Choices

Ken Wu

6/24/2021

if statement

Basic syntax of an if statement in R:

```
# Single argument
if (condition) true_action
# Optional else statement
if (condition) true_action else false_action
```

If condition is TRUE, true_action is evaluated; if condition is FALSE, the optional false_action is evaluated.

```
# Grade
grade <- function(x) {
  if (x >= 90) {
    "A"
  } else if (x >= 80) {
    "B"
  } else if (x >= 50) {
    "C"
  } else {
    "F"
  }
}
# Test
grade(x = 50)
[1] "C"
grade(x = 90)
[1] "A"
grade(x = 81)
[1] "B"
```

- When using a single argument form without the else statement, if *invisibly* returns NULL if the condition is FALSE. Embrace the function with () to see the NULL value.

```
# No else statement
grade_new <- function(x) {
  if (x >= 90) {
    "A"
  } else if (x >= 80) {
    "B"
```

```
  } else if (x >= 50) {
    "C"
  }
}
# Test
(grade_new(x = 34))
NULL
```

- The condition should evaluate to a *single* TRUE or FALSE.

Vectorized if

- **ifelse(test = an object which can be coerced to logical mode, yes = return values for true elements of test, no = return values for false elements of test)** from base R

The function ifelse() returns a value with the same shape as test which is filled with elements selected from either yes or no depending on whether the element of test is TRUE or FALSE.

- **if_else(condition, true, false, missing = NULL)** from dpylr

Compared to the base ifelse(), this function is more strict. It checks that true and false are the same type. This strictness makes the output type more predictable, and makes it somewhat faster.

```
# Vector
x <- c(1:10, NA_real_, NA_real_)
x
 [1]  1  2  3  4  5  6  7  8  9 10 NA NA
# Vector of remainders
x %% 5
 [1]  1  2  3  4  0  1  2  3  4  0 NA NA
# Logical vector of whether each remainder is zero
x %% 5 == 0
 [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE    NA    NA
# Integer equivalent to the logical vector above
which(x = (x %% 5 == 0))
[1]  5 10
# Vectorize if
ifelse(test = (x %% 5 == 0), yes = "XXX", no = "No")
 [1] "No"  "No"  "No"  "No"  "XXX" "No"  "No"  "No"  "No"  "XXX" NA    NA
# Using dpylr's function
if_else(condition = (x %% 2 == 0), true = "even", false = "odd", missing = "Missing")
 [1] "odd"     "even"    "odd"     "even"    "odd"     "even"    "odd"
 [8] "even"    "odd"     "even"    "Missing" "Missing"
```

- The case_when() function allows you to vectorise multiple if_else() statements. It is an R equivalent of the SQL CASE WHEN statement. If no cases match, NA is returned. Use TRUE ~ to handle the ELSE group.

- Note that NA values in the vector x do not get special treatment. If you want to explicitly handle NA values you can use the `is.na` function:

```
dplyr::case_when(
  x %% 35 == 0 ~ "fizz buzz",
  x %% 5 == 0 ~ "fizz",
  x %% 7 == 0 ~ "buzz",
  is.na(x) ~ "???",
  TRUE ~ "other"
)
 [1] "other" "other" "other" "other" "fizz"  "other" "buzz"  "other" "other"
[10] "fizz"  "???"   "???"
```

---

switch() statement

- Example 1: We can replace:

```
# Function
x_option <- function(x) {
  if (x == "a") {
    "option 1"
  } else if (x == "b") {
    "option 2"
  } else if (x == "c") {
    "option 3"
  } else {
    stop("Invalid `x` value")
  }
}
# Test
x_option("a")
[1] "option 1"
x_option("c")
[1] "option 3"
```

with a more succinct:

```
# Succinct function
x_option_succinct <- function(x) {
  switch(x,
    a = "option 1",
    b = "option 2",
    c = "option 3",
    stop("Invalid `x` value")
  )
}
# Test
x_option_succinct("b")
[1] "option 2"
```

The last component of a switch() should always throw an error, otherwise unmatched inputs will invisibly return NULL:

```
(switch("c",
  a = 1,
  b = 2
))
NULL
```

- Example 2

```
# Function
centre <- function(x, type) {
  switch(type,
    mean = mean(x),
    median = median(x),
    trimmed = mean(x, trim = .1),
    max = max(x),
    sd = sd(x),
    IQR = IQR(x),
    stop("Invalid input")
  )
}
# Test
x <- rcauchy(n = 10)
centre(x, type = "mean")
[1] -3.883641
centre(x, type = "median")
[1] -0.2236557
centre(x, type = "trimmed")
[1] -0.4383195
centre(x, type = "sd")
[1] 14.32828
centre(x, type = "max")
[1] 8.445577
```

- Example 3: If multiple inputs have the same output, you can leave the right hand side of = empty and the input will "fall through" to the next value.

```
# Function
legs <- function(x) {
  switch(x,
    cow = ,
    horse = ,
    dog = 4,
    human = ,
    chicken = 2,
    plant = 0,
```

```
    stop("Unknown input")
  )
}
# Test
legs("plant")
[1] 0
legs("human")
[1] 2
legs("horse")
[1] 4
legs("dog")
[1] 4
legs("cow")
[1] 4
```

Exercise 1

- Example 1: character and integer results in character

```
# Yes has numeric type and no has character type
ifelse(test = c(TRUE, FALSE), yes = 1, no = "no")
[1] "1"  "no"
# Returns a character vector
str(ifelse(test = c(TRUE, FALSE), yes = 1, no = "no"))
 chr [1:2] "1" "no"
```

- Example 2

```
# Test condition is a logical NA
ifelse(test = NA, yes = 1, no = "no")
[1] NA
# Returns a logical NA
typeof(ifelse(test = NA, yes = 1, no = "no"))
[1] "logical"
str(ifelse(test = NA, yes = 1, no = "no"))
 logi NA
```

- Example 3: integer and double results in double

```
# Yes has integer type and no has double type
ifelse(test = c(TRUE, FALSE), yes = as.integer(x = 4), no = as.double(x = 2.2))
[1] 4.0 2.2
# Returns a double vector
typeof(ifelse(test = c(TRUE, FALSE), yes = as.integer(x = 4), no = as.double(x = 2.2)))
[1] "double"
```

- Example 4: integer and logical results in integer

```
# Yes has integer type and no has logical type
ifelse(test = c(TRUE, FALSE), yes = TRUE, no = as.integer(x = 4))
[1] 1 4
# Returns an integer vector
str(ifelse(test = c(TRUE, FALSE), yes = TRUE, no = as.integer(x = 4)))
 int [1:2] 1 4
```

When "yes" and "no" are not the same type, the coercion type fixed order is used: character → double → integer → logical. Then, the type of the resulting vector will be coerced from logical to any other type depending on whether test is TRUE or FALSE. It will be coerced from logical to accommodate first any values taken from yes and then any values taken from no.

Exercise 2

The condition argument in the if statement requires a length-one logical vector that is not NA. *Other types are coerced to logical if possible, ignoring any class.* Therefore, if also accepts a numeric vector where 0 is treated as FALSE and all other numbers are treated as TRUE.

```
# This function below results in a length one integer vector 10
length(x = 1:10)
[1] 10
# Structure
str(length(x = 1:10))
 int 10
# Since 10 is a non-zero it is treated as TRUE when converting to a logical vector
as.logical(length(x = 1:10))
[1] TRUE
# If statement should return "no empty" because length(x = 1:10) evaluates to TRUE
if (length(x = 1:10)) "not empty" else "empty"
[1] "not empty"
# Coercion of a length on vector 0
as.logical(length(numeric(0)))
[1] FALSE
# If statement evaluates to 0 and so FALSE action is carried out
if (length(numeric(0))) "not empty" else "empty"
[1] "empty"
```

```
rm(list = ls())
```