CS 42—Introduction to Python

Thursday, October 25, 2018

Summary

Today, we'll start exploring the Python programming language. We'll also use Python to delve deeper into the concepts of bindings and scope, which we discussed briefly in Racket.

Terminology

Note: We've seen some of these terms before, but it's helpful to review them.

primitive value: In a programming language, a primitive value is a literal value that's built into the language. Examples from Python include `42` (an integer), `42.0` (a floating point number), and `'platypus'` (a string).

object: For now, we will use the terms "value" and "object" interchangeably. (A more thorough definition of object is coming later in the semester.)

primitive operation: In a programming language, a primitive operation is one that's built into the language. Examples from Python include `+` and `print`.

runtime: The time during which a program is running.[1]

expression: An expression is a sequence of operations that can be evaluated at runtime to produce a value, for example at runtime, the expression `1 + 2` evaluates to `3`.

side effect: In programming, a side effect is something the code does to change state, in a way that persists even after the code has been evaluated. Examples include: assignment, printing to the screen, and reading from the keyboard.

statement: A statement is a sequence of operations that has a side effect, for example `print(1 + 2)`. Some statements are also expressions—they have a side effect *and* result in a value—for example, any function call in Python that has a side effect and returns a value.

parameter of a function definition: a name associated with an argument value.

argument to a function call: the value passed to a function.

binding (noun): A binding associates a name with a value. Bindings happen at runtime, as the result of assignment statements such as `x = 3`, which binds the name `x` to the value `3`. A function call also creates bindings between each parameter name and its corresponding argument value.

namespace: A namespace is a runtime collection of zero or more bindings.

reference: At runtime, a reference looks up a name's value. For example, `x = y` first references `y`'s value, then binds that value to `x`.

scope: The scope of a variable is the part of the program where the variable is valid. In other words, if `x` is bound to `3`, `x`'s scope is all the places in the program where it would be valid to reference `x`.

---

[1] There's another meaning that computer scientists have for "runtime", which is, "how expensive is it?" For example, "The runtime of this algorithm is O(n)."

Scopes and namespaces in Python

Python code has three[2] 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 a function. The function definition introduces the names of the parameters, plus any names introduced by assignments that appear in the body of the function. 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 builtin functions (e.g., `print`) 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.

At runtime, 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" (i.e., non-indented) 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 `print`. There is always one active builtin namespace.

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 running it. As the program runs, Python introduces and modifies 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 a top-level (i.e., unindented) assignment or function definition, it makes a binding in the global namespace. If Python sees a function call, it creates a local namespace for that function. In this local namespace, it binds each parameter name to its corresponding argument value. Then, Python starts at the first line of the function and runs the function. While running a function, 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 followed the original function call.

When a running program refers to a variable's name, Python tries 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`.

---

[2] Okay, it's actually four kinds of scopes. For a more nuanced and complete description, see: tinyurl.com/hyry3xa

Next time: a fun(ctional programming) lab in Python.