Основы программирования в R

Условные конструкции и цикл `for`

Алла Тамбовцева, НИУ ВШЭ

Содержание

Условные конструкции

Для того, чтобы в R осуществлял различные действия в зависимости от того, выполняется ли определённое условие или нет, понадобятся операторы условия: `if` и `else`. Например, мы сохраняем в R оценку по курсу и хотим, чтобы он выводил на экран "Good!" и "You are genius!!!", если она больше или равна 5 и "Bad!" — во всех остальных случаях.

```
mark <- 2

if (mark >= 5){
  print("Good!")
  print("You are genius!!!")
}else{
  print("Bad.")
}
```

```
## [1] "Bad."
```

Само условие записывается в круглых скобках, а то, что R должен сделать в случае, если условие выполняется, перечисляется в фигурных скобках. Логика работы конструкции *if-else* ничем не отличается от Python, просто в R другой синтаксис, который требует наличие скобок вместо отступов. Если мы не поставим фигурные скобки, R не поймёт, что все строки выше, начиная с `print("Good!")` и заканчивая `print("Bad.")`, относятся к одному блоку кода, который нужно запускать сразу.

Если в случае выполнения условия R должен сделать что-то, что умещается в одну строчку кода, фигурные скобки можно опустить (и вообще всё записать в одну строчку):

```
if (mark >= 5) print("Good.") else print("Bad.")
```

```
## [1] "Bad."
```

Если необходимо применить несложную конструкцию *if-else* к каждому элементу вектора, можно воспользоваться функцией `ifelse()`. Сохраним вектор оценок `grades` и на его основе создадим вектор `grades_binary`, где 1 будет соответствовать оценкам выше 3, а 0 — всем остальным оценкам.

```
grades <- c(5, 7, 3, 4, 10, 8)
grades_binary <- ifelse(grades > 3, 1, 0)
grades_binary
```

```
## [1] 1 1 0 1 1 1
```

В функции `ifelse()` на первом месте указывается условие, на втором — значение, которое должно возвращаться, если условие выполняется, на третьем — значение, которое должно возвращаться, если условие не выполняется.

Иногда мы сталкиваемся с необходимостью использовать множественные условия, например, какие-то два условия должны выполняться одновременно. Для этого потребуются те же символы, которые используются в логических выражениях (`&` и `|`). Напишем код, который будет выдавать разные комментарии по оценке в 10-балльной шкале:

- оценка меньше 4: “Bad.”;
- оценка не ниже 4, но ниже 6: “Not bad.”;
- оценка не ниже 6, но ниже 8: “Good.”;
- оценка не ниже 8, но не выше 10: “Excellent.”;
- оценка выше 10: “We are the champions, my friend...” (R поет).

```
mark <- 12
if (mark < 4) print("Bad.") else{
  if (mark >= 4 & mark < 6) print("Not bad.")
  if (mark >= 6 & mark < 8) print("Good.")
  if (mark >= 8 & mark <= 10) print("Excellent.")
  if (mark > 10) print("We are the champions, my friend...")
}
```

```
## [1] "We are the champions, my friend..."
```

Примечание: конечно, здесь можно было бы обойтись и без вложенной конструкции *if-else*, но мы немного усложнили код, чтобы ещё раз потренироваться с постановкой различного вида скобок.

А вот пример с оператором `|`, ситуация, когда мы выводим на экран “Excellent”, если оценка или 8, или 9, или 10:

```
if (mark == 8 | mark == 9 | mark == 10) print("Excellent")
```

Как можно заметить, так же, как и в Python, часть с `else` в условной конструкции здесь тоже является необязательной.

Интересное отличие R от Python заключается в том, что здесь есть базовая функция, которая отвечает за логическую операцию XOR, исключающее «или». В то время как оператор `|` возвращает значение `True`, если хотя бы одно из условий верно, функция `xor()` возвращает значение `True`, если ровно одно из условий верно.

Сравним:

```
# оба условия верны -> хотя бы одно верно -> True
3 < 5 | 7 < 9
```

```
## [1] TRUE
```

```
# оба условия верны -> неверно, что верно только одно -> False
xor(3 < 5, 7 < 9)
```

```
## [1] FALSE
```

А вот в случае, когда только одно из условий верно, `xor()` вернет `True`:

```
xor(3 < 5, 7 > 9)
```

```
## [1] TRUE
```

Цикл `for`

Чтобы повторять действия в R и при этом не копировать один и тот же код много раз, используются циклы. Существует два основных цикла: цикл `for` и цикл `while`. На практике чаще используется цикл

`for`, потому что цикл `while` более специфический и более медленный (и программу с `while` очень легко зациклить).

Стоит также отметить, что для выполнения стандартных задач в рамках анализа данных в R циклы используются нечасто, так как в R многие операции векторизованы — функции применяются сразу к наборам значений, например, к векторам.

```
# не нужно проходить по вектору x и возводить каждый элемент в квадрат
x <- c(1, 2, 3, 8)
x ** 2
```

```
## [1]  1  4  9 64
```

Смысл цикла `for`: «для каждого элемента в наборе (векторе) сделай что-то». У нас есть вектор `v`, и мы хотим вывести его элементы на экран:

```
v <- c(0, 1, NA, 0, 1)
for (i in v){
  print(i)
}
```

```
## [1] 0
## [1] 1
## [1] NA
## [1] 0
## [1] 1
```

То, по чему итерируем (пробегаемся), указывается в круглых скобках, а то, что делаем в ходе цикла — в фигурных.

Для перебора можно задействовать последовательности:

```
for (n in 1:10){
  print(n)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

А теперь совместим цикл и условные конструкции. Вспомним про вектор оценок `grades`:

```
grades
```

```
## [1]  5  7  3  4 10  8
```

Выведем краткий комментарий на каждую из оценок:

```
for (g in grades){
  if (g > 3) print("Pass")
  if (g <= 3) print("Fail")
}
```

```
## [1] "Pass"
```

```
## [1] "Pass"
## [1] "Fail"
## [1] "Pass"
## [1] "Pass"
## [1] "Pass"
```