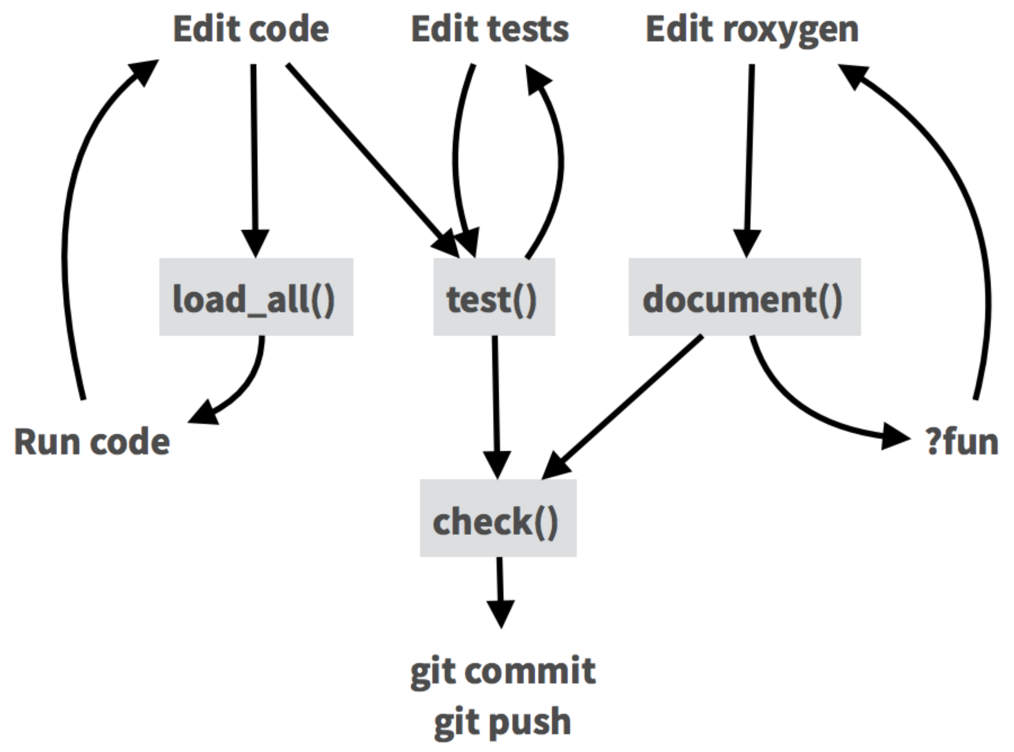
Continuous development: The virtuous circle of writing/documenting/testing

Physalia course 2023

Instructor: Jacques Serizay

Package development workflow



- **load_all()** (Ctrl/Cmd + Shift + L) — Load code
- **document()** (Ctrl/Cmd + Shift + D) — Rebuild docs and NAMESPACE
- **test()** (Ctrl/Cmd + Shift + T) — Run tests
- **check()** (Ctrl/Cmd + Shift + E) — Check complete package

https://raw.githubusercontent.com/rstudio/cheatsheets/main/package-development.pdf

Package development workflow

```
> devtools::create_package()
> usethis::use_readme_md()
> usethis::use_news_md()
> usethis::use_gpl3_license()
```

```
myPackage/
    DESCRIPTION
    README.md
    NAMESPACE
    NEWS
    LICENSE
```

Package development workflow

Write functions

```
> devtools::load_all()
```

```
myPackage/
    R/
        functions.R
        utils.R
    DESCRIPTION
    README.md
    NAMESPACE
    NEWS
    LICENSE
```

Package development workflow

Write functions

Document functions
- Arguments
- Imports
- examples

```
myPackage/
    R/
        functions.R
        utils.R
    DESCRIPTION
    README.md
    NAMESPACE
    NEWS
    LICENSE
```

Package development workflow

Write functions

Document functions
- Arguments
- Imports
- examples

```
> devtools::document()
> devtools::run_examples()
> devtools::load_all()
```

```
myPackage/
  R/
    functions.R
    utils.R
  man/
    myfunction.Rd
  DESCRIPTION
  README.md
  NAMESPACE
  NEWS
  LICENSE
```

Package development workflow

Write functions

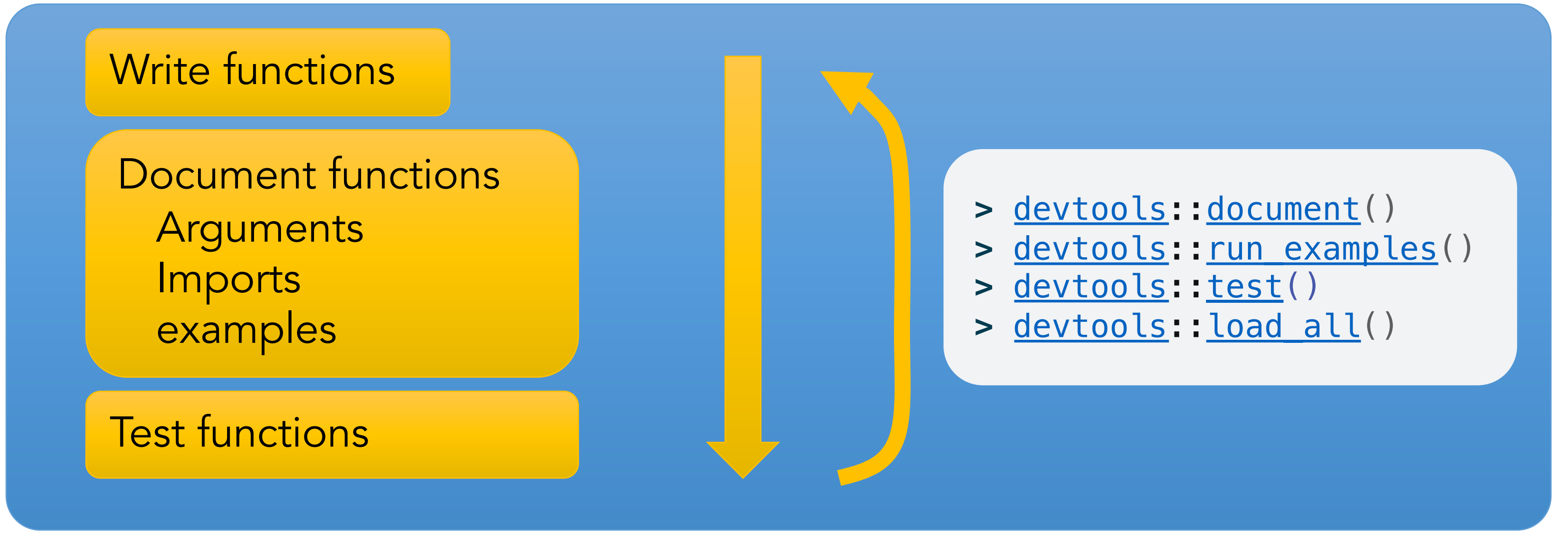
Document functions
- Arguments
- Imports
- examples

Test functions

```
> devtools::document()
> devtools::run_examples()
> devtools::test()
> devtools::load_all()
```

```
myPackage/
    R/
        functions.R
        utils.R
    man/
        myfunction.Rd
    tests/
        testthat.R
        testthat/
            test-myfun.R
    DESCRIPTION
    README.md
    NAMESPACE
    NEWS
    LICENSE
```

Package development workflow



```
> devtools::document()
> devtools::run_examples()
> devtools::test()
> devtools::load_all()
```

```
> devtools::check()
```

```
Git commit
Github push
```

```
myPackage/
  R/
    functions.R
    utils.R
  man/
    myfunction.Rd
  tests/
    testthat.R
    testthat/
      test-myfun.R
  data/
    toy-data.rda
  inst/
    ext/
      raw.bed
  DESCRIPTION
  README.md
  NAMESPACE
  NEWS
  LICENSE
```

Package development workflow

Write functions

Document functions
- Arguments
- Imports
- examples

Test functions

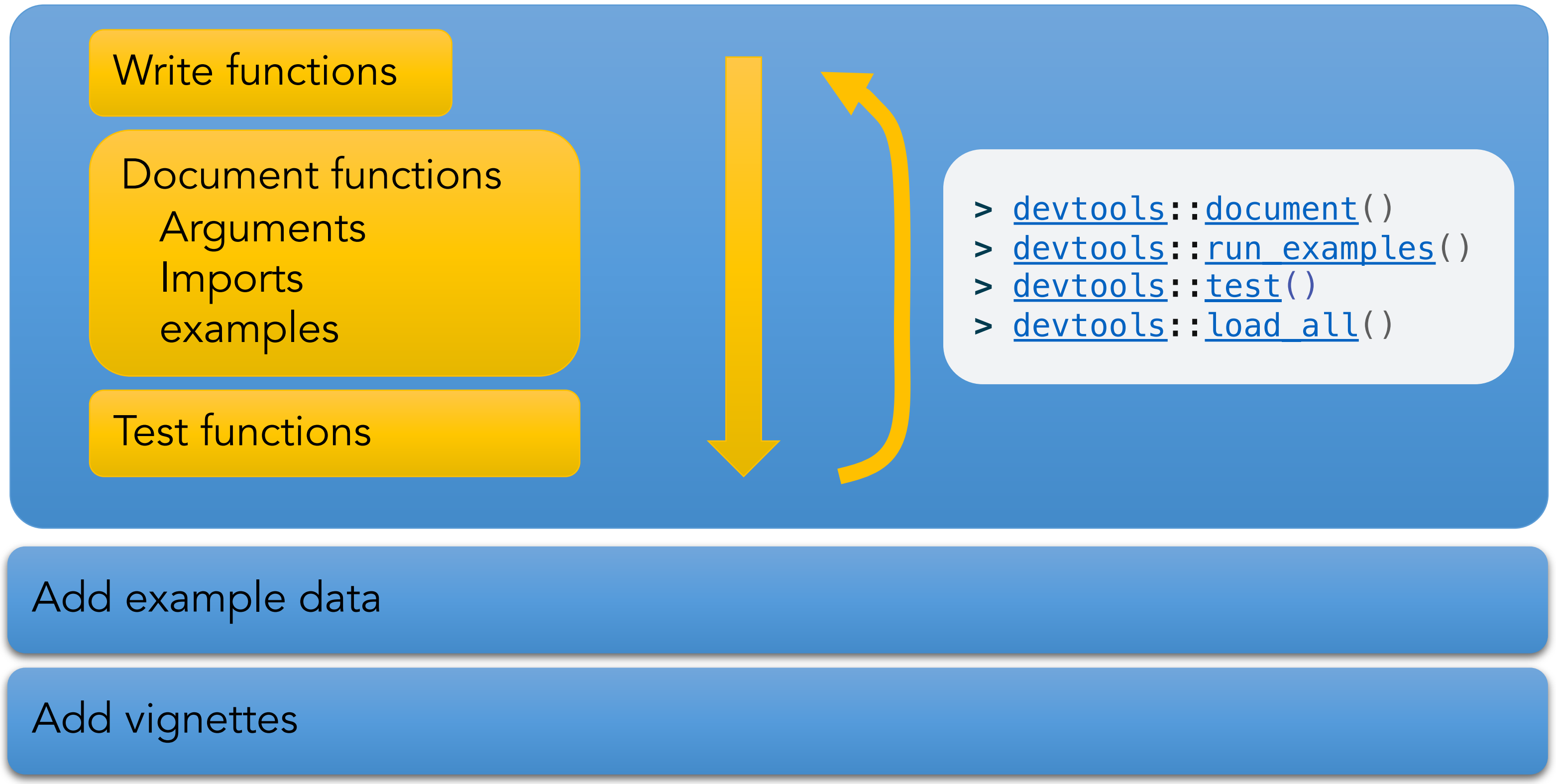
```
> devtools::document()
> devtools::run_examples()
> devtools::test()
> devtools::load_all()
```

Add example data

```
Git commit
Github push
```

```
myPackage/
  R/
    functions.R
    utils.R
  man/
    myfunction.Rd
  tests/
    testthat.R
    testthat/
      test-myfun.R
  data/
    toy-data.rda
  inst/
    ext/
      raw.bed
  DESCRIPTION
  README.md
  NAMESPACE
  NEWS
  LICENSE
```

Package development workflow

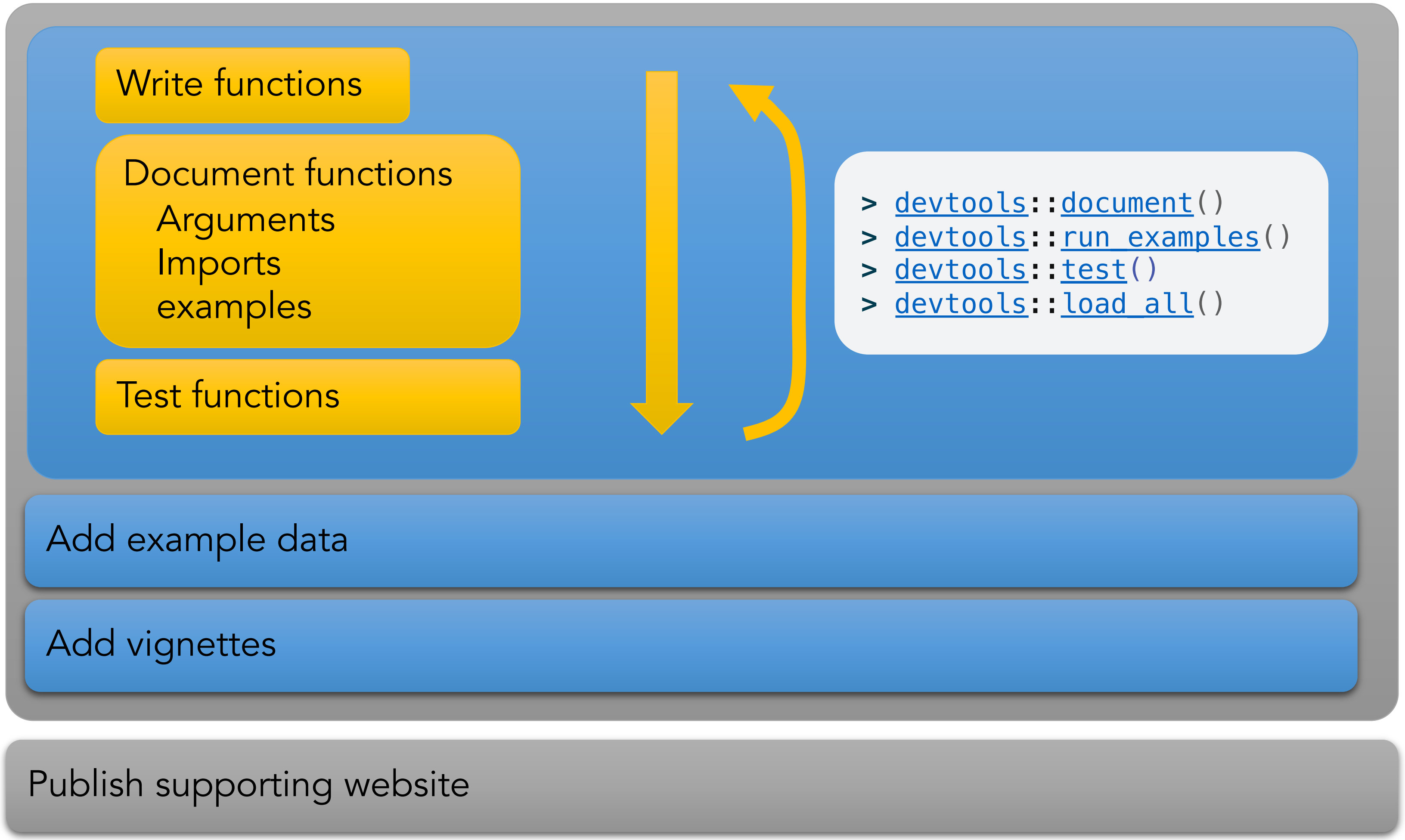


```
> devtools::document()
> devtools::run_examples()
> devtools::test()
> devtools::load_all()
```

```
Git commit
Github push
```

```
myPackage/
    R/
        functions.R
        utils.R
    man/
        myfunction.Rd
    tests/
        testthat.R
        testthat/
            test-myfun.R
    data/
        toy-data.rda
    inst/
        ext/
            raw.bed
    vignettes/
        myPackage.Rmd
    DESCRIPTION
    README.md
    NAMESPACE
    NEWS
    LICENSE
```

Package development workflow

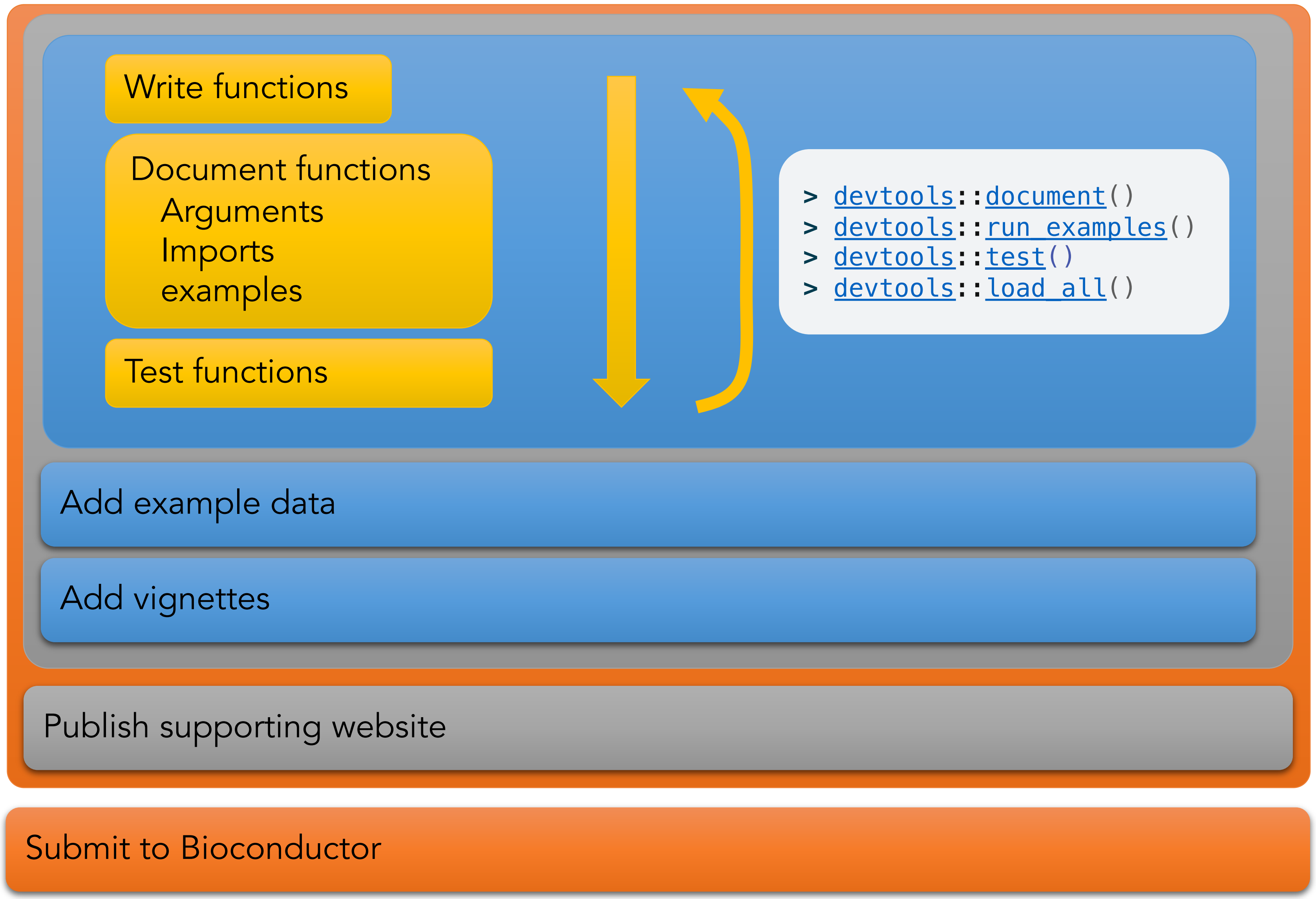


```
> devtools::document()
> devtools::run_examples()
> devtools::test()
> devtools::load_all()
```

```
Git commit
Github push
```

```
myPackage/
    R/
        functions.R
        utils.R
    man/
        myfunction.Rd
    tests/
        testthat.R
        testthat/
            test-myfun.R
    data/
        toy-data.rda
    inst/
        ext/
            raw.bed
    vignettes/
        myPackage.Rmd
    DESCRIPTION
    README.md
    NAMESPACE
    NEWS
    LICENSE
    _pkgdown.yml
```

Package development workflow



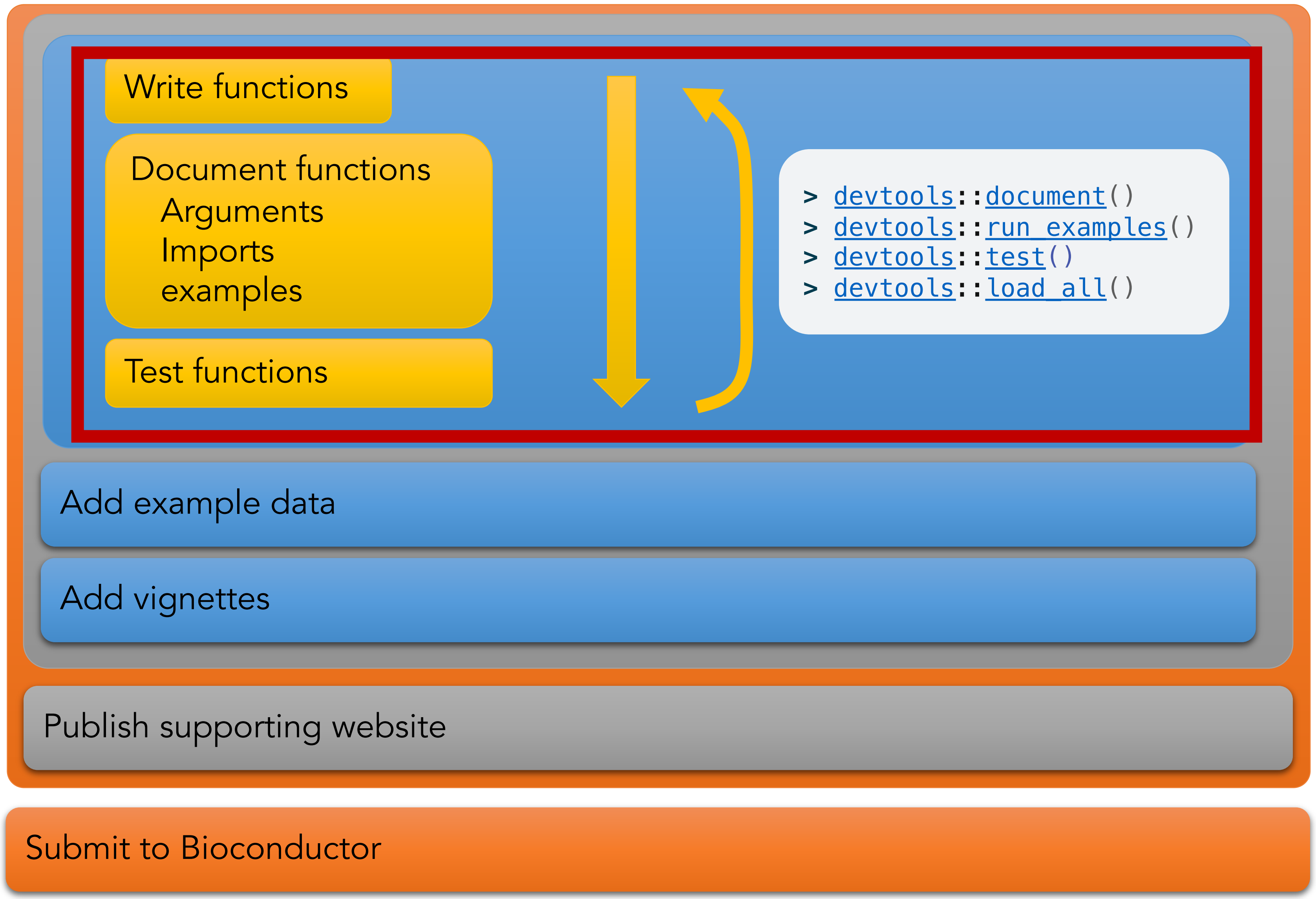
```
> devtools::document()
> devtools::run_examples()
> devtools::test()
> devtools::load_all()
```

```
myPackage/
    R/
        functions.R
        utils.R
    man/
        myfunction.Rd
    tests/
        testthat.R
        testthat/
            test-myfun.R
    data/
        toy-data.rda
    inst/
        ext/
            raw.bed
    vignettes/
        myPackage.Rmd
    DESCRIPTION
    README.md
    NAMESPACE
    NEWS
    LICENSE
    _pkgdown.yml
```

```
Git commit
Github push
```

Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS

Package development workflow



```
> devtools::document()
> devtools::run_examples()
> devtools::test()
> devtools::load_all()
```

```
myPackage/
    R/
        functions.R
        utils.R
    man/
        myfunction.Rd
    tests/
        testthat.R
        testthat/
            test-myfun.R
    data/
        toy-data.rda
    inst/
        ext/
            raw.bed
    vignettes/
        myPackage.Rmd
    DESCRIPTION
    README.md
    NAMESPACE
    NEWS
    LICENSE
    _pkgdown.yml
```

```
Git commit
Github push
```

Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS

How to write functions

Write functions

```r
myfun <- function(arg1, arg2, ...) {


   ## Internal checkups
   if (...) {
      stop("There has been an error. Aborting now.")
   }


    ## Internal processing steps
    step1 <- ...(arg1)
    step2 <- ...(step1)
    step3 <- ...(step2, arg2)


    ## Computing and returning result
    res <- list(step3, ...(arg2), ...)
    return(res)


}
```

How to write functions

Write functions

- Always put .R files containing functions in R/. The easiest to create these files is to run `use_r(”…”)`.

`source(“.”)` and `load_all(“.”)` do not behave the same way: while `source` dumps all .R files found in directory and recursively, `load_all(“.”)` specifically reads in .R files from R/ folder.

How to write functions

Write functions

- Prefer many short functions over a single massive function. Bioconductor advises functions shorter than 100 lines.

Functions are <u>immensely</u> easier to test/debug this way.

Documenting functions

Document functions
- Arguments
- Imports
- examples

“Roxygen” function documentation works by adding @tags before your function, such as:
- @title
- @description
- @details
- @params
- @returns
- @imports
- @export
- @examples

```r
#' @title
#' Paste of vector elements
#'
#' @description
#’ `myPaste` returns a string and a numerical value
#’ pasted together.
#'
#' @details
#' This is a generic function: methods can be
#' defined for it directly or via the
#' [Summary()] group generic. For this to work
#' properly, the arguments `...` should be
#' unnamed, and dispatch is on the first
#' argument.
#'
#' @param arg1 character A character string.
#' @param arg2 numeric A numerical value to append
#' to the character string provided in \code{arg1}.
#'
#' @returns character A string with \code{arg1} and
#' \code{arg2} pasted together
#'
#' @importFrom glue glue
#'
#' @export
#'
#' @examples
#' myPaste("Jacques' cat is ", 3)

myPaste <- function(arg1, arg2) {

   ## Internal checkups
   if (!is.character(arg1) | !is.numeric(arg2)) {
      stop("There has been an error. Aborting now.")
   }

   ## Internal processing steps
   res <- glue::glue(arg1, arg2)

   ## Return result
   return(res)

}
```

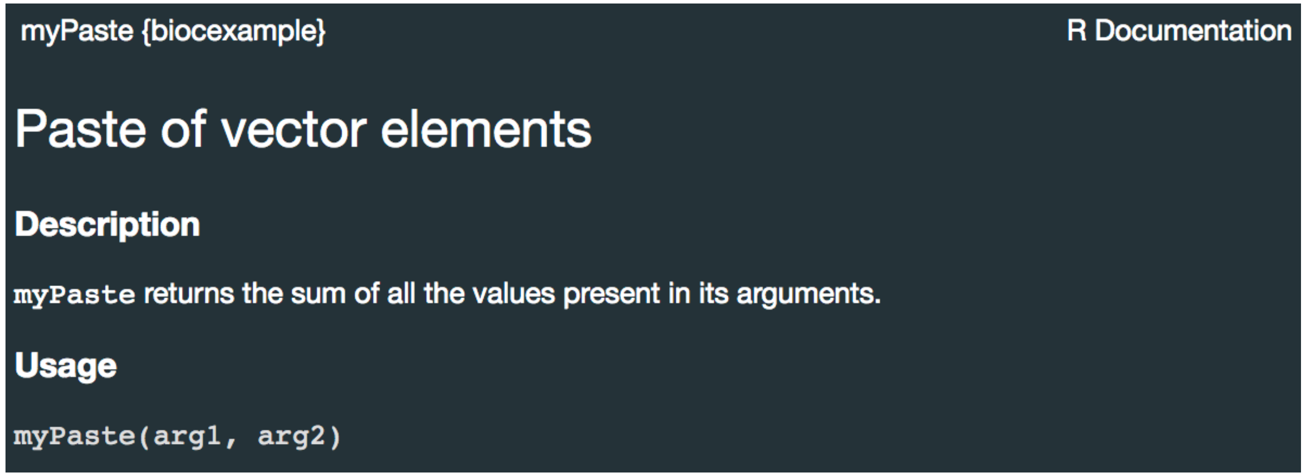
https://cran.r-project.org/web/packages/roxygen2/vignettes/rd.html

Documenting functions

Document functions
- Arguments
- Imports
- examples

```
devtools::document()
```

```
#' @title
#' Paste of vector elements
#'
#' @description
#' `myPaste` returns a string and a numerical #'
value pasted together.


myPaste <- function(arg1, arg2) {
  ...
}
```



myPaste {biocexample} R Documentation

Paste of vector elements

Description

`myPaste` returns the sum of all the values present in its arguments.

Usage

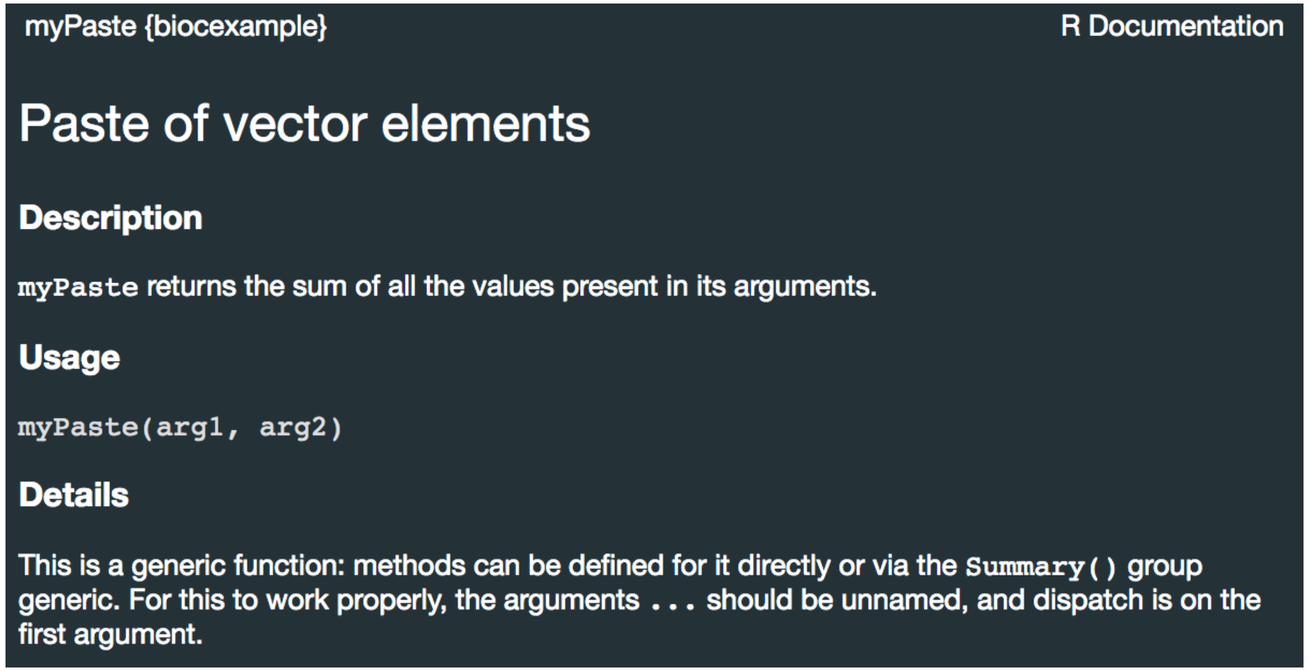
```
myPaste(arg1, arg2)
```

Documenting functions

Document functions
- Arguments
- Imports
- examples

```
devtools::document()
```

```
#' @title
#' Paste of vector elements
#'
#' @description
#' `myPaste` returns a string and a numerical
#' value pasted together.
#'
#' @details
#' This is a generic function: methods can be
#' defined for it directly or via the
#' [Summary()] group generic. For this to work
#' properly, the arguments `...` should be
#' unnamed, and dispatch is on the first
#' argument.


myPaste <- function(arg1, arg2) {
  ...
}
```


myPaste {biocexample} R Documentation

Paste of vector elements

Description

`myPaste` returns the sum of all the values present in its arguments.

Usage

```
myPaste(arg1, arg2)
```

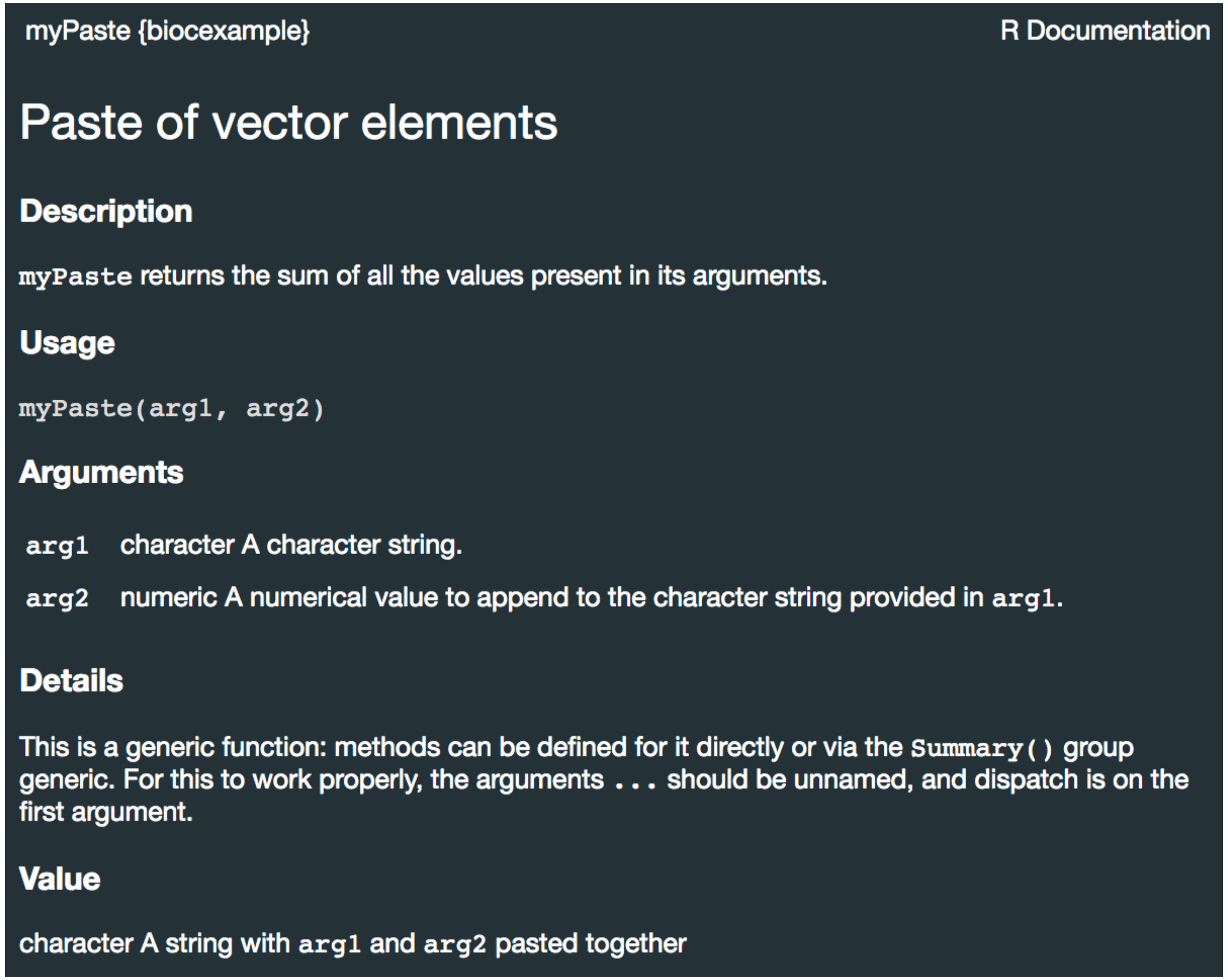
Details

This is a generic function: methods can be defined for it directly or via the `Summary()` group generic. For this to work properly, the arguments `...` should be unnamed, and dispatch is on the first argument.

Documenting functions

Document functions
- Arguments
- Imports
- examples

```
#' @title
#' Paste of vector elements
#'
#' @description
#' `myPaste` returns a string and a numerical
#' value pasted together.
#'
#' @details
#' This is a generic function: methods can be
#' defined for it directly or via the
#' [Summary()] group generic. For this to work
#' properly, the arguments `...` should be
#' unnamed, and dispatch is on the first
#' argument.
#'
#' @param arg1 character A character string.
#' @param arg2 numeric A numerical value to append
#' to the character string provided in \code{arg1}.
#'
#' @returns character A string with \code{arg1} and
#' \code{arg2} pasted together
#'


myPaste <- function(arg1, arg2) {
  ...
}
```



myPaste {biocexample}   R Documentation

Paste of vector elements

Description

`myPaste` returns the sum of all the values present in its arguments.

Usage

```
myPaste(arg1, arg2)
```

Arguments

`arg1`	character A character string.
`arg2`	numeric A numerical value to append to the character string provided in `arg1`.

Details

This is a generic function: methods can be defined for it directly or via the `Summary()` group generic. For this to work properly, the arguments `...` should be unnamed, and dispatch is on the first argument.

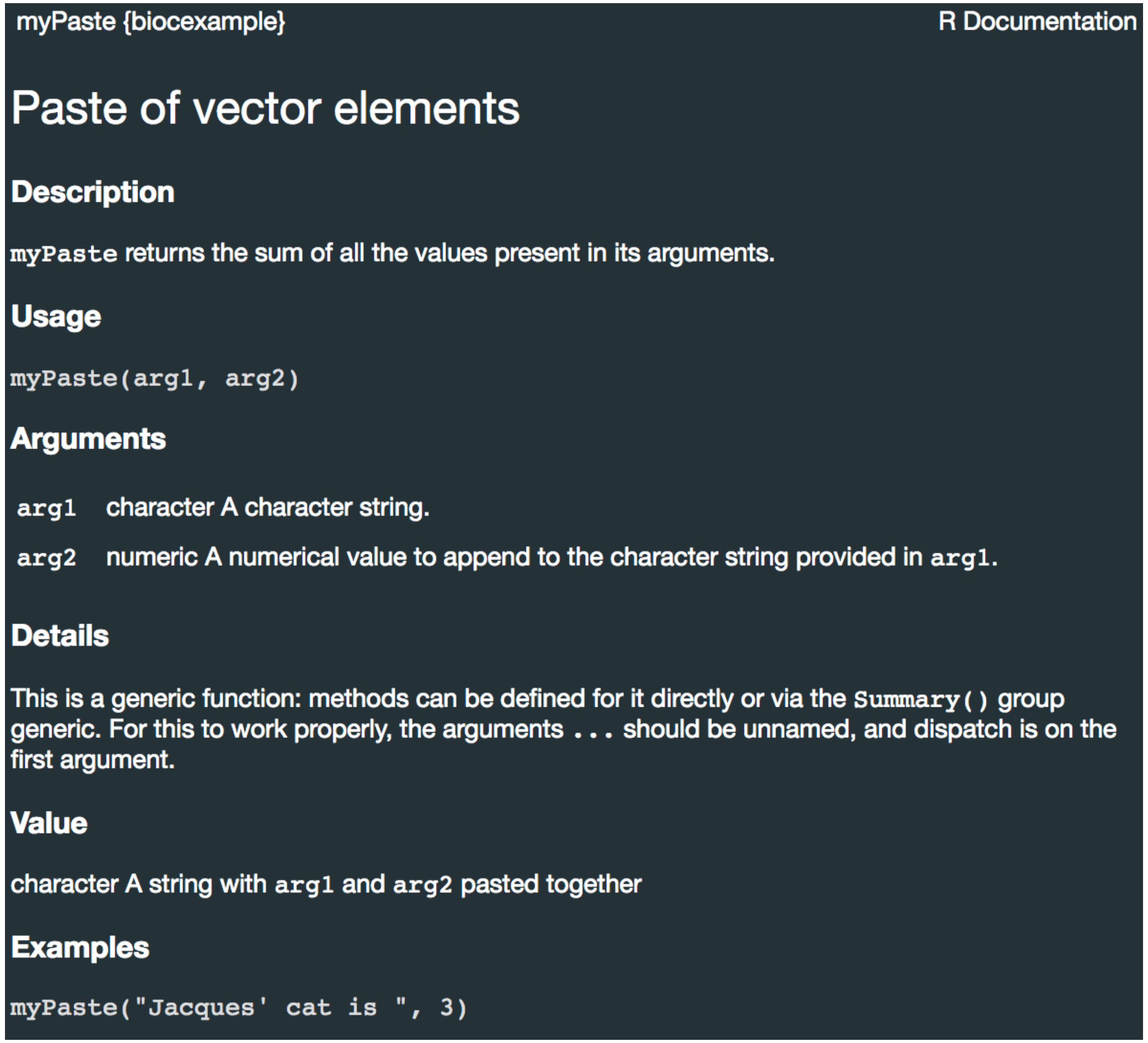
Value

character A string with `arg1` and `arg2` pasted together

Documenting functions

Document functions
- Arguments
- Imports
- examples

```
#' @title
#' Paste of vector elements
#'
#' @description
#' `myPaste` returns a string and a numerical value
#' pasted together.
#'
#' @details
#' This is a generic function: methods can be
#' defined for it directly or via the
#' [Summary()] group generic. For this to work
#' properly, the arguments `...` should be
#' unnamed, and dispatch is on the first
#' argument.
#'
#' @param arg1 character A character string.
#' @param arg2 numeric A numerical value to append
#' to the character string provided in \code{arg1}.
#'
#' @returns character A string with \code{arg1} and
#' \code{arg2} pasted together
#'
#' @examples
#' myPaste("Jacques' cat is ", 3)

myPaste <- function(arg1, arg2) {
  ...
}
```



myPaste {biocexample} R Documentation

Paste of vector elements

Description

`myPaste` returns the sum of all the values present in its arguments.

Usage

```
myPaste(arg1, arg2)
```

Arguments

`arg1`	character A character string.
`arg2`	numeric A numerical value to append to the character string provided in `arg1`.

Details

This is a generic function: methods can be defined for it directly or via the `Summary()` group generic. For this to work properly, the arguments `...` should be unnamed, and dispatch is on the first argument.

Value

character A string with `arg1` and `arg2` pasted together

Examples

```
myPaste("Jacques' cat is ", 3)
```

Documenting functions

Document functions
- Arguments
- Imports
- examples

```r
#' @title
#' Paste of vector elements
#'
#' @description
#' `myPaste` returns a string and a numerical value
#' pasted together.
#'
#' @details
#' This is a generic function: methods can be
#' defined for it directly or via the
#' [Summary()] group generic. For this to work
#' properly, the arguments `...` should be
#' unnamed, and dispatch is on the first
#' argument.
#'
#' @param arg1 character A character string.
#' @param arg2 numeric A numerical value to append
#' to the character string provided in \code{arg1}.
#'
#' @returns character A string with \code{arg1} and
#' \code{arg2} pasted together
#'
#' @examples
#' myPaste("Jacques' cat is ", 3)
#'
#' @importFrom glue glue
#' @export

myPaste <- function(arg1, arg2) {

   ## Internal checkups
   if (!is.character(arg1) | !is.numeric(arg2)) {
      stop("There has been an error. Aborting now.")
   }

   ## Internal processing steps
   res <- glue::glue(arg1, arg2)

   ## Return result
   return(res)

}
```

Documenting functions

Document functions
  Arguments
  Imports
  examples

- Do not forget to `@export` the user-level functions!
- Internal functions (those that should not be used by regular users) should start with a dot (`.`).

```r
#' internal_check function
#'
#' This function is not meant to be used
#' interactively
#'
#' @param arg
#' @return logical

.check_fun <- function(arg) {
  if (...) return(TRUE)
}
```

```r
#' import function
#'
#' @param path
#' @return Value
#' @export

import <- function(path) {
  .check_fun()
  ...
}
```

Testing functions

Test functions

- Tests are implemented to make sure each fundamental brick of your package works, but also that the whole package in itself works (especially if there are many complex, nested functions)

Testing functions

Test functions

- Tests are implemented to make sure each fundamental brick of your package works, but also that the whole package in itself works (especially if there are many complex, nested functions)

"Rien ne sert de courir, mieux vaut partir à point." (Slow and steady wins the race)

– The tortoise in that kid story

– But for real: Jean de la Fontaine

Testing functions

Test functions

- Tests are implemented to make sure each fundamental brick of your package works, but also that the whole package in itself works (especially if there are many complex, nested functions)

- Always put .R files containing <u>**tests**</u> in `**tests/testthat**`. The easiest to create these files is to run `use_test("...")`.

Testing functions

Test functions

- Tests are implemented to make sure each fundamental brick of your package works, but also that the whole package in itself works (especially if there are many complex, nested functions)

- Always put .R files containing **<u>tests</u>** in `**tests/testthat**`. The easiest to create these files is to run `use_test("...")`.

```
> usethis::use_testthat()

✓ Setting active project to
'/Users/jacquesserizay/biocexample'
✓ Adding 'testthat' to Suggests field in
DESCRIPTION
✓ Setting Config/testthat/edition field
in DESCRIPTION to '3'
✓ Creating 'tests/testthat/'
✓ Writing 'tests/testthat.R'
• Call `use_test()` to initialize a
basic test file and open it for editing.
```

Testing functions

Test functions

- Tests are implemented to make sure each fundamental brick of your package works, but also that the whole package in itself works (especially if there are many complex, nested functions)

- Always put .R files containing **tests** in `**tests/testthat**`. The easiest to create these files is to run `use_test("...")`.

```
> usethis::use_testthat()

✓ Setting active project to
'/Users/jacquesserizay/biocexample'
✓ Adding 'testthat' to Suggests field in
DESCRIPTION
✓ Setting Config/testthat/edition field
in DESCRIPTION to '3'
✓ Creating 'tests/testthat/'
✓ Writing 'tests/testthat.R'
• Call `use_test()` to initialize a
basic test file and open it for editing.
```

```
> usethis::use_test(
    name = ‘myPaste’
)
```

Testing functions

Test functions

- Tests are implemented to make sure each fundamental brick of your package works, but also that the whole package in itself works (especially if there are many complex, nested functions)

- Always put .R files containing **tests** in `**tests/testthat**`. The easiest to create these files is to run `use_test("...")`.

```
> usethis::use_testthat()

✓ Setting active project to
'/Users/jacquesserizay/biocexample'
✓ Adding 'testthat' to Suggests field in
DESCRIPTION
✓ Setting Config/testthat/edition field
in DESCRIPTION to '3'
✓ Creating 'tests/testthat/'
✓ Writing 'tests/testthat.R'
• Call `use_test()` to initialize a
basic test file and open it for editing.
```

```
> usethis::use_test(
    name = ‘myPaste’
)
```

Tests/testthat/test-myPaste.R

```
test_that("myPaste works", {
  expect_equal(
    myPaste("Jacques is ", 30),
    "Jacques is 30"
  )
})
```

Testing functions

Test functions

- Tests are implemented to make sure each fundamental brick of your package works, but also that the whole package in itself works (especially if there are many complex, nested functions)

- Always put .R files containing <u>**tests**</u> in `**tests/testthat**`. The easiest to create these files is to run `use_test("...")`.

```
> usethis::use_testthat()

✓ Setting active project to
'/Users/jacquesserizay/biocexample'
✓ Adding 'testthat' to Suggests field in
DESCRIPTION
✓ Setting Config/testthat/edition field
in DESCRIPTION to '3'
✓ Creating 'tests/testthat/'
✓ Writing 'tests/testthat.R'
• Call `use_test()` to initialize a
basic test file and open it for editing.
```

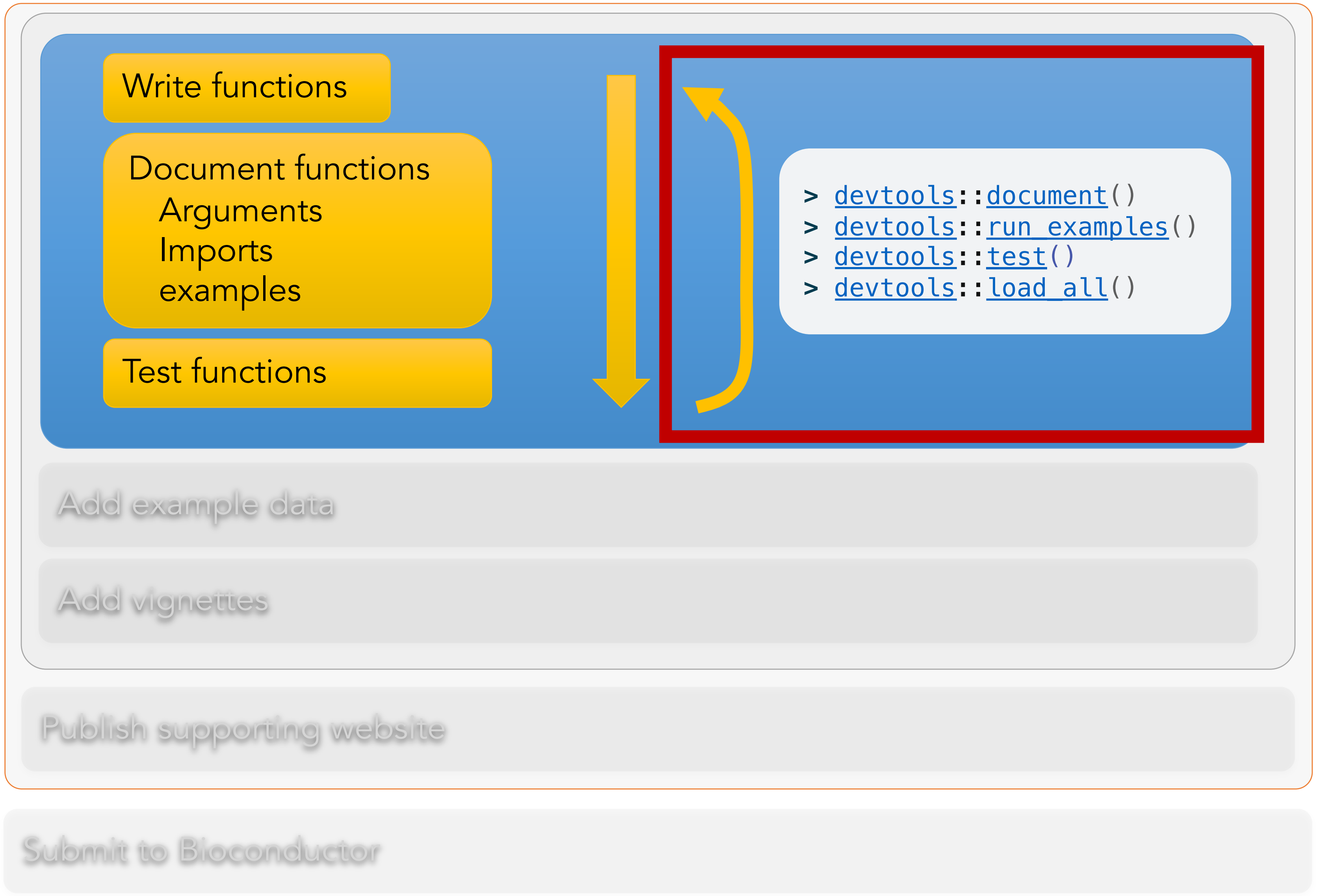
```
> usethis::use_test(
    name = 'myPaste'
)
```

Tests/testthat/test-myPaste.R

```
test_that("myPaste works", {
  expect_equal(
    myPaste("Jacques is ", 30),
    "Jacques is 30"
  )
})
```

```
> devtools::test()

i Loading biocexample
i Testing biocexample
✓ | OK F W S | Context
✓ |  1 | myPaste [0.2 s]

══ Results ═══════════════════════
Duration: 0.2 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 1 ]

Woot!
```

Rinse-and-repeat!!!



```
> devtools::document()
> devtools::run_examples()
> devtools::test()
> devtools::load_all()
```

```
myPackage/
  R/
    functions.R
    utils.R
  man/
    myfunction.Rd
  tests/
    testthat.R
    testthat/
      test-myfun.R
  DESCRIPTION
  README.md
  NAMESPACE
  NEWS
  LICENSE
```

Thorough checks

Tools to run longer, integrated checks:

- devtools::check()
- BiocCheck::BiocCheck()
- rcmdcheck:: rcmdcheck()