Exercícios 1

1 Expressões

Uma expressão em **R** corresponde a uma sequência de operações, e que pode incluir chamadas de variáveis e de funções, que poderá ser avaliada, tendo um (e apenas um) retorno.

1. Avalie as seguintes expressões:

   - `123 * 321`
   - `42 ** 2`
   - `42 ^ 2`
   - `34 - 2 * 6`
   - `4 + 2 == 42`
   - `1 + sum(2, 3)`

   Solução:

```
> 123 * 123
[1] 15129
# o expoente pode ser obtido com os operadores ** e ^ (acento circunflexo)
> 42 ** 2
[1] 1764
> 42 ^ 2
[1] 1764
> 34 - 2 * 6
[1] 22
# exemplo com o operador lógico de igualdade
> 4 + 2 == 42
[1] FALSE
# uma expressão pode envolver a chamada de uma função
> 1 + sum(2, 3)
[1] 6
```

   Notas:

   - Para inserir um comentário no código, ou comentar uma instrução, basta colocar um cardinal `#` antes do texto pretendido;
   - O **R** respeita a precedência dos operadores aritméticos, bem como de outros operadores próprios, pelo que deve-se ter cuidado com a ordem das operações, e usar parêntesis caso seja necessário;

2. Guarde o resultado de uma expressão numa variável.

Solução:

```
> (w = 1 + sum(2, 3))
[1] 6
```

Notas:

- Em **R**, existem três operadores de atribuição: `=`, `<-` e `->`, pode-se usar qualquer um deles, mas convém ser consistente na sua utilização durante a escrita do código;
- Neste exemplo, como a variável está colocada à esquerda, só se poderia utilizar os dois primeiros operadores de atribuição;
- Colocando uma expressão de atribuição entre parêntesis é uma forma rápida de guardar o resultado e visualizá-lo de seguida, sem ter de se escrever o nome da variável.

3. Use essa variável numa outra expressão.

Solução:

```
> x = w + 2
> x
[1] 8
```

Guardar o resultado de uma expressão, que pode ser complicada, numa variável auxiliar, pode ser uma forma de evitar repetição de código e de o tornar mais legível.

4. Atribua, à mesma variável, o número 42. O que aconteceu ao resultado da expressão que havia sido lá guardado?

Solução:

```
> w = 42
> w
[1] 42
```

A operação de atribuição é destrutiva, isto é, se for atribuído um novo valor a uma variável existente, o valor anterior deixa de ser acessível.

2 Vectores – Parte 1

Vectores constituem uma forma simples, mas poderosa, de armazenar informação em **R**, mas apenas de um só tipo. Considere a pauta mostrada na Tabela 1.

1. Guarde a tabela em dois vectores, `nomes` e `notas`. Atente à nota em falta. O que acontecerá se colocar todos os dados num só vector?

Tabela 1: Pauta de CEGI

Nome	Nota
André	10
Carolina	12
João	–
Mariana	15
Pedro	16
Vânia	13
Zé	7

Solução:

```
nomes = c("André", "Carolina", "João", "Mariana", "Pedro", "Vânia", "Zé")
notas = c(10, 12, NA, 15, 16, 13, 7)
```

Os dados em falta devem ser representados pelo valor especial `NA`.

Se colocar todos os dados num só vector, terá duas desvantagens: *a*) torna-se mais difícil aceder aos seus elementos, pois perde-se a *individualidade* das colunas da tabela; e *b*) o tipo de objecto dos valores correspondentes às notas é automaticamente convertido de `numeric` para `character`, uma vez que um vector apenas suporta um único tipo de objecto de cada vez.

2. Verifique, com um operador lógico, se os dois vectores têm o mesmo tamanho.

Solução:

```
> length(nomes) == length(notas)
[1] TRUE
```

3. Aceda ao nome e à nota do **5º** aluno.

Solução:

```
> nomes[5]
[1] "Pedro"
> notas[5]
[1] 16
```

4. Corrigiu-se o teste do João, ele teve 8. Atribua-lhe, no vector `notas`, a sua cotação.

Solução:

Sabendo que o João está colocado na 3ª posição dos vector `notas`, basta aceder ao elemento do vector nessa posição.

```
notas[3] = 8
> notas
[1] 10 12  8 15 16 13  7
```

Alternativamente, é possível alterar o vector **notas** sem saber qual a posição correspondente à nota do João, tirando essa informação a partir do vector **nomes**, com uma operação booleana. Atenção que esta solução só é válida se não existirem dois alunos com o mesmo nome.

```
notas[nomes == "João"] = 8
```

Outra solução, mais genérica, pode ser obtida recorrendo à função `is.na`. Esta função devolve o valor lógico `TRUE` se o seu argumento for `NA`. Aplicada num vector, o seu retorno é também um vector.

```
notas[is.na(notas)] = 8
```

Esta solução tem a vantagem de encontrar um qualquer número de alunos sem nota atribuída. Se o objectivo fosse apenas saber em que posições do vector **notas** ocorrem valores em falta, `NA`, a solução passaria por usar a função `which`.

```
> which(is.na(notas))
[1] 3
```

5. Calcule a média das notas.

Solução:

```
> mean(notas)
[1] 11.57143
```

6. O docente responsável decidiu atribuir bónus aos alunos. Some uma cotação de bonificação a cada nota.

Solução:

Supondo que o valor de bónus é de 1 valor, e que será atribuído a todos os alunos:

```
notas = notas + 1
> notas
[1] 11 13  9 16 17 14  8
```

O valor 1 foi somado a cada um dos elementos do vector **notas**.

Imagine, agora, que o valor de bónus pode ser de 1 ou 0,5 valores. Desprezando a forma como é decidida a atribuição de uma destas hipóteses aos alunos, uma solução seria:

```
> notas + c(1, 0.5)
[1] 11.0 12.5  9.0 15.5 17.0 13.5  8.0
Warning message:
In notas + c(1, 0.5) :
  longer object length is not a multiple of shorter object length
```

Repare que os elementos do vector `c(1, 0.5)` foram repetidos tantas vezes quantas as necessárias, até serem percorridas todos os elementos do vector `notas`. A mensagem de aviso dada pelo **R** informa que os dois vectores, `notas` e `c(1, 0.5)`, não têm comprimentos múltiplos (o primeiro tem 7 elementos, o outro tem apenas 2). Por esse motivo, o segundo elemento do vector menor foi repetido menos uma vez.

Estas duas soluções mostram, assim, exemplos da *regra da reciclagem*.

3 Sequências

Outra forma fácil e útil de criar vectores é através de sequências. Estas podem ser definidas ou até serem aleatórias.

1. Crie as seguintes sequências:
   - de 1 a 20
   - de 1 a 20, mas de 2 em 2
   - 15 elementos de 1 a 20
   - 20 elementos a começar em 1, espaçados por 0,4
   - o número 1 repetido 100 vezes

Solução:

```
> # de 1 a 20
> 1:20
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
> # de 1 a 20, mas de 2 em 2
> seq(1, 20, 2)
 [1]  1  3  5  7  9 11 13 15 17 19
# 15 elementos de 1 a 20
> seq(1, 20, length=15)
 [1]  1.000000  2.357143  3.714286  5.071429  6.428571  7.785714  9.142857
 [8] 10.500000 11.857143 13.214286 14.571429 15.928571 17.285714 18.642857
[15] 20.000000
# 20 elementos a começar em 1, espaçados por 0,4
> seq(from=1, length=20, by=0.4)
 [1] 1.0 1.4 1.8 2.2 2.6 3.0 3.4 3.8 4.2 4.6 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2
[20] 8.6
# o número 1 repetido 100 vezes
> rep(1, 100)
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Para uma sequência simples, com números inteiros consecutivos, em ordem crescente ou decrescente, basta usar o operador `:`.

Para uma maior manipulação da sequência, a função `seq` é mais indicada. Esta função admite vários argumentos, sendo os mais importantes: **from** (a partir de), **to** (até), **by** (incremento), e **length** (número de elementos).

A função `rep` também permite criar uma sequência a partir da repetição de outros objectos de tipo básico, além do tipo `numeric` (e.g., `character`).

2. Em **R**, é fácil criar sequências aleatórias, partindo de diferentes distribuições estatísticas. As funções que criam essas sequências têm a forma ***rfunc***, por exemplo, `rnorm`, `runif` e `rt`. Crie as seguintes sequências aleatórias:

   - 10 elementos de uma distribuição normal padrão
   - 10 elementos de uma distribuição normal com média 13 e desvio padrão 2
   - 100 elementos de uma distribuição uniforme entre 1 e 20
   - 5 elementos de uma distribuição *t-student* com 10 graus de liberdade

Solução:

```
> rnorm(10)
[1]  0.3893928 -0.8923915  1.4273738 -0.8857474  0.9354829  1.3247520
[7] -0.8557412 -0.2787961 -1.2497442  0.6856208
> rnorm(10, mean=13, sd=2)
[1] 12.81522 13.64655 13.47899 10.56444 15.53645 13.72218 11.47630 12.50179
[9] 14.89978 12.98178
> runif(100, 1, 20)
[1] 12.890452 13.809040  1.845204  3.036405 10.317447  2.687971 18.951556
[8] 10.019654  3.862149 18.049652 17.856375  2.917227 12.797862 13.444384
[15]  1.453109  6.730706  9.319316 14.636486 12.255608  1.659550  1.907945
[22] 13.974350  6.655288  3.489309 16.469166  2.655856  6.802787 18.126680
[29] 12.420356 15.783056  9.551025 18.970067  2.717150 16.061868 12.821729
[36] 14.888721  7.398816 19.788180  7.951151 16.325348  2.979728 18.666736
[43]  6.958047 12.265295 14.980399 13.104052  3.292474 19.787362 12.883868
[50]  3.825523 12.087716 15.161680 17.281745 17.906548 13.996133 10.566860
[57] 10.671736 13.204635  7.980697 15.047350  5.762803  9.393275  8.774348
[64]  9.017975 14.615049 16.550216 12.790073 16.638168  5.216438 11.186398
[71] 13.895636 11.503153 13.032967 14.990417 16.357368  1.553141 18.481447
[78]  7.045387  8.081076 18.434614 17.275692 19.368941  3.098027  3.095955
[85] 12.725635 17.716566 16.419147  2.190231 18.240423  7.308344 12.962402
[92]  2.592744  5.913642  6.257219 12.704005  9.532116  2.970483 12.454670
[99]  9.860412 17.268342
> rt(5, df=10)
[1]  1.5456106 -0.9824119  1.9551162  0.7600165 -0.7377210
```

Notas:

- Por defeito, a função `rnorm` já devolve elementos tirados aleatoriamente de uma função normal padrão, tal como se pode verificar com a instrução `args(rnorm)`;
- Estas funções devolvem sequências aleatórias e, como tal, é esperado que a sequência obtida em diferentes computadores, e/ou em diferentes instantes no tempo, seja diferente;
- Quando se avalia uma expressão na consola do **R**, a linha com o respectivo resultado, no caso de se tratar de um vector, começa com `[1]`. Esse número informa que se estão a visualizar

os elementos pertencentes ao vector resultante, começando no primeiro. Noutro exemplo, no retorno da instrução acima `runif(100, 1, 20)`, pode-se facilmente perceber qual é o elemento na 74ª posição, bastando para tal localizar a linha que começa com `[71]` e contar 4 elementos para a direita, encontrando, assim, o valor `14.990417`.

4 Vectores – Parte 2

As sequências são uma forma útil de aceder, em simultâneo, a vários elementos de um vector. Tire proveito da indexação **com** vectores para responder às seguintes questões.

1. Obtenha as notas dos primeiros 3 alunos.

Solução:

```
> notas[1:3]
[1] 10 12  8
```

2. Liste os nomes de todos os alunos excepto dos últimos 3.

Solução:

```
> nomes[-(5:7)]
[1] "André"    "Carolina" "João"     "Mariana"
> nomes[-((length(nomes)-2):length(nomes))]
[1] "André"    "Carolina" "João"     "Mariana"
```

As duas soluções dão o mesmo resultado, mas a segunda tem a vantagem de funcionar independentemente do número de elementos no vector `notas`, enquanto que a primeira solução só funciona quando este vector tem 7 elementos.

3. Quais são os alunos que tiveram positiva?

Solução:

```
> nomes[notas >= 10]
[1] "André"    "Carolina" "Mariana"  "Pedro"    "Vânia"
```

4. Que notas houve entre 10 e 13?

Solução:

```
> notas[notas >= 10 & notas <= 13]
[1] 10 12 13
```

5. Descubra se há alunos muito diferentes, e quais os seus nomes, verificando se houve notas menores que 8 e maiores que 15.

Solução:

```
> notas[notas < 8 | notas > 15]
[1] 16  7
> nomes[notas < 8 | notas > 15]
[1] "Pedro" "Zé"
```

6. Devolva as notas de 5 alunos escolhidos aleatoriamente.

Solução:

```
> notas[runif(5, 1, length(notas))]
[1] 13 12 16 16 16
```

A função `runif` devolve números reais, no entanto, durante a indexação de um vector, o **R** aceita e converte automaticamente números reais em números inteiros. Para a conversão ser explícita, pode-se usar a função `as.integer`:

```
notas[as.integer(runif(5, 1, length(notas)))]
```

Caso também se pretenda obter os nomes dos alunos, é necessário guardar os índices numa variável auxiliar, caso contrário, por serem escolhidos aleatoriamente, os nomes e as notas não corresponderão aos dados na Tabela 1:

```
> (indices.amostra = as.integer(runif(5, 1, length(notas))))
[1] 3 1 2 6 1
> notas[indices.amostra]
[1]  8 10 12 13 10
> nomes[indices.amostra]
[1] "João"     "André"    "Carolina" "Vânia"    "André"
```

5 Matrizes

Uma matriz, em **R**, é a extensão de um vector para duas dimensões.

1. Crie um novo vector, `idades`, com as idades dos alunos.

Solução:

Pode optar por criar o vector idades com um conjunto de quaisquer valores. Alternativamente, pode dar uso às funções que criam sequências aleatórias para criar um conjunto de dados fictício. Por exemplo, admitindo que a idade dos alunos segue uma distribuição normal, com média 20 e desvio padrão 2:

```
> idades = as.integer(rnorm(n=length(nomes), mean=20, sd=2))
> idades
[1] 22 20 18 17 20 19 19
```

2. Construa uma matriz, `pauta`, a partir dos vectores `notas` e `idades`.

Solução:

```
> pauta = cbind(notas, idades)
> pauta
notas idades
[1,]    10     22
[2,]    12     20
[3,]     8     18
[4,]    15     17
[5,]    16     20
[6,]    13     19
[7,]     7     19
```

3. Use o vector `nomes` para atribuir etiquetas às linhas da matriz `pauta`.

Solução:

```
> rownames(pauta) = nomes
> pauta
notas idades
André       10     22
Carolina    12     20
João         8     18
Mariana     15     17
Pedro       16     20
Vânia       13     19
Zé           7     19
```

4. Troque as linhas pelas colunas dessa matriz.

Solução:

```
> t(pauta)
André Carolina João Mariana Pedro Vânia Zé
notas     10       12    8      15    16    13  7
idades    22       20   18      17    20    19 19
```