiplication, integer and float division, and modulo have equal precedence which is below that of exponentiation; addition and subtraction have equal precedence which is below that of multiplication, division, and modulo.

Operations of equal precedence are evaluated left to right except for exponentiation operations which are evaluated right to left.

The negative sign (−) and positive sign (+) can be used as unary operators, e.g., −x changes the sign of x. The expression +x is valid but has no effect on the value of x.

Parentheses can be used to change the order or precedence.

Statements are complete commands.

The equal sign = is the assignment operator. The value of the expression on the right side of the equal sign is assigned to the lvalue on the left side of the equal sign.

An lvalue is a general name for something that can appear to the left side of the assignment operator. It is typically a variable that must be a valid identifier.

A variable can also be thought of as a name within a namespace. A namespace maps names to their corresponding values (or objects).

Valid identifiers start with a letter or underscore followed by any number of letters, digits, and underscores.

There are 33 keywords that cannot be used as identifiers.

Augmented operators can be used as shorthand for assignment statements in which an identifier appears in an arithmetic operation on the left side of the equal sign and on the right side of the assignment operator. For example, \(x += 1\) is equivalent to \(x = x + 1\).

Simultaneous assignment occurs when multiple comma-separated expressions appear to the right side of an equal sign and an equal number of comma-separated lvalues appear to the left side of the equal sign.

A statement can span multiple lines if it is enclosed in parentheses or if the newline character at the end of each line of the statement (other than the last) is escaped using a backslash.

Multiple statements can appear on one line if they are separated by semicolons.

A magic number is a numeric literal whose underlying meaning is difficult to understand from the code itself. Named constants should be used in the place of magic numbers.

### Chapter 3 Summary

- **input()**: Prompts the user for input with its string argument and returns the string the user enters.
- **int()**: Returns the integer form of its argument.
- **float()**: Returns the float form of its argument.
- **eval()**: Returns the result of evaluating its
string argument as any Python expression, including arithmetic and numerical expressions.

Functions such as the four listed above can be nested. Thus, for example, `float(input())` can be used to obtain input in string form which is then converted to a float value.

Chapter 4 Summary

The template for defining a function is:

```python
def <function_name>(<params>):
    <body>
```

where the function name is a valid identifier, the formal parameters are a comma-separated list of variables, and the body consists of an arbitrary number of statements that are indented to the same level.

A function is called/invoked by writing the function name followed by parentheses that enclose the actual parameters which are also known as the arguments.

A function that does not explicitly return a value is said to be a void function. Void functions return `None`.

A variable defined as a formal parameter or defined in the body of the function is not defined outside the function, i.e., the variables only have local scope. Variables accessible throughout a program are said to have global scope.

Generally, a function should obtain data via its parameters and return data via a return statement.

`print()` does not return anything (it generates output) and the return statement does not print anything (it serves to return a value).

If comma-separated expressions are given as part of the return statement, the values of these expressions are returned as a collection of values that can be used with simultaneous assignment. The values are in a tuple as described in Chap. 6.

Function definitions may be nested inside other functions. When this is done, the inner function is only usable within the body of the function in which it is defined. Typically such nesting is not used.

The scoping rules for functions are the same as for variables. Anything defined inside a function, including other functions, is local to that function. Variables and functions defined external to functions have global scope and are visible “everywhere.”

Often programs are organized completely in terms of functions. A function named `main()` is, by convention, often the first function called at the start of a program (but after defining all the functions). The statements in `main()` provide the other function calls that are necessary to complete the program. Thus, the program consists of a number of function definitions and the last line of the program file is a call to `main()`.

An optional parameter it created by assigning a default value to the formal parameter in the header of the function definition.