**Python Terminology**

* **>>>** - The typical Python prompt of the interactive shell. Often seen for code examples that can be tried right away in the interpreter.
* **Argument -** A value passed to a [function](https://docs.python.org/2/glossary.html#term-function) (or [method](https://docs.python.org/2/glossary.html#term-method)) when calling the function. There are two types of arguments:
  + *keyword* argument: an argument preceded by an identifier (e.g. name=) in a function call or passed as a value in a dictionary preceded by \*\*. For example, 3 and 5 are both keyword arguments in the following calls to [complex()](https://docs.python.org/2/library/functions.html#complex):
* complex(real=3, imag=5)
* complex(\*\*{'real': 3, 'imag': 5})
* *positional argument*: an argument that is not a keyword argument. Positional arguments can appear at the beginning of an argument list.
* complex(3, 5)
* **dictionary** - A built-in Python data type composed of arbitrary keys and values; sometimes called a "hash" or a "hash map" in other languages, although this is technically a misnomer (hashing is one way to implement an associative array but not the only way). The use of dict much resembles that for list, but the keys can be any object with a \_\_hash\_\_ function, not just integers starting from zero. Examples: d = {'A':65, 'B':66}, d = dict([('A', 65), ('B', 66)]), d['C'] = 67
* **docstring** - A string that appears as the lexically first expression in a module, class definition or function/method definition is assigned as the \_\_doc\_\_ attribute of the object where it is available to documentation tools or the help() builtin function.
* **exception** - Python handles all errors with exceptions. An exception is a signal that an error or other unusual condition has occurred. There are a number of built-in exceptions, which indicate conditions like reading past the end of a file, or dividing by zero. You can also define your own exceptions.
* **floor division -** Mathematical division that rounds down to nearest integer. The floor division operator is //. For example, the expression 11 // 4 evaluates to 2 in contrast to the 2.75 returned by float true division. Note that (-11) // 4 is -3 because that is -2.75 rounded *downward*.
* **function** - A block of code that is invoked by a "calling" program, best used to provide an autonomous service or calculation.
* **IDLE** - an Integrated Development Environment for Python. IDLE is a basic editor and intepreter environment that ships with the standard distribution of Python. Good for beginners and those on a budget, it also serves as clear example code for those wanting to implement a moderately sophisticated, multi-platform GUI application.
* **immutable** - An object with fixed value. Immutable objects include numbers, strings and tuples. Such an object cannot be altered. A new object has to be created if a different value has to be stored. They play an important role in places where a constant hash value is needed such as the keys of a **dictionary**.
* **Importing -** The process by which Python code in one module is made available to Python code in another module.
* **integer division** - Mathematical division discarding any remainder, for example 3 / 2 returns 1, in contrast to the 1.5 returned by float division. Also called "floor division". When dividing two integers the outcome will always be another integer (having the floor function applied to it). However, if one of the operands is another numeric type (such as a float), the result will be coerced (see coercion) to a common type. For example, an integer divided by a float will result in a float value, possibly with a decimal fraction. Integer division can be forced by using the '//' operator instead of the '/' operator.
* **interactive** - Python has an interactive interpreter which means that you can try out things and directly see its result, just launch python with no arguments. A very powerful way to test out new ideas, inspect libraries (remember x.\_\_doc\_\_ and help(x)) and improve programming skills.
* **list** - A built-in Python datatype, which is a **mutable** sorted **sequence** of values. Note that only sequence itself is mutable; it can contain **immutable** values like strings and numbers. Any Python **first-class object** can be placed in a tuple as a *value*.
* **list comprehension** - A neat syntactical way to process elements in a sequence and return a list with the results. result = ["0x%02x" % x for x in range(256) if x % 2 == 0] generates a list of strings containing hex numbers (0x..) that are even and in the range from 0 to 255. The if part is optional' all elements are processed when it is omitted.
* **mapping** - A container object (such as dict) that supports arbitrary key lookups using \_\_getitem\_\_.
* **module** ???
* **mutable** - Mutable objects can change their value but keep their id(). See also **immutable**.
* **namespace** - The place where a variable is stored in a Python program's memory. Namespaces are implemented as a dictionary. There are the local, global and builtins namespaces and the nested namespaces in objects (in methods). Namespaces support modularity by preventing naming conflicts. For instance, \_\_builtins\_\_.open() and os.open() are distinguished by their namespaces. Namespaces also aid readability and maintainabilty by making it clear which modules implement a function. For instance, writing random.seed() and itertools.izip() will make it clear that those functions are implemented by the random and itertoolsmodules respectively.
* **regular expression** - A formula for matching strings that follow some pattern. Regular expressions are made up of normal characters and *metacharacters*. In the simplest case, a regular expression looks like a standard search string. For example, the regular expression "testing" contains no metacharacters. It will match "testing" and "123testing" but it will not match "Testing". Metacharacters match some expressions like '.' metacharacter match any single character in a search string.
* **string** - One of the basic types in Python that store text. In Python 2.X strings store text as a 'string of bytes', and so the string type can also be used to store binary data. Also see Unicode.
* **triple-quoted string** - A string that is bounded by three instances of either the double quote mark (") or the single quote mark ('). For instance:

'''This is such a string'''

They are useful for multiple reasons: they allow you to include both single and double quotes within a string quite easily, and they can span multiple lines without the use of line-continuation characters (very useful in [docstrings](https://wiki.python.org/moin/PythonGlossary" \l "docstring)).

* **tuple** - (pronounced TUH-pul or TOO-pul) A built-in Python datatype, which is an **immutable** ordered **sequence** of values. Note that only the sequence itself is immutable. If it contains a **mutable** value such as a **dictionary**, that value's content may be changed (e.g. adding new key/value pair). Any Python **first-class object** can be placed in a tuple as a *value*.
* **type** - A "sort" or "category" of data that can be represented by a programming language. Types differ in their properties (such as mutability and immutability), the methods and functions applicable to them, and in their representations. Python includes, among others, the string, integer, long, floating point, list, tuple, and dictionary types.
* **unicode** - The unicode type is the companion to the string type. They are used to store text with characters represented as Unicode code points.
* **whitespace** - The unconventional use of space characters (' ') to control the flow of a program. Instead of a loosely-enforced ideal, this is an integral part of Python syntax. It's a tradeoff between readability and flexibility in favor of the former.
* **Zen of Python** - listing of Python design principles and philosophies that are helpful in understanding and using the language effectively. The listing can be found by typing "import this" at the interactive prompt.