Topic 1 Interlude: Writing functions

Basics of coding in R

Sonja Petrović

Created for ITMD/ITMS/STAT 514

Spring 2021.

Context

How are we using R?

- Code is modular, nothing is "from scratch"
- **Functions** are the basic building block of what we do.

Context

How are we using R?

- Code is modular, nothing is "from scratch"
- **Functions** are the basic building block of what we do.

Functions

- We have used a lot of built-in functions: `mean()`, `subset()`, `plot()`, `read.table()`...
- An important part of programming and data analysis is to write custom functions
- Functions help make code **modular**
- Functions make debugging easier
- Remember: this entire class is about applying *functions* to *data*

What is a function?

> *A function is a machine that turns **input objects** (arguments) into an **output object** (return value) according to a definite rule.*

- Let's look at a really simple function

```
addOne <- function(x) {
  x + 1
}
```

- `x` is the **argument** or **input**
- The function **output** is the input `x` incremented by 1

```
addOne(12)
```

```
[1] 13
```

More interesting example

- Here's a function that returns a % given a numerator, denominator, and desired number of decimal values

```
calculatePercentage <- function(x, y, d) {
  decimal <- x / y# Calculate decimal value
  round(100 * decimal, d) # Convert to % and round to d digits
}

calculatePercentage(27, 80, 1)
```

```
[1] 33.8
```

- If you're calculating several %'s for your report, you should use this kind of function instead of repeatedly copying and pasting code

Function returning a list

- Here's a function that takes a person's full name (FirstName LastName), weight in lb and height in inches and converts it into a list with the person's first name, person's last name, weight in kg, height in m, and BMI.

```
createPatientRecord <- function(full.name, weight, height) {
  name.list <- strsplit(full.name, split=" ")[[1]]
  first.name <- name.list[1]
  last.name <- name.list[2]
  weight.in.kg <- weight / 2.2
  height.in.m <- height * 0.0254
  bmi <- weight.in.kg / (height.in.m ^ 2)
  list(first.name=first.name, last.name=last.name,
       weight=weight.in.kg, height=height.in.m,
       bmi=bmi)
}
```

Trying out the function

```
createPatientRecord("Michael Smith", 185, 12 * 6 + 1)
```

```
$first.name
[1] "Michael"

$last.name
[1] "Smith"

$weight
[1] 84.09091

$height
[1] 1.8542

$bmi
[1] 24.45884
```

Another example: 3 number summary

- Calculate mean, median and standard deviation

```
threeNumberSummary <- function(x) {
  c(mean=mean(x), median=median(x), sd=sd(x))
}
x <- rnorm(100, mean=5, sd=2) # Vector of 100 normals
                              # with mean 5 and sd 2
threeNumberSummary(x)
```

```
    mean   median       sd
5.162890 5.246974 1.758908
```

If-else statements

- Oftentimes we want our code to have different effects depending on the features of the input
- Example: Calculating a student's letter grade
  - If grade >= 90, assign A
  - Otherwise, if grade >= 80, assign B
  - Otherwise, if grade >= 70, assign C
  - In all other cases, assign F
- To code this up, we use if-else statements

If-else Example: Letter grades

```
calculateLetterGrade <- function(x) {
  if(x >= 90) {
    grade <- "A"
  } else if(x >= 80) {
    grade <- "B"
  } else if(x >= 70) {
    grade <- "C"
  } else {
    grade <- "F"
  }
  grade
}

course.grades <- c(92, 78, 87, 91, 62)
sapply(course.grades, FUN=calculateLetterGrade)
```

```
[1] "A" "C" "B" "A" "F"
```

return()

- In the previous examples we specified the output simply by writing the output variable as the last line of the function
- More explicitly, we can use the `return()` function

```
addOne <- function(x) {
  return(x + 1)
}

addOne(12)
```

```
[1] 13
```

- We will generally avoid the `return()` function, but you can use it if necessary or if it makes writing a particular function easier.
- Google's style guide suggests explicit returns. Most do not.

License

This document is disstributed to ITMD/ITMS/STAT 514, Spring 2021, at Illinois Tech.