Some vocabulary

We’ll use the terms “value” and “object” interchangeably.
We’ll use the terms “name” and “variable” interchangeably.

A **binding** is a runtime pair: name \( \mapsto \) value.

A **namespace** is a runtime collection of bindings.

At runtime, an **assignment** binds a name to a value. An assignment is a “write” operation: it changes the state of the world.
Example: `values = [1, 2, 3]`

At runtime, a **reference** looks up a name’s value. A reference is a “read” operation: it observes the state of the world.
Example: `print(values)`

At runtime, a **modification** changes the value of an object. (Note: to change its value, the object’s type must be mutable, e.g., a list, a set, or a dictionary.)
Example: `values[0] = 1000`

A name’s **scope** is the region of text in which a name is valid. As long as a name has been bound to a value, it’s legal to refer to that name in the rest of the scope.

Python scopes

Every name (e.g., of a variable, or a function, etc.) in Python has a scope: the region of text in which it’s valid to use that name to look up its value.

When we bind a name to a value (e.g., through assignment), we say that we’ve “introduced that name” in its scope.

Once a name has been introduced, we’re allowed to use that name to look up its value. We say that the name is “in scope”. If a name has no meaning in a scope, we say it is “out of scope”.

Python defines three kinds of scopes:

- **The global scope** covers all the text in a file (or in an interactive session). Names for global variables and functions are in the global scope. We sometimes also call this scope the file, module or session scope. There is only one global scope.

- A **function’s scope** covers all the text in the body of that function. The function definition introduces the names of the parameters, plus any names introduced by assignments in the body of the functions. Names introduced in the body of the function are called “local variables”. There are as many function scopes as there are function definitions.

- The **builtin scope** covers the entire program text. Names of built-in functions (e.g., `help` and `int`) are valid in this scope. There is only one builtin scope. Our programs don’t typically introduce new names to this scope; our programs just refer to names in this scope.

---

1 Okay, it’s actually four kinds of scopes, but the fourth kind is subtle enough that we don’t need to worry about it right now. For a more nuanced and complete description, see: https://tinyurl.com/hvry3xa
Python namespaces

A namespace keeps track of all the bindings for a scope, as the program runs. A namespace is, essentially, a dictionary that Python creates and modifies, as the program runs. In this dictionary, a key is a variable name, and a value is the object to which that name is bound.

At any point in the program, there are at most three active namespaces:

- **The local namespace** corresponds to the scope of the function that is currently running. It contains bindings for all the parameters and local variables of that function. If no function is currently running, then there is no active local namespace.

- **The global namespace** corresponds to the global scope. It contains all the bindings for global variables and “top-level” definitions (e.g., functions). There is always one active global namespace.

- **The builtin namespace** corresponds to the builtin scope. It contains all the bindings for the built-in functions such as `help` and `int`. There is always one active builtin namespace.

How Python uses namespaces

When Python starts up, it creates a builtin namespace and populates it with bindings for all the built-in functions.

Next, Python creates an empty, global namespace.

Then, Python starts at the first line of the program and starts executing. As the program runs, Python introduces and modifying bindings in the appropriate namespace. Python determines which namespace is appropriate based on the scope of the variable being introduced / modified.

Every time Python sees an assignment or a function definition, it will make a binding in the global namespace.

If Python sees a function call, it will create a local namespace for that function. In this local namespace, it will bind each parameter name to its corresponding argument value. Then, Python starts at the first line of the function and starts executing. Python creates bindings for local variables in the local namespace. When Python sees a `return` statement, it destroys the local namespace before executing the line of code that follows the original function call.

When a running program refers to a variable’s name, Python will try to look up the value for that name. (Instead of saying “Python looks up the value for a name”, we often say “Python resolves the name”.) Python always runs the same algorithm to resolve a name:

1. **Local**: If there is an active local namespace, look for a binding there. If a binding exists for the name, then the corresponding value is used.

2. **Global**: If there is no active local namespace or if resolution fails for the local namespace, then look for a binding in the global namespace. If a binding exists for the name, then the corresponding value is used.

3. **Builtin**: If resolution fails for the global namespace, then look for a binding in the builtin namespace. If a binding exists for the name, then the corresponding value is used.

4. **Error**: If resolution fails for the builtin namespace, throw a `NameError`.

Next time: object-oriented programming (in Python)