Exercícios 3

Funções

Em **R**, as funções são também objectos e podem ser manipuladas como tal. Têm três principais propriedades:

argumentos Conjunto dos parâmetros que a função vai receber, em que alguns podem ter um valor por defeito e ser opcionais. Uma forma rápida de consultar os argumentos de uma função é através da função `args`.

corpo Ao conjunto de expressões avaliado na execução da função chama-se corpo da função. É possível visualizar o corpo de uma função através da função `body`.

ambiente Uma função é definida num determinado ambiente (*environment*) activo, o que irá determinar os objectos visíveis e acessíveis pela função, também conhecido como alcance (*scope*).

1. Considere o seguinte `data.frame` que contém uma pauta com as notas dos alunos de CEGI:

```
> dput(pauta)
structure(list(`Teste SAS` = c(14L, 14L, 13L, 15L), `Teste R` = c(13L,
19L, 10L, 12L), `Trabalho 1` = c(18L, 6L, 16L, 9L), `Trabalho 2` = c(11L,
14L, 12L, 9L)), .Names = c("Teste SAS", "Teste R", "Trabalho 1",
"Trabalho 2"), row.names = c("Rui", "Manuel", "Mariana", "Carolina"
), class = "data.frame")
```

   A seguinte função, usa uma estrutura iterativa para encontrar o nome dos alunos que tiveram positiva (≥ 10) em todas as componentes.

```
tudo_positivo = function (notas) {
        aluno = 1
        nomes = c()
        while (aluno <= nrow(notas)) {
                if (all(notas[aluno, ] >= 10)) {
                        nomes = c(nomes, rownames(notas)[aluno])
                }
                aluno = aluno + 1
        }
        return (nomes)
}
```

```
> tudo_positivo(pauta)
[1] "Rui"     "Mariana"
```

   O **R** é uma linguagem versátil e compatível com o paradigma de programação funcional, sendo possível resolver este exercício de uma forma mais curta e eficiente (além de elegante). Reescreva a função `tudo_positivo` sem usar uma estrutura iterativa.

   Nota: A função `dput()` é útil para recriar objectos definidos em R. Neste exemplo, o `data.frame` `pauta` estava guardado numa sessão em R, e com esta função é possível guardar uma expressão que permite recriar o mesmo objecto. Assim, para ter esta pauta na sua sessão, basta copiar o resultado da chamada da função `dput()`.

Solução:

```
tudo_positivo = function(notas) {
        rownames(notas)[apply(notas, 1, function(x) all(x >= 10))]
}
```

2. A função `mais_faceis` mostra as componentes em que mais alunos (por defeito, pelo menos 3 alunos) tiveram positiva. Esta função utiliza a coerção de valores lógicos para valores numéricos.

```
mais_faceis = function(notas, lim=3) {
        componentes = c()
        for (componente in 1:ncol(notas)) {
                if (sum(notas[, componente] >= 10) >= lim) {
                        componentes = c(componentes, (colnames(notas)[componente]))
                }
        }
        return (componentes)
}
```

```
> mais_faceis(pauta)
[1] "Teste SAS"  "Teste R"    "Trabalho 2"
```

Reescreva esta função usando o paradigma da programação funcional (sem estruturas iterativas).

Solução:

```
mais_faceis = function(notas, lim=3) {
        colnames(notas)[apply(notas, 2, function(x) sum(x >= 10) >= lim)]
}
```

3. O vector `rivers` contém o comprimento, em milhas, dos 141 principais rios norte americanos.

Escreva a função `perc`, que recebe dois argumentos: um vector numérico, `v` e um número, `x`. A função devolve o percentil de `v` correspondente ao valor `x`.

Por exemplo, considerando o vector `rivers`:

```
> perc(rivers, 680)
[1] 75.1773
> perc(rivers, 425)
[1] 50.35461
> perc(rivers, 10)
[1] 0
> perc(rivers, 10000)
[1] 100
```

Estes valores correspondem aos obtidos com a função `quantile`.

```
> quantile(rivers)
0%   25%   50%   75%   100%
135  310   425   680   3710
```

Adaptado do Exame de Época Especial de 2014/2015.

Solução:

```
perc = function (v, x) {
        return (length(subset(v, v <= x)) / length(v) * 100)
}
```

4. Atente ao seguinte bloco de código:

```
x = 2
z = 56.3
f = function(x) {
        a = 34
        y = x / 4 * a * z
        y
}
f(21)
a
```

Qual o valor das variáveis `a`, `x` e `z` dentro do ambiente da função? O que aconteceu ao valor da variável `a` após a execução da função?

Solução:

```
f = function(x) {
        a = 34
        y = x / 4 * a * z
        cat("Variável x: ", x, "\nVariável a: ", a, "\nVariável z: ", z, "\n")
        y
}

> x = 2
> z = 56.3
> f(21)
Variável x:  21
Variável a:  34
Variável z:  56.3
[1] 10049.55
```

Os valores visualizados correspondem aos valores que aquelas variáveis têm **dentro** da função:

- A variável `x` toma o valor passado como argumento da função. Não é o mesmo objecto que existe fora da função, embora tenha o mesmo nome (existem em ambientes diferentes).
- A variável `a` só existe dentro da função e tem o valor que recebe dentro desta.
- A variável `z` existe fora da função, mas quando acedida dentro da função, como não existe nenhum outro objecto com o mesmo nome, é procurado um objecto com esse nome no ambiente imediatamente acima do ambiente da função `f`.

5. Defina uma função para calcular o factorial (Equação 1) de um dado número natural.

$$\text{fact}(n) = \begin{cases} 1 & \text{se } n = 0 \\ n \times \text{fact}(n-1) & \text{se } n > 0 \end{cases} \tag{1}$$

> Recursion in computer science is a method where the solution to a problem depends on solutions to smaller instances of the same problem (as opposed to iteration).
>
> — Graham et al., 1990

Agora defina a mesma função mas utilizando uma definição recursiva.

Solução:

```
# definição iterativa
factIter = function(n) {
        f = 1
        for (i in 2:n) f = f * i
        f
}
# definição recursiva
fact = function(n) {
        if (n <= 1) 1
        else n * fact(n - 1)
}
```

6. Dado um vector `x` com números inteiros, construa um novo vector com a soma dos números iguais que existem em `x`.

   Por exemplo, no seguinte vector

   ```
   x = (1, 2, 5, 4, 3, 1, 1, 4, 2, 6, 7, 2, 4, 1, 5)
   ```

   a soma dos números 1 dá 4 (há quatro 1), a soma dos números 2 dá 6 (há três 2), a soma dos números 3 dá 3 (há um 3), etc., tal que o resultado esperado é o seguinte:

   ```
   [1]  4  6  3 12 10  6  7
   ```

Solução:

```
> sapply(sort(unique(x)), function(i) sum(x[x == i]))
[1]  4  6  3 12 10  6  7
```