**1.8 Glossary**

**problem solving:** The process of formulating a problem, finding a solution, and expressing it.

**high-level language:** A programming language like Python that is designed to be easy for humans to read and write.

**low-level language:** A programming language that is designed to be easy for a computer to run; also called “machine language” or “assembly language”.

**portability:** A property of a program that can run on more than one kind of computer.

**interpreter:** A program that reads another program and executes it

**prompt:** Characters displayed by the interpreter to indicate that it is ready to take input from the user.

**program:** A set of instructions that specifies a computation.

**print statement:** An instruction that causes the Python interpreter to display a value on the screen.

**operator:** A special symbol that represents a simple computation like addition, multiplication, or string concatenation.

**value:** One of the basic units of data, like a number or string, that a program manipulates.

**type:** A category of values. The types we have seen so far are integers (type int), floating point numbers (type float), and strings (type str).

**integer:** A type that represents whole numbers.

**floating-point:** A type that represents numbers with fractional parts.

**string:** A type that represents sequences of characters.

**natural language:** Any one of the languages that people speak that evolved naturally.

**formal language:** Any one of the languages that people have designed for specific purposes, such as representing mathematical ideas or computer programs; all programming languages are formal languages.

**token:** One of the basic elements of the syntactic structure of a program, analogous to a word in a natural language.

**syntax:** The rules that govern the structure of a program.

**parse:** To examine a program and analyze the syntactic structure.

**bug:** An error in a program.

**debugging:** The process of finding and correcting bugs.

**2.9 Glossary**

**variable:** A name that refers to a value.

**assignment:** A statement that assigns a value to a variable.

**state diagram:** A graphical representation of a set of variables and the values they refer to.

**keyword:** A reserved word that is used to parse a program; you cannot use keywords like if, def, and while as variable names.

**operand:** One of the values on which an operator operates.

**expression:** A combination of variables, operators, and values that represents a single result.

**evaluate:** Tosimplifyanexpressionbyperformingtheoperationsinordertoyieldasingle value.

**statement:** A section of code that represents a command or action. So far, the statements we have seen are assignments and print statements.

**execute:** To run a statement and do what it says.

**interactive mode:** A way of using the Python interpreter by typing code at the prompt.

**script mode:** A way of using the Python interpreter to read code from a script and run it.

**script:** A program stored in a file.

**order of operations:** Rules governing the order in which expressions involving multiple operators and operands are evaluated.

**concatenate:** To join two operands end-to-end.

**comment:** Information in a program that is meant for other programmers (or anyone reading the source code) and has no effect on the execution of the program.

**syntax error:** An error in a program that makes it impossible to parse (and therefore impossible to interpret).

**exception:** An error that is detected while the program is running.

**semantics:** The meaning of a program.

**semantic error:** An error in a program that makes it do something other than what the programmer intended.

* 1. **Glossary**

**function:** A named sequence of statements that performs some useful operation. Functions may or may not take arguments and may or may not produce a result.

**function definition:** A statement that creates a new function, specifying its name, parameters, and the statements it contains.

**function object:** A value created by a function definition. The name of the function is a variable that refers to a function object.

**header:** The first line of a function definition.

**body:** The sequence of statements inside a function definition.

**parameter:** A name used inside a function to refer to the value passed as an argument.

**function call:** A statement that runs a function. It consists of the function name followed by an argument list in parentheses.

**argument:** A value provided to a function when the function is called. This value is as- signed to the corresponding parameter in the function.

**local variable:** A variable defined inside a function. A local variable can only be used inside its function.

**return value:** The result of a function. If a function call is used as an expression, the return value is the value of the expression.

**fruitful function:** A function that returns a value.

**void function:** A function that always returns None.

None**:** A special value returned by void functions.

**module:** A file that contains a collection of related functions and other definitions.

**import statement:** A statement that reads a module file and creates a module object.

**module object:** A value created by an import statement that provides access to the values defined in a module.

**dot notation:** The syntax for calling a function in another module by specifying the module name followed by a dot (period) and the function name.

**composition:** Using an expression as part of a larger expression, or a statement as part of a larger statement.

**flow of execution:** The order statements run in.

**stack diagram:** A graphical representation of a stack of functions, their variables, and the

values they refer to.

**frame:** A box in a stack diagram that represents a function call. It contains the local variables and parameters of the function.

**traceback:** A list of the functions that are executing, printed when an exception occurs.

**5.13 Glossary**

**floor division:** An operator, denoted //, that divides two numbers and rounds down (to- ward zero) to an integer.

**modulus operator:** An operator, denoted with a percent sign (%), that works on integers and returns the remainder when one number is divided by another.

**boolean expression:** An expression whose value is either True or False.

**relational operator:** One of the operators that compares its operands: ==, !=, >, <, >=, and <=.

**logical operator:** One of the operators that combines boolean expressions: and, or, and

not.

**conditional statement:** A statement that controls the flow of execution depending on some condition.

**condition:** The boolean expression in a conditional statement that determines which branch runs.

**compound statement:** A statement that consists of a header and a body. The header ends with a colon (:). The body is indented relative to the header.

**branch:** One of the alternative sequences of statements in a conditional statement.

**chained conditional:** A conditional statement with a series of alternative branches.

**Nested conditional:** A conditional statement that appears in one of the branches of another conditional statement.

**return statement:** A statement that causes a function to end immediately and return to the caller.

**recursion:** The process of calling the function that is currently executing.

**base case:** A conditional branch in a recursive function that does not make a recursive call.

**infinite recursion:** A recursion that doesn’t have a base case, or never reaches it. Eventually, an infinite recursion causes a runtime error.

**6.10 Glossary**

**temporary variable:** A variable used to store an intermediate value in a complex calculation.

**dead code:** Part of a program that can never run, often because it appears after a return statement.

**Incremental development:** A program development plan intended to avoid debugging by adding and testing only a small amount of code at a time.

**scaffolding:** Code that is used during program development but is not part of the final version.

**guardian:** A programming pattern that uses a conditional statement to check for and handle circumstances that might cause an error.