```
j = 0
i = j
k = 12.0
j = 2 * k
assert i != j
```

The important difference between dynamic variables and static variables is that *static variables with C types have C semantics*, which changes the behavior of assignment. It also means these variables follow C coercion and casting rules.

In the previous example, `i = j` *copies* the integer data at `j` to the memory location reserved for `i`. This means that `i` and `j` refer to independent entities, and can evolve separately.

As with C, we can declare several variables of the same type at once:

```
cdef int i, j, k
cdef float price, margin
```

Also, we can provide an optional initial value:

```
cdef int i = 0
cdef long int j = 0, k = 0
cdef float price = 0.0, margin = 1.0
```

Inside a function, `cdef` statements are indented and the static variables declared are local to that function. All of these are valid uses of `cdef` to declare local variables in a function `integrate`:

```
def integrate(a, b, f):
    cdef int i
    cdef int N=2000
    cdef float dx, s=0.0
    dx = (b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s * dx
```

An equivalent way to declare multiple variables is by means of a `cdef` block, which groups the declarations in an indented region:

```
def integrate(a, b, f):
    cdef:
        int i
        int N=2000
        float dx, s=0.0
    # ...
```

This groups long lists of `cdef` declarations nicely, and we will use both forms throughout this book.