Урок 4. Функции

С готовыми функциями R мы уже активно работали. Например, когда округляли числа:

```
round(4.6)
```

```
## [1] 5
```

Или суммировали элементы вектора:

```
sum(c(5, 7, 8)) # да и когда создавали вектор, тоже
```

```
## [1] 20
```

Теперь попробуем написать свою функцию. Например, напишем функцию, которая будет принимать на вход вектор и возвращать число элементов, которые делятся на 5 или 10 (иногда полезно для выявления придуманных, сфальсифицированных значений в данных — считается, что люди склонны при придумывании добавлять «удобные» числа: круглые или кратные 5).

Так устроен R, что функцию мы тоже сохраняем в переменную. То есть, если мы хотим создать функцию, мы должны создать переменную с таким названием и присвоить ей объект типа «функция». Давайте назовем нашу функцию `fraud()`, от английского «фальсификация, обман».

```
fraud <- function(){}
```

В круглых скобках укажем аргумент — объект, который функция принимает на вход и с которым она работает. В фигурных скобках указывается тело функции — набор действий, которое она должна выполнять.

```
fraud <- function(v){
n <- length(v[v %% 5 == 0 | v %% 10 == 0])
n
}
```

Для проверки кратности 5 или 10 мы воспользовались оператором %% для определения остатка от деления. Если число нацело делится на 5, то остаток от деления равен 0. То же относится и к 10. Кстати, если число делится на 10, то оно точно делится и на 5. Получается, нам не нужно такое сложное условие – можно обойтись только одной проверкой! Запишем упрощенный код:

```
fraud <- function(v){
n <- length(v[v %% 5 == 0])
n
}
```

Очень полезно обращать внимание на подобные свойства чисел и других объектов, потому что часто знания о них позволяют оптимизировать код: сделать его более лаконичным и быстрым.

Как известно из Python, функция в программировании обычно принимает некоторый объект на вход и что-то возвращает на выходе. Для возвращения используют оператор `return`. В R этот оператор тоже есть, но иногда можно обойтись без него: функция по умолчанию вернет объект, который записан на последней строке кода в теле функции. Проверим – применим функцию к вектору:

```
fraud(c(15, 23, 45, 17, 20)) # действительно
```

```
## [1] 3
```

Если привычнее работать с `return`, давайте его добавим:

```
fraud <- function(v){
n <- length(v[v %% 5 == 0 | v %% 10 == 0])
```

```
return(n)
}
fraud(c(4, 5, 15, 6, 17, 24, 10))
```

```
## [1] 3
```

Обратите внимание, в отличие от Python, в R return() является функцией, а не оператором, поэтому вокруг возвращаемого объекта добавляются круглые скобки.

В Python мы бы создавали функцию так:

```
def fraud(v):
  ...
  return n
```

Как и в Python, функция в R может иметь как основной (в `return`), так и побочный результат. Например, помимо возвращения значения, она может выводить что-то на экран:

```
fraud <- function(v){
    n <- length(v[v %% 5 == 0])
    print("Well done")
    return(n)
}
```

Напоследок убедимся, что функции от двух и более аргументов в R тоже можно создавать. Напишем простенькую функцию, которая принимает на вход два числа и возвращает первое в степени другого.

```
my.test<- function(a, b){
res <- a ^ b
return(res)
}
my.test(3, 4)
```

```
## [1] 81
```

В целом, написание функций в R очень похоже на написание функций в Python, если не смотреть на внутреннее устройство объектов и синтаксис. Однако есть важное отличие: функции в R не умеют возвращать сразу несколько объектов, если они не объединены в вектор или иную структуру явно. В Python такое было возможно, так как при при постановки запятой между элементами автоматически образуется кортеж - структура данных, то есть, другими словами, Python умеет «склеивать» отдельные элементы в структуры, а R нет. Справиться с этой проблемой легко: если функция возвращает несколько объектов, их нужно объединить в вектор вручную. Посмотрим на доработанном примере функции выше:

```
my.test2 <- function(a, b){
  res1 <- a ^ b
  res2 <- b ^ a
  return(c(res1, res2))

}
my.test2(9, 2)
```

```
## [1]  81 512
```

А сейчас самое время выполнить практическое задание.