

# **Functions**

**Spring 2023**

April 26 2023

# 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  
sqrt  
function (x) .Primitive("sqrt")
```

```
# using the sqrt function  
sqrt(4)  
[1] 2
```

## 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 or tibble**
- **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) # arg1, arg2, etc.
{ # 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) ## arg1, arg2, etc.
{ # 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) # arg1, arg2, etc.
{ # 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) # arg1, arg2, etc.
{ # 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) # arg1, arg2, etc.
{ # 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) # arg1, arg2, etc.
{ # 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) # arg1, arg2, etc.  
{ # 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

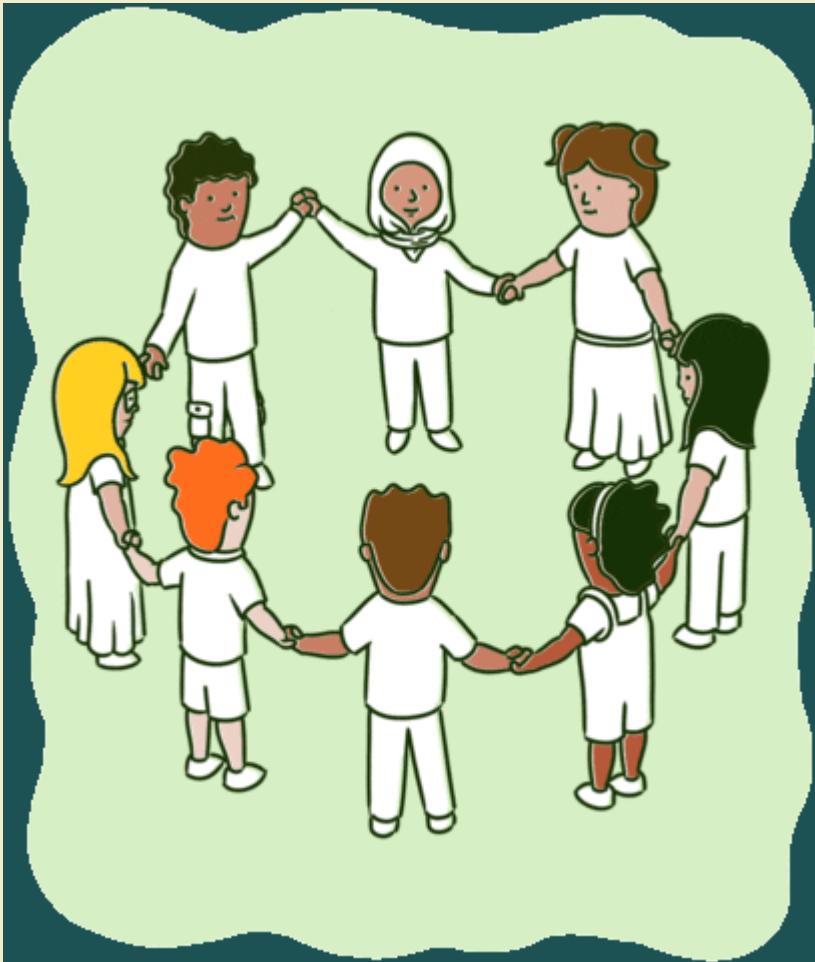
```
# Basic Set Up  
my_awesome_function <- function(x,y) # arg1, arg2, etc.  
{ # Brackets that house the code  
  # Some code to execute  
  z = x*y  
  return(z) # Return a data value  
} # Close the Brackets
```

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

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

10:00

# ✍ GROUP ACTIVITY 1

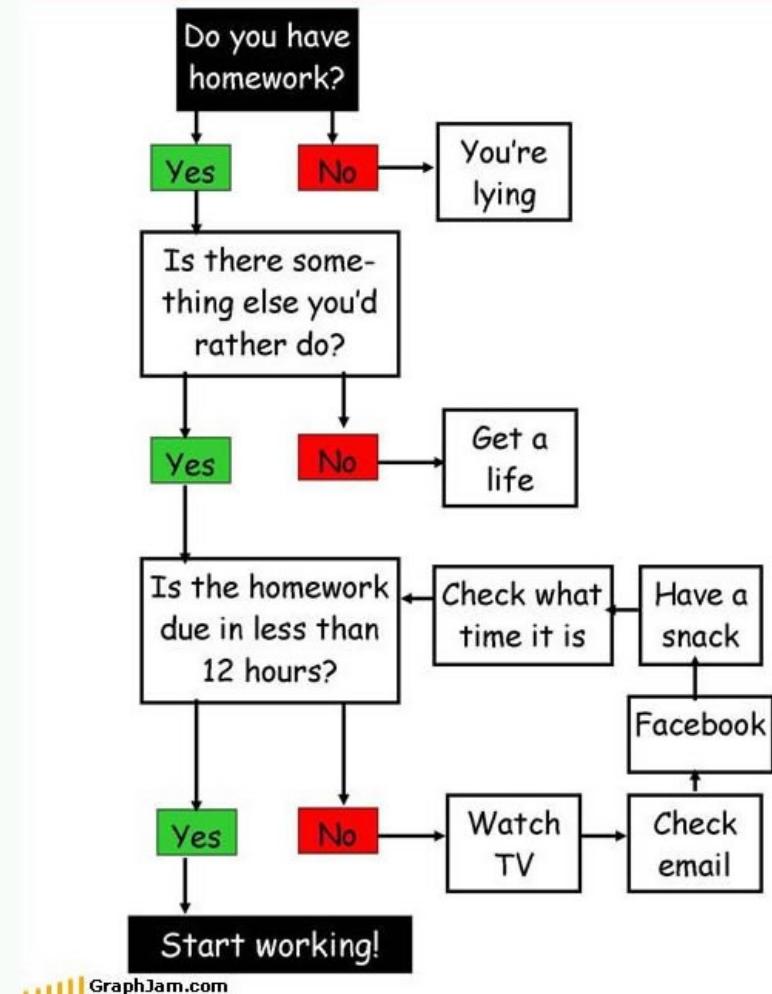


- *Let's go over to maize server/local Rstudio and our class moodle*
- *Get the class activity 14.Rmd file*
- *Work on activity 1*

# 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(TRUE, "Positive", "Negative")
[1] "Positive"
```

```
dplyr:::if_else(FALSE, "Positive", "Negative")
[1] "Negative"
```

```
x <- 1:5
ifelse(x<3, "Positive", "Negative")
[1] "Positive" "Positive" "Negative" "Negative" "Negative"
```

## if and ifelse

```
x <- c(3, 4, 6, -1)
y <- c("5", "c", "9", 1)
```

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

```
# override the default cutoff of 0
cutoff_make0(x, cutoff = 4)
[1] 0 4 6 0
```

```
# no cutoff given, defaults to 0
cutoff_make0(x)
[1] 3 4 6 0
```

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

Let's talk about word tokenization, word clouds,  
and sentiment analysis using *tidytext* principles  
!!

## Tidy Text

- *tidy data principles*
- *works with existing data manipulation tools*
- *streamlined integration with other text mining libraries*



# Tokenization

```
text_data %>%  
  unnest_tokens(output = word,  
                input = text,  
                token = "words") %>%  
kable()
```

word
this
is
seriously
well
put
together
honestly
i
don't
even
know
how

# Counting words

```
text_data %>%  
  unnest_tokens(word, text) %>%  
  count(word, sort = TRUE) %>% kable()
```

word	n
big	3
this	3
and	2
i	2
in	2
is	2

# Stopwords

- `tidytext` comes with a database of common stop words
- carry little to no unique information, and need to be removed

```
stop_words %>% sample_n(10)
# A tibble: 10 × 2
  word    lexicon
  <chr>   <chr>
1 although SMART
2 very    snowball
3 regarding SMART
4 must    onix
5 looks   SMART
6 most    SMART
7 looking SMART
8 above   onix
9 someone SMART
10 latterly SMART
```

What is the average sentiment of Amazon shoppers purchasing musical instruments?



SQUIZZL MUSIC 66 66 66 66 66

# Sentiments in Amazon Musical Instruments Reviews

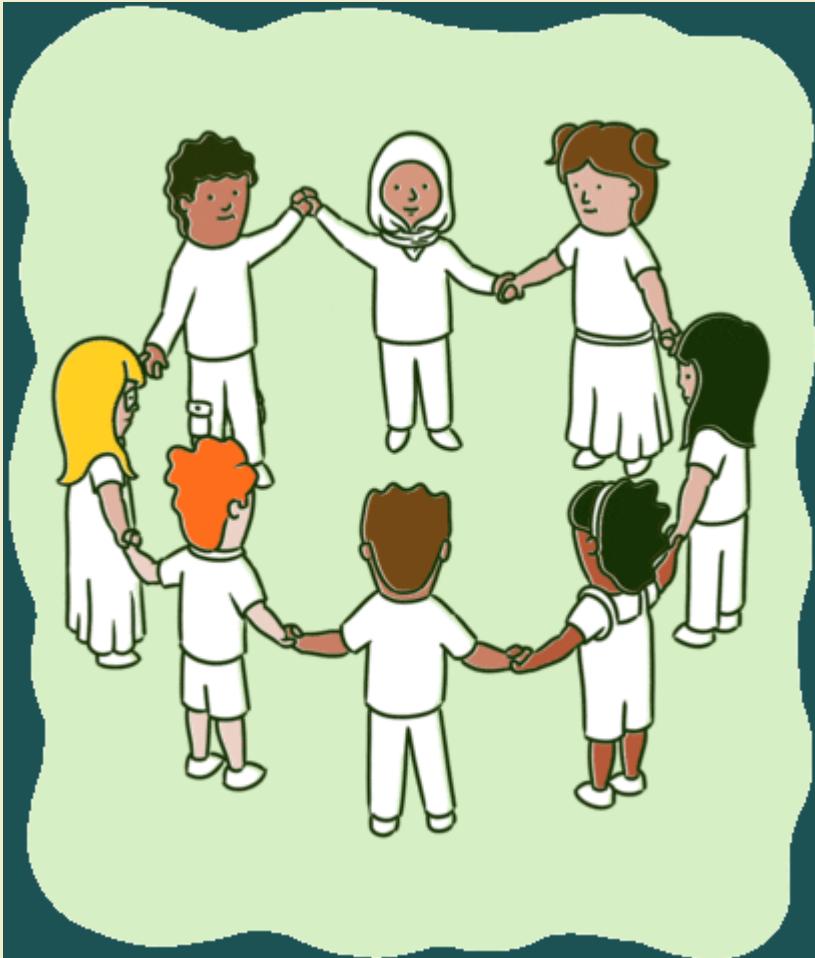
```
library(wordcloud)
library(reshape2) # for acast function
set.seed(123) # for reproducibility

musical_instr_reviews %>%
  select(reviewText) %>%
  unnest_tokens(output=word,
                input=reviewText) %>%
  anti_join(stop_words) %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment,
        value.var = "n",
        fill = 0) %>%
  comparison.cloud(colors = c("blue","purple"),
                   scale = c(2,0.5),
                   max.words = 100,
                   title.size = 2)
```



10:00

# GROUP ACTIVITY 2



- Work on activity 2
- Ask me questions