Writing Basic Functions

- A simple function can be constructed as follows:
-
- You decide on the name of the function. The function command shows R that you are writing a function. Inside the parenthesis you outline the input objects required and decide what to call them. The commands occur inside the . The name of whatever output you want goes at the end of the function.

More Complex Functions

The following function returns several values in the form of a list:

```
myfunc <- function(x)
{ # x is expected to be a numeric vector # function returns
mean, sd, min, and max of the vector x the.mean <- mean(x)
<- sd(x)
the.min <- min(x)
the.max <- max(x)
return(list(average=the.mean,stand.dev=the.sd,minimum=the.m
maximum=the.max)) }
```

Argument Matching

- How does *R* know which arguments are which?
- It uses argument matching.
- Argument matching is done in a few different ways.
  - The arguments are matched by their positions. The first supplied argument is matched to the first formal argument and so on, e.g. when writing sf2 we specified that a1 comes first, a2 second and a3 third. Using sf2(2, 3, 4) assigns 2 to a1, 3 to a2 and 4 to a3.
  - The arguments are matched by name. A named argument is matched to the formal argument with the same name, e.g. sf2 arguments have names a1, a2 and a3.
  - Can do things like sf2(a1=2, a3=3, a2=4), sf2(a3=2, a1=3, a2=4), etc.
  - Name matching happens first, then positional matching is used for any unmatched arguments.

Default values

We can also give some/all arguments default values.

```
mypower <- function(x, pow=2)
{
x^pow
}
```

If a value for the argument 'pow' is not specified in the function call, a value of 2 is used.

```
mypower(4)
[1] 16
```

If a value for pow is specified, that value is used.

```
mypower(4, 3)  #x=4, pow=3
[1] 64
```