HOPR

Michael Whitfield

28 Oct 2024

Good names	Names that cause errors
a	1trial
b	$
FOO	^mean
my_var	2nd
.day	!bad

Capitalization

R is case-sensitive, so `name` and `Name` will refer to different objects:

```
Name <- 1
name <- 0

Name + 1
## 2
```

Finally, R will overwrite any previous information stored in an object without asking you for permission. So, it is a good idea to *not* use names that are already taken:

```
my_number <- 1
my_number
## 1

my_number <- 999
my_number
## 999
```

If you give R two vectors of unequal lengths, R will repeat the shorter vector until it is as long as the longer vector, and then do the math, as shown in Figure 2.4. This isn't a permanent change–the shorter vector will be its original size after R does the math. If the length of the short vector does not divide evenly into the length of the long vector, R will return a warning message. This behavior is known as *vector recycling*, and it helps R do element-wise operations:

```
1:2
## 1 2

1:4
## 1 2 3 4

die
## 1 2 3 4 5 6

die + 1:2
## 2 4 4 6 6 8

die + 1:4
## 2 4 6 8 6 8
Warning message:
In die + 1:4 :
  longer object length is not a multiple of shorter object length
```

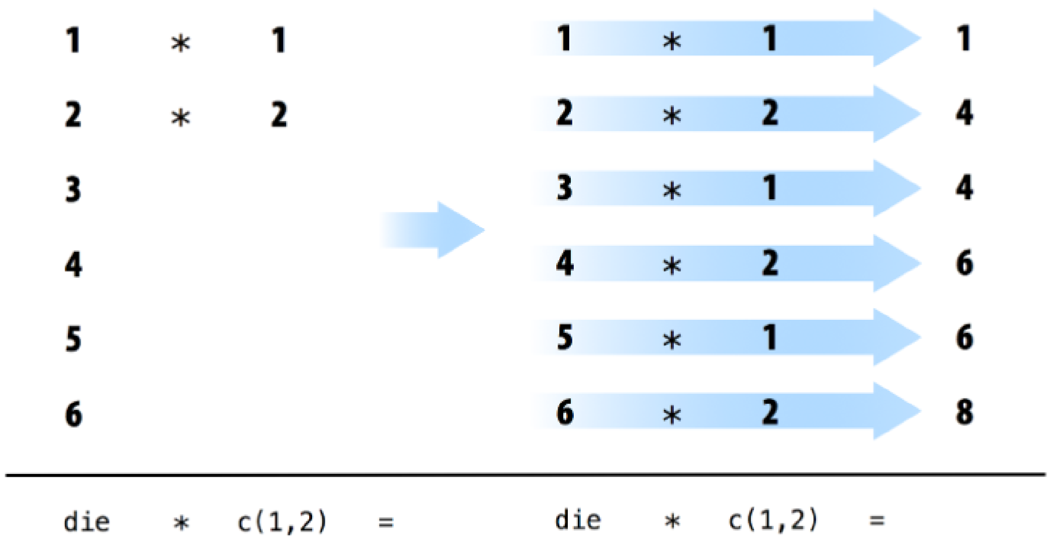


Figure 2.4: R will repeat a short vector to do element-wise operations with two vectors of uneven lengths.

Element-wise operations are a very useful feature in R because they manipulate groups of values in an orderly way. When you start working with data sets, element-wise operations will ensure that values from one observation or case are only paired with values from the same observation or case. Element-wise operations also make it easier to write your own programs and functions in R.

But how do you know which argument names to use? If you try to use a name that a function does not expect, you will likely get an error:

```
round(3.1415, corners = 2)
## Error in round(3.1415, corners = 2) : unused argument(s) (corners = 2)
```

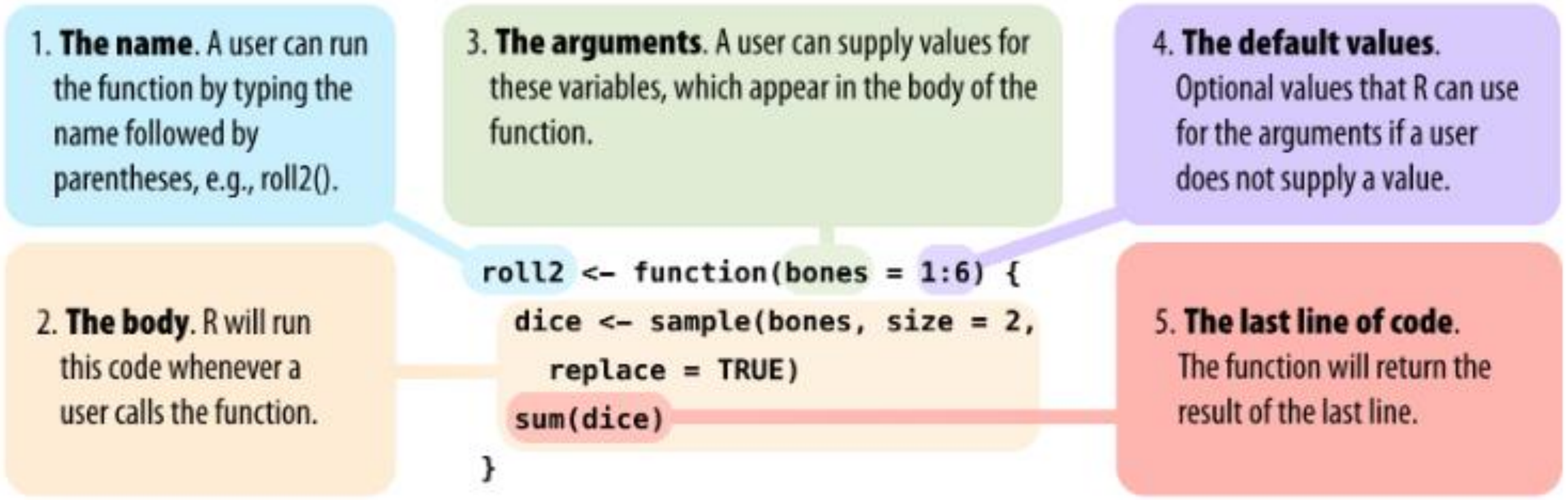
If you're not sure which names to use with a function, you can look up the function's arguments with `args`. To do this, place the name of the function in the parentheses behind `args`. For example, you can see that the `round` function takes two arguments, one named `x` and one named `digits`:

```
args(round)
## function (x, digits = 0)
## NULL
```

Did you notice that `args` shows that the `digits` argument of `round` is already set to 0? Frequently, an R function will take optional arguments like `digits`. These arguments are considered optional because they come with a default value. You can pass a new value to an optional argument if you want, and R will use the default value if you do not. For example, `round` will round your number to 0 digits past the decimal point by default. To override the default, supply your own value for `digits`:

```
round(3.1415)
## 3

round(3.1415, digits = 2)
## 3.14
```


1. The name. A user can run the function by typing the name followed by parentheses, e.g., roll2().
3. The arguments. A user can supply values for these variables, which appear in the body of the function.
4. The default values. Optional values that R can use for the arguments if a user does not supply a value.
2. The body. R will run this code whenever a user calls the function.
5. The last line of code. The function will return the result of the last line.

```r
roll2 <- function(bones = 1:6) {
  dice <- sample(bones, size = 2,
    replace = TRUE)
  sum(dice)
}
```

Figure 2.6: Every function in R has the same parts, and you can use function to create these parts. Assign the result to a name, so you can call the function later.

Syntax question – what's the difference?

- a = 12
- a <- 12
- # function to sum the values