Functions

Stat 220

Bastola

February 04 2022

What are functions?

A function is a type of object in R that can perform a specific task.

- Functions take arguments as input and output some manipulated form of the input data.
- A function is specified first with the object name, then parentheses with arguments inside.

```
# a simple in-built function
log(4)
[1] 1.386294
```

When to write functions?

- Using the same code more than once
- Complicated operation
- Vectorization

Function arguments

- `x`, `y`, `z`: vectors.
- `w`: a vector of weights.
- `df`: a data frame.
- `i`, `j`: numeric indices (typically rows and columns).
- `n`: length, or number of rows.
- `p`: number of columns.

Writing Functions

```
# Basic Set Up
my_awesome_function <- function(x,y) # Arguments broken up by commas
{ # Brackets that house the code
  # Some code to execute
  z = x*y
  return(z) # Return a data value
} # Close the Brackets
```

Writing Functions

```
# Basic Set Up
my_awesome_function <- function(x,y) # Arguments broken up by commas
{ # Brackets that house the code
  # Some code to execute
  z = x*y
  return(z) # Return a data value
} # Close the Brackets
```

Writing Functions

```
# Basic Set Up
my_awesome_function <- function(x,y) # Arguments broken up by commas
{ # Brackets that house the code
  # Some code to execute
  z = x*y
  return(z) # Return a data value
} # Close the Brackets
```

Writing Functions

```
# Basic Set Up
my_awesome_function <- function(x,y) # Arguments broken up by commas
{ # Brackets that house the code
  # Some code to execute
  z = x*y
  return(z) # Return a data value
} # Close the Brackets
```

Writing Functions

```
# Basic Set Up
my_awesome_function <- function(x,y) # Arguments broken up by commas
{ # Brackets that house the code
  # Some code to execute
  z = x*y
  return(z) # Return a data value
} # Close the Brackets
```

Writing Functions

```
# Basic Set Up
my_awesome_function <- function(x,y) # Arguments broken up by commas
{ # Brackets that house the code
  # Some code to execute
  z = x*y
  return(z) # Return a data value
} # Close the Brackets
```

Writing Functions

```
# Basic Set Up
my_awesome_function <- function(x,y) # Arguments broken up by commas
{ # Brackets that house the code
  # Some code to execute
  z = x*y
  return(z) # Return a data value
} # Close the Brackets
```

```
my_awesome_function(x=5,y=6)
[1] 30
```

Writing Functions

```
# Basic Set Up
my_awesome_function <- function(x,y) # Arguments broken up by commas
{ # Brackets that house the code
  # Some code to execute
  z = x*y
  return(z) # Return a data value
} # Close the Brackets
```

```
my_awesome_function(x=5,y=6)
[1] 30
```

```
my_awesome_function(x=7,y=8)
[1] 56
```

More complicated functions

```
Operation_Times <- function(x, y, op){
   first_op <- switch(op,
     plus = x + y,
     minus = x - y,
     times = x * y,
     divide = x / y,
     stop("Unknown op!")
   )

   return(first_op * 2)
}
```

```
Operation_Times(x=5, y=6, op = "plus")
[1] 22
```

More complicated functions

```
Operation_Times <- function(x, y, op){
   first_op <- switch(op,
     plus = x + y,
     minus = x - y,
     times = x * y,
     divide = x / y,
     stop("Unknown op!")
   )

   return(first_op * 2)
}
```

```
Operation_Times(x=5, y=6, op = "plus")
[1] 22
```

```
Operation_Times(x=5, y=6, op = "minus")
[1] -2
```

Your Turn 1

Please git clone the repository on simple functions to your local folder. Write functions to calculate the variance and skewness of $X = \{12, 45, 54, 34, 56, 30, 67, \mathrm{NA}\}$.

$$\mathrm{Var}(x) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

$$\mathrm{Skew}(x) = \frac{\frac{1}{n-2}\left(\sum_{i=1}^{n} (x_i - \bar{x})^3\right)}{\mathrm{Var}(x)^{3/2}}$$

where $x_i = (\sum_i^n x_i)/n$ is the sample mean.

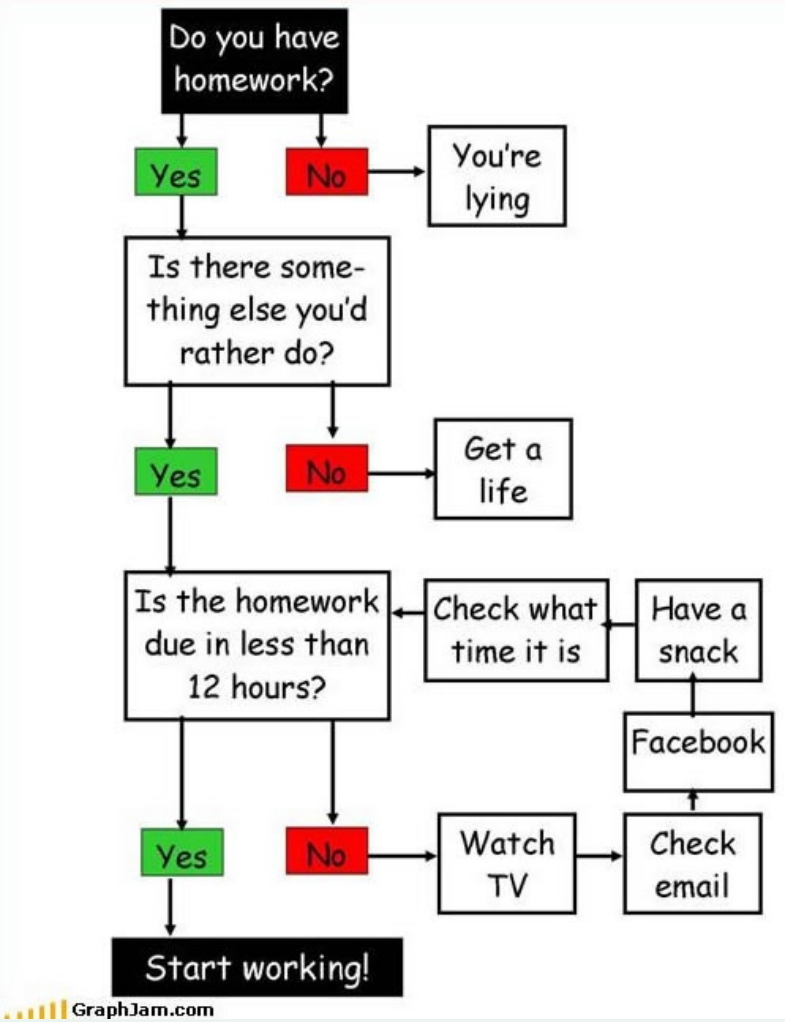
HInt: Use $\sum()$

04:00

Conditional Execution

Allows code to:

- become more flexible
- adapt to the input arguments
- have certain “control flow” constructs



if - else

```
if(TRUE){
  print("Positive")
}else{
  print("Negative")
}
[1] "Positive"
```

```
if(FALSE){
  print("Positive")
}else{
  print("Negative")
}
[1] "Negative"
```

ifelse()

- Same idea just vectorized

```
ifelse(T,"Positive","Negative")
[1] "Positive"
```

```
ifelse(F,"Positive","Negative")
[1] "Negative"
```

```
x <- 1:10
ifelse( x<5, "Positive","Negative")
 [1] "Positive" "Positive" "Positive" "Positive" "Negative" "Negative"
 [7] "Negative" "Negative" "Negative" "Negative"
```

if and ifelse

```
x <- c(3, 4, 6)
y <- c("5", "c", "9")
```

```
# Use `if` for single condition tests
cutoff_make0 <- function(x, cutoff = 0){
  if(is.numeric(x)){
    ifelse(x < cutoff, 0, x)
  } else warning("The input provided is not a numeric vector")
}
```

```
cutoff_make0(x, cutoff = 4)
[1] 0 4 6
```

```
cutoff_make0(y, cutoff = 4)
Warning in cutoff_make0(y, cutoff = 4): The input provided is not a numeric
vector
```

Your Turn 2

Write a function that turns (e.g.) a vector $X = \{\text{“}a\text{”}, \text{“}b\text{”}, \text{“}c\text{”}\}$ into the string ” $a : b : c$ ”. Think carefully about what it should do if given a vector of length 0 or 1.

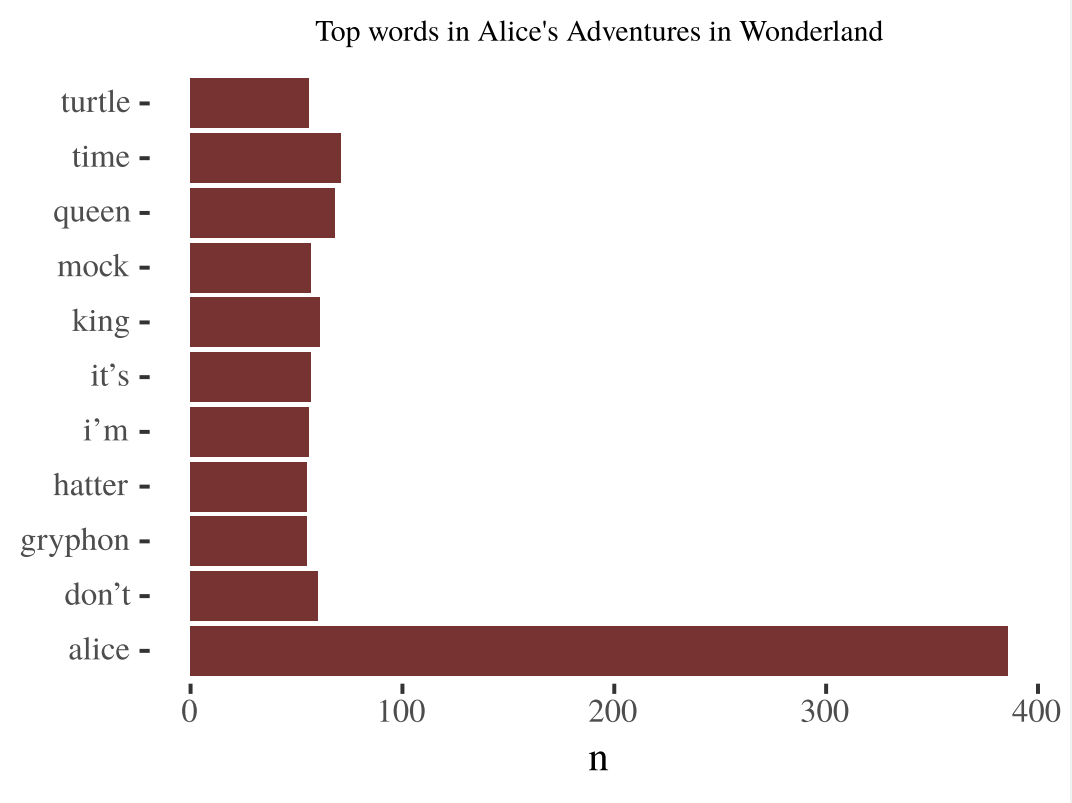
Try the function with these inputs:

```
x0 <- c()
x1 <- c("a")
x2 <- c("a","b")
x3 <- c("a","b","c")
```

04:00

Visualizing Word Counts in Alice's Adventures in Wonderland

```
alice %>%
  mutate(linenumber=row_number()) %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words) %>%
  count(word, sort=TRUE) %>%
  top_n(10) %>%
  ggplot(aes(word, n)) + theme_tufte()
  geom_col(fill = "#773232") +
  xlab(NULL) +
  coord_flip() +
  ggtitle("Top words in Alice's Adventu
  theme(plot.title = element_text(hjus
```



Let function do it!

```
# Function to take the book id of books from project Gutenberg
# Return a data tibble with smart words and their counts, sorted

word_count <- function(book_id) {
    my_favorite_book <- gutenberg_download(book_id) %>%
    mutate(linenumber=row_number()) %>%
    unnest_tokens(word, text) %>%
    anti_join(stop_words) %>%
    count(word, sort=TRUE)
my_favorite_book
}
```

Let function do it!

```
# Function to take the book id of books from project Gutenberg
# Return a data tibble with smart words and their counts, sorted

word_count <- function(book_id) {
    my_favorite_book <- gutenberg_download(book_id) %>%
    mutate(linenumber=row_number()) %>%
    unnest_tokens(word, text) %>%
    anti_join(stop_words) %>%
    count(word, sort=TRUE)
my_favorite_book
}
```

Let function do it!

```
# Function to take the book id of books from project Gutenberg
# Return a data tibble with smart words and their counts, sorted

word_count <- function(book_id) {
    my_favorite_book <- gutenberg_download(book_id) %>%
    mutate(linenumber=row_number()) %>%
    unnest_tokens(word, text) %>%
    anti_join(stop_words) %>%
    count(word, sort=TRUE)
my_favorite_book
}
```

Let function do it!

```
# Function to take the book id of books from project Gutenberg
# Return a data tibble with smart words and their counts, sorted

word_count <- function(book_id) {
    my_favorite_book <- gutenberg_download(book_id) %>%
    mutate(linenumber=row_number()) %>%
    unnest_tokens(word, text) %>%
    anti_join(stop_words) %>%
    count(word, sort=TRUE)
my_favorite_book
}
```

Let function do it!

```
# Function to take the book id of books from project Gutenberg
# Return a data tibble with smart words and their counts, sorted

word_count <- function(book_id) {
    my_favorite_book <- gutenberg_download(book_id) %>%
    mutate(linenumber=row_number()) %>%
    unnest_tokens(word, text) %>%
    anti_join(stop_words) %>%
    count(word, sort=TRUE)
my_favorite_book
}
```

Let function do it!

```
# Function to take the book id of books from project Gutenberg
# Return a data tibble with smart words and their counts, sorted

word_count <- function(book_id) {
    my_favorite_book <- gutenberg_download(book_id) %>%
    mutate(linenumber=row_number()) %>%
    unnest_tokens(word, text) %>%
    anti_join(stop_words) %>%
    count(word, sort=TRUE)
my_favorite_book
}
```

Let function do it!

```
# Function to take the book id of books from project Gutenberg
# Return a data tibble with smart words and their counts, sorted

word_count <- function(book_id) {
    my_favorite_book <- gutenberg_download(book_id) %>%
    mutate(linenumber=row_number()) %>%
    unnest_tokens(word, text) %>%
    anti_join(stop_words) %>%
    count(word, sort=TRUE)
my_favorite_book
}
```

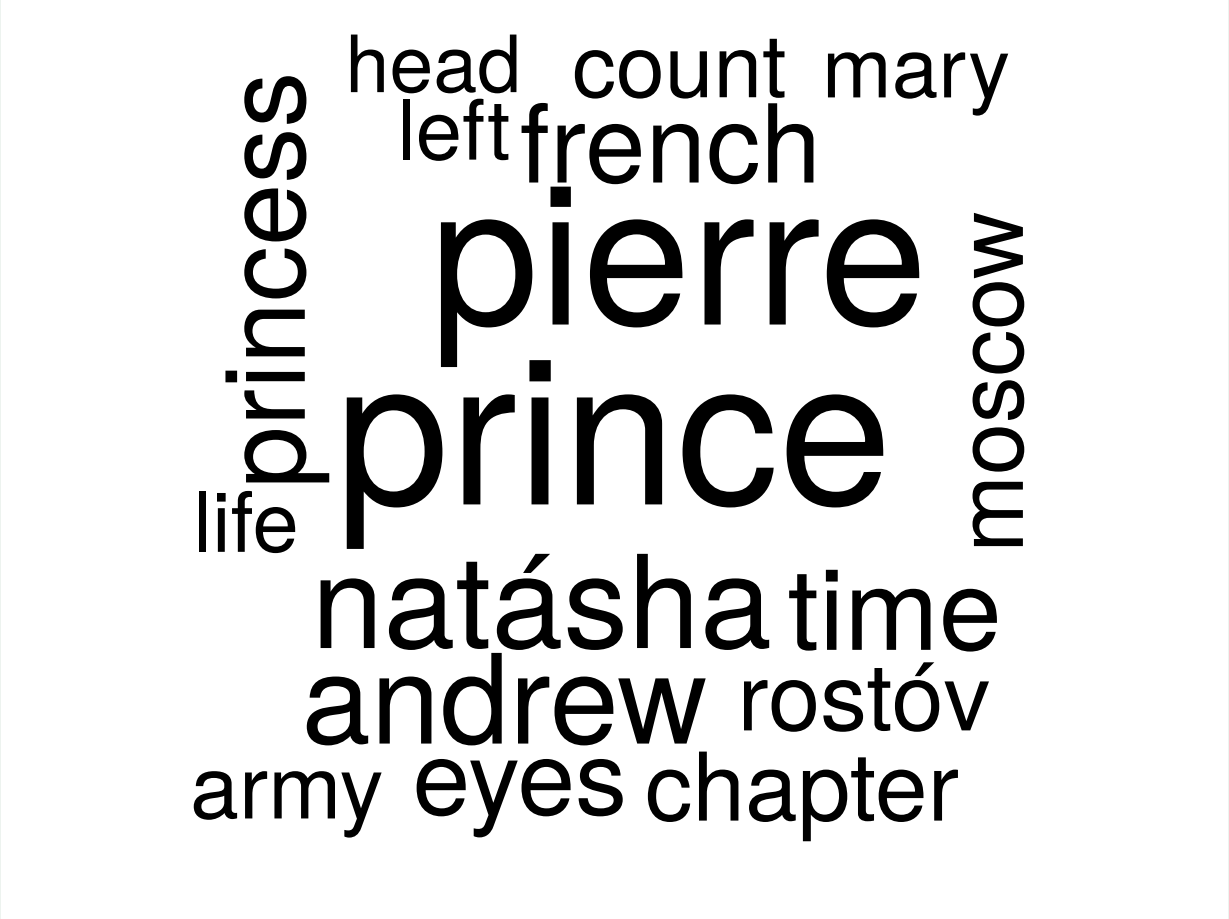
My favorite book!

```
gutenberg_metadata %>%
  filter(title == "War and Peace")
# A tibble: 1 × 8
  gutenberg_id title    author gutenberg_autho… language gutenberg_books… rights
         <int> <chr>    <chr>              <int> <chr>    <chr>            <chr>
1         2600 War and… Tolst…               136 en       Napoleonic(Book… Publi…
# … with 1 more variable: has_text <lgl>
```

```
(mybook <- word_count(book_id= 2600))
# A tibble: 17,479 × 2
   word          n
   <chr>     <int>
 1 prince     1891
 2 pierre     1796
 3 natásha    1098
 4 andrew     1047
```

Word Cloud

```
wordcloud(words= mybook$word,freq =  mybook$n, max.words = 20, random.order = FALSE)
```



Words in War and Peace

Your Turn 3

a. Go to Project Gutenberg. Search the book ID of your favorite book and find the most used word in this book.

b. Make a function that takes the resulting data tibble and the maximum number of words as arguments to plot a word cloud.

c. Modify the `word_count()` function so that it outputs words starting with a particular alphabet.

05:00