



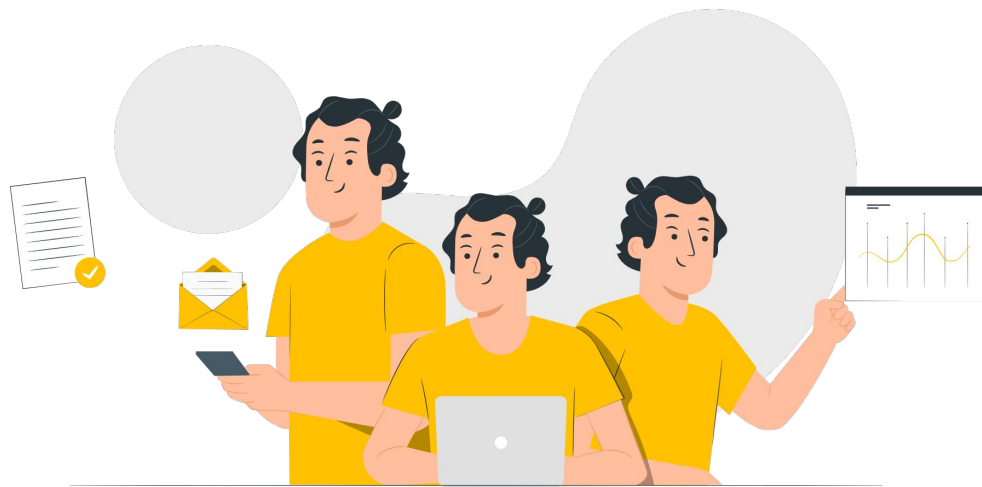
# Programming Foundation

LIVE 04 - Data Science Bootcamp

# Programming foundation

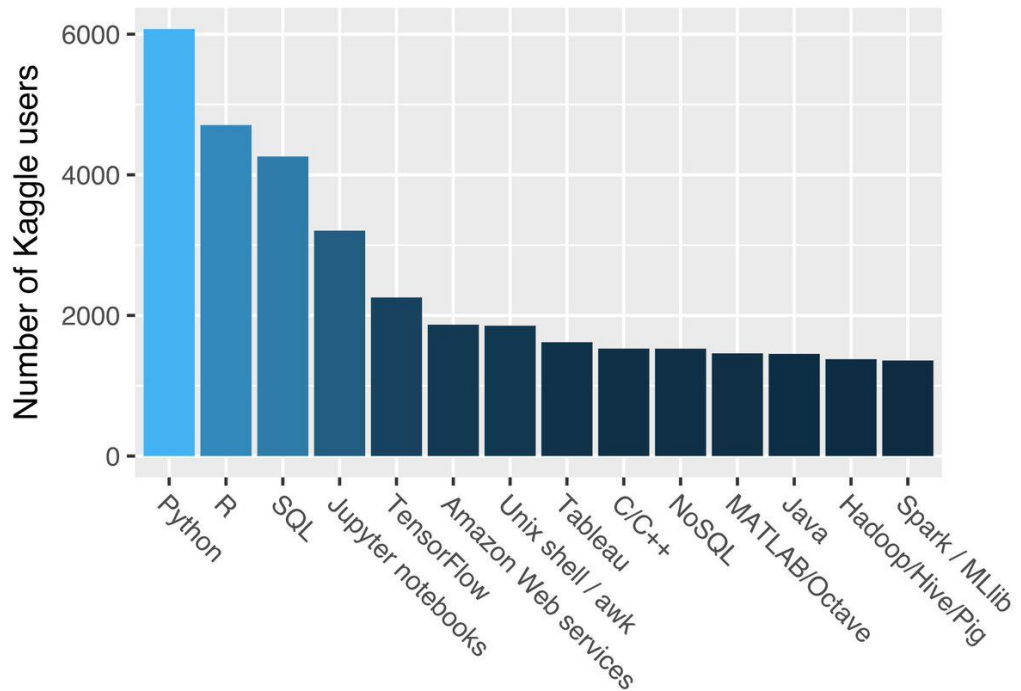
1. Variables
2. Data types
3. Data structures
4. Control flow
5. Function

Language of choice: R



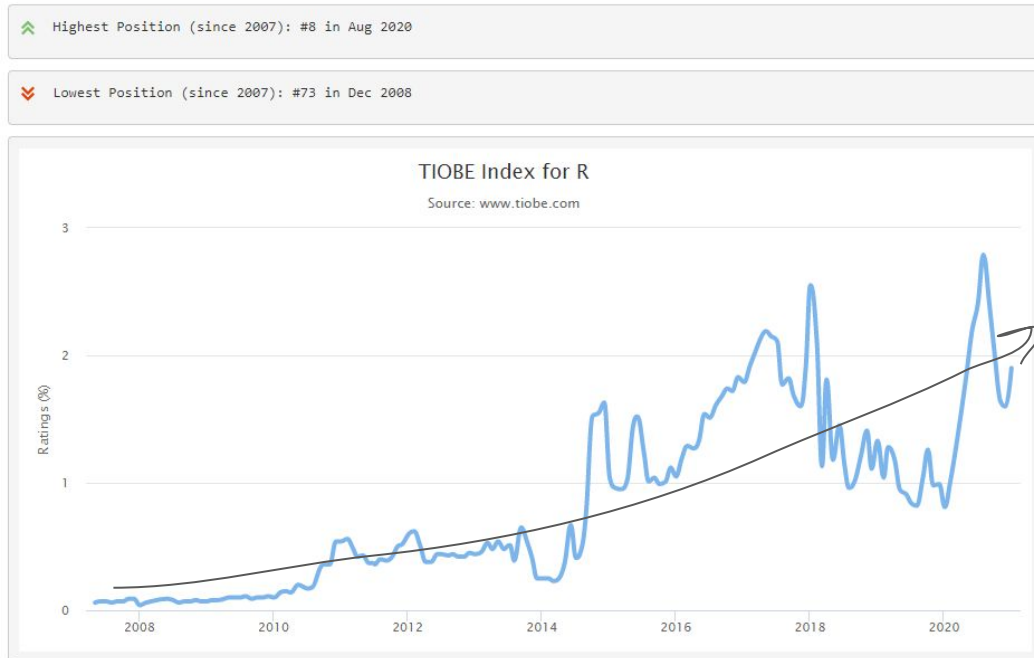


## Data Science Tools/ Languages





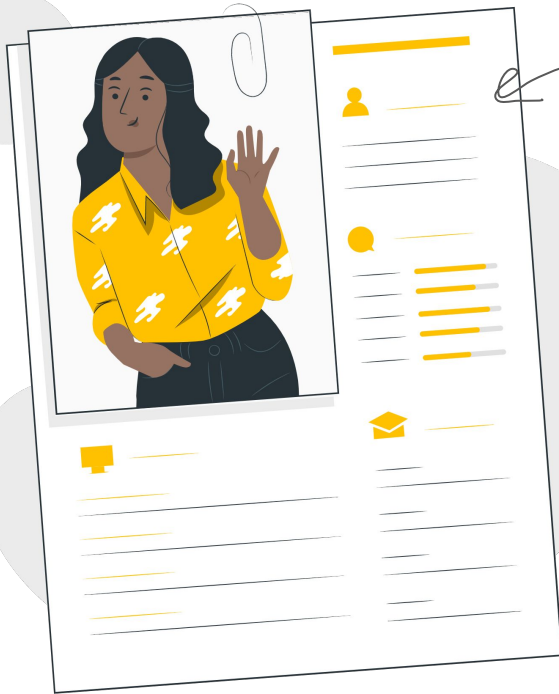
# The importance of R in data science ecosystem



R is among top three  
language for data science

R

# Will R help you get a new job?



Yes 3000 times!



## Data Analyst

PwC Thailand · Bangkok, Bangkok City,  
Thailand

Posted 1 week ago · 1,165 views

Easy Apply

Save

### Qualifications

- Bachelor's or master's degree in Business Analytics, Computer Science, Engineering, Information Systems or Technologies, Statistics or other related disciplines that possess an analytical component.
- 1-3 years of experience in developing, analysing, and interpreting data analytics solutions using some of the following software:
- Analytical software (e.g. R, SAS)
- Data visualisation tools (e.g. Qlikview, PowerBI,)
- Relational database management tools (e.g. SQL)



# RStudio Desktop IDE

The Comprehensive R Archive Network ([r-project.org](https://r-project.org))

RStudio - RStudio

The screenshot displays the RStudio Desktop IDE interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. The toolbar below the menu bar contains icons for file operations, running code, and other functions. The main editor window shows an R script with the following code:

```
9 myList
10 sapply(myList, summary)
11
12 die <- 1:6
13 die %>% die
14 die %>% die
15
16 1:5 %>% 11:15
17 sum(1:5 * 11:15)
18
19 ## P(A|B) = P(B|A) * P(A) / P(B)
20 ## B = positive result (from a doctor)
21 ## A = covid disease
22
23 p.ba = .999
24 p.a = 100000/6700000
25 p.b = p.ba*p.a + .02*(1-p.a)
26
27 p.ab = p.ba * p.a / p.b
28 print(p.ab)
29
30 read.csv("https://gist.githubusercontent.com/seankross/a412dfhd88b3db70b74b/raw/5f23f993cd87c283ce766e7ac6b329ee7cc2e1d1/mtcars.csv")
31
```

The console window on the right shows the output of the script:

```
> state.name
[1] "Alabama" "Alaska" "Arizona" "Arkansas"
[5] "California" "Colorado" "Connecticut" "Delaware"
[9] "Florida" "Georgia" "Hawaii" "Idaho"
[13] "Illinois" "Indiana" "Iowa" "Kansas"
[17] "Kentucky" "Louisiana" "Maine" "Maryland"
[21] "Massachusetts" "Michigan" "Minnesota" "Mississippi"
[25] "Missouri" "Montana" "Nebraska" "Nevada"
[29] "New Hampshire" "New Jersey" "New Mexico" "New York"
[33] "North Carolina" "North Dakota" "Ohio" "Oklahoma"
[37] "Oregon" "Pennsylvania" "Rhode Island" "South Carolina"
[41] "South Dakota" "Tennessee" "Texas" "Utah"
[45] "Vermont" "Virginia" "Washington" "West Virginia"
[49] "Wisconsin" "Wyoming"

> grep("A", state.name)
[1] 1 2 3 4

> grep("A", state.name, value = T)
[1] "Alabama" "Alaska" "Arizona" "Arkansas"
> |
```

The Environment pane at the bottom left shows the following objects:

- Data
  - kmean\_result: List of 9
  - m: 32 obs. of 2 variables
  - mtcars: 32 obs. of 9 variables
  - myList: List of 2
- Values
  - die: int [1:6] 1 2 3 4 5 6
  - i: 10L
  - p.a: 0.0149253731343284
  - p.ab: 0.430789133247089
  - p.b: 0.0346119402985075
  - p.ba: 0.999
  - play: 10
  - prize: num [1:10] 0 0 0 0 0 150 150 0 0 0

The Packages pane at the bottom right shows a list of installed and available packages, including abind, AeneasHousing, ade4, assertthat, backports, base64enc, BH, bit, bit64, blob, brio, broom, C50, and cclr.



R

## Remember these FACTS

1. R is case sensitive
2. R keeps data in computer memory
3. R is REPL (read, eval, print, loop)
4. Everything that exists in R is an **object**
5. Everything that happens in R is a **function** call

Computer Memory






## Set up working directory

```
# get working directory  
getwd()
```

```
# set working directory  
setwd("C:/Users/Hello/Desktop/")
```



Function



## Basic calculation

```
# simple calculator
```

```
1 + 1
```

```
5 - 2
```

```
5 * 3
```

```
6 / 2
```

```
10 %% 2
```

Modulo

A hand-drawn arrow pointing from the word 'Modulo' to the double percent sign operator (%%) in the line '10 %% 2'.

```
5 ** 2
```

power


A hand-drawn arrow pointing from the word 'power' to the double asterisk operator (\*\*) in the line '5 \*\* 2'.



## Create variables

```
# create new variables  
income <- 6000  
expense <- 3200  
saving <- income - expense
```

Assign operator

A hand-drawn arrow pointing from the text 'Assign operator' to the '<-' operator in the first line of code.

```
# remove variables  
rm(saving)
```

Remove variable

A hand-drawn arrow pointing from the text 'Remove variable' to the 'rm' function in the second line of code.



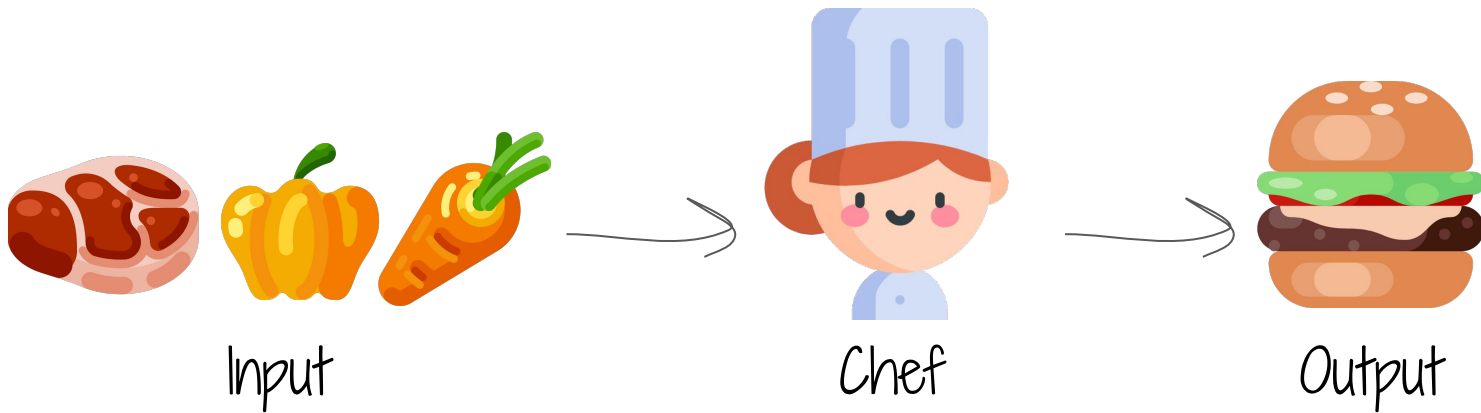
## Print characters (string)

```
# print hello world  
print("hello world")  
print('hello world')
```

We can use double or single  
quote (double is preferred)

```
# there are other functions too  
cat("R is awesome!")
```

## How function works

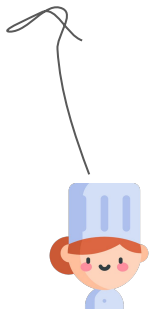


R

## How function works (R is very similar to Google Sheets)



**function( input1, input2 )**

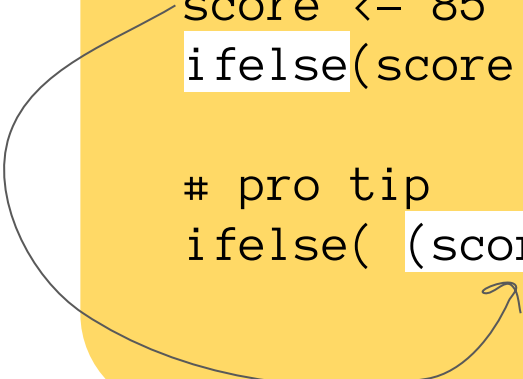


## R

## Create a condition in R

```
# ifelse function in R
score <- 85
ifelse(score >= 80, "Passed", "Failed")

# pro tip
ifelse((score <- 85) >= 80, "Passed", "Failed")
```



Assign variable on the fly



## Data types

What is the class (type) of this variable?

```
# numeric
```

```
class(100)
```

```
# character
```

```
class("Hello World")
```

```
# logical
```

```
class(TRUE) # FALSE
```





## Data types - Factor

```
# create an animal factor  
animal <- c("cat", "dog", "hippo", "cat", "dog")
```

```
# convert animal to factor  
animal <- as.factor(animal)  
class(animal)
```



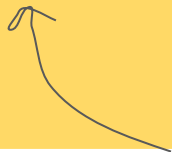
Change from character to factor



## Change data types

```
# is to check class/ type  
is.character(100)
```

```
# as to convert class/ type  
x <- 100  
x <- as.character(x)
```



Change from numeric to character



## TRUE is 1 and FALSE is 0

```
# relationship between numeric and logical
```

```
1 == TRUE
```

```
0 == FALSE
```

```
# we can summarize logical vector
```

```
sum( c(T, T, F, F, T) ) ## sum( c(1,1,0,0,1) )
```

```
mean( c(T, T, F, F, T) )
```



## Compare values in R

```
# compare using these operators
```

```
1 + 1 == 2
```

```
5 * 2 != 20
```

```
5 > 2
```

```
5 >= 2
```

```
5 < 10
```

```
5 <= 10
```

```
"Hello" == "hello"
```

# What is data structures?

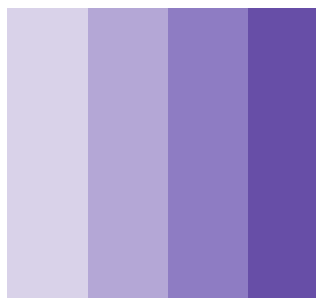
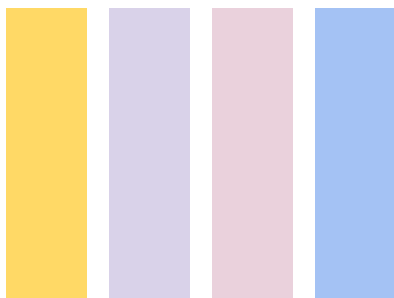


Vector



Matrix

List



Data Frame



## Data structures - vector

concat values

```
# create simple vector with c()  
numbers <- c(100, 120, 250, 300, 550)  
friends <- c("David", "John", "Anna")
```

```
# rep() to replicate  
rep("Hello", 10)
```


```
# seq() to generate sequence  
seq(from = 1, to = 100, by = 2)
```



## Data structures - matrix

```
# create matrix from vector  
numbers <- c(100, 120, 150, 200, 290, 300)  
dim(numbers) <- c(2, 3)
```

```
# create using matrix()  
m1 <- matrix(1:6, ncol=2, byrow=TRUE)
```



Easy to use : )



## Data structures - list

List can keep a lot of data, different types

```
# create a customer using list
customer_01 <- list(
  fname = "David",
  lname = "Beckham",
  age = 42,
  favourite_food = c("Apple", "Tom Yum"),
  avg_spending = 5600,
  currency = "US Dollar"
)
```





## Data structures - data frame

|    | A       | B         | C              | D             | E                 | F           | G      | H    |
|----|---------|-----------|----------------|---------------|-------------------|-------------|--------|------|
| 1  | species | island    | bill_length_mm | bill_depth_mm | flipper_length_mm | body_mass_g | sex    | year |
| 2  | Adelie  | Torgersen | 39.1           | 18.7          | 181               | 3750        | male   | 2007 |
| 3  | Adelie  | Torgersen | 39.5           | 17.4          | 186               | 3800        | female | 2007 |
| 4  | Adelie  | Torgersen | 40.3           | 18            | 195               | 3250        | female | 2007 |
| 5  | Adelie  | Torgersen | NA             | NA            | NA                | NA          | NA     | 2007 |
| 6  | Adelie  | Torgersen | 36.7           | 19.3          | 193               | 3450        | female | 2007 |
| 7  | Adelie  | Torgersen | 39.3           | 20.6          | 190               | 3650        | male   | 2007 |
| 8  | Adelie  | Torgersen | 38.9           | 17.8          | 181               | 3625        | female | 2007 |
| 9  | Adelie  | Torgersen | 39.2           | 19.6          | 195               | 4675        | male   | 2007 |
| 10 | Adelie  | Torgersen | 34.1           | 18.1          | 193               | 3475        | NA     | 2007 |
| 11 | Adelie  | Torgersen | 42             | 20.2          | 190               | 4250        | NA     | 2007 |
| 12 | Adelie  | Torgersen | 37.8           | 17.1          | 186               | 3300        | NA     | 2007 |
| 13 | Adelie  | Torgersen | 37.8           | 17.3          | 180               | 3700        | NA     | 2007 |
| 14 | Adelie  | Torgersen | 41.1           | 17.6          | 182               | 3200        | female | 2007 |
| 15 | Adelie  | Torgersen | 38.6           | 21.2          | 191               | 3800        | male   | 2007 |
| 16 | Adelie  | Torgersen | 34.6           | 21.1          | 198               | 4400        | male   | 2007 |
| 17 | Adelie  | Torgersen | 36.6           | 17.8          | 185               | 3700        | female | 2007 |
| 18 | Adelie  | Torgersen | 38.7           | 19            | 195               | 3450        | female | 2007 |
| 19 | Adelie  | Torgersen | 42.5           | 20.7          | 197               | 4500        | male   | 2007 |
| 20 | Adelie  | Torgersen | 34.4           | 18.4          | 184               | 3325        | female | 2007 |
| 21 | Adelie  | Torgersen | 46             | 21.5          | 194               | 4200        | male   | 2007 |
| 22 | Adelie  | Biscoe    | 37.8           | 18.3          | 174               | 3400        | female | 2007 |
| 23 | Adelie  | Biscoe    | 37.7           | 18.7          | 180               | 3600        | male   | 2007 |
| 24 | Adelie  | Biscoe    | 35.9           | 19.2          | 189               | 3800        | female | 2007 |



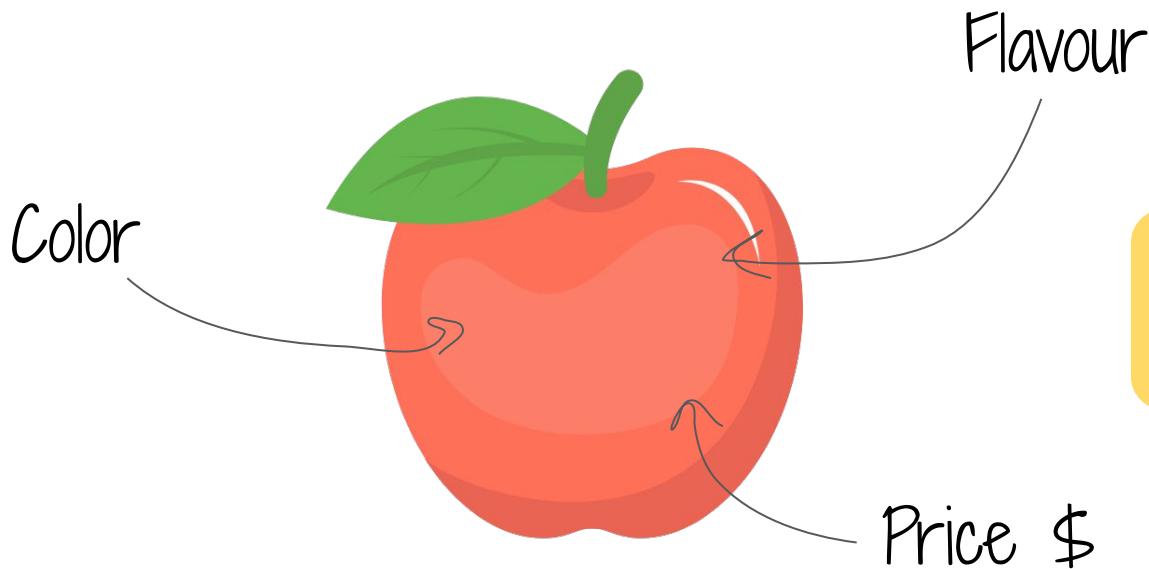
## Data structures - data frame

```
# create vectors with same length
id <- 1:5
friends <- c("David", "John", "Anna", "Barry", "Garth")
ages <- c(30, 25, 22, 32, 29)
gender <- factor(c("M", "M", "F", "M", "M"))
movie_lover <- c(TRUE, TRUE, TRUE, FALSE, FALSE)

# create a new dataframe
data.frame(id, friends, ages, gender, movie_lover)
```

R

## Everything that exists in R is an object



```
apple$color  
apple$flavour  
apple$price
```

## R Get data from a data frame using \$ [] or [[]]

```
# get a column you want
```

```
mtcars$mpg
```

```
mtcars["mpg"]
```

```
mtcars[["mpg"]]
```

different operator return  
different data structure



```
# do some stats
```

```
mpg <- mtcars$mpg
```

```
mean(mpg); median(mpg); sum(mpg)
```

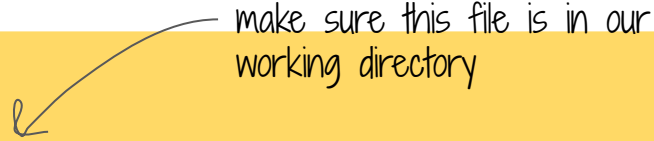
R is full of statistics functions





## Read and write CSV data from RStudio

```
# read csv file  
read.csv("penguins.csv")  
read.csv("url to csv file on the internet")
```

A handwritten note in black ink says "make sure this file is in our working directory". A curved arrow points from this note to the "penguins.csv" argument in the first `read.csv()` function call.

make sure this file is in our  
working directory

```
# write csv file  
write.csv(penguins, "penguins_2.csv",  
row.names=FALSE)
```



## Subset using []

```
# create a vector  
friends = c(David = 30, John = 28, Marry = 25)
```

```
# subset by position  
friends[1]
```

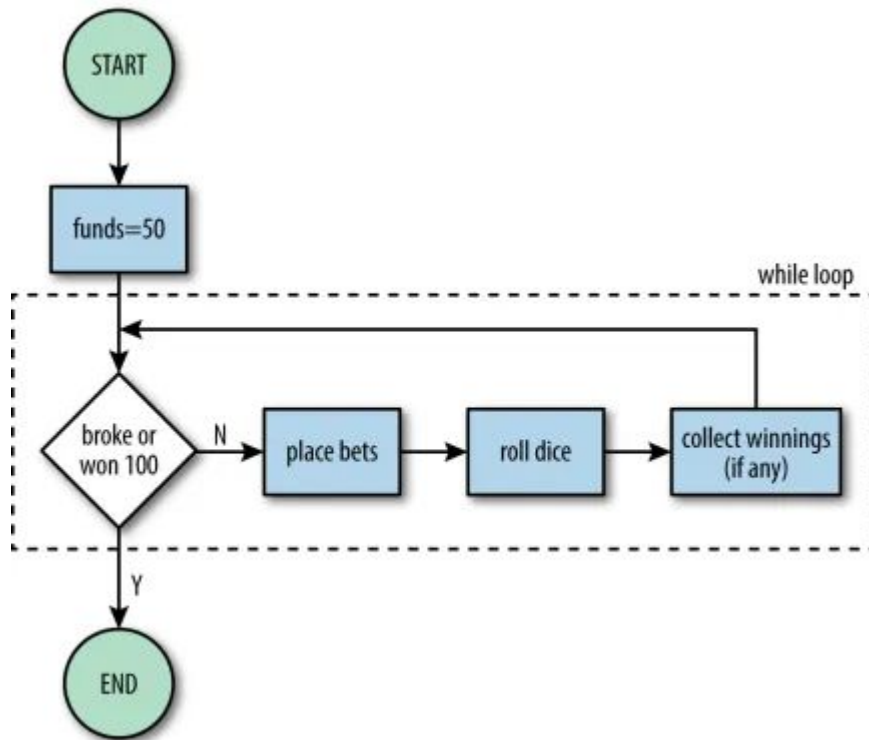
```
# subset by condition  
friends[ friends < 30 ]
```

We use [ ] to subset  
information

```
# subset by name  
friends[ "David" ]
```

R

## What is control flow?




We can write code to control our program behaviours

If  
For  
While

## Control flow - if

```
# if else block  
score <- 85  
  
if (score >= 80) {  
  print("Passed")  
} else {  
  print("Failed")  
}
```

condition

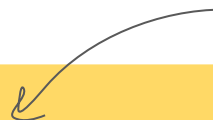







## Control flow - for

We know how many  
times the loop will run




```
# for loop
friends <- c("David", "John", "Mary")

for (friend in friends) {
  print(paste("Hello!", friend))
}
```



## Control flow - while

```
# while loop  
  
while (TRUE) {  
  do something  
}
```



We (might not) know how many  
times the loop will run



## Control flow - while

Counter variable

```
# while loop
count <- 0
while (count < 5) {
  print("Hello World")
  count <- count + 1
}
```

While loop condition

Update counter  
variable



## Get text input from a user

*readline() gets character input*

```
# get input from user
user_name <- readline("What is your name?: ")
cat("Hello", user_name)
```

```
# convert input to numeric if needed
age <- readline("How old are you?: ")
age <- as.numeric(age)
age * 2
```

R

## Create your own functions


```
# create a simple function  
add_two_nums <- function(a, b) {  
  a + b  
}
```

function keyword



```
# power function  
my_power <- function(base, power=2) {  
  base ** power  
}
```

default argument



R

## Project 01 - Rock Scissors Paper



**ROCK**



**SCISSORS**



**PAPER**

We will write this game

Rock Scissors Paper

Play against computer (random)

Keep track of wins, losses, and ties



# We learn through building hands-on projects

```
# Rock Paper Scissors
# Rock = 1, Paper = 2, Scissors = 3, Exit = 4

actions <- c("Rock", "Paper", "Scissors", "Exit")
win <- 0
loss <- 0
tie <- 0

while (TRUE) {
  player_move <- as.numeric(readline("Choose your move: Rock[1], Paper[2], Scissors[3], Exit[4]: "))
  if (player_move == 4) {
    cat("Good Bye!")
    break
  }

  player_move <- actions[player_move]
  computer_move <- actions[sample(1:3, 1)]

  if (player_move == computer_move) {
    tie <- tie + 1
  } else if (player_move == "Rock" & computer_move == "Paper") {
    loss <- loss + 1
  } else if (player_move == "Paper" & computer_move == "Scissors") {
    loss <- loss + 1
  } else if (player_move == "Scissors" & computer_move == "Rock") {
    loss <- loss + 1
  } else {
    win <- win + 1
  }

  cat("Player Move:", player_move, "\n")
  cat("Computer Move:", computer_move, "\n")
  cat(win, loss, tie)
}
```



ROCK



SCISSORS



PAPER



# Refactoring Your Code

```
# Rock Paper Scissors
# Rock = 1, Paper = 2, Scissors = 3, Exit = 4

actions <- c("Rock", "Paper", "Scissors", "Exit")
win <- 0
loss <- 0
tie <- 0
```

```
while (TRUE) {
  player
  if (pl
  cat(
  brea
  }
  player
  comput
  if (pl
  tie
  } else
  loss
  } else
  loss
  } else
  loss <- loss + 1
  } else {
  win <- win + 1
  }
  cat("Player Move:", player_move, "\n")
  cat("Computer Move:", computer_move, "\n")
  cat(win, loss, tie)
}
```

Make your code more Readable

1. Write code first version
2. Improve your code



ROCK



SCISSORS



PAPER



R

**You can search for pattern with Regular Expression**

Google Sheets is the best! You can use it for free, 0\$ cost.

R

Find G\_\_S\_\_

$G[a-z]^+ S[a-z]^+$




**Google Sheets** is the best! You can use it for free, 0\$ cost.

R

## Find 0-9 number

Google Sheets is the best! You can  
use it for free, 0\$ cost.

[0-9]





## Regular Expression Basics

**^A**      Ant, Amsterdam, America

**s\$**      Toys, SNSDs, APPLEs

**c.t**      cat, cot, cet, cCt, c8t



## Regular Expression Character Class

|                    |                             |
|--------------------|-----------------------------|
| <code>[ABC]</code> | match A B or C              |
| <code>[A-Z]</code> | match all capital letters   |
| <code>[A-z]</code> | match all letters           |
| <code>[a-z]</code> | match all lowercase letters |
| <code>[0-9]</code> | match digits                |



## Regular Expression Quantifiers

|       |                               |
|-------|-------------------------------|
| *     | match zero or more            |
| +     | match one or more             |
| ?     | match zero or one             |
| {5}   | match exactly 5 characters    |
| {3,5} | match min 3, max 5 characters |

[Regular expression - Wikipedia](#)

R

## More Examples : )

`[0-9]{5}`

apples?

`^[AB][0-9]{4}`

match exactly 5 digits

apple, apples

A1150, B2324, A3599



## Basic regular expression in R

```
# look at state.name  
state.name
```

```
# find matched string  
grep("A", state.name)  
grepl("A", state.name)
```

Find pattern



```
# replace A with Aloha  
gsub("A", "Aloha", state.name)
```

Replace pattern






## R Tip - we can extract information using gsub()

```
# create strings
foods <- c("I like hotdog", "I like hamburger")

# get the liked object
gsub("I like ", "", foods)
```



Replace "I like " with empty string, cool huh!?



# Programming Foundation

LIVE 04 - Data Science Bootcamp