Day 1

Content for this week

- Day 1 - Recap of basics in R with some additional advanced concepts
- Day 2 - tidyverse and plotting different kinds of plots in R
- Day 3 - Probability, permutations and combinations, distributions of data, descriptive statistics, parametric and non-parametric tests
- Day 4 - Vector spaces and matrices/dimensionality reduction and basics of linear algebra
- Day 5 - Different types of regressions

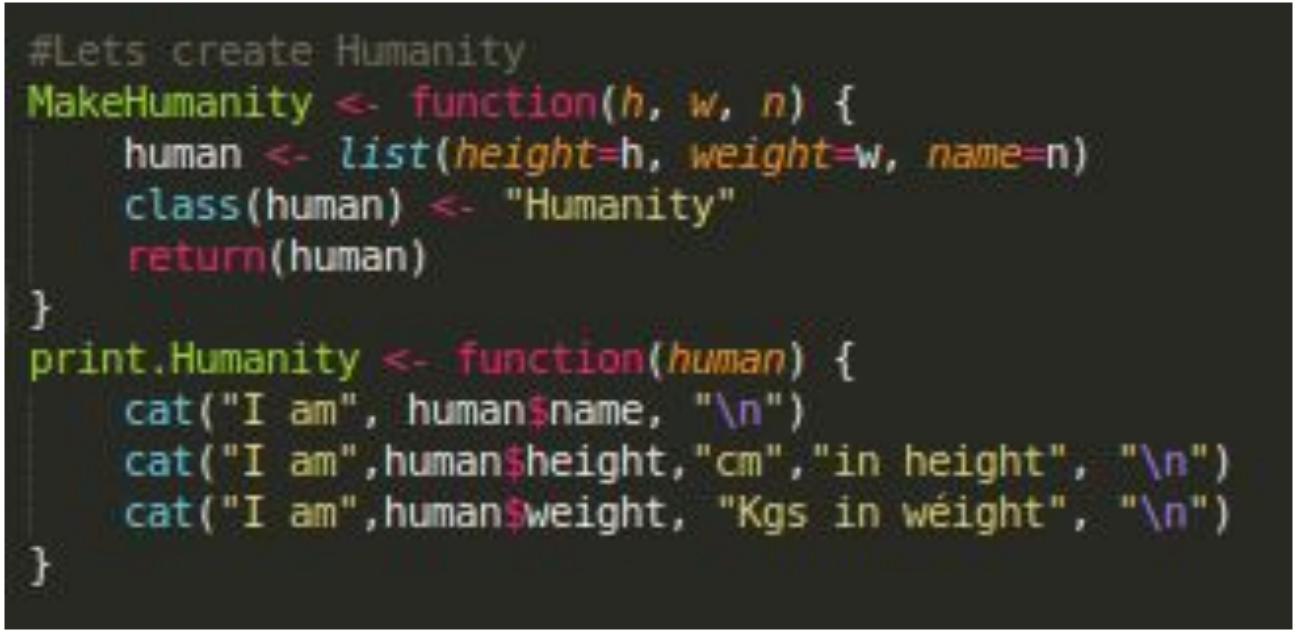
Basics in R

- Data structures and OOP in R
- Recursive functions, infix operators and switch functions
- Efficient looping
- Exercises - Also contains some logical questions for you to think!

Creating new Data structures

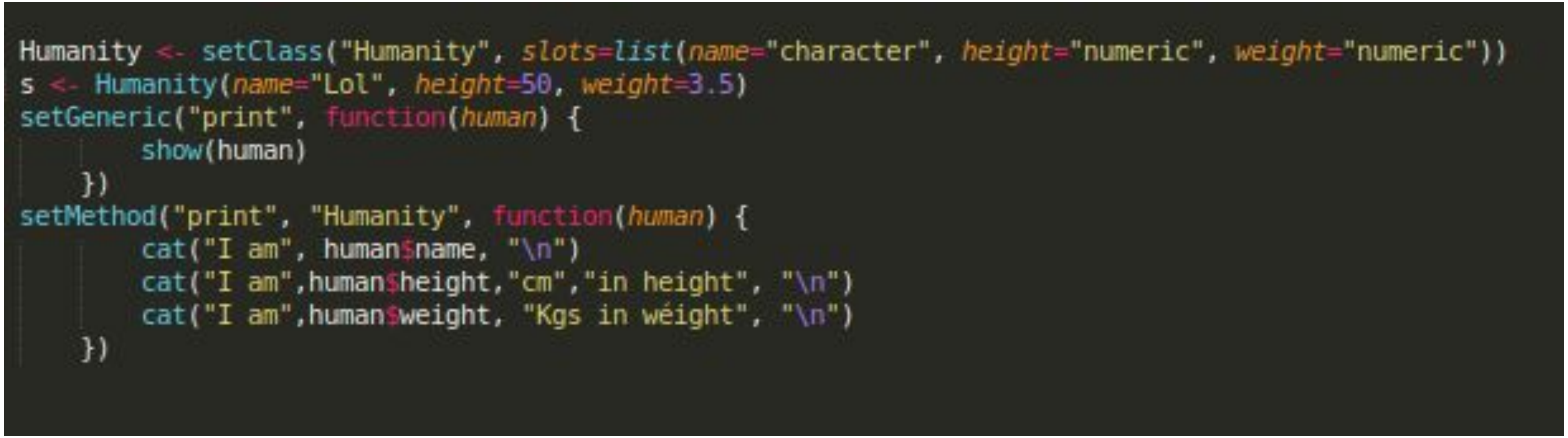
- **S3 :** A list of lists
- **S4 :** An object with different slots - different from list
- **R7 :** Relatively new, is sort of a combination of S3 and S4 but most importantly this is proper object oriented programming

S3 example code



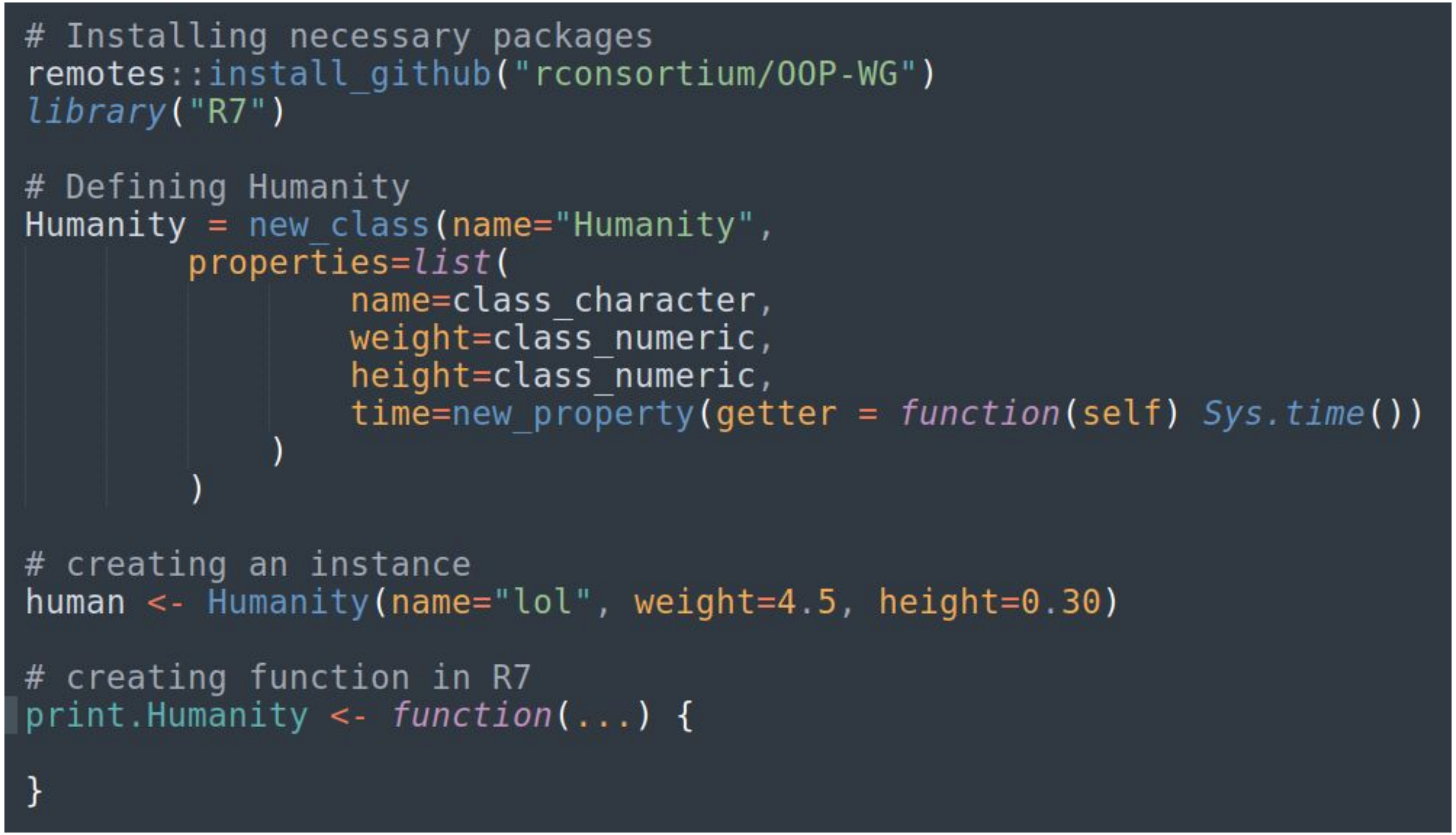
```
#Lets create Humanity
MakeHumanity <- function(h, w, n) {
    human <- list(height=h, weight=w, name=n)
    class(human) <- "Humanity"
    return(human)
}
print.Humanity <- function(human) {
    cat("I am", human$name, "\n")
    cat("I am",human$height,"cm","in height", "\n")
    cat("I am",human$weight, "Kgs in wéight", "\n")
}
```

S4 example code



```r
Humanity <- setClass("Humanity", slots=list(name="character", height="numeric", weight="numeric"))
s <- Humanity(name="Lol", height=50, weight=3.5)
setGeneric("print", function(human) {
        show(human)
    })
setMethod("print", "Humanity", function(human) {
        cat("I am", human$name, "\n")
        cat("I am",human$height,"cm","in height", "\n")
        cat("I am",human$weight, "Kgs in wéight", "\n")
    })
```

R7 Example code



```r
# Installing necessary packages
remotes::install_github("rconsortium/OOP-WG")
library("R7")

# Defining Humanity
Humanity = new_class(name="Humanity",
        properties=list(
                name=class_character,
                weight=class_numeric,
                height=class_numeric,
                time=new_property(getter = function(self) Sys.time())
            )
        )

# creating an instance
human <- Humanity(name="lol", weight=4.5, height=0.30)

# creating function in R7
print.Humanity <- function(...) {

}
```

What is Object-oriented programming

- The concept of programming in which you treat things like “objects” literally
- An object is used to store the data of any kind and its respective class is the blueprint for that object
- Some things you need to know for design pattern!
  - **Abstraction:** Show some data, hide the rest.
  - **Encapsulation:** Get everything together! Don’t separate data and methods.
  - **Inheritance:** Recreate parent class with modifications.
  - **Polymorphism:** Hierarchy of classes with different inheritances of methods.

Why Use Object Oriented Programming

- There is a predefined structure for everything
- Troubleshooting becomes easier
- Reuse of code - saves time
- Can be made flexible through inheritance

R7 vs S3

S3 objects have a class attribute. R7 objects also have an R7_class attribute that contains the object that defines the class.

S3 objects have *attributes*. R7 objects have *properties* (which are built on top of attributes). This means that you can still access properties using the
attr() function.

R7 vs S4

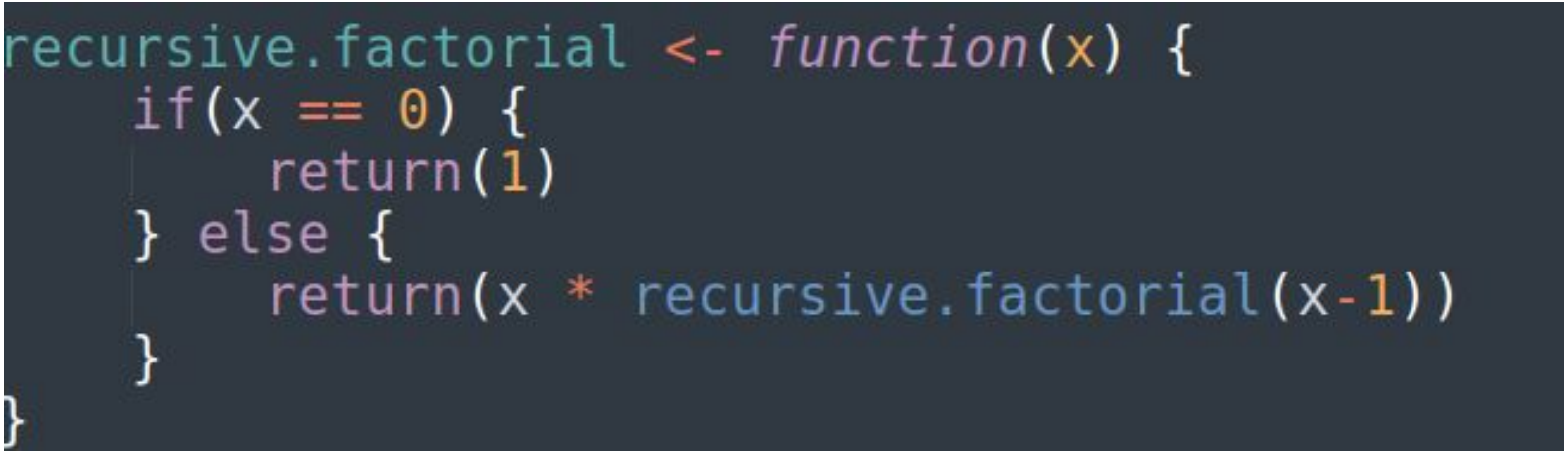
The main difference between the two is that, in R7 objects, properties can be dynamic. As with S3, you can combine R7 methods with S4 generics, and vice versa. S4 classes can extend S3 classes (which extends to cover R7 classes). However, R7 classes cannot be used to extend S4 classes.

Recursive Functions

A function that calls itself is called a recursive function and this technique is known as recursion.

This special programming technique can be used to solve problems by breaking them into smaller and simpler sub-problems.

Example of a recursive function

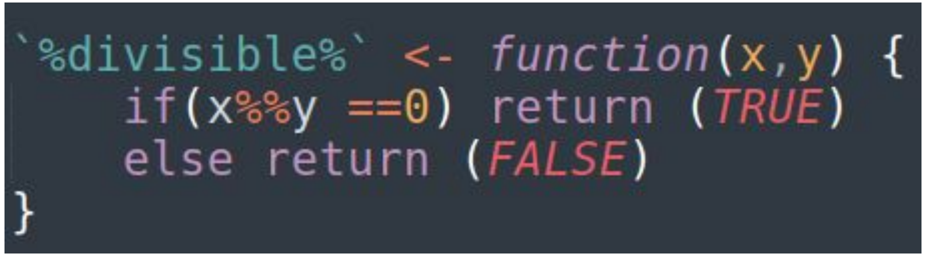


```r
recursive.factorial <- function(x) {
    if(x == 0) {
        return(1)
    } else {
        return(x * recursive.factorial(x-1))
    }
}
```

Infix operators

the expression a+b is actually calling the function `+`() with the arguments a and b, as `+`(a, b)

To create new infix operators you can do the following:



```
`%divisible%` <- function(x,y) {
    if(x%%y ==0) return (TRUE)
    else return (FALSE)
}
```

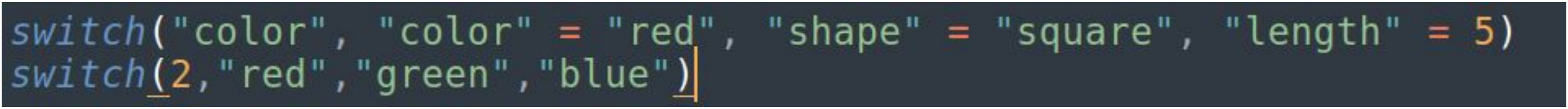
Predefined infix operators in R

%%	Remainder operator
%/%	Integer division
%*%	Matrix multiplication
%o%	Outer product
%x%	Kronecker product
%in%	Matching operator

Switch functions

The switch() function in R tests an expression against elements of a list.

If the value evaluated from the expression matches item from the list, the corresponding value is returned.



```
switch("color", "color" = "red", "shape" = "square", "length" = 5)
switch(2,"red","green","blue")
```

Efficient Looping in R

For loops are not traditionally fast in R. There are other looping options that are better:

- apply
- lapply
- vapply
- sapply
- tapply

Apply

apply(X, MARGIN, FUN)

Here:

- x: an array or matrix
- MARGIN: take a value or range between 1 and 2 to define where to apply the function:
  - MARGIN=1`: the manipulation is performed on rows
  - MARGIN=2`: the manipulation is performed on columns
  - MARGIN=c(1,2)` the manipulation is performed on rows and columns
- FUN: tells which function to apply. Built functions like mean, median, sum, min, max and even user-defined functions can be applied>

Example:

```
m1 <- matrix(C<-(1:10),nrow=5, ncol=6)
```

```
a_m1 <- apply(m1, 2, sum)
```

Lapply

```
lapply(X, FUN)
```

Arguments:

- X: A vector or an object
- FUN: Function applied to each element of x

Example:

```
movies <- c("SPIDERMAN","BATMAN","VERTIGO","CHINATOWN")

movies_lower <-lapply(movies, tolower)
```

Sapply

```
sapply(X, FUN)
```

Arguments:

- X: A vector or an object
- FUN: Function applied to each element of x

Example:

Sapply vs lapply

The difference between lapply and sapply functions is that the sapply function is a wrapper of the lapply function and it returns a vector, matrix or an array instead of a list.

```
sapply(..., simplify=FALSE) == lapply(...)
```

Additionally both of these are same:

- `sapply(some_list, `[`, 1)`
- `sapply(some_list, function(x) x[1])`

tapply

```
tapply(X, INDEX, FUN = NULL)
```

Arguments:

- X: An object, usually a vector
- INDEX: A list containing factor
- FUN: Function applied to each element of x

Example:

```
data(iris)

tapply(iris$Sepal.Width, iris$Species, median)
```

On to Exercises

References

- https://www.datamentor.io/r-programming/
- https://ademos.people.uic.edu/Chapter4.html
- R documentation