Lista Prática 2

Exercício 1 (`swirl`). *Usando o pacote* `swirl`*, acesse o curso "R Programming" e faça as seguintes lições:*

a) 10: lapply and sapply

b) 12: Looking at Data

c) 13: Simulation

Adicionalmente, instale o curso "Getting and Cleaning Data" usando a seguinte função `swirl::install_course("Getting and Cleaning Data")` *e faça as seguintes lições:*

d) 1: Manipulating Data with dplyr

e) 2: Grouping and Chaining with dplyr

Ao instalar o curso e entrar na lição "1: Manipulating Data with dplyr" aparecerá o erro:

```
Error in yaml.load(readLines(con, warn = readLines.warn), error.label = erro
r.label,  :
  (C:/Users/fhnis_r40seqz/AppData/Local/Programs/R/R-4.3.0/library/swirl/Cour
ses/Getting_and_Cleaning_Data/Manipulating_Data_with_dplyr/lesson.yaml) Scann
er error: while scanning a tag at line 205, column 9 did not find expected wh
itespace or line break at line 205, column 19
```

Copie o endereço até a pasta "Manipulating_ Data_ with_ dplyr" (destacado na imagem acima) e substitua o arquivo "lesson.yaml" pelo arquivo disponível em:
https://fhnishida.netlify.app/project/rec5004/lesson.yaml

Por fim, use `swirl::install_course("Exploratory Data Analysis")` *para instalar o curso "Exploratory Data Analysis" e faça as seguintes lições:*

f) 5: Base Plotting System

g) 9: GGPlot2 Part2[1]

Exercício 2. *A base de dados deste exercício foi obtida do site Hospital Compare (*`http://hospitalcompare.hhs.gov`*) administrado pelo Departamento de Saúde e Serviços Humanos dos EUA. O propósito deste site é prover dados e informação sobre a qualidade dos tratamentos de mais de 4 mil hospitais certificados pelo Medicare. O arquivo que utilizaremos é o* `medicare.csv`[2]*, que contém informação sobre taxa de mortalidade de 30 dias para alguns problemas de saúde.*

Comece carregando a base de dados com o seguinte código e observe a sua estrutura:

[1] O 1º exercício irá pedir para fazer um gráfico usando `qplot()` que é uma função pouco utilizada. Você pode dar `skip()` para avançar para a parte com gráficos usando a função `ggplot()`

[2] Versão resumida da base de dados fornecida no curso da John Hopkins no Coursera

```r
outcome = read.csv2("https://fhnishida.netlify.app/project/rec5004/medicare.csv")
```

Como exercício, escreva a função `best()` *que terá dois argumentos: (I) a sigla de um Estado americano, e (II) o nome do problema de saúde. Essa função, a partir base de dados* `outcome`*, retorna um vetor de texto com o nome do hospital que possui a melhor (menor) taxa de mortalidade (Death Rate), dadas sigla de Estado e problema de saúde. O nome do hospital está na coluna "Hospital.Name" e os problemas de saúde podem ser "heart attack", "heart failure" e "pneumonia". Se houver empate de melhor hospital, o nome do hospital a ser retornado pela função é dado pela ordem alfabética – se "Hospital A" e "Hospital B" estiverem empatados, deve-se retornar "Hospital A". A função deve ter a seguinte estrutura:*

```r
best = function(estado, problema) {
    # Filtre a base de dados pela sigla do estado

    # Reordene a coluna taxa de mortalidade da doença selecionada de forma
    crescente e, em caso de empate, coloque os nomes dos hospitais de forma alfabé
    tica.

    # Retorne o nome do hospital com a menor taxa de mortalidade
    }
```

Como referência, seguem alguns exemplos de output da função:

```r
> best("TX", "heart failure")
[1] "FORT DUNCAN MEDICAL CENTER"

> best("MD", "heart attack")
[1] "JOHNS HOPKINS HOSPITAL, THE"
```

Usando a função `best()` *escrita, quais são os resultados retornados para os casos:*

a) `estado = "SC"` *e* `problema = "heart attack"`

b) `estado = "NY"` *e* `problema = "pneumonia"`

c) `estado = "AK"` *e* `problema = "pneumonia"`

Resposta: A ideia aqui é reordenar a base de dados, de forma que o hospital desejado fique na primeira linha (e seu nome será retornado pela função `best()`). Um das dificuldades é que precisamos reordenar uma das três colunas da taxa de mortalidade, além do nome do hospital. Uma forma de fazer isso é criando uma estrutura condicional para cada uma das doenças (*heart attack*, *heart failure* e *pneumonia*) e reordenando via função `arrange()`. Usando esta função, precisamos informar a coluna de taxa de mortalidade da doença incluída no argumento `problema` e, depois, o nome do hospital (para caso haja empate de nas taxas de mortalidade).

```r
best = function(estado, problema) {
    # Filtre a base de dados pela sigla do estado
    bd = outcome %>% filter(State == estado)

    # Reordene a coluna de taxa de mortalidade da doença selecionada
    # de forma crescente e, em caso de empate, coloque os nomes dos
    # hospitais de forma alfabética.
    if (problema == "heart attack") {
        bd = bd %>% arrange(DeathRatesHeartAttack, Hospital.Name)
    } else if (problema == "heart failure") {
```

```r
        bd = bd %>% arrange(DeathRatesHeartFailure, Hospital.Name)
    } else if (problema == "pneumonia") {
        bd = bd %>% arrange(DeathRatesPneumonia, Hospital.Name)
    }

    # Retorne o nome do hospital com a menor taxa de mortalidade
    bd$Hospital.Name[1]
}
```

a) "MUSC MEDICAL CENTER"

b) "MAIMONIDES MEDICAL CENTER"

c) "YUKON KUSKOKWIM DELTA REG HOSPITAL"

□

Exercício 3. *Carregue bases de dados de GDP para 190 países ranqueados e de informações educacionais a partir do seguinte código:*

```r
gdp = read.csv2("https://fhnishida.netlify.app/project/rec5004/GDP.csv")
educ = read.csv2("https://fhnishida.netlify.app/project/rec5004/EDSTATS.csv")
```

a) Mescle as duas bases de dados a partir do código de país de 3 dígitos. Quantos códigos tiveram correspondência em ambas bases? Organize a base de dados mesclada de forma decrescente no GDP (de modo que USA fica na última linha). Qual é o país na 13ª linha?

b) Quais são os rankings médios para os grupos de renda "High income: OECD" e "High income: nonOECD"?

c) Divida os países em 5 grupos de (quase) mesmo tamanho de acordo com seus GDPs, criando a variável "GDP.Group". Depois, faça uma tabela cruzada entre ela e "Income.Group". Quantos países estão entre as nações com maior renda (High: OECD/nonOECD) e estão entre o 1º e 2º quintis do GDP?

Resposta:

a) 189 correspondências e 13º país é "St. Kitts and Nevis". Uma forma de saber a quantidade de correspondências ao mesclar duas bases é fazendo um *inner join* (junção que mantém apenas valores presentes em ambas bases) e contando a quantidade linhas restantes.

```r
library(dplyr)
## JUNTANDO BASES E MANTENDO APENAS CASOS QUE APARECEM EM AMBAS
joined = merge(gdp, educ, by.x="Country", by.y="CountryCode", all=FALSE)
nrow(joined) # núm. de correspondencias entre as duas bases (já que ALL=FALSE)
```

```
[1] 189
```

```r
# Ordenando GDP de forma crescente
joined = joined %>% arrange(GDP)
joined[13, 1:4] # 13o país na ordem decrescente (e 4 primeiras colunas)
```

```
  CountryCode Rank          Long.Name.x GDP
           KNA  178 St. Kitts and Nevis 767
```

b) 32.9 e 91.9. Podemos filtrar a base de dados para cada um dos grupos e fazer as médias, mas há uma forma mais "elegante" usando `group_by()` e `summarise()`. Podemos agrupar a base de dados pela variável categórica `Income.Group` e fazer as médias de rank:

```
## AVERAGE RANKING
joined %>% group_by(Income.Group) %>% summarise(
  mean_rank = mean(Ranking)
)
```

```
  Income.Group          mean_rank
1 High income: OECD          33.0
2 High income: nonOECD       91.9
3 Low income                134.
4 Lower middle income       108.
5 Upper middle income        92.1
```

c) 5 (4 nonOECD + 1 OECD). Precisamos criar a variável `Group.GDP` que é uma transformação de variável contínua em 5 categorias de tamanhos (quase) iguais. Para isto, vamos usar as funções `cut()` e `quantile()`:

- `cut()`: classifica uma variável contínua em categorias, porém é necessário informar um vetor de valores que determinam seus intervalos (*breaks*). A princípio, não sabemos os valores de GDP que dividem a amostra em 5 grupos de tamanhos (quase) iguais, então precisamos usar a função abaixo.
- `quantile()`: Precisamos encontrar os valores de GDP que correspondem ao mínimo, aos 4 quintis e ao máximo, que correspondem aos quantis de 0%, 20%, 40%, 60%, 80% e 100%, respectivamente. Logo, precisamos informar na função `quantile()`, o vetor GDP e o vetor de quantis $\{0\%, 20\%, 40\%, 60\%, 80\%, 100\%\}$.

```
breaks = quantile(joined$GDP, seq(0, 1, 0.20))
breaks
```

```
        0%        20%        40%        60%        80%       100%
      40.0     4248.4    15988.4    50824.4   262691.0 16244600.0
```

Note que obtivemos os valores de GDP mínimo (0%), 1º quintil (20%), 2º quintil (40%), 3º quintil (60%), 4º quintil (80%) e máximo (100%), respectivamente. Agora, vamos usá-los para classificar cada observação da base de dados em cada um dos 5 intervalos, criados por este 6 números que obtivemos pela função `quantile()`:

```
joined = joined %>% mutate(
    GDP.Group = cut(GDP, breaks)
)
class(joined$GDP.Group) # Factor
levels(joined$GDP.Group) # Categorias de forma crescente
```

```
[1] "factor"

[1] "(40,4.25e+03]"         "(4.25e+03,1.6e+04]"    "(1.6e+04,5.08e+04]"
[4] "(5.08e+04,2.63e+05]"   "(2.63e+05,1.62e+07]"
```

Note que `e+N` significa $\times 10^N$ e que criamos 5 categorias, cujos intervalos são abertos à esquerda. Vamos visualizar as 6 primeiras observações das variáveis GDP e GDP.Group:

```
head(joined[,c("GDP", "GDP.Group")])
```

```
  GDP      GDP.Group
1  40           <NA>
2 175 (40,4.25e+03]
3 182 (40,4.25e+03]
4 228 (40,4.25e+03]
5 263 (40,4.25e+03]
6 326 (40,4.25e+03]
```

Como a base foi reordenada crescentemente, as seis primeiras observações foram classificadas no primeiro intervalo, exceto o país com GDP mínimo, pois o intervalo é aberto à esquerda (dá para incluí-la usando `cut(..., include.lowest = TRUE)`). Agora, podemos fazer uma tabela cruzada de grupos de GDP e de Renda usando `table()`:

```
table(joined$GDP.Group, joined$Income.Group)
```

```
                       High: nonOECD High: OECD Low Lower middle Upper middle
  (40,4.25e+03]                    2          0  11           15            9
  (4.25e+03,1.6e+04]               4          1  16            9            8
  (1.6e+04,5.08e+04]               8          1   9           11            8
  (5.08e+04,2.63e+05]              5         10   1           13            9
  (2.63e+05,1.62e+07]              4         18   0            5           11
```

Note que cada linha (grupo de GDP) possui 37 ou 38 países, já que dividimos usando quintis. Para a resposta do item, precisamos verificar as colunas com renda alta (*High*) e na segunda linha (2º grupo com menor GDP – entre 1º e 2º quintis).

□

Exercício 4. *Considere a base de dados* `mtcars` *(nativa no R) que contém o consumo de combustível e mais 10 características automobilísticas para 32 carros (modelos 1973-74). Em particular, queremos analisar a relação entre o consumo de combustível (mpg, em milhas por galão) e o peso do automóvel (wt, em mil libras) pelo seguinte modelo:*

$$mpg_i = \alpha + \beta wt_i + \varepsilon_i$$

a) *Assuma* $\hat{\alpha} = 36$ *e* $\hat{\beta} = -4$ *e calcule a soma dos quadrados dos resíduos* $\sum_{i=1}^{N} \hat{\varepsilon}_i^2$*, tal que um resíduo é dado por:* $\hat{\varepsilon}_i = mpg_i - \widehat{mpg}_i$.

b) *Crie uma função* `resid_quad(alpha, beta)` *que recebe como inputs possíveis valores de* $\hat{\alpha}$ *e* $\hat{\beta}$*, e retorna a soma dos quadrados dos resíduos do modelo acima a partir da base de dados* `mtcars`.

c) *Considere os seguintes vetores de possíveis valores de* $\hat{\alpha}$ *e de* $\hat{\beta}$*:*
`alpha_grid = seq(34, 38, length=11)` *e* `beta_grid = seq(-6, -2, length=11)`
Crie uma matriz de dimensão 11×11 *em que cada linha corresponde a um valor de* $\hat{\alpha}$ *e cada coluna corresponde a um valor de* $\hat{\beta}$*. Preencha essa matriz usando a função* `resid_quad()` *e verifique qual par* $(\hat{\alpha}, \hat{\beta})$ *minimiza a soma dos quadrados dos resíduos do modelo acima.*

Resposta:

a) Encontramos os valores ajustados usando $\widehat{mpg}_i = \hat{\alpha} - \hat{\beta} wt_i = 36 - 4wt_i$:

```
mpg_ajustado = (36 + -4 * mtcars$wt)
```

Os resíduos quadráticos são calculados pela diferença entre os valores ajustados com os valores observados de *mpg*, elevando ao quadrado.

```
resid = mtcars$mpg - mpg_ajustado
sum(resid^2) # soma dos quadrados dos resíduos
```

```
[1] 627.7735
```

b) A função segue do código feito em (a), mas com input de um `alpha` e um `beta`:

```
resid_quad = function(alpha, beta) {
    mpg_ajustado = alpha + beta * mtcars$wt
    resid = mtcars$mpg - mpg_ajustado
    sum(resid^2)
}

resid_quad(36, -4) # Testando para os valores do item (a)
```

```
[1] 627.7735
```

c) Aqui, vamos criar uma matrix 11×11 em que cada linha corresponde a um valor de `alpha_grid` e cada coluna a um valor de `beta_grid`. Aplicaremos a função criada `resid_quad()` para preencher a matriz, considerando cada combinação de valores dos grids:

```
# Definindo os possíveis valores de alpha e beta
alpha_grid = seq(34, 38, length=11)
beta_grid = seq(-6, -2, length=11)

# Criando matriz de zeros com dimensão 11 x 11
matriz = matrix(0, nrow=length(alpha_grid), ncol=length(beta_grid))

# Preenchendo a matriz com soma dos quadrados dos resíduos , dadas todas possí
    veis combinações de valores de alpha e de beta
for (i in 1:length(alpha_grid)) {
    for (j in 1:length(beta_grid)) {
        matriz[i, j] = resid_quad(alpha_grid[i], beta_grid[j])
    }
}

round(matriz, 1) # Visualizando matriz (com 1 casa decimal)
min(matriz) # Menor soma de resíduos quadráticos
```

```
       [,1]  [,2]  [,3]  [,4]  [,5]   [,6]   [,7]   [,8]   [,9]  [,10]  [,11]
 [1,] 1222.2 820.1 533.5 362.4 306.7  366.6  542.0  832.8 1239.1 1760.9 2398.3
 [2,] 1089.2 720.1 466.4 328.2 305.5  398.4  606.7  930.4 1369.7 1924.5 2594.7
 [3,]  966.5 630.3 409.6 304.3 314.6  440.4  681.6 1038.3 1510.5 2098.3 2801.4
 [4,]  854.0 550.7 363.0 290.7 333.9  492.6  766.8 1156.4 1661.6 2282.3 3018.4
 [5,]  751.7 481.4 326.6 287.3 363.4  555.1  862.2 1284.8 1822.9 2476.5 3245.6
 [6,]  659.7 422.4 300.5 294.1 403.2  627.8  967.8 1423.4 1994.5 2681.0 3483.0
 [7,]  577.9 373.5 284.6 311.2 453.2  710.7 1083.7 1572.3 2176.3 2895.7 3730.7
```

```
 [8,]  506.4 334.9 279.0 338.5 513.4  803.9 1209.9 1731.3 2368.3 3120.7 3988.6
 [9,]  445.1 306.6 283.6 376.0 583.9  907.4 1346.3 1900.7 2570.5 3355.9 4256.8
[10,]  394.1 288.5 298.4 423.8 664.7 1021.0 1492.9 2080.2 2783.1 3601.4 4535.2
[11,]  353.3 280.6 323.5 481.8 755.6 1144.9 1649.7 2270.0 3005.8 3857.1 4823.8

[1] 278.9546
```

Porém, não queremos saber o menor valor de soma dos quadrados dos resíduos, mas sim os $\hat{\alpha}$ e $\hat{\beta}$ que geraram esse valor. Para isto, vamos usar a função `which(..., arr.ind=TRUE)` (argumento `arr.ind=TRUE` faz retornar os índices de cada dimensão da matriz/array):

```r
# Índices de alpha e de beta que minimizam desvio quadrático
indices = which(matriz == min(matriz), arr.ind = TRUE)
indices
```

```
     row col
[1,]   8   3
```

Agora, vamos extrair os índices de `alpha` e de `beta` do objeto `indice`, que serão utilizados para retornar os seus valores:

```r
# Atribuindo os índices a objetos
i_alpha = indices[1, "row"]
i_beta = indices[1, "col"]

# Buscando valores de alpha e de beta a partir dos índices acima
alpha_grid[i_alpha]
beta_grid[i_beta]
```

```
[1] 36.8
[1] -5.2
```

Logo, considerando os valores disponíveis nos grids, a combinação $\hat{\alpha} = 36.8$ e $\hat{\beta} = -5.2$ geraram a menor soma dos quadrados dos resíduos.

□

Exercício 5. *Faça uma amostragem de 100 números sem reposição do vetor de números inteiros `1:100`. Usando estruturas condicionais e de repetição/loop, reordene o vetor de forma decrescente.*

a) Crie um algoritmo usando os seguintes passos:

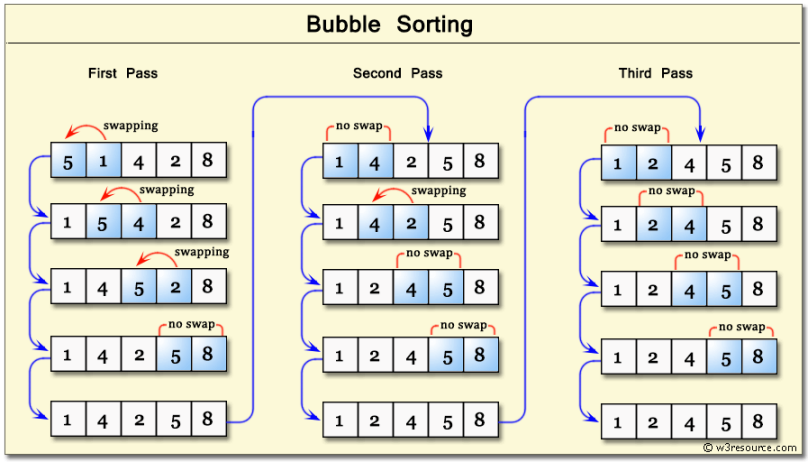
1. *Compare o número da 1ª posição com o da 2ª e, caso este seja maior, troque a ordem.*
2. *Compare o número da 2ª posição com o da 3ª e, caso este seja maior, troque a ordem.*

⋮

99. *Compare o número da 99ª posição com o da 100ª e, caso este seja maior, troque a ordem.*

b) Os passos executados no item anterior foram suficientes? Adapte seu algoritmo para que o vetor esteja totalmente ordenado decrescentemente.

Resposta: O procedimento descrito no enunciado é o algoritmo de ordenação em bolha (de forma crescente no exemplo abaixo):



Primeiro, crie o vetor descrito no enunciado:

```
vetor = sample(1:100, 100)
print(vetor)
```

```
  [1]  80   9  20  95  24  43  65  72  29  88  47  71  56  69  37  54  79
 [18]  16   3  84  67  12  28  83   6  50  25  86  70  30  22 100  36  61
 [35]  51  45  10  98  14  26  89  55  48  21  44  66  64  94  15  49  40
 [52]  96  97   8   5  53  68  35  18  34  52  32  62   7  60  58  93  87
 [69]  38  77  82  46  23   4  27  31  11  19  99  13  91   2   1  81  57
 [86]  85  17  63  41  76  59  74  42  90  39  78  92  73  33  75
```

Iniciando no 1º elemento do vetor, faremos uma comparação entre este elemento e o seguinte. Se este for maior do que aquele, inverteremos a ordem e, caso o elemento seja seguido por um número menor, a ordem continuará. Logo, faremos a mesma análise para a próxima dupla do vetor e repetiremos até chegar ao penúltimo elemento do vetor, que será comparado ao último.

```
for (i in 1:(length(vetor) - 1)) {
        if (vetor[i] < vetor[i + 1]) {
            aux = vetor[i + 1]
            vetor[i + 1] = vetor[i]
            vetor[i] = aux
        }
    }
```

```
  [1]  80  20  95  24  43  65  72  29  88  47  71  56  69  37  54  79  16
 [18]   9  84  67  12  28  83   6  50  25  86  70  30  22 100  36  61  51
 [35]  45  10  98  14  26  89  55  48  21  44  66  64  94  15  49  40  96
 [52]  97   8   5  53  68  35  18  34  52  32  62   7  60  58  93  87  38
 [69]  77  82  46  23   4  27  31  11  19  99  13  91   3   2  81  57  85
 [86]  17  63  41  76  59  74  42  90  39  78  92  73  33  75   1
```

Note que este loop fez apenas o menor número ir à última posição do vetor. Precisamos rodar outros loops para levar o 2º menor número para a penúltima posição, e assim por diante.

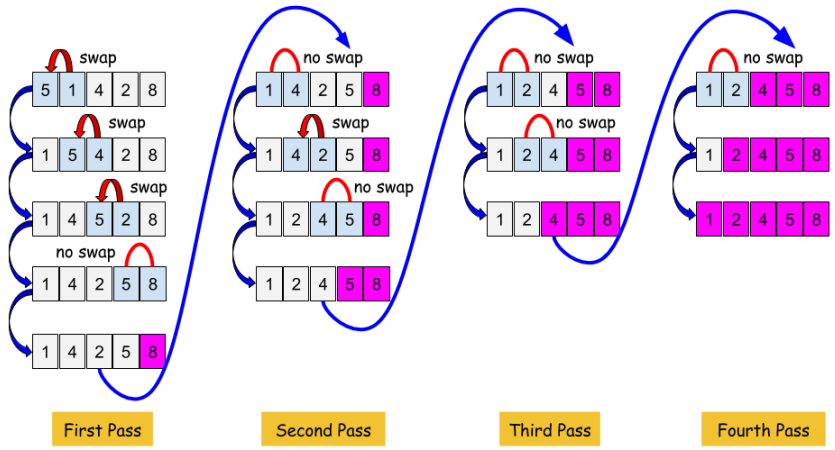
```
for (num_loops in 0:99) {
    for (i in 1:99) {
        if (vetor[i] < vetor[i + 1]) {
            aux = vetor[i + 1]
            vetor[i + 1] = vetor[i]
            vetor[i] = aux
        }
    }
}
```

```
  [1] 100  99  98  97  96  95  94  93  92  91  90  89  88  87  86  85  84
 [18]  83  82  81  80  79  78  77  76  75  74  73  72  71  70  69  68  67
 [35]  66  65  64  63  62  61  60  59  58  57  56  55  54  53  52  51  50
 [52]  49  48  47  46  45  44  43  42  41  40  39  38  37  36  35  34  33
 [69]  32  31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16
 [86]  15  14  13  12  11  10   9   8   7   6   5   4   3   2   1
```

[Opcional] Outra possibilidade (mais eficiente), é fazer o 2º loop a ir até o antepenúltimo elemento (comparando com o penúltimo), e assim por diante. Isso evita fazer comparações com elementos que você já tem certeza que estão na ordem certa:

```
for (num_loops in 0:98) {
    for (i in 1:(length(vetor) - 1 - num_loops)) {
        if (vetor[i] < vetor[i + 1]) {
            aux = vetor[i + 1]
            vetor[i + 1] = vetor[i]
            vetor[i] = aux
        }
    }
}
```

```
  [1] 100  99  98  97  96  95  94  93  92  91  90  89  88  87  86  85  84
 [18]  83  82  81  80  79  78  77  76  75  74  73  72  71  70  69  68  67
 [35]  66  65  64  63  62  61  60  59  58  57  56  55  54  53  52  51  50
 [52]  49  48  47  46  45  44  43  42  41  40  39  38  37  36  35  34  33
 [69]  32  31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16
 [86]  15  14  13  12  11  10   9   8   7   6   5   4   3   2   1
```



□