**Procedural Abstraction:**
Functions

also known as procedures, methods, subroutines

**Chunking**

- Very limited working capacity
- Nearly unlimited complexity

**Modular Structure**

- Brain-size chunks
  - Not too many details. Not too many interactions.
  - Something to focus on.
- Units of work
  - Good size for a work assignment. Good size to build and test before moving on.
- Units of change
  - Something that can be replaced

---

**O, be some other name! – Juliet**

```python
def diff(a, b):
    return a - b

... then in main ...
a = 5
b = 7
c = diff(b, a)
## ?? What happens
```
```python
def diff(a, b):
    return a - b

... then in main ...
a = 5
b = 7
c = diff(b, a);
```

```python
def diff(int a, int b):
    return a - b

... then in main ...
a = 5
b = 7
c = diff(b, a)
```

The called method gets its own copies of the inputs, by position (not by name).

```
def foo(x, y):
    x = 17
    y = 19

    x = 3
    m = 22
    foo(x, m)
    print "Now", x, "and", m
```

"Pass by value"

```
def foo(a, b, c, d):
```

"What does it print?"

The "actual arguments" x, y, 37.489, "et tu, Brute?" are copied into
the "formal arguments" of the function.
The copies become distinct, local variables.
What makes a good function?

- Simplifies the code that calls it
- Isolates a design decision (easier to change)
- Used more than once
- Can be tested separately

...  

*A good function may have only some of these properties. Few have all.*

Bad function smells

- Complicated description
  - If the *simplest* description is “*blahblah and blah and blah except blah or blah*”, maybe it shouldn’t be a method

- Have to keep looking back at it
  - I should be able to **use** the function without remembering details of how it works