- Institut Pasteur de Madagascar
- December 2022

Writing For Loops, If-Else Statements, and Functions in R

- **E^2M^2: Ecological and Epidemiological Modeling in Madagascar**

The Power of Programming

- So far, much of what we saw demonstrates how to use R like an extremely smart calculator.
  - We write commands and it executes them.

The Power of Programming

- So far, much of what we saw demonstrates how to use R like an extremely smart calculator.

- The true power of the program comes from allowing R to query large datasets and make decisions for you.

The Power of Programming

- So far, much of what we saw demonstrates how to use R like an extremely smart calculator.
- The true power of the program comes from allowing R to query large datasets and make decisions for you.
- Three key programming tools are helpful:
  1. If-else statements
  2. For-loops
  3. Functions

The Power of Programming

- So far, much of we learned demonstrates how to use R like an extremely smart calculator.
- The true power of the program comes from allowing R to query large datasets and make decisions for you.
- Three key programming tools are helpful:
  1. If-else and ifelse statements
  2. For-loops
  3. Functions

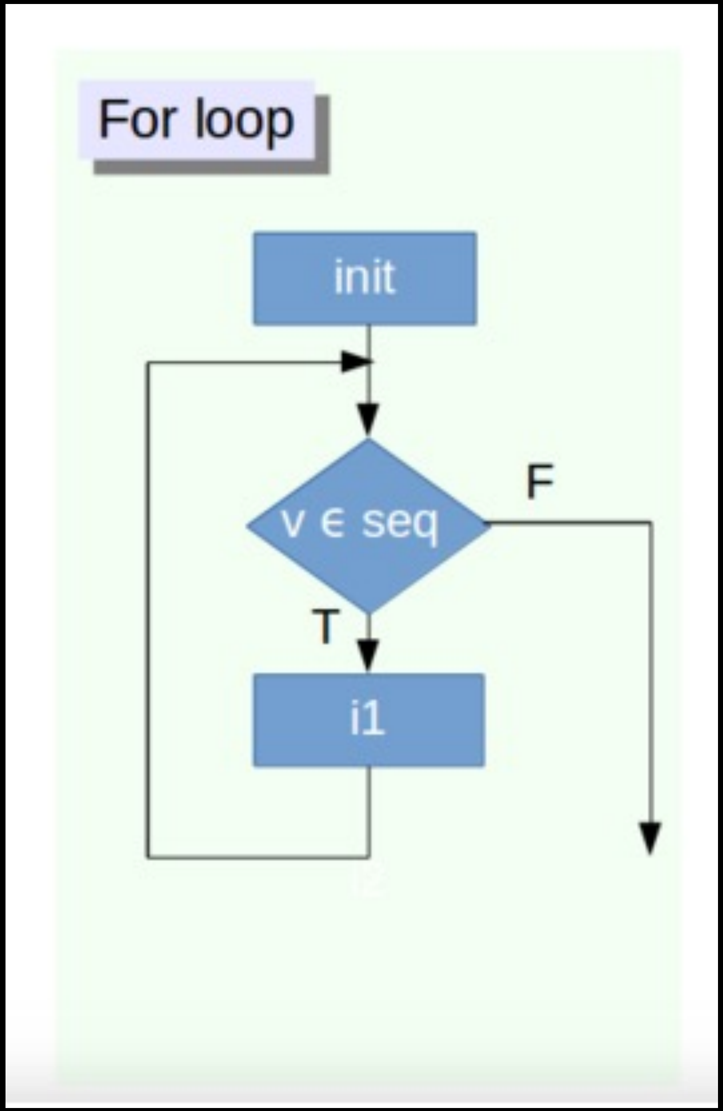
  Allow you to control the flow of our programming and cause different things to happen depending on the value of tests

For-loops

- “Looping”, “cycling”, “iterating” is nothing more than automating a multi-step process by organizing sequences of actions or ‘batch’ processes and by grouping the parts that need to be repeated.
- For loops execute for a prescribed number of times, as controlled by a counter or an index, incremented at each iteration cycle.

For-Loops

```
for (variable in vector)
        { do something }
```



```
for (i in 1:20) {
print(paste("I am student",i))
}
```

Tells the loop how many times to run

```
for (i in 1:20) {
print(paste("I am student",i))
}
```

Tells the loop how many times to run

```
for (i in 1:20){
print(paste("I am student",i))
}
```

Function to be run i times

```
for (i in 1:20) {
print(paste("I am student",i))
}
```

The print command is very important.
Without it the functions will only run internal to the loop

If Statements

If condition is TRUE, then perform some action; otherwise do not perform that action.

```
if (condition is TRUE)
        { do something }
```

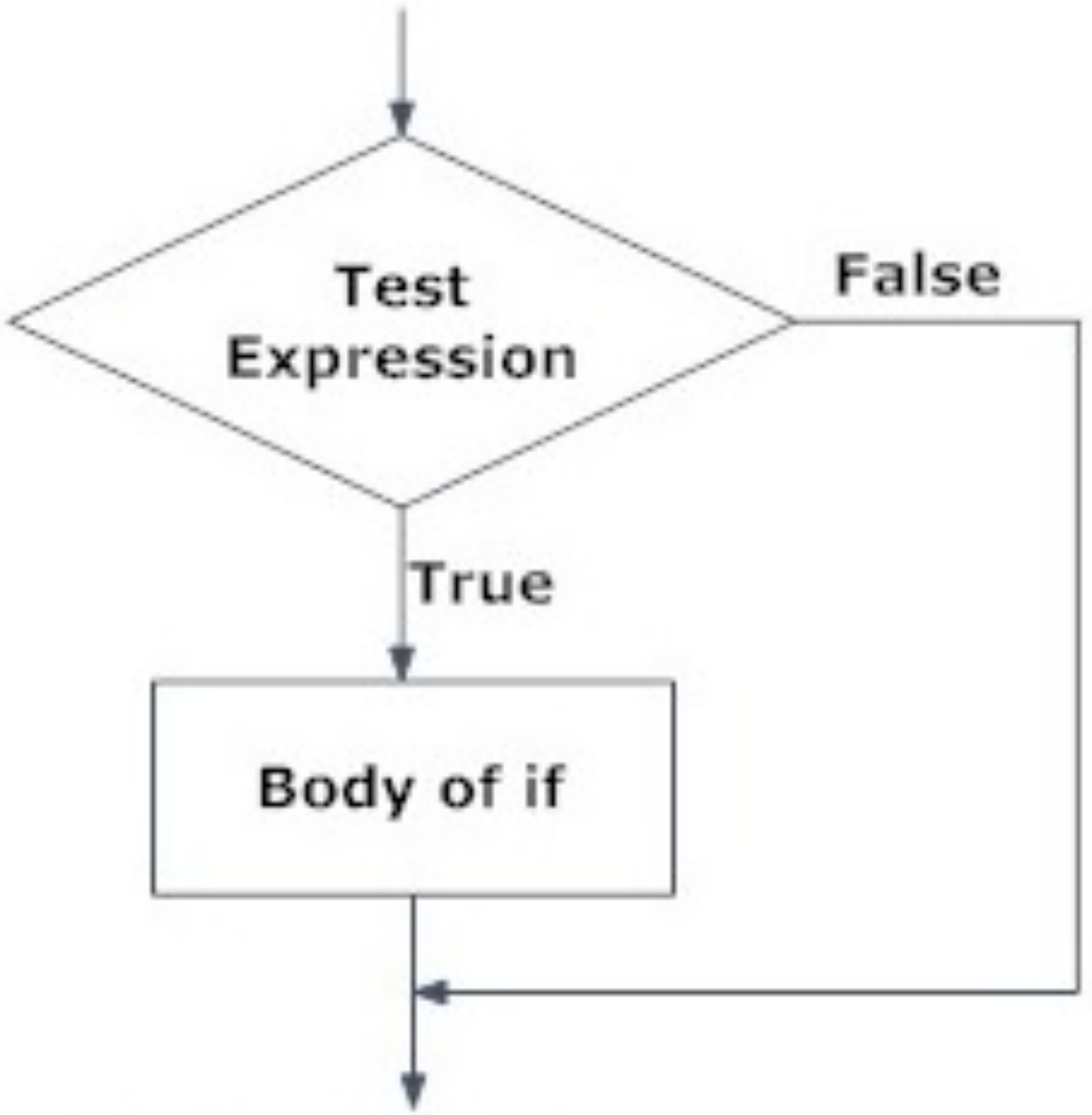


Fig: Operation of if statement

If-Else Statements

If condition is TRUE, then perform some action; otherwise do not perform that action.

```
if (condition is TRUE)
            { do something }
else { do different thing }
```

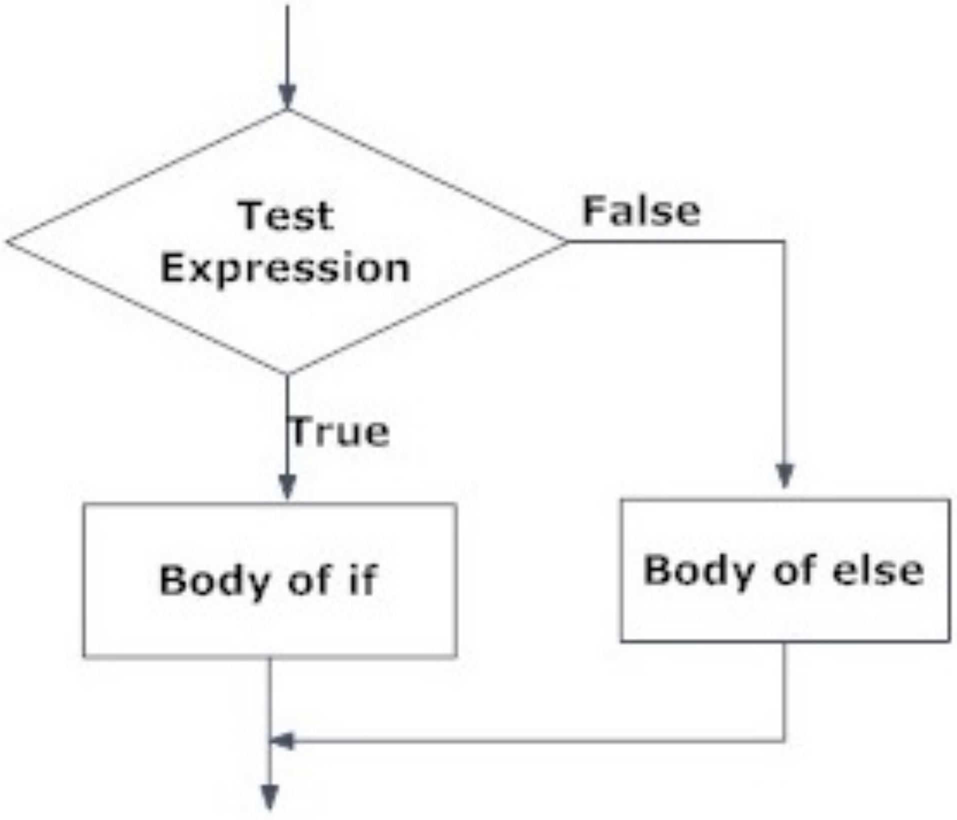


Fig: Operation of if...else statement

If-Else Statements

If condition is TRUE, then perform some action; otherwise do not perform that actic

if (condition is TRUE)

{ do something } else

{ do different thing }

IMPORTANT: else must be in the same line as the closing braces of the if statement.

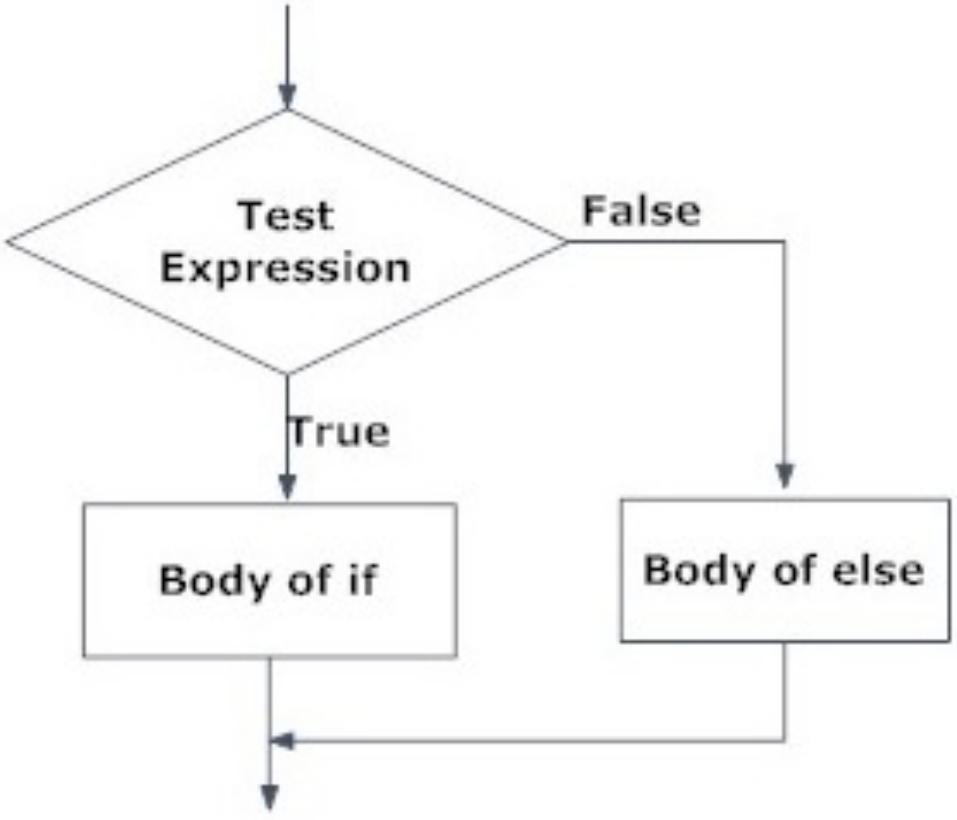
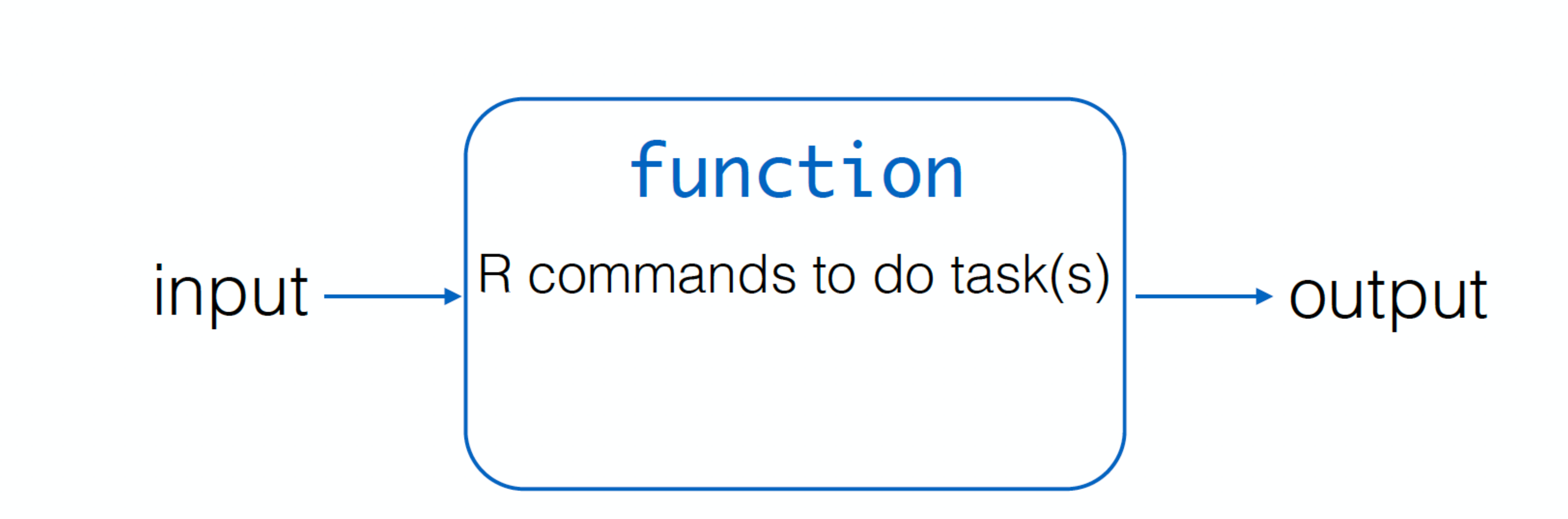


Fig: Operation of if...else statement

Functions

- A function is a piece of code written to carry out a specified task;



input

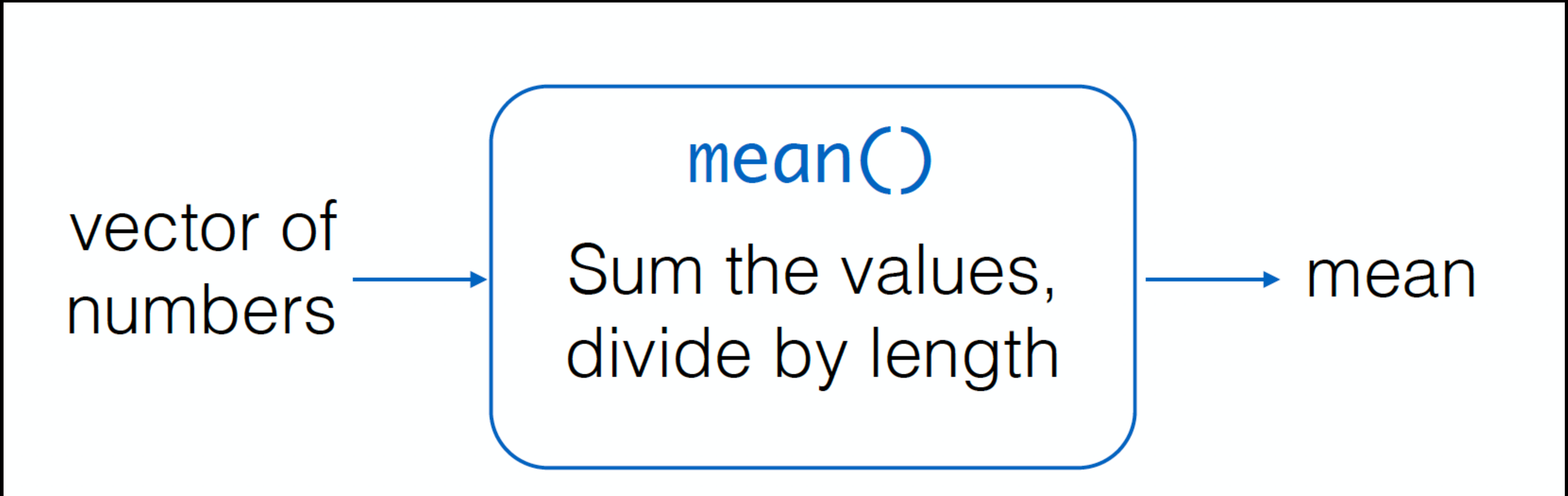
function

R commands to do task(s)

output

Functions

- A function is a piece of code written to carry out a specified task;
- mean(x), sum(x),….rep(x,y)



vector of numbers → mean() Sum the values, divide by length → mean

Functions

- A function is a piece of code written to carry out a specified task;
- mean(x), sum(x),….rep(x,y)
- Lots of pre-written functions organized in multitude of packages.
- If you can not find a function in R to do what you need, you can write your own function

Why write functions?

- Any time you find yourself wanting to do the same thing many times
- Editing data
- Repeating similar analyses on different variables
- Creating a similar graph from different variables
- Running simulations
- Lots of other reasons I’m sure…

Functions

```
function_name <- function(argument1, argument2) {
                  command
return(output)}
```

where the code in between the curly braces is the *body* of the function.

Functions

- Things to consider:
  - Function allows you to define exactly what you want to do
  - Name your User Defined Function.
  - Make sure that the name that you choose for the function is not an R reserved word. This means that you, for example, don't want to pick the name of an existing function for your own UDF.

- Start with a very simple version of what you want to accomplish and build from there

- You want to make sure each little piece works before you invest the time to create a complex thing:

- **Remember: you can always try to run any line of code you are confused about!**

We want to simulate a coin toss

- We want to simulate a coin toss and find out the proportion of tails that are recovered for n different toss trials.

```
coin<-function(n){
Tail<-rbinom(n,1,.5)
numTail<-sum(Tail)
propTail <- numTail/n
return(propTail)
}
```

```
coin<-function(n){
conduct n toss trials with a 50% prob. of getting tail
Tail<-rbinom(n,1,.5)
numTail<-sum(Tail)
propTail <- numTail/n
return(propTail)
}
```

```
coin<-function(n){
```

conduct n toss trials with a 50% prob. of getting tail

```
Tail<-rbinom(n,1,.5)
```

count number of Tails

```
numTail<-sum(Tail)
propTail <- numTail/n
return(propTail)
}
```

```
coin<-function(n){
conduct n toss trials with a 50% prob. of getting tail
Tail<-rbinom(n,1,.5)
count number of Tails
numTail<-sum(Tail)
divide number of Tails by number of trials
propTail <- numTail/n
return(propTail)
}
```

```
coin<-function(n){
conduct n toss trials with a 50% prob. of getting tail
Tail<-rbinom(n,1,.5)
count number of Tails
numTail<-sum(Tail)
divide number of Tails by number of trials
propTail <- numTail/n
return(propTail)
} return() determines what the product of the function is
```

```
coin<-function(n){
conduct n toss trials with a 50% prob. of getting tail
Tail<-rbinom(n,1,.5)
count number of Tails
numTail<-sum(Tail)
divide number of Tails by number of trials
propTail <- numTail/n
return(propTail)
} return() determines what the product of the function is
```

Help me add flexibility to this function by allowing me to change the probability of getting tails!

Take home messages

- Start small and build up
- Work out the kinks bit by bit before investing too much time into writing a big function

Take home messages

- Start small and build up
- Work out the kinks bit by bit before investing too much time into writing a big function
- Things that can look very complex at first can be broken down into small parts, which makes them less threatening

Take home messages

- Start small and build up
- Work out the kinks bit by bit before investing too much time into writing a big function
- Things that can look very complex at first can be broken down into small parts, which makes them less threatening
- Writing functions and simulations is not that hard, you have all the tools already!