Inteligencia Artificial

R Crash Course

LABSIN

AGENDA:

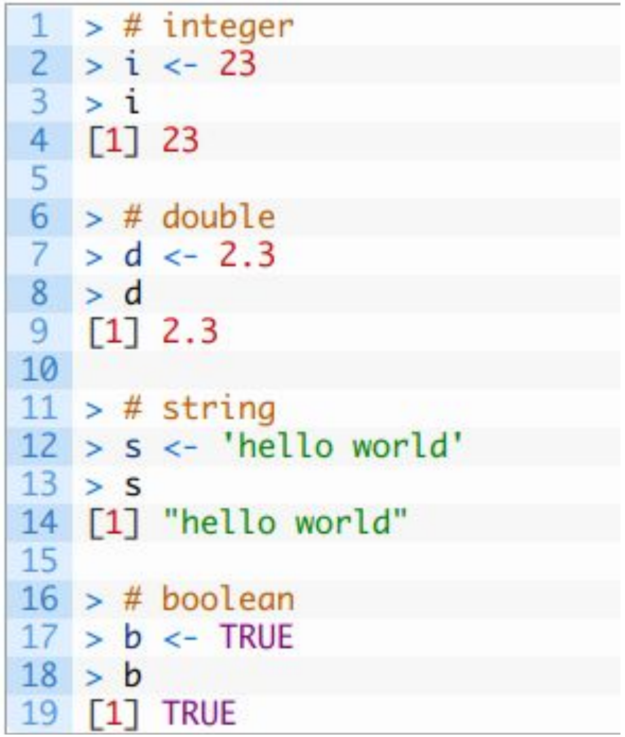
The examples in this section are split into the following sections:

1. Assignment
2. Data Structures
3. Flow Control
4. Functions
5. Packages

Start the R interactive environment (type R on the command line)

and let’s get started.

ASSIGNMENT



```
> # integer
> i <- 23
> i
[1] 23

> # double
> d <- 2.3
> d
[1] 2.3

> # string
> s <- 'hello world'
> s
[1] "hello world"

> # boolean
> b <- TRUE
> b
[1] TRUE
```

The key to assignment in R is the arrow operator (<-) for assignment.

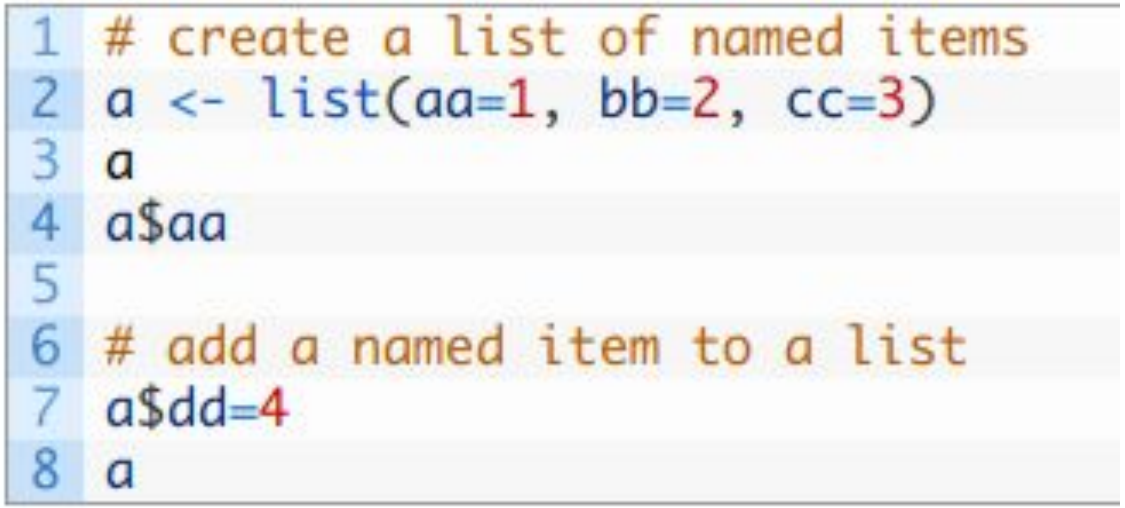
DATA STRUCTURES

There three data structures that you will use the most in R:

1. Vectors
2. Lists
3. Matrices
4. Data Frames

LISTS

Lists provide a group of named items.



```r
# create a list of named items
a <- list(aa=1, bb=2, cc=3)
a
a$aa

# add a named item to a list
a$dd=4
a
```

LISTAS

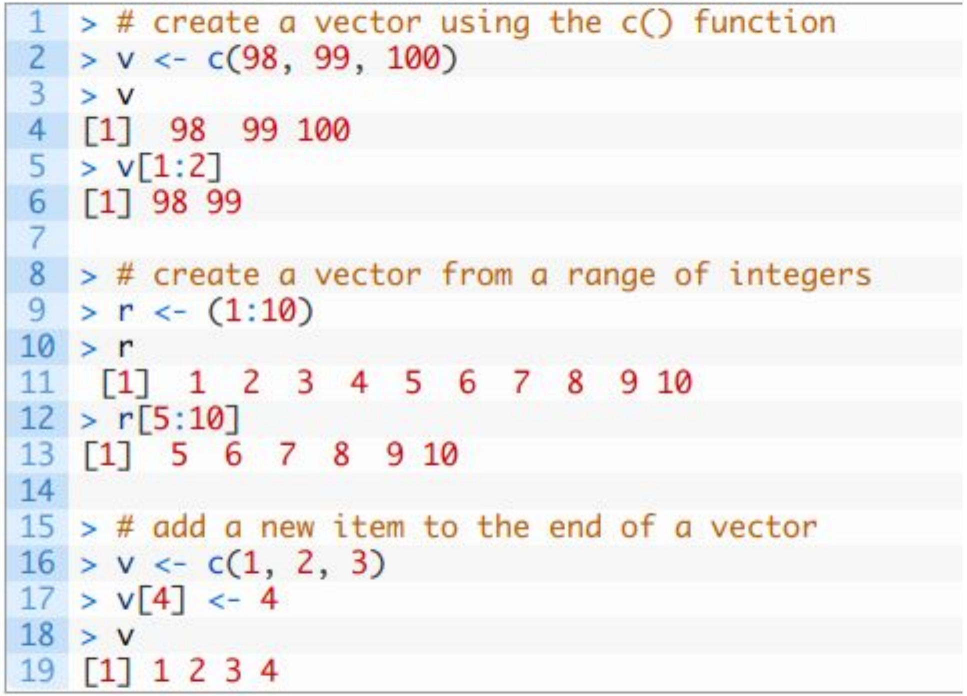
Lists provide a group of named items.

1. You can define a new list with the list() function.
2. A list can be initialized with values or empty.
3. Note that the named values in the list can be accessed using the dollar operator ($).
4. Once referenced, they can be read or written.
5. This is also how new items can be added to the list.

VECTORS

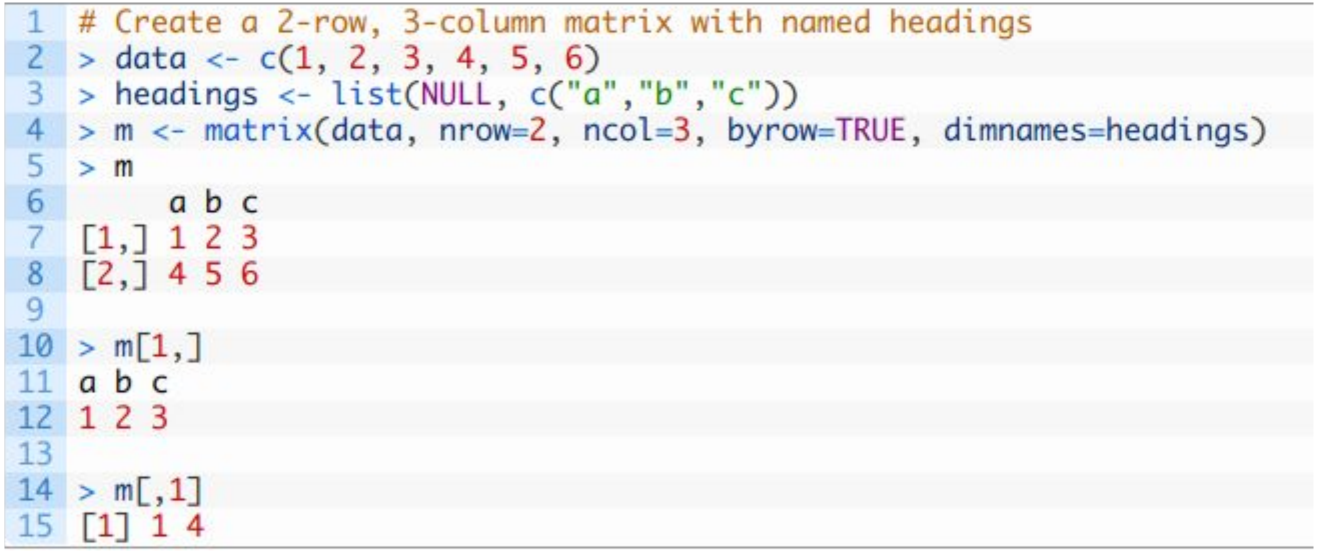
Vectors are lists of data that can be the same types.

- Notice that vectors are 1-index (indexes start at 1 not 0).
- You will use the *c()* function a lot to concatenate variables into a vector.



```
> # create a vector using the c() function
> v <- c(98, 99, 100)
> v
[1]  98  99 100
> v[1:2]
[1] 98 99

> # create a vector from a range of integers
> r <- (1:10)
> r
 [1]  1  2  3  4  5  6  7  8  9 10
> r[5:10]
[1]  5  6  7  8  9 10

> # add a new item to the end of a vector
> v <- c(1, 2, 3)
> v[4] <- 4
> v
[1] 1 2 3 4
```

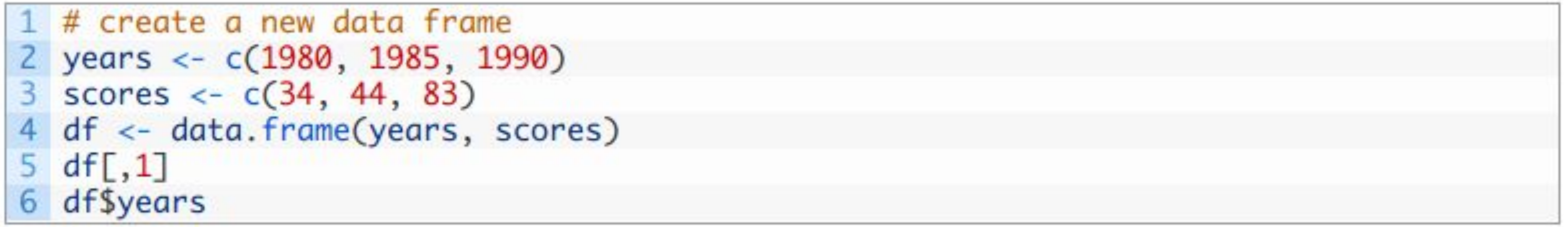
Matrices


```
# Create a 2-row, 3-column matrix with named headings
> data <- c(1, 2, 3, 4, 5, 6)
> headings <- list(NULL, c("a","b","c"))
> m <- matrix(data, nrow=2, ncol=3, byrow=TRUE, dimnames=headings)
> m
     a b c
[1,] 1 2 3
[2,] 4 5 6

> m[1,]
a b c
1 2 3

> m[,1]
[1] 1 4
```

A matrix is a table of data. It has dimensions (rows and columns) and the columns can be named.

- Note the syntax to index into rows [1,] and columns [,1] of a matrix.

DataFrames



```
# create a new data frame
years <- c(1980, 1985, 1990)
scores <- c(34, 44, 83)
df <- data.frame(years, scores)
df[,1]
df$years
```

- A **matrix** is much simpler structure, intended for mathematical operations. A **data frame** is more suited to representing a table of data and is expected by modern implementations of machine learning algorithms in R.

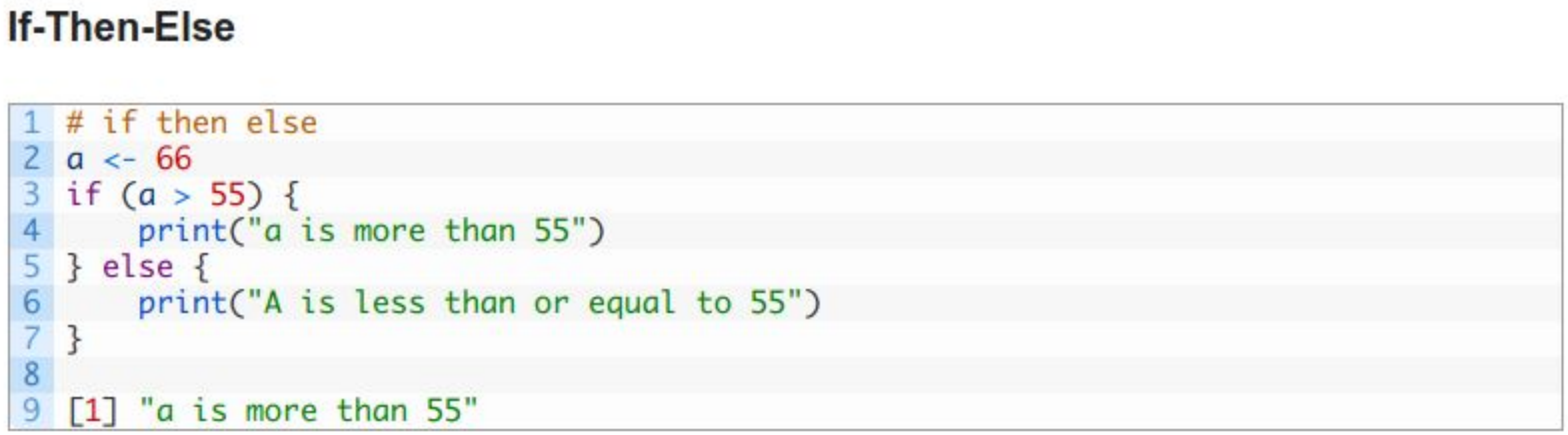
- Note that you can index into rows and columns of a data frame just like you can for a matrix. Also note that you can reference a column using its name (*df$years*)

FLOW CONTROLS

R supports all the same flow control structures that you are used to.

- If-Then-Else
- For Loop
- While Loop

FLOW CONTROLS

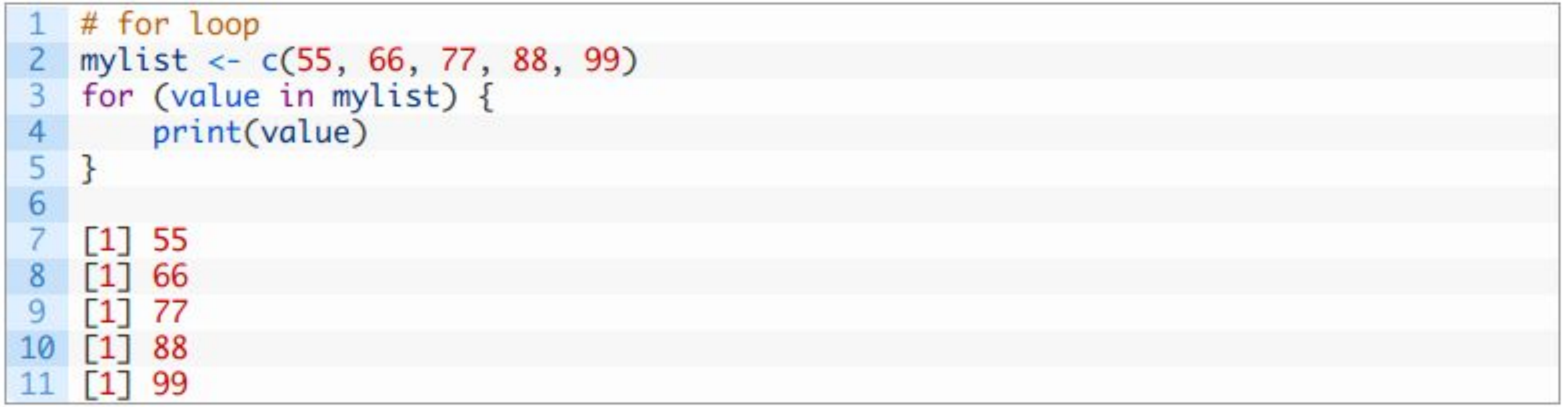


If-Then-Else

```
# if then else
a <- 66
if (a > 55) {
    print("a is more than 55")
} else {
    print("A is less than or equal to 55")
}

[1] "a is more than 55"
```

FLOW CONTROLS

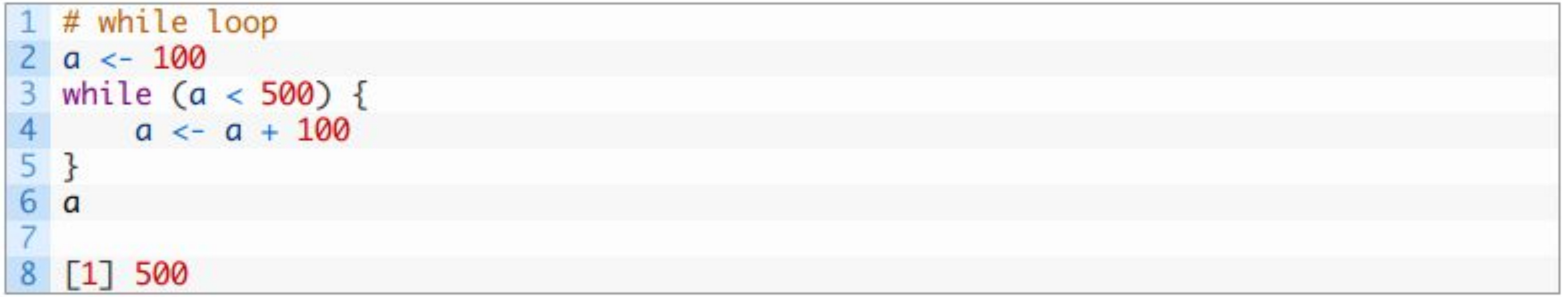
For Loop



```
# for loop
mylist <- c(55, 66, 77, 88, 99)
for (value in mylist) {
    print(value)
}

[1] 55
[1] 66
[1] 77
[1] 88
[1] 99
```

FLOW CONTROLS

While Loop



```r
# while loop
a <- 100
while (a < 500) {
    a <- a + 100
}
a

[1] 500
```

FUNCTIONS

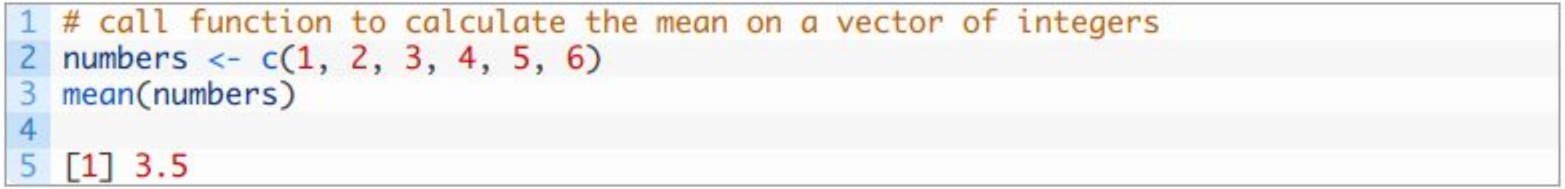
Functions let you group code and call that code repeatedly with arguments.

The two main concerns with functions are:

1. **Calling Functions**
2. **Help For Functions**
3. **Writing Custom Functions**

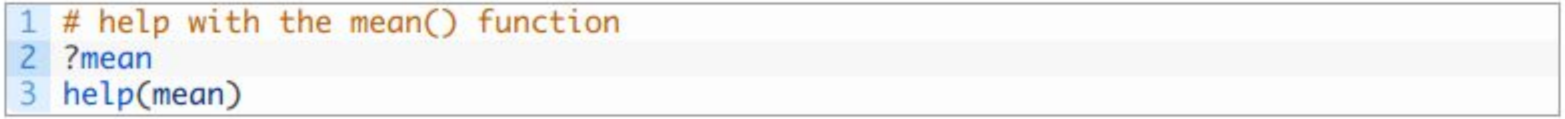
CALL FUNCTIONS

R has many built in functions and additional functions can be provided by installing and loading third-party packages.



```r
# call function to calculate the mean on a vector of integers
numbers <- c(1, 2, 3, 4, 5, 6)
mean(numbers)

[1] 3.5
```

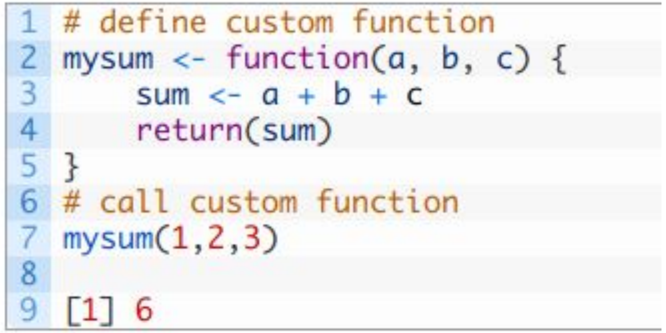
Help for Functions



```
# help with the mean() function
?mean
help(mean)
```

- You can help help with a function in R by using the question mark operator (?) followed by the function name.
- Alternatively, you can call the *help()* function and pass the function name you need help with as an argument (e.g. *help(mean)*).

CUSTOM FUNCTIONS

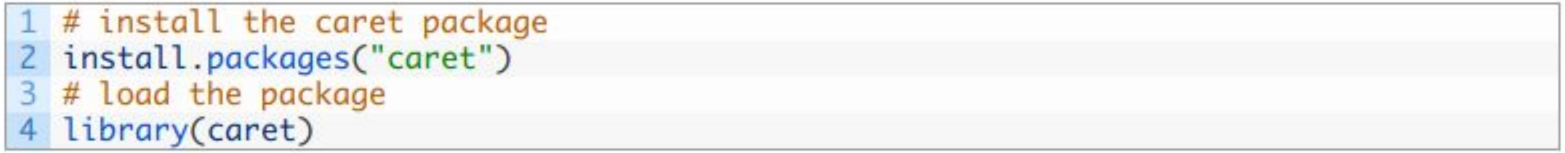


```
# define custom function
mysum <- function(a, b, c) {
    sum <- a + b + c
    return(sum)
}
# call custom function
mysum(1,2,3)

[1] 6
```

You can define your own functions that may or may not take arguments or return a result.

PACKAGES

You can install a package hosted on CRAN by calling a function. It will then pop-up a dialog to ask you which mirror you would like to download the package from.



```r
# install the caret package
install.packages("caret")
# load the package
library(caret)
```

For example, here is how you can install the caret package which is very useful in machine learning:

Things To Remember

Here are five quick tips to remember when getting started in R:

- **Assignment**. R uses the arrow operator (<-) for assignment, not a single equals (=).
- **Case Sensitive.** The R language is case sensitive, meaning that C() and c() are two different function calls.
- **Help**. You can help on any operator or function using the help() function or the ? operator and help with packages using the double question mark operator (??).
- **How To Quit**. You can exit the R interactive environment by calling the q() function.
- **Documentation**. R installs with a lot of useful documentation. You can review it in the browser by typing: *help.start()*

LABSIN

Preguntas?
Opiniones?

UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO