Glossary

disclaimer: these definitions are intended to be applied narrowly to the topic of this course, i.e., Object Oriented Python. If a more universal definition is needed, search online or consult a definitive reference.

**abstract class**

A parent class that is not intended to be instantiated, but instead to be subclassed (i.e., inherited from). It may provide methods that can be called by a child class, or may provide *abstract methods.*

**abstract method**

A method declared in a parent class that is expected (and required) to be implemented in the inheriting class.

**argument**

An object passed to a function or method.

**attribute**

An object associated with a class or instance, and accessed using **object.attribute** syntax. See also: *class attribute*, *instance attribute.*

**attribute lookup**

See *lookup.*

**base class**

See *parent class.*

**boolean**

An object type that holds only the values **True** and **False**.

**brittle**

Code that has dependencies such that a change to one part of the code will adversely affect or may break another part of the code. Brittle code is harder to maintain because of these dependencies.

**bug**

An error in the code that causes the code to fail or perform incorrectly.

**built-in**

Refers to variable names that are available in any Python program, usually built-in functions.

**call**

To invoke a function or method. The syntax of a call is the function or method name followed by parentheses, optionally containing arguments.

**callable attribute**

Synonymous with *method:* an attribute of a class or instance that also serves as a function and can be called; see *method*.

**calling code**

Code that makes use of other modules or classes, calling its functions and methods. Often refers to the code that imports a module and then makes use of its code by calling it.

**"cheap"**

Code that makes minimal use of memory and time in execution. Compare with *expensive*.

**child class**

An inheriting class, one that inherits from a *parent class*.

**class**

A code construct that defines the attributes and behavior of instances constructed from it as well as its own attributes and behavior. The core concept behind Object-Oriented Programming.

**class attribute**

A variable defined in the class (or through the class object) that is available as an attribute of the class object, i.e. **class.attribute**

**class method**

As constrasted with an instance method, a method that takes the class as its implicit first argument. Usually decorated with the **@classmethod** decorator.

**"classic" class**

See *new-style class*.

**construct**

Create a new instance by calling the class, e.g. **inst = MyClass()**

**constructor**

The specially-named method **\_\_init\_\_()** which is called automatically when a new instance is constructed. More generically this term can refer to calling the class itself, e.g. **inst = MyClass()**

**debugging**

Figuring out what is wrong with your code -- using an IDE's debugger, the Python debugger (**pdb** module) or simple print statements.

**decorator**

A special function that modifies the behavior of another function. In use, they are noted with an @ sign, i.e.:

@staticmethod

def dothis():

print 'hello'

**decoupling**

Refers to writing (or rewriting) code so that there are limited dependencies between functions, modules, etc.

**derived class**

See *child class.*

**descriptor**

An object attribute that has getter and setter methods defined for it. Descriptors provide control for how an object sets and gets attribute values.

**duck typing**

Refers to determining the type, or general type, of an object by what it can and can't do, i.e. which methods it supports. From the aphorism "if it walks like a duck and quacks like a duck, it must be a duck." In contrast with type inspection (i.e., **print type(obj)**), duck typing is considered to be more "Pythonic".

**encapsulation**

Refers to the coding discipline of maintaining the integrity of the attribute values ("state") of an instance, "integrity" here meaning that the values are correct for the purposes of the object and class. The first of the three "pillars" of Object Oriented Programming.

**enclosing**

When a function is nested inside another function, the outer function can be said to encompass the *enclosing* scope of the inner function.

**execute**

Run a program. Cause its statements to be executed one at a time.

**"expensive"**

An operation or code block that makes excessive use of memory and/or time in execution. Compare with "cheap".

**extend**

In an inheriting or child class, implement a parent class method such that the parent class method is called, and additional code is executed in the child.

**function**

In Python, refers to a built-in or user-defined function that stands alone, in contrast to a *method*, which is a function that works with a class and/or instance.

**function, built-in**

A function defined by Python and available in every Python program. Examples: **len()**, **type()**, **str()**, **print** (print has special syntax and does not require parentheses - in Python 2 it is known as a statement).

**function definition**

The code that defines a function and shows what arguments it takes and what it does, starting with a **def** declaration and followed by an indented block.

**function decorator**

See decorator.

**globals**

In a Python script/module, variables that are declared in the main body of the code, i.e. not in a function. See **local**. Classes enjoy a different variable name lookup scheme, so variables are either class variables/attributes or instance attributes, not **global** or **local**.

**Guido**

Guido van Rossum, the original author of Python and its BDFL, or Benevolent Dictator for Life. Continues to oversee the Python community's efforts to develop and extend the language.

**high-level language**

See *low-level language*.

**IDE**

Integrated Development Environment. A code editor that is aware of the language you are coding in and provides syntax highlighting, syntax and usage checks, a debugger, etc. to power the code writing process.

**implementation**

Code that supports a feature or an object-oriented interface. Can also refer to the code inside a method.

**inherit**

In Object-Oriented Programming, a class that lists another class in its inheritance list as part of its class definition. An inheriting class extends its attribute lookup to the parent class, so that an attribute accessed in an instance of the child class may also look for that attribute in the parent class.

**inheritance**

The second of the three pillars of object-oriented programming. See *inherit*.

**inspection**

Examining an object or class to determine its attributes, or other qualities such as its type.

**instance**

A code construct that is created from a class "blueprint" and is of its class' type, encapsulates its own data and has access to methods and other attributes defined in the class. Synonymous with *object*.

**instance attribute**

A value bound to an instance/object. Accessed through **object.attribute** syntax. An attribute is always bound to an object, whether a string, integer, list, or any other type.

**instance method**

A method defined in the class that is designed to work with the instance. The most common type of method, two others being *class method* and *static method*. An instance method can be identified by the presence of *self* (the instance) as the first argument in the arguments list. Instance methods generally work with *self*, to read and/or write its attributes.

**instantiate**

See *construct.*

**interface**

In Object-Oriented Programming, an instance's *interface* signifies the attributes and methods through which we can communicate with it.

In inheritance, *interface* refers to a class that specifies what methods a child class must implement. Generally speaking, the Pythonic way is to define an *abstract* class rather than an interface class.

**literal**

A value entered literally into code, such as **5** or **"hello"**

**local**

When referring to Python variables, *local* variables are those that are defined inside a function.

**lookup**

Refers to the resolution (i.e., accessing the value) of a variable or attribute, through a name. In classes and objects, "lookup" refers to *attribute lookup*, in which Python tries to access the value of an attribute, i.e. with the syntax **object.attribute**. Any access of this kind will cause Python to look for the attribute first in the instance, then in its class, then in any parent classes of its class.

**low-level language**

A programming language that is "closer to the machine" in terms of memory management and machine architecture. Lower-level programs (like those written in C and C++) are usually precompiled for the machine they will run on, and often require the programmer to define the memory types and sizes that will be required to run the program. This results in programs that run faster and more efficiently, but take more time and planning to write. Higher-level languages (like Python and php) allow the programmer much more flexibility by handling memory allocation, which reults in programs that run slower and less efficiently, but are much easier to develop.

**"magic" attributes or methods**

Those that begin and end with a double underscore signify attributes that are not intended to be accessed directly by the calling code, but are usually accessed as a result of some other operation (for example, **\_\_add\_\_** is accessed when we use the "+" sign).

**metaclass**

A class that defines other classes. Classes can be constructed from metaclasses, which can establish attributes and methods (and thus behaviors) that the resulting class will incorporate.

**method**

A *callable attribute*; a function defined in the class that is designed to work with the class or its instances, or to do other work related to the class.

**method resolution order (mro)**

In inheritance, the order in which Python will search for attributes in an inheritance tree. Becomes significant when there are same-named attributes or methods in more than one class within such a tree. Python generally follows a "depth first" approach, but this order is slightly different when more than one class inherits from the same parent class.

**module**

A python script, whether one designed to be executed directly or to be imported by another Python script, or both. The term is most often used to mean scripts that are imported.

**mro**

See *method resolution order*.

**namespace**

A space in which names live and can be accessed. In procedural programming this usually refers to the *global*, *local*, *enclosing* and *built-in* namespaces. In Object-Oriented Programming, this can refer to attribute lookup, in which Python attempts to access the value of an attribute first in the instance, then in the class, then in any inherited classes, each of which has its own namespace.

**new-style class**

A class that inherits from the parent class **object**. The new-style paradigm unifies classes and types, providing a unified object model and a full meta-model. New-style classes were introduced in conjunction with important new internal features such as subclassing built-in types, static methods and class methods, properties, descriptors, a refined method resolution order, the *super* keyword, metaclasses and more. It should be noted that up to an intermediate level, the differences are usually not significant aside from the useful new features that are available.

**object**

See *instance*.

**object-relational mapping**

In databases, a mapping or reconfiguring of data contained in objects and object attributes as would be found (for example) in Python objects, to tables as would be found in a relational database. Enables developers to work with objects without having to consider how they are stored. Contrast with *object databases* which store objects natively.

**old-style class**

Classes that follow the paradigm in use before the introduction of *new-style classes*, q.v. Also known as *classic-style* classes, old-style is the default in Python 2 (provided for backward compatibility), and was removed in Python 3.

**overload / override**

To implement a method in a child class that would otherwise be inherited from a parent class.

**parent class**

A class from which a *child class* inherits.

**pass**

To send an argument or arguments to a function or method.

**persistence**

Storing data (as a data structure, instance or class) on disk so that it is available during a subsequent run of a program. Constrasted with in-memory storage, i.e. variables.

**polymorphism**

When two instances/objects of different classes have same-named methods that do something conceptually similar, the method can be called on either object and it will do the right thing for that object. The third of the three pillars of Object-Oriented Programming.

**primitive**

A value that is not an object (such as an integer or float value), as exists in most other languages. In Python everything is an object, so the term is not used. Such values are also known as *literals*. (The term "literal" may be used to describe values as entered directly in code, but even these values are objects in Python.)

**procedural**

All programs consist of procedures, but this term is usually used to mean "non-Object-Oriented".

**program**

A series of statements collected in a file or files.

**programming language**

A language for writing *programs*. Sometimes contrasted with *scripting language* as a language that is compiled in a separate step, before it is run, although the term is also used broadly to mean both programming and scripting languages. Common programming languages are Java, C, C++ as well as those also known as scripting languages, such as Python, Perl, php and Ruby.

**Pythonic**

Refers to coding Python the "right" way; although there are a number of axioms in Python, this term is used rather broadly to refer to what the user of the term thinks is the right thing to do. (This doesn't meant there isn't broad agreement over many aspects of what is considered *Pythonic.*)

**relational**

In mathematics, a *relation* is data arranged in a tabular structure. A *relational database* stores its data in tables.

**run**

Execute a program. Cause its statements to be executed one at a time.

**"self"**

In an instance method, the name customarily used to refer to the instance. Instance methods can always be identified by the presence of **self** as the first argument in the method's arguments list. When a method is called on an instance, the instance is automatically passed as the implicit first argument, and this is customarily called **self**.

**serialization**

Generally refers to the writing of data to disk. Object serialization writes an object to disk so that it can be read and used during a subsequent run of a program, or by another program. It is extremely convenient in that it is not necessary to translate the object's values to some other form such as tabular (relational).

**script**

See *program* and *scripting language*

**scripting language**

A programming language that is generally regarded as having particular qualities: its programs are compiled at the time of execution; it allows variables to change type during execution (dynamic typing); it is used to write shorter programs. Usually applied to higher-level languages like Python, Perl and Ruby.

**state**

In Object-Oriented Programming, the attribute values an object holds. As an object's values change, so does its state.

**static method**

A method within a class that is not designed to work with an instance (an *instance method*) nor with the class (a *class method*), but nevertheless does work related to the class' purpose (such as a utility function used by other methods in the class). Such methods are usually decorated with **@staticmethod**.

**subclass (noun)**

A child class, a class that inherits from a parent or superclass. Such a class will list the parent class(es) in its in inheritance list as part of the class definition.

**subclass (verb)**

To inherit from another class.

**subroutine**

Generally synonymous with *function*. A routine that is called by another part of the program.

**superclass**

A parent class or base class. A class from which another class inherits. Such a class will be listed in the inheriting class' inheritance list as part of its definition.

**terminal window**

A window designed to allow acess to the operating system through text-based commands. Although an IDE can execute our Python programs, it is important to also know how to launch them through a terminal window.

**three pillars of OOP**

Encapsulation, Inheritance and Polymorphism, q.v.