Mini Curso

O objetivo desse mini-curso[1] é apresentar aos alunos conceitos básicos sobre a linguagem e sua sintaxe. Esse mini-curso foi estruturado para atividades de 8 horas, divididas em dois dias. O primeiro dia é centrado na lógica do ambiente de programação em linha de comando (CLI: Command Line Interface) e o segundo dia na utilização de ferramentas avançadas através da interface gráfica Rcommander. Em grande parte, o material aqui apresentado é um recorte do material contido no nosso wiki EcoR

Professores

- *Alexandre Adalardo de Oliveira*

> **Aula Relâmpago** Há em nosso material algo que chamamos de Aula Relâmpago que é parte das atividades preparatórias do curso integral. Sugerimos que faça essa atividade após finalizar esse mini curso.

O repositório

O CRAN (The Comprehensive R Archive Network) é o repositório oficial do R. Lá encontrarão todo o material necessário para utilizar essa ferramente de análise e apresentação gráfica de dados. Nossa primeira atividade é navegar nesse repositório:

- digite R CRAN no google e encontre a página oficial do R;
- leia o Task Views [2] do Envirometrics;
- usar Search > Rseek buscar **'square root'** e **'pca'**;
- em Packages buscar EcoVirtual e depois entrar na sua dependência RcmdrPlugin.EcoVirtual

Sintaxe e operações básica no R

A sintaxe básica do R é:

```
nomeobjeto <- nomedafuncao(argumento1, argumento2, argumento3, …)
```

Uma forma de ler essa linha de código é: estou atribundo ao objeto ***nomedoobjeto*** o resultado da função ***nomedafuncao*** com as seguintes configurações: `argumento1`, `argumento2`, ...

Uma função executa uma tarefa ou conjunto de tarefas acopladas. Normalmente, mas não necessariamente, a função atua em um objeto de dados que é definido nos primeiros argumentos. Funções podem conter funções, rode o exemplo abaixo:

```
## função para construir sequencias
seq(from=0, to =100, by=10)

## para construir um gráfico!
plot(x=seq(from=0, to=100, by=10), y= seq(to=100, from =0, by=10)*5)

## note que podemos usar funções dentro de funções para construir os
argumentos!
```

A tradução do código acima é:

1. `faça um sequencias de valores de 0 a 100 a intervalos regulares de 10 unidades`;
2. `faça um gráfico onde os valores de x são uma sequência de valores que vão de 0 a 100 em intervalos de 10 e y é a mesma sequência onde cada valor foi multiplicado por 5`

A última linha de código acima pode ser desmembrado da seguinte forma, tendo a mesma *tradução*, ou seja executando a mesma tarefa:

```
meux <- seq(from=0, to=100, by=10)
meuy <- seq(to=100, from =0, by=10)*5
plot(x=meux, y=meuy, pch=16)
```

Esse novo código podemos traduzir como: criar o ojbeto `meux` com uma sequência de valores de 0 a 100 com intervalos de 10; criar o objeto `meuy` com uma sequência de valores iguais a anterior multiplicado por 5; em seguida faça um gráfico com os valores de `meux` no eixo x e `meuy` no eixo y, representados no gráfico pelo simbolo 16 (circulo preenchido).

Para a inclusão dos argumentos na função há duas maneiras que podem ser combinadas:

- colocar o nome dos argumentos, como feito acima;
- usar os argumentos na ordem definida na função sem indicar o nome;

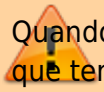
```
plot(meux, meuy)
```

- usar ambos critérios:

```
plot(meux, pch=16, y = meuy)
```

Note que no último caso os argumentos nomeados não precisam estar na ordem definida na documentação da função. Sim, você deve estar perguntando **`Como raios vou saber a sequência dos argumentos de uma função`** . Simples **`OLHE O HELP`**

UM ERRO COMUM



Quando chamamos um função sem incluir o parênteses o que temos como resultado é que o R mostra o código da função, que nada mais é que um texto. Digite `ls` no console do R! O que aparece é o código da função `ls()`

Atribuições

Um conceito importante é o de atribuir o resultado de uma operação a um objeto. Para isso é necessário utilizar os símbolos de atribuição = ou <- . Retorne ao tópico acima e tente reconhecer quando um resultado de uma operação foi atribuído a um objeto e quando foi apenas retornado no console do R. **Essa diferença é importante!**

O que devo salvar?

Os usuários de programas padrões (editor de texto, planilhas eletrônicas, pacotes estatísticos) normalmente estão preocupados em salvar sua sessão de trabalho a todo momento com receio de perder algo. Para o bom usuário do  a sessão de trabalho no R não é o mais importante. Como trabalhamos no `código` ou `script` associado ao R e não diretamente nele, o código contém toda a sequencia de comandos que foram utilizados e portanto todo o nosso trabalho. O que o bom usuário do R deve se preocupar é em **salvar o código e os dados**. É só o que precisamos para reproduzir todo o trabalho de manipulação, análise e visualização de resultados.

- Tutorial
- Exercícios
- Apostila

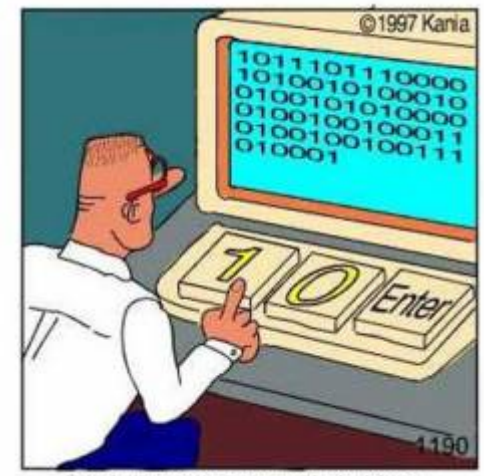
1a. Introdução ao R: bases da linguagem

Antes de iniciar essa primeira aulas veja a videoaula sobre o esquema do curso em Curso IBUSP - 2020

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

Video

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 [3] 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 internete 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: 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!

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.

Classe dos 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)
```

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 objeto 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`[4].

```
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, que é a 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 fazem 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.

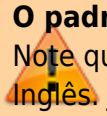
Argumentos

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 atribuíndo o valor `1,23456789` a um objeto e depois solicitando para que ele seja mostrado na tela com apenas 3 digitos.

```
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. Já em Português, o padrão é utilizar a vírgula como indicação de decimais, o que 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)
```

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 manipula números o argumento `digits` faz sentido, quando o objeto é da classe `characters` esse argumento é desprezado. Aqui tem um conceito avançado da linguagem, a função `print` chama um método que executa diferentes procedimentos dependendo da classe do objeto que ela manipula. Podemos dizer que o método é um conjunto de funções. 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 [5)].

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 hiperlink [6], use e abuse!

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 `arq3` 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 [7].

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 algebra básicas no R usam os mesmos simbolos que na matemática tradicional: +, -, / e *.

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

Como vimos na sessão 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 contem 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 pudessemos operar mais de um valor de uma vez. Como armazenar mais de um valor em um objeto? Usamos uma função `c` que significa **c**oncatenar ou **c**ombinar. Os elementos combinados são a estrutura básica de dados no R, que é o objecto 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 uma posição.

```
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 vector anterior!

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

Agora, como fazemos para acessar algum elemento 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 uma detalhe importante. Em muitas linguagem computacionais, diria até que a maioria das linguagens mais populares, a indexação começar 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)]
```

Classes Date

Crie objetos com as datas do tri e tetracampeonatos mundiais do Brasil[8]:

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

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

```
copa94 - copa70
```

Isto 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 [9] 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 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.

Dado Lógico

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 alguma 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 logico
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
```

Classe Fator

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 a 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, incluíndo 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.

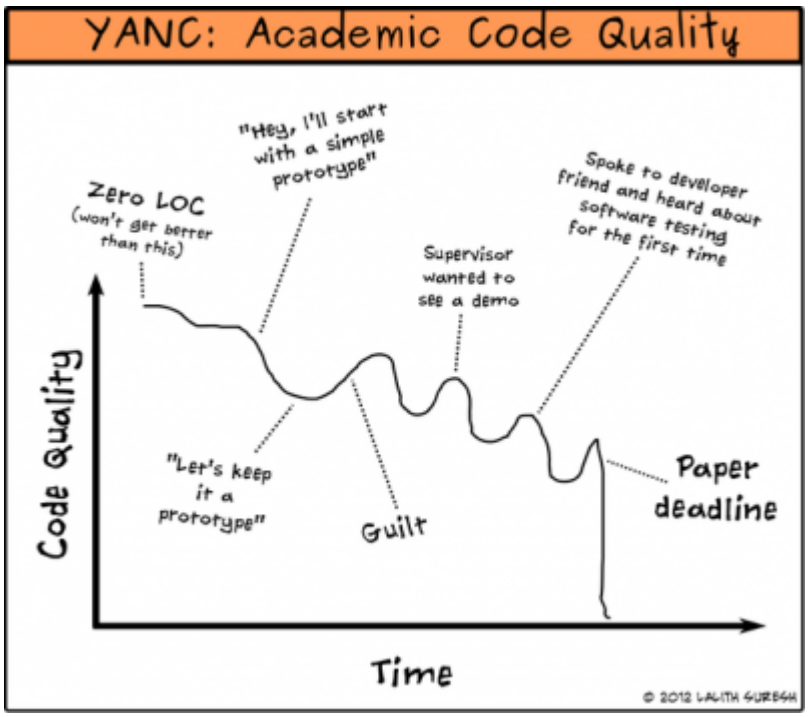
O Código

Antes de continuar a introdução aos conceitos básicos do R, vamos enteder 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**[10], 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 abondonar a interface do R online que estavamos usando para rodar o código e vamos, a patir 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.

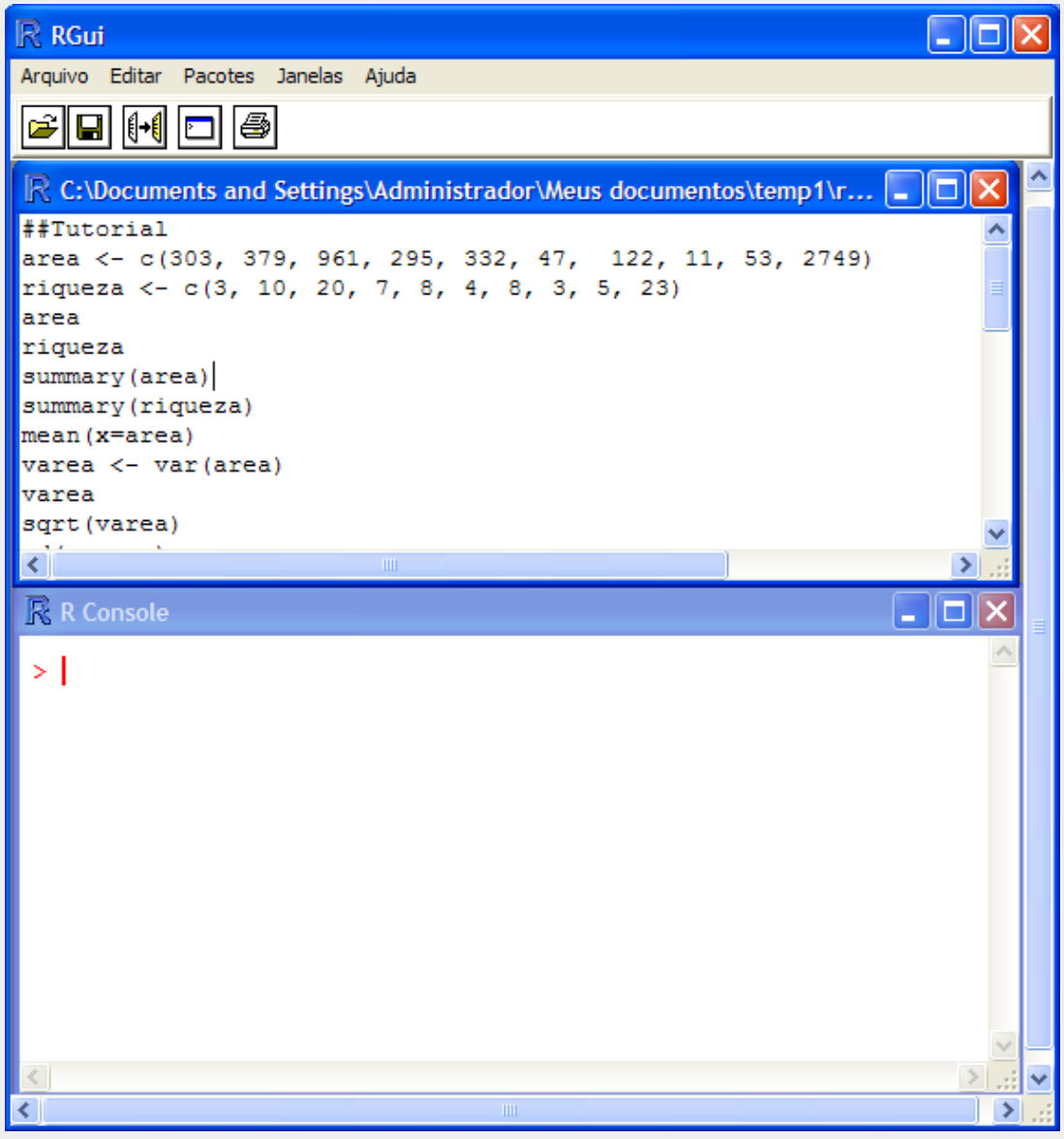
O editor de códigos do R-GUI no Windows e no Mac é bastante simples e costuma ser uma boa opção inicial para usuários deste sistema. Para esta disciplina ele é suficiente.

No Linux não uma 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

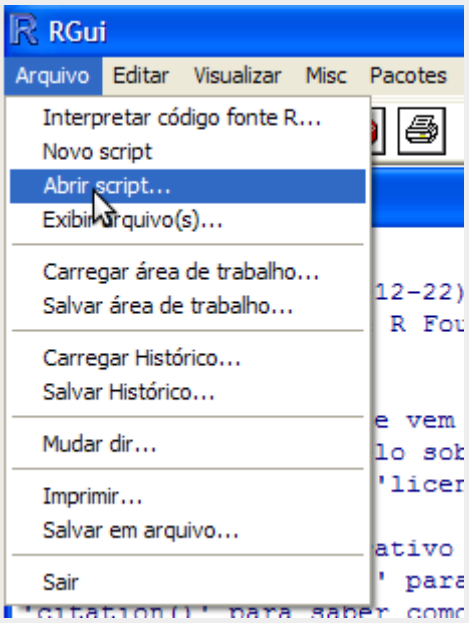
sistema operacional deve reconhecer a extensão com sendo do R automaticamente.
2. a janela na parte inferior é o console do R, ou seja o programa propriamente dito. Essa janela recebe os comandos de código e envia ao interpretador do R, que por sua vez, retorna o resultado final do processamento[11)].

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 [12)] conhecida do seu computador, associada a essa disciplina, com o nome e extensão `tutorial01.r` [13)];
- 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

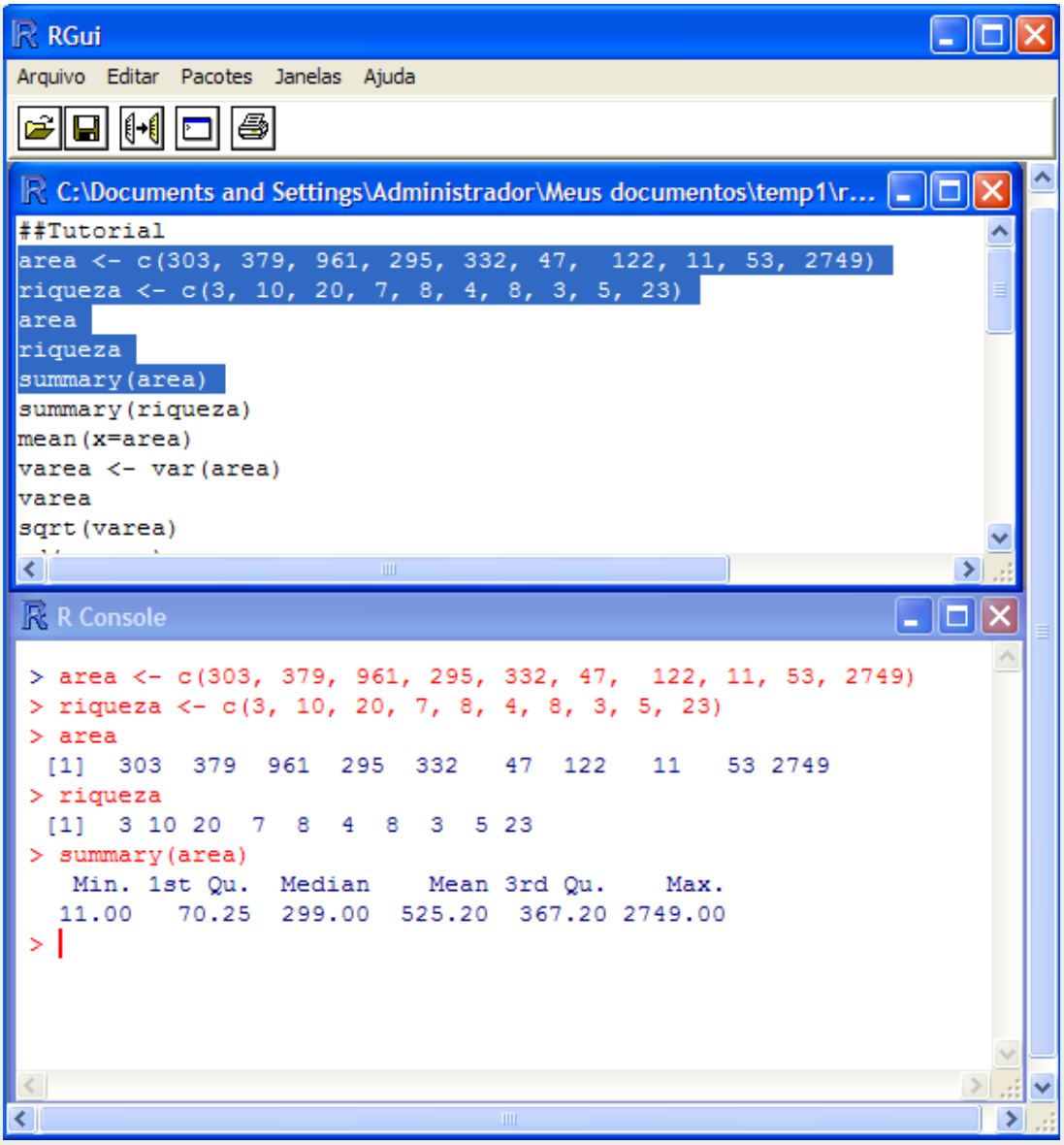
com as linhas seguintes;

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 mesagens de erro ao longo do processamento do script.



Comentários no código

Para fazer comentários no código, usamos o simbolo 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.

Agora vamos submeter o *script* salvo no nosso sistema de correção automática de código, chamado **notaR**.

Agora que já fez seu primeiro exercício no notaR. Siga para a aba de exercícios

para seguir os execícios 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 sitema 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.

2020/08/12 06:11

Você já aprendeu que **o código é tudo!**, mas **o código comentado é ainda melhor!**. Um ótimo hábito para quem está programando é comentar as linhas de códigos. Para iniciantes é interessante comentar todos os comandos, ajuda a fixar o que foi feito e ajuda muito quando estiver procurando algo que já fez parecido um dia. Para comentar linhas ou parte do código utilize o simbolo # como exemplificado no código abaixo:

```
######################
## Código da primeira aula do mini-curso R 2015
######################
##Tutorial
area <- c(303, 379, 961, 295, 332, 47,  122, 11, 53, 2749) # criando objeto com as áreas dos fragmentos
riqueza <- c(3, 10, 20, 7, 8, 4, 8, 3, 5, 23) # criando objeto com as riquezas associadas
area   # mostra os dados do objeto area criado acima
riqueza # mostra os dados do objeto riqueza criado acima
summary(area)  # cria um resumo dos dados contidos no objeto area
summary(riqueza)    # cria um resumo dos dados contidos no objeto riqueza
mean(x=area)     # calcula a média dos dados de area
varea <- var(area)  # calcula a variância dos dados de area e guarda no objeto varea
varea    # mostra a variância calculada acima
sqrt(varea)   # calcula a raiz quadrada do objeto varea
sd(x=area)    # calcula o desvio padrão dos valores que estão no objeto area
plot(x=area, y=riqueza, xlab="Area (ha)", ylab="Número de Espécies")  # faz um gráfico do riqueza por área

## Fim!
```

- Tutorial
- Exercícios
- Apostila

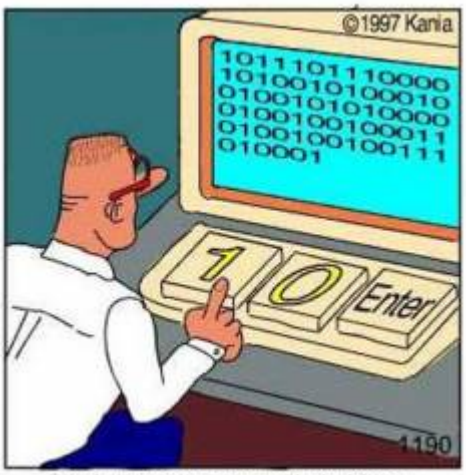
1a. Introdução ao R: bases da linguagem

Antes de iniciar essa primeira aulas veja a videoaula sobre o esquema do curso em Curso IBUSP - 2020

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

Video

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 [14)] 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 internete 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: 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!

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.

Classe dos 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)
```

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 objeto 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`[15].

```
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, que é a 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 fazem 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.

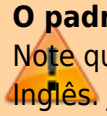
Argumentos

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 atribuíndo o valor `1,23456789` a um objeto e depois solicitando para que ele seja mostrado na tela com apenas 3 digitos.

```
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. Já em Português, o padrão é utilizar a vírgula como indicação de decimais, o que 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)
```

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 manipula números o argumento `digits` faz sentido, quando o objeto é da classe `characters` esse argumento é desprezado. Aqui tem um conceito avançado da linguagem, a função `print` chama um método que executa diferentes procedimentos dependendo da classe do objeto que ela manipula. Podemos dizer que o método é um conjunto de funções. 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 [16].

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 hiperlink [17], use e abuse!

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 [18].

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 algebra básicas no R usam os mesmos simbolos que na matemática tradicional: +, -, / e *.

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

Como vimos na sessão 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 contem 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 pudessemos operar mais de um valor de uma vez. Como armazenar mais de um valor em um objeto? Usamos uma função c que significa **c**oncatenar ou **c**ombinar. Os elementos combinados são a estrutura básica de dados no R, que é o objecto 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 uma posição.

```
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 vector anterior!

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

Agora, como fazemos para acessar algum elemento 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 uma detalhe importante. Em muitas linguagem computacionais, diria até que a maioria das linguagens mais populares, a indexação começar 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)]
```

Classes Date

Crie objetos com as datas do tri e tetracampeonatos mundiais do Brasil[19)]:

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

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

```
copa94 - copa70
```

Isto 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 [20] 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 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.

Dado Lógico

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 alguma 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:

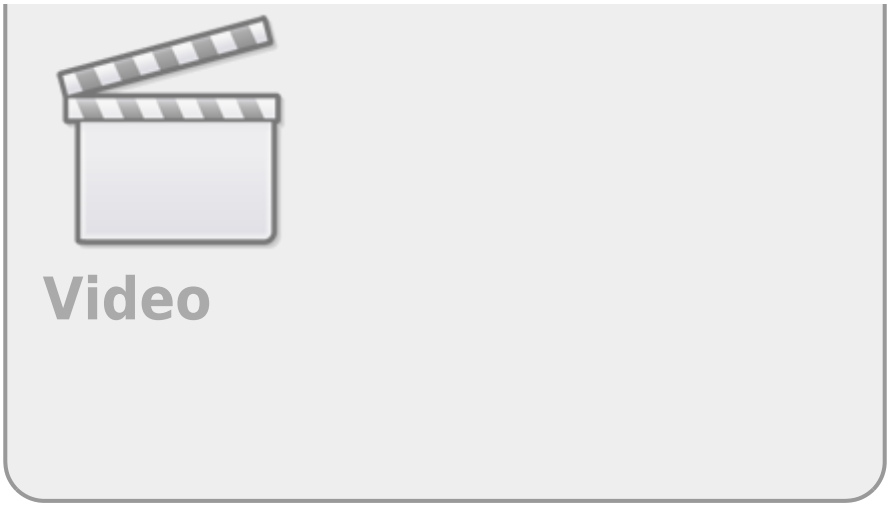
```
numero <- 1.23456789
numero < 1
numero > 1
## abaixo primeiro imprime o valor e depois faz o teste logico
print(numero, digits = 4) == numero
round(numero, digits = 4) == numero
numero > 1 & log(numero) > 1
numero > 1 | log(numero) > 1
```

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
```

Classe Fator



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 a 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, incluíndo 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.

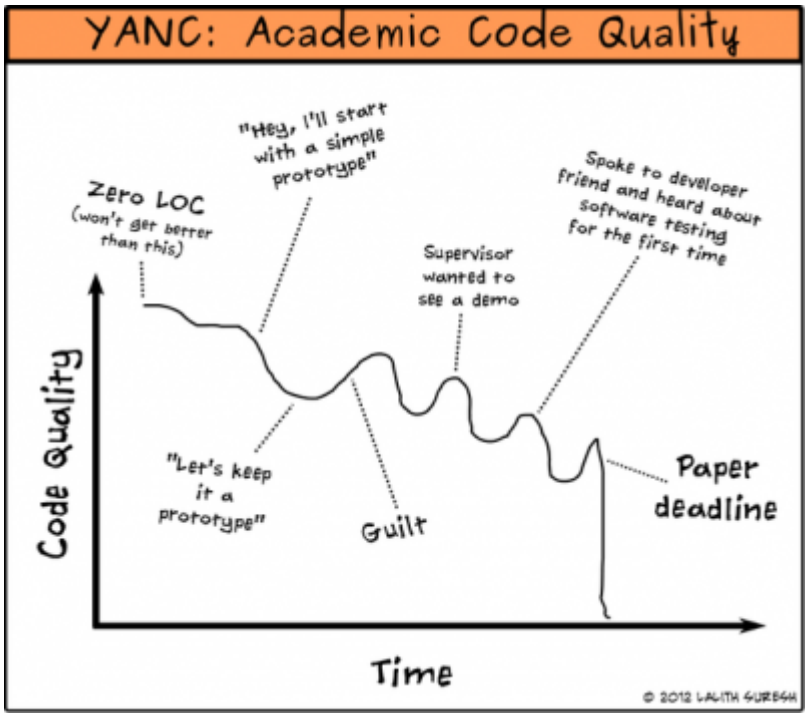
O Código

Antes de continuar a introdução aos conceitos básicos do R, vamos enteder 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**[21)], 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 abondonar a interface do R online que estavamos usando para rodar o código e vamos, a patir de agora, produzir script ou códigos!

Editor de Código

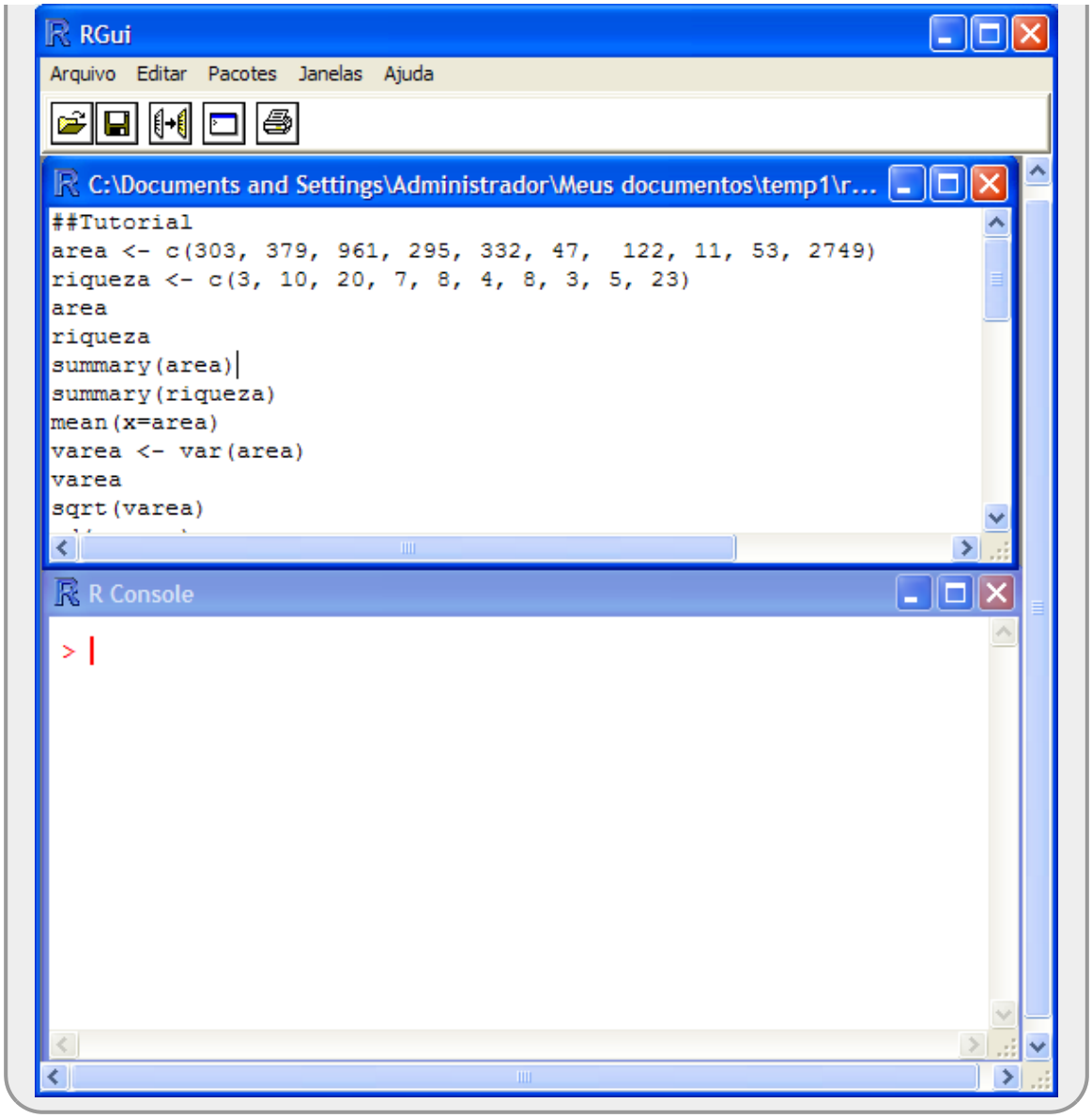


Um editor de código nada mais é do que um editor de texto puro como o bloco de notas do Windows. Algumas funcionalidades são bem vindas, como por exemplo, enviar a linha de código diretamente para o console do R sem a necessidade de copiar e colar.

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 uma 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[22)].

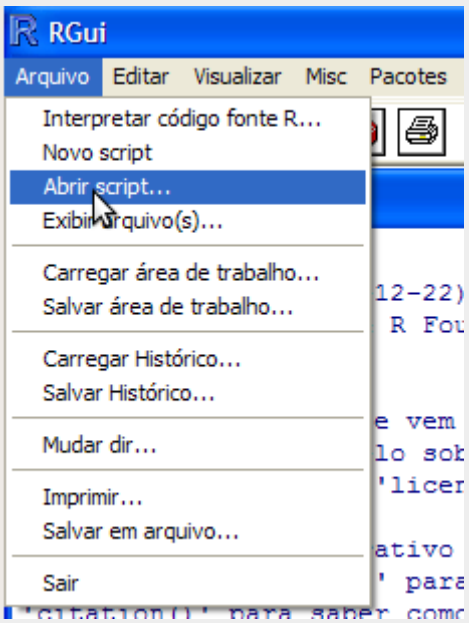
Para evitar confusão e perda de trabalho é importante digitar as informações que serão transmitidas ao R (linhas de código) no arquivo texto e ir passando esses comandos ao R. Uma boa prática também é comentar as linhas de código para

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 [23] conhecida do seu computador, associada a essa disciplina, com o nome e extensão `tutorial01.r` [24];
- Execute o R e abra o *script* que salvou, utilizando a opção do menu "Arquivo/Abrir script":



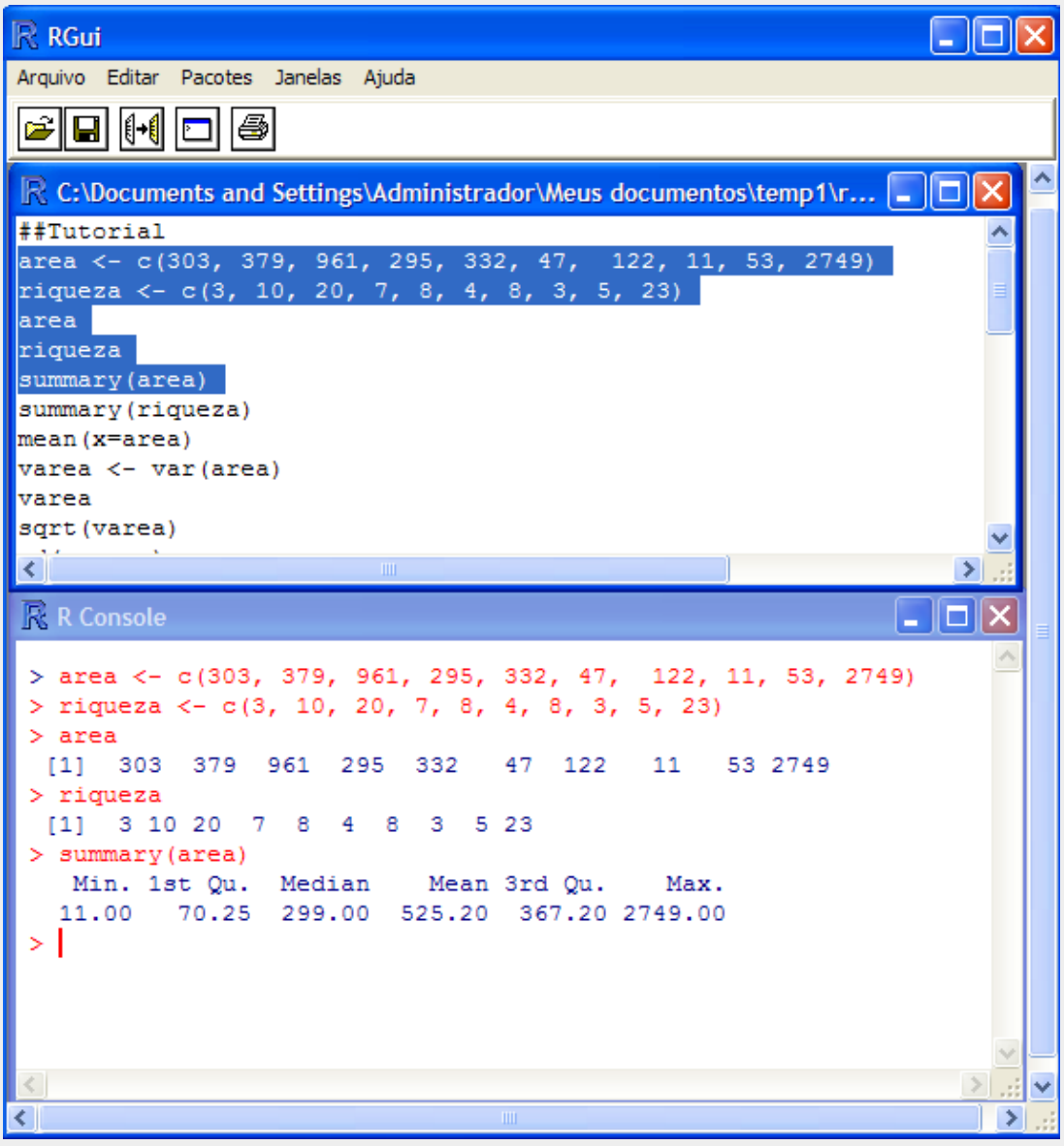
- Vá para a janela do *script*, coloque o cursor na primeira linha e tecle `Ctrl-r`. Faça o mesmo com as linhas seguintes;

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 mesagens de erro ao longo do processamento do script.



Comentários no código

Para fazer comentários no código, usamos o simbolo 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.

Agora vamos submeter o *script* salvo no nosso sistema de correção automática de código, chamado **notaR**.

Agora que já fez seu primeiro exercício no notaR. Siga para a aba de exercícios para seguir os execícios 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 sitema 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.

2020/08/12 06:11

Socorro

A primeira resposta ao aluno que perguntar algo é: **Você olhou a documentação do help?**. O usuário do R adora mostrar o que sabe e ajudar os iniciantes, mas isso não é o melhor caminho para aprender. Queremos ajudá-lo a resolver os problemas quando encontrá-los e para isso vamos indicar os caminhos e cabe ao aluno exercitá-los. A documentação do R, em um primeiro momento, é árida e muitas vezes difícil de entender. Leia com atenção, as respostas estão lá! Entretanto, é preciso familiaridade com a terminologia e um pouco de intimidade com a linguagem. Isso, só se adquire praticando!

Crie o bom hábito de ler a documentação de todas as funções que utilizar, **SEMPRE LEIA O HELP!**

help

Vamos olhar a documentação das funções que usamos acima. Há dois jeitos de fazê-lo, usando a função `help("nomedafunção")` ou o atalho `?"nomedafunção"`

```
help(c)
## a função c é uma das mais usadas, serve para concatenar valores em um
objeto.
```

```
?source
```

Encontrando as funções

Para acessar o help é necessário saber o nome da função. Caso não tenha ideia do nome de uma função básica, por exemplo, a raiz quadrada, a melhor opção é usar o RCard (R Reference Card), a segunda é buscar no google, como a letra R é pouco informativa use combinada com "`R language`" ou "`R CRAN`"

help.start

A função help.start abre a página de help do programa no seu navegador. Nela encontrará toda a documentação que é instalada junto com o R, além das funções dos pacotes que foram instalados na sua máquina. Para inicia-lo digite

```
## abrindo a ajuda do R no navegador
help.start()
```

Atividade

- Procure pelo pacote `graphics` e entre nos tópicos `graphics-package` e `plot`;
- Procure o pacote `sudoku`;
- Vá no site do CRAN e

Instalando e carregando pacotes

Existem mais de 7 mil pacotes no R-CRAN, Você já viu como buscá-los no repositório. Agora vamos entender dois conceitos básico na linguagem que muita gente confude: `instalar` e `carregar`. Para usar as ferramentas (geralmente `funções`) de um pacote é preciso antes que ele esteja no seu computador instalado. Para isso usamos a função `install.packages()`. Para usar um pacote instalado é necessário carregá-lo na sua sessão do R usando a função `library()`.

Pacotes

Pacotes são conjuntos de funcionalidades (funções e dados) distribuídos em conjunto para realizar tarefas específicas. Por exemplo, o pacote **vegan** carrega na sua área de trabalho (deixa disponível para uso) um conjunto de ferramentas para análises de dados de ecologia de comunidades. Para usar os pacotes disponíveis no R [25)]

é necessário entender as diferenças entre **baixar** (download) o pacote do repositório e **carregar** em sua área de trabalho. Para baixar algum pacote disponível no repositório CRAN do R é necessário utilizar o comando *install.packages()* com o nome do pacote entre "" dentro do parenteses[26)].

```
install.packages("vegan")
```

Outra forma é usar o menu da interface gráfica e selecionar. Siga as instruções: (1) selecione o repositório mais próximo (p.ex: *Brazil(SP1)*) e em seguida navegue na barra de pacotes e selecione o que deseja. Se não houver nenhuma mensagem de erro, significa que o download do pacote foi realizado com sucesso.

Caso o pacote esteja instalado ele aparecerá entre os hiperlinks da página de ajuda hipertexto da função *help.start()*. Entre na página do pacote e navegue pelas opções e funções que forem de seu interesse. Escolha uma função (decorana) e em seguida tente apresentar o ajuda dela pelo R:

```
help.start()
help(decorana)
```

A mensagem (ou algo similar): *"No documentation for 'decorana' in specified packages and libraries..."*, significa que a sua sessão do R não encontrou a documentação referente a função, apesar do pacote estar instalado em nosso computador. Isso aconteceu porque não carregamos a função em nossa área de trabalho, para cada projeto, precisamos carregar aqueles pacotes que vamos necessitar (normalmente nas primeiras linhas de comando do nosso código):

```
library(vegan)

example(vegan)
```

Podemos imaginar a nossa sessão do R como uma bancada de trabalho em uma oficina, cercada por vários armários que contém as ferramentas que precisamos para realizar uma tarefa. Dependendo da tarefa que vamos realizar (arrumar uma moto, construir uma cadeira...) abrimos os armários que contem as ferramentas necessárias à tarefa desejada e apenas esses (função *library()*). Caso não tenhamos as ferramentas necessárias para uma tarefa específica (consertar um relógio), precisamos ir na loja de ferramentas (repositório) e comprar conjunto de ferramentas de relojoeiro (função *install.packates("watch")* que vem em um armário que colocamos ao lado dos outros em nossa oficina.

2020/08/12 06:11

Operações Matemáticas

As operações matemáticas simples são apresentadas abaixo, reproduza os comandos para se familiarizar com o código:

```
4+5
4*5
2/4
2^4
```

As regras de precedência são as mesmas da aritmética básica:

```
2 + 4 * 5
(2 + 4) * 5
2 + 2^2 * 5
(2 + 2)^2 * 5

## tente fazer a operação abaixo em uma calculadora!!

1 - (1 + 10^(-15))
```

Operações matemáticas comuns

```
sqrt(9)   # Raiz Quadrada
abs(-1)  # Módulo ou valor absoluto
abs(1)
log(10)   # Logaritmo natural ou neperiano
log(10, base = 10) # Logaritmo base 10
log10(10)   # Também logaritmo de base 10
log(10, base = 3.4076) # Logaritmo base 3.4076
exp(1)       # Exponencial
```

Operações com Vetores

As operações algébricas com vetores apresentam duas características importantes: **equivalência de posição** e **ciclagem do vetor de menor tamanho**. A **equivalência** define que os valores serão operados respeitando suas posições equivalentes entre dois ou mais vetores de mesmo tamanho. No caso de vetores de tamanhos diferentes, há a ciclagem dos elementos do vetor de menor tamanho. Nesse último caso, se o vetor maior é múltiplo do menor, não há nenhuma mensagem na operação. Caso não possa ciclar todos os elementos o mesmo número de vezes, o R opera parte do vetor menor no último ciclo e avisa com uma mensagem de alerta: `longer object length is not a multiple of shorter object length`.

Estude os resultados dos comandos nos três exemplos abaixo para entender as operações com vetores.

```
a <- 1:10
b <- -(1:10)
a+b
sum(a, b)
```

Aqui temos a característica da **equivalência**. Os vetores de mesmos tamanhos, quando operados algebricamente, operam os valores da posições equivalentes.

```
a <- seq(from = 0, to = 1, length = 9)
b <- seq(from = 0, to = 0.5, length = 9)
a
b
```

```
a/b
b/a
1+b/a
(1+b)/a
```

Nesse último bloco de código temos, além da equivalência entre as posições de vetores de mesmo tamanho, algumas operações algébricas que retornam palavras reservadas no R: `NaN`, `Inf` e `-Inf`: O `NaN` significa `not a number`, ou seja um valor numérico indefinido ou irrepresentável. Operações como `0/0` ou `√-1` retornam `NaN` . Dividir um número real por `0` retorna `Inf` ou `-Inf` [27)].

```
a <- rep( c(1, 2), each = 3 )
length(a)
b <- rep( c(3, 4), 6)
length(b)
a+b
```

```
a <- rep( c(1, 2, 3), each = 3 )
length(a)
b <- rep( c(3, 4), length = 7)
length(b)
a+b
```

Nestes blocos temos o princípio da ciclagem. Ao realizar operações com vetores de tamanhos diferentes, o vetor menor é ciclado, ou seja, reutilizado para a operação. Note que se o tamanho dos vetores for múltiplo todos as posições do vetor serão utilizada um mesmo número de vezes e o R não retorna nenhuma mensagem[28)]. Se os vetores não forem múltiplos a operação é realizada porém com uma mensagem de aviso.

2020/08/12 06:11

Gerando dados

Vimos que a função `c()` serve para concatenar valores em um objeto. Vamos usá-la para gerar alguns objetos e introduzir a função `ls()` que lista todos os objetos criados na nossa sessão até o momento:

```
A1 <- c(1,2,3)
A2 <- c(10,20,30)
b <- c(A1,A2)
ls()
```

Consulte a página de ajuda da função ls:

```
help(ls)
```

Onde você verá a explicação para o argumento `pattern`. Execute, então, este comando:

```
ls(pattern="A")
```

Para mudar os nomes de objetos e apagar os antigos, experimente:

```
a.1 <- A1
a.2 <- A2
ls()
rm( list=c("A1","A2") )
ls()
```

Que tem o mesmo efeito de:

```
rm(list=ls(pattern="A"))
```

Ou de

```
rm(A1,A2)
```

Verifique!

Criando sequencias de valores

Uma das vantagens da linguagem R é que ela opera em vectores, como vimos acima nas funções matemáticas. Para gerar sequencias de valores podemos usar o `c()`, mas existem outras funções básicas para agilizar a geração de sequências. Execute as linhas de código abaixo e veja a documentação relacionada a cada função:

```
1:10
(1:10) * 10

seq(from=0, to=100, by=10)
seq(from=0, to=100, len=10)

rep("A", times=10)
rep(c("A", "B", "C"), times=10)
rep(c("A", "B", "C"), each=10)

## nao criamos nenhum objeto até agora!
## vamos guardar a próxima sequência em um objeto:

seqABC <- rep(c("A", "B", "C"), times=c(2,3,5))
seqABC

paste(seqABC, 1:4, sep="_")
```

Entendeu essa última função? Veja a documentação do `paste()`.

Lógica vetorial

Vetores são a estrutura básica do R, procure sempre raciocinar em vetores e operações vetorias. Por

exemplo, para calcular 2 elevado a expoentes de 0 a 10, a lógica de operação escalar é :

```
2^0
2^1
2^2
2^3
...
```

A lógica da operação vetorial é:

```
2^(0:10)
```

ou

```
2^seq(from=0,to=10, by=1)
```

Tipos de vetores

Já fizemos vetores de duas naturezas: numéricos e caracteres. A diferença entre os dois é que o numérico podemos operar matematicamente, caracteres apenas manipular com funções específicas para isso. Veja o código abaixo e veja se entende o que acontece:

Classes Date

Crie objetos com as datas do tri e tetracampeonatos mundiais do Brasil[29]:

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

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

```
copa94 - copa70
```

Isto 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 [30] 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 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.

2020/08/12 06:11

numbers, integers, characters & as.Date

Além dos vetores

Além dos vetores, outros tipos de objetos são muito utilizados no R, os mais importantes são: (1) matrizes; (2) data frames e (3) listas. As funções para criar esses objetos são: `matrix()`, `data.frame()` e `list()`. Matrizes e data frames são estruturas retangulares de dados, a diferença entre elas é que matriz só recebe um tipo de natureza de dados (números, caracteres, ou vetor lógico que veremos mais a frente). A lista pode acomodar qualquer natureza de dados em diferentes formas.

Veja o código abaixo para entender esses objetos:

```
a=1:5
a
b=factor(rep(c("a","b","c"), each=3))
b
c=data.frame(sec=c("XIX", "XX", "XXI"),inicio=c(1801,1901,2001))
```

```
c
d = matrix(round(runif(36,0,6)),ncol=8)
d
minha.lista = list(um.vetor=a, um.fator=b, um.data.frame=c, uma.matriz= d)
minha.lista
```

Atributos dos objetos

Os objetos possuem atributos, dois deles já vimos: (1) a classe a que pertence e (2) a natureza das variáveis que o compõem.

Estrutura de um objeto

Essa é uma das funções mais importantes para diagnosticar um objeto. Fornece informações importantes sobre sua estruturação. Antes de rodar o código abaixo, verifique se os objetos estão na sua área de trabalho.

```
## verficando se quais objetos estão presentes na área de trabalho:

ls()

## verificando a estrutura dos objetos

str(minha.lista)
str(minha.lista$um.data.frame)
str(d)
```

Dimensões de um objeto

As dimensões de um objeto são um atributo importante. Nos indica quanta informação está contida e como esta estruturada. Ela é fornecida pela `str()`, mas pode ser acessada pelas funções `length()` e `dim()`.

```
length(b)
length(minha.lista)
dim(minha.lista$um.data.frame)
```

Extraindo e Modificando

Para modificar algum elemento de um objeto, use a lógica **visualizar & modificar**. Primeiro visualize o elemento, certificando-se que é ele que deseja modificar ou remover. O mesmo código para visualizar é utilizado em associação à atribuição (`= < -`) para modificar o elemento desejado.

Indexadores

O indexadores definem a posição de um elemento em um objeto e a forma de usá-los depende da classe de objeto que pretende manipular.

```
########################
## Indexacao "[]" &  "$"
########################
x=LETTERS[1:6]
x
x[1]
x[1:3]
x[c(1,1,3,5)]
x[-2]
x[-c(2,4)]

### INDEXAÇÃO COM LÓGICA #####
ALTURA=c(1.85, 1.78, 1.92, 1.63, 1.81, 1.55)
ALTURA
SEXO = factor(rep(c("M","F"),each=3))
SEXO
PESO <- c(80, 100, 115,  70,  65,  50)
PESO

## OPERAÇÕES LÓGICAS
ALTURA >= 1.8
SEXO=="M"

homens.altos <- (ALTURA > 1.8) & (SEXO=="M")
homens.altos

PESO[homens.altos]


### ALTERANDO SUCONJUNTOS ######

ALTURA
ALTURA > 1.8
ALTURA[ALTURA > 1.80]
ALTURA[ALTURA > 1.80] <- c(1.86, 1.93, 1.82)
ALTURA

### CRIANDO UM DATA FRAME ######

pessoas <- data.frame(alt = ALTURA, sex = SEXO)
pessoas
pessoas$peso
pessoas$peso <- c(80, 100, 115,  70,  65,  50)
pessoas
pessoas$peso[2]
```

```
pessoas[2, 3]
pessoas[, "sex"]
pessoas[c(1, 4, 5), "alt"]
```

Lendo dados e salvando resultados

read.table

A melhor forma de ler um arquivo de dados é tranformá-lo em arquivo texto onde os campos são separados por algum caractere, o mais comum é virgula e ponto-virgula (`.csv`); ou tabulação (`.txt`). Um cuidado que precisamos ter ao ler arquivos é com relação ao simbolo de decimal. Em português, usamos a `virgula`, o que pode confundir com a separação de caractere do arquivo `.csv` e portanto precisa ser informado à função `read.table`.

Veja os principais opções de argumentos da função read.table:

argumento	estado	significado
header	TRUE	primeira linha é nome das variáveis
sep	"\t"	separador tabulação
sep	" "	separador espaço
sep	";"	separador ponto-virgula
dec	"."	simbolo ponto
dec	","	simbolo vírgula
as.is	TRUE	não transforma variável em fator

Exercício

- Salve o arquivo Conjunto de Dados: Insetos galhadores (Simulação) no seu diretório de trabalho;
- Abra-o no Excel;
- Salve como texto, separado por tabulação;
- Leia o arquivo com `read.table` e guarde no objeto `galha`;
- Verifique se a estrutura de `galha` está correta.

Salvando dados

Para salvar um conjunto de dados há dois formatos básicos no R. Podemos salvar o arquivo como texto, usando a função `write.table` ou como arquivo de dados do R usando a função `save`. Esse último tem o formato `.RData` e a desvantagem de não poder se lido em outro programa. A vantagem do `.RData` é que pode guardar estruturas complexas de de dados e resultados, inclusive salvar todos os objetos de uma área de trabalho.

Exercício

- Verifique os objetos presentes na sua área de trabalho usando a função `ls()`;
- Confirme que há o objeto chamado `pessoas`, caso não haja retorne ao tópico extraindo_e_modificando para recriá-lo;
- salve o objeto com a função `save`, consulte o help para entender os argumentos:

```
save(pessoas, file="pessoas.RData")
```

- tente abrir o aquivo criado
- faça o mesmo usando o `write.table` e crie o arquivo `pessoas.csv`, separado por tabulação e sem nome de linhas [31)]
- abra esse arquivo no Excel

save.image

Para salvar todos os objetos de sua área de trabalho, use a função `save.image`.

- salve seu código do script no arquivo `codeAula1.r`;
- salve todos os objetos da sua área de trabalho no arquivo `minicurso1.RData`;
- * remova todos os objetos da sua área de trabalho;

```
rm(list = ls())
```

- feche o R;
- abra o R pelo arquivo criado acima `minicurso.RData`;
- verifique que objetos existem na área de trabalho;
- feche novamente o R;
- abra o seu script `codeAula1.r` em uma sessão vazia do R;
- rode todo o script acima na sessão do R;
- verifique os objetos da sua área de trabalho;

Não se perca!

Respire fundo!
Sabemos que é muita coisa para aprender no mesmo dia.
É preciso praticar!

Cada um dos tópicos que vimos até aqui está explicado em mais detalhes no material do curso. Para cada um deles temos um tutorial, um capítulo da apostila online e um série de exercícios. Uma boa estratégia é estudar os tópicos utilizando a sequência: tutorial, apostila e exercício!

Vamos revisar alguns conceitos visto até agora. Agora é com o professor!

Manipulando dados

Funções para usar funções

Uma maneira inteligente de usar o R é automatizar tarefas usando as funções que executam outras funções nos dados, dependendo da classe do objeto. A família de funções `apply` opera dessa forma. Vamos testá-la, usando o data frame `pessoas`. Siga o script abaixo tentando entender cada passo, comente seu código, além dos comentários já incluídos para lembrar do que as funções fazem. Lembre-se **sempre olhe a documentação das funções no** `help`.

```
#################
## familia apply
#################

ls()
str(pessoas)

#altura media  por sexo
tapply(X = pessoas$alt, INDEX= pessoas$sex, FUN= mean)
tapply(X = pessoas$alt, INDEX= pessoas$sex, FUN= sd)
## quantos homens e mulheres?
table(pessoas$sex)
table(pessoas$sex, pessoas$alt>1.70)
```

```
## qual a media de altura e peso (independente do sexo)
apply(pessoas[,c(1,3)], 2, mean)

## e essa ultima?
aggregate(x = pessoas[,c(1,3)], by = list(pessoas$sex), FUN = mean)
## veja o help da funcao
```

Gráficos

O R é muito bom para fazer gráficos, aqui vamos apresentar algumas funcionalidade do pacote básico `graphics`. Existem outros pacotes que produzem gráficos mais refinados por padrão, porém, quase tudo pode ser feito apenas com as funções do pacote gráfico. Três conceitos são importantes para controlar e formatar o seu gráfico: `funções de alto nível`, `funções subordinadas` e `parâmetros gráficos` . Existem várias opções de pacotes para a elaboração de gráficos no R. Aqui vamos apresentar a lógica do pacote `graphics` que é carregado automaticamente na sessão do R. Outros pacotes como `grid`, `latice` e `ggplot2` estão presentes na distribuição de instalação do R, mas não são carregados automaticamente. Apesar de ser um pacote de funções básicas é possível fazer gráficos muito elaborados apenas usando as ferramentas contidas no `graphics`. Três elementos são importantes para entender a lógica do `graphics`: (1) funções de alto nível; (2) funções subordinadas e (3) parâmetros graficos.

Funções de alto nível

As funções de auto nível são responsáveis pela estruturação básica de um gráfico, abrem o dispositivo de apresentação (a janela gráfica) e constroem os elementos básicos do gráfico. As funções subordinadas incluem elementos em um gráfico ativo. Veja a diferença rodando o código abaixo:

```
##############################
## graficos
## funcoes de alto nivel: abrem o dispositivo de tela
example(plot)
example(contour)
example(hclust)
### demos
demo(image)
demo(persp)
#########################
## funcoes subordinadas
example(points)
example(segments)
example(rect)
example(axis)
example(colors)
####################
```

Parâmetros do dispositivo gráficos

Os parâmetros do dispositivo gráfico por sua vez muda a estrutura do dispositivo gráfico antes de abrí-lo. Deve ser usado antes rodar uma função gráfica de alto nível. Para interagir com os parâmetros gráficos utilizamos a função `par()`. Veja o help dessa função:

```
?par
```

UM GRÁFICO

Vamos usar o código de um gráfico [32] para entender como elaborar um gráfico no R. Primeiro, vamos mudar os parâmetros do gráfico antes de iniciar, mudamos cada parâmetro em um linha para que possa buscar no help informações para entender o que o parâmetro significa e completar os comentários em cada linha.

```
########## dados ####################
riqueza <- c(15,18,22,24,25,30,31,34,37,39,41,45)
area <- c(2,4.5,6,10,30,34,50,56,60,77.5,80,85)
box <- c(10,13,12,14,15,12,14,15,20,23,22,21,26,27,28,25)
z <- c(50,42,33,29,25,19,17,15,10,11,8,9)
samples <- rep(1:4,each=4)
model <- lm(riqueza~area)
model1 <- glm(z~area, poisson)
#####################################
# mudando parametros do grafico
par(mfrow=c(2,1)) # divide o dispositivo grafico em paineis: duas linhas e
uma coluna
par(mar=c(3,5,2,2)) # complete aqui o que esse comando significa
par(cex.axis=1.3) # ...
par(cex.lab=1.5) # ...
par(family="serif") # ...
par(las=1) # ...
par(tcl=0.3) # ...
par(mgp=c(2,0.3,0)) # ...
par(bty="u") # ...
```

Siga fazendo o gráfico utilizando o código abaixo procurando entender cada função e argumento utilizado. Comente as linhas.

```
plot(riqueza~area, xlab="Area (ha)", ylab="Riqueza de especies\n de aves",
cex=1.5, pch=16,  ylim=c(8, 50), xaxp=c(0,100,4), col="firebrick3")
text(10,50, "a", cex=1.8)
abline(model, lwd=1.5,col="firebrick3")
#par(new=TRUE)
points(z~area, cex=1.5, pch=17, col="mediumblue")
axis(4)
xv<-seq(0,100,0.2)
```

```
yv<-exp(predict(model1,list(area=xv)))
lines(xv,yv, lwd=1.5, lty=2, col="mediumblue")
```

Interação com o gráfico

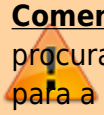
Algumas [33)] funções são interativas. A funcao `locator` retorna as coordenadas cartesianas da posição do cursor do mouse quando o botão é apertado. Note que o R fica esperando que esse evento ocorra e ele deve acontecer no painel ativo da janela gráfica no espaço entre os eixos x e y. Vamos utilizá-la para colocar uma legenda na janela ativa do gráfico.

```
xy <- locator(1)

## clique em uma posicao no espaco entre os  eixos x e y do painel ativo

legend(xy, legend=c("sub-bosque", "matriz"), pch=c(16, 17),
col=c("firebrick3","mediumblue") , bty="n")
```

Vamos continuar o gráfico, agora passando para o segundo painel da janela. Isso ocorre automaticamente quando utilizarmos uma função de auto nível. Antes disso, vamos mudar os parâmetros desse novo painel usando a função `par`.



Comente o código Para fixar e facilitar quando estiver procurando algum dos comandos utilizados nessa atividade para a construção do seu próprio gráfico, comente as linhas de código!

```
par(mar=c(5,5,0.5,2))
par(bty="l")
boxplot(box~samples,names= c("", "", "", ""),col="grey")
mtext(c("Tipo 1", "Tipo 2", "Tipo 3", "Tipo 4"),side=
1,cex=1.3,line=0.3,at=c(1,2,3,4))
mtext("Diversidade genética", side=2, cex=1.5, line=2.5, las=0)
mtext("Morfotipo", side=1, cex=1.7, line=3)
text(0.7,28, "b", cex=1.8)
```

Modelos Lineares

```
############################
##### lm() ANOVA
############################
## dados
## producao agricula em diferentes tipos de solo
```

```r
are=c(6,10,8,6,14,17,9,11,7,11) # solo arenoso
arg=c(17,15,3,11,14,12,12,8,10,13) # solo argiloso
hum=c(13,16,9,12,15,16,17,13,18,14) # solo humico
planta=c(are,arg,hum) # juntando os dados
#### criando um fator que representa os solos ###
fator.solo=as.factor(rep(c("are", "arg", "hum"),each=10))
#############################
aov.solo<-aov(planta~fator.solo) # funcao para fazer uma anova (aov)
summary(aov.solo)
### mesmo modelo usando o lm()
lm.solo<-lm(planta~fator.solo)
summary(lm.solo)
anova(lm.solo) ## funcao para calcular a tabela de anova do modelo
#################################
##### regressao linear
#################################
library(MASS) # carrega o pacote MASS
data(Animals) # ativa os dado Animals do pacote MASS
str(Animals) # 28 observacoes de tamanho do corpo e peso do cerebro de
diferentes animais
plot(brain~body,data=Animals)
## parece nao haver relacao!
plot(brain~body,data=Animals, log="xy")
## agora sim!
plot(log(brain) ~ log(body), data = Animals)
### criando um modelo de log(brain) ~ log(body)
anim.m1 <- lm(log(brain)~log(body),data=Animals)
## colocando a linha do modelo no grafico dos dados
abline(anim.m1, col="red")
## resultados do modelo
summary(anim.m1)
```

1)
criado por — *Alexandre Adalardo de Oliveira* *2015/10/05* para o mini-curso da semana temática da biologia - IBUSP
2)
menu da barra à esquerda da página
3) 14)
,
funções também são uma classe de objetos
4) 15)
,
O caracter ? funciona como um atalho para essa função
5) 16)
,
arredondamentos podem ser danosos, principalmente para cálculos sequenciais e números pequenos
6) 17)
,
alguns links não funcionam, mas a maior parte sim
7) 18)
,
para atribuir o resultado a um objeto e ao mesmo tempo mostrar na tela, utilize parênteses iniciando e fechando a linha de comando
8) 19) 29)
, ,
fonte: FIFA
9) 20) 30)
, ,

procedimento de transformar uma classe em outra
[10) 21)]
,
Console é a interface de interação com o interpretador da linguagem: recebe o comando, envia ao interpretador e retorna a resposta. O que vinhamos usando no início desse tutorial é um interpretador online do R
[11) 22)]
,
quando a tarefa solicitada é a representação de um gráfico, uma nova janela é aberta, um dispositivo gráfico.
[12) 23)]
,
diretório de trabalho é o nome técnico desta pasta para o R
[13) 24)]
,
quando o sistema operacional não mostra a extensão dos arquivos é preciso configura-lo para que seja apresentado
[25)]
“ Currently, the CRAN package repository features 14270 available packages.” — *Alexandre Adalardo de Oliveira 2019/05/22 14:14* “Currently, the CRAN package repository features 5217 available packages.” — *Alexandre Adalardo de Oliveira 2014/02/17 14:31*
[26)]
a princípio todas as palavras que escrevemos sem aspas no R ele busca como sendo objetos presentes em nossa área de trabalho ou pacotes carregados ou instalados
[27)]
apesar de divisão por 0 não ser definida matematicamente, o R considera que o valor a ser dividido é muito próximo de 0
[28)]
lembre-se de sempre inspecionar os objetos que você criar!
[31)]
veja o help!
[32)]
autoria de Cristina Banks
[33)]
poucas no pacote `graphics`