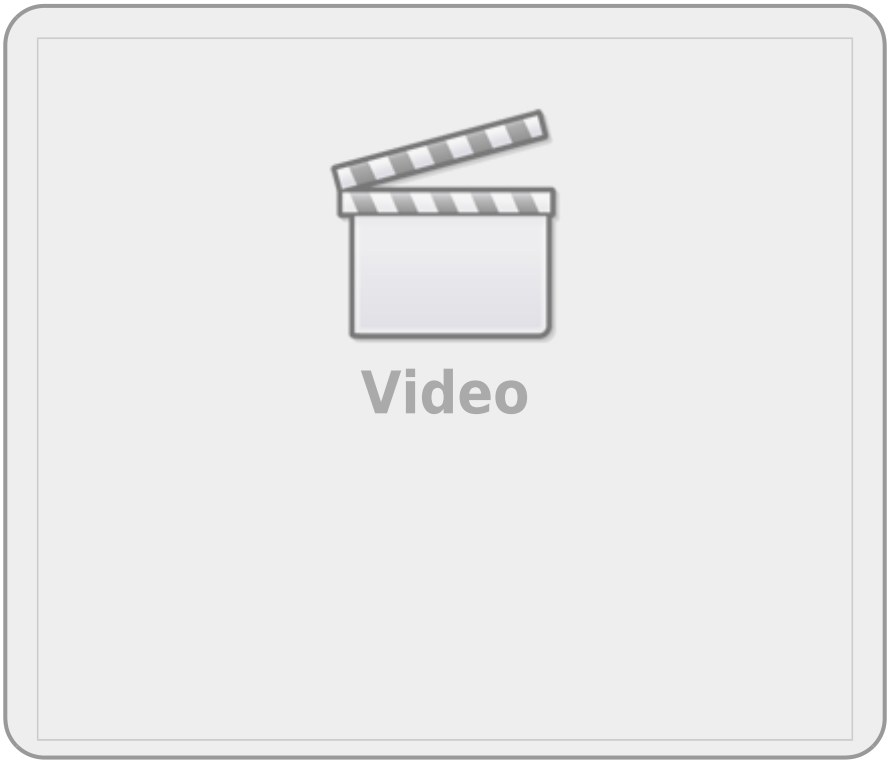
- Tutorial
- Exercícios
- Apostila

1a. Introdução ao R: bases da linguagem

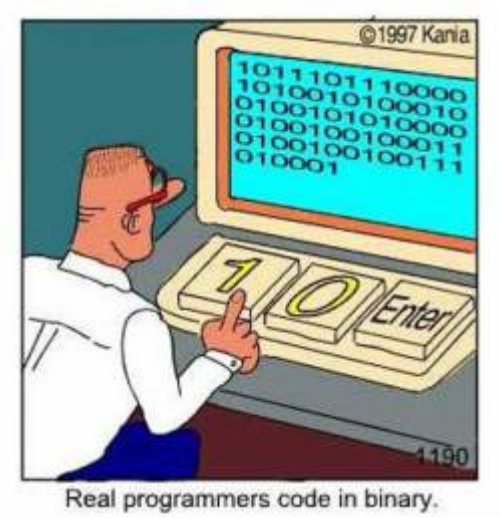
Antes de iniciar essa primeira aulas veja a vídeo-aula sobre o esquema do curso em Curso IBUSP/ESALQ - 2024

Meu nome é Alexandre e costumo falar devagar nas vídeo-aulas. Como elas estão em um canal do Youtube é possível acelerar clicando em `Settings` (símbolo de engrenagem que aparece na barra inferior do vídeo) e em seguida em `Playback speed`. Procure a sua velocidade!



Entre as várias características que definem uma linguagem computacional está a forma como o código é implementado pelo sistema operacional, ou seja, como a linguagem do programa é transformada em linguagem de máquina. Há dois tipos básicos de implementação: compilação e interpretação. O R faz parte do segundo grupo, por isso podemos conversar com o programa a cada linha de comando. Além disso, nossa conversa com o R é definida como uma linguagem de alto nível, significando que a sintaxe é similar à linguagem humana e se distancia da linguagem da máquina que

é binária, só contendo zeros e uns.



Real programmers code in binary.

Outra característica do R é que ele é uma linguagem orientada a objetos, ou seja, manipulamos objetos com procedimentos inerentes à classe a que eles pertencem. Essas características do R fazem com que esse ambiente de programação seja similar a uma oficina onde matéria-prima (objetos) e ferramentas [1)] são manipuladas para efetuar uma tarefa que normalmente se resume na construção de outros objetos, ou '*obras de arte virtuais*'. Vamos entrar nessa oficina!

ateliêR

Vamos usar uma interface web para rodar o R. Nos quadros **rdrr.io** é possível submeter linhas de código a um servidor que interpreta o código do R e retorna o resultado da operação em uma outra janela. Caso o servidor não esteja disponível, ou a conexão da internet não seja boa, é possível rodar as linhas de código em uma sessão do R no seu computador, apenas copiando e colando as linhas de código apresentadas antes dos quadros do **rdd.io**.

O comando que vamos executar é:

Hello, world!

```
print("Hello, world")
```

Clique no botão RUN abaixo!

Os computeiros dizem que a primeira coisa que devemos fazer quando aprendemos uma linguagem computacional é fazê-la dizer Hello, world!. Pronto, já fizemos nossa primeira tarefa na linguagem R!

No código acima, ao clicar em **RUN** enviamos o comando `print("Hello, world!")` para o interpretador do R e recebemos o resultado dessa operação. Apesar da simplicidade desse exemplo, temos alguns conceitos básicos da sintaxe do R que são importantes.

Note que temos no comando acima caracteres (letras, símbolos e espaços em branco) que estão agrupados entre aspas "`Hello, world!`". Esse é o primeiro conceito importante: **O que está entre aspas o R interpreta como sendo caracteres**. Parece óbvio, mas veja o que acontece ao rodar o código abaixo:

```
print(Hello)
```

A mensagem `Error in print(Hello) : object 'Hello' not found`, significa que o R não encontrou o objeto com o nome `Hello`. Nossa segunda definição: **caracteres que não estão entre aspas o R interpreta como sendo o nome de objetos**. No caso, o objeto com o nome `Hello` não foi encontrado!

Atribuição

Ok! E como fazemos para criar um objeto no R? Para isso usamos as funções de atribuição. Na linguagem temos 3 tipos de atribuições utilizando símbolos diferentes:

ATRIBUIÇÃO NO R

- junção dos caracteres `<` e `-`: atribuição à esquerda;
- caracter `=`: o mesmo que acima, atribuição à esquerda;
- junção de `-` e `>`: atribuição à direita;

Nas regras de boas práticas de estilo da linguagem, em geral, se diz que deve-se usar a primeira forma, que a segunda é aceitável, mas que não devemos usar a terceira!

Outra boa prática de estilo da linguagem é, uma vez escolhida a forma a ser usada, usar sempre a mesma forma ao longo de seus ***scripts***. Iremos definir o que é um ***script*** no item 'O Código' desse tutorial.

Vamos criar nosso primeiro objeto no R:

```
Hello <- "Hello, world!"
```

Parece que nada aconteceu, mas atribuímos ao objeto chamado `Hello` os caracteres que compõem a frase `Hello, world!`. Após criar o objeto podemos manipulá-lo ou apenas chamá-lo para exibir o que foi atribuído a ele.

```
Hello
```

Agora temos novamente o retorno de "`Hello, world!`", mas dessa vez a frase vem do objeto `Hello`. Quando chamamos um objeto que existe no R ele nos retorna o que está armazenado nele.

Classes de objetos

Todo o objeto criado em uma sessão do R é atribuído a uma classe. Isso é feito automaticamente pelo R, caso o usuário não explicite a classe do objeto na sua criação. Para acessar a classe do objeto que

criamos, utilizamos a função `class`:

```
Hello <- "Hello, world!"
Hello
class(Hello)
```

A classe 'Function'

No nosso primeiro código do R, havia um objeto chamado `print`. Vamos visualizar a classe a que pertence esse objeto:

```
class(print)
```

O R nos diz que esse objeto é da classe função. Os objetos da classe `function` em geral estão associados a uma documentação que nos ajudam a entender como usar essa ferramenta. Para acessar a documentação no R, utilizamos outra ferramenta que é a função `help`[2)].

```
help(print)
```

Uma questão interessante aqui é que estamos usando uma ferramenta, a função `help`, para manipular o objeto `print`, que por sua vez também é uma função. O que acontece se chamarmos o objeto `help` sem os parênteses?

```
help
```

É muito importante diferenciar o objeto que contém o código da função do procedimento ao executar essa função. A diferença entre um e outro está em um detalhe pequeno que são os parênteses `(...)` que acompanham o nome da função. O nome da função acompanhado dos parênteses faz com que o procedimento associado a esse objeto seja executado. Caso não seja acompanhada dos parênteses, o objeto da classe função irá retornar aquilo que está atribuído a ele: o texto de código que a função contém.

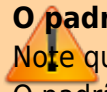
Argumentos das funções

Na documentação da função `print` há a descrição de argumentos que, entre outras coisas, flexibilizam o procedimento da função. O primeiro argumento, chamado `x` é o objeto que será manipulado, um outro argumento dessa função é o `digits`. Vamos usá-lo:

```
print(x= 1.23456789, digits = 3)
```

Para explicitar que estamos manipulando objetos, podemos fazer o procedimento em duas etapas, primeiro atribuindo o valor `1,23456789` a um objeto e depois solicitando para que ele seja mostrado na tela com apenas 3 dígitos.

```
numero <- 1.23456789
print(x= numero, digits = 3)
```



O padrão decimal do R
Note que o R utiliza o símbolo de `.` para indicar o decimal no padrão de números em Inglês. O padrão em Português, que é o uso de `,` como indicador, não funciona no R!

Agora vamos ver a diferença na manipulação que o `print` faz, dependendo da classe do objeto:

```
palavra <- "1.23456789"
print(x= palavra, digits = 3)
```

As classes 'numeric' e 'character'

Porque o objeto `numero` é manipulado diferentemente do objeto `palavra`? Por que são objetos de classes diferentes e a função `print` reconhece essa diferença e trata eles de forma diferente. Quanto a função manipula números (i.e. classe `numeric`) o argumento `digits` faz sentido. Quando o objeto é da classe `character` esse argumento é desprezado. Aqui tem um conceito avançado da linguagem: a função `print` executa diferentes procedimentos dependendo da classe do objeto que ela manipula.

Já vimos anteriormente que para acessar a classe a que um objeto pertence podemos usar a função `class`:

```
class(numero)
class(palavra)
```

Vamos agora usar uma outra função para exemplificar a sintaxe básica do R. A função em questão é `round`, que arredonda um valor numérico até a casa decimal solicitada. Diferentemente do `print`, que não modifica o valor, apenas imprime ele com o número de casa decimais solicitado, o `round`, por sua vez, faz a transformação arredondando o valor [3].

A primeira ação que deve ter ao utilizar uma função no R é: **SEMPRE LER A DOCUMENTAÇÃO**. A documentação do `round` descreve que ele também tem um argumento chamado `digits`.

```
outro <- round(numero, 4)
```

Cadê o nome do argumento?

Note que o código acima não tem o nome dos argumentos. Estamos usando uma das regras dos argumentos no R que é a posição. Caso o nome não seja dado, o R usa a posição para atribuir o valor ao argumento. É possível usar ambas regras, posição e nome, o que é

bastante comum. Uma outra regra é a do padrão único do nome simplificado. Por exemplo, o `dig = 3` será reconhecido como `digits = 3` desde que não haja nenhum outro argumento que comece com `dig` no nome. Como sabemos a posição e nome dos argumentos? No `help`. Consulte sempre a documentação! Quase todas as funções que aparecem nos códigos do wiki estão conectadas a sua documentação por um hiperlink. Portanto, use e abuse!

Sintaxe básica

A sintaxe básica do R pode ser definida como:

```
object <- tool(x, arg2 = y, arg3 = z)
```

Podemos ler o comando acima como sendo: "utilize a ferramenta `tool` para manipular o objeto x tendo o argumento `arg2` com o atributo y e a opção `arg3` como z. O resultado dessa manipulação é armazenado no objeto de nome `object`. Note que o R, nesse caso, não devolveria nada na tela, pois o resultado da manipulação é atribuído a um objeto [4)].

Estrutura e tipos de dados

Até aqui vimos dois tipos de informação que podem ser manipuladas no R: caracteres e números. Os números, por sua vez, podem ser de dois tipos: números com decimais (`numeric`) e inteiros (`integer`). Essa distinção é importante para a maneira como o R armazena essa informação na memória do computador, de resto elas funcionam como números racionais na matemática clássica. No capítulo seguinte vamos tratar das funções matemáticas mais a fundo. Aqui vamos apenas ver as bases conceituais dos tipos de dados básicos e qual a estrutura básica de armazenamento em objetos. As operações da álgebra básicas no R usam os mesmos símbolos que na matemática tradicional: +, -, / e *.

```
10 + 10
10 - 10
10 / 10
10 * 10
```

Como vimos na seção anterior podemos atribuir valores numéricos a um objeto. Depois disso, podemos manipular os valores indiretamente por intermédio do objeto.

```
dez <- 10
dez + dez
dez - dez
dez / dez
dez * dez
```

Atribuímos o valor `10` ao objeto `dez` e depois manipulamos o objeto `dez`. Isso não parece ser uma vantagem. Estamos trocando dois dígitos, o valor `10`, por um objeto que contém 3 letras, o `dez`. A vantagem começa quando atribuímos o resultado de operações a um outro objeto.

```
cem <- dez * dez
dezmil <- cem * cem
cem
dezmil
```

Vetores

Ficaria ainda melhor se pudéssemos operar mais de um valor de uma vez. Como armazenar mais de um valor em um objeto? Usamos a função `c` que significa concatenar ou combinar. Os elementos combinados são a estrutura básica de dados no R, que é o objeto da classe `vector`. Esse é o elemento básico dos objetos no R. Mesmo que o objeto só tenha um elemento, trata-se de um vetor com comprimento igual a um.

```
contadez <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
contadez
```

Indexação de Vetores

Note que, antes de iniciar a apresentação dos valores que estão no vetor `contadez` o R apresenta o valor 1 entre colchetes `[1]`. Caso o nosso vetor fosse longo e tivesse que ser apresentado em várias linhas, outros valores em colchetes iriam iniciar essa outras linhas de apresentação dos dados. Esses valores representam a indexação do elemento que inicia a linha de apresentação do conteúdo do vetor. Ou seja, o elemento na posição 1, no nosso caso é o valor 1. Vamos inverter esse vetor, em seguida combiná-lo com o vetor anterior!

```
invertedez <- rev(contadez)
descontadez <- c(invertedez, contadez)
descontadez
```

Agora, como fazemos para acessar elementos específicos dentro desse vetor? Para isso usamos a indexação de posição.

Por padrão no R o primeiro elemento de um vetor está na posição 1.

Essa frase pouco informativa é um detalhe importante. Em muitas linguagens computacionais, diria até que a maioria das linguagens mais populares, a indexação começa pela posição definida como `0` (zero)! Mais a frente vamos usar outras indexações de vetores e de outras classes de objetos de dados. Abaixo temos alguns exemplos, simples para vetores:

```
descontadez
descontadez[7]
```

```
descontadez[c(1, 5, 10, 20)]
```

A classe 'Date'

Crie objetos com as datas do tri e tetracampeonatos mundiais do Brasil[5]:

```
copa70 <- "21/06/70"
copa94 <- "17/07/94"
```

Qual a diferença em dias entre essas datas? A subtração retorna um erro (verifique):

```
copa94 - copa70
```

Esse erro acontece porque os objetos são caracteres, uma classe que obviamente não permite operações aritméticas. Já sabemos verificar a classe de um objeto, digitando o código:

```
class(copa70)
class(copa94)
```

O resultado seria `character` para ambos!

Mas o R tem uma classe para datas, que é `Date`. Vamos fazer a coerção [6] dos objetos para esta classe, verificar se a coerção foi bem sucedida, e repetir a subtração. O código para isso está descrito abaixo:

```
copa70 <- as.Date(copa70, format = "%d/%m/%y")
copa94 <- as.Date(copa94, format = "%d/%m/%y")
class(copa70)
class(copa94)
copa94 - copa70
```

NOTA: o argumento `format` da função `as.Date` informa o formato em que está o conjunto de caracteres que deve ser transformado em data, no caso dia/mês/ano (`%d/%m/%y`), todos com dois algarismos. Veja a ajuda da função para outros formatos.

Inclua na janela do R online abaixo o código que gera os objetos `copa70` e `cop94`, em seguida verifique a classe a que pertencem, e depois faça a transformação para a classe `Date` e a subtração entre eles.

Comentando meu código

Ao submeter uma linha de comando ao R é possível incluir comentários usando o

símbolo de # . O hashtag ou sustenido indica ao R que a partir daquele ponto até o final da linha o código não deve ser interpretado.

A classe 'logical'

Até o momento, vimos algumas naturezas de informação que podemos armazenar e manipular no R: caracteres, datas e números. Uma outra natureza importante de dado básico no R é chamada de lógica. As palavras TRUE e FALSE e também as abreviações T e F são reservadas para esse fim. Uma questão importante dos dados lógicos é que a eles também são associadas os valores 0 e 1, para FALSE e TRUE, respectivamente. Veja abaixo como podemos operá-los algebricamente:

```
TRUE + TRUE
TRUE / FALSE
TRUE * FALSE
```

Além disso, o R retorna TRUE ou FALSE quando fazemos algum procedimento utilizando operadores lógicos.

Operadores Lógicos

- == : igual
- != : diferente
- > : maior que
- < : menor que
- >= : maior ou igual
- <=: menor ou igual
- | : uma das condições
- & : ambas as condições

Alguns exemplos de operações lógicas no R:

```
numero <- 1.23456789
numero < 1
numero > 1
## abaixo primeiro imprime o valor e depois faz o teste lógico
print(numero, digits = 4) == numero
round(numero, digits = 4) == numero
numero > 1 & log(numero) > 1
numero > 1 | log(numero) > 1
```

NOTA: Note que a igualdade é definida por dois caracteres de igualdade: "==". Se usarmos apenas um carácter de igualdade no R isso será interpretado como

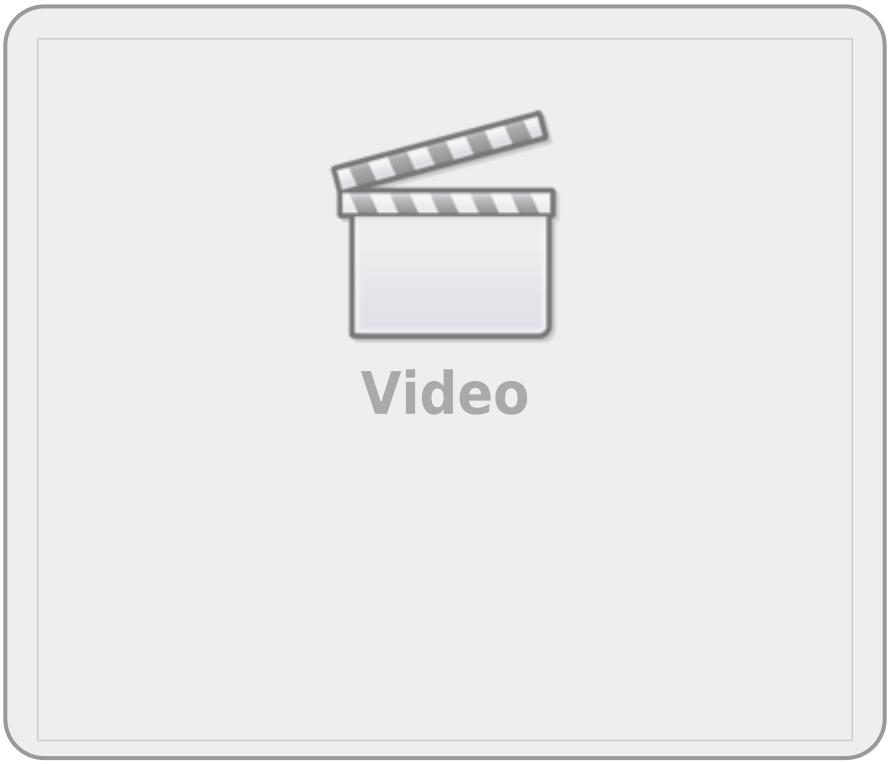
uma atribuição, como o sinal ”←“.

A operação lógica também funciona com vetores, obedecendo a posição dos elementos:

```
contadez <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
invertedez <- rev(contadez)
invertedez
invertedez > contadez
```

A classe 'factor'

Para a melhor compreender essa classe de objetos no R, o Prof. Alexandre preparou uma vídeo-aula específica, disponível abaixo.



Imagine um experimento em que classificamos as plantas em uma escala de herbivoria com os níveis: “alto”, “médio”, “baixo” e “nulo”. Vamos criar um objeto que representa o valor desta medida de herbivoria em uma amostra de 14 plantas:

```
herb <- c("A", "M", "M", "A", "A", "M", "M", "B", "A", "A", "A", "A", "B",
"A")
```

E então criar um objeto da classe fator com estes valores:

```
herbFactor <- factor(herb)
```

Usamos a função `table` para contar o número de observações em cada nível do fator, cujo resultado

atribuímos a um outro objeto. Os valores são exibidos se digitamos o nome do objeto.

```
herbTable <- table(herbFactor)
herbTable
```

A função para gerar gráficos `plot` pode ser aplicada diretamente ao objeto desta tabela:

```
plot(herbTable)
```

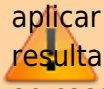
Rode o código abaixo e avalie o que está sendo produzido em cada linha de comando . Caso fique com dúvidas a primeira coisa a fazer é consultar o `help()` da função. O quadro onde temos o código abaixo, pode ser editado e pode rodar novamente com outro código. Fique à vontade para explorar a documentação das funções que estamos apresentando.

Note que na tabela e na figura os níveis não estão ordenados da forma como deveriam e falta o nível de herbivoria nula. Isto acontece porque, ao criar uma variável de fator a partir de um vetor de valores, o R cria níveis apenas para os valores presentes, e ordena estes níveis alfabeticamente. Caso um nível não tenha sido observado nos dados, ele fica de fora da variável, mas o correto seria ele ter a contagem como `0` .

Para ordenar o fator e incluir um nível que não foi representado na amostra usamos o argumento `levels` da função `fator`:

```
herbFactor <- factor(herb, levels = c("N", "B", "M", "A"))
```

Modifique o código da janela acima, incluindo o argumento `levels` na função `factor` e rode novamente o código todo na janela abai



NOTA: há uma classe para fatores ordenados que poderia se aplicar aqui, mas seu uso tem implicações importantes nos resultados de algumas análises, que no momento não vêm ao caso. Mais informações a respeito na ajuda da função factor.

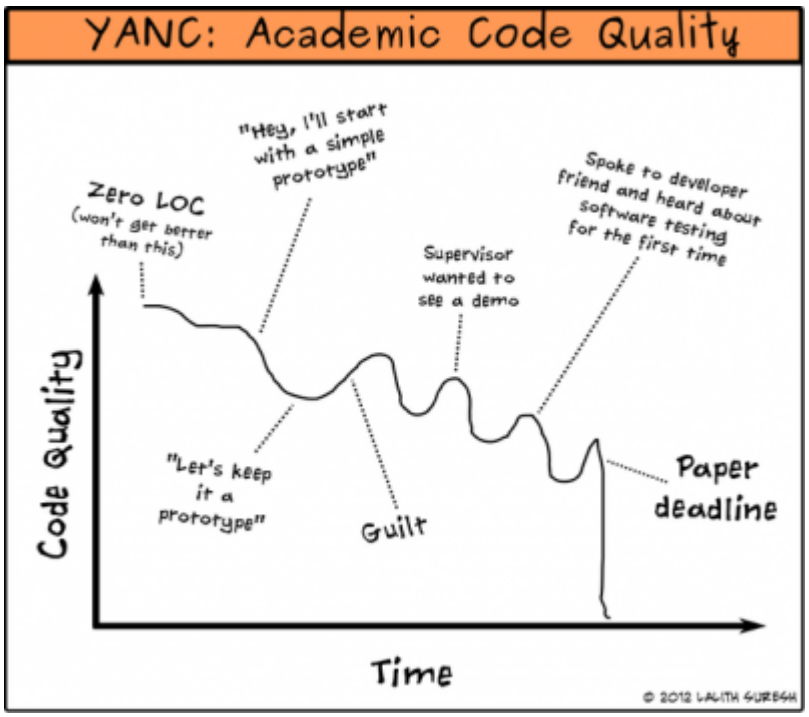
O Código

Antes de continuar a introdução aos conceitos básicos do R, vamos entender uma conduta importante em programação. Um dos primeiros hábitos que você deve adquirir para trabalhar com o R é **não digitar os comandos diretamente no console do R**[7)], e sim em um arquivo texto, que chamamos de ***script*** ou ***código***. Essa intermediação entre o texto do comando e o interpretador, feita pelo script, é importante pois garante que o que está sendo direcionado ao R é armazenado no arquivo texto, que por fim, pode ser salvo e armazenado no computador, como um registro do procedimento executado e para ser utilizar novamente quando necessário.

Reprodutibilidade do procedimento

Quando trabalhamos em uma planilha eletrônica, a partir de dados brutos, podemos salvar os gráficos ou os dados modificados após manipulados. Entretanto, o procedimento não é salvo. Se precisar fazer o mesmo procedimento para outro conjunto de dados precisará lembrar todas as etapas e a ordem em que foram executadas. Em programação, o script é nosso roteiro do procedimento que foi executado. Para repetir um procedimento é só executar novamente o script. Isso incrementa muito a reprodutibilidade do nosso procedimento, uma qualidade muito importante para a ciência de um modo geral, mas também para o dia a dia. Por isso, a partir desse momento no curso, iremos abandonar a interface do R online que estávamos usando para rodar o código e vamos, a partir de agora, produzir script ou códigos!

Editor de Código



Um editor de código nada mais é do que um editor de texto puro como o bloco de notas do Windows. Algumas funcionalidades são bem vindas, como por exemplo, enviar a linha de código diretamente para o console do R sem a necessidade de copiar e colar.

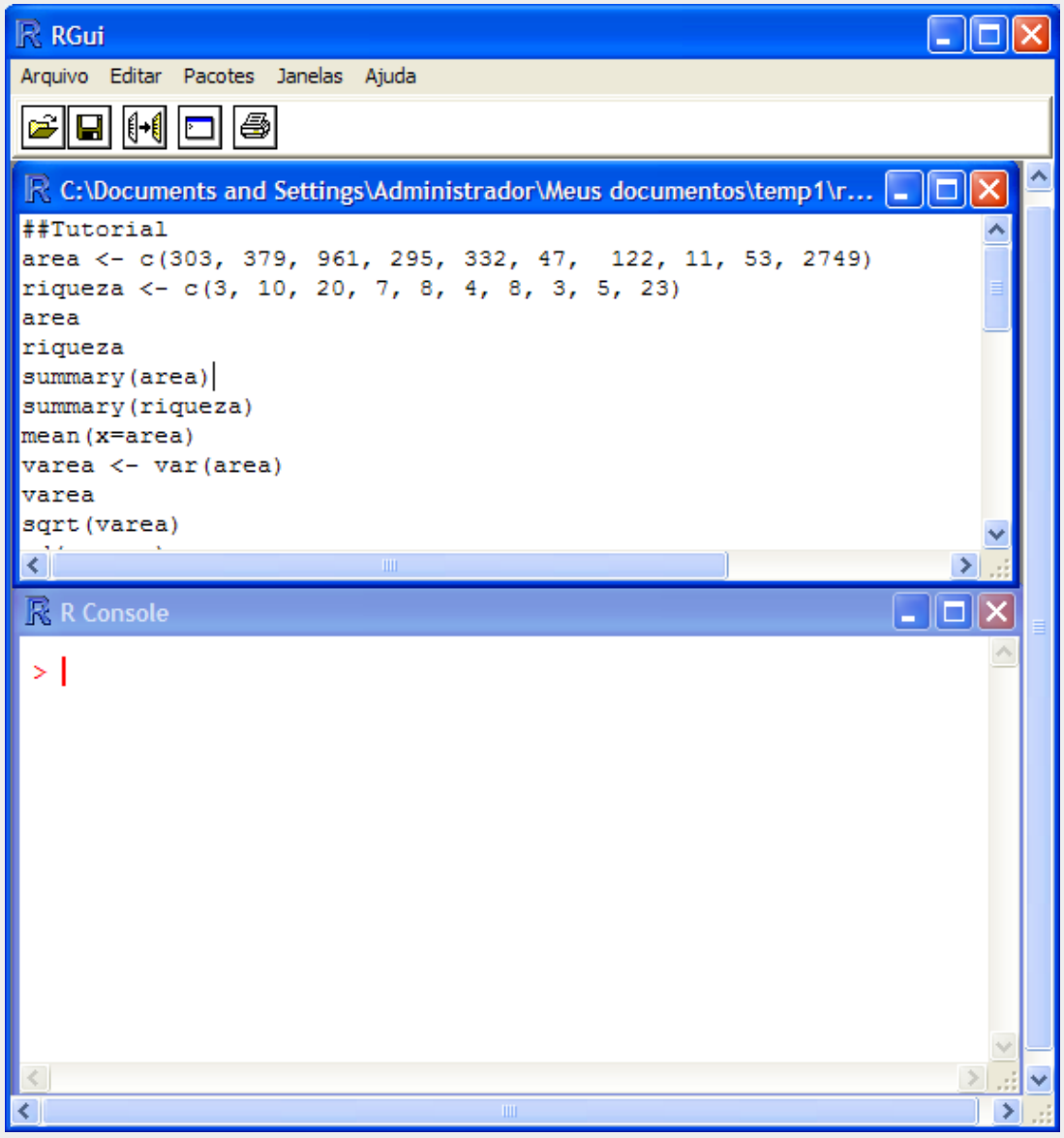
A instalação básica do R contém uma interface gráfica de usuário (R-GUI) simples, tanto no Windows como no IOS, que acompanha um editor de códigos.

O editor de códigos do R-GUI no Windows e no Mac é bastante simples e costuma ser uma boa opção inicial para usuários deste sistema. Para esta disciplina ele é suficiente.

No Linux não há uma GUI padrão para o R, e esta escolha deve ser feita logo no início.

Na página de material de apoio há uma seção com várias dicas sobre interfaces para o R para lhe ajudar.

A figura abaixo é uma captura de tela do R-GUI do Windows, mas no MAC o editor é similar, e você pode manter a mesma lógica. Deixe sempre uma janela de código aberta acima da janela do R, como na imagem abaixo:



Interface de usuário R-GUI

Na figura acima há duas janelas com funcionamentos e objetivos muito distintos.

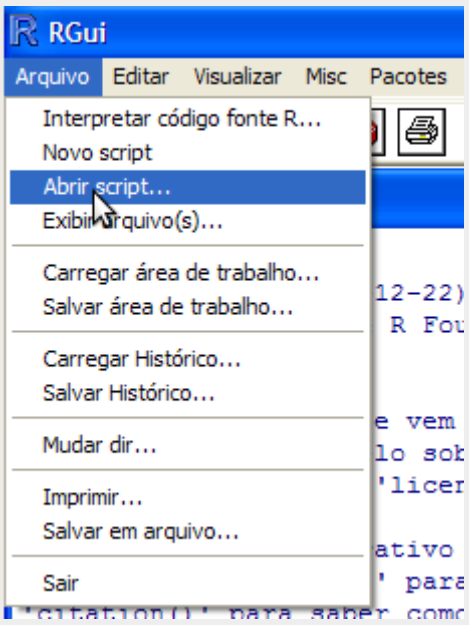
1. a janela da parte superior apresenta um arquivo de texto puro que pode ser editado e salvo como texto. Por padrão salvamos esses arquivos com a extensão `.r` ou `.R` para reconhecermos que é um script da linguagem R. O sistema operacional deve reconhecer a extensão com sendo do R automaticamente.
2. a janela na parte inferior é o console do R, ou seja o programa propriamente dito. Essa janela recebe os comandos de código e envia ao interpretador do R, que por sua vez, retorna o resultado final do processamento[8)].

Para evitar confusão e perda de trabalho é importante digitar as informações que serão transmitidas ao R (linhas de código) no arquivo texto e ir passando esses comandos ao R. Uma boa prática também é comentar as linhas de código para que outras pessoas, ou mesmo a pessoa que criou o código, possam entender ou lembrar o que o código executa.

É imprescindível aprender a se organizar dentro da lógica do ambiente de programação, com o risco de perder trabalho ou ficar completamente perdido entre as tarefas que executa.

O primeiro Script

- Copie todas as linhas de códigos que foram processados nesse tutorial até o momento em arquivo texto simples no bloco de nota do Windows ou algum outro programa simples de texto (TextEdit no macOS);
- Salve o arquivo em uma pasta [9)] conhecida do seu computador, associada a essa disciplina, com o nome e extensão `tutorial01.r` [10)];
- Execute o R e abra o *script* que salvou, utilizando a opção do menu “Arquivo/Abrir script”:



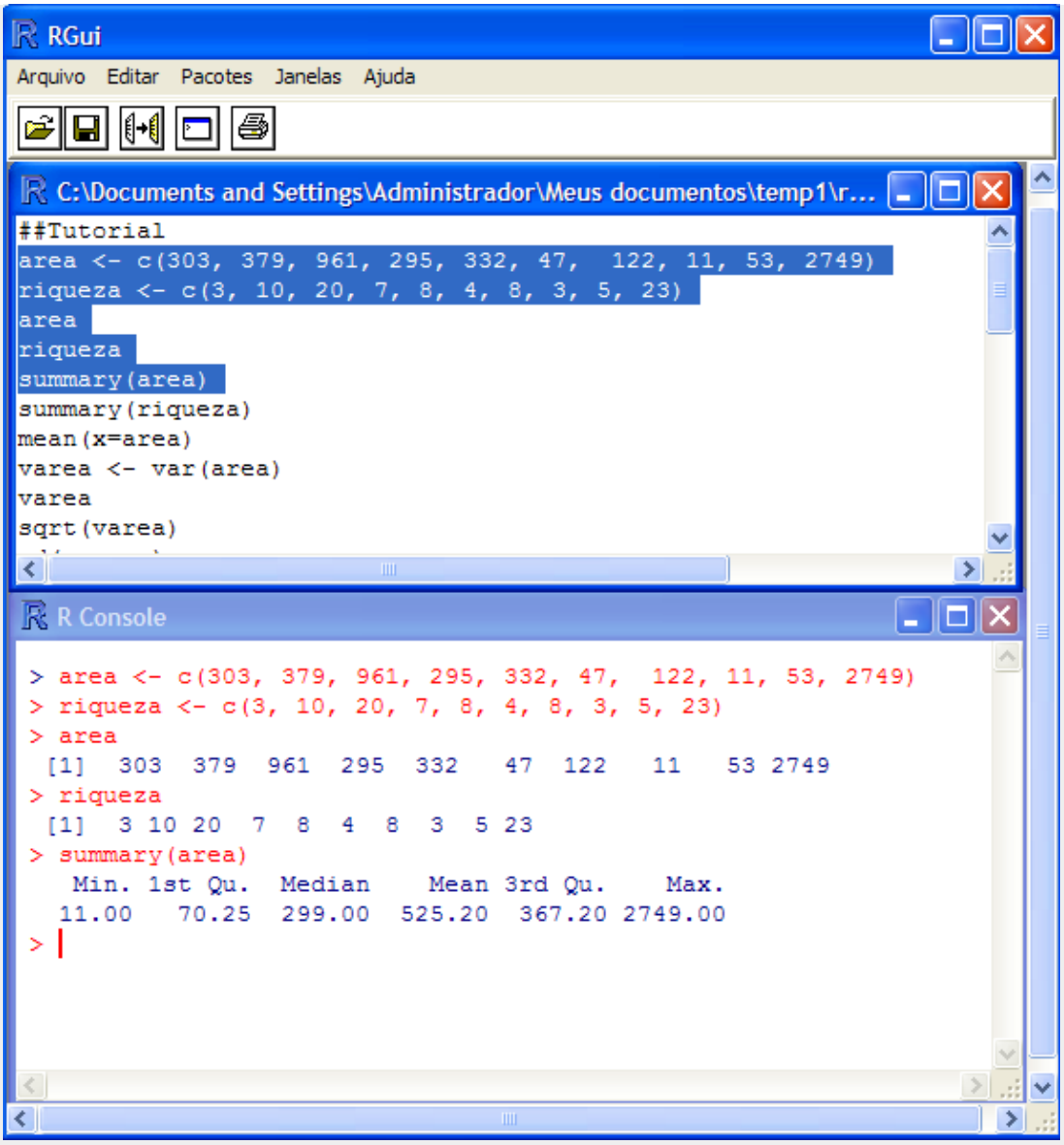
- Vá para a janela do *script*, coloque o cursor na primeira linha e tecle `Ctrl-r`. Faça o mesmo com as linhas seguintes;

Para Usuários de MAC

Para enviar comandos do editor de código do R-GUI para o R utilize *Command+Enter* ou *Command+Return*.

Veja o material RMacOSX

- Coloque o título no arquivo como sendo `## Tutorial Introducao ao R`
- Na linha seguinte coloque a data como comentário: `## Data: ….`;
- Comente cada bloco de código com o nome do tópico que o código está associado no roteiro
- Comente ou retire qualquer linha de código que tenha gerado erros durante o processamento;
- Retire a redundância na atribuição dos abjetos, mas cuidado com objetos que são sobrescritos, veja o `copa70`, por exemplo;
- Ao final selecione todas as linhas do script, inclusive comentários e e tecle `Ctrl-r` para submeter tudo ao interpretador do R;
- Garanta que não há mensagens de erro ao longo do processamento do script.



Comentários no código

Para fazer comentários no código, usamos o símbolo de # . Qualquer conteúdo na linha de comando depois do # não é interpretado pelo R. Utilizamos os comentários, em geral, para tornar o código autoexplicativo.

- Salve o *script* com estas modificações.

Você terá que submeter o *script* salvo no nosso sistema de correção automática de código, chamado **notaR**. Portanto, que o arquivo foi salvo corretamente.

Siga para a aba de exercícios para seguir os exercícios desse tópico. Os exercícios ficarão embutidos nesse wiki, mas deixaremos sempre o link para o

notaR caso prefiram abrir a plataforma diretamente. **Lembre-se de logar no sistema notaR** antes de fazer os exercícios e não deixe de passar pela aba da apostila, ela e complementar aos tutoriais, apesar de alguma redundância desejável.

1)
funções também são uma classe de objetos
2)
O caracter ? funciona como um atalho para essa função
3)
arredondamentos podem ser danosos, principalmente para cálculos sequenciais e números pequenos
4)
para atribuir o resultado a um objeto e ao mesmo tempo mostrar na tela, utilize parênteses iniciando e fechando a linha de comando
5)
fonte: FIFA
6)
procedimento de transformar uma classe em outra
7)
Console é a interface de interação com o interpretador da linguagem: recebe o comando, envia ao interpretador e retorna a resposta. O que vínhamos usando no início desse tutorial é um interpretador online do R
8)
quando a tarefa solicitada é a representação de um gráfico, uma nova janela é aberta, um dispositivo gráfico.
9)
diretório de trabalho é o nome técnico desta pasta para o R
10)
quando o sistema operacional não mostra a extensão dos arquivos é preciso configura-lo para que seja apresentado