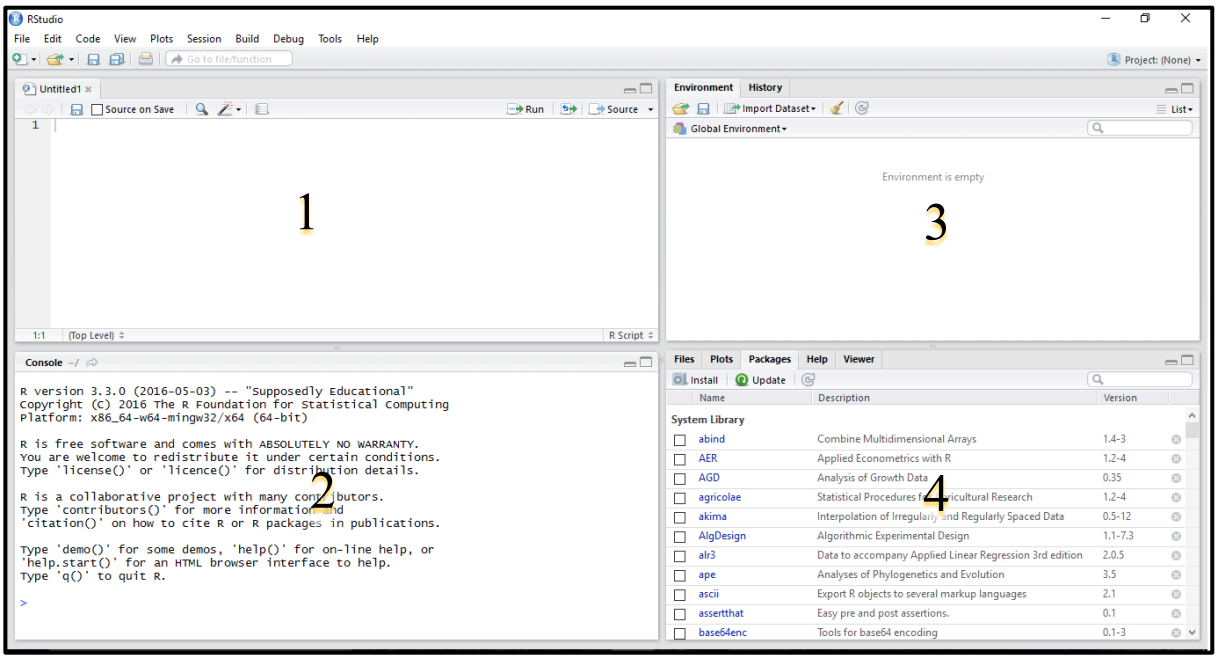
# R – Programming Introduction – Part 1

## Why R

- It is a programming language developed by statisticians for statisticians.
- Free software environment for statistical computing, data analysis and scientific research.
- It is free and comes with an easily installable set of packages to help with areas relating from data access, data cleaning to analysis and reporting.
- Strong community with lot of libraries and in-built functions.
- Powerful visualization support.
- It also has one of the best IDE (R-Studio).

Below is how the R-studio interface will look like




The R studio has 4 windows

1. Source/Scripting window (where you do your entire development)
2. Console window (where R studio runs your scripts and shows the outputs)
3. Environment/History window (where the data created as a part of your script is saved)
4. Help window (which provides documentation and many other functionalities)

## Getting Started with R-studio

- Go to Source window and type 3 + 4.
- To execute the code, press Ctrl + Enter (or Command + Return for Mac).
- Congrats!! You ran you first line of code in R-studio.
- To verify the output, look at the console window.

## Agenda of this activity

In this activity, we are going to learn the below aspects of R Programming

1. Basic operations (Arithmetic and Assignment)
2. Data types
3. Relational operations
4. Conditional statements
5. Coercion
6. Vectors
7. Vectors – Deep dive
8. Vectors – Special functions

## 1. Basic Operations (Arithmetic and Assignment)

Mathematical operations that can be used are:

| Operation | Meaning |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| ^ | Exponent |
| %% | Remainder |
| %/% | Integer Division |

Please try the below operations

```
# Addition
22+23

# Subtraction
23-1

# Multiplication
23 * 2

# Division
24/6

# Power
2^2

# Modulo (Returns remainder of integer division)
21%%4
```

To create a variable and assign a value, try the below syntax (do not forget to execute the

```
x = 41
```

command)

Now, look at the environment window. The variable that is created will be stored and will be visible there.
Print out the value of the variable x using below syntax

```
# auto-printing
x
# explicit printing
print(x)
```

---

Activity

1. First Problem

    - Multiply 13 with 9 and then subtract 10 and add 35?
    - Save the above calculation in a variable y
    - Divide this variable by 42 and store it in a new variable x
    - What will be the final output?

2. Second Problem
   - Assign a value to the variables named apples and oranges
   - Add these two variables together
   - Create a variable fruit to store the sum of apples and oranges

## 2. Data types

R works with numerous data types. Some of the most basic types to get started are:

1. Decimals values like 4.5 are called numeric.
2. Natural numbers like 4 are called integers.
3. Boolean values (TRUE or FALSE) are called logical.
4. Text (or string) values are called characters

```
# Integer type
my_numeric <- 42.1
# String
my_character <- "Universe"
# Boolean
my_logical <- FALSE
# Numeric
my_integer <- 40
```

Once the above variables are created, the verify the datatypes of each of the variables, use the in-built command called ‘class’.

```
class(my_numeric)
class(my_character)
class(my_logical)
class(my_integer)
```

Some of the in-built functions for mathematical operations are given below for reference.

| R command | function |
|---|---|
| `abs()` | absolute value, $ |
| `cos(), sin(), tan()` | cosine, sine, tangent |
| `exp(x)` | exponential function |
| `log(x)` | natural (base-e) logarithm |
| `log10(x)` | base-10 logarithm |
| `sqrt(x)` | square root |
| `sum()` | sum of the entries in a vector |
| `diff()` | first differences |
| `cumsum()` | cumulative sum |

Activity :

1. Create a variable 'length' and assign 15 to it. Also create a variable 'width' and assign 6 to it. Now, create a variable 'area' which is the product of length and width variables.
2. Take four variables x1, y1, x2, y2 and assign some numbers to them. Now find the distance between the points (x1, y1) and (x2, y2)

## 3. Relational Operations

Below are the relational operations carried with respective syntax

| Operation | Meaning |
|---|---|
| == | Check for equality |
| != | Check for inequality |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |
| & | And |
| \| | Or |

Try the below operations:

```
A = 100
B = 200
A == B
A != B
A < B
A > B
A >= 100
A <= 111
```

Also, Below are some of the Logical operations that will help in evaluating a group of relational operations together

| Operation | Meaning |
|---|---|
| & | And |
| \| | Or |
| ! | Not |

```
C = TRUE
D = FALSE

# Logical OR operator
C | D
# Logical AND operator
C & D
# Logical NOT operator
!C
```

Activity

1. Assign a number to variable a and another number to variable b. Now compare these 2 variables using 'greater than equal to' operator, and observe the result.
2. (TRUE + TRUE) * FALSE , what does this expression evaluate to and why?

## 4. Conditional statements

IF statement: The 'If' statement's skeleton looks like:

Syntax:

```
if(<condition>){
        <instruction>
        <instruction>
}
```

Example:

```
# if statement
age = 19

if(age > 18){
  print("You are an adult")
}
```

IF- Else : If the condition is not true, then to execute alternate instructions, add them in the 'else' condition. The skeleton is shown below

Syntax:

```
if(<condition>){
        <instruction>
        <instruction>
} else {
        <instruction>
        <instruction>
}
```

Example:

```
age = 15

if(age > 18){
  print("You can vote")
} else {
  print("not allowed!!")
}
```

Nested IF statements: In case you want to validate multiple statements, those conditions can be added in else condition as show in below example.

```
x = 0
if (x < 0) {
  print("Negative number")
} else if (x > 0) {
  print("Positive number")
} else {
  print("Zero")
}
```

Ifelse condition: a fast way to check the 'if' condition and return a value conditionally is shown by below skeleton:

Syntax:
ifelse(<condition>, <True statement>, <False statement>)

example:

```
a = 3
ifelse(a %% 2 == 0, "even", "odd")
```

Activity

1. Assign numeric values to 2 variables a and b, now write an if else statement which performs if a is greater than b, it prints 'Hello World!!' otherwise, 'Bye!!'

## 5. Coercion

For converting the datatypes into respective datatypes, use the below inbuilt functions
as.numeric – to numeric
as.logical – to boolean
as.character – to string

```
x <- 2
x
class(x)
```

```
y = as.numeric(x)
class(y)
y
```

```
y = as.logical(x)
class(y)
y
```

```
c= as.character(x)
c
```

## 6. Vectors

The c() function can be used to create vectors of objects. Below are some of the examples.

```
# numeric
v1 <- c(0.5, 0.6)
class(v1)
```

```
# logical
v2 <- c(TRUE, FALSE)
class(v2)
```

```
# logical
v3 <- c(T, F)
class(v3)
```

```
# character
v4 <- c("a", "b", "c")
class(v4)
```

If you want to create a integer vector, It can be done with the following syntax:

```
# integer sequence
v5 <- 9:29
v5
class(v5)
```

To initialize an empty numeric vector:

```
x <- vector("numeric")
class(x)
```

To append a value to the vector:

```
x = c(x, 4)
print(x)
```

## 7. Vectors – Deep dive

There are a few in-built functions to understand a vector.
For example, ‘str’ command will give the structure of a vector (or any other object)

```
# numeric vector
v1=c(1,2,3,4,5)
# character vector
v2=c("a","b","c","d","e")
# logical vector
v3=c(TRUE,FALSE,TRUE,FALSE,TRUE)
```

```
# to view structure of the vector
str(v1)
str(v2)
str(v3)
```

To get the length of the vector, use the 'length' command.

```
length(v1)
```

Similarly, max and min functions are available.

```
max(v1)

min(v2)
```

**Vector Sub-setting:**

We can refer to a particular element or a set of elements using the indices. In R language the index starts with 1 and the subsequent elements in a vector can be referred using the respective integers.

Example:

```
# create a vector
x <- c(2,4,6,6,8,2)
# refer to the first element
x[1]
# refer to the fifth element
x[5]
# refer to the first 4 elements
x[1:4]

# To extract 2nd and 4th elements
x[c(2,4)]
```

Now, you can also check whether all the elements in a vector follow a particular condition or not.

```
# check if each element in x are greater than 4
x > 4
```

In the above line of code, if you have 5 elements in vector x, the output is a Boolean vector of same size where each element is greater than 4 or not.

To extract only the elements satisfying the condition the sub-setting can be used again as shown below.

```
# extract those values from x that are greater than 4
x[x>4]
```

Activity

1. Consider two vectors: (3, 5, 6, 8) & (3, 3, 3, 4)
    - Add the 2 vectors
    - Subtract the 2 vectors
    - Multiply the 2 vectors
    - Divide the 2 vectors
2. Take the vector (3, 5, 6, 8), and perform the following operation:

- Write a conditional statement, which gives TRUE if the value is greater than 5.5 and FALSE for any other case.
- Now, the output will be a vector with boolean values. extract only TRUE's from this vector based on indices

## 8. Vectors – Special functions

Seq Function:

To create sequence of numbers in a given pattern.

```
# this kind sequence is already covered above
x <- 1:20
x
```

```
# create a sequence from 0 to 100 in steps of 2
myseq1 <- seq(from=0,to=100, by=2)
myseq1
```

```
# create a sequence of 10 numbers from 2 in steps of 0.5
myseq2=seq(length=10, from=2, by=0.5)
myseq2
```

```
# create a sequence from 0 to 100 in steps of 10
myseq3 <- seq(from = 0, to = 100, by = 10)
myseq3
```

'rep' Function:

‘rep’ function is used to repeat a particular vector

```
r = 6
r
# repeat r 5 times
rep(r,5)
```

```
# take the sequence of 1 to 4 and repeat each element twice
rep(1:4, each = 2)
rep(1:4, times=2)
```

```
rep(1:4, times=c(2,1,2,1))

# show first 4 only in the output
rep(1:4, each = 2, len = 4)
```

```
# 8 integers plus two recycled 1's to fit the length
rep(1:4, each = 2, len = 10)
```

```
#when both each and times are given, first ‘each’ is applied
and then ‘times’ is applied
# length 24, 3 complete replications

rep(1:4, each = 2, times = 3)
```