Environments in Python

Our subset of Python is now complex enough that the meaning of programs is non-obvious. What if a formal parameter has the same name as a built-in function? Can two functions share names without confusion? To resolve such questions, we must describe environments in more detail.

An environment in which an expression is evaluated consists of a sequence of *frames*, depicted as boxes. Each frame contains*bindings*, each of which associates a name with its corresponding value. There is a single *global* frame. Assignment and import statements add entries to the first frame of the current environment. So far, our environment consists only of the global frame.

```
1  from math import pi
2  tau = 2 * pi
```

Edit code

< Back Program terminated Forward >

Global frame

pi	3.142
tau	6.283

This *environment diagram* shows the bindings of the current environment, along with the values to which names are bound. The environment diagrams in this text are interactive: you can step through the lines of the small program on the left to see the state of the environment evolve on the right. You can also click on the "Edit code" link to load the example into the Online Python Tutor, a tool created by Philip Guo for generating these environment diagrams. You are encouraged to create examples yourself and study the resulting environment diagrams.

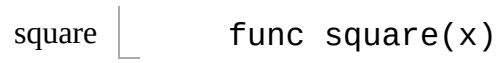
A `def` statement also binds a name to the function created by the definition. The resulting environment after defining `square`appears below:

```
1  from operator import mul
```

Global frame

mul	func mul(...)

```
2  def square(x):
3      return mul(x, x)
```



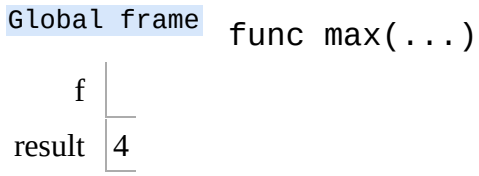
Edit code

< Back Program terminated Forward >

Notice that the name of a function is repeated, once in the global frame, and once as part of the function itself.

This repetition is intentional: many different names may refer to the same function, but that function itself has only one intrinsic name. However, looking up the value for a name in an environment only inspects bound names. The intrinsic name of a function **does not**play a role in look up. In the example we saw earlier,

```
1  f = max
2  result = f(2, 3, 4)
```



Edit code

< Back Program terminated Forward >

The name *max* is the intrinsic name of the function, and that's what you see printed as the value for `f`. In addition, both the names`max` and `f` are bound to that same function in the global environment.

Function Signatures. Functions differ in the number of arguments that they are allowed to take. To track these requirements, we draw each function in a way that shows the function name and its formal parameters. The user-defined function `square` takes only`x`; providing more or fewer arguments will result in an error. A description of the formal parameters of a function is called the function's signature.

The function `max` can take an arbitrary number of arguments. It is rendered as `max(...)`. Regardless of the number of arguments taken, all built-in functions will be rendered as `<name>(...)`, because these primitive functions were never explicitly defined.

Source : http://inst.eecs.berkeley.edu/~cs61A/book/chapters/functions.html#environments