Excerpt from the docs.python.org description of

**The Python Data Model**

From [https://docs.python.org/3.5/reference/datamodel.html](https://docs.python.org/3.5/reference/datamodel.html)

**Modules**

Modules are a basic organizational unit of Python code, and are created by the `import` system as invoked either by the `import` statement (see `import`), or by calling functions such as `importlib.import_module()` and built-in `__import__()` . A module object has a namespace implemented by a dictionary object (this is the dictionary referenced by the `__globals__` attribute of functions defined in the module). Attribute references are translated to lookups in this dictionary, e.g., `m.x` is equivalent to `m.__dict__['x']` . A module object does not contain the code object used to initialize the module (since it isn’t needed once the initialization is done).

Attribute assignment updates the module’s namespace dictionary, e.g., `m.x = 1` is equivalent to `m.__dict__['x'] = 1`.

Special read-only attribute: `__dict__` is the module’s namespace as a dictionary object.

...

**Custom classes**

Custom class types are typically created by class definitions (see section Class definitions). A class has a namespace implemented by a dictionary object. Class attribute references are translated to lookups in this dictionary, e.g., `C.x` is translated to `C.__dict__['x']` (although there are a number of hooks which allow for other means of locating attributes).

When the attribute name is not found there, the attribute search continues in the base classes. This search of the base classes uses the C3 method resolution order which behaves correctly even in the presence of ‘diamond’ inheritance structures where there are multiple inheritance paths leading back to a common ancestor. Additional details on the C3 MRO used by Python can be found in the documentation accompanying the 2.3 release at [https://www.python.org/download/releases/2.3/mro/](https://www.python.org/download/releases/2.3/mro/).

When a class attribute reference (for class `C`, say) would yield a class method object, it is transformed into an instance method object whose `__self__` attributes is `C`. When it would
yield a static method object, it is transformed into the object wrapped by the static method object. See section Implementing Descriptors for another way in which attributes retrieved from a class may differ from those actually contained in its `__dict__`.

Class attribute assignments update the class’s dictionary, never the dictionary of a base class.

A class object can be called (see above) to yield a class instance (see below).

Special attributes: `__name__` is the class name; `__module__` is the module name in which the class was defined; `__dict__` is the dictionary containing the class’s namespace; `__bases__` is a tuple (possibly empty or a singleton) containing the base classes, in the order of their occurrence in the base class list; `__doc__` is the class’s documentation string, or None if undefined.

**Class instances**

A class instance is created by calling a class object (see above). A class instance has a namespace implemented as a dictionary which is the first place in which attribute references are searched. When an attribute is not found there, and the instance’s class has an attribute by that name, the search continues with the class attributes. If a class attribute is found that is a user-defined function object, it is transformed into an instance method object whose `__self__` attribute is the instance. Static method and class method objects are also transformed; see above under “Classes”. See section Implementing Descriptors for another way in which attributes of a class retrieved via its instances may differ from the objects actually stored in the class’s `__dict__`. If no class attribute is found, and the object’s class has a `__getattr__()` method, that is called to satisfy the lookup.

Attribute assignments and deletions update the instance’s dictionary, never a class’s dictionary. If the class has a `__setattr__()` or `__delattr__()` method, this is called instead of updating the instance dictionary directly.

Class instances can pretend to be numbers, sequences, or mappings if they have methods with certain special names. See section Special method names.

Special attributes: `__dict__` is the attribute dictionary; `__class__` is the instance’s class.