Creating a loop and function

🔧 Fix Me!

How to create a loop, which will repeat given action x-times, and how to define the function will be demonstrated on creating the script for *random walk*.

The problem

First, the short description of the problem, which should be translated into a script. Imagine that at the beginning, you are standing at the point zero ($x_1 = 0$). In each subsequent step (x_2, x_3, ...), you go forward, but not straight, but either slightly left or slightly right. Decision about the direction of the step is random and could be done e.g. by tossing the coin (you have two possible outcome, heads or tails, and lets say that heads = right and tails = left).

Numerically it will look as following: The current position is zero ($x_1 = 0$). If you should go to left (i.e. the coin shows tails), than add to the current position one ($x_2 = 1$); if you should go right, subtract one ($x_2 = -1$).

Next step is exactly the same, and the new decision about the direction is independent from the previous. The question is, where we will end up after, say, 1000 random steps. One would intuitively expect that, because we walk right and left randomly, in total we still head more or less forward. But is it like this?

Creating the script

Step by step

First, define an empty vector, which will be used to deposit information about your current position:

```
steps.rw <- vector (mode = 'numeric', length = 1000)
```

The firest argument indicates that the vector will contain numbers (alternatives would be `character` or `logical`), the second argument defines the length of the vector, i.e. how many elements it will contain.

And now, you start to walk forward. To choose randomly the direction, use function `sample`, with two arguments, x - the vector from which the elements will be randomly selected, and `size` - number of selected elements from x. The following script:

```
sample (x = c(-1,1), size = 1)

[1] -1
```

will randomly select one value from the vector, containing numbers `-1` and 1, in this case value `-1`. You just made the first step to the right.

Vector `steps.rw` contains only zeroes:

```
steps.rw
   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  [79] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  ...
  ...
```

First element is therefore zero, which accord to your initial position. The second element can be calculated as following:

```
steps.rw [2] <- steps.rw [1] + sample (c(-1,1), 1)
```

You may have realized that in function `sample`, there are no names of arguments (`x` and `size`) - they are not needed, until you write the values for arguments in the defined order (what is this order you can check in the help to the function `sample`, which you call typing `?sample`).

Further steps can be defined in the following way:

```
steps.rw [3] <- steps.rw [2] + sample (c(-1,1), 1)
steps.rw [4] <- steps.rw [3] + sample (c(-1,1), 1)
steps.rw [5] <- steps.rw [4] + sample (c(-1,1), 1)
steps.rw [6] <- steps.rw [5] + sample (c(-1,1), 1)
steps.rw [7] <- steps.rw [6] + sample (c(-1,1), 1)
steps.rw [8] <- steps.rw [7] + sample (c(-1,1), 1)
steps.rw [9] <- steps.rw [8] + sample (c(-1,1), 1)
steps.rw [10] <- steps.rw [9] + sample (c(-1,1), 1)
```

Use of loop

As you may guess, the approach above wouldn't take you too far, as you would have to repeat the line 1000 times. It's time to introduce function to create loop. The name of the function is `for`:

```
for (k in seq (2, 1000))
  {
  steps.rw [k] <- steps.rw [k-1] + sample (c(-1,1),1)
  }
```

The script says the following: the variable k will subsequently be of value 2 to 1000, and with each of this value make the following: at the k-th position of vector `steps.rw` save the value, calculated from k-1 position of vector `steps.rw` by adding randomly selected value `-1` or `1`.

```
steps.rw

   [1]   0  -1   0  -1   0  -1  -2  -3  -4  -3  -2  -3  -4  -3  -4  -5  -4  -5  -4
-5  -4  -5  -6  -7  -8  -9 -10  -9 -10 -11 -10  -9 -10 -11 -10  -9 -10 -11 -10
  [40] -11 -10 -11 -10  -9 -10  -9  -8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3
4   3   2   1   2   3   2   1   2   3   4   5   6   5   4   3   2   3   2   3
  [79]   4   5   6   7   8   7   8   7   6   7   8   7   8   7   8   7   8   7   6
5   6   5   4   5   6   7   8   9  10   9   8   7   8   9  10  11  10   9   8
 [118]   9   8   9  10  11  10  11  12  11  12  13  12  13  14  13  14  15  16  15
16  15  16  15  14  13  12  11  12  11  10  11  12  11  12  13  14  13  14  15
 [157]  16  17  16  17  18  17  16  15  14  13  12  11  10  11  12  13  12  11  10
9   8   9   8   9  10   9   8   9  10   9   8   7   8   9  10   9   8   7   8
...
...
```

You ma draw all this:

```
plot (x = steps.rw, type = 'l')
```

The figure is not here, because for each run the figure looks different. Argument x in the function `plot` contains values you want to draw. Theoretically, the arguments should be two, x and y, i.e. the values for x and y axis; if the y argument is missing, the function proceeds as following: the values of x argument are drawn on vertical axis, and horizontal axis is represented by the order of values in the vector.

🔧 Fix Me!

Definice a použití funkce

Pokaždé, když daný cyklus o tisíci krocích provedu znovu, dostanu jiný tvar mé náhodné chůze. Možná by se hodilo vidět, co se stane, když cyklus náhodné chůze zopakuji třeba 25 krát. Mohl bych jednoduše cyklus o 1000 krocích zanořit do dalšího cyklu, který by tentokrát definoval počet opakování celé procedury. Půjdu na to ale ještě jinak. Z celého skriptu, který mi vytvoří jednu náhodnou chůzi (o daném počtu kroků, které musím udělat od bodu nula) vytvořím funkci.

```
random.walk <- function (no.steps)
    {
    steps.rw <- vector (mode = 'numeric', length = no.steps)
    for (k in seq (2, no.steps))
      {
      steps.rw [k] <- steps.rw [k-1] + sample (c(-1,1),1)
      }
    }
```

Nově vytvořená funkce se jmenuje `random.walk` a vyžaduje jediný argument s názvem `no.steps`, tedy počet kroků. V prvním kroku funkce vytvoří prázdnou proměnnou steps.rw, která představuje numerický vektor o délce `no.steps`. Ve druhém kroku se tato proměnná začíná plnit hodnotami z cyklu, který jsme použili už před chvílí - cyklus začíná pozicí 2 a končí pozicí o hodnotě `no.steps`.

Zkusíme funkci použít:

```
random.walk (1000)
```

Pokud jsem někde neudělal chybu, funkce proběhne, ale nevrátí mi žádný výsledek. Proměnná `steps.rw` v rámci funkce skutečně vznikla, ale zůstala ve funkci “uvězněna”. Je potřeba, aby ji funkce “poslala ven”, což zařídím tak, že na konci definice funkce napíšu její jméno, stejně jako když se chci podívat na to, co proměnná obsahuje:

```
random.walk <- function (no.steps)
    {
    steps.rw <- vector (mode = 'numeric', length = no.steps)
    for (k in seq (2, no.steps))
      {
      steps.rw [k] <- steps.rw [k-1] + sample (c(-1,1),1)
      }
    steps.rw
    }
```

Tady jen malá odbočka: pokud nechci funkci definovat celou znovu, tak jak jsem to udělal nyní, mohu ji jen otevřít pro editaci příkazem `edit`:

```
random.walk <- edit (random.walk)
```

Editovanou verzi je třeba přiřadit znovu do proměnné stejného (případně jiného) jména, aby se mi zeditované změny uložily - pokud použiju pouze `edit (random.walk)`, po editaci se mě eRko sice zeptá na uložení změn, ale vlastní funkce se nezmění. Analogií je funkce `fix (random.walk)`, která se naopak používá jako taková a změny se automaticky uloží do definice původní funkce [1)].

Znovu se podívám, jak funkce pracuje (použiju menší počet kroků, aby výsledný vektor nebyl tak dlouhý pro výpis):

```
random.walk (100)

  [1]   0  -1   0   1   2   1   0  -1  -2  -1   0  -1   0  -1  -2  -3  -4  -5  -4
-5  -4  -3  -4  -5  -6  -5  -6  -7  -8  -9 -10  -9  -8  -9  -8  -7  -8  -9 -10
 [40] -11 -12 -13 -14 -15 -14 -13 -12 -11 -12 -11 -12 -13 -14 -13 -12 -13 -12 -13
-12 -11 -12 -13 -12 -11 -10 -11 -12 -11 -10  -9  -8  -9  -8  -9 -10 -11 -12 -13
 [79] -12 -13 -14 -13 -14 -13 -12 -13 -14 -13 -14 -13 -12 -13 -14 -13 -14 -13 -14
-15 -16 -15
```

Celé to můžu případně nakreslit:

```
plot (random.walk (1000), type = 'l')
```

A teď onen slíbený obrázek, ve kterém se mi nakreslí několik náhodných chůzí najednou. Udělám nejdříve úpravu grafického rozhraní (budeme mu říkat dejme tomu kreslící plátno) tak, že ho rozdělím na 25 polí (5 řádků, 5 sloupečků), což se provede nastavením parametru `mfrow`. Dále chci, aby vlastní obrázek neměl žádné okraje (*margins*), kam se normálně kreslí popisky os a jednotlivé hodnoty - to provedu nastavením parametru `mar` na samé nuly. O detailech a dalších grafických parametrech se dozvíte v nápovědě k funkci `par`.

```
par (mfrow = c (5,5))
par (mar = c (0,0,0,0))
```

A nyní vytvořím cyklus, který mi těch 25 obrázků postupně nakreslí:

```
for (i in seq (1, 25))
    {
    plot (random.walk (1000), type = 'l', ann = F, axes = F)
    box ()
    }
```

V tomto cyklu ve skutečnosti proměnnou `i` na nic nepotřebujeme, její definice nám pouze zajistí, že se cyklus zopakuje právě 25 krát. V kreslící funkci `plot` se objevily další dva argumenty: `ann = F` znamená, že se nebude vykreslovat annotace os (jejich názvy), `axes = F` zase zajistí, že se nebudou osy ani jednotlivé značky vůbec vykreslovat. Výsledný graf by pak ale "visel úplně ve vzduchu", proto je nakonec ještě doplněna funkce `box ()`, která kolem grafu nakreslí krabici (v místech, kde původně byly osy).

Souhrn skriptu pro random walk

```
set.seed (212121)  # pokud nastavíte seed stejné jako já, dostanete úplně stejný
obrázek

random.walk <- function (no.steps)
{
  steps.rw <- vector (mode = 'numeric', length = no.steps)
  for (k in seq (2, no.steps))
  {
```

```
    steps.rw [k] <- steps.rw [k-1] + sample (c(-1,1),1)
  }
  steps.rw
}


par (mfrow = c (5,5))
par (mar = c (0,0,0,0))

for (i in seq (1, 25))
{
  plot (random.walk (1000), type = 'l', ann = F, axes = F)
  box ()
}
```

Vlastní obrázek najdete zde.

Skript pro 3D random walk

```
library (rgl) # pokud neni knihovna nainstalovana, pouzijte install.packages
("rgl")

no.steps <- 10000
steps.rw <- matrix (0, ncol = 3, nrow = no.steps)
for (k in seq (2, no.steps))
  {
  steps.rw[k,] <- steps.rw[k-1,]
  which.to.add <- sample (1:3, 1)
  steps.rw[k, which.to.add] <- steps.rw[k, which.to.add] + sample (c (-1,1), 1)
  }

rgl.linestrips (steps.rw)

# pokud chcete proces kresleni animovat, pouzijte nasledujici cyklus:
# for (i in seq (2, no.steps)) rgl.linestrips (steps.rw[1:i,])
```

[1)]
Funkce `edit` a `fix` se dají s úspěchem použít i na editaci objektů typu `matrix` a `data.frame` - RStudio, pokud ho používáte, však editaci matic a datových rámců funkcí `edit` nepodporuje.

From:
https://anadat-r.davidzeleny.net/ - **Analysis of community ecology data in R**

Permanent link:
https://anadat-r.davidzeleny.net/doku.php/en:cycle_and_function

Last update: **2017/10/11 20:36**