Programmieren I (Python)

Christian Osendorfer

2023-11-17

Memory

Frames

- A **frame** keeps track of variable-to-value **bindings** (see Python Tutor).
- Every function call keeps track of its own set of bindings (e.g. it has its own **scope**), so every function call has a corresponding frame (*call frame*).
- The **global frame**, or “Global” for short, is the starting frame for every Python program.
- Every frame except for Global has a parent frame. When a function is called, its corresponding frame immediately has a parent — that parent is the frame in which that function was called.
- When doing variable lookup, if you can’t find a variable in the current frame, look it up in *Global* (if not found, generate an error).

```
1 a = 2
2 def get_a():
3   return a
4 # What is happening when you assign a value to 'a' *in the function*?
```

Variables/Values

- A **variable** (or **name**) is distinct from the **value** which it is assigned (**bound** to).
- A variable is a box with a label, the value is the thing you put into the box.
- For non-scalar types (e.g. functions, lists, your classes, …) the value of a variable is a **reference to an object**. Python Tutor shows this as an arrow (resembling a **pointer**).
- There are three distinct areas of memory (in Python):
  - Global
  - Call Stack (where all call frames are managed)
  - Heap (where all objects exist)

Global

- A place to store *global variables*.
- Lasts until Python is quit.
- Global variables
  - Any assignment not in a function definition
  - Module and function definitions

Call Stack

- A call frames is created on a function call
- Functions call other functions: many call frames

Heap

- All objects exits on the heap (see Python Tutor)
  - For example, modules and function definitions, too!
- In order to access an object on the heap, a variable referencing it is necessary.
  - If no variable references an object, it will be removed (maybe not immediately)
    - In Python Tutor, this object disappears immediately.
    - Denoted *Garbage collection*.
- Variables referencing objects are in Global, in call frames or in other objects!

Modules

- Importing a module:
  - creates a global variable (with the name of the module)
  - puts the module object onto the heap
    - module variables
    - module functions
  - puts the reference to the module object into the global variable.
- `from module import ...`: dump the imported names into *Global*!

Module vs Class

- Module object sounds like a class object? Why have this distinction?
- Classes can have many instances.
- Each module is a **unique instance**.
  - only one possibility for e.g. `pi`.
    - that's why we import these.
  - Also called **singleton** *objects*.

Functions

- A function definition:
  - creates a global variable (name of the function).
  - creates an object representing the function body.
  - puts a reference to this object into the variable.
- Function names are just variables.

Function calls

- Evaluate the operands of the function call. If an operand is itself a function call, apply *this* procedure to it.
- Generate a call frame. Store information about the parent frame (e.g. how to get there).
- Bind function variables (parameters) to evaluted operands.
- Execute the body of the function.
- Write the return value into the parent frame.