PROFISSÃO POLICIAL

Informática

Professor Fabio Rosar

INFORMÁTICA
Professor Fabio Rosar

Sumário

Programação em Python e R

1 PROGRAMAÇÃO EM PYTHON

Python é uma das linguagens orientadas a objetos mais usadas atualmente e é um interpretador de comandos gratuito. Python utiliza como extensão de arquivo **.py** ou **.pyw** ou **.pyi**.

Assim como outros ambientes de desenvolvimento, o Python possui diversas interfaces e as mais conhecidas são o Pycharm e o IDLE. Nos exemplos utilizarei o IDLE.

Algumas características:

- Tipos de variáveis: string, int e float
  - `String`: texto
  - `Int`: números inteiros
  - `Float`: números reais

- Nem sempre é obrigatório declarar a variável. Ao ser criada, automaticamente o ambiente já reconhece, na maioria dos casos.

- É case sensitive: diferencia maiúsculas/minúsculas nos comandos e objetos.

- A variável tem que iniciar com letra, não pode iniciar com números.

- Os objetos/variáveis não podem ter o mesmo nome de funções e outros argumentos como os descritos abaixo.

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Figura 1 - Exceções para nome de variável/objeto

1.1 Comandos

A partir desse momento você deve estar pensando: certo, e como vou praticar?

Não precisa instalar absolutamente nada!! Siga os passos abaixo e vou te ensinar como chegar numa console python online e praticar os comandos. Mais adiante, neste material, ensino como baixar e instalar o Python (sempre é melhor), caso queira.

1. Acesse https://www.python.org/shell/ e aguarde.

2. O console virtual será carregado e você pode iniciar a digitação dos comandos quando aparecer “>>>”

3. A qualquer momento você pode digitar **quit()** para sair da console. Para voltar ao início da console aperte F5 no navegador.

4. Alguns comandos podem funcionar somente no IDLE (console instalável em Windows e Linux)

`>>>`	Prompt de comando para escrita
`"`	Utilizado em argumento de texto
`quit()` `exit()`	Fechar o console
`help()` `help(objeto)`	Ajuda
`=`	Atribui valor a um objeto
`print(valor)`	Imprimir/mostrar na tela
`type(variável)`	Indica o tipo da variável
`input (valor)`	Atribuição a variável por entrada do usuário
`dir()`	Mostra os objetos
`Del(objeto)`	Excluir um objeto
`ALT+P`	Repete o último comando
`CTRL+F6`	Reinicia o IDLE
`F5`	Executa uma função

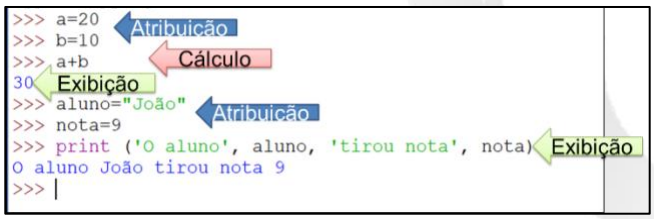
1.2 Operadores

Abaixo a ordem dos precedentes dos operadores matemáticos.

( )	Parênteses
* *	Potência
* /	Multiplicação Divisão
+	Adição
−	Subtração
/ /	Parte inteira da divisão
%	Resto da divisão

O “+” entre argumentos de “texto”, faz a operação de concatenação (juntar).

Antes dos exemplos, vamos entender a tela do IDLE.



Atribuição: quando um valor é “inserido” em um objeto. No exemplo acima, **a** e **b** são objetos e recebem, respectivamente, os valores **20** e **10**.

ATENÇÃO: se for praticar, o símbolo “>>>” representa apenas um início de entrada de linha de comando. Não digite junto do comando.

Ex: `>>> (7+3)*10`

O comando é apenas **(7+3)*10**

Exibição: o que retorna para o usuário, na tela.

Exemplo 1:

```
>>> (7+3)*10
100
>>> 7+3*10
37
>>>”policia"+"federal"
'policiafederal'
>>>
```

Exemplo 2:

```
>>> 4**2
16
>>> 7//3
2
>>> 7%3
1
>>>
```

Se tiver multiplicação e divisão na mesma operação, resolve primeiro quem está mais à esquerda.
Exemplo:

```
>>> 8*4/2
16.0
>>> 4/2*8
16.0
>>>
```

Os espaços entre os valores e os operadores podem existir ou não.
Exemplo:

```
>>> 8     *     2
16
>>> 4     +     6
10
>>>
```

Operadores lógicos

>	Maior que
<	Menor que
==	Igual a
>=	Maior ou igual que
<=	Menor ou igual que
!=	Diferente de

Exemplo:

```
>>> 10!=20
True
>>> 20>10
True
>>> 30<10
False
>>> 10==20
False
>>>
```

Os valores que trabalhamos acima podem ser atribuídos a objetos. Os objetos são criados quando eles recebem sua primeira atribuição. Na atribuição não existe resposta na tela para o usuário.

Exemplo1:

```
>>> a=20
>>> b=30
>>> c=a+b
>>> c
50
>>> a=10
>>> c
50
>>> c=a+b
>>> c
40
>>>
```

No exemplo acima, primeiramente, o objeto **a** recebeu o valor **20** e ele estava em uma operação em **c**. Depois de um novo valor assumir **a**, **c** deverá ser "reatribuído" para trazer o resultado correto.

Exemplo2:

```
>>> nome="João"
>>> nota=9
>>> print("O aluno ",nome," tirou nota ",nota)
O aluno  João  tirou nota  9
>>>
```

O objeto **nome** e **nota** foram criados e utilizados em uma frase com o comando **print**.

1.3 Objetos, vetores e matrizes

Vou utilizar uma tela do Excel para iniciar a explicação de vetores e matrizes.

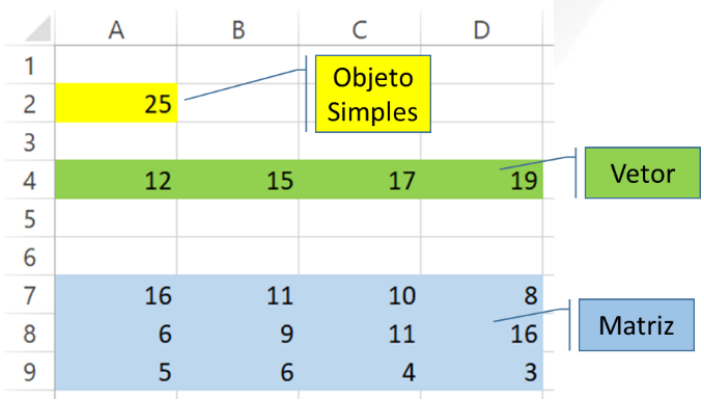


Figura 2 - Vetores e Matrizes

A2 é um objeto simples que recebe o valor 25.

O intervalo de A4:D4 é um vetor (uma sequência de valores em linha)

O intervalo de A7:D9 é uma matriz (tabela com linhas e colunas)

Vetor: vários valores em apenas uma dimensão (uma linha). Também pode ser chamado de lista.

No Python o vetor é criado com "[ ]" e os valores, dentro dos colchetes, são separados por vírgulas.

Ex1:

```
>>> notas=[]
>>> notas
[]
>>>
```

Criei um vetor, sem valores, vazio.

Ex2:

```
>>> notas=[5,7,9]
>>> notas
[5, 7, 9]
>>>
```

Vetor **notas** com valores 5, 7, 9

Ex3:

```
>>> notas=[8,6,notas]
>>> notas
[8, 6, [5, 7, 9]]
>>>
```

Nesse exemplo adicionei um vetor dentro de outro.

Os vetores podem assumir valores de diversos tipos de dados.

```
>>> casa=[40,"porta",20,"janela","cozinha"]
>>> casa
[40, 'porta', 20, 'janela', 'cozinha']
>>>
```

Mostrar valores por índices [] do vetor

```
>>> casa=[40,"porta",20,"janela","cozinha"]
>>> casa
[40, 'porta', 20, 'janela', 'cozinha']
>>> casa[0]
40
>>> casa[4]
'cozinha'
>>> casa[3]
'janela'
>>> casa[0:3]
[40, 'porta', 20]
>>>
```

Nesse exemplo, com o comando casa[0] solicitei para trazer como resposta o índice **0** do vetor **casa**.

casa[0:3] retorna os índices de **0** a **3** do vetor casa.

Índice de vetores, dentro de vetores:

```
>>> vetor1=[3,7,6,9]
>>> vetor2=["Ana","Fred","André","Bea"]
>>> total=[vetor1,vetor2]
>>> vetor1
[3, 7, 6, 9]
>>> vetor2
['Ana', 'Fred', 'André', 'Bea']
>>> total
[[3, 7, 6, 9], ['Ana', 'Fred', 'André', 'Bea']]
>>> total[1]
['Ana', 'Fred', 'André', 'Bea']
>>> total[0]
[3, 7, 6, 9]
>>>
```

Note que a posição **1** do vetor **total** é o **vetor1**. O **vetor1** foi atribuído a **total**.

É possível somar dois vetores dentro de um. O resultado é um vetor só:

```
>>> total=vetor1+vetor2
>>> total
[3, 7, 6, 9, 'Ana', 'Fred', 'André', 'Bea']
>>>
```

Se quisesse apenas mostrar na tela, e não fazer atribuição, poderia ser feito da seguinte forma:

```
>>> vetor1+vetor2
[3, 7, 6, 9, 'Ana', 'Fred', 'André', 'Bea']
>>>
```

É possível multiplicar(repetir) os valores de um vetor.

```
>>> vetor1
[3, 7, 6, 9]
>>> vetor1*3
[3, 7, 6, 9, 3, 7, 6, 9, 3, 7, 6, 9]
>>>
```

Para verificar se algum valor está no vetor:

```
>>> vetor2
['Ana', 'Fred', 'André', 'Bea']
>>> "fred" in vetor2
False
>>> "Fred" in vetor2
True
>>>
```

O primeiro resultado foi **False**, será que você descobre o motivo? Isso mesmo, o Python é Case Sensitive, ou seja, **fred≠Fred**.

É possível adicionar itens a um vetor usando o **append()**

```
>>> casa=[40,"porta",20,"janela","cozinha"]
>>> casa
[40, 'porta', 20, 'janela', 'cozinha']
>>> casa.append("sala")
>>> casa
[40, 'porta', 20, 'janela', 'cozinha', 'sala']
>>>
```

Nesse caso, o valor é inserido ao fim do vetor.

```
>>> casa.append(["banheiro", "escada"])
>>> casa
[40, 'porta', 20, 'janela', 'cozinha', 'sala', ['banheiro',
'escada']]
>>>
```

Adicionamos mais dois valores ao vetor casa.

É possível também adicionar itens a um vetor com o **insert()** diretamente a um índice.

```
>>> casa.insert(0,10)
>>> casa
[10,  40,  'porta',  20,  'janela',  'cozinha',  'sala',
['banheiro', 'escada']]
>>>
```

Inseri o valor 10 ao índice 0 do vetor casa.

```
>>> casa.insert(1,"Trinco")
>>> casa
[10, 'Trinco', 40, 'porta', 20, 'janela', 'cozinha', 'sala',
['banheiro', 'escada']]
>>>
```

É possível clonar um vetor **[:]**.

```
>>> x=[5,6,7,8,9]
>>> x
[5, 6, 7, 8, 9]
>>> y=x[:]
>>> y
[5, 6, 7, 8, 9]
>>>
```

Clonei o vetor **x** em **y**.

Para remover um item do vetor, pode ser utilizado o método **pop()**

```
>>> casa
[10, 'Trinco', 40, 'porta', 20, 'janela', 'cozinha', 'sala']
>>> casa.pop()
'sala'
>>> casa
[10, 'Trinco', 40, 'porta', 20, 'janela', 'cozinha']
>>>
```

Exclui o último índice.

```
>>> casa
[10, 'Trinco', 40, 'porta', 20, 'janela', 'cozinha']
>>> casa.pop(3)
'porta'
>>> casa
[10, 'Trinco', 40, 20, 'janela', 'cozinha']
>>>
```

Exclui o índice 3 do vetor casa.

Pode se utilizar também o **del[]**

```
>>> casa=[40,"porta",20,"janela","cozinha"]
>>> casa
[40, 'porta', 20, 'janela', 'cozinha']
>>> del casa[1]
>>> casa
[40, 20, 'janela', 'cozinha']
>>>
```

Para excluir um valor conhecido, que o índice não é conhecido, de um vetor, pode ser utilizado o método **remove()**

```
>>> casa
[10, 'Trinco', 40, 20, 'janela', 'cozinha']
>>> casa.remove("janela")
>>> casa
[10, 'Trinco', 40, 20, 'cozinha']
>>>
```

Removi o valor “janela”.

Para ordenar um vetor deve-se usar o método **sort()** e para inverter essa ordem deve-se utilizar o **reverse()**

```
>>> valores=[20,10,30,7,90,8]
>>> valores
[20, 10, 30, 7, 90, 8]
>>> valores.sort()
>>> valores
[7, 8, 10, 20, 30, 90]
>>>
```

Criei um vetor valores com alguns números e ordenei.

```
>>> paises=["Brasil", "Inglaterra", "Argentina"]
>>> paises.sort()
>>> paises
['Argentina', 'Brasil', 'Inglaterra']
>>> paises.reverse()
>>> paises
['Inglaterra', 'Brasil', 'Argentina']
>>>
```

Após criar o vetor **paises** usei o **sort** e o **reverse**.

Listas aninhadas (uma embaixo da outra) são frequentemente utilizadas para representar **matrizes**. Por exemplo, a matriz:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
>>> matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> matriz
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> matriz[2]
[7, 8, 9]
>>> matriz[2][1]
8
>>> matriz[0]
[1, 2, 3]
>>> matriz[0][0]
1
>>>
```

1.4 Funções

Função **list()** cria uma lista de sequências de objetos onde cada objeto pode ser referenciado por um índice.

```
>>> list("informática")
['i', 'n', 'f', 'o', 'r', 'm', 'á', 't', 'i', 'c', 'a']
>>> x=list("informática")
>>> x[2]
'f'
>>> x[0]
'i'
>>>
```

```
>>> a=list(range(1,10))
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

Usando o **list()** com o **range()** conseguimos extrair uma sequência de valores com início e fim.

A função **input()** é utilizada para inserir valores em um objeto a partir de uma interação com o usuário.

```
>>> nome=input("Digite o nome do aluno: ")
Digite o nome do aluno:
```

Após a pergunta o usuário digita o valor, que é inserido no objeto. Veja abaixo como ficou a tela completa com a resposta do usuário:

```
>>> nome=input("Digite o nome do aluno: ")
Digite o nome do aluno: Ana
>>> nome
'Ana'
>>>
```

Observe o exemplo abaixo:

```
>>> nota1=input("Digite a nota 1 do aluno: ")
Digite a nota 1 do aluno: 5
>>> nota2=input("Digite a nota 2 do aluno: ")
Digite a nota 2 do aluno: 7
>>> total=nota1+nota2
>>> total
'57'
>>>
```

O valor **57** na verdade é a concatenação do valor **5** com o valor **7**. Isso ocorre pois o Python, na utilização do **input**, insere o valor no objeto como **String** (texto). Anteriormente vimos que não é obrigatória a declaração de variáveis mas nesse caso é importante, veja abaixo, irei declarar a variável como float.

```
>>> nota1=float(input("Digite a nota 1 do aluno: "))
Digite a nota 1 do aluno: 5
>>> nota2=float(input("Digite a nota 2 do aluno: "))
Digite a nota 2 do aluno: 7
>>> total=nota1+nota2
>>> total
12.0
>>>
```

O IDLE, possui um ambiente de script (Menu File → New File). O script é uma sequência de comandos executados todos de uma só vez. Adiante explicarei, dentro do software, como funciona. Veja:

Ambiente do Script

```
aluno=input("Digite o nome do aluno: ")
nota1=float(input("Digite a nota 1 do aluno: "))
nota2=float(input("Digite a nota 2 do aluno: "))
print(aluno, nota1, nota2)
```

Ambiente de execução:

```
Digite o nome do aluno: Ana
Digite a nota 1 do aluno: 8
Digite a nota 2 do aluno: 9
Ana 8.0 9.0
>>>
```

A função **def()** serve para criar, definir, gerar uma função como o usuário quiser.

Ex1: função **media** para calcular a média de um aluno.

```
def media(aluno,n1,n2,n3,n4):
    m=float((n1+n2+n3+n4)/4)
    print("A média do aluno ",aluno," é ",m)
```

Após a execução do script, execute a função, passando os parâmetros necessários.

```
>>> media("Felipe",8,9,7,10)
A média do aluno  Felipe  é  8.5
>>>
```

Aprimorando um pouco mais o código. Após a função criada ela já é executada.

```
def media(aluno,n1,n2,n3,n4):
    m=float((n1+n2+n3+n4)/4)
    print("A média do aluno ",aluno," é ",m)

media(aluno=str(input("Digite o nome do aluno: ")),
      n1=float(input("Digite a nota 1: ")),
      n2=float(input("Digite a nota 2: ")),
      n3=float(input("Digite a nota 3: ")),
      n4=float(input("Digite a nota 4: ")))
```

Resultado

```
Digite o nome do aluno: Bea
Digite a nota 1: 8
Digite a nota 2: 7
Digite a nota 3: 9
Digite a nota 4: 8
A média do aluno  Bea  é  8.0
>>>
```

A função **if()** é muito similar à função **se** do Excel.

Exemplo 1:

```
nota=int(input("Digite a nota :"))
if nota>7:
    print("Aprovado")
else:
    print("Reprovado")
```

Resultado

```
Digite a nota :9
Aprovado
>>>
```

Nesse caso, se a nota for maior que 7, mostrará na tela o valor **Aprovado** senão, mostrará **Reprovado**.

O IF termina sempre com um “:”.

Podemos **criar uma função** usando **def()** para a situação de alunos, com o **if()**

Código:

```
def media(aluno,n1,n2,n3,n4):
    m=float((n1+n2+n3+n4)/4)
    if m>7:
        print("O aluno ", aluno," está Aprovado")
    else:
        print("O aluno ", aluno," está Reprovado")

media(aluno=str(input("Digite o nome do aluno: ")),
      n1=float(input("Digite a nota 1: ")),
      n2=float(input("Digite a nota 2: ")),
      n3=float(input("Digite a nota 3: ")),
      n4=float(input("Digite a nota 4: ")))
```

Cenário 1:

```
Digite o nome do aluno: Pedro
Digite a nota 1: 8
Digite a nota 2: 9
Digite a nota 3: 7
Digite a nota 4: 8
O aluno  Pedro  está Aprovado
>>>
```

Cenário 2:

```
Digite o nome do aluno: Beto
Digite a nota 1: 4
Digite a nota 2: 6
Digite a nota 3: 5
Digite a nota 4: 3
O aluno  Beto  está Reprovado
>>>
```

A função **elif()** adiciona recursos à função **if()**

```
nota=int(input("Digite a nota :"))
if nota>7:
    print("Aprovado")
elif nota<5:
    print("Reprovado")
else:
    print("Recuperação")
```

Cenários:

```
Digite a nota :8
Aprovado
>>>
Digite a nota :6
Recuperação
>>>
Digite a nota :4
Reprovado
>>>
```

A função **for()** gera um loop de repetição até que toda a condição seja satisfeita. A função **while()** tem uma estrutura similar.

Ex1:

```
alunos=["Ana","Bea","Pedro"]
for a in alunos:
    print(a)
```

Resultado:

```
Ana
Bea
Pedro
>>>
```

É possível adicionar um bloco de código ao final do loop, utilizando **for else**.

```
alunos=["Ana","Bea","Pedro"]
for a in alunos:
    print(a)
else:
    print("Lista completada com sucesso")
```

Resultado:

```
Ana
Bea
Pedro
Lista completada com sucesso
>>>
```

É possível ainda fazer um **for()** com uma interação da mensagem final:

```
alunos=["Ana","Bea","Pedro"]
for a in alunos:
    print(a)
else:
    turma=input("Digite o nome da turma: ")
print("Lista da turma ",turma, " completada com sucesso")
```

Interação

```
Ana
Bea
Pedro
Digite o nome da turma:
```

Resultado

```
Ana
Bea
Pedro
Digite o nome da turma: VIP
Lista da turma  VIP  completada com sucesso
>>>
```

A função **while()** executa algumas instruções "enquanto" uma condição é verdadeira/atendida.

```
x=0
while x<10:
    print("X é menor que 10, X é ", x)
    x=x+1
```

Resultado

```
X é menor que 10, X é  0
X é menor que 10, X é  1
X é menor que 10, X é  2
X é menor que 10, X é  3
X é menor que 10, X é  4
X é menor que 10, X é  5
X é menor que 10, X é  6
X é menor que 10, X é  7
X é menor que 10, X é  8
X é menor que 10, X é  9
```

Nesse caso, enquanto "**X**" for **menor do que 10**, aparecerá na tela a frase que está na função **print**. Note o objeto "X" no início do script: iniciei "**zerando**" esse objeto. Essa parte do código **x=x+1** vai incrementando o valor de X até que chegue em 10 e finalize o código.

Aprofundando o **while** juntamente com vetores, **append** e **for**.

Exercício para criar uma turma de alunos.

```
x=1
aluno=[" "]
qtde=int(input("Digite o número de alunos: "))
aluno1=input("Digite o nome do aluno: ")
aluno[0]=aluno1
while x<qtde:
    aluno.append(input("Digite o nome do aluno: "))
    x=x+1
turma=input("Digite o nome da turma: ")
print("Lista de alunos da turma ", turma)
for a in aluno:
    print(a)
```

Cenário

```
Digite o número de alunos: 5
Digite o nome do aluno: Ana
Digite o nome do aluno: André
Digite o nome do aluno: Felipe
Digite o nome do aluno: Bea
Digite o nome do aluno: Mauro
Digite o nome da turma: VIP
Lista de alunos da turma  VIP
Ana
André
Felipe
Bea
Mauro
>>>
```

A função **len()** exibe o comprimento de um vetor

```
>>> vetor1=[3,7,6,9]
>>> vetor2=["Ana","Fred","André","Bea"]
>>> total=[vetor1,vetor2]
>>> len(vetor1)
4
>>> len(vetor2)
4
>>> len(total)
2
>>>
```

Funções **min()** e **max()** trazem o menor e o maior valor, respectivamente, de um vetor.

```
>>> a=[7,5,12,15,10]
>>> min(a)
5
>>> max(a)
15
>>>
```

Função **sum()** efetua a soma dos valores de um vetor.

```
>>> a=[7,5,12,15,10]
>>> sum(a)
49
>>>
```

2 PROGRAMAÇÃO EM R

Linguagem orientada a objetos e muito usado em manipulação da matéria de Estatística. Utilizada no desenvolvimento para cálculos estatísticos e gráficos.

Objetos ou entidades lógicas em muitas linguagens também são chamados de variável. Todo objeto precisa ter um nome, deve ser identificado e armazena valores. A célula do Excel pode ser comparada a um objeto.

Em regra, um objeto contém apenas um valor e deve começar com uma letra. Quase todos os comandos do R servem para "mostrar algo na tela". Os comandos devem ser digitados respeitando a diferenciação entre maiúsculo e minúsculo: R é case sensitive.

2.1 Tipos de dados

A declaração da variável não é obrigatória em R. Basta criar o objeto e o interpretador reconhece o tipo de dado.

`as.numeric`	Quando o valor é número
`as.character`	Quando o valor é texto
`as.logical`	Quando é valor lógico (TRUE ou FALSE)

2.2 Sintaxe e comandos

Para testar os comandos é interessante baixar e instalar o Console R. Se não for possível, siga os passos abaixo para praticar R em um navegador web, sem precisar instalar. Alguns comandos podem não funcionar, mas os principais eu testei e funcionou.

1. Acesse https://posit.cloud/ e clique em “Sign Up” (caso seja o primeiro acesso);
    a. Você pode clicar também em “Log in” e efetuar o login diretamente pela conta Google;
2. Em “Cloud Free” clique em “Learn more”;
3. Na próxima página clique em “Sign UP”;
4. Faça seu cadastro. É rápido e gratuito;
5. Ao acessar a plataforma, à direita clique em “New Project” e após “New Rstudio Project”;
6. Aguarde a plataforma criar o projeto novo;

7. Pronto, após aparecer o caractere “>” você já pode começar os testes.

`>`	Prompt de comando para escrita
`[1]`	Resposta para um comando (primeiro valor do comando) se tiver mais de uma linha aparecerá outro valor Exemplo: [1] valores descritos de um comando…. [17] na linha dois aparece o décimo sétimo valor… [35] na terceira linha aparece o trigésimo quinto...
`"`	Utilizado em argumento de texto

`q()`
ou
`quit()`
	Fecha o console

`help()`
ou
`help(tópico)`
ou
`help.search(item)`
	Ajuda do programa ou de um comando no site do R

`help.start()`	Ajuda interna do programa.

`=`
ou
`<--`
ou
`->`
	Atribui valor a um objeto/variável (a seta sempre aponta para o objeto)

`ls()`	Lista todos os objetos que existem na console

`rm(objeto)` ou `remove(objeto)`	Apaga/remove o objeto
`rm(list=ls())`	Remove todos os objetos
`x:y`	Apresenta uma sequência de valores de "x" até "y" Exemplo 1 >1:5 >[1] 1 2 3 4 5 > Exemplo 2 >3:10 >[1] 3 4 5 6 7 8 9 10 >

2.3 Ordem dos precedentes

Abaixo a ordem dos precedentes dos operadores matemáticos.

( )	Parênteses
^ ou **	Potência
* /	Multiplicação Divisão
+ -	Adição Subtração
%%	Resto da divisão

Se tiver multiplicação e divisão na mesma operação, resolve primeiro quem está mais à esquerda.

Exemplo 1:
```
>30 + 10
[1] 40
>
```

Exemplo 2:
```
> 2 ** 3
[1] 8
>
```

Exemplo 3:
```
>30 + 10 * 2**3
[1]  110
>
```

Exemplo 4:
```
>7%%3
[1]  1
>
```

No Exemplo 3 resolve primeiro **2**3** (potência) que resulta em **8**, depois resolve **10*8** (multiplicação) que resulta em **80** e depois soma com 30.

Os espaços entre os valores e os operadores podem existir ou não.

Exemplo 1:
```
>8/4
[1] 2
>
```

Exemplo 2:
```
>8  /  4
[1] 2
>
```

Operadores lógicos

>	Maior
<	Menor
==	Igual
>=	Maior ou igual
<=	Menor ou igual
!=	Diferente

Exemplo 1
```
> 20 > 30
[1] FALSE
>
```

Exemplo 2
```
> 20<30
[1] TRUE
>
```

Exemplo 3

```
> 20 == 20
[1] TRUE
>
```

Os valores que trabalhamos acima podem ser atribuídos aos objetos de diversas formas. Os objetos são criados quando eles recebem sua primeira atribuição. Na atribuição não existe resposta “[1]” na tela para o usuário.

Exemplo 1

```
> nota1 = 8
>nota2 = 9
> aluno <- “José”
> “Agente” -> prova
```

Podemos usar os valores desses objetos para mostrar algo na tela para o usuário. Na exibição dos objetos existe uma resposta “[1]”.

Exemplo 1

```
> aluno
[1] José
>
```

Exemplo 2

```
> prova
[1] Agente
>
```

Exemplo 3

```
>nota1
nota2
[1] 17
>
```

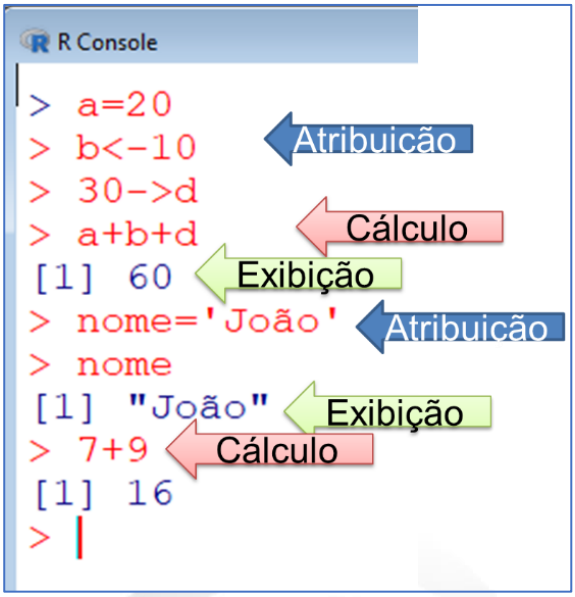
Exemplo 4

```
+ >nota2 + 10
  [1] 19
  >
```

Exemplo 5

```
> c('O aluno ',aluno,' fez a prova para ',prova,' e ficou com média ', (nota1+nota2)/2)
[1] "O aluno "  "José" " fez a prova para " "Agente" " e ficou com média " "8.5"
```

Exemplo e explicação de atribuição, cálculo e exibição dentro do Console R.



2.4 Erros

O console, geralmente, aponta onde estão os erros. A forma simplificada é apontar se o erro está ao "lado esquerdo" ou "lado direito", isso ocorre quando é utilizado o "=" para atribuir valor a objetos, por exemplo.

Exemplo 1:

```
>nome = 'Pedro'
>5 = nota
Error in 5 = nota : lado esquerdo da atribuição inválida
```

O erro ocorreu pois objetos devem ter seu nome iniciado com letras.

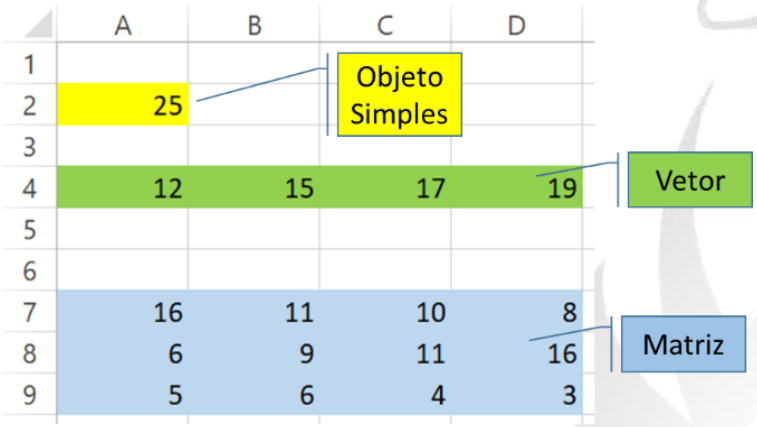
2.5 Objetos simples

Possuem apenas um valor.

```
>x=20
>
```

2.6 Vetores, Matrizes e Listas

Vou utilizar uma tela do Excel para iniciar a explicação de vetores e matrizes.



A2 é um objeto simples que recebe o valor 25.
O intervalo de A4:D4 é um vetor (uma sequência de valores em linhas)
O intervalo de A7:D9 é uma matriz (tabela com linhas e colunas)

Vetor: vários valores em apenas uma dimensão (uma linha). Todos os valores devem ser do mesmo tipo.

```
>x=c(10, 20, 30)
>
```

Esse é um vetor de 3 posições pois o objeto "x" está recebendo 3 valores

Para mostrar um valor do vetor a sintaxe é: **objeto[posição]**

Exemplo 1
```
>x[2]
>[1] 20
>
```

Exemplo 2
```
>x[3]
>[1] 30
>
```

Para adicionar ou alterar um valor do vetor a sintaxe é: **objeto[posição]=novovalor**

Exemplo 1
```
>x[4]=35
>
```
Adicionei, ao vetor x o valor 35

Exemplo 2
```
>x[4]=40
>
```
Alterei a posição 4 do vetor x para o valor 40.

Exemplo 3
```
> a=2:12
> a
 [1]  2  3  4  5  6  7  8  9 10 11 12
> a+1
 [1]  3  4  5  6  7  8  9 10 11 12 13
>
```

Criei o vetor "a" com intervalo de 2 a 12. Logo após fiz a operação de **a+1** ou seja, somei 1 aos valores já existentes do vetor **a**.

Matriz: vários valores em várias dimensões (várias linhas e várias colunas). Vários vetores "empilhados". Para criar uma Matriz, primeiramente é necessário criar um vetor.

```
> v1=c(1:10)
> v1
 [1]  1  2  3  4  5  6  7  8  9 10
>
```

Acima, criei um vetor com nome "v1" e adicionei uma concatenação de intervalo de valores de 1 até 10.

Após criado o vetor, criarei a matriz, representada abaixo por "m1"

```
> m1=matrix(data=v1,nrow=5,ncol=2)
> m1
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
>
```

A função "**matrix**" serve para criação de matriz e sua sintaxe engloba:
data=vetorcriado,nrow=númerolinhas,ncol=númerodecolunas
Acima eu criei uma matriz com o vetor "v1" composta de 5 linhas e 2 colunas

Posso criar também dessa forma:

```
> m2=matrix(v1,2,5)
> m2
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
>
```

Criei a matriz "m2" com o vetor "v1" com 2 linhas e 5 colunas.

Visualizando os valores em suas posições
Sintaxe: **matriz[linha,coluna]**

```
> m2
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

Ex1: visualizado o valor da linha 1, coluna 3 da matriz "m2"

```
> m2[1,3]
[1] 5
```

Ex2: visualizando o valor da linha 2, coluna 5 de m2

```
> m2[2,5]
[1] 10
```

Ex3: visualizando os valores da linha 1

```
> m2[1,]
[1] 1 3 5 7 9
```

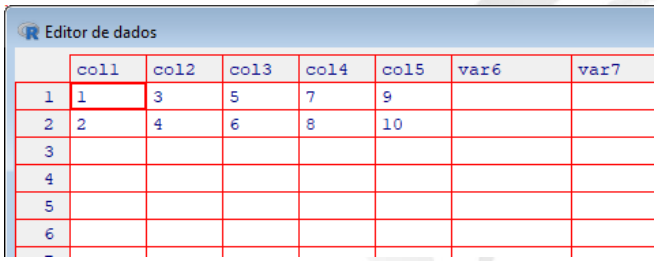
Ex4: visualizando os valores da coluna 4

```
> m2[,4]
[1] 7 8
>
```

É possível abrir a matriz em modo de janela para visualizar ou manipular os dados desta, com a função **fix(matriz)**

Ex:

```
>fix(m2)
```



É possível criar matriz de forma direta, informando apenas as colunas e as linhas são criadas automaticamente.

```
> matrix(1:21,ncol=3)
     [,1] [,2] [,3]
[1,]    1    8   15
[2,]    2    9   16
[3,]    3   10   17
[4,]    4   11   18
[5,]    5   12   19
[6,]    6   13   20
[7,]    7   14   21
>
```

Criei uma matriz com intervalor de dados iniciando em 1 e finalizando em 12. Com 3 colunas.

É possível criar uma matriz direta informando somente o número de linhas, também. As colunas são criadas automaticamente.

```
> matrix(1:21,nrow=3)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    4    7   10   13   16   19
[2,]    2    5    8   11   14   17   20
[3,]    3    6    9   12   15   18   21
>
```

Perceba que a sequência é criada em colunas. É possível pedir para que a sequência seja criada em linhas.

```
> matrix(1:21,nrow=3, byrow=TRUE)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    2    3    4    5    6    7
[2,]    8    9   10   11   12   13   14
[3,]   15   16   17   18   19   20   21
>
```

O argumento “byrow” estando como “TRUE” informa que a sequência é por linhas.

Se o “byrow” estiver “FALSE” é o estado natural (como se o byrow não fosse usado). Veja:

```
> matrix(1:21,nrow=3, byrow=FALSE)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    4    7   10   13   16   19
[2,]    2    5    8   11   14   17   20
[3,]    3    6    9   12   15   18   21
>
```

Lista é um objeto que armazena valores diferentes de tipos de dados diferentes. Permite tipificar seus componentes.

A função **list()** é utilizada para criar uma lista.

Ex1: criando uma lista

```
> alunos=list(nome='Ana', nota=9)
```

Ex2: exibindo a lista

```
> alunos
```

```
$`nome`
[1] "Ana"

$nota
[1] 9
```

Ex3: exibindo um campo da lista usando índice [n]

```
> alunos[2]
$`nota`
[1] 9
```

Usando índice, sem o rótulo $`nota` usa 2 colchetes.

```
> alunos[[2]]
[1] 9
>
```

Ex3: acessando componente na lista pelo rótulo

```
> alunos$nota
[1] 9
> alunos$nome
[1] "Ana"
>
```

2.7 Funções

Muito similar ao que vemos no Excel porém em programação as funções podem fazer parte de scripts, que podem ser considerados pequenos blocos que executam algo e retornam um resultado.

Até a sintaxe básica das funções é muito similar ao Excel e ao Calc:
nomedafunção(argumentos)

Um objeto nunca poderá ter o mesmo nome que uma função. Por exemplo, não posso dar nome a objetos como **sqrt** ou **factorial** e assim por diante.

Objetos podem receber funções. Ex:

```
> x=seq(1,7,2)
> x
[1] 1 3 5 7
>
```

Acima, o objeto "x" recebeu uma função "seq". Nesse caso, a função "seq" criou uma sequência, um vetor. Veja abaixo como demonstro o valor de uma posição do vetor.

`> x`
`[1] 1 3 5 7`
`> x[2]`
`[1] 3`
`> x[4]`
`[1] 7`
`>`
	Vetor "x" na posição "2" tem o valor 3.
	Vetor "x" na posição "4" tem o valor 7.

Função **sqrt** (n) raiz quadrada

```
>sqrt (16)
[1] 4
>
```

Função **factorial** (n) fatorial

```
>factorial (4)
>[1] 24
>
```

Nesse caso é a multiplicação do 4 pelos seus anteriores, na matemática a notação seria 4!

Função **c()** utilizada para fazer inserções de vetores em objetos. O "c" é de "concatenar" valores ou criar "conjuntos" de valores.

```
>c(10,20,30,40)
>[1] 10 20 30 40
>
```

Função **rep(x,y)** repete "y" vezes o valor "x".
Ex1: repetir o valor 10, 5 vezes.

```
> rep(10,5)
```

```
[1] 10 10 10 10 10
>
```

Ex2: repetir a concatenação “c” de 2, 4 e 6, por 3 vezes.

```
> rep(c(2,4,6),3)
[1] 2 4 6 2 4 6 2 4 6
>
```

Função **seq(x,y,z)** cria uma sequência iniciando em X, finalizando em Y com intervalos em Z.

```
> seq(10,20,3)
[1] 10 13 16 19
>
>seq(10,20,5)
[1] 10 15 20
>
>seq(10,30,5)
[1] 10 15 20 25 30
>

> seq(10,1,-2)
[1] 10  8  6  4  2
>
```

Função **gl(i,r)** cria um vetor com “i” níveis, de 1 a “i” com “r” repetições

```
> gl(3,2)
[1] 1 1 2 2 3 3
Levels: 1 2 3
>

> gl(5,3)
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
Levels: 1 2 3 4 5
>

> gl(3,5)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3
>
```

```
Função abs() valor absoluto
>abs(-3)
>[1] 3
>
```

Função **length(valor)** informa o tamanho do vetor
Primeiro criamos um vetor

```
> ex=gl(5,3)
> ex
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
Levels: 1 2 3 4 5
>

> length(ex)
[1] 15
>
```

Função **sort(valor)** ordena vetor
Primeiro criamos um vetor

```
> ex=gl(5,3)
> ex
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
Levels: 1 2 3 4 5
>

> sort(ex)
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
Levels: 1 2 3 4 5
>
```

Função **sample(vetor,amostra)** retira uma amostra de um determinado vetor. Originariamente não repete valores.
1º vou criar os vetores **x y z**

```
> y<-1:10
> x=10:20
> seq(1,7,2)->z
> y
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
> x
 [1] 10 11 12 13 14 15 16 17 18 19 20
> z
[1] 1 3 5 7
```

Ex1: amostra de 3 valores aleatórios do vetor y

```
> sample(y,3)
[1] 4 9 7
```

Ex2: amostra de 3 valores aleatórios do vetor y

```
> sample(y,3)
[1] 7 8 9
```

Ex3: amostra de 3 valores aleatórios do vetor y

```
> sample(y,3)
[1] 6 1 7
```

Ex4: amostra de 5 valores aleatórios do vetor y

```
> sample(y,5)
[1] 1 7 5 9 2
```

Ex5: amostra de 2 valores aleatórios do vetor x

```
> sample(x,2)
[1] 14 15
```

Ex6: amostra de 2 valores aleatórios do vetor x

```
> sample(x,2)
[1] 17 12
```

Ex7: amostra de 4 valores aleatórios do vetor x

```
> sample(x,4)
[1] 15 16 14 13
>
```

Função **append(vetor,conteúdo)** adiciona conteúdo a um vetor já existente.

```
> a
 [1]  2  3  4  5  6  7  8  9 10 11 12
> append(a,20)
 [1]  2  3  4  5  6  7  8  9 10 11 12 20
```

```
> a
 [1]  2  3  4  5  6  7  8  9 10 11 12
>
```

Acima, já existia o vetor **a** e adicionei apenas na visualização o valor 20. Perceba que apenas mostrou o valor 20, após a execução do append. Se for necessário mudar o vetor **a** para que ele contenha o 20, do exemplo acima, esse valor deve ser atribuído ao vetor **a**.

```
> a
 [1]  2  3  4  5  6  7  8  9 10 11 12
> a=append(a,20)
> a
 [1]  2  3  4  5  6  7  8  9 10 11 12 20
>
```

Caso queira adicionar mais valores (concatenados) deve ser utilizada a função **c**.

```
> a=append(a,c(30,40,50,60))
> a
 [1]  2  3  4  5  6  7  8  9 10 11 12 20 30 40 50 60 30 40
50 60
>
```

Função **names()** transforma os índices dos valores de um vetor para nomes.

```
> z=c(7,9,8)
> z
[1] 7 9 8
> names(z)=c("mario","ana","maria")
> z
mario   ana maria
    7     9     8
>
```

O usuário pode criar sua própria função. Veja:

Uma função que retorne a soma de um número par se a entrada for par e número ímpar se a entrada for ímpar.

```
somarex <- function(x, y = 2) {
  x + y
}
```

Nesta função:

- **x** é um argumento obrigatório.
- **y** é um argumento opcional que tem um valor padrão de 2.

Se você chamar a função **somarex** apenas com um valor para **x**, o valor de **y** será automaticamente 2. Se você fornecer um valor para **y**, ele substituirá o valor padrão.

Exemplo 1: Usando o valor padrão de y (2)

```
resultado1 <- somarex(5)
print(resultado1)  # Resultado: 5 + 2 = 7
```

Exemplo 2: Usando o valor padrão de y (2)

```
resultado1 <- somarex(4)
print(resultado1)  # Resultado: 4 + 2 = 6
```

Ao usar somarex(5) o número 5 será o valor de x.

Internamente na função, ficaria assim

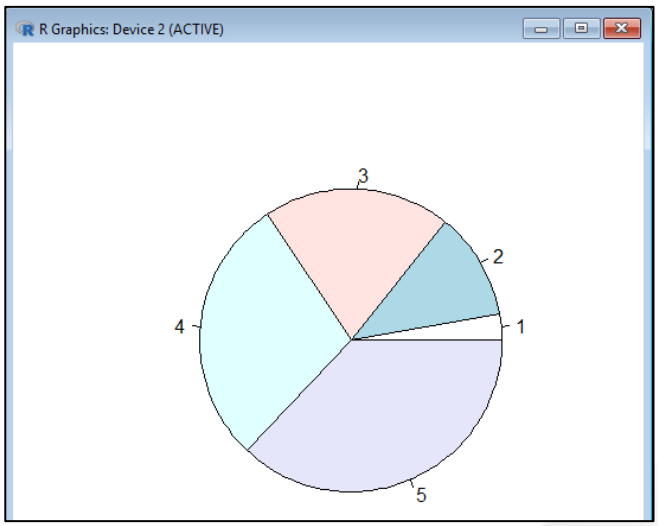
```
somarex <- function(x = 5, y = 2) {
  x + y
}
```

2.8 Gráficos

Por ser uma linguagem de programação voltada à estatística, a função de construção de gráficos torna-se importantíssima. Na construção de um gráfico, uma janela extra e específica deve ser utilizada.

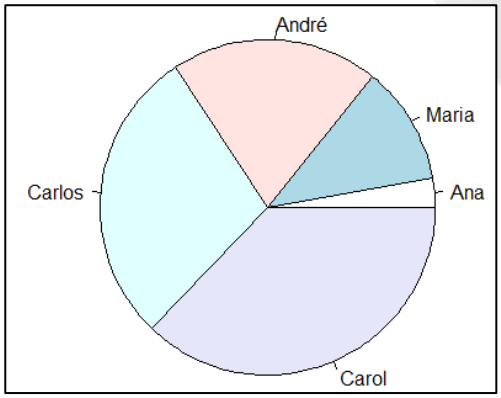
Função **pie()** serve para construir um gráfico de "torta" de um determinado vetor

```
> x=seq(1,15,3)
> x
[1]  1  4  7 10 13
> pie(x)
>
```



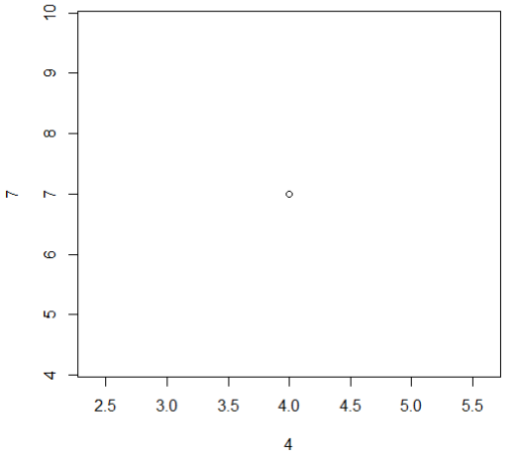
Representação do gráfico **pie** e as posições (1, 2, 3, 4, 5) do vetor. Se usar a função **names()** descrita um pouco acima, os rótulos do gráfico ganham nomes.

```
> x
[1]  1  4  7 10 13
> names(x)=c("Ana","Maria","André","Carlos","Carol")
> x
   Ana  Maria  André Carlos  Carol
     1      4      7     10     13
> pie(x)
>
```



Função **plot()** cria gráficos usando as coordenadas X e Y.

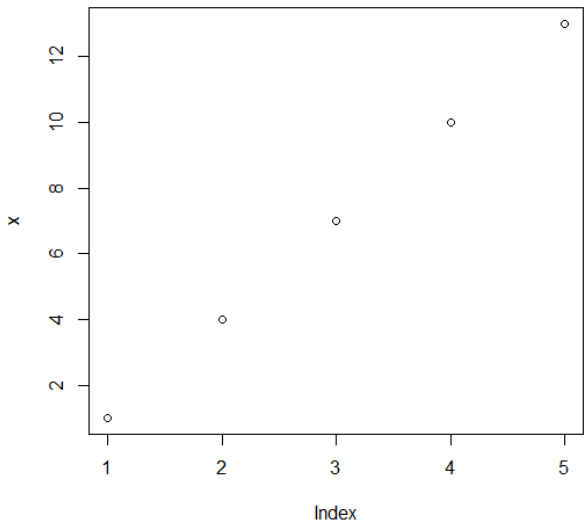
```
> plot(4,7)
>
```



Criou um ponto de intersecção entre 4 e 7.

Usando vetores

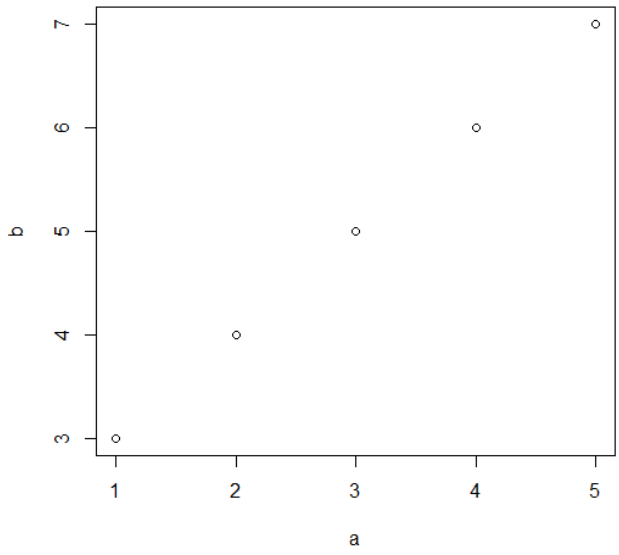
```
> x
   Ana  Maria  André Carlos  Carol
     1      4      7     10     13
> plot(x)
>
```



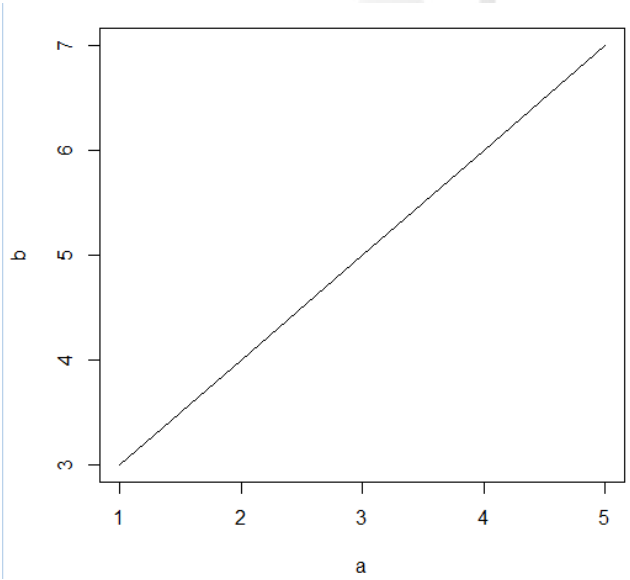

```
> a=(1:5)
> b=(3:7)
> a
[1] 1 2 3 4 5
> b
[1] 3 4 5 6 7
```

```
> plot(a,b)
>
```

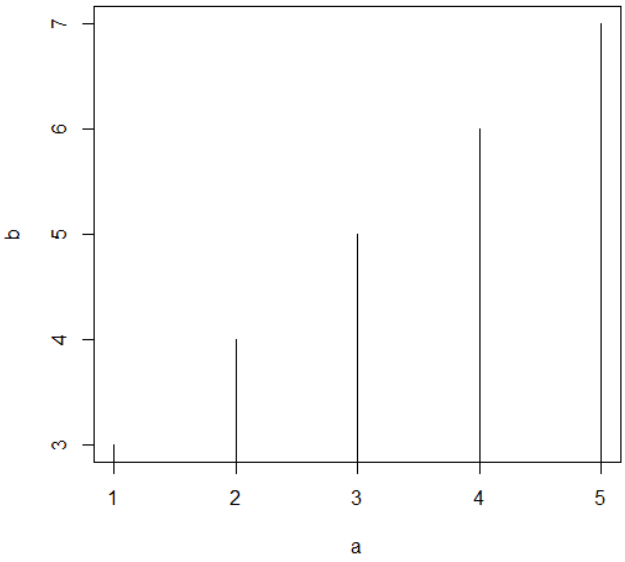
Os valores do vetor **a** fazem parte do **X** e os valores do vetor **b** fazem parte do **Y.**



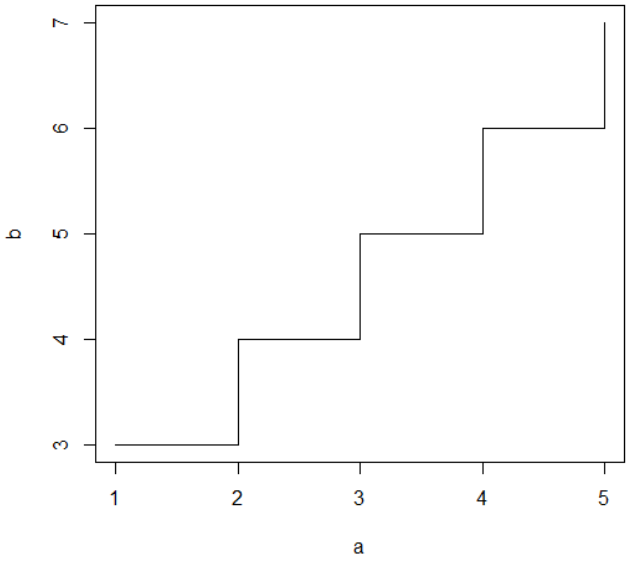

```
> plot(a,b,type="l")
>
```



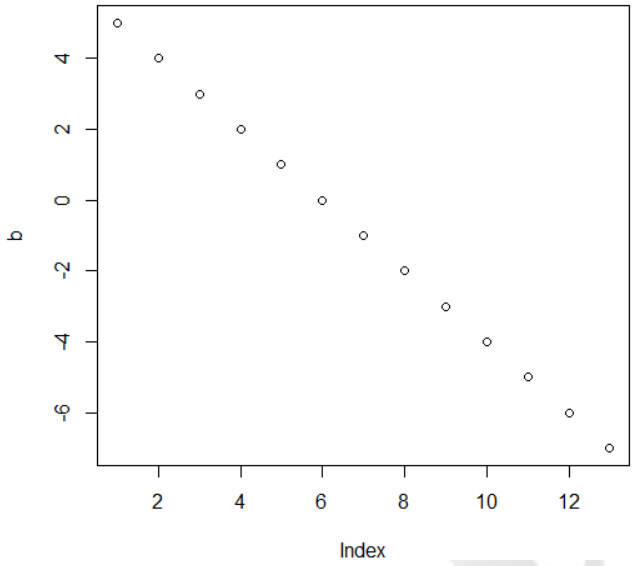
```
> plot(a,b,type="h")
>
```



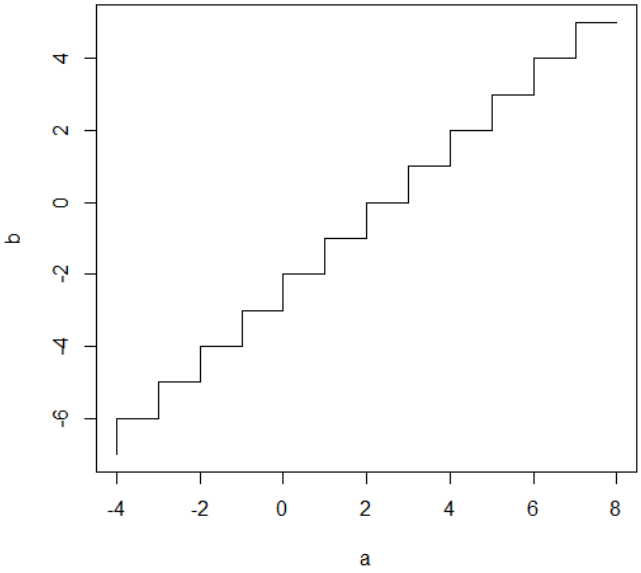

```
> plot(a,b,type="s")
>
```



```
> b
 [1]  5  4  3  2  1  0 -1 -2 -3 -4 -5 -6 -7
> plot(b)
>
```

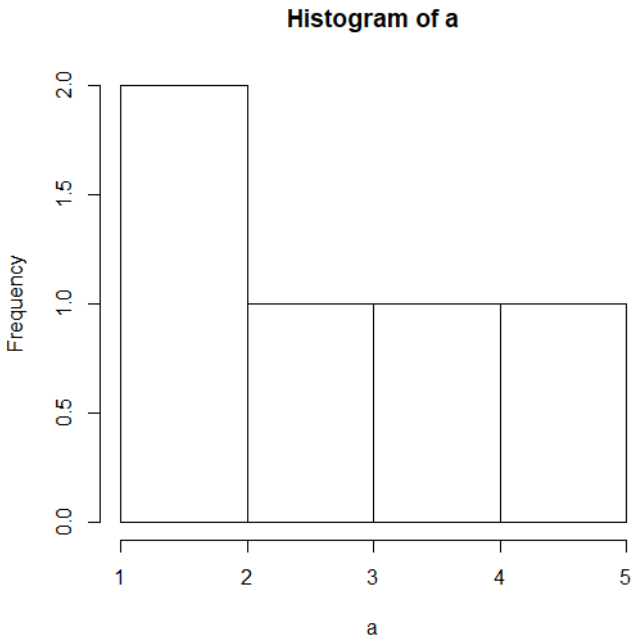



```
> a
 [1]  8  7  6  5  4  3  2  1  0 -1 -2 -3 -4
> b
 [1]  5  4  3  2  1  0 -1 -2 -3 -4 -5 -6 -7
> plot(a,b,type="s")
>
```

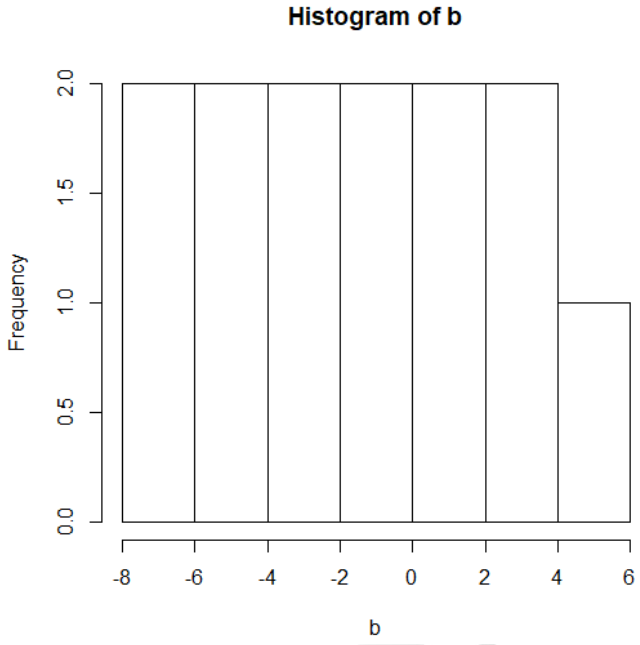


Função hist() cria um gráfico de histograma.

```
> a
[1] 1 2 3 4 5
> hist(a)
>
```



```
> b
 [1]  5  4  3  2  1  0 -1 -2 -3 -4 -5 -6 -7
> hist(b)
>
```



3 INSTRUÇÕES EM PYTHON E R

Para fazer o download em Windows:

Python: acesse https://www.python.org/downloads/ faça o download do Python última versão. Depois de baixado e instalado vá até o Menu Iniciar e pesquise por “IDLE”. IDLE, é o console principal do Python.

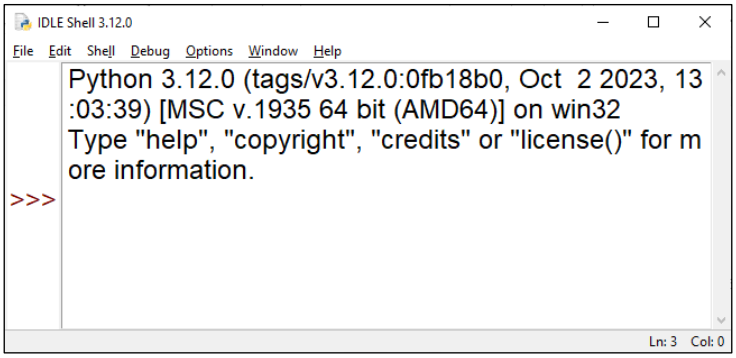


Figura 3 - Console IDLE do Python

R: acesse https://cran.r-project.org/bin/windows/base/R-4.3.2-win.exe que o R já será baixado e instalado na máquina.

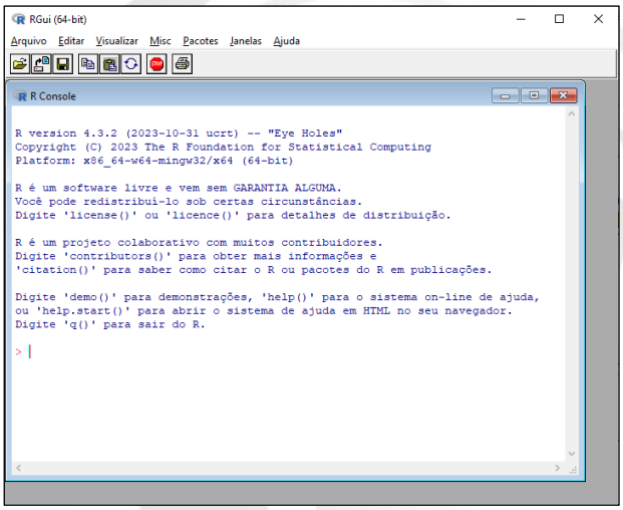
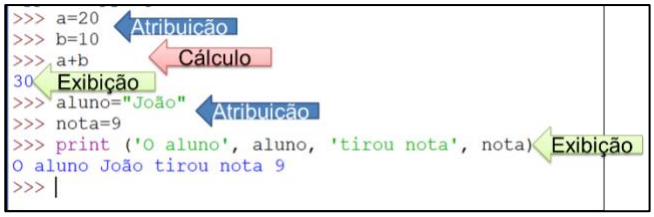


Figura 4 - Console R

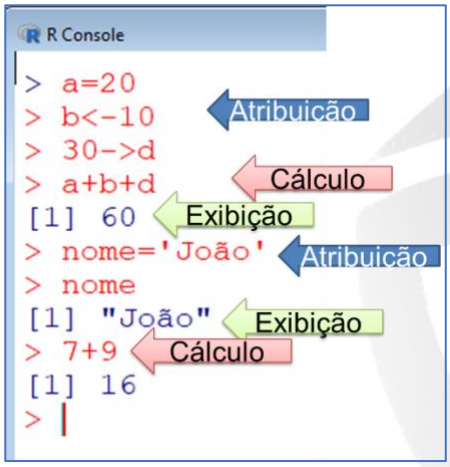
Todo comando que for dado de entrada nos consoles, para ser executado, deve-se apertar ENTER.

3.1 Entendendo as telas dos consoles

Python



R



Vamos começar:

Abra o console IDLE e digite

3+4 (aperte ENTER)

3+4*2 (aperte ENTER)

O resultado terá que ser:

Veja que foi respeitada a ordem dos precedentes no segundo comando, primeiro multiplicou e depois fez adição.

Agora abra o console R e faça o mesmo. Digite:

3+4 (aperte ENTER)

3+4*2 (aperte ENTER)

O resultado:

```
R é um projeto colaborativo com muitos
Digite 'contributors()' para obter mai
'citation()' para saber como citar o R

Digite 'demo()' para demonstrações, 'h
ou 'help.start()' para abrir o sistema
Digite 'q()' para sair do R.

[Área de trabalho anterior carregada]

> 3+4
[1] 7
> 3+4*2
[1] 11
>
```

Perceba que logo após o [1] vem a resposta.

A partir de agora farei os comandos e exercícios em Python e após finalizar todo o estudo de Python, partiremos para o R.

FICA ALERTA, GUERREIRO(A)!

Não copie o código desse material e cole os comandos nos consoles. Para praticar é fundamental que você digite os comandos em cada console e compreenda o funcionamento.

3.2 Python na prática

Abra o console IDLE do Python. Execute os comandos conforme os quadros abaixo.

Ignore os caracteres “>>>” que são nativos do IDLE.

```
>>> a=5
>>> b=7
>>> c=2
>>> a
5
>>> b
7
>>> c
2
>>>
```

Nesse momento foram criados os objetos **a**, **b** e **c**. Dentro dos objetos existem os valores, **5**, **7** e **2** respectivamente.

```
>>> d=a+b
>>> d
12
>>> concurso="Polícia"+" "+"Federal"
>>> concurso
'Polícia Federal'
>>>
```

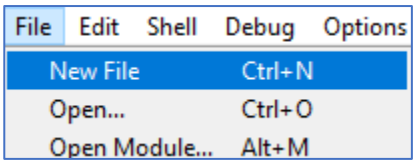
O texto deve estar entre “ aspas duplas “ ou ‘aspas simples’. O sinal de “+” faz a operação de concatenação de textos.

```
>>> cargo="agente"
>>> candidato="Felipe"
>>> insc=123
>>> print('O candidato ',candidato," fará a prova para ",concurso," para o cargo de ",cargo)
```

Resultado: `O candidato Felipe fará a prova para Polícia Federal para o cargo de agente`

Veja que no exemplo acima eu usei tanto “aspas duplas” quanto ‘aspas simples’.

O IDLE possui uma interface de scripts, onde o usuário escreve todo o código e logo após manda executar. Clique em "File"→"New File"



Sempre que fizer uma interação nesse ambiente, você deve salvar o arquivo, antes de executar. A execução é feita com **F5** (tecla importante para sua prova).

Vamos aos testes. No ambiente de scripts, digite o seguinte:

```
nome="Ana"
nota=9
print(nome," tirou nota ", nota)
```

Logo após aperte F5. O programa vai pedir para salvar o script. Escolha um local e salve. O nome do arquivo e local não fazem diferença na execução do script.

O resultado:

```
Ana  tirou nota  9
```

Retornando para a interface de scripts, vamos praticar. Lembrando que, a interface de scripts é o ambiente do arquivo que você salvou acima.

Vamos começar a utilizar o método **input()** para entrada de dados nesse ambiente de script. Apague tudo na tela e digite os seguintes comandos e logo após aperte **F5** para executar.

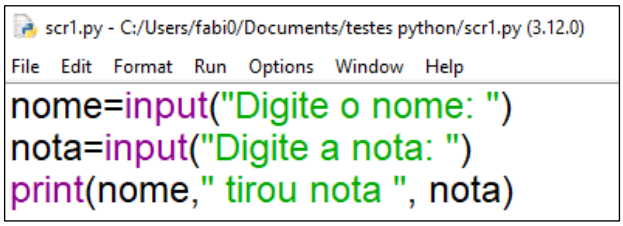
```
nome=input("Digite o nome: ")
nota=input("Digite a nota: ")
print(nome," tirou nota ", nota)
```

O input adiciona o valor digitado pelo usuário ao objeto.

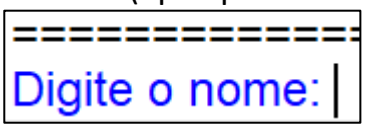
Possível resultado, com os dados que eu digitei (Maria, 9):

```
Digite o nome: Maria
Digite a nota: 9
Maria  tirou nota  9
```

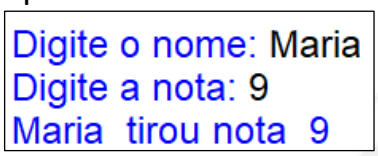
No ambiente de script:



No IDLE (após pressionar F5 no ambiente de scripts)



Após dar a entrada nos dados



É possível fazer cálculos também, com os valores que foram inseridos ao objeto com input, porém temos que tomar um cuidado.

```
nota1=input("Digite a nota1: ")
nota2=input("Digite a nota2: ")
media=(nota1+nota2)/2
print("A media foi ",media)
```

Resultado

```
Digite a nota1: 7
Digite a nota2: 8
Traceback (most recent call last):
  File
"C:\Users\fabi0\AppData\Local\Programs\Python\Python37-
32\tste.py", line 3, in <module>
    media=(nota1+nota2)/2
TypeError: unsupported operand type(s) for /: 'str' and 'int'
>>>
```

Esse erro ocorreu pois o **input()** adiciona valores de **string** (texto) ao objeto.

Veja o que acontece se, ao invés da média, eu somar os valores das notas.

```
nota1=input("Digite a nota1: ")
nota2=input("Digite a nota2: ")
total=nota1+nota2
print("A soma foi ",total)
```

Resultado

```
Digite a nota1: 7
Digite a nota2: 8
A soma foi  78
>>>
```

Concatenou “7” e “8” pois reconheceu como string (texto).

Para sanar o problema e fazer com que os valores sejam reconhecidos numericamente, precisamos **declarar essas variáveis**. Nesse caso pode ser com **int** (números inteiros) ou **float** (números reais). A declaração de variáveis demonstra ao sistema qual o tipo de dado que estou utilizando naquele momento. Muito similar ao que aprendemos no Excel.

```
nota1=int(input("Digite a nota1: "))
nota2=int(input("Digite a nota2: "))
total=nota1+nota2
print("A soma foi ",total)
```

Resultado

```
Digite a nota1: 7
Digite a nota2: 8
A soma foi  15
>>>
```

Agora consigo fazer a média

```
nota1=int(input("Digite a nota1: "))
nota2=int(input("Digite a nota2: "))
media=(nota1+nota2)/2
print("A média foi ",media)
```

Manipulação de Strings

```
>>> concurso="Polícia Federal"
>>> concurso
'Polícia Federal'
>>>
```

Acima, criei um objeto **concurso** e atribui o valor **Polícia Federal** a ele. Vamos fazer o "fatiamento" dessa string.

```
>>> concurso
'Polícia Federal'
>>> concurso[0]
'P'
>>> concurso[0:3]
'Pol'
>>> concurso[8:]
'Federal'
>>> concurso[:8]
'Polícia '
>>> concurso[:]
'Polícia Federal'
>>> concurso[0]+concurso[8]
'PF'
>>> concurso[0:8:2]
'Plca'
>>>
```

Para facilitar, sempre que quiser repetir o último comando pressione **ALT+P (P**revious).

O que está em **[]** mostrará o índice da posição onde quero manipular a string.

P	o	l	í	c	i	a		F	e	d	e	r	a	l
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Concurso[0]: traz o caractere da posição 0.

Concurso[0:3]: retorna os caracteres da posição 0 até a posição 3, **excluindo o caractere final**. Ou seja, é de 0 a 2.

Concurso[8:]: retorna os caracteres da posição 8 até o fim.

Concurso[:8]: retorna os caracteres do início até a posição 8.

Concurso[0:8:2]: retorna os caracteres de 0 a 8 (excluindo o 8), com saltos de 2 em 2.

Até agora trabalhamos com objetos que recebem apenas um valor. A partir de agora vamos exercitar o conceito de **vetores** e logo após de **matrizes**.

Pense num objeto como uma caixa, que cabe apenas um item. Para colocar outro, o anterior deve ser substituído.

Vou dar um exemplo usando o que curto: cerveja. Especificamente em lata.

 Neste isopor(objeto) cabe apenas uma lata por vez.

```
>>> isopor="heineken"
>>> isopor
'heineken'
>>> isopor="bud"
>>> isopor
'bud'
>>> isopor="amstel"
>>> isopor
'amstel'
>>>
```

Perceba que o objeto “isopor” não existia, foi criado quando inseri o primeiro valor. Perceba também que posso colocar outros “valores” no objeto, desde que o anterior seja substituído.

Pense num vetor como se fosse uma peça única de isopor com várias posições em linha reta.

```
>>> isopor=["heineken","bud","amstel","Miller"]
>>> isopor
['heineken', 'bud', 'amstel', 'Miller']
>>> isopor[0]
'heineken'
>>> isopor[2]
'amstel'
>>>
```

O vetor “isopor” acima, tem 4 posições, que vão da posição “0” até a posição “3”. No exemplo acima:

Posição	Valor
[0]	heineken
[1]	bud
[2]	amstel
[3]	Miller

FICA ALERTA, GUERREIRO(A)!

Perceba que o vetor começa em ZERO [0]. Nesse caso, a posição que para nós parece a posição 4 (Miller) para o Python é a posição 3.

Outro exemplo:

```
>>> alunos=["Ana","Bea","Xande","Felipe"]
>>> alunos
['Ana', 'Bea', 'Xande', 'Felipe']
>>> alunos[3]
'Felipe'
>>> alunos[4]
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    alunos[4]
IndexError: list index out of range
>>> alunos[2]
'Xande'
>>>
```

O comando **alunos[4]** trouxe um erro pois não existe a posição **4** do vetor **alunos**.

Imagine agora um "engradado" de isopor de latinhas de cerveja. No exemplo ao lado teríamos 9 posições vagas para latinhas, 3 linhas e 3 colunas em uma "matriz" ou tabela. A matriz nada mais é do que vários vetores empilhados, com posições (índices) endereçados por linhas e colunas, similar ao Excel.

```
>>>  engradado=[['Heineken',  'Bud',  'Amstel'],  ['Faxe',
'Dado', 'Paulaner'], ['Erdinger', 'Miller', 'DAB']]
>>> engradado[1][2]
'Paulaner'
>>> engradado[0][0]
'Heineken'
>>> engradado[2][2]
'DAB'
>>> engradado[0][1]
'Bud'
>>>
```

Visualmente, a matriz em Python ficaria assim, nesse caso:

Posição	0	1	2
0	Heineken	Bud	Amstel
1	Faxe	Dado	Paulaner
2	Erdinger	Miller	DAB

Com essa base, já é possível criar os códigos que estão no material específico de Python, em relação às funções e outros comandos.

3.3 R na prática

Abra o console R e digite os valores abaixo.

```
> a=5
> b<-7
> 2->c
>
```

Nesse momento foram criados os objetos **a**, **b** e **c**. Dentro dos objetos existem os valores, **5**, **7** e **2** respectivamente.
Perceba que os objetos (que podem ser chamados de variáveis também), recebem valores no Python usando o símbolo "=", somente. Na linguagem R, os valores podem ser adicionados aos objetos da seguinte forma.

Objeto=valor

Objeto<-valor

Valor -> objeto

Atente que a “seta” aponta sempre para o objeto, nunca para o valor.

```
> concurso="Polícia Federal"
> concurso
[1] "Polícia Federal"
> t1<-"Cargo "
> 'Agente'->t2
> c(t1,t2)
[1] "Cargo " "Agente"
>
```

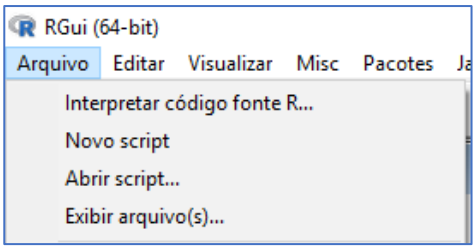
O texto deve estar entre “duplas” ou ‘simples’. A função **c** faz a operação de concatenação de valores.

```
> cargo="agente"
> candidato="Felipe"
> insc=123
> c("O candidato ",candidato," fará a prova para ",concurso,"
para o cargo de ",cargo)
[1] "O candidato "        "Felipe"              " fará a
prova para " "Polícia Federal"      " para o cargo de "
"agente"
>
```

Resultado: O candidato Felipe fará a prova para Polícia Federal para o cargo de agente.

Note que, por se tratar de um ambiente de programação mais voltado para a estatística, o retorno da mensagem não é tão amigável quanto o Python.

O R possui uma interface de scripts, onde o usuário escreve todo o código e logo após manda executar. Clique em "Arquivo"→"Novo Script"



Diferentemente do Python, a janela de script do R não abre uma segunda janela e sim uma segunda "aba" dentro do R, veja como ficou para mim.

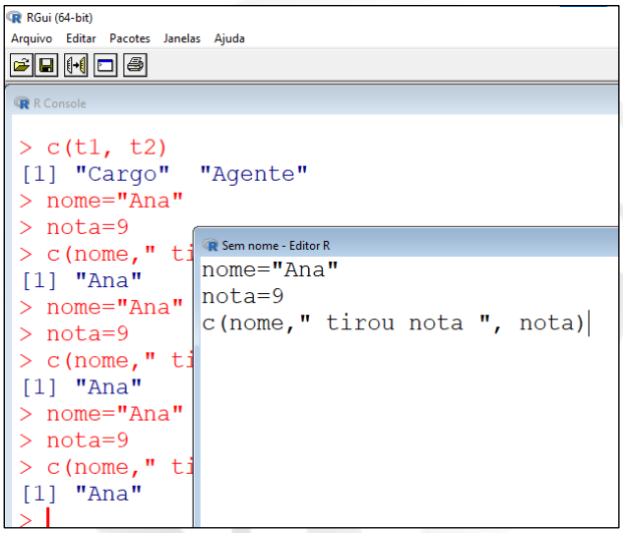


Figura 5 - R Console + Editor R

A execução deve ser feita pelo menu "Editar".

Vamos aos testes. No ambiente de scripts, digite o seguinte:

```
nome="Ana"
nota=9
c(nome," tirou nota ", nota)
```

O resultado:

```
> nome="Ana"
> nota=9
> c(nome," tirou nota ", nota)
[1] "Ana"           " tirou nota " "9"
```

Até agora trabalhamos com objetos que recebem apenas um valor. A partir de agora vamos exercitar o conceito de vetores e logo após de matrizes.

Pense num objeto como uma caixa, que cabe apenas um item. Para colocar outro, o anterior deve ser substituído.

Vou dar um exemplo usando o que curto: cerveja, novamente. Especificamente em lata.

 Neste isopor(objeto) cabe apenas uma lata por vez.

```
> isopor="heineken"
> isopor
[1] "heineken"
> isopor="bud"
> isopor
[1] "bud"
> isopor="amstel"
> isopor
[1] "amstel"
>
```

Perceba que o objeto “isopor” não existia, foi criado quando inseri o primeiro valor. Perceba também que posso colocar outros “valores” no objeto, desde que o anterior seja substituído.

 Pense num vetor como se fosse uma peça única de isopor com várias posições em linha reta.

```
> isopor=c("heineken","bud","amstel","Miller")
> isopor
[1] "heineken" "bud"      "amstel"   "Miller"
> isopor[1]
[1] "heineken"
> isopor[2]
[1] "bud"
>
```

O vetor “isopor” acima, tem 4 posições, que vão da posição “1” até a posição “4”. No exemplo acima:

Posição	Valor
[1]	heineken
[2]	bud
[3]	amstel
[4]	Miller

Diferente do Python, o primeiro valor de um vetor é “1”.

Outro exemplo:

```
> alunos=c("Ana", "Bea", "Xande", "Felipe")
> alunos[3]
[1] "Xande"
> alunos[4]
[1] "Felipe"
> alunos[5]
[1] NA
> alunos[2]
[1] "Bea"
>
```

O comando **alunos[5]** trouxe um erro pois não existe a posição **5** do vetor **alunos**.

Imagine agora um “engradado” de isopor de latinhas de cerveja. No exemplo ao lado teríamos 9 posições vagas para latinhas, 3 linhas e 3 colunas em uma “matriz” ou tabela. A matriz nada mais é do que vários vetores empilhados, com posições (índices) endereçados por linhas e colunas, similar ao Excel.

```
> cervejas=c('Heineken', 'Bud', 'Amstel', 'Faxe', 'Dado',
'Paulaner', 'Erdinger', 'Miller', 'DAB')
```

```
> engradado=matrix(cervejas,nrow=3,ncol=3)
> engradado
     [,1]       [,2]       [,3]
[1,] "Heineken" "Faxe"     "Erdinger"
[2,] "Bud"      "Dado"     "Miller"
[3,] "Amstel"   "Paulaner" "DAB"
> engradado[1][2]
[1] NA
> engradado[1,2]
[1] "Faxe"
> engradado[2,2]
[1] "Dado"
> engradado[1,1]
[1] "Heineken"
> engradado[1,]
[1] "Heineken" "Faxe"     "Erdinger"
> engradado[,3]
[1] "Erdinger" "Miller"   "DAB"
>
```

Em R, antes de criar uma **matrix** com a função **matrix(vetor,linhas,colunas)** é necessário criar um vetor. Criei o vetor **cervejas** e logo após criei a matriz **engradado** com 3 linhas e 3 colunas.

Visualmente, a matriz em R ficaria assim, nesse caso:

Posição	1	2	3
1	Heineken	Faxe	Erdinger
2	Bud	Dado	Miller
3	Amstel	Paulaner	DAB

4 QUESTÕES DE RENDIMENTO

01 (CEBRASPE|2021|PF|ESCRIVÃO)
Com relação a conceitos de programação Python e R, julgue o item que se segue.
O resultado do código R seguinte será "12".

```
f<- function(x) {
      g <- function(y) {
            y + z
      }
      z <- 4
      x + g(x)
}
z <- 10
f (4)
```

Resolução

Essa questão foi dada como certa no gabarito preliminar e depois trocaram pois o “12” está entre aspas e tudo que fica entre aspas, em R, é uma variável do tipo string (texto) e em toda a função foi utilizado apenas número inteiros. O resultado correto dessa função seria 12 (sem aspas).

Vamos resolver por partes:
Esse código R define uma função chamada **f** que envolve outra função chamada **g**.

```
g <- function(y) {
  y + z
}
```

Aqui, **g** é uma função que toma um argumento **y** e retorna a soma de **y** e **z**. Note que **z** não está definido dentro da função **g**, mas a função **g** pode acessar **z** do ambiente em que foi criada.

```
z <- 4
```

Definindo a variável z dentro da função f:

```
z <- 4
```

Aqui, **z** é uma variável local para a função **f** e tem um valor de 4.

Definindo a função f:

```
f <- function(x) {
  x + g(x)
}
```

A função **f** toma um argumento **x** e retorna a soma de **x** e o resultado da função **g(x)**.

Definindo a variável z fora da função f:

```
z <- 10
```

Aqui, **z** é uma variável fora da função **f** e tem um valor de 10. Esta é uma variável diferente da variável **z** definida dentro da função **f**. (A banca colocou só para enrolar). Tendo ou não tendo essa linha, o resultado seria o mesmo pois está fora da função.

Chamando a função f com argumento 4:

```
result <- f(4)
result
```

Quando você chama **f(4)**, **x** assume o valor de 4 na função **f**. Dentro da função **f**, ela calcula **x + g(x)**. A função **g** usa o valor de **z** que está definido no ambiente da função **f**. Portanto, o resultado final seria **4 + (4 + 4)**, o que é igual a 12.
Então, o valor retornado quando você chama **f(4)** é 12. Note que o valor de **z** fora da função **f** (que é 10) não afeta o cálculo, pois a função **g** usa o valor de **z** do ambiente onde foi definida (que é 4 no caso).

ERRADA

02 (CEBRASPE|2021|PF|ESCRIVÃO)
Com relação a conceitos de programação Python e R, julgue o item que se segue.
O código Python a seguir apresenta como resultado "True".

```
x = bool(-3)
y = bool("True"*x)
z = bool("False")
print (x and y and z)
```

Resolução

Em Python, a função **bool()** é usada para converter um valor para o tipo booleano. A função **bool()** retorna **False** se o valor passado for considerado "falso" e **True** se for considerado "verdadeiro".
Vamos analisar o código linha por linha:

1. **x = bool(-3)**: Aqui, **-3** é avaliado como verdadeiro, pois qualquer número diferente de zero é considerado verdadeiro quando convertido para booleano. Portanto, **x** será **True**.
2. **y = bool("True"*x)**: Aqui, **"True"*x** resultará em a string **"True"** repetida uma vez, pois **x** é **True**. Então, **y** será **True** também.
3. **z = bool("False")**: Aqui, a string **"False"** é avaliada como verdadeira, porque qualquer string não vazia é considerada verdadeira quando convertida para booleano. Portanto, **z** será **True**.
4. **print(x and y and z)**: Agora, a expressão **x and y and z** é avaliada. Todos os valores (**x**, **y**, **z**) são **True**, e a expressão **and** retorna o último valor verdadeiro. Portanto, o resultado final impresso será **True** e não **"True"(string).**

ERRADA

03 (CEBRASPE|2018|PF|AGENTE)
Julgue o próximo item, relativo a noções de programação Python e R.
Considere o programa a seguir, na linguagem Python.

```
letras == ["P", "F"]
for x in letras
{
print(x)
}
```

A sintaxe do programa está correta e, quando executado, ele apresentará o seguinte resultado.
PF

Resolução

Em Python não deve-se utilizar "{ }" para início e fim de função. Além disso o símbolo == refere-se a "igual a" (símbolo de igualdade). Para que o objeto letras receba o vetor, deve-se utilizar apenas o símbolo de = (igual).

Veja o código solicitado para a resposta proposta.

```
letras = ["P", "F"]
for x in letras:
    print(x, end='')
```

ERRADA

04 (CEBRASPE|2018|PF|AGENTE)
Julgue o próximo item, relativo a noções de programação Python e R.
Considere o programa a seguir, na linguagem Python.

```
if 5>2
{
print("True!")
}
```

A sintaxe do programa está correta e, quando executado, ele apresentará o seguinte resultado.

```
True!
```

Resolução

Como relatado na questão acima, as chaves "{ }" não são utilizadas no Python. Em Python a indentação substitui as chaves para delimitar os blocos de código. A indentação pode ser feita com um espaço simples ou um espaço de tabulação, tanto faz.

Veja o código solicitado para a uma resposta como a banca solicitou.

```
if 5 > 2:
    print("True!")
```

ERRADA.

05 (CEBRASPE|2018|PF|AGENTE)

Julgue o próximo item, relativo a noções de programação Python e R.
Considere o programa a seguir, escrito em R.

```
x <- TRUE
y <- FALSE
print (xy)
```

Após a execução do programa, será obtido o seguinte resultado.

```
[1] FALSE
```

Resolução

O erro da questão está na linha

```
print (xy)
```

Nesse caso **xy** está sendo considerado um objeto e este não foi declarado como os objetos **x** e **y**. Para mostrar o resultado de **x** e de **y** o correto seria

```
print (x & y)
```

Poderia também criar um objeto **xy** e para o resultado retornar `FALSE`.

Duas formas de reescrita desse código para trazer o resultado solicitado:

Forma 1: linha que solicita mostrar na tela os objetos **x** e **y**.

```
x <- TRUE
y <- FALSE
print (x & y)
```

Forma 2: declarando o objeto **xy** como FALSE

```
x <- TRUE
y <- FALSE
xy <- FALSE
print (xy)
```

ERRADA.

06 (CEBRASPE|2018|PF|AGENTE)
Julgue o próximo item, relativo a noções de programação Python e R.
Considere o programa a seguir, escrito em R.

```
x <- c (3, 5, 7)
y <- c (1, 9, 11)
print (x + y)
```

Após a execução do programa, será obtido o seguinte resultado.

```
[1] 36
```

Resolução

O resultado correto seria:

```
[1]  4 14 18
```

A soma de x+y será de cada posição do vetor

x	3	5	7
y	1	9	11
x+y	**4**	**14**	**18**

ERRADA.

07 (CEBRASPE|2018|PF|PERITO-TODAS AS ÁREAS)
Com relação à programação Python e R, julgue o item que segue.
Considere os seguintes comandos na programação em Python.

```
a = " Hello, World! "
print(a.strip())
```

Esses comandos, quando executados, apresentarão o resultado a seguir.

```
a[0]=Hello,:
a[1]=World!
```

Resolução

O resultado seria:

```
Hello, World!
```

strip() (retira espaços no início e no fim da frase).

Caso o usuário quisesse o resultado semelhante proposto pela banca, deveria utilizar:

```
a = " Hello, World! "
result = a.strip().split(',')
print(f"a[0]={result[0]}")
print(f"a[1]={result[1]}")
```

split(',') divide a string em uma lista usando a vírgula como delimitador.

Resultado

```
a[0]=Hello
a[1]= World!
```

ERRADA.

08 (CEBRASPE|2018|PF|PERITO-TODAS AS ÁREAS)
Com relação à programação Python e R, julgue o item que segue.
Considere os comandos a seguir, na linguagem R, os quais serão executados no ambiente do R, e considere, ainda, que > seja um símbolo desse ambiente.

```
> helloStr <- "Hello world!"
> print(helloStr)
```

Nesse caso, após a execução dos comandos, será obtido o resultado a seguir.

```
[1] "Hello world!"
```

Resolução

Essa já é mais tranquila:
O objeto `helloStr` recebe o valor `"Hello world!"` logo após é solicitado mostrar na tela o conteúdo do objeto `helloStr`.
CERTA.

PROFISSÃO POLICIAL

CONCURSEIRO QUE PRETENDE SER POLICIAL NÃO FAZ RATEIO