Session 7
Functions 1

R for Social Data Science

Jeffrey Ziegler, PhD

Assistant Professor in Political Science & Data Science
Trinity College Dublin

Fall 2022

ROAD MAP FOR TODAY

Last time:

- Straight-line and branching programs

- Algorithms

- Conditional statements

- Loops and Iteration

This time:

- Decomposition and abstraction

- Arguments, environments

- Function definition and function call
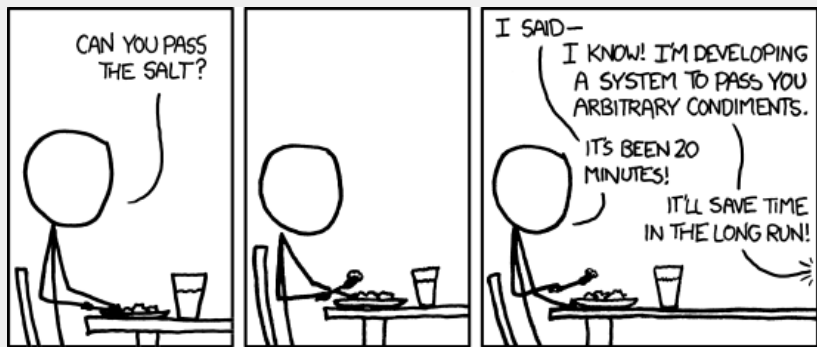
Decomposition and abstraction

Source: IKEA

DECOMPOSITION AND ABSTRACTION

- So far:
    - ▶ built-in types
    - ▶ assignments
    - ▶ branching and looping constructs
- In principle, any problem can be solved just with those
- But a solution would be non-modual and hard-to-maintain
- Functions provide *decomposition* and *abstraction*

Source: xkcd

Functions in R

- Function call is the centerpiece of computation in R

- It involves function object and objects that are supplied as arguments

- Functions in R do not have side-effects (nonlocal modifications of input objects)

- In R we use function 'function()' to create a function object

- Functions are also referred to as *closures* in some R documentation

```
<function_name> <- function(<arg_1>, <arg_2>, ..., <arg_n>) {
<function_body>
}
```

```r
1 foo <- function(arg) {
2   # <function_body>
3 }
```

Function components

- Body ('body()') - code inside the function

- List of arguments ('formals()') - controls how function is called

- Environment/scope/namespace ('environment()') - location of function's definition and variables

Function components example

```
1  is_positive <- function(num) {
2    if (num > 0) {
3      return(TRUE)
4    } else {
5      return(FALSE)
6    }
7  }
8  body(is_positive)
```

```
{
if (num > 0) {
return(TRUE)
} else {
return(FALSE)
} }
```

```
1  formals(is_positive)
```

```
$num
```

```
1  environment(is_positive)
```

```
<environment: R_GlobalEnv>
```

FUNCTION CALL

- Function is executed until:
  - ▶ Either 'return()' function is encountered
  - ▶ There are no more expressions to evaluate
- Function call always returns a value:
  - ▶ Argument of 'return()' function call
  - ▶ Value of last expression if no 'return()' (implicit return)
- Function can return only one object
  - ▶ But you can combine multiple R objects in a list

Function call example

```
1  is_positive <- function(num){
2    if (num > 0){
3      res <- TRUE
4    } else{
5      res <- FALSE
6    }
7    return(res)
8  }
9  res_1 <- is_positive(5)
10 res_2 <- is_positive(-7)
11 print(res_1)
12 print(res_2)
```

```
[1] TRUE
[1] FALSE
```

IMPLICIT RETURN EXAMPLE

```
1  is_positive <- function(num){
2    if (num > 0){
3      res <- TRUE
4    } else{
5      res <- FALSE
6    }
7    res
8  }
9  res_1 <- is_positive(5)
10 res_2 <- is_positive(-7)
11 print(res_1)
12 print(res_2)
```

```
[1] TRUE
[1] FALSE
```

IMPLICIT RETURN EXAMPLE

```
1  # While this function provides the same functionality as
     the two versions above
2  # This is an example of a bad programming style, return
     value is very unintuitive
3  is_positive <- function(num){
4    if (num > 0){
5      res <- TRUE
6    } else{
7      res <- FALSE
8    }
9  }
10 res_1 <- is_positive(5)
11 res_2 <- is_positive(-7)
12 print(res_1)
13 print(res_2)
```

```
[1] TRUE
[1] FALSE
```

Function arguments

- *Arguments* provide a way of giving input to a function

- Arguments in function definition are *formal arguments*

- Arguments in function invocations are *actual arguments*

- When a function is invoked (called) arguments are matched and bound to local variable names

- R matches arguments in 3 ways:
    1. by <u>exact name</u>
    2. by <u>partial name</u>
    3. by <u>position</u>

- It is a good idea to only use unnamed (positional) for main (first one or two) arguments

FUNCTION ARGUMENTS EXAMPLE

```
1  format_date <- function(day, month, year, reverse = TRUE){
2    if (isTRUE(reverse)){
3      formatted <- paste(as.character(year), as.character(month), as
       .character(day), sep = "-")
4    } else {
5      formatted <- paste(as.character(day), as.character(month), as.
       character(year), sep = "-")
6    }
7    return(formatted)
8  }
9  format_date(4,10,2021)

   [1] "2021-10-4"

1  # Technically correct, but rather unintuitive
2  format_date(y = 2021, m = 10, d = 4)

   [1] "2021-10-4"

1  format_date(y = 2021, m = 10, d = 4, FALSE)

   [1] "2021-10-4"
```

NESTED FUNCTIONS

```
1   which_integer <- function(num){
2     even_or_odd <- function(num){
3       if (num %% 2 == 0){
4         return("even")
5       } else {
6         return("odd")
7       }
8     }
9     eo <- even_or_odd(num)
10    if (num > 0){
11      return(paste0("positive ", eo))
12    } else if (num < 0){
13      return(paste0("negative ", eo))
14    } else {
15      return("zero")
16    }
17  }
18  which_integer(-43)
```

```
[1] "negative odd"
```

```
1   even_or_odd(-43)
```

```
Error in even_or_odd(-43) : could not find function "even_or_odd"
```

R environment basics

- Variables (aka names) exist in an *environment* (aka namespace/scope in Python)

- The same R object can have different names

- Binding of objects to names (assignment) happens within a specific environment

- Most environments get created by function calls

- Approximate hierarchy of environments:

  - *Execution* environment of a function

  - *Global* environment of a script

  - *Package* environment of any loaded packages

  - *Base* environment of base R objects

R ENVIRONMENT EXAMPLE

```r
1  x <- 42
2  # is equivalent to:
3  # Binding R object '42', double vector of length 1, to name 'x' in
       the global environment
4  assign("x", 42, envir = .GlobalEnv)
5  x
```

```
   [1] 42
```

```r
1  x <- 5
2  foo <- function(){
3     x <- 12
4     return(x)
5  }
6  y <- foo()
7  print(y)
8  print(x)
```

```
   [1] 12
   [1] 5
```

EVERY OPERATION IS A FUNCTION CALL

Matthew Kay @mjskay · Oct 31
remember kids everything in #rstats is a function

```
y = 9

for (x in 1:y) {
  if (x > 2) {
    xy = x * y
    cat(x, "*", y, "is", xy, "\n")
  }
}
#> 3 * 9 is 27
#> 4 * 9 is 36
#> 5 * 9 is 45
#> 6 * 9 is 54
#> 7 * 9 is 63
#> 8 * 9 is 72
#> 9 * 9 is 81
```

```
`=`(y, 9)

`for`(x, `:`(1, y), `{`(
  `if`(`>`(x, 2), `{`(
    `=`(xy, `*`(x, y)),
    cat(x, "*", y, "is", xy, "\n")
  ))
))
#> 3 * 9 is 27
#> 4 * 9 is 36
#> 5 * 9 is 45
#> 6 * 9 is 54
#> 7 * 9 is 63
#> 8 * 9 is 72
#> 9 * 9 is 81
```

💬 11   ↻ 38   ❤ 377   ⬆

EXAMPLES OF OPERATORS AS FUNCTION CALLS

```
1  '+'(3, 2) # Equivalent to: 3 + 2
```

```
[1] 5
```

```
1  '<-'(x, c(10, 12, 14)) # x <- c(10, 12, 14)
```

```
[1]  10 12 14
```

```
1  '['(x, 3) # x[3]
```

```
[1] 14
```

```
1  '>'(x, 10) # x > 10
```

```
[1] FALSE  TRUE  TRUE
```

OVERVIEW

This time:

- Decomposition and abstraction

- Arguments, environments

- Function definition and function call

Next time:

- Functionals

- Scoping in R