INTERMEDIATE R

Relational Operators

Equality ==

```
> TRUE == TRUE
[1] TRUE

> TRUE == FALSE
[1] FALSE
```

```
> "hello" == "goodbye"
[1] FALSE
```

```
> 3 == 2
[1] FALSE
```

Inequality !=

```
> TRUE != TRUE
[1] FALSE

> TRUE != FALSE
[1] TRUE
```

```
> "hello" != "goodbye"
[1] TRUE
```

```
> 3 != 2
[1] TRUE
```

< and >

```
> 3 < 5
[1] TRUE

> 3 > 5
[1] FALSE

> "Hello" > "Goodbye"
[1] TRUE

> TRUE < FALSE
[1] FALSE
```

Alphabetical Order!

TRUE coerces to 1
FALSE coerces to 0

<= and >=

```
> 5 >= 3
[1] TRUE

> 3 >= 3
[1] TRUE
```

Relational Operators & Vectors

```
> linkedin <- c(16, 9, 13, 5, 2, 17, 14)

> linkedin
[1] 16  9 13  5  2 17 14

> linkedin > 10
[1]  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE

> facebook <- c(17, 7, 5, 16, 8, 13, 14)

> facebook
[1] 17  7  5 16  8 13 14

> facebook <= linkedin
[1] FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE
```

Let’s practice!

INTERMEDIATE R

Logical Operators

Logical Operators

- AND operator &
- OR operator |
- NOT operator !

AND operator &

```
> TRUE & TRUE
[1] TRUE

> FALSE & TRUE
[1] FALSE

> TRUE & FALSE
[1] FALSE

> FALSE & FALSE
[1] FALSE
```

Only TRUE if both are TRUE

FALSE otherwise

AND operator &

```
> x <- 12

    TRUE      TRUE
> x > 5 & x < 15
[1] TRUE

> x <- 17

    TRUE     FALSE
> x > 5 & x < 15
[1] FALSE
```

OR operator |

```
> TRUE | TRUE
[1] TRUE

> TRUE | FALSE
[1] TRUE

> FALSE | TRUE
[1] TRUE

> FALSE | FALSE
[1] FALSE
```

TRUE if at least one is TRUE

Only FALSE if both FALSE

OR operator |

```
> y <- 4

   TRUE   FALSE
> y < 5 | y > 15
[1] TRUE

> y <- 14

  FALSE   FALSE
> y < 5 | y > 15
[1] FALSE
```

NOT operator !

```
> !TRUE
[1] FALSE

> !FALSE
[1] TRUE

> !(x < 5)
> x >= 5
```

```
> is.numeric(5)
[1] TRUE

> !is.numeric(5)
[1] FALSE

> is.numeric("hello")
[1] FALSE

> !is.numeric("hello")
[1] TRUE
```

Logical Operators & Vectors

```
> c(TRUE, TRUE, FALSE) & c(TRUE, FALSE, FALSE)
[1]  TRUE FALSE FALSE

> c(TRUE, TRUE, FALSE) | c(TRUE, FALSE, FALSE)
[1]  TRUE  TRUE FALSE

> !c(TRUE, TRUE, FALSE)
[1] FALSE FALSE  TRUE
```

& vs &&, | vs ||

```
> c(TRUE, TRUE, FALSE) & c(TRUE, FALSE, FALSE)
[1]  TRUE FALSE FALSE

> c(TRUE, TRUE, FALSE) && c(TRUE, FALSE, FALSE)
[1] TRUE

> c(TRUE, TRUE, FALSE) | c(TRUE, FALSE, FALSE)
[1]  TRUE  TRUE FALSE

> c(TRUE, TRUE, FALSE) || c(TRUE, FALSE, FALSE)
[1] TRUE
```

Let’s practice!

INTERMEDIATE R

Conditional Statements

if statement

```
if(condition) {
  expr
}
```

```
> x <- -3

> if(x < 0) {
    print("x is a negative number")
  }
[1] "x is a negative number"
```

if statement

```
if(condition) {
  expr
}
```

```
> x <- 5
     FALSE
> if(x < 0) {
    print("x is a negative number") Not executed
  }
  No printout!
```

else statement

```
if(condition) {
  expr1
} else {
  expr2
}
```

else statement

```
if(condition) {
  expr1
} else {
  expr2
}
```

```
> x <- -3
     TRUE
> if(x < 0) {
    print("x is a negative number")
  } else {
    print("x is either a positive number or zero")
  }
[1] "x is a negative number"
```

else statement

```
if(condition) {
  expr1
} else {
  expr2
}
```

```
> x <- 5
     FALSE
> if(x < 0) {
    print("x is a negative number")
  } else {
    print("x is either a positive number or zero")
  }
[1] "x is either a positive number or zero"
```

else if statement

```
if(condition1) {
  expr1
} else if(condition2) {
  expr2
} else {
  expr3
}
```

else if statement

```
> x <- -3
     TRUE
> if(x < 0) {
    print("x is a negative number")
  } else if(x == 0) {
    print("x is zero")
  } else {
    print("x is a positive number")
  }
[1] "x is a negative number"
```

```
if(condition1) {
  expr1
} else if(condition2) {
  expr2
} else {
  expr3
}
```

else if statement

```
> x <- 0
      FALSE
> if(x < 0) {
    print("x is a negative number")
  } else if(x == 0) { TRUE
    print("x is zero")
  } else {
    print("x is a positive number")
  }
[1] "x is zero"
```

```
if(condition1) {
  expr1
} else if(condition2) {
  expr2
} else {
  expr3
}
```

else statement

```
> x <- 5

      FALSE
> if(x < 0) {
    print("x is a negative number")
  } else if(x == 0) { FALSE
    print("x is zero")
  } else {
    print("x is a positive number")
  }
[1] "x is a positive number"
```

```
if(condition1) {
  expr1
} else if(condition2) {
  expr2
} else {
  expr3
}
```

if, else if, else

```
> x <- 6
         TRUE
> if(x %% 2 == 0) {
    print("divisible by 2")
X } else if(x %% 3 == 0) {
X   print("divisible by 3")
X } else {
X   print("not divisible by 2 nor by 3...")
X }
[1] "divisible by 2"
```

Let’s practice!

INTERMEDIATE R

while loop

while loop

```
while(condition) {
  expr
}
```

```
> ctr <- 1

> while(ctr <= 7) {
```

while loop

```
while(condition) {
  expr
}
```

```
> ctr <- 1

> while(ctr <= 7) {
    print(paste("ctr is set to", ctr))
```

while loop

```
while(condition) {
  expr
}
```

```
> ctr <- 1

> while(ctr <= 7) {
    print(paste("ctr is set to", ctr))
    ctr <- ctr + 1
  }
```

while loop

```
while(condition) {
  expr
}
```

```
> ctr <- 1                                                  ctr: 1
              TRUE
> while(ctr <= 7) {
    print(paste("ctr is set to", ctr))
    ctr <- ctr + 1          increment ctr
  }
[1] "ctr is set to 1"
```

while loop

```
while(condition) {
  expr
}
```

```
> ctr <- 1                                            ctr: 2
          TRUE
> while(ctr <= 7) {
    print(paste("ctr is set to", ctr))
    ctr <- ctr + 1     increment ctr
  }
[1] "ctr is set to 2"
```

while loop

```
while(condition) {
  expr
}
```

ctr: 7

TRUE

```
> ctr <- 1

> while(ctr <= 7) {
    print(paste("ctr is set to", ctr))
    ctr <- ctr + 1
  }
[1] "ctr is set to 7"
```

increment ctr

while loop

```
while(condition) {
  expr
}
```

ctr: 8

```
> ctr <- 1
```

FALSE

```
> while(ctr <= 7) {
    print(paste("ctr is set to", ctr))
    ctr <- ctr + 1
  }
```

No printout!

while loop

```
> ctr <- 1

> while(ctr <= 7) {
    print(paste("ctr is set to", ctr))
    ctr <- ctr + 1
  }
[1] "ctr is set to 1"
[1] "ctr is set to 2"
...
[1] "ctr is set to 7"

> ctr
[1] 8
```

infinite while loop

```
> ctr <- 1

> while(ctr <= 7) {
    print(paste("ctr is set to", ctr))
    ctr <- ctr + 1
  }
[1] "ctr is set to 1"
[1] "ctr is set to 1"
[1] "ctr is set to 1"
[1] "ctr is set to 1"
[1] "ctr is set to 1"
[1] "ctr is set to 1"
[1] "ctr is set to 1"
...
```

break statement

```
> ctr <- 1
> while(ctr <= 7) {
    if(ctr %% 5 == 0) {
      break
    }
    print(paste("ctr is set to", ctr))
    ctr <- ctr + 1
  }
[1] "ctr is set to 1"
[1] "ctr is set to 2"
[1] "ctr is set to 3"
[1] "ctr is set to 4"
```

TRUE

Break if ctr is a 5-fold

while loop stops if ctr is 5: no more printouts

Let’s practice!

INTERMEDIATE R

for loop

for loop

```
for(var in seq) {
  expr
}
```

```
> cities <- c("New York", "Paris",
              "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> cities
[1] "New York"   "Paris"  ...  "Cape Town"
```

for loop

```
for(var in seq) {
  expr
}
```

```
> cities <- c("New York", "Paris",
              "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> for(var in seq) {
    expr
  }
```

for loop

```
for(var in seq) {
  expr
}
```

```
> cities <- c("New York", "Paris",
              "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> for(city in cities) {
    expr
  }
```

for loop

```
for(var in seq) {
  expr
}
```

```
> cities <- c("New York", "Paris",
              "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> for(city in cities) {
    print(city)
  }
```

for loop

```
for(var in seq) {
  expr
}
```

```
> cities <- c("New York", "Paris",
              "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> for(city in cities) {
    print(city)
  }
[1] "New York"
```

city: "New York"

for loop

```
for(var in seq) {
  expr
}
```

```
> cities <- c("New York", "Paris",
              "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> for(city in cities) {
    print(city)
  }
[1] "Paris"
```

city: "Paris"

for loop

```
> cities <- c("New York", "Paris",
              "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> for(city in cities) {
    print(city)
  }
[1] "New York"
[1] "Paris"
[1] "London"
[1] "Tokyo"
[1] "Rio de Janeiro"
[1] "Cape Town"
```

for loop over list

```
> cities <- list("New York", "Paris",
                 "London", "Tokyo",
                 "Rio de Janeiro", "Cape Town")

> for(city in cities) {
    print(city)
  }
[1] "New York"
[1] "Paris"
[1] "London"
[1] "Tokyo"
[1] "Rio de Janeiro"
[1] "Cape Town"
```

break statement

```
> cities <- list("New York", "Paris",
                 "London", "Tokyo",
                 "Rio de Janeiro", "Cape Town")

> for(city in cities) {
    if(nchar(city) == 6) {
      break
    }
    print(city)
  }
```

break statement

```
> cities <- list("New York", "Paris",
                 "London", "Tokyo",
                 "Rio de Janeiro", "Cape Town")

> for(city in cities) {
    if(nchar(city) == 6) {
      break
    }
    print(city)
  }
[1] "New York"
```

city: "New York"

FALSE

break statement

```
> cities <- list("New York", "Paris",
                 "London", "Tokyo",
                 "Rio de Janeiro", "Cape Town")

> for(city in cities) {                        city: "Paris"
    if(nchar(city) == 6) {  FALSE
      break
    }
    print(city)
  }
[1] "Paris"
```

break statement

```
> cities <- list("New York", "Paris",
                 "London", "Tokyo",
                 "Rio de Janeiro", "Cape Town")

> for(city in cities) {
    if(nchar(city) == 6) {
      break
    }
    print(city)
  }
```

city: "London"

TRUE

for loop abandoned, no printout

break statement

```
> cities <- list("New York", "Paris",
                 "London", "Tokyo",
                 "Rio de Janeiro", "Cape Town"))

> for(city in cities) {
    if(nchar(city) == 6) {
      break
    }
    print(city)
  }
[1] "New York"
[1] "Paris"
```

next statement

```
> cities <- list("New York", "Paris",
                 "London", "Tokyo",
                 "Rio de Janeiro", "Cape Town")

> for(city in cities) {
    if(nchar(city) == 6) {
      next
    }
    print(city)
  }
[1] "New York"
[1] "Paris"
[1] "Tokyo"
[1] "Rio de Janeiro"
[1] "Cape Town"
```

next: *skip* to next iteration

"London" is not printed!

for loop: v2

```
> cities <- c("New York", "Paris",
              "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> for(city in cities) {
    print(city)
  }
```

for loop: v2

```
> cities <- c("New York", "Paris",
              "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> for(i in 1:length(cities)) {
    print(city)
  }
```

1:6 == c(1, 2, 3, 4, 5, 6)

for loop: v2

```
> cities <- c("New York", "Paris",
              "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> for(i in 1:length(cities)) {
    print(cities[i])
  }
[1] "New York"
[1] "Paris"
[1] "London"
[1] "Tokyo"
[1] "Rio de Janeiro"
[1] "Cape Town"
```

for loop: v2

```
> cities <- c("New York", "Paris",
              "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> for(i in 1:length(cities)) {
    print(paste(cities[i], "is on position",
                i, "in the cities vector."))
  }
[1] "New York is on position 1 in the cities vector."
[1] "Paris is on position 2 in the cities vector."
[1] "London is on position 3 in the cities vector."
[1] "Tokyo is on position 4 in the cities vector."
[1] "Rio de Janeiro is on position 5 in the cities vector."
[1] "Cape Town is on position 6 in the cities vector."
```

for loop: wrap-up

```
> cities <- c("New York", "Paris",
              "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")


> for(city in cities) {
    print(city)
  }
```

+ Concise
+ Easy to read
- No access to looping index

```
> for(i in 1:length(cities)) {
    print(cities[i])
  }
```

- Harder to read and write
+ More versatile

Let’s practice!

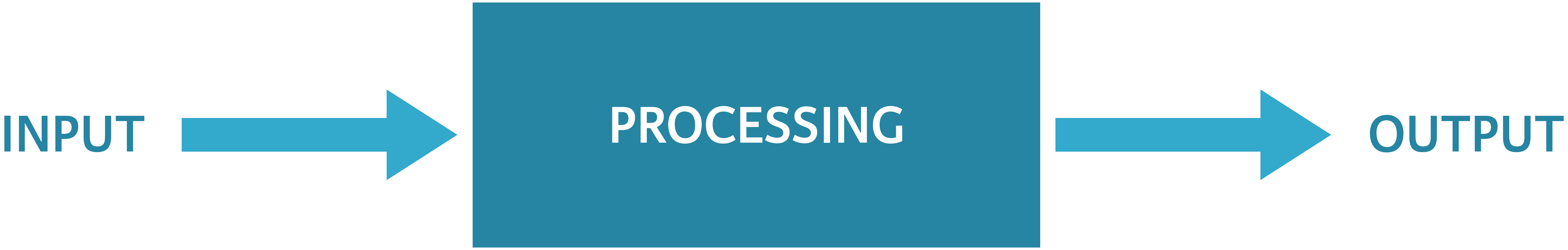
INTERMEDIATE R

Functions

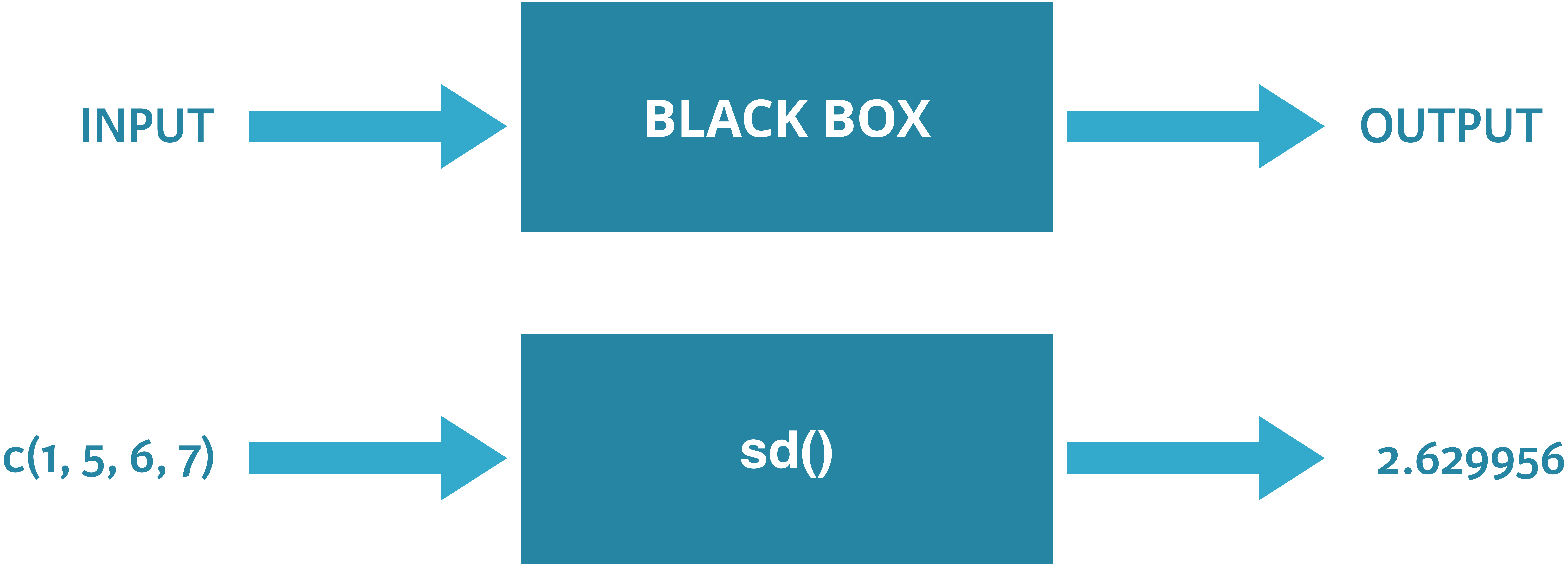
Functions

- You already know 'em!
- Create a list: `list()`
- Display a variable: `print()`

Black box principle



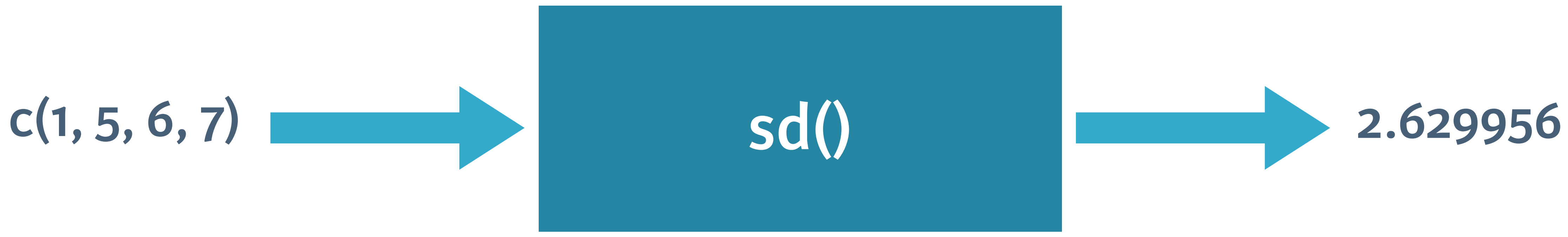
Black box principle



INPUT → BLACK BOX → OUTPUT

c(1, 5, 6, 7) → sd() → 2.629956

Call function in R



c(1, 5, 6, 7)

sd()

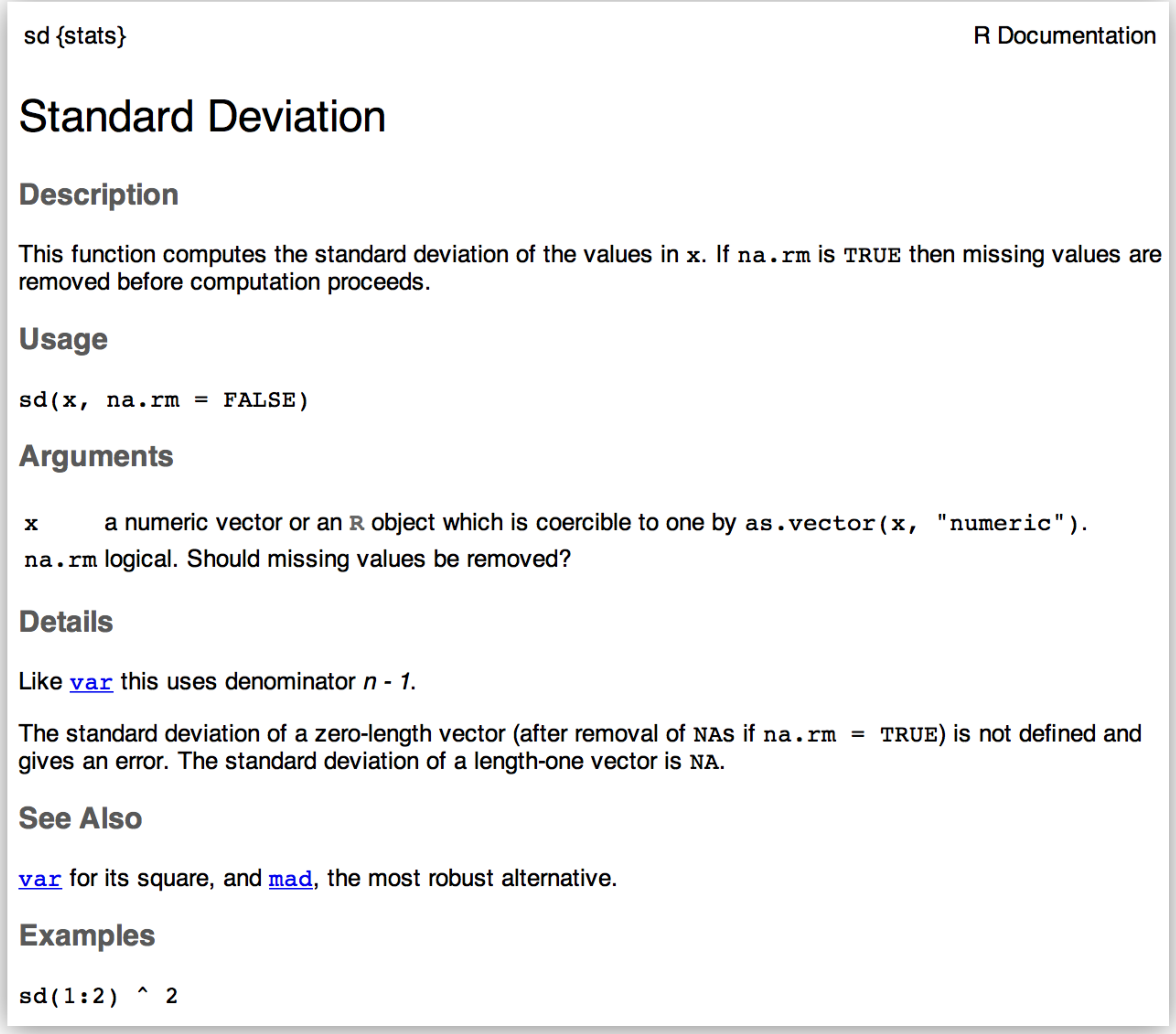
2.629956

```
> sd(c(1, 5, 6, 7))
[1] 2.629956

> values <- c(1, 5, 6, 7)

> sd(values)
[1] 2.629956

> my_sd <- sd(values)

> my_sd
[1] 2.629956
```

Function documentation

```
> help(sd)

> ?sd
```

```
sd(x, na.rm = FALSE)
```



sd {stats} R Documentation

Standard Deviation

Description

This function computes the standard deviation of the values in `x`. If `na.rm` is `TRUE` then missing values are removed before computation proceeds.

Usage

```
sd(x, na.rm = FALSE)
```

Arguments

`x` a numeric vector or an R object which is coercible to one by `as.vector(x, "numeric")`.

`na.rm` logical. Should missing values be removed?

Details

Like `var` this uses denominator *n - 1*.

The standard deviation of a zero-length vector (after removal of `NA`s if `na.rm = TRUE`) is not defined and gives an error. The standard deviation of a length-one vector is `NA`.

See Also

`var` for its square, and `mad`, the most robust alternative.

Examples

```
sd(1:2) ^ 2
```

Questions

```
sd(x, na.rm = FALSE)
```

- Argument names: `x`, `na.rm`
- `na.rm = FALSE`
- `sd(values)` works?

Argument matching

```
sd(x, na.rm = FALSE)
```

x in first position

- By position

```
> sd(values)
```

values in first position → R assigns values to x

- By name

```
> sd(x = values)
```

explicitly assign values to x

na.rm argument

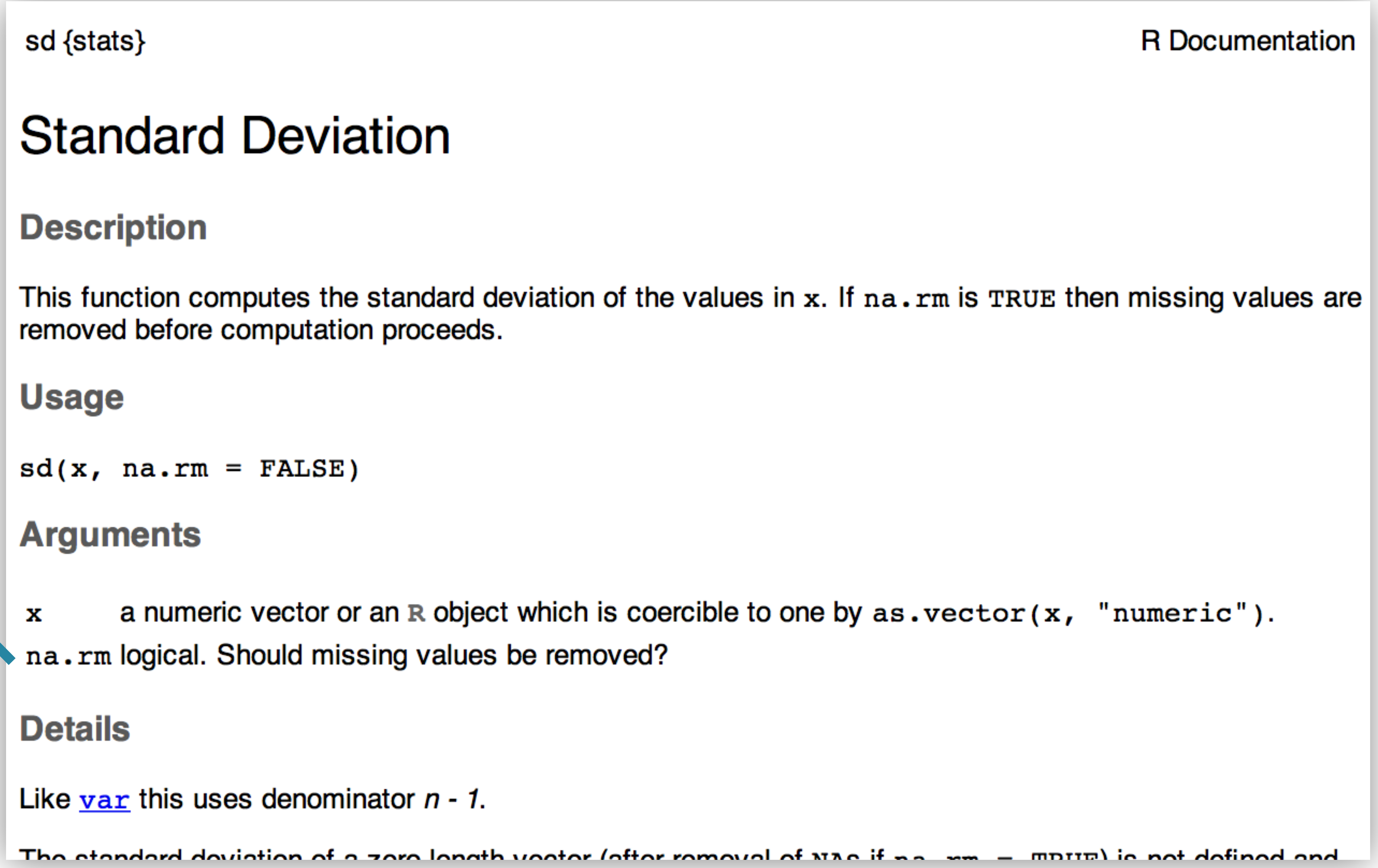
na.rm: logical. Should missing values be removed?

```
> values <- c(1, 5, 6, NA)

> sd(values)
[1] NA


> sd(values, TRUE)
[1] 2.645751
```

Matching by position

by position **by name**

```
> sd(values, na.rm = TRUE)
[1] 2.645751
```



sd {stats} R Documentation

Standard Deviation

Description

This function computes the standard deviation of the values in `x`. If `na.rm` is `TRUE` then missing values are removed before computation proceeds.

Usage

```
sd(x, na.rm = FALSE)
```

Arguments

`x` a numeric vector or an R object which is coercible to one by `as.vector(x, "numeric")`.

`na.rm` logical. Should missing values be removed?

Details

Like `var` this uses denominator *n - 1*.

The standard deviation of a zero-length vector (after removal of `NA`s if `na.rm = TRUE`) is not defined and

na.rm is FALSE by default

```
sd(x, na.rm = FALSE)
```

sd(values) works?

```
> values <- c(1, 5, 6, 7)

> sd(values)
[1] 2.629956

> sd()
Error in is.data.frame(x) : argument "x" is missing,
with no default
```

```
sd(x, na.rm = FALSE)
```

x has no default

`na.rm` is FALSE by default

Useful trick

```
> args(sd)
function (x, na.rm = FALSE)
NULL
```

Wrap-up

- Functions work like a black box
- Argument matching: by position or by name
- Function arguments can have defaults

Let’s practice!

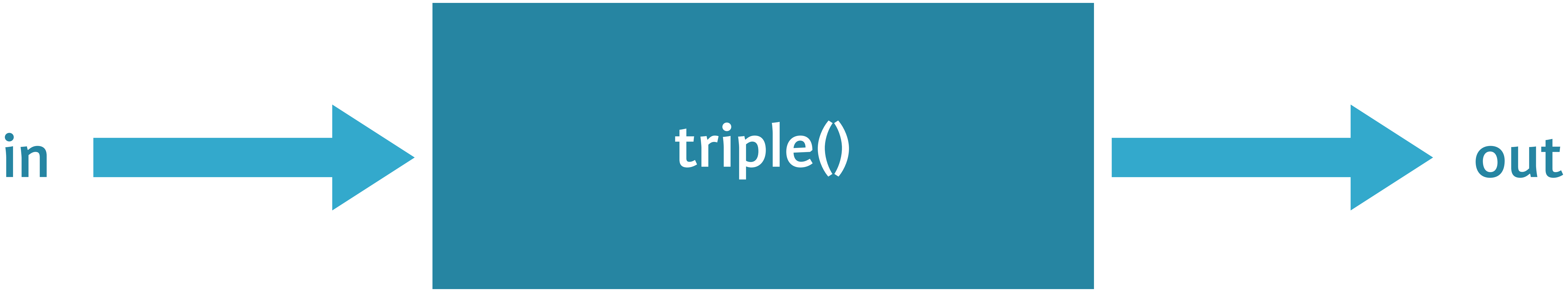
INTERMEDIATE R

Writing Functions

When write your own?

- Solve a particular, well-defined problem
- Black box principle
- If it works, inner workings less important

The triple() function

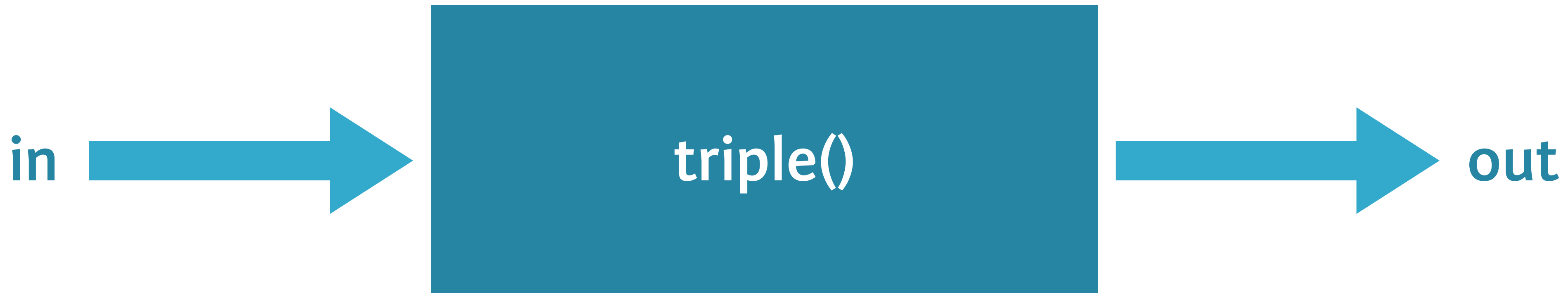


The triple() function

in
triple()
out

```
my_fun <- function(arg1, arg2) {
  body
}
```

The triple() function



```
triple <- function(arg1, arg2) {
  body
}
```

The triple() function

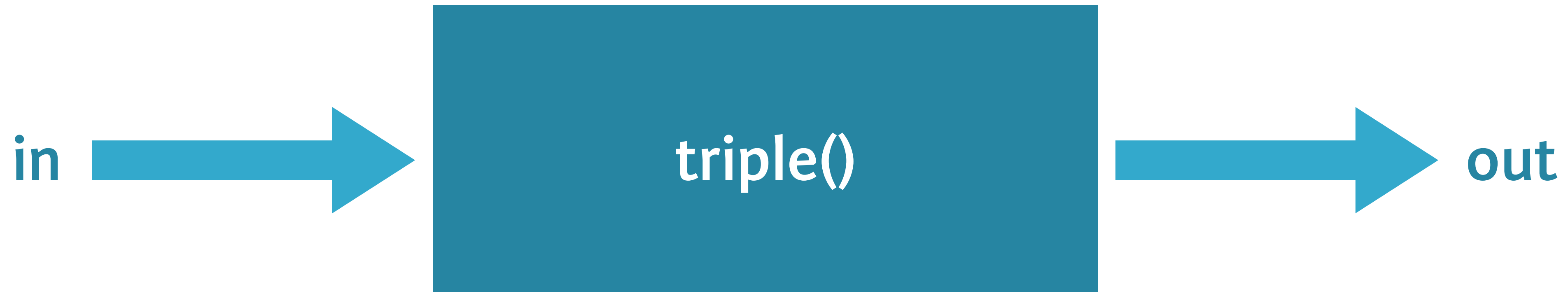
in

triple()

out

```
triple <- function(x) {
  body
}
```

The triple() function



```
triple <- function(x) {
  3 * x
}
```

The triple() function

```
> triple <- function(x) {
    3 * x
  }
```

```
> ls()
[1] "triple"
```

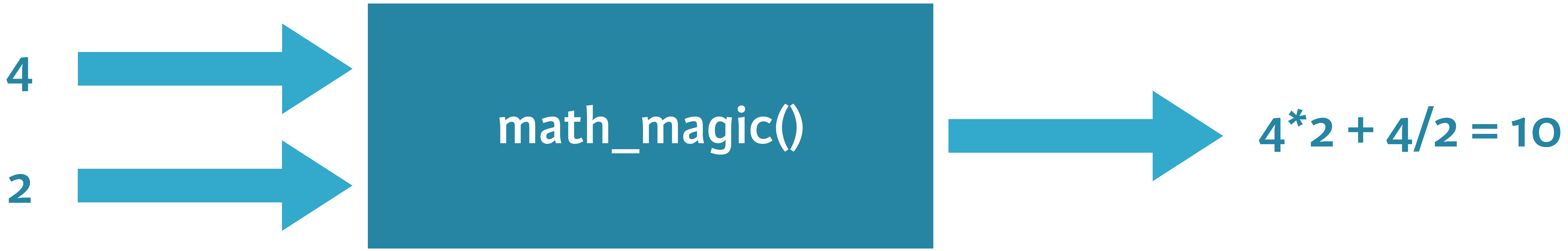
```
> triple(6)
[1] 18
```

Numeric 6 matched to argument x (by pos)
Function body is executed: 3 * 6
Last expression = return value

return()

```
> triple <- function(x) {
    y <- 3 * x
    return(y)
  }

> triple(6)
[1] 18
```

The math_magic() function



The math_magic() function

```
my_fun <- function(arg1, arg2) {
  body
}
```

The math_magic() function

```
math_magic <- function(arg1, arg2) {
  body
}
```

The math_magic() function

```
math_magic <- function(a, b) {
  body
}
```

The math_magic() function

```
math_magic <- function(a, b) {
  a*b + a/b
}
```

```
> math_magic(4, 2)
[1] 10

> math_magic(4)
Error in math_magic(4) : argument "b" is missing, with
no default
```

Optional argument

```
math_magic <- function(a, b = 1) {
  a*b + a/b
}
```

```
> math_magic(4)
[1] 8

> math_magic(4, 0)
[1] Inf
```

Use return()

```
math_magic <- function(a, b = 1) {
  if(b == 0) {
    return(0)     return 0 and exit function
  }
  a*b + a/b       not reached if b is 0
}
```

```
> math_magic(4, 0)
[1] 0
```

Let’s practice!

INTERMEDIATE R

R Packages

R Packages

- Where do mean(), list() and sample() come from?
- Part of R packages
- Code, data, documentation and tests
- Easy to share
- Examples: base, ggvis

Install packages

- base package: automatically installed
- ggvis package: not installed yet

```
> install.packages("ggvis")
```

- CRAN: Comprehensive R Archive Network

Load packages

- load package = attach to search list

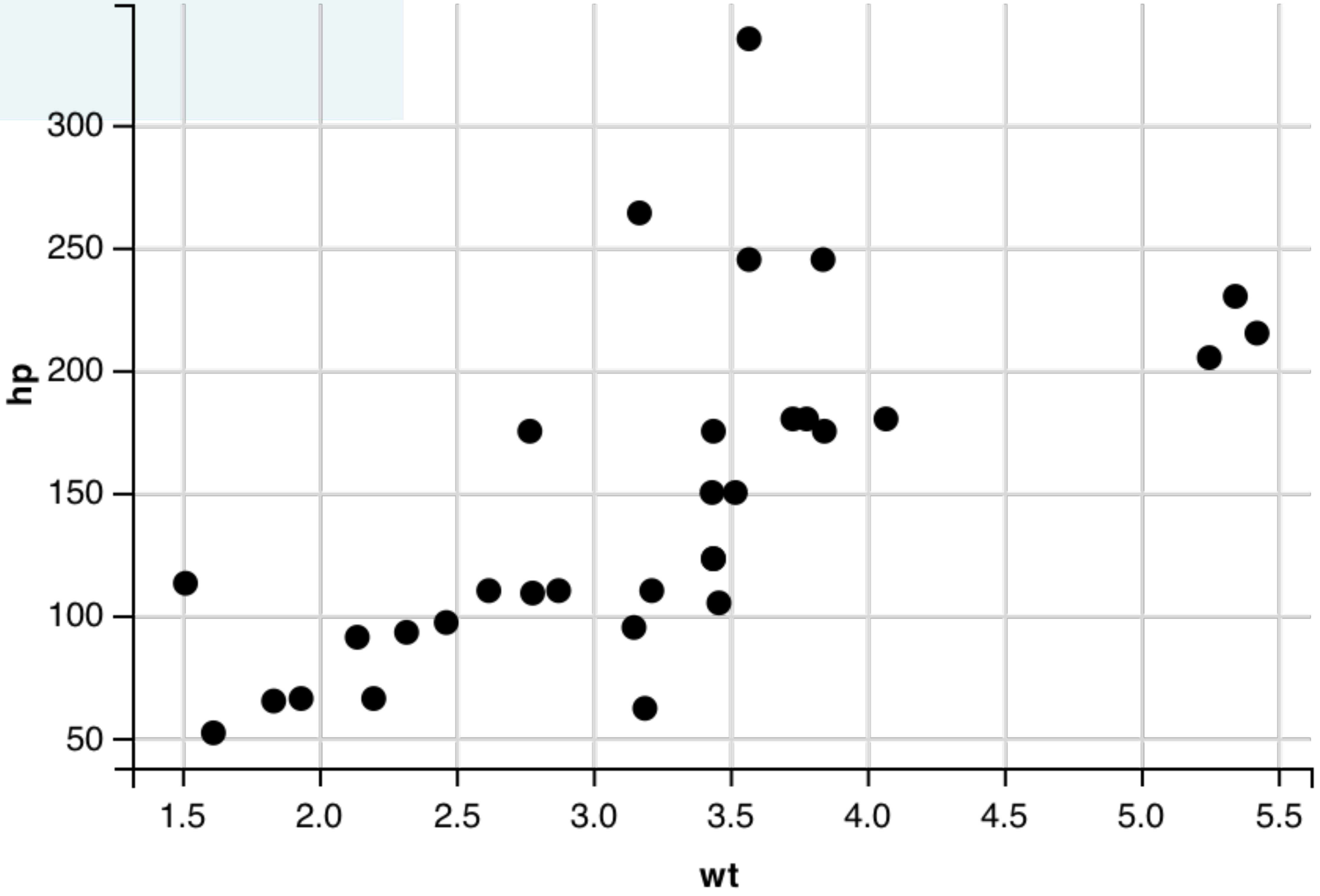
```
> search()
[1] ".GlobalEnv"  ...  "Autoloads"   "package:base"
```

- 7 packages are attached by default
- ggvis not attached by default

```
> ggvis(mtcars, ~wt, ~hp)
Error: could not find function "ggvis"
```

Load packages: library()

```
> library("ggvis")

> search()
 [1] ".GlobalEnv"  "package:ggvis" ... "package:base"

> ggvis(mtcars, ~wt, ~hp)
```



Load packages: require()

```
> library("data.table")
Error in library("data.table") : there is no package called
‘data.table’

> require("data.table")
Loading required package: data.table
Warning message: ...

> result <- require("data.table")
Loading required package: data.table
Warning message: ...

> result
[1] FALSE
```

Wrap-up

- Install packages: install.packages()
- Load packages: library(), require()
- Load package = attach package to search list
- Google for cool R packages!

INTERMEDIATE R

Let’s practice!

INTERMEDIATE R

lapply

NYC: for

```
> nyc <- list(pop = 8405837,
              boroughs = c("Manhattan", "Bronx", "Brooklyn",
                           "Queens", "Staten Island"),
              capital = FALSE)

> for(info in nyc) {
    print(class(info))
  }
[1] "numeric"
[1] "character"
[1] "logical"
```

NYC: lapply()

```
> nyc <- list(pop = 8405837,
              boroughs = c("Manhattan", "Bronx", "Brooklyn",
                           "Queens", "Staten Island"),
              capital = FALSE)

> lapply(nyc, class)
$pop
[1] "numeric"

$boroughs
[1] "character"

$capital
[1] "logical"
```

Cities: for

```
> cities <- c("New York", "Paris", "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> num_chars <- c()
> for(i in 1:length(cities)) {
    num_chars[i] <- nchar(cities[i])
  }

> num_chars
[1]  8  5  6  5 14  9
```

Cities: lapply()

```
> cities <- c("New York", "Paris", "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> lapply(cities, nchar)
[[1]]
[1] 8

[[2]]
[1] 5

...

[[6]]
[1] 9
```

Cities: lapply()

```
> cities <- c("New York", "Paris", "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> unlist(lapply(cities, nchar))
[1]  8  5  6  5 14  9
```

Oil

```
> oil_prices <- list(2.37, 2.49, 2.18, 2.22, 2.47, 2.32)
> triple <- function(x) {
    3 * x
  }
> result <- lapply(oil_prices, triple)
> str(result)
List of 6
 $ : num 7.11
 $ : num 7.47
 $ : num 6.54
 $ : num 6.66
 $ : num 7.41
 $ : num 6.96
> unlist(result)
[1] 7.11 7.47 6.54 6.66 7.41 6.96
```

Oil

```
> oil_prices <- list(2.37, 2.49, 2.18, 2.22, 2.47, 2.32)
> multiply <- function(x, factor) {
    x * factor
  }

> times3 <- lapply(oil_prices, multiply, factor = 3)
> unlist(times3)
[1] 7.11 7.47 6.54 6.66 7.41 6.96

> times4 <- lapply(oil_prices, multiply, factor = 4)
> unlist(times4)
[1] 9.48 9.96 8.72 8.88 9.88 9.28
```

Let’s practice!

INTERMEDIATE R

sapply

lapply()

- Apply function over list or vector
- Function can return R objects of different classes
- List necessary to store heterogeneous content
- However, often homogeneous content

Cities: lapply()

```
> cities <- c("New York", "Paris", "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> result <- lapply(cities, nchar)

> str(result)
List of 6
 $ : int 8
 $ : int 5
 $ : int 6
 $ : int 5
 $ : int 14
 $ : int 9

> unlist(lapply(cities, nchar))
[1]  8  5  6  5 14  9
```

Cities: sapply()

```
> cities <- c("New York", "Paris", "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> unlist(lapply(cities, nchar))
[1]  8  5  6  5 14  9


> sapply(cities, nchar)
 New York   Paris  London   Tokyo  Rio de Janeiro  Cape Town
        8       5       6       5              14          9

> sapply(cities, nchar, USE.NAMES = FALSE)
[1]  8  5  6  5 14  9
```

USE.NAMES is TRUE by default

Cities: sapply()

```
> first_and_last <- function(name) {
    name <- gsub(" ", "", name)
    letters <- strsplit(name, split = "")[[1]]
    c(first = min(letters), last = max(letters))
  }

> first_and_last("New York")
first  last
  "e"   "Y"

> sapply(cities, first_and_last)
      New York Paris  London  Tokyo  Rio de Janeiro  Cape Town
first "e"      "a"    "d"     "k"    "a"             "a"
last  "Y"      "s"    "o"     "y"    "R"             "w"
```

Unable to simplify?

```
> unique_letters <- function(name) {
    name <- gsub(" ", "", name)
    letters <- strsplit(name, split = "")[[1]]
    unique(letters)
  }

> unique_letters("London")
[1] "L" "o" "n" "d"
```

Unable to simplify?

```
> lapply(cities, unique_letters)
[[1]]
[1] "N" "e" "w" "Y" "o" "r" "k"

[[2]]
[1] "P" "a" "r" "i" "s"

[[3]]
[1] "L" "o" "n" "d"

[[4]]
[1] "T" "o" "k" "y"

...
```

```
> sapply(cities, unique_letters)
$`New York`
[1] "N" "e" "w" "Y" "o" "r" "k"

$Paris
[1] "P" "a" "r" "i" "s"

$London
[1] "L" "o" "n" "d"

$Tokyo
[1] "T" "o" "k" "y"

...
```

sapply did not simplify
Can be dangerous!

Let’s practice!

INTERMEDIATE R

vapply

Recap

- **lapply()**
  apply function over list or vector
  output = list

- **sapply()**
  apply function over list or vector
  try to simplify list to array

- **vapply()**
  apply function over list or vector
  explicitly specify output format

sapply() & vapply()

```
> cities <- c("New York", "Paris", "London", "Tokyo",
              "Rio de Janeiro", "Cape Town")

> sapply(cities, nchar)
 New York   Paris  London   Tokyo  Rio de Janeiro  Cape Town
        8       5       6       5              14          9
```

```
vapply(X, FUN, FUN.VALUE, ..., USE.NAMES = TRUE)
```

```
> vapply(cities, nchar, numeric(1))
 New York   Paris  London   Tokyo  Rio de Janeiro  Cape Town
        8       5       6       5              14          9
```

vapply()

```
> first_and_last <- function(name) {
    name <- gsub(" ", "", name)
    letters <- strsplit(name, split = "")[[1]]
    return(c(first = min(letters), last = max(letters)))
  }

> sapply(cities, first_and_last)
      New York Paris London Tokyo Rio de Janeiro Cape Town
first "e"      "a"   "d"    "k"   "a"            "a"
last  "Y"      "s"   "o"    "y"   "R"            "w"

> vapply(cities, first_and_last, character(2))
      New York Paris London Tokyo Rio de Janeiro Cape Town
first "e"      "a"   "d"    "k"   "a"            "a"
last  "Y"      "s"   "o"    "y"   "R"            "w"
```

vapply() errors

```
> vapply(cities, first_and_last, character(2))
      New York Paris London Tokyo Rio de Janeiro Cape Town
first "e"      "a"   "d"    "k"   "a"            "a"
last  "Y"      "s"   "o"    "y"   "R"            "w"


> vapply(cities, first_and_last, character(1))
Error in vapply(cities, first_and_last, character(1)) :
  values must be length 1,
 but FUN(X[[1]]) result is length 2


> vapply(cities, first_and_last, numeric(2))
Error in vapply(cities, first_and_last, numeric(2)) :
  values must be type 'double',
 but FUN(X[[1]]) result is type 'character'
```

unique_letters()

```
> unique_letters <- function(name) {
    name <- gsub(" ", "", name)
    letters <- strsplit(name, split = "")[[1]]
    unique(letters)
  }
```

vapply() > sapply()

```
> sapply(cities, unique_letters)
$`New York`
[1] "N" "e" "w" "Y" "o" "r" "k"

...

$`Cape Town`
[1] "C" "a" "p" "e" "T" "o" "w" "n"

> vapply(cities, unique_letters, character(4))
Error in vapply(cities, unique_letters, character(4)) :
  values must be length 4,
 but FUN(X[[1]]) result is length 7
```

vapply() is safer than sapply()!

Let’s practice!

INTERMEDIATE R

Useful Functions

Loads of useful functions

- sapply(), vapply(), lapply()
- sort()
- print()
- identical()
- …

Mathematical utilities

```r
v1 <- c(1.1, -7.1, 5.4, -2.7)
v2 <- c(-3.6, 4.1, 5.8, -8.0)
mean(c(sum(round(abs(v1))), sum(round(abs(v2)))))
```

abs()

```
v1 <- c(1.1, -7.1, 5.4, -2.7)
v2 <- c(-3.6, 4.1, 5.8, -8.0)
mean(c(sum(round(abs(v1))), sum(round(abs(v2)))))
```

```
> abs(c(1.1, -7.1, 5.4, -2.7))
[1] 1.1 7.1 5.4 2.7
> abs(c(-3.6, 4.1, 5.8, -8.0))
[1] 3.6 4.1 5.8 8.0
```

```
mean(c(sum(round(c(1.1, 7.1, 5.4, 2.7))),
       sum(round(c(3.6, 4.1, 5.8, 8.0)))))
```

round()

```
v1 <- c(1.1, -7.1, 5.4, -2.7)
v2 <- c(-3.6, 4.1, 5.8, -8.0)
mean(c(sum(round(abs(v1))), sum(round(abs(v2)))))
```

```
mean(c(sum(round(c(1.1, 7.1, 5.4, 2.7))),
       sum(round(c(3.6, 4.1, 5.8, 8.0)))))
```

```
> round(c(1.1, 7.1, 5.4, 2.7))
[1] 1 7 5 3
> round(c(3.6, 4.1, 5.8, 8.0))
[1] 4 4 6 8
```

```
mean(c(sum(c(1, 7, 5, 3)),
       sum(c(4, 4, 6, 8))))
```

sum()

```
v1 <- c(1.1, -7.1, 5.4, -2.7)
v2 <- c(-3.6, 4.1, 5.8, -8.0)
mean(c(sum(round(abs(v1))), sum(round(abs(v2)))))
```

```
mean(c(sum(c(1, 7, 5, 3)),
       sum(c(4, 4, 6, 8))))
```

```
> sum(c(1, 7, 5, 3))
[1] 16
> sum(c(4, 4, 6, 8))
[1] 22
```

```
mean(c(16, 22))
```

mean()

```
> mean(c(16, 22))
[1] 19
```

```
> v1 <- c(1.1, -7.1, 5.4, -2.7)
> v2 <- c(-3.6, 4.1, 5.8, -8.0)
> mean(c(sum(round(abs(v1))), sum(round(abs(v2)))))
[1] 19
```

Functions for data structures

```
li <- list(log = TRUE,
           ch = "hello",
           int_vec = sort(rep(seq(8, 2, by = -2), times = 2)))
```

```
sort(rep(seq(8, 2, by = -2), times = 2)))
```

seq()

```
li <- list(log = TRUE,
           ch = "hello",
           int_vec = sort(rep(seq(8, 2, by = -2), times = 2)))
```

```
sort(rep(seq(8, 2, by = -2), times = 2)))
```

```
> seq(1, 10, by = 3)
[1]  1  4  7 10

> seq(8, 2, by = -2)
[1] 8 6 4 2
```

```
sort(rep(c(8, 6, 4, 2), times = 2))
```

rep()

```
li <- list(log = TRUE,
           ch = "hello",
           int_vec = sort(rep(seq(8, 2, by = -2), times = 2)))
```

```
sort(rep(c(8, 6, 4, 2), times = 2))
```

```
> rep(c(8, 6, 4, 2), times = 2)
[1] 8 6 4 2 8 6 4 2

> rep(c(8, 6, 4, 2), each = 2)
[1] 8 8 6 6 4 4 2 2
```

```
sort(c(8, 6, 4, 2, 8, 6, 4, 2))
```

sort()

```
li <- list(log = TRUE,
           ch = "hello",
           int_vec = sort(rep(seq(8, 2, by = -2), times = 2)))
```

```
> sort(c(8, 6, 4, 2, 8, 6, 4, 2))
[1] 2 2 4 4 6 6 8 8

> sort(c(8, 6, 4, 2, 8, 6, 4, 2), decreasing = TRUE)
[1] 8 8 6 6 4 4 2 2
```

```
> sort(rep(seq(8, 2, by = -2), times = 2))
[1] 2 2 4 4 6 6 8 8
```

str()

```
> li <- list(log = TRUE,
             ch = "hello",
             int_vec = sort(rep(seq(8, 2, by = -2), times = 2)))
> str(li)
List of 3
 $ log    : logi TRUE
 $ ch     : chr "hello"
 $ int_vec: num [1:8] 2 2 4 4 6 6 8 8
```

is.*(), as.*()

```
> is.list(li)
[1] TRUE

> is.list(c(1, 2, 3))
[1] FALSE

> li2 <- as.list(c(1, 2, 3))

> is.list(li2)
[1] TRUE

> unlist(li)
     log       ch  int_vec1  int_vec2  ... int_vec7  int_vec8
  "TRUE"  "hello"       "2"       "2"  ...      "8"       "8"
```

append(), rev()

```
str(append(li, rev(li)))
```

```
> str(rev(li))
List of 3
 $ int_vec: num [1:8] 2 2 4 4 6 6 8 8
 $ ch     : chr "hello"
 $ log    : logi TRUE

> str(append(li, rev(li)))
List of 6
 $ log    : logi TRUE
 $ ch     : chr "hello"
 $ int_vec: num [1:8] 2 2 4 4 6 6 8 8
 $ int_vec: num [1:8] 2 2 4 4 6 6 8 8
 $ ch     : chr "hello"
 $ log    : logi TRUE
```

INTERMEDIATE R

Let’s practice!

INTERMEDIATE R

Regular Expressions

Regular Expressions

- Sequence of (meta)characters
- Pattern existence
- Pattern replacement
- Pattern extraction
- grep(), grepl()
- sub(), gsub()

grepl()

```
> animals <- c("cat", "moose", "impala", "ant", "kiwi")
```

```
grepl(pattern = <regex>, x = <string>)
```

```
> grepl(pattern = "a", x = animals)
[1]  TRUE FALSE  TRUE  TRUE FALSE

> grepl(pattern = "^a", x = animals)
[1] FALSE FALSE FALSE  TRUE FALSE

> grepl(pattern = "a$", x = animals)
[1] FALSE FALSE  TRUE FALSE FALSE

> ?regex
```

grep()

```
> animals <- c("cat", "moose", "impala", "ant", "kiwi")
```

```
> grepl(pattern = "a", x = animals)
[1]  TRUE FALSE  TRUE  TRUE FALSE

> grep(pattern = "a", x = animals)
[1] 1 3 4

> which(grepl(pattern = "a", x = animals))
[1] 1 3 4

> grep(pattern = "^a", x = animals)
[1] 4
```

sub(), gsub()

```
> animals <- c("cat", "moose", "impala", "ant", "kiwi")
```

```
sub(pattern = <regex>, replacement = <str>, x = <str>)
```

```
> sub(pattern = "a", replacement = "o", x = animals)
[1] "cot"    "moose"  "impola" "ont"    "kiwi"

> gsub(pattern = "a", replacement = "o", x = animals)
[1] "cot"    "moose"  "impolo" "ont"    "kiwi"
```

sub(), gsub()

```
> animals <- c("cat", "moose", "impala", "ant", "kiwi")
```

```
> sub(pattern = "a", replacement = "o", x = animals)
[1] "cot"    "moose"  "impola" "ont"    "kiwi"

> gsub(pattern = "a", replacement = "o", x = animals)
[1] "cot"    "moose"  "impolo" "ont"    "kiwi"
```

```
> gsub(pattern = "a|i", replacement = "_", x = animals)
[1] "c_t"    "moose"  "_mp_l_" "_nt"    "k_w_"

> gsub(pattern = "a|i|o", replacement = "_", x = animals)
[1] "c_t"    "m__se"  "_mp_l_" "_nt"    "k_w_"
```

INTERMEDIATE R

Let’s practice!

INTERMEDIATE R

Times & Dates

Today, right now!

```
> today <- Sys.Date()
> today
[1] "2015-05-07"

> class(today)
[1] "Date"
```

```
> now <- Sys.time()
> now
[1] "2015-05-07 10:34:52 CEST"

> class(now)
[1] "POSIXct" "POSIXt"
```

Create Date objects

```
> my_date <- as.Date("1971-05-14")
> my_date
[1] "1971-05-14"

> class(my_date)
[1] "Date"

> my_date <- as.Date("1971-14-05")
Error in charToDate(x) :
  character string is not in a standard unambiguous format

> my_date <- as.Date("1971-14-05", format = "%Y-%d-%m")
> my_date
[1] "1971-05-14"
```

Default format
"%Y-%m-%d"

%Y = 4-digit year
%m = 2-digit month
%d = 2-digit day

Create POSIXct objects

```
> my_time <- as.POSIXct("1971-05-14 11:25:15")
> my_time
[1] "1971-05-14 11:25:15 CET"
```

Date arithmetic

```
> my_date
[1] "1971-05-14"
```

days incremented by 1

```
> my_date + 1
[1] "1971-05-15"

> my_date2 <- as.Date("1998-09-29")

> my_date2 - my_date
Time difference of 10000 days
```

POSIXct arithmetic

```
> my_time
[1] "1971-05-14 11:25:15 CET"

> my_time + 1
[1] "1971-05-14 11:25:16 CET"

> my_time2 <- as.POSIXct("1974-07-14 21:11:55 CET")

> my_time2 - my_time
Time difference of 1157.407 days
```

seconds incremented by 1

Under the hood

```
> my_date
[1] "1971-05-14"

> unclass(my_date)
[1] 498

> my_time
[1] "1971-05-14 11:25:15 CET"

> unclass(my_time)
[1] 43064715
attr(,"tzone")
[1] ""
```

498 days from January 1, 1970

>43MM seconds from January 1, 1970, 00:00:00

Dedicated R Packages

- lubridate
- zoo
- xts

INTERMEDIATE R

Let’s practice!