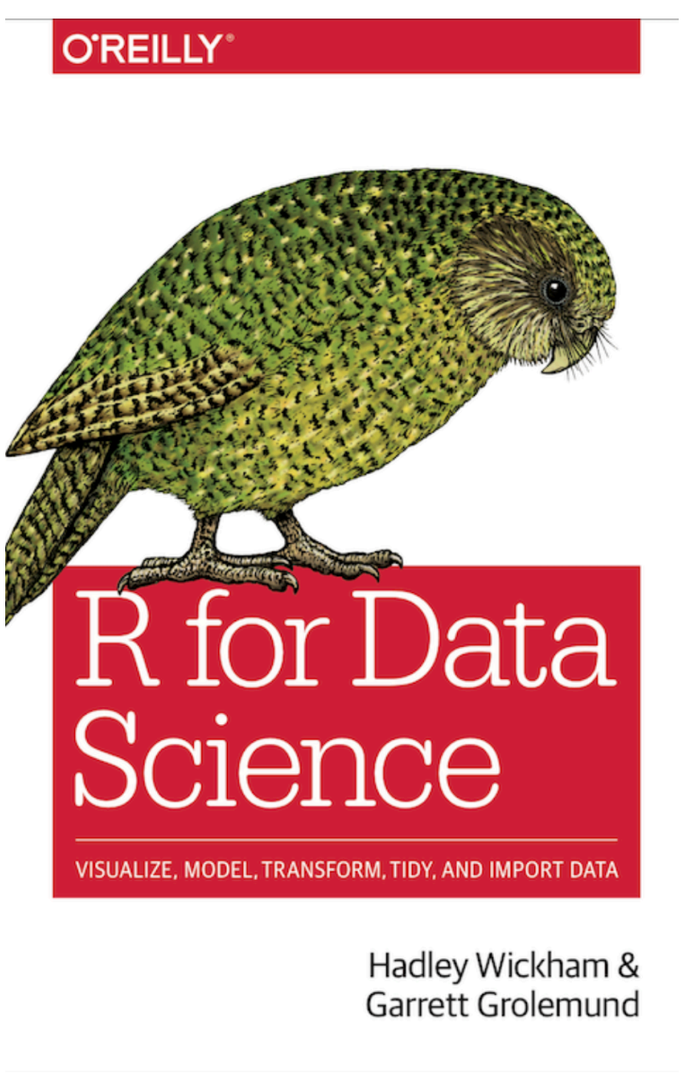
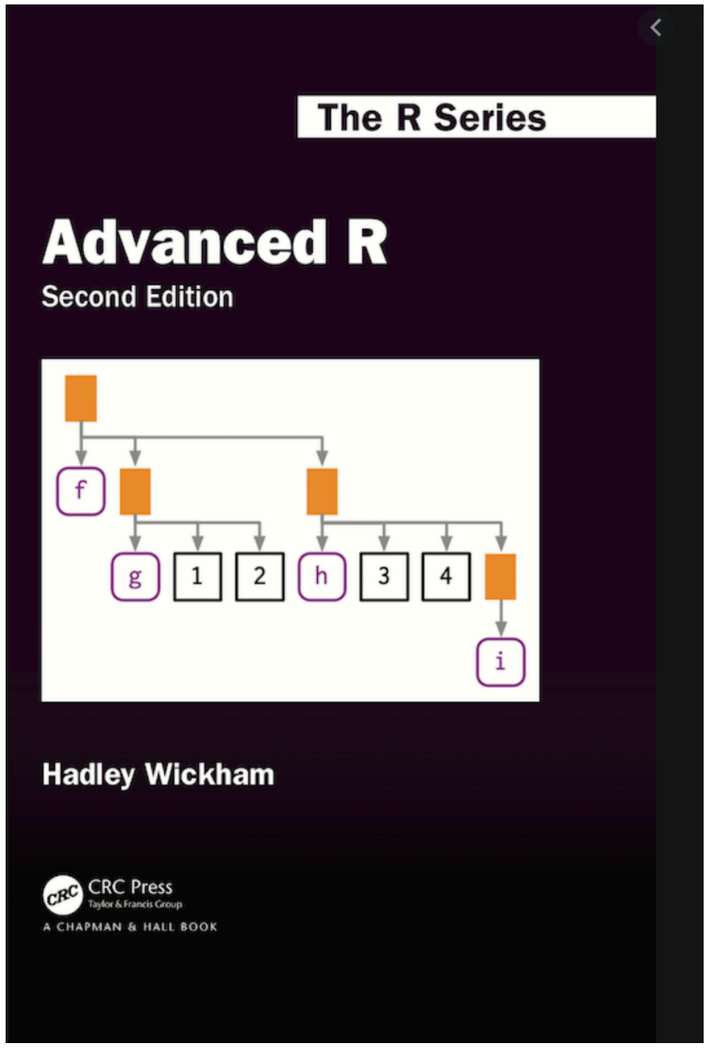
Functions


O'REILLY
R for Data Science
VISUALIZE, MODEL, TRANSFORM, TIDY, AND IMPORT DATA
Hadley Wickham &
Garrett Grolemund


The R Series
Advanced R
Second Edition
f
g
1
2
h
3
4
i
Hadley Wickham
CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Outline

Function components

Outline

Function components

body(), formals(), environment()

Outline

Function components

body(), formals(), environment()

Arguments

Outline

Function components

body(), formals(), environment()

Arguments

Intro to scoping

What is R?

R is an implementation of the S programming language, which was created 1976 by John Chambers at Bell Labs

R was created by Ross Ihaka and Robert Gentleman in 2000

Although R and S are slightly different, they share the same design principle:

> “[W]e wanted users to be able to begin in an interactive environment, where they did not consciously think of themselves as programming. Then as their needs became clearer and their sophistication increased, they should be able to slide gradually into programming, when the language and system aspects would become more important.”

R’s basic rules:

- Everything that exists is an object
- Everything that happens is a function call

Function Basics

What is a function?

Function Basics

What is a function?

A function is an operator that takes in some objects and then returns some object

Function Basics

What is a function?

A function is an operator that takes in some objects and then returns some object

```
2+2
```

Function Basics

What is a function?

A function is an operator that takes in some objects and then returns some object

```
2+2
```

```
sum(c(1,2,3,4,5))
```

Function Basics

What is a function?

A function is an operator that takes in some objects and then returns some object

```
2+2
```

```
sum(c(1,2,3,4,5))
```

```
f <- function(x) {
x^3
}
```

```
f(4)
```

Function Basics

Defining a function:

Function Basics

Defining a function:

```
f <- function(x) {
x^3
}
```

Function Basics

Defining a function:

```
f <- function(x) {
x^3
}
```

Function parts:

Function Basics

Defining a function:

```
f <- function(x) {
x^3
}
```

Function parts:

```
formals(f)
```

```
body(f)
```

```
environment(f)
```

Function Basics

Defining a function:

```
f <- function(x) {
x^3
}
```

How does R see a function?

Function parts:

```
formals(f)
```

```
body(f)
```

```
environment(f)
```

Function Basics

Defining a function:

```
f <- function(x) {
x^3
}
```

How does R see a function?

```
str(f)
```

```
attributes(f)
```

Function parts:

```
formals(f)
```

```
body(f)
```

```
environment(f)
```

Function Basics

There are special functions that don’t behave like the others! They are called primitives.

Function Basics

There are special functions that don't behave like the others! They are called primitives.

Ex: the `sum()` function

Function Basics

There are special functions that don't behave like the others! They are called primitives.

Ex: the `sum()` function

```
sum
> function (..., na.rm = FALSE)  .Primitive("sum")
```

Function Basics

There are special functions that don't behave like the others! They are called primitives.

Ex: the `sum()` function

```
sum
> function (..., na.rm = FALSE)  .Primitive("sum")
```

`body(sum)` ⟶ `NULL`

Function Basics

There are special functions that don't behave like the others! They are called primitives.

Ex: the `sum()` function

```
sum
> function (..., na.rm = FALSE)  .Primitive("sum")
```

```
body(sum) ⟶ NULL
formals(sum) ⟶ NULL
```

Function Basics

There are special functions that don't behave like the others! They are called primitives.

Ex: the `sum()` function

```
sum
> function (..., na.rm = FALSE)  .Primitive("sum")
```

```
body(sum) ⟶ NULL
formals(sum) ⟶ NULL
environment(sum) ⟶ NULL
```

Function Arguments

Understanding the arguments of a function is very important to grasping R

Function Arguments

Understanding the arguments of a function is very important to grasping R

```
ggplot(data, mapping = aes(), ..., environment = parent.frame())
```

Function Arguments

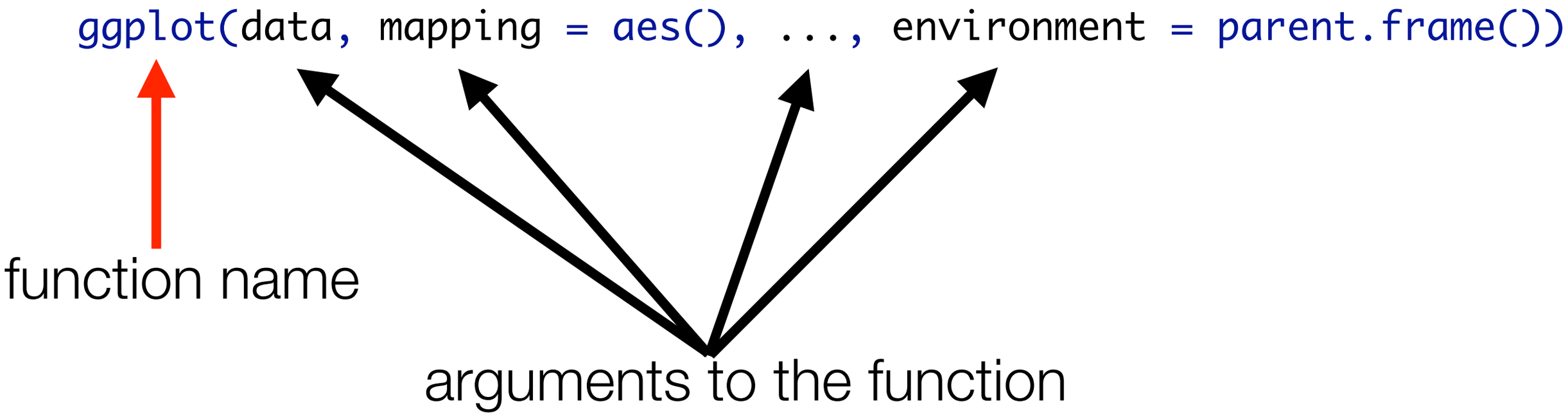
Understanding the arguments of a function is very important to grasping R

```
ggplot(data, mapping = aes(), ..., environment = parent.frame())
```

function name

Function Arguments

Understanding the arguments of a function is very important to grasping R



Function Arguments

Required vs. Defaulted Arguments

Function Arguments

Required vs. Defaulted Arguments

Required arguments need to be supplied at run time for function to run

Function Arguments

Required vs. Defaulted Arguments

Required arguments need to be supplied at run time for function to run

Unless otherwise specified, defaulted arguments will default to the value(s) set in the function.

Function Arguments

Required vs. Defaulted Arguments

Required arguments need to be supplied at run time for function to run

Unless otherwise specified, defaulted arguments will default to the value(s) set in the function.

```
ggplot(data, mapping = aes(), ..., environment = parent.frame())
```

Function Arguments

Required vs. Defaulted Arguments

Required arguments need to be supplied at run time for function to run

Unless otherwise specified, defaulted arguments will default to the value(s) set in the function.

```
ggplot(data, mapping = aes(), ..., environment = parent.frame())
```

required argument

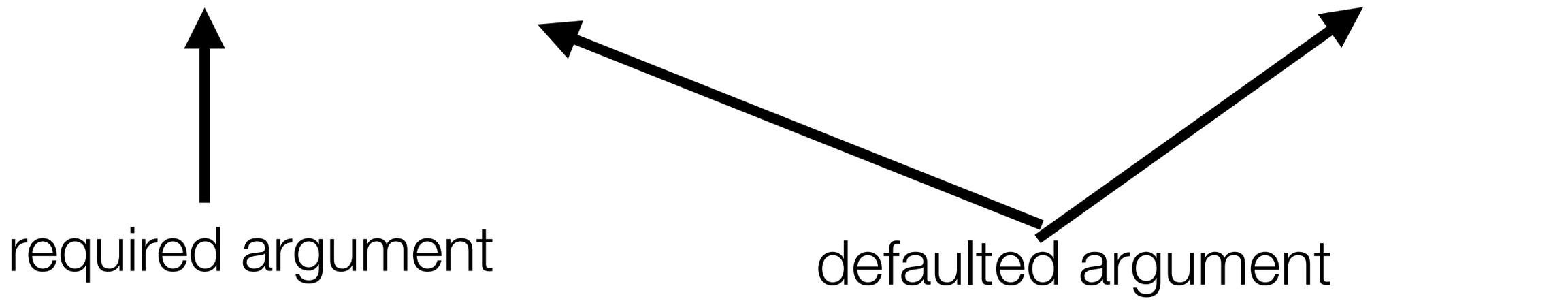
Function Arguments

Required vs. Defaulted Arguments

Required arguments need to be supplied at run time for function to run

Unless otherwise specified, defaulted arguments will default to the value(s) set in the function.

```
ggplot(data, mapping = aes(), ..., environment = parent.frame())
```



Function Arguments

Ex: simulating draws from a normal distribution:

Function Arguments

Ex: simulating draws from a normal distribution:

Done with the rnorm() function

Function Arguments

Ex: simulating draws from a normal distribution:

Done with the rnorm() function

```
rnorm(10)
```

```
[1]  0.7002779  1.0693103  0.5427495  0.3072563  1.2381663
[6] 1.1809495 -0.8345444 -0.8011416  0.4679774  0.6066023
```

Function Arguments

Ex: simulating draws from a normal distribution:

Done with the rnorm() function

```
rnorm(10)
```

```
[1]  0.7002779  1.0693103  0.5427495  0.3072563  1.2381663
[6] 1.1809495 -0.8345444 -0.8011416  0.4679774  0.6066023
```

Anything weird? What kind of normal did it sample from?

Function Arguments

Ex: simulating draws from a normal distribution:

Done with the rnorm() function

```
rnorm(10)
```

```
[1]  0.7002779  1.0693103  0.5427495  0.3072563  1.2381663
[6] 1.1809495 -0.8345444 -0.8011416  0.4679774  0.6066023
```

Anything weird? What kind of normal did it sample from?

```
rnorm(n, mean = 0, sd = 1)
```

Function Arguments

Ex: simulating draws from a normal distribution:

Done with the rnorm() function

```
rnorm(10)
```

```
[1]  0.7002779  1.0693103  0.5427495  0.3072563  1.2381663
[6] 1.1809495 -0.8345444 -0.8011416  0.4679774  0.6066023
```

Anything weird? What kind of normal did it sample from?

```
rnorm(n, mean = 0, sd = 1)
```

n is required because there is no way of knowing how many draws you want.

Function Arguments

Ex: simulating draws from a normal distribution:

Done with the rnorm() function

```
rnorm(10)
```

```
[1]  0.7002779  1.0693103  0.5427495  0.3072563  1.2381663
[6] 1.1809495 -0.8345444 -0.8011416  0.4679774  0.6066023
```

Anything weird? What kind of normal did it sample from?

```
rnorm(n, mean = 0, sd = 1)
```

n is required because there is no way of knowing how many draws you want.

mean and standard deviation are defaulted to be the standard normal distribution $N(0,1)$

Function Arguments

R documentation can help you determine which arguments are required and which ones are defaulted:

Function Arguments

R documentation can help you determine which arguments are required and which ones are defaulted:

Normal {stats} R Documentation

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

Usage

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

Function Arguments

How are arguments matched in R?

Function Arguments

How are arguments matched in R?

```
f <- function(location, scale, shape) {
        c(location, scale, shape)
}
```

Function Arguments

How are arguments matched in R?

```
f <- function(location, scale, shape) {
        c(location, scale, shape)
}
```

How will the following behave?

Function Arguments

How are arguments matched in R?

```
f <- function(location, scale, shape) {
        c(location, scale, shape)
}
```

How will the following behave?

```
f(1,2,3)
```

Function Arguments

How are arguments matched in R?

```
f <- function(location, scale, shape) {
        c(location, scale, shape)
}
```

How will the following behave?

```
f(1,2,3)
```

```
[1] 1 2 3
```

Function Arguments

How are arguments matched in R?

```
f <- function(location, scale, shape) {
        c(location, scale, shape)
}
```

How will the following behave?

```
f(1,2,3)
```

```
[1] 1 2 3
```

```
f(scale = 1, 3, 2)
```

Function Arguments

How are arguments matched in R?

```
f <- function(location, scale, shape) {
        c(location, scale, shape)
}
```

How will the following behave?

```
f(1,2,3)
```

```
[1] 1 2 3
```

```
f(scale = 1, 3, 2)
```

```
[1] 3 1 2
```

Function Arguments

How are arguments matched in R?

```
f <- function(location, scale, shape) {
        c(location, scale, shape)
}
```

How will the following behave?

```
f(1,2,3)
```

```
[1] 1 2 3
```

```
f(scale = 1, 3, 2)
```



```
[1] 3 1 2
```

```
f(0, l = 3, scale = 2)
```

Function Arguments

How are arguments matched in R?

```
f <- function(location, scale, shape) {
        c(location, scale, shape)
}
```

How will the following behave?

```
f(1,2,3)
```

```
[1] 1 2 3
```

```
f(scale = 1, 3, 2)
```

```
[1] 3 1 2
```

```
f(0, l = 3, scale = 2)
```

```
[1] 3 2 0
```

Function Arguments

R's argument name matching rules:

1. By exact name
2. By partial name
3. By Position

```
f <- function(location, scale, shape) {
       c(location, scale, shape)
}
```

```
f(1,2,3)
```

```
[1] 1 2 3
```

```
f(scale = 1, 3, 2)
```

```
[1] 3 1 2
```

```
f(0, l = 3, scale = 2)
```

```
[1] 3 2 0
```

The … Argument

The … argument can mean two things in a function call

The … Argument

The … argument can mean two things in a function call

1. You are passing arguments to a function further down the line in the function
2. The function accepts a varying # of arguments

The … Argument

The … argument can mean two things in a function call

1. You are passing arguments to a function further down the line in the function
2. The function accepts a varying # of arguments

```
f <- function(x, y,...) {
x + y + rnorm(1, ...)
}
```

The … Argument

The … argument can mean two things in a function call

1. You are passing arguments to a function further down the line in the function
2. The function accepts a varying # of arguments

```
f <- function(x, y,...) {
       x + y + rnorm(1, ...)
       }
f(1,2, mean = 100, sd = 5)
```

The … Argument

The … argument can mean two things in a function call

1. You are passing arguments to a function further down the line in the function
2. The function accepts a varying # of arguments

```
f <- function(x, y,...) {
      x + y + rnorm(1, ...)
      }
```

```
f(1,2, mean = 100, sd = 5)
```

```
[1] 105.6643
```

The … Argument

The … argument can mean two things in a function call

1. You are passing arguments to a function further down the line in the function
2. The function accepts a varying # of arguments

```
f <- function(x, y,...) {
     x + y + rnorm(1, ...)
     }
```

```
f(1,2, mean = 100, sd = 5)
```

```
[1] 105.6643
```

```
sum(1,2,3,4,5,6,7,8,9,10, 15:20)
```

The … Argument

The … argument can mean two things in a function call

1. You are passing arguments to a function further down the line in the function
2. The function accepts a varying # of arguments

```
f <- function(x, y,...) {
     x + y + rnorm(1, ...)
     }
```

```
f(1,2, mean = 100, sd = 5)
```

→ `[1] 105.6643`

```
sum(1,2,3,4,5,6,7,8,9,10, 15:20)
```

→ `[1] 160`

Scoping in R

Scoping defines the range of functionality for a variable

Scoping in R

Scoping defines the range of functionality for a variable

R primarily performs lexical scoping

Scoping in R

Scoping defines the range of functionality for a variable

R primarily performs lexical scoping

Understanding the scoping rules will help make your R experience much more enjoyable!

Scoping in R

Scoping defines the range of functionality for a variable

R primarily performs lexical scoping

Understanding the scoping rules will help make your R experience much more enjoyable!

Try to think through the code below and predict the output!

Scoping in R

Scoping defines the range of functionality for a variable

R primarily performs lexical scoping

Understanding the scoping rules will help make your R experience much more enjoyable!

Try to think through the code below and predict the output!

```
x <- 5
x <- function(x) x^2
f <- function(x) x(x)
x <- function(x) x - 1
f(2)
```

Scoping in R

Scoping defines the range of functionality for a variable

R primarily performs lexical scoping

Understanding the scoping rules will help make your R experience much more enjoyable!

Try to think through the code below and predict the output!

```
x <- 5
x <- function(x) x^2
f <- function(x) x(x)
x <- function(x) x - 1
f(2)
```

```
[1] 1
```

lexical scoping

R has a systematic set of rules for looking up the values of variables that may be defined differently in different places

Real world analogy Problem : I need to find my keys

1. Check the room I am currently in
2. Check the room I was in last
3. Check yesterday's pants' pockets
4. Check the living room table
5. Check the dining room table
6. Getting desperate: check the kitchen
7. Desperate : check the bathroom
8. Give up: have to stay home

R has a similar searching strategy

Credit: David Kahle

Scoping in R

R's rules

Scoping in R

R's rules

1. Look inside the function
2. Look where the function was defined
3. Look where that function was defined
4. Keep going up the "ladder" until you hit the top

Scoping in R

R's rules

1. Look inside the function
2. Look where the function was defined
3. Look where that function was defined
4. Keep going up the "ladder" until you hit the top

```
fun <- function(x) {
x + y
}
```

Scoping in R

R's rules

1. Look inside the function
2. Look where the function was defined
3. Look where that function was defined
4. Keep going up the "ladder" until you hit the top

```
fun <- function(x) {
x + y
}

fun(2)
```

Scoping in R

R's rules

1. Look inside the function
2. Look where the function was defined
3. Look where that function was defined
4. Keep going up the "ladder" until you hit the top

```
fun <- function(x) {
x + y
}

fun(2)
```

```
Error in fun(2) : object 'y' not found
```

Scoping in R

R's rules

1. Look inside the function
2. Look where the function was defined
3. Look where that function was defined
4. Keep going up the "ladder" until you hit the top

```
fun <- function(x) {
x + y
}

fun(2)
```

```
Error in fun(2) : object 'y' not found
```

```
y <- 5
```

Scoping in R

R's rules

1. Look inside the function
2. Look where the function was defined
3. Look where that function was defined
4. Keep going up the "ladder" until you hit the top

```
fun <- function(x) {
x + y
}

fun(2)
```

```
Error in fun(2) : object 'y' not found
```

```
y <- 5

fun(2)
```

Scoping in R

R's rules

1. Look inside the function
2. Look where the function was defined
3. Look where that function was defined
4. Keep going up the "ladder" until you hit the top

```
fun <- function(x) {
x + y
}

fun(2)
```

```
Error in fun(2) : object 'y' not found
```

```
y <- 5

fun(2)
```

```
[1] 7
```

Scoping in R

Fun (and dangerous!) example:

Scoping in R

Fun (and dangerous!) example:

R relies on scoping for everything!

Scoping in R

Fun (and dangerous!) example:

R relies on scoping for everything!

```
`+` <- function(x,y) {
stop("I will never add again!")
}
```

Scoping in R

Fun (and dangerous!) example:

R relies on scoping for everything!

```
`+` <- function(x,y) {
stop("I will never add again!")
}
1+2
```

Scoping in R

Fun (and dangerous!) example:

R relies on scoping for everything!

```
`+` <- function(x,y) {
stop("I will never add again!")
}
```

```
1+2
```

```
Error in 1 + 2 : I will never add again!
```

Scoping in R

Fun (and dangerous!) example:

R relies on scoping for everything!

```
`+` <- function(x,y) {
stop("I will never add again!")
}
```

```
1+2
```

```
Error in 1 + 2 : I will never add again!
```

```
sum(1:2)
```

Scoping in R

Fun (and dangerous!) example:

R relies on scoping for everything!

```
`+` <- function(x,y) {
stop("I will never add again!")
}
```

```
1+2
```

```
Error in 1 + 2 : I will never add again!
```

```
sum(1:2)
```

```
`+` <- function(e1, e2){
if(runif(1) < 0.1){
sum(c(e1, e2, 1))
} else {
sum(c(e1, e2))
}
}
replicate(100, 1 + 1)
```