

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

<i>absolute path:</i>	A path that starts from the topmost directory in the file system. (Chapter 14)
<i>accumulator:</i>	A variable used in a loop to add up or accumulate a result. (Chapter 10)
<i>algorithm:</i>	A general process for solving a category of problems. (Chapter 7)
<i>aliasing:</i>	A circumstance where two or more variables refer to the same object. (Chapter 10)
<i>analysis of algorithms:</i>	A way to compare algorithms in terms of their run time and/or space requirements. (Chapter 20)
<i>argument:</i>	A value provided to a function when the function is called. This value is assigned to the corresponding parameter in the function. (Chapter 3)
<i>assert statement:</i>	A statement that checks a condition and raises an exception if it fails. (Chapter 16)
<i>assignment:</i>	A statement that assigns a value to a variable. (Chapter 2)
<i>attribute:</i>	One of the named values associated with an object. (Chapter 15)
<i>augmented assignment:</i>	A statement that updates the value of a variable using an operator like <code>+=</code> . (Chapter 10)
<i>base case:</i>	A conditional branch in a recursive function that does not make a recursive call. (Chapter 5)
<i>benchmarking:</i>	The process of choosing between data structures by implementing alternatives and testing them on a sample of the possible inputs. (Chapter 13)
<i>Big-Oh notation:</i>	Notation for representing an order of growth; for example, $O(n)$ represents the set of functions that grow linearly. (Chapter 20)
<i>body:</i>	The sequence of statements inside a function definition. (Chapter 3)
<i>Boolean expression:</i>	An expression whose value is either <code>True</code> or <code>False</code> . (Chapter 5)
<i>branch:</i>	One of the alternative sequences of statements in a conditional statement. (Chapter 5)
<i>bug:</i>	An error in a program. (Chapter 1)

<i>bytes object:</i>	An object similar to a string. (Chapter 14)
<i>call graph:</i>	A diagram that shows every frame created during the execution of a program, with an arrow from each caller to each callee. (Chapter 11)
<i>catch:</i>	To prevent an exception from terminating a program using the <code>try</code> and <code>except</code> statements. (Chapter 14)
<i>chained conditional:</i>	A conditional statement with a series of alternative branches. (Chapter 5)
<i>child class:</i>	A new class created by inheriting from an existing class; also called a "subclass". (Chapter 18)
<i>class attribute:</i>	An attribute associated with a class object. Class attributes are defined inside a class definition but outside any method. (Chapter 18)
<i>class diagram:</i>	A diagram that shows the classes in a program and the relationships between them. (Chapter 18)
<i>class object:</i>	An object that contains information about a programmer-defined type. The class object can be used to create instances of the type. (Chapter 15)
<i>class:</i>	A programmer-defined type. A class definition creates a new class object. (Chapter 15)
<i>comment:</i>	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)
<i>composition:</i>	Using an expression as part of a larger expression, or a statement as part of a larger statement. (Chapter 3)
<i>compound statement:</i>	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)
<i>concatenate:</i>	To join two operands end-to-end. (Chapter 2)
<i>condition:</i>	The Boolean expression in a conditional statement that determines which branch runs. (Chapter 5)
<i>conditional expression:</i>	An expression that has one of two values, depending on a condition. (Chapter 19)

<i>conditional statement:</i>	A statement that controls the flow of execution depending on some condition. (Chapter 5)
<i>counter:</i>	A variable used to count something, usually initialized to zero and then incremented. (Chapter 8)
<i>crossover point:</i>	The problem size where two algorithms require the same run time or space. (Chapter 20)
<i>data encapsulation:</i>	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)
<i>data structure:</i>	A collection of related values, often organized in lists, dictionaries, tuples, etc. (Chapter 12)
<i>database:</i>	A file whose contents are organized like a dictionary with keys that correspond to values. (Chapter 14)
<i>dead code:</i>	Part of a program that can never run, often because it appears after a {return} statement. (Chapter 6)
<i>debugging:</i>	The process of finding and correcting bugs. (Chapter 1)
<i>declaration:</i>	A statement like {global} that tells the interpreter something about a variable. (Chapter 11)
<i>decrement:</i>	An update that decreases the value of a variable. (Chapter 7)
<i>deep copy:</i>	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 <code>deepcopy</code> function in the <code>copy</code> module. (Chapter 15)
<i>default value:</i>	The value given to an optional parameter if no argument is provided. (Chapter 13)
<i>delimiter:</i>	A character or string used to indicate where a string should be split. (Chapter 10)
<i>dependency:</i>	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)
<i>designed development:</i>	A development plan that involves high-level insight into the problem and more planning than incremental development or prototype development. (Chapter 16)
<i>deterministic:</i>	Pertaining to a program that does the same thing each time it runs, given the same inputs. (Chapter 13)

<i>development plan:</i>	A process for writing programs. (Chapter 4)
<i>dictionary:</i>	A mapping from keys to their corresponding values. (Chapter 11)
<i>directory:</i>	A named collection of files, also called a folder. (Chapter 14)
<i>docstring:</i>	A string that appears at the top of a function definition to document the function's interface. (Chapter 4)
<i>dot notation:</i>	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)
<i>element:</i>	One of the values in a list (or other sequence), also called items. (Chapter 10)
<i>embedded object:</i>	An object that is stored as an attribute of another object. (Chapter 15)
<i>empty string:</i>	A string with no characters and length 0, represented by two quotation marks. (Chapter 8)
<i>encapsulation:</i>	The process of transforming a sequence of statements into a function definition. (Chapter 4)
<i>encode:</i>	To represent one set of values using another set of values by constructing a mapping between them. (Chapter 18)
<i>equivalent:</i>	Having the same value. (Chapter 10)
<i>evaluate:</i>	To simplify an expression by performing the operations in order to yield a single value. (Chapter 2)
<i>exception:</i>	An error that is detected while the program is running. (Chapter 2)
<i>execute:</i>	To run a statement and do what it says. (Chapter 2)
<i>factory:</i>	A function, usually passed as a parameter, used to create objects. (Chapter 19)
<i>file object:</i>	A value that represents an open file. (Chapter 9)
<i>filter:</i>	A processing pattern that traverses a list and selects the elements that satisfy some criterion. (Chapter 10)
<i>flag:</i>	A Boolean variable used to indicate whether a condition is true. (Chapter 11)

<i>floating-point:</i>	A type that represents numbers with fractional parts. (Chapter 1)
<i>floor division:</i>	An operator, denoted <code>//</code> , that divides two numbers and rounds down (toward negative infinity) to an integer. (Chapter 5)
<i>flow of execution:</i>	The order statements run in. (Chapter 3)
<i>formal language:</i>	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)
<i>format operator:</i>	An operator, <code>%</code> , 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)
<i>format sequence:</i>	A sequence of characters in a format string, like <code>{ %d}</code> , that specifies how a value should be formatted. (Chapter 14)
<i>format string:</i>	A string, used with the format operator, that contains format sequences. (Chapter 14)
<i>frame:</i>	A box in a stack diagram that represents a function call. It contains the local variables and parameters of the function. (Chapter 3)
<i>fruitful function:</i>	A function that returns a value. (Chapter 3)
<i>function call:</i>	A statement that runs a function. It consists of the function name followed by an argument list in parentheses. (Chapter 3)
<i>function definition:</i>	A statement that creates a new function, specifying its name, parameters, and the statements it contains. (Chapter 3)
<i>function object:</i>	A value created by a function definition. The name of the function is a variable that refers to a function object. (Chapter 3)
<i>function:</i>	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)
<i>functional programming style:</i>	A style of program design in which the majority of functions are pure. (Chapter 16)
<i>gather:</i>	An operation that collects multiple arguments into a tuple. (Chapter 12)
<i>generalization:</i>	The process of replacing something unnecessarily specific (like a number) with something appropriately general (like a variable or parameter). (Chapter 4)

<i>generator expression:</i>	An expression with a {for} loop in parentheses that yields a generator object. (Chapter 19)
<i>global statement:</i>	A statement that declares a variable name global. (Chapter 11)
<i>global variable:</i>	A variable defined outside a function. Global variables can be accessed from any function. (Chapter 11)
<i>guardian:</i>	A programming pattern that uses a conditional statement to check for and handle circumstances that might cause an error. (Chapter 6)
<i>HAS-A relationship:</i>	A relationship between two classes where instances of one class contain references to instances of the other. (Chapter 18)
<i>hash function:</i>	A function used by a hashtable to compute the location for a key. (Chapter 11)
<i>hashable:</i>	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)
<i>hashtable:</i>	A data structure that represents a collection of key-value pairs and performs search in constant time. (Chapter 20)
<i>hashtable:</i>	The algorithm used to implement Python dictionaries. (Chapter 11)
<i>header:</i>	The first line of a function definition. (Chapter 3)
<i>high-level language:</i>	A programming language like Python that is designed to be easy for humans to read and write. (Chapter 1)
<i>identical:</i>	Being the same object (which implies equivalence). (Chapter 10)
<i>immutable:</i>	The property of a sequence whose items cannot be changed. (Chapter 8)
<i>implementation:</i>	A way of performing a computation. (Chapter 11)
<i>import statement:</i>	A statement that reads a module file and creates a module object. (Chapter 3)
<i>increment:</i>	An update that increases the value of a variable (often by one). (Chapter 7)
<i>incremental development:</i>	A program development plan intended to avoid debugging by adding and testing only a small amount of code at a time. (Chapter 6)

<i>index:</i>	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)
<i>infinite loop:</i>	A loop in which the terminating condition is never satisfied. (Chapter 7)
<i>infinite recursion:</i>	A recursion that doesn't have a base case, or never reaches it. Eventually, an infinite recursion causes a runtime error. (Chapter 5)
<i>information hiding:</i>	The principle that the interface provided by an object should not depend on its implementation, in particular the representation of its attributes. (Chapter 17)
<i>inheritance:</i>	The ability to define a new class that is a modified version of a previously defined class. (Chapter 18)
<i>initialization:</i>	An assignment that gives an initial value to a variable that will be updated. (Chapter 7)
<i>instance attribute:</i>	An attribute associated with an instance of a class. (Chapter 18)
<i>instance:</i>	An object that belongs to a class. (Chapter 15)
<i>instantiate:</i>	To create a new object. (Chapter 15)
<i>integer:</i>	A type that represents whole numbers. (Chapter 1)
<i>interactive mode:</i>	A way of using the Python interpreter by typing code at the prompt. (Chapter 2)
<i>interface:</i>	A description of how to use a function, including the name and descriptions of the arguments and return value. (Chapter 4)
<i>interpreter:</i>	A program that reads another program and executes it. (Chapter 1)
<i>invariant:</i>	A condition that should always be true during the execution of a program. (Chapter 16)
<i>invocation:</i>	A statement that calls a method. (Chapter 8)
<i>IS-A relationship:</i>	A relationship between a child class and its parent class. (Chapter 18)
<i>item:</i>	In a dictionary, another name for a key-value pair. (Chapter 11)
<i>item:</i>	One of the values in a sequence. (Chapter 8)
<i>iteration:</i>	Repeated execution of a set of statements using either a recursive function call or a loop. (Chapter 7)

<i>iterator:</i>	An object that can iterate through a sequence, but which does not provide list operators and methods. (Chapter 12)
<i>key-value pair:</i>	The representation of the mapping from a key to a value. (Chapter 11)
<i>key:</i>	An object that appears in a dictionary as the first part of a key-value pair. (Chapter 11)
<i>keyword argument:</i>	An argument that includes the name of the parameter as a ``keyword''. (Chapter 3)
<i>keyword:</i>	A reserved word that is used to parse a program; you cannot use keywords like <code>if</code> , <code>def</code> , and <code>while</code> as variable names. (Chapter 2)
<i>leading term:</i>	In a polynomial, the term with the highest exponent. (Chapter 20)
<i>linear:</i>	An algorithm whose run time is proportional to problem size, at least for large problem sizes. (Chapter 20)
<i>list comprehension:</i>	An expression with a {for} loop in square brackets that yields a new list. (Chapter 19)
<i>list:</i>	A sequence of values. (Chapter 10)
<i>local variable:</i>	A variable defined inside a function. A local variable can only be used inside its function. (Chapter 3)
<i>logical operator:</i>	One of the operators that combines Boolean expressions: <code>and</code> , <code>or</code> , and <code>not</code> . (Chapter 5)
<i>lookup:</i>	A dictionary operation that takes a key and finds the corresponding value. (Chapter 11)
<i>loop:</i>	A part of a program that can run repeatedly. (Chapter 4)
<i>low-level language:</i>	A programming language that is designed to be easy for a computer to run; also called ``machine language" or ``assembly language". (Chapter 1)
<i>machine model:</i>	A simplified representation of a computer used to describe algorithms. (Chapter 20)
<i>map:</i>	A processing pattern that traverses a sequence and performs an operation on each element. (Chapter 10)
<i>mapping:</i>	A relationship in which each element of one set corresponds to an element of another set. (Chapter 11)



<i>memo:</i>	A computed value stored to avoid unnecessary future computation. (Chapter 11)
<i>method:</i>	A function that is associated with an object and called using dot notation. (Chapter 4)
<i>method:</i>	A function that is defined inside a class definition and is invoked on instances of that class. (Chapter 17)
<i>modifier:</i>	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)
<i>module object:</i>	A value created by an {import} statement that provides access to the values defined in a module. (Chapter 3)
<i>module:</i>	A file that contains a collection of related functions and other definitions. (Chapter 3)
<i>modulus operator:</i>	An operator, denoted with a percent sign %, that works on integers and returns the remainder when one number is divided by another. (Chapter 5)
<i>multiplicity:</i>	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)
<i>multiset:</i>	A mathematical entity that represents a mapping between the elements of a set and the number of times they appear. (Chapter 19)
<i>natural language:</i>	Any one of the languages that people speak that evolved naturally. (Chapter 1)
<i>nested conditional:</i>	A conditional statement that appears in one of the branches of another conditional statement. (Chapter 5)
<i>nested list:</i>	A list that is an element of another list. (Chapter 10)
<i>None:</i>	A special value returned by void functions. (Chapter 3)
<i>object diagram:</i>	A diagram that shows objects, their attributes, and the values of the attributes. (Chapter 15)
<i>object-oriented language:</i>	A language that provides features, such as programmer-defined types and methods, that facilitate object-oriented programming. (Chapter 17)

<i>object-oriented programming:</i>	A style of programming in which data and the operations that manipulate it are organized into classes and methods. (Chapter 17)
<i>object:</i>	Something a variable can refer to. An object has a type and a value. (Chapter 10)
<i>object:</i>	Something a variable can refer to. For now, you can use ``object'' and ``value'' interchangeably. (Chapter 8)
<i>operand:</i>	One of the values on which an operator operates. expression: A combination of variables, operators, and values that represents a single result. (Chapter 2)
<i>operator overloading:</i>	Changing the behavior of an operator like + so it works with a programmer-defined type. (Chapter 17)
<i>operator:</i>	A special symbol that represents a simple computation like addition, multiplication, or string concatenation. (Chapter 1)
<i>optional argument:</i>	A function or method argument that is not required. (Chapter 8)
<i>order of growth:</i>	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)
<i>order of operations:</i>	Rules governing the order in which expressions involving multiple operators and operands are evaluated. (Chapter 2)
<i>override:</i>	To replace a default value with an argument. (Chapter 13)
<i>parameter:</i>	A name used inside a function to refer to the value passed as an argument. (Chapter 3)
<i>parent class:</i>	The class from which a child class inherits. (Chapter 18)
<i>parse:</i>	To examine a program and analyze the syntactic structure. (Chapter 1)
<i>path:</i>	A string that identifies a file. (Chapter 14)
<i>persistent:</i>	Pertaining to a program that runs indefinitely and keeps at least some of its data in permanent storage. (Chapter 14)
<i>pipe object:</i>	An object that represents a running program, allowing a Python program to run commands and read the results. (Chapter 14)
<i>polymorphic:</i>	Pertaining to a function that can work with more than one type. (Chapter 17)

<i>portability:</i>	A property of a program that can run on more than one kind of computer. (Chapter 1)
<i>positional argument:</i>	An argument that does not include a parameter name, so it is not a keyword argument. (Chapter 17)
<i>postcondition:</i>	A requirement that should be satisfied by the function before it ends. (Chapter 4)
<i>precondition:</i>	A requirement that should be satisfied by the caller before a function starts. (Chapter 4)
<i>print statement:</i>	An instruction that causes the Python interpreter to display a value on the screen. (Chapter 1)
<i>problem solving:</i>	The process of formulating a problem, finding a solution, and expressing it. (Chapter 1)
<i>program:</i>	A set of instructions that specifies a computation. (Chapter 1)
<i>prompt:</i>	Characters displayed by the interpreter to indicate that it is ready to take input from the user. (Chapter 1)
<i>prototype and patch:</i>	A development plan that involves writing a rough draft of a program, testing, and correcting errors as they are found. (Chapter 16)
<i>pseudorandom:</i>	Pertaining to a sequence of numbers that appears to be random, but is generated by a deterministic program. (Chapter 13)
<i>pure function:</i>	A function that does not modify any of the objects it receives as arguments. Most pure functions are fruitful. (Chapter 16)
<i>quadratic:</i>	An algorithm whose run time is proportional to $n^2$ , where $n$ is a measure of problem size. (Chapter 20)
<i>raise statement:</i>	A statement that (deliberately) raises an exception. (Chapter 11)
<i>reassignment:</i>	Assigning a new value to a variable that already exists. (Chapter 7)
<i>recursion:</i>	The process of calling the function that is currently executing. (Chapter 5)
<i>reduce:</i>	A processing pattern that traverses a sequence and accumulates the elements into a single result. (Chapter 10)
<i>reduction to a previously solved problem:</i>	A way of solving a problem by expressing it as an instance of a previously solved problem. (Chapter 9)

<i>refactoring:</i>	The process of modifying a working program to improve function interfaces and other qualities of the code. (Chapter 4)
<i>reference:</i>	The association between a variable and its value. (Chapter 10)
<i>relational operator:</i>	One of the operators that compares its operands: <code>==</code> , <code>!=</code> , <code>&gt;</code> , <code>&lt;</code> , <code>&gt;=</code> , and <code>&lt;=</code> . (Chapter 5)
<i>relative path:</i>	A path that starts from the current directory. (Chapter 14)
<i>return statement:</i>	A statement that causes a function to end immediately and return to the caller. (Chapter 5)
<i>return value:</i>	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)
<i>reverse lookup:</i>	A dictionary operation that takes a value and finds one or more keys that map to it. (Chapter 11)
<i>rubber duck debugging:</i>	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)
<i>scaffolding:</i>	Code that is used during program development but is not part of the final version. (Chapter 6)
<i>scatter:</i>	An operation that makes a sequence behave like multiple arguments. (Chapter 12)
<i>script mode:</i>	A way of using the Python interpreter to read code from a script and run it. (Chapter 2)
<i>script:</i>	A program stored in a file. (Chapter 2)
<i>search (1):</i>	A pattern of traversal that stops when it finds what it is looking for. (Chapter 8)
<i>search (2):</i>	The problem of locating an element of a collection (like a list or dictionary) or determining that it is not present. (Chapter 20)
<i>semantic error:</i>	An error in a program that makes it do something other than what the programmer intended. (Chapter 2)
<i>semantics:</i>	The meaning of a program. (Chapter 2)
<i>sequence:</i>	An ordered collection of values where each value is identified by an integer index. (Chapter 8)

<i>shallow copy:</i>	To copy the contents of an object, including any references to embedded objects; implemented by the <code>copy</code> function in the <code>copy</code> module. (Chapter 15)
<i>shape error:</i>	An error caused because a value has the wrong shape; that is, the wrong type or size. (Chapter 12)
<i>shell:</i>	A program that allows users to type commands and then executes them by starting other programs. (Chapter 14)
<i>singleton:</i>	A list (or other sequence) with a single element. (Chapter 11)
<i>slice:</i>	A part of a string specified by a range of indices. (Chapter 8)
<i>special case:</i>	A test case that is atypical or non-obvious (and less likely to be handled correctly). (Chapter 9)
<i>stack diagram:</i>	A graphical representation of a stack of functions, their variables, and the values they refer to. (Chapter 3)
<i>state diagram:</i>	A graphical representation of a set of variables and the values they refer to. (Chapter 2)
<i>statement:</i>	A section of code that represents a command or action. So far, the statements we have seen are assignments and print statements. (Chapter 2)
<i>string:</i>	A type that represents sequences of characters. (Chapter 1)
<i>subject:</i>	The object a method is invoked on. (Chapter 17)
<i>syntax error:</i>	An error in a program that makes it impossible to parse (and therefore impossible to interpret). (Chapter 2)
<i>syntax:</i>	The rules that govern the structure of a program. (Chapter 1)
<i>temporary variable:</i>	A variable used to store an intermediate value in a complex calculation. (Chapter 6)
<i>text file:</i>	A sequence of characters stored in permanent storage like a hard drive. (Chapter 14)
<i>token:</i>	One of the basic elements of the syntactic structure of a program, analogous to a word in a natural language. (Chapter 1)
<i>traceback:</i>	A list of the functions that are executing, printed when an exception occurs. (Chapter 3)

<i>traverse:</i>	To iterate through the items in a sequence, performing a similar operation on each. (Chapter 8)
<i>tuple assignment:</i>	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)
<i>tuple:</i>	An immutable sequence of elements. (Chapter 12)
<i>type-based dispatch:</i>	A programming pattern that checks the type of an operand and invokes different functions for different types. (Chapter 17)
<i>type:</i>	A category of values. The types we have seen so far are integers (type <code>int</code> ), floating-point numbers (type <code>float</code> ), and strings (type <code>str</code> ). (Chapter 1)
<i>update:</i>	An assignment where the new value of the variable depends on the old. (Chapter 7)
<i>value (1):</i>	One of the basic units of data, like a number or string, that a program manipulates. (Chapter 1)
<i>value (2):</i>	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 <code>value</code> . (Chapter 11)
<i>variable:</i>	A name that refers to a value. (Chapter 2)
<i>veneer:</i>	A method or function that provides a different interface to another function without doing much computation. (Chapter 18)
<i>void function:</i>	A function that always returns <code>None</code> . (Chapter 3)
<i>worst case:</i>	The input that makes a given algorithm run slowest (or require the most space). (Chapter 20)
<i>zip object:</i>	The result of calling a built-in function <code>zip</code> ; an object that iterates through a sequence of tuples. (Chapter 12)