Intro to Functions in R

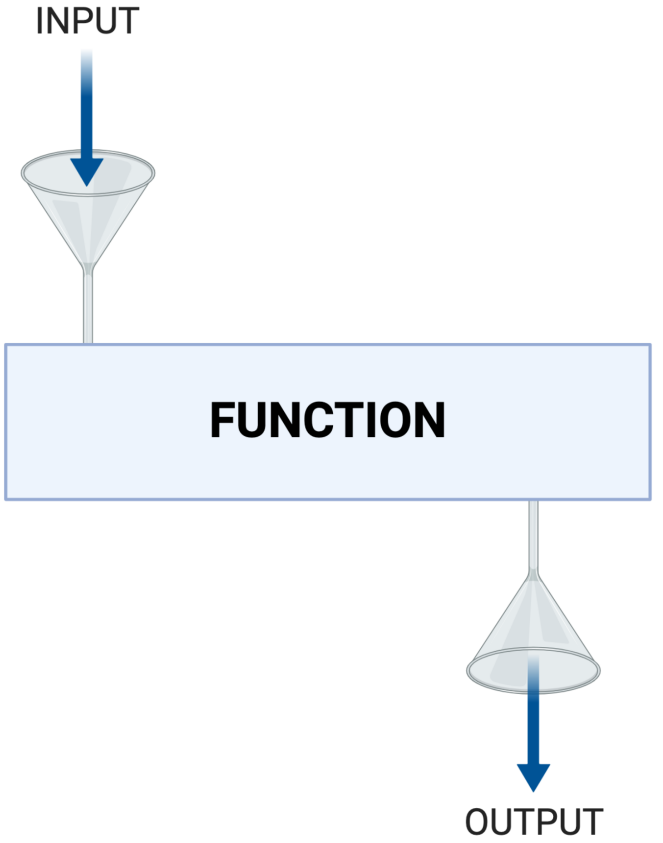
Martha Cooper, tRopical_R

Get these slides:
github.com/MarthaCooper/CTBMB_Retreat_2019

2019/09/27 (updated: 2020-08-03)

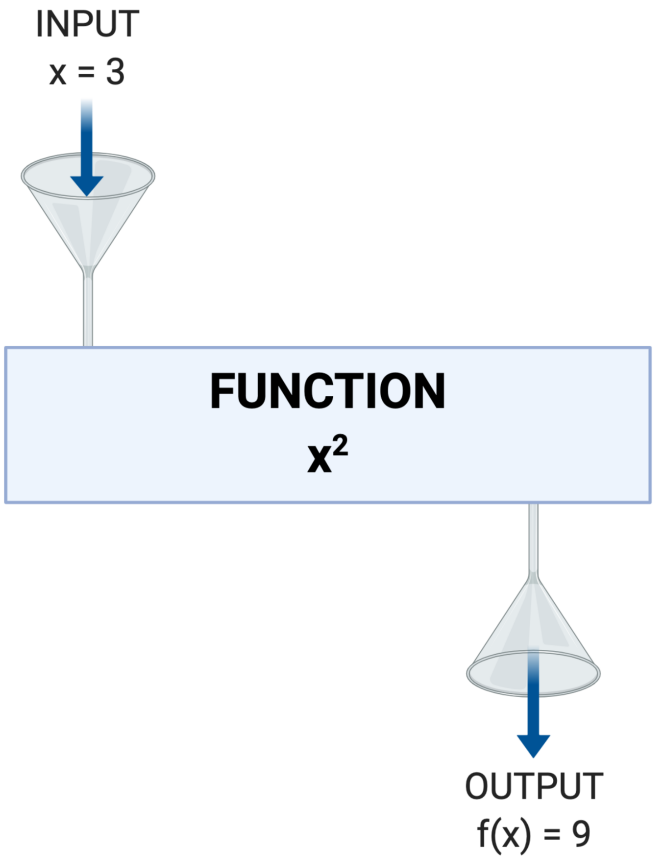
What is a function?

- A function is a reusable piece of code that does a specific task



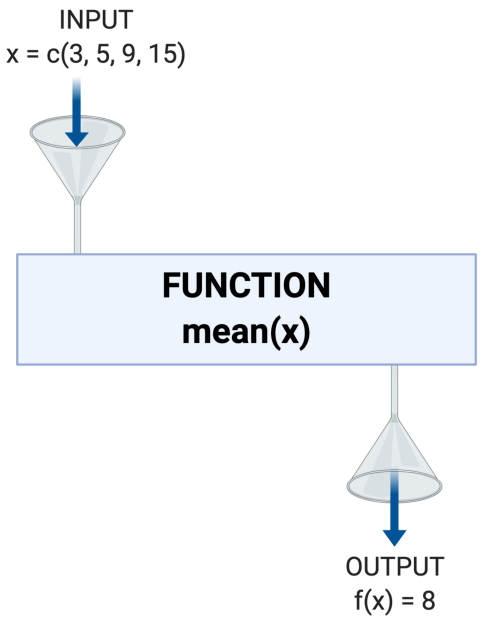
What is a function?

- A function is a resuable piece of code that does a specific task



R has built in functions

- Schematic



- R code

```r
#input
x <- c(3,5,9,15)


#function
mean(x)
```

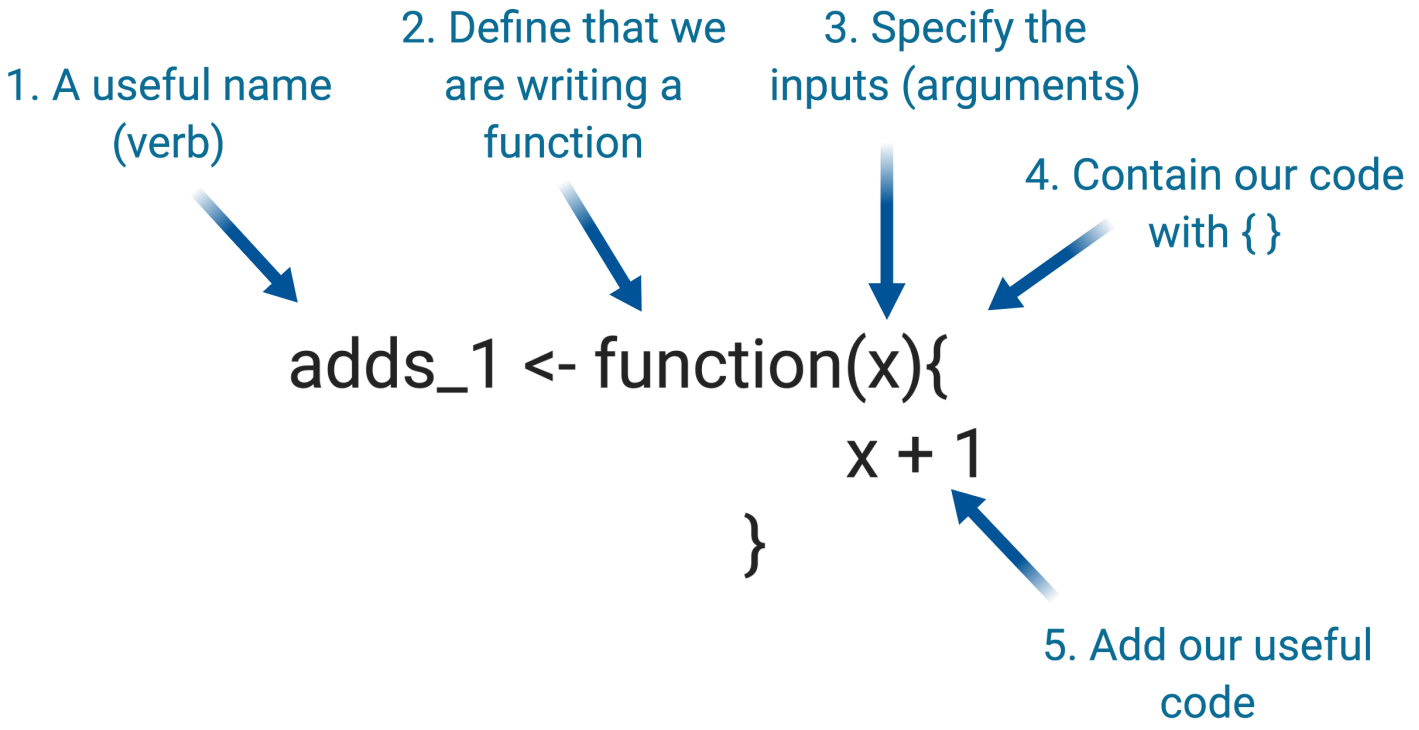
```
## [1] 8
```

Function Arguments

- Function inputs are called arguments
- `mean()` has 3 arguments
- `?mean` to view arguments

```
x # a numeric vector

trim # fractions of outliers
     # you would like to
     # trim from each end

na.rm # option to remove
      # missing values
```

Function Syntax



Function Syntax

```
add_1 <- function(x){
  x + 1
}
```

The signature is the user interface

```
function(x)
```

The body

```
{
  x + 1
}
```

"You should consider writing a function whenever you've copied and pasted a block of code more than twice"

Hadley Wickham

3 reasons to use functions

1. It makes code easier to understand

- You can give the function an evocative name that makes your code easier to understand

```
## without using a function

x + 1

## with using a function

add_1 <- function(x){
  x + 1
}

  add_1(x)
```

- Emphasizes the action that is being done

3 reasons to use functions

2. Making changes to your code is easier

- You only need to make the change in one place - in the function

```
#### It is easy to modify a function

add_1 <- function(x){
  x + 1
}

add_2 <- function(x){
  x + 2
}
```

3 reasons to use functions

3. Less mistakes

- It becomes easy to repeat steps in your analysis multiple times

```
add_1(x)

add_1(y)

add_1(df$i)
```

- And there is no danger you'll make mistakes by copy-and-pasting chunks of code

Embedding Functions

```
# Experiment_1
exp_1 <- read.csv("exp_1_data.csv") #import
exp_1_filtered <- filter(exp1, value < 10) #filter
```

Embedding Functions

```
# Experiment_1
exp_1 <- read.csv("exp_1_data.csv") #import
exp_1_filtered <- filter(exp1, value < 10) #filter
```

```
# Experiment_2
exp_2 <- read.csv("exp_2_data.csv") #import
exp_2_filtered <- filter(exp2, value < 10) #filter
```

Embedding Functions

```
# Experiment_1
exp_1 <- read.csv("exp_1_data.csv") #import
exp_1_filtered <- filter(exp1, value < 10) #filter
```

```
# Experiment_2
exp_2 <- read.csv("exp_2_data.csv") #import
exp_2_filtered <- filter(exp2, value < 10) #filter
```

```
# Experiment_3
exp_3 <- read.csv("exp_3_data.csv") #import
exp_3_filtered <- filter(exp2, value < 10) #filter
```

Embedding Functions

```r
# Experiment_1
exp_1 <- read.csv("exp_1_data.csv") #import
exp_1_filtered <- filter(exp1, value < 10) #filter
```

```r
# Experiment_2
exp_2 <- read.csv("exp_2_data.csv") #import
exp_2_filtered <- filter(exp2, value < 10) #filter
```

```r
# Experiment_3
exp_3 <- read.csv("exp_3_data.csv") #import
  exp_3_filtered <-  filter(exp2, value < 10)
```

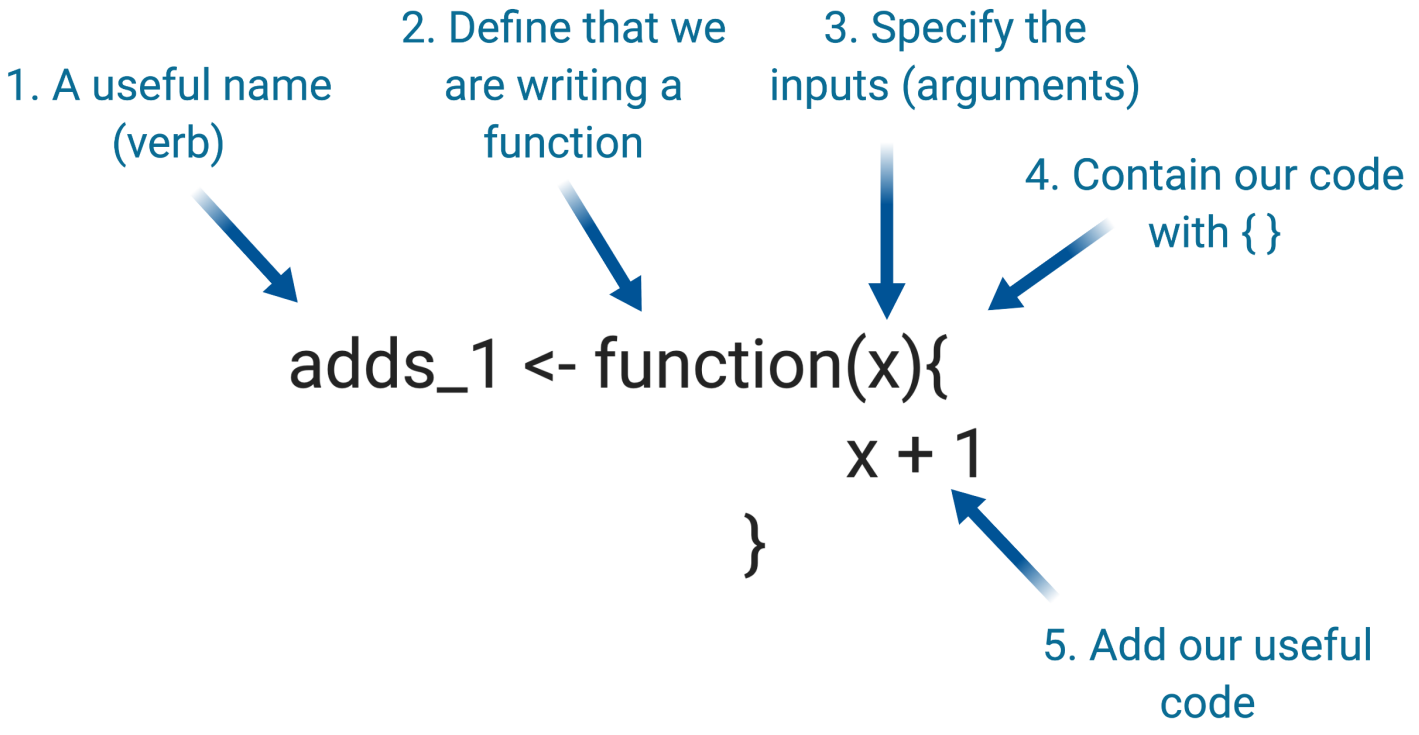
Embedding Functions

- Write a function to load and filter the dataset
- Easier to read, less mistakes

```
#write a function to import and filter data
import_and_filter <- function(path_to_file){

  x <- read.csv(path_to_file) #import
  x <- filter(x, value < 10) #filter
  return(x)
}
```

```
#Use function
exp_1 <- import_and_filter("exp_1_data.csv")
exp_2 <- import_and_filter("exp_2_data.csv")
exp_3 <- import_and_filter("exp_3_data.csv")
```

How do you write your own functions?



Write the following statement as a function

- What does this code do?
- How many arguments does it need?

```
(x / y)*100
```

Write the following statement as a function

- What does this code do?
- How many arguments does it need?

```
percentage <- function(x, y){
  (x / y)*100
}
```

Write your own function that multiplies the input by 10

Write your own function that multiplies the input by 10

1. A useful name?

```
times10 <-
```

Write you own function that multiplies the input by 10

2. Define that you are writing a function

```
times10 <- function
```

Write your own function that multiplies the input by 10

3. Add your inputs (arguments)

```
times10 <- function(x)
```

Write your own function that multiplies the input by 10

3. Surround your code with { }

```
times10 <- function(x){

}
```

Write your own function that multiplies the input by 10

4. Add your useful code

```
times10 <- function(x){
  10*x
}
```

Write your own function that multiplies the input by 10

```
times10 <- function(x){
  10*x
}

x <- 35
times10(x)
```

```
## [1] 350
```

Documentation

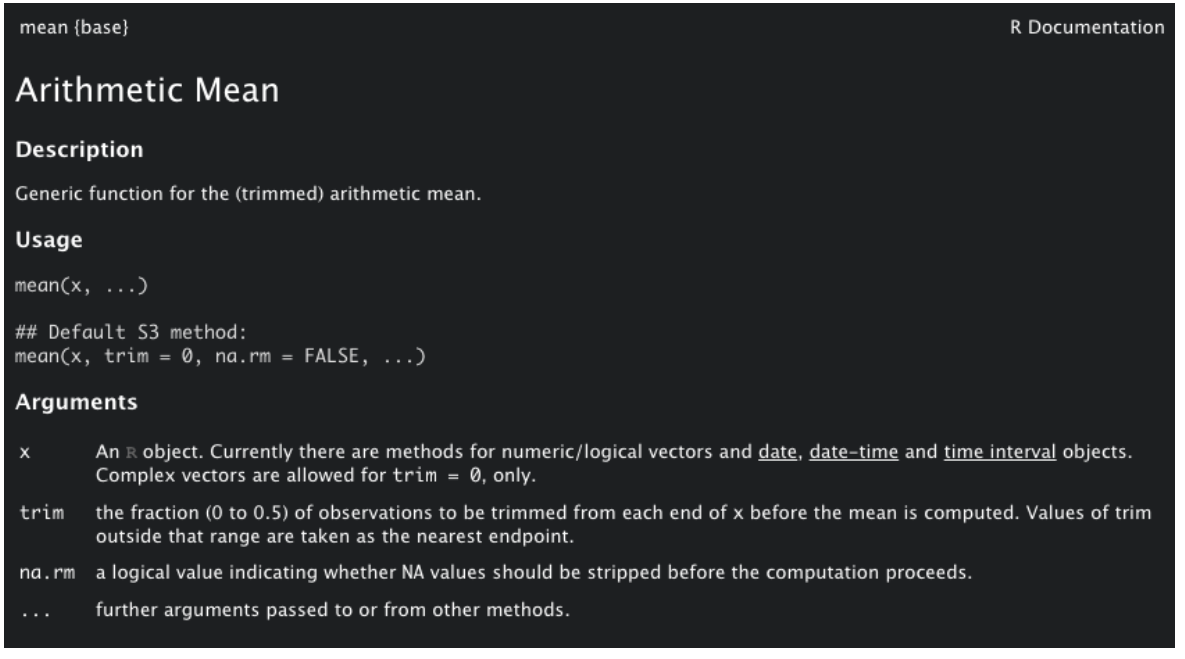
Documentation is essential, even if the only person reading your code will be you in the future

```
#multiplies the input by ten
#input must be numeric
times10 <- function(x){
  10*x
}
```

Documentation

Documentation for existing R functions can be found using ?

```
?mean
```



mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

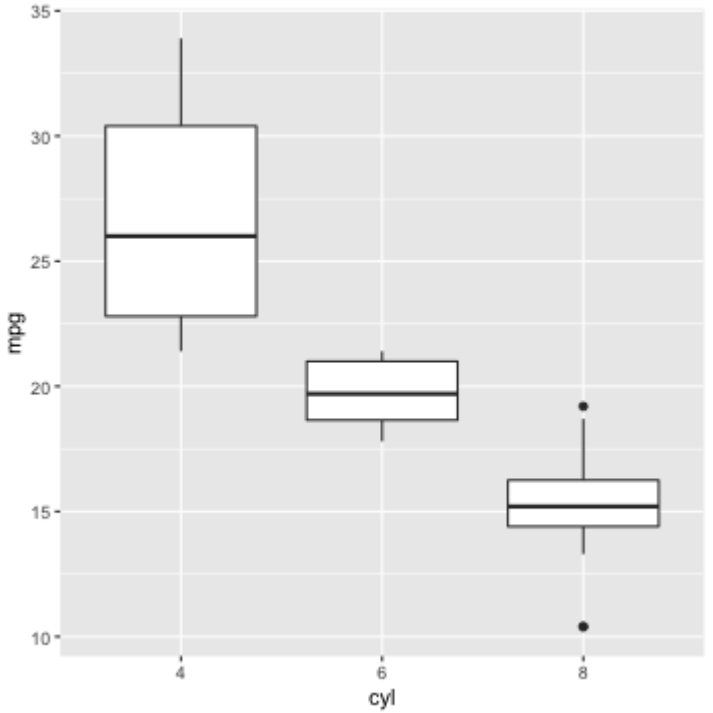
```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

x	An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for trim = 0, only.
trim	the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.
...	further arguments passed to or from other methods.

Packages contain functions

```
library(ggplot2)
ggplot(data = mtcars, aes(x = as.factor(cyl), y = mpg))+
  geom_boxplot()+
  xlab("cyl")
```



Embed the following functions in your own function

```
my_data <- read.csv("my_csv_file.csv")
my_processed_data <- mutate(my_data, total = sum)
```

Embed the following functions in your own function

```
input_and_process <- function(my_csv_file){
  my_data <- read.csv(my_csv_file)
  my_processed_data <- mutate(my_data, total = sum(my_column))
  return(my_processed_data)
}
```

Real World Challenge

You have 70 datafiles in csv format

- You need to import each one
- Transform each dataset from wide to long format
- Save each one to a new CSV file

Solve with a function

Next: putting your function into a package...

Slides available at: github.com/MarthaCooper/CTBMB_Retreat_2019

- Slides created via the R package xaringan
- Theme created with the R package xaringanthemer
- Help has come from remark.js, knitr, and R Markdown.