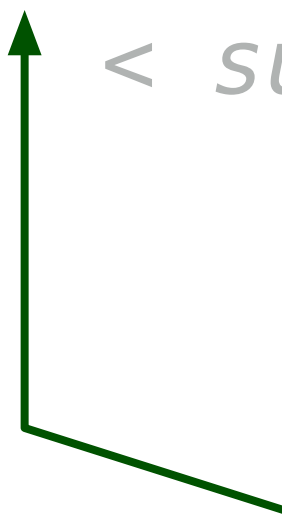**Anatomy of a function. In particular, a function called 'monkey':**

```python
def monkey(name, age):
  print "Monkey name: " + name
  if name == "Rufus":
    print "Great name, Rufus!"
  elif name == "Sputnik":
    print "Same as my dog."
  print "Monkey's age: " + str(age)
  if age < 4:
    print "Monkey is young."
```

```python
def monkey(name, age):
    < suite of code goes here >
```

The 'def' keyword tells python that we are about to define a new function. It expects the stuff after it to follow a specific format. In other words, what you type here has to be grammatically correct, or it will not know what you mean and *totally freak out*.
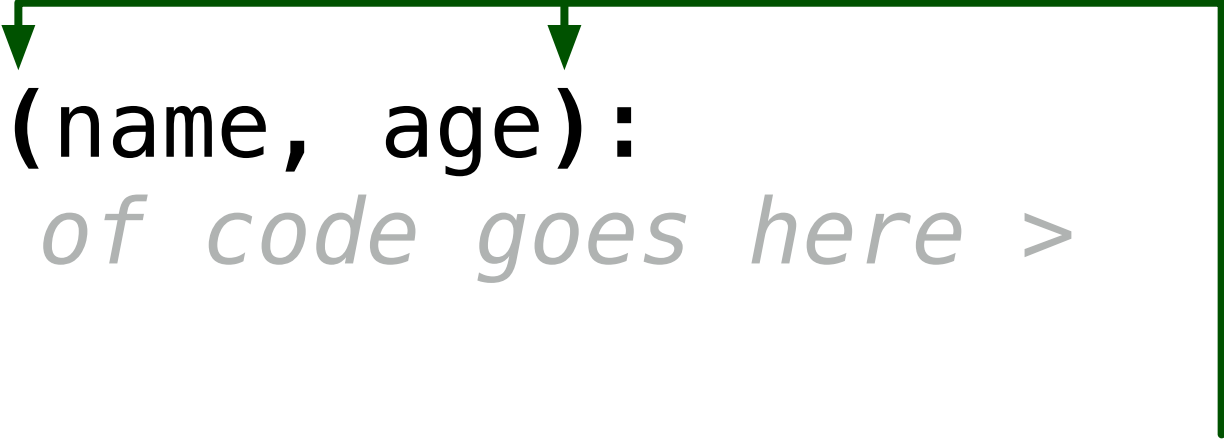
```
def monkey(name, age):
    < suite of code goes here >
```

The first item in the function definition after the 'def' keyword is the *name of the function we are defining.* In this case the function is artfully called 'monkey'.

(parentheses)

```
def monkey(name, age):
     < suite of code goes here >
```

Just after the function name 'monkey' we must
have a pair of parentheses that contain
parameter names. Even if your function doesn't
need to take any parameters, you still need to
have them. For example, "def foo():" is the
start of a function def that doesn't take
parameters.

```python
def monkey(name, age):
    < suite of code goes here >
```

Inside those much-needed parentheses, we put
our comma separated list of parameters. In
this case, we're going to do something with
this particular monkey's *name* and *age*. We
could name these anything we want, but since
we're good programmers, we will make them give
us a clue about how they are used.

colon:

```
def monkey(name, age):
    < suite of code goes here >
```

After the training parenthesis, we have to
have a colon. This tells python we're done
with this line and (on the next line) we're
going to start looking at the code that the
function will run.

function body

All of this gray text is the function body. It
is executed when we call 'monkey'.

```
def monkey(name, age):        ✓
    print "Monkey name: " + name
    if name == "Rufus":
        print "Great name, Rufus!"
    elif name == "Sputnik":
        print "Same as my dog."
    print "Monkey's age: " + str(age)
    if age < 4:
        print "Monkey is young."
```
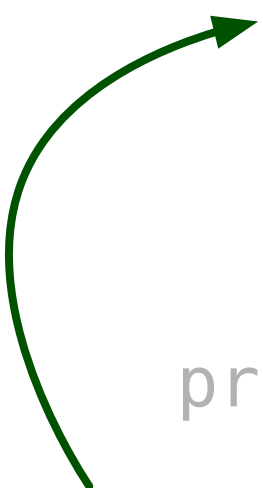
indentation matters

In Python, we use indentation to indicate which lines of code are grouped together. Notice how the 'def' is on its own indentation level, but all the code in the function is indented *at least* one level to the right?

```python
def monkey(name, age):
    print "Monkey name: " + name
    if name == "Rufus":
        print "Great name, Rufus!"
    elif name == "Sputnik":
        print "Same as my dog."
    print "Monkey's age: " + str(age)
    if age < 4:
        print "Monkey is young."
```

indentation matters

```python
def monkey(name, age):
    print "Monkey name: " + name
    if name == "Rufus":
        print "Great name, Rufus!"
    elif name == "Sputnik":
        print "Same as my dog."
    print "Monkey's age: " + str(age)
    if age < 4:
        print "Monkey is young."
print "I'm not in the monkey function."
```

All of the stuff to the right of the dashed line is the suite of code for 'monkey'. The print statement afterwards is **not** *because it is at the same indentation as the 'def' keyword.*
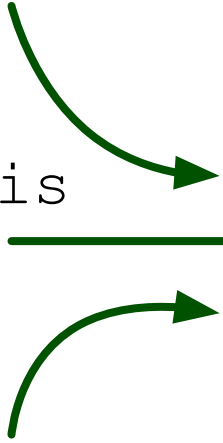
indentation matters

if clause starts here

one line is indented

this is **not** part of the if clause code because it is at same indentation as the *if* keyword.

```python
def monkey(name, age):
    print "Monkey name: " + name
    if name == "Rufus":
        print "Great name, Rufus!"
    elif name == "Sputnik":
        print "Same as my dog."
    print "Monkey's age: " + str(age)
    if age < 4:
        print "Monkey is young."
```

```
numbers = [5, 9, 10, 2]
squares = [ ]
for x in numbers:
    sq = x * x
    print "x squared: " + str(sq)
    squares.append(sq)
print "squares: " + str(squares)
```

This prints out:

```
x squared: 25
x squared: 81
x squared: 100
x squared: 4
squares: [25, 81, 100, 4]
```

The 'for' keyword tells Python we are about
to do a loop. This kind of loop needs two
things: something to loop through, and a name
to give one particular item in the list.

```
for x in numbers:
    <suite of code>
```

give iteration variable a name

We need to give a name to one particular element in our list. In this case we'll call it 'x'.

```
         ↓
  for x in numbers:
      <suite of code>
```

The first time through the loop, 'x' is the first item in 'numbers'. The second time, it is the second item in 'numbers'. And so on, until we run out of numbers.

We tell Python which list we want to iterate through by naming it here, at the end, but before the colon.

for x in **numbers:**
        <suite of code>

This can actually be anything that we can iterate over, but we've only seen lists so far. The 'range' function gives a list, remember.

To get going on the for loop after we've said what we're iterating over, and what we call the individual value each time through the loop, put a colon.

```
for x in numbers:
    <suite of code>
```

Just like with a function. Or an if. Or an elif. Or many many other things in Python.

code to run each iteration

Last, we have a suite of code starting on the line after the colon. This suite is indented at least one more than the 'for' keyword that started our loop definition.

```
for x in numbers:
    <suite of code>
```

This is exactly like how the 'monkey' function's suite was. It has to be indented at least one more than the keyword that started it. When we see code that is indented at (or less than) the 'for' keyword, that's not part of the loop.

look at a loop in context

```
1  numbers = [5, 9, 10, 2]
2  squares = [ ]
3  for x in numbers:
4      sq = x * x
5      print "x squared: " + str(sq)
6      squares.append(sq)
7  print "squares: " + str(squares)
```

When we run this, we progress from line to line, but in a loop, we might go back to the top of the loop if we're not done. The sequence of execution in this case is:

Lines:
1, 2, 3, 4, 5, 6, 4, 5, 6, 4, 5, 6, 4, 5, 6, 7

Loop iter.   first   second   third   last
              x=5   x=9   x=10   x=2