**Alphabetic glossary compiled from *Think Python* by Allen B Downey**

*absolute path:* A path that starts from the topmost directory in the file system. (Chapter 14)

*accumulator:* A variable used in a loop to add up or accumulate a result. (Chapter 10)

*algorithm:* A general process for solving a category of problems. (Chapter 7)

*aliasing:* A circumstance where two or more variables refer to the same object. (Chapter 10)

*analysis of algorithms:* A way to compare algorithms in terms of their run time and/or space requirements. (Chapter 20)

*argument:* A value provided to a function when the function is called. This value is assigned to the corresponding parameter in the function. (Chapter 3)

*assert statement:* A statement that checks a condition and raises an exception if it fails. (Chapter 16)

*assignment:* A statement that assigns a value to a variable. (Chapter 2)

*attribute:*  One of the named values associated with an object. (Chapter 15)

*augmented assignment:* A statement that updates the value of a variable using an operator like +=. (Chapter 10)

*base case:* A conditional branch in a recursive function that does not make a recursive call. (Chapter 5)

*benchmarking:* The process of choosing between data structures by implementing alternatives and testing them on a sample of the possible inputs. (Chapter 13)

*Big-Oh notation:* Notation for representing an order of growth; for example, $O(n)$ represents the set of functions that grow linearly. (Chapter 20)

*body:* The sequence of statements inside a function definition. (Chapter 3)

*Boolean expression:* An expression whose value is either True or False. (Chapter 5)

*branch:* One of the alternative sequences of statements in a conditional statement. (Chapter 5)

*bug:* An error in a program. (Chapter 1)

*bytes object*: An object similar to a string. (Chapter 14)

*call graph:* A diagram that shows every frame created during the execution of a program, with an arrow from each caller to each callee. (Chapter 11)

*catch:* To prevent an exception from terminating a program using the try and except statements. (Chapter 14)

*chained conditional:* A conditional statement with a series of alternative branches. (Chapter 5)

*child class:* A new class created by inheriting from an existing class; also called a ``subclass''. (Chapter 18)

*class attribute:* An attribute associated with a class object. Class attributes are defined inside a class definition but outside any method. (Chapter 18)

*class diagram:* A diagram that shows the classes in a program and the relationships between them. (Chapter 18)

*class object:* An object that contains information about a programmer-defined type. The class object can be used to create instances of the type. (Chapter 15)

*class:* A programmer-defined type. A class definition creates a new class object. (Chapter 15)

*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. (Chapter 2)

*composition:* Using an expression as part of a larger expression, or a statement as part of a larger statement. (Chapter 3)

*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. (Chapter 5)

*concatenate:* To join two operands end-to-end. (Chapter 2)

*condition:* The Boolean expression in a conditional statement that determines which branch runs. (Chapter 5)

*conditional expression:* An expression that has one of two values, depending on a condition. (Chapter 19)

*conditional statement:* A statement that controls the flow of execution depending on some condition. (Chapter 5)

*counter:* A variable used to count something, usually initialized to zero and then incremented. (Chapter 8)

*crossover point:* The problem size where two algorithms require the same run time or space. (Chapter 20)

*data encapsulation:* A program development plan that involves a prototype using global variables and a final version that makes the global variables into instance attributes. (Chapter 18)

*data structure*: A collection of related values, often organized in lists, dictionaries, tuples, etc. (Chapter 12)

*database:* A file whose contents are organized like a dictionary with keys that correspond to values. (Chapter 14)

*dead code:* Part of a program that can never run, often because it appears after a {return} statement. (Chapter 6)

*debugging:* The process of finding and correcting bugs. (Chapter 1)

*declaration:* A statement like {global} that tells the interpreter something about a variable. (Chapter 11)

*decrement:* An update that decreases the value of a variable. (Chapter 7)

*deep copy:* To copy the contents of an object as well as any embedded objects, and any objects embedded in them, and so on; implemented by the deepcopy function in the copy module. (Chapter 15)

*default value:* The value given to an optional parameter if no argument is provided. (Chapter 13)

*delimiter:* A character or string used to indicate where a string should be split. (Chapter 10)

*dependency:* A relationship between two classes where instances of one class use instances of the other class, but do not store them as attributes. (Chapter 18)

*designed development:* A development plan that involves high-level insight into the problem and more planning than incremental development or prototype development. (Chapter 16)

*deterministic:* Pertaining to a program that does the same thing each time it runs, given the same inputs. (Chapter 13)

*development plan:* A process for writing programs. (Chapter 4)

*dictionary:* A mapping from keys to their corresponding values. (Chapter 11)

*directory:* A named collection of files, also called a folder. (Chapter 14)

*docstring:* A string that appears at the top of a function definition to document the function's interface. (Chapter 4)

*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. (Chapter 3)

*element:* One of the values in a list (or other sequence), also called items. (Chapter 10)

*embedded object:* An object that is stored as an attribute of another object. (Chapter 15)

*empty string:* A string with no characters and length 0, represented by two quotation marks. (Chapter 8)

*encapsulation:* The process of transforming a sequence of statements into a function definition. (Chapter 4)

*encode:* To represent one set of values using another set of values by constructing a mapping between them. (Chapter 18)

*equivalent:* Having the same value. (Chapter 10)

*evaluate:* To simplify an expression by performing the operations in order to yield a single value. (Chapter 2)

*exception:* An error that is detected while the program is running. (Chapter 2)

*execute:* To run a statement and do what it says. (Chapter 2)

*factory:* A function, usually passed as a parameter, used to create objects. (Chapter 19)

*file object:* A value that represents an open file. (Chapter 9)

*filter:* A processing pattern that traverses a list and selects the elements that satisfy some criterion. (Chapter 10)

*flag:* A Boolean variable used to indicate whether a condition is true. (Chapter 11)

*floating-point:* A type that represents numbers with fractional parts. (Chapter 1)

*floor division:* An operator, denoted //, that divides two numbers and rounds down (toward negative infinity) to an integer. (Chapter 5)

*flow of execution:* The order statements run in. (Chapter 3)

*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. (Chapter 1)

*format operator:* An operator, %, that takes a format string and a tuple and generates a string that includes the elements of the tuple formatted as specified by the format string. (Chapter 14)

*format sequence:* A sequence of characters in a format string, like { \%d}, that specifies how a value should be formatted. (Chapter 14)

*format string:* A string, used with the format operator, that contains format sequences. (Chapter 14)

*frame:* A box in a stack diagram that represents a function call. It contains the local variables and parameters of the function. (Chapter 3)

*fruitful function:* A function that returns a value. (Chapter 3)

*function call:* A statement that runs a function. It consists of the function name followed by an argument list in parentheses. (Chapter 3)

*function definition:* A statement that creates a new function, specifying its name, parameters, and the statements it contains. (Chapter 3)

*function object:* A value created by a function definition. The name of the function is a variable that refers to a function object. (Chapter 3)

*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. (Chapter 3)

*functional programming style:* A style of program design in which the majority of functions are pure. (Chapter 16)

*gather:* An operation that collects multiple arguments into a tuple. (Chapter 12)

*generalization:* The process of replacing something unnecessarily specific (like a number) with something appropriately general (like a variable or parameter). (Chapter 4)

*generator expression:* An expression with a {for} loop in parentheses that yields a generator object. (Chapter 19)

*global statement:* A statement that declares a variable name global. (Chapter 11)

*global variable:* A variable defined outside a function. Global variables can be accessed from any function. (Chapter 11)

*guardian:* A programming pattern that uses a conditional statement to check for and handle circumstances that might cause an error. (Chapter 6)

*HAS-A relationship:* A relationship between two classes where instances of one class contain references to instances of the other. (Chapter 18)

*hash function:* A function used by a hashtable to compute the location for a key. (Chapter 11)

*hashable:* A type that has a hash function. Immutable types like integers, floats and strings are hashable; mutable types like lists and dictionaries are not. (Chapter 11)

*hashtable:* A data structure that represents a collection of key-value pairs and performs search in constant time. (Chapter 20)

*hashtable:* The algorithm used to implement Python dictionaries. (Chapter 11)

*header:* The first line of a function definition. (Chapter 3)

*high-level language:* A programming language like Python that is designed to be easy for humans to read and write. (Chapter 1)

*identical:* Being the same object (which implies equivalence). (Chapter 10)

*immutable:* The property of a sequence whose items cannot be changed. (Chapter 8)

*implementation:* A way of performing a computation. (Chapter 11)

*import statement:* A statement that reads a module file and creates a module object. (Chapter 3)

*increment:* An update that increases the value of a variable (often by one). (Chapter 7)

*incremental development:* A program development plan intended to avoid debugging by adding and testing only a small amount of code at a time. (Chapter 6)

*index:* An integer value used to select an item in a sequence, such as a character in a string. In Python indices start from 0. (Chapter 8)

*infinite loop:* A loop in which the terminating condition is never satisfied. (Chapter 7)

*infinite recursion:* A recursion that doesn't have a base case, or never reaches it. Eventually, an infinite recursion causes a runtime error. (Chapter 5)

*information hiding:* The principle that the interface provided by an object should not depend on its implementation, in particular the representation of its attributes. (Chapter 17)

*inheritance:* The ability to define a new class that is a modified version of a previously defined class. (Chapter 18)

*initialization:* An assignment that gives an initial value to a variable that will be updated. (Chapter 7)

*instance attribute:* An attribute associated with an instance of a class. (Chapter 18)

*instance:* An object that belongs to a class. (Chapter 15)

*instantiate:* To create a new object. (Chapter 15)

*integer:* A type that represents whole numbers. (Chapter 1)

*interactive mode:* A way of using the Python interpreter by typing code at the prompt. (Chapter 2)

*interface:* A description of how to use a function, including the name and descriptions of the arguments and return value. (Chapter 4)

*interpreter:* A program that reads another program and executes it. (Chapter 1)

*invariant:* A condition that should always be true during the execution of a program. (Chapter 16)

*invocation:* A statement that calls a method. (Chapter 8)

*IS-A relationship:* A relationship between a child class and its parent class. (Chapter 18)

*item:* In a dictionary, another name for a key-value pair. (Chapter 11)

*item:* One of the values in a sequence. (Chapter 8)

*iteration:* Repeated execution of a set of statements using either a recursive function call or a loop. (Chapter 7)

*iterator:* An object that can iterate through a sequence, but which does not provide list operators and methods. (Chapter 12)

*key-value pair:* The representation of the mapping from a key to a value. (Chapter 11)

*key:* An object that appears in a dictionary as the first part of a key-value pair. (Chapter 11)

*keyword argument:* An argument that includes the name of the parameter as a ``keyword''. (Chapter 3)

*keyword:* A reserved word that is used to parse a program; you cannot use keywords like if, def, and while as variable names. (Chapter 2)

*leading term:* In a polynomial, the term with the highest exponent. (Chapter 20)

*linear:* An algorithm whose run time is proportional to problem size, at least for large problem sizes. (Chapter 20)

*list comprehension:* An expression with a {for} loop in square brackets that yields a new list. (Chapter 19)

*list:* A sequence of values. (Chapter 10)

*local variable:* A variable defined inside a function. A local variable can only be used inside its function. (Chapter 3)

*logical operator:* One of the operators that combines Boolean expressions: and, or, and not. (Chapter 5)

*lookup:* A dictionary operation that takes a key and finds the corresponding value. (Chapter 11)

*loop:* A part of a program that can run repeatedly. (Chapter 4)

*low-level language:* A programming language that is designed to be easy for a computer to run; also called ``machine language'' or ``assembly language''. (Chapter 1)

*machine model:* A simplified representation of a computer used to describe algorithms. (Chapter 20)

*map:* A processing pattern that traverses a sequence and performs an operation on each element. (Chapter 10)

*mapping:* A relationship in which each element of one set corresponds to an element of another set. (Chapter 11)

*memo:* A computed value stored to avoid unnecessary future computation. (Chapter 11)

*method:* A function that is associated with an object and called using dot notation. (Chapter 4)

*method:* A function that is defined inside a class definition and is invoked on instances of that class. (Chapter 17)

*modifier:* A function that changes one or more of the objects it receives as arguments. Most modifiers are void; that is, they return {None}. (Chapter 16)

*module object:* A value created by an {import} statement that provides access to the values defined in a module. (Chapter 3)

*module:* A file that contains a collection of related functions and other definitions. (Chapter 3)

*modulus operator:* An operator, denoted with a percent sign %, that works on integers and returns the remainder when one number is divided by another. (Chapter 5)

*multiplicity:* A notation in a class diagram that shows, for a HAS-A relationship, how many references there are to instances of another class. (Chapter 18)

*multiset:* A mathematical entity that represents a mapping between the elements of a set and the number of times they appear. (Chapter 19)

*natural language:* Any one of the languages that people speak that evolved naturally. (Chapter 1)

*nested conditional:* A conditional statement that appears in one of the branches of another conditional statement. (Chapter 5)

*nested list:* A list that is an element of another list. (Chapter 10)

None: A special value returned by void functions. (Chapter 3)

*object diagram:* A diagram that shows objects, their attributes, and the values of the attributes. (Chapter 15)

*object-oriented language:* A language that provides features, such as programmer-defined types and methods, that facilitate object-oriented programming. (Chapter 17)

*object-oriented programming:* A style of programming in which data and the operations that manipulate it are organized into classes and methods. (Chapter 17)

*object:* Something a variable can refer to. An object has a type and a value. (Chapter 10)

*object:* Something a variable can refer to. For now, you can use ``object'' and ``value'' interchangeably. (Chapter 8)

*operand:* One of the values on which an operator operates. expression: A combination of variables, operators, and values that represents a single result. (Chapter 2)

*operator overloading:* Changing the behavior of an operator like + so it works with a programmer-defined type. (Chapter 17)

*operator:* A special symbol that represents a simple computation like addition, multiplication, or string concatenation. (Chapter 1)

*optional argument:* A function or method argument that is not required. (Chapter 8)

*order of growth:* A set of functions that all grow in a way considered equivalent for purposes of analysis of algorithms. For example, all functions that grow linearly belong to the same order of growth. (Chapter 20)

*order of operations:* Rules governing the order in which expressions involving multiple operators and operands are evaluated. (Chapter 2)

*override:* To replace a default value with an argument. (Chapter 13)

*parameter:* A name used inside a function to refer to the value passed as an argument. (Chapter 3)

*parent class:* The class from which a child class inherits. (Chapter 18)

*parse:* To examine a program and analyze the syntactic structure. (Chapter 1)

*path:* A string that identifies a file. (Chapter 14)

persistent: Pertaining to a program that runs indefinitely and keeps at least some of its data in permanent storage. (Chapter 14)

*pipe object:* An object that represents a running program, allowing a Python program to run commands and read the results. (Chapter 14)

*polymorphic:* Pertaining to a function that can work with more than one type. (Chapter 17)

*portability:* A property of a program that can run on more than one kind of computer. (Chapter 1)

*positional argument:* An argument that does not include a parameter name, so it is not a keyword argument. (Chapter 17)

*postcondition:* A requirement that should be satisfied by the function before it ends. (Chapter 4)

*precondition:* A requirement that should be satisfied by the caller before a function starts. (Chapter 4)

*print statement:* An instruction that causes the Python interpreter to display a value on the screen. (Chapter 1)

*problem solving:* The process of formulating a problem, finding a solution, and expressing it. (Chapter 1)

*program:* A set of instructions that specifies a computation. (Chapter 1)

*prompt:* Characters displayed by the interpreter to indicate that it is ready to take input from the user. (Chapter 1)

*prototype and patch:* A development plan that involves writing a rough draft of a program, testing, and correcting errors as they are found. (Chapter 16)

*pseudorandom:* Pertaining to a sequence of numbers that appears to be random, but is generated by a deterministic program. (Chapter 13)

*pure function:* A function that does not modify any of the objects it receives as arguments. Most pure functions are fruitful. (Chapter 16)

*quadratic:* An algorithm whose run time is proportional to n2, where n is a measure of problem size. (Chapter 20)

*raise statement:* A statement that (deliberately) raises an exception. (Chapter 11)

*reassignment:* Assigning a new value to a variable that already exists. (Chapter 7)

*recursion:* The process of calling the function that is currently executing. (Chapter 5)

*reduce:* A processing pattern that traverses a sequence and accumulates the elements into a single result. (Chapter 10)

*reduction to a previously solved problem:* A way of solving a problem by expressing it as an instance of a previously solved problem. (Chapter 9)

*refactoring:* The process of modifying a working program to improve function interfaces and other qualities of the code. (Chapter 4)

*reference:* The association between a variable and its value. (Chapter 10)

*relational operator:* One of the operators that compares its operands: ==, !=, >, <, >=, and <=. (Chapter 5)

*relative path:* A path that starts from the current directory. (Chapter 14)

*return statement:* A statement that causes a function to end immediately and return to the caller. (Chapter 5)

*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. (Chapter 3)

*reverse lookup:* A dictionary operation that takes a value and finds one or more keys that map to it. (Chapter 11)

*rubber duck debugging:* Debugging by explaining your problem to an inanimate object such as a rubber duck. Articulating the problem can help you solve it, even if the rubber duck doesn't know Python. (Chapter 13)

*scaffolding:* Code that is used during program development but is not part of the final version. (Chapter 6)

*scatter:* An operation that makes a sequence behave like multiple arguments. (Chapter 12)

*script mode:* A way of using the Python interpreter to read code from a script and run it. (Chapter 2)

*script:* A program stored in a file. (Chapter 2)

*search (1):* A pattern of traversal that stops when it finds what it is looking for. (Chapter 8)

*search (2):* The problem of locating an element of a collection (like a list or dictionary) or determining that it is not present. (Chapter 20)

*semantic error:* An error in a program that makes it do something other than what the programmer intended. (Chapter 2)

*semantics:* The meaning of a program. (Chapter 2)

*sequence:* An ordered collection of values where each value is identified by an integer index. (Chapter 8)

*shallow copy:* To copy the contents of an object, including any references to embedded objects; implemented by the copy function in the copy module. (Chapter 15)

*shape error:* An error caused because a value has the wrong shape; that is, the wrong type or size. (Chapter 12)

*shell:* A program that allows users to type commands and then executes them by starting other programs. (Chapter 14)

*singleton:* A list (or other sequence) with a single element. (Chapter 11)

*slice:* A part of a string specified by a range of indices. (Chapter 8)

*special case:* A test case that is atypical or non-obvious (and less likely to be handled correctly). (Chapter 9)

*stack diagram:* A graphical representation of a stack of functions, their variables, and the values they refer to. (Chapter 3)

*state diagram:* A graphical representation of a set of variables and the values they refer to. (Chapter 2)

*statement:* A section of code that represents a command or action. So far, the statements we have seen are assignments and print statements. (Chapter 2)

*string:* A type that represents sequences of characters. (Chapter 1)

*subject:* The object a method is invoked on. (Chapter 17)

*syntax error:* An error in a program that makes it impossible to parse (and therefore impossible to interpret). (Chapter 2)

*syntax:* The rules that govern the structure of a program. (Chapter 1)

*temporary variable:* A variable used to store an intermediate value in a complex calculation. (Chapter 6)

*text file:* A sequence of characters stored in permanent storage like a hard drive. (Chapter 14)

*token:* One of the basic elements of the syntactic structure of a program, analogous to a word in a natural language. (Chapter 1)

*traceback:* A list of the functions that are executing, printed when an exception occurs. (Chapter 3)

*traverse:* To iterate through the items in a sequence, performing a similar operation on each. (Chapter 8)

*tuple assignment:* An assignment with a sequence on the right side and a tuple of variables on the left. The right side is evaluated and then its elements are assigned to the variables on the left. (Chapter 12)

*tuple:* An immutable sequence of elements. (Chapter 12)

*type-based dispatch:* A programming pattern that checks the type of an operand and invokes different functions for different types. (Chapter 17)

*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). (Chapter 1)

*update:* An assignment where the new value of the variable depends on the old. (Chapter 7)

*value (1):* One of the basic units of data, like a number or string, that a program manipulates. (Chapter 1)

*value (2):* An object that appears in a dictionary as the second part of a key-value pair. This is more specific than our previous use of the word ``value''. (Chapter 11)

*variable:* A name that refers to a value. (Chapter 2)

*veneer:* A method or function that provides a different interface to another function without doing much computation. (Chapter 18)

*void function:* A function that always returns None. (Chapter 3)

*worst case:* The input that makes a given algorithm run slowest (or require the most space). (Chapter 20)

*zip object:* The result of calling a built-in function {zip}; an object that iterates through a sequence of tuples. (Chapter 12)