PYTHON E R PARA CIÊNCIA DE DADOS

Domine as duas linguagens de programação mais poderosas para Data Science.

FERNANDO FELTRIN

AVISOS

VERSÃO DE DEMONSTRAÇÃO

Para todos os exemplos, assim como seus respectivos códigos completos, vide o livro completo.

LIVROS

[Python do ZERO à Programação Orientada a Objetos]

[Programação Orientada a Objetos]

[Tópicos Avançados em Python]

[Ciência de Dados e Aprendizado de Máquina]

[Arrays com Python + Numpy]

[Inteligência Artificial com Python]

[Redes Neurais Artificiais]

[Análise Financeira com Python]

[Visão Computacional em Python]

[Python + TensorFlow 2.X]

CURSO

[Curso Python do ZERO à Programação Orientada a Objetos]

Mais 15 horas de vídeos objetivos que lhe ensinarão programação em linguagem Python de forma simples, prática e objetiva.

ÍNDICE

INTRODUÇÃO

A chamada computação científica é uma das áreas de especialização dentro da tecnologia da informação, que, justificando seu nome, trata de diversos segmentos, desde aplicações matemático-estatísticas até inteligência artificial por meio de redes neurais artificiais.

No âmbito da programação ao longo das últimas décadas foram se desenvolvendo uma série de ferramentas específicas para esses fins, com complexidade proporcional ao tipo de implementação.

Por parte das linguagens de programação, vimos um crescimento exponencial de linguagens como Python e R justamente por sua aplicabilidade nestes tipos de problemas computacionais.

Hoje em meados de 2020, boa parte dos cientistas de dados e entusiastas da área concentram seus esforços em programar por meio de Python e R, uma vez que ambas linguagens permitem a implementação de arquiteturas de código de forma simples em sintaxe e eficiente em performance.

Levando em consideração estes pontos, é interessante para o desenvolvedor que queira se especializar nesta área aprender tais linguagens, e esse processo não necessariamente tem de ser algo complexo e maçante.

Python e R, quando contextualizadas para aplicações de ciências de dados, oferecem meios e métodos para abstrair certos problemas codificando-os de forma simples e funcional.

Sendo assim, não desmerecendo nem descartando o uso de outras linguagens de programação, o programador que dominar Python e R em sua essência certamente estará um passo à frente dos demais, uma vez que com suas bibliotecas dedicadas a este nicho de computação, o mesmo poderá criar arquiteturas de código de altíssima complexidade de forma fácil e objetiva.

Ao longo deste pequeno livro estaremos fazendo uma rápida leitura dos fundamentos de tais linguagens,

equiparando as mesmas de modo que tudo o que será implementado em exemplos será demonstrado nas duas linguagens. Por fim, estaremos implementando de forma prática uma rede neural artificial em ambas linguagens para entendermos por completo desde a estrutura lógica até o funcionamento da mesma em seu contexto real de aplicação.

METODOLOGIA

A proposta principal deste livro é trazer de forma clara e objetiva os fundamentos das linguagens Python e R, sendo assim, estaremos entendendo tópico por tópico do referencial teórico, acompanhado dos respectivos exemplos para ambas linguagens.

Dessa forma estaremos unificando a lógica e teoria que é comum para tais linguagens, visualizando as particularidades de cada uma por meio de seus exemplos. Na verdade você irá reparar que boa parte dos códigos serão muito parecidos, senão iguais, em sua estrutura sintática e forma de execução, dependendo do contexto.

Exemplo de sintaxe comum para ambas linguagens.

Python

```
[1]  1  print('Olá Mundo!!!')
     2
```

```
Olá Mundo!!!
```

R

```
[1]  1  print('Olá Mundo!!!')
     2
```

```
[1] "Olá Mundo!!!"
```

Exemplo de sintaxe diferente entre as linguagens.

Python

```
[2]  1  lista1 = ['Ana','Carlos','Daniela','Fernando']
     2  print(lista1)
     3
```

```
['Ana', 'Carlos', 'Daniela', 'Fernando']
```

R

```
[3]  1  lista1 = c('Ana','Carlos','Daniela','Fernando')
     2  print(lista1)
     3
```

```
[1] "Ana"      "Carlos"   "Daniela"  "Fernando"
```

Nessa lógica, todo e qualquer tópico abordado terá seus exemplos apresentados dessa forma, primeiro exibindo o bloco de código em Python, em seguida em R.

Caso você esteja aprendendo do zero tais linguagens não se preocupe pois todo o essencial será devidamente explicado e exemplificado.

INSTALAÇÃO DAS DEPENDÊNCIAS

Tanto para a linguagem Python quanto para a linguagem R, é necessário realizar a instalação de neu núcleo, uma vez que tais linguagens não vem por padrão incluídas no sistema operacional Windows.

O processo de instalação é bastante simples, realizado como quando instalamos qualquer outro tipo de programa em nosso sistema.

Para instalação da linguagem Python basta acessar o site python.org e baixar o instalador referente a última versão estável da linguagem de acordo com o tipo de arquivos de seu sistema operacional.

Da mesma forma, acessando o site r-project.org é possível ter acesso ao instalador da última versão do núcleo da linguagem R.

Uma vez instaladas ambas linguagens, é necessário escolher uma IDE que seja compatível para ambas as linguagens. Em função da popularização da linguagem R, basicamente todas IDEs mais conhecidas como Visual Studio Code, PyCharm, Spyder, Vim, Emacs oferecem suporte total ou parcial a linguagem R, permitindo que você use uma só IDE para ambas linguagens.

Não menos importante, caso você busque uma IDE exclusiva para linguagem R, a comunidade em geral faz grande uso do próprio RStudio, ambiente com interface e funcionalidades muito próximas as da IDE Spyder da suíte Anaconda. Lembrando que não existe necessariamente uma IDE melhor que outra, você deve escolher a que atenda suas particularidades de modo satisfatório.

FUNDAMENTOS DE PYTHON E R

Estrutura Básica de um Programa

Uma das primeiras coisas a serem entendidas é no tocante ao que consideramos a estrutura básica de um programa.

Diferentemente de outras linguagens de programação onde é preciso se criar todo um esqueleto de configurações para finalmente começar a programar, Python e R possuem “baterias inclusas” como o jargão diz.

Em outras palavras, a partir do momento que você abre sua IDE e cria um novo arquivo você já pode imediatamente começar a programar as funcionalidades de seu programa, pois todas as estruturas básicas já estão criadas e pré-configuradas.

Partindo para prática, apenas como exemplo, vamos criar um simples programa que recebe dois números e realiza a soma dos mesmos. *Por hora não se preocupe com todas características textuais que serão apresentadas, tudo será devidamente explicado ao seu tempo, apenas entenda a partir desse exemplo como é simples a criação de um programa que, como dito anteriormente, realiza a soma de dois números.

Python

```
[5]  1  num1 = int(input('Digite um número: '))
     2  num2 = int(input('Digite outro número: '))
     3  print(num1+num2)
```

```
Digite um número: 8
Digite outro número: 3
11
```

R

```
[5]  1  num1 = as.integer(readline('Digite um número: '))
     2  num2 = as.integer(readline('Digite outro número: '))
     3  print(num1+num2)
```

```
Digite um número: 8
Digite outro número: 3
[1] 11
```

Nesse exemplo, é pedido que o usuário digite dois números, o interpretador realiza a leitura e a soma de tais números, exibindo em tela o resultado dessa operação.

Repare que um simples programa como esse, pode ser literalmente criado em apenas 3 linhas de código, sendo totalmente funcional nesse estado.

Sendo assim, tanto em Python quanto em R precisamos definir claramente qual será o objetivo de nosso programa, qual a proposta do mesmo, e a partir desse ponto podemos nos focar na criação do código em si.

De acordo com o propósito do programa iremos implementar suas funcionalidades de forma que o mesmo realize certas ações como esperado, normalmente interagindo com o usuário no que diz respeito aos comandos que serão executados pelo mesmo.

Tipos de Dados

Independentemente da linguagem de programação que você está aprendendo, na computação em geral trabalhamos com dados, e os classificamos conforme nossa necessidade.

Para ficar mais claro, raciocine que na programação precisamos separar os dados quanto ao seu tipo. Por exemplo uma string, que é o termo reservado para qualquer tipo de dado alfanumérico (qualquer tipo de palavra/texto que contenha letras e números).

Já quando vamos fazer toda e qualquer operação aritmética, precisamos tratar os números conforme seu tipo,

por exemplo o número 8, que para programação é um int (número inteiro), enquanto o número 8.2 é um float (número com casa decimal).

O ponto que você precisa entender de imediato é que não podemos misturar tipos de dados diferentes em nossas operações, porque o interpretador não irá conseguir distinguir que tipo de operação você quer realizar uma vez que ele faz uma leitura léxica e "literal" dos dados.

Por exemplo, podemos dizer que "Maria tem 8 anos", e nesse contexto, para o interpretador, o 8 é uma string, é como qualquer outra palavra dessa mesma sentença.

Já quando pegamos dois números para somá-los por exemplo, o interpretador espera que esses números sejam int ou float, mas nunca uma string, uma vez que é impossível "somar textos".

Isto pode parecer muito confuso de imediato, mas com os exemplos que iremos posteriormente abordar você irá de forma automática diferenciar os dados quanto ao seu tipo e seu uso correto.

Segue um pequeno exemplo dos tipos de dados mais comuns que usaremos tanto em Python quanto em R:

Python

Tipo	Descrição	Exemplo
Int	Número real inteiro, sem casas decimais	12
Float	Número com casas decimais	12.8
Bool	Booleano / Binário (0 ou 1)	True ou False / o ou 1
String	Texto com qualquer caractere alfanumérico	'palavra' "marca d'água"
List	Listas(s)	[2, 'Pedro', 15.9]
Dict	Dicionário(s)	{'nome':'João da Silva', 'idade': 32}

R

Tipo	Descrição	Exemplo
Int	Número real inteiro, sem casas decimais	12
Float	Número com casas decimais	12.8
Bool	Booleano / Binário (0 ou 1)	T ou F / o ou 1
String	Texto com qualquer caractere alfanumérico	'palavra' "marca d'água"
List	Listas(s)	C(2, 'Pedro', 15,9)
Array	Array(s)	array(1:10, c(1,2,3,4,5))
DataFrame	Dados em forma de tabela com linhas e colunas	data.frame(x = 1:3, y = c(1,2,3,4))

Repare também que cada tipo de dado possui uma sintaxe própria para que o interpretador os reconheça como tal.

Vamos criar algumas variáveis (que veremos em detalhes no capítulo a seguir) apenas para atribuir diferentes tipos de dados, de forma que possamos finalmente visualizar como cada tipo de dados deve ser representado em sua forma escrita.

Python

```
1  numero1 = 1987
2  numero2 = 19,90
3  palavra1 = 'Fernando'
4  frase1 = 'Seja Bem-Vindo!!!'
5  comando1 = True
6  lista1 = ['Ana', 33, 'BR', 179,99]
7  dicionario1 = {'Nome':'Maria', 'Idade':38}
8  tupla = ('Arroz', 'Feijão', 'Macarrão')
9
```

R

```r
numero1 = 1987
numero2 = 19,90
palavra1 = 'Fernando'
frase1 = 'Seja Bem-Vindo!!!'
comando1 = T
lista1 = c('Ana', 33, 'BR', 179,99)
vetor1 = c(1,2,3,4,5,6,7,8)
matriz1 = matrix(1:6, ncol=3, nrol=2)
data1 = data.frame(x = 1:3, y = c(1,2,3,4))

```

Como dito anteriormente, note como a sintaxe é muito próxima entre as linguagens, igual e/ou muito parecidas para algumas estruturas de código.

Porém note que se tratando de tipos de dados diferentes existem algumas diferenças entre as linguagens, pois não existe uma equivalência perfeita entre certos tipos de dados.

Python por ser uma linguagem mais generalista possui tipos de dados padrão e comum a praticamente todas linguagens de programação, sendo possível ampliar o suporte a mais tipos de dados fazendo o uso de algumas bibliotecas.

Já o R por sua vez por ser focado em computação científica possui nativamente algumas estruturas de dados como, por exemplo, DataFrames, enquanto não possui suporte por padrão a dicionários, por exemplo. *Também sendo possível a implementação por meio de bibliotecas externas.

Tenha em mente que uma das justificativas de existirem linguagens de programação diferentes é o propósito de suas aplicações. Enquanto Python foi desenvolvida para para todo e qualquer finalidade, R em sua idealização foi criada especificamente para ciência de dados.

Comentários

Desculpe a redundância, mas comentários dentro das linguagens de programação servem realmente para comentar determinados blocos de código, para criar anotações sobre o mesmo.

É uma prática bastante comum à medida que implementamos novas funcionalidades em nosso código ir comentando-o também, para facilitar nosso entendimento quando revisarmos o mesmo.

Essa prática também é bastante comum quando pegamos códigos de domínio público, normalmente o código virá com comentários do seu autor explicando os porquês de determinadas linhas de código.

A sintaxe para comentar nossos códigos tanto em Python quanto em R é bastante simples, bastando usar o símbolo # e em seguida escrever o devido comentário.

Falando apenas do Python, quando precisamos fazer algum comentário mais elaborado, que irá ter mais de uma linha de tamanho, podemos usar ‘’’aspas triplas antes de nosso comentário e depois dele para o terminar’’’.

A ideia de comentar determinados blocos de código é uma maneira do programador colocar anotações sobre determinadas linhas de código.

Tenha em mente que o interpretador não lê o que o usuário escreveu em forma de comentário, tudo o que estiver após # (ou entre ‘’’ no caso do Python) o interpretador irá simplesmente ignorar.

Comentário de até uma linha.

Python

```
1  ano_nascimento = 1987
2  # Ano de nascimento de Fernando.
3
```

R

```
1  ano_atual = 2020
2  # Ano em que estamos vivendo.
3
```

Comentário de mais de uma linha (apenas Python).

Python

```python
ano_nascimento = 1987
'''Fernando Feltrin nasceu no ano de 1987,
    filho de Alberto Pereira Feltrin e de
    Tânia Mara Belomé Feltrin.'''

```

Apenas concluindo essa linha de raciocínio, uma vez que tudo o que estiver sinalizado como comentário será ignorado por nosso interpretador, esta linha ou bloco de código não terá efeito sobre a funcionalidade ou performance do programa.

Variáveis / Objetos

Uma variável basicamente é um espaço alocado na memória ao qual iremos armazenar um dado, valor ou informação, simples assim.

Imagine que você tem uma escrivaninha com várias gavetas, uma variável é uma dessas gavetas ao qual podemos guardar dentro dela qualquer coisa (qualquer tipo de dado) e ao mesmo tempo ter acesso fácil a este dado sempre que necessário durante a execução de nosso programa.

Tanto Python quanto R são linguagens de programação dinamicamente tipadas, ou seja, quando trabalhamos com variáveis/objetos (itens aos quais iremos atribuir dados ou valores), podemos trabalhar livremente com qualquer tipo de dado e se necessário, também alterar o tipo de uma variável a qualquer momento.

Outra característica importante de salientar neste momento é que Python, R, assim como outras linguagens, ao longo do tempo foram sofrendo uma série de mudanças que trouxeram melhorias em sua forma de uso.

Na data de publicação deste livro estamos usando a versão 3.8 da linguagem Python, 4.0 da linguagem R, onde se

comparado com as versões anteriores das mesmas, houve uma série de simplificações em relação a maneira como declaramos variáveis, definimos funções, iteramos dados, etc…

Por fim apenas raciocine que não iremos estar nos focando em sintaxe antiga ou que está por ser descontinuada, não há sentido em nos atermos a isto, todo e qualquer código que será usado neste livro respeitará a sintaxe atual.

Declarando uma variável

A declaração básica de uma variável/objeto sempre seguirá uma estrutura lógica onde, toda variável deve ter um nome (desde que não seja uma palavra reservada ao sistema) e algo atribuído a ela (qualquer tipo de dado ou valor).

Partindo diretamente para prática, vamos ver um exemplo de declaração de uma variável:

Python / R

```
1  variavel1 = 3.14159265
2
```

Nesse caso inicialmente declaramos uma variável de nome variavel1 que recebe como atributo o número 3.14159265.

Repare que aqui, entre o nome da variável e seu respectivo atributo existe um símbolo de igual “ **=** “, nesse contexto esse símbolo é um operador de atribuição, ou seja, ele simplesmente diz a nosso interpretador que 3.14159265 é um dado associado a variavel1.

Python / R

```
1  variavel1 = 3.14159265
2
3  print(variavel1)
4
```

```
3.14159265
```

Aqui, apenas para fins de exemplo, na segunda linha do código usamos a função print( ), que veremos em detalhes nos capítulos seguintes, onde dentro de seus "parênteses" está instanciando a variável que acabamos de criar.

A execução dessa linha de código irá exibir em tela para o usuário o dado/valor que for conteúdo, que estiver atribuído a variável variavel1.

Neste caso o retorno será: 11

Conforme você progredir em seus estudos de programação você irá notar que é comum possuirmos várias variáveis, na verdade, quantas forem necessárias em nosso programa.

Não existe um limite específico, você pode usar à vontade quantas variáveis forem necessárias desde que respeite a sua sintaxe e que elas tenham um propósito no código.

Outro ponto importante é que quando atribuímos qualquer dado ou valor a uma variável o tipo de dado é implícito, ou seja, se você por exemplo atribuir a uma variável simplesmente um número 6, o interpretador automaticamente identifica esse dado como um int (dado numérico do tipo inteiro). Se você inserir um ponto '.' seguido de outro número, 5 por exemplo, tornando esse número agora 6.5, o interpretador automaticamente irá reconhecer esse mesmo dado agora como float (número de ponto flutuante).

O mesmo ocorre quando você abre aspas ' ' para digitar algum dado a ser atribuído a uma variável, automaticamente o interpretador passará a trabalhar com aquele dado o tratando como do tipo string (palavra, texto ou qualquer combinação alfanumérica de caracteres).

Posteriormente iremos ver que também é possível declarar explicitamente o tipo de um dado e até convertê-lo de um tipo para outro.

Python	**R**
nome_da_variavel = atributo num1 = 25 nome = 'Fernando'	nome_da_variavel = atributo num1 = 25 nome = 'Fernando' nome_da_variavel <- atributo num1 <- 25 nome <- 'Fernando'

Em ambas as linguagens temos a possibilidade de realizar a mesma ação escrevendo de formas diferentes, existe também a equivalência entre as linguagens, onde a sintaxe será a mesma para ambas, da mesma forma, cada linguagem tem suas particularidades que entenderemos em detalhes quando for o momento certo.

Repare no quadro acima, apenas exemplificando, existem maneiras de declarar nossas variáveis usando da mesma sintaxe em Python e em R, ou escolher a forma exclusiva de uma das linguagens, cabendo ao desenvolvedor escolher qual modo adotar para seus códigos.

Python e R também são linguagens case sensitive, ou seja, ela diferencia caracteres maiúsculos de minúsculos, logo, existem certas maneiras de declarar variáveis que são permitidas enquanto outras não, gerando conflito com o interpretador.

A grosso modo podemos declarar variáveis usando qualquer letra minúscula e o símbolo "_" underline simples no lugar do espaço.

Existem formas permitidas e formas proibidas de como iremos declarar variáveis, veremos em detalhe a seguir.

```
nome
nome2
nome_de_variavel

```

Sintaxe usual permitida para declaração de variáveis.

```
Nome              Nome de variável iniciando em letra maiúscula
NOME              Nome de variável escrita totalmente em letras maiúsculas
8dados            Nome de variável iniciando com números
_numero           Nome de variável iniciando com caractere especial
minha variavel    Nome de variável com espaços
número = 12345    Nome de variável contendo acentos
(variavel2)       Nome de variável encapsulado em chaves, parênteses ou colchetes

```

Sintaxe não permitida / não recomendada, pois pode gerar erros de interpretação.

Um último ponto a ser abordado neste momento é que ainda falando do que é permitido o uso ou não, temos também uma característica importante para ambas as linguagens que são suas palavras reservadas ao sistema.

Quando estamos falando em palavras reservadas ao sistema, raciocine que tais palavras são dedicadas a funções internas da linguagem, sendo assim, é impossível fazer o uso das mesmas, por exemplo, como nome para uma variável.

Python

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

Palavras reservadas exclusivas em Python.

R

```
1  if      else  repeat  while   function
2  for     in    next    break   TRUE
3  FALSE   NULL  Inf     NaN     NA
4  NA_integer_   NA_real_  NA_complex_
```

Palavras reservadas exclusivas em R.

Funções Básicas

Funções, falando de forma bastante básica, são linhas ou blocos de códigos aos quais executarão uma determinada ação em nosso código.

Inicialmente trabalharemos com as funções mais básicas que existem, chamadas funções de entrada e de saída, responsáveis por interagir com o usuário e exibir em tela uma determinada resposta, respectivamente.

Funções podem receber parâmetros de execução (ou não), dependendo da necessidade, uma vez que esses parâmetros nada mais são do que as informações de como os dados deverão interagir internamente para realizar uma determinada função.

Pela sintaxe tanto em Python quanto em R, “chamamos” uma função pelo seu nome, logo em seguida, entre ( ) parênteses podemos definir seus parâmetros, instanciar variáveis ou escrever linhas de código à vontade, desde que não nos esqueçamos que este bloco de código fará parte (será de propriedade) desta função…

Posteriormente trataremos das funções propriamente ditas, como criamos funções personalizadas que realizam uma determinada ação, por hora, para conseguirmos dar prosseguimento em nossos estudos, precisamos entender que existem uma série de funções pré-programadas, prontas para uso, com parâmetros internos que podem ser implícitos ou explícitos ao usuário.

As mais comuns delas, print( ) e input( ) em Python e print( ) e readline( ) em R, respectivamente nos permitirão exibir em tela o resultado de uma determinada ação de um bloco de código, e interagir com o usuário de forma que ele consiga por meio do teclado inserir dados em nosso programa.

Função print( )

Quando estamos criando nossos programas, é comum que de acordo com as instruções que programamos, recebamos alguma saída, seja ela alguma mensagem ou até mesmo a realização de uma nova tarefa.

Uma das saídas mais comuns é exibirmos, seja na tela para o usuário ou em console (em programas que não possuem uma interface gráfica), uma mensagem, para isto, na linguagem python usamos a função print( ).

Na verdade anteriormente já usamos ela enquanto estávamos exibindo em tela o conteúdo de uma variável, mas naquele capítulo esse conceito de fazer o uso de uma função estava lá apenas como exemplo e para obtermos algum retorno de nossos primeiros códigos, agora iremos de fato entender o mecanismo de funcionamento deste tipo de função.

Python / R

```
1  print('Seja Bem-Vindo!!!')
2
```

```
Seja Bem-Vindo!!!
```

Repare na sintaxe: A função print( ) tem como parâmetro (o que está dentro de parênteses) uma string com a mensagem Seja Bem-Vindo!!!.

Todo parâmetro é delimitado por ( ) parênteses e toda string é demarcada por ‘ ‘ aspas para que o interpretador reconheça esse tipo de dado como tal.

Função input( )

Em todo e qualquer programa é natural que haja interação do usuário com o mesmo, de modo que com algum dispositivo de entrada o usuário dê instruções ou adicione dados.

Começando pelo básico, em nossos programas a maneira mais rudimentar de captar os dados do usuário será por intermédio da função input( ), por meio dela podemos pedir, por exemplo que o usuário digite um dado ou valor, que internamente será atribuído a uma variável.

Python

```
1  nome = input('Digite o seu nome: ')
2
```

Inicialmente declaramos uma variável de nome nome que recebe como atributo a função input( ) que por sua vez dentro tem uma mensagem para o usuário.

Assim que o usuário digitar alguma coisa e pressionar a tecla ENTER, esse dado será atribuído a variável nome.

Em seguida, de acordo com o que vimos anteriormente, por meio da função print( ) podemos exibir em tela uma mensagem definida, concatenada ao nome digitado pelo usuário e atribuído a variável nome.

Python

```
1  nome = input('Digite o seu nome: ')
2
3  print('Bem vindo', nome)
4

Digite o seu nome: Fernando
Bem vindo Fernando
```

Executando esse bloco de código o retorno é, como esperado, a mensagem Bem vindo Fernando.

Função readline( ) e função paste( )

Para a linguagem R especificamente, não temos a função input( ) nativa do Python, a função em si claro que temos, porém não escrita pela mesma sintaxe.

Em R, sempre que quisermos interagir com o usuário fazendo com que o mesmo digite algo faremos o uso da função readline( ).

R

```
1  nome = readline('Digite o seu nome: ')
2
```

Novamente, de início criamos uma variável de nome nome que por sua vez chama a função readline( ) com uma mensagem ao usuário.

Da mesma forma que o exemplo anterior, assim que o mesmo digitar algo e pressionar ENTER, o conteúdo será atribuído a variável nome.

R

```
1  nome = readline('Digite o seu nome: ')
2
3  paste("Bem Vindo", nome)
4
```

```
Digite o seu nome: Fernando
'Bem Vindo Fernando'
```

Reproduzindo o mesmo exemplo, agora em R, temos outra particularidade. Como dito anteriormente, a linguagem R tem suporte a função print( ) porém ela é mais restrita quanto ao modo de uso em comparação ao Python, pois não suporta exibir juntamente diversos tipos de dados.

Para contornar isso, em R sempre que quisermos exibir em tela um conteúdo composto de mais de um tipo de dado devemos fazer o uso da função paste( ).

Sendo assim, executando esse bloco de código, o retorno é a mensagem Bem Vindo Fernando.

Apenas fazendo um adendo, em Python, como parâmetro de nossa função print( ) podemos instanciar múltiplos tipos de dados, inclusive um tipo de dado mais de uma vez.

Python

```
1  nome = input('Digite o seu nome: ')
2  idade = input('Digite a sua idade: ')
3
4  print('Bem vindo', nome, 'você tem', idade, 'anos.')
5
```

```
Digite o seu nome: Fernando
Digite a sua idade: 33
Bem vindo Fernando você tem 33 anos.
```

R

```
1  nome = readline('Digite o seu nome: ')
2  idade = readline('Digite a sua idade: ')
3
4  paste('Bem Vindo', nome, 'você tem', idade, 'anos.')
```

```
Digite o seu nome: Fernando
Digite a sua idade: 33
'Bem Vindo Fernando você tem 33 anos.'
```

O mesmo em R deve ser feito pela função equivalente past( ).

R

```
4  print('Bem Vindo', nome, 'você tem', idade, 'anos.')
```

```
Digite o seu nome: Fernando
Digite a sua idade: 33
Warning message in print.default("Bem Vindo", nome, "você tem", idade,
"NAs introduced by coercion"
Error in print.default("Bem Vindo", nome, "você tem", idade, "anos."):
Traceback: invalid 'digits' argument

1. print("Bem Vindo", nome, "você tem", idade, "anos.")
2. print.default("Bem Vindo", nome, "você tem", idade, "anos.")
```

Tentar executar o mesmo bloco de código por meio da função print( ) irá gerar um erro.

O ponto a se destacar aqui é que, embora as linguagens tenham muitas semelhanças, existem particularidades entre as mesmas que devem ser respeitadas, pois nem todo código em Python terá sua sintaxe igual a R e vice versa, porém sempre haverá maneiras equivalentes de se realizar as mesmas ações em ambas linguagens.

Python	**R**
print( )	print( )
print('Olá Mundo!!!') #suporta todo e qualquer tipo de dado	print('Olá Mundo!!!') #suporta apenas um tipo de dado
	paste('Bem-Vindo', variável) #suporta múltiplos tipos de dados

Interação entre variáveis

Agora entendidos os conceitos básicos de como fazer o uso de variáveis, como exibir seu conteúdo em tela por meio da função print( ) e até mesmo interagir com o usuário via função input( ), hora de voltarmos a trabalhar os conceitos

de variáveis, aprofundando um pouco mais sobre o que pode ser possível fazer a partir delas.

A partir do momento que declaramos variáveis e atribuímos valores a elas, podemos fazer a interação entre elas (interação entre seus atributos).

Python / R

```
1  ano = 2020
2  nascimento = 1987
3
```

Inicialmente criamos uma variável de nome ano que recebe como atributo 2020 e uma segunda variável nascimento que recebe como atributo o número 1987.

Python / R

```
4  idade = ano - nascimento
5
```

Na sequência criamos uma variável de nome idade que faz a soma entre ano e nascimento realizando a operação instanciando as próprias variáveis. A partir deste momento o interpretador irá trabalhar com os dados/valores que estiverem associados/guardados nessas variáveis. O resultado da soma será guardado em soma.

Python / R

```
6  print(idade)
7
8  print(ano - nascimento)
9
```

A partir daí podemos simplesmente exibir em tela via função print( ) o valor de idade, assim como podemos realizar esta mesma operação diretamente dentro da função print( ).

Python / R

```
ano = 2020
nascimento = 1987

idade = ano - nascimento

print(idade)

print(ano - nascimento)

```

```
33
33
```

O resultado, como esperado para ambas funções print( ) é o número 33, referente a subtração dos valores entre ano e nascimento.

Lembrando que para criar um retorno um pouco mais elaborado, com uma mensagem que incorpora o resultado obtido, temos de respeitar as diferenças sintáticas entre Python e R.

Python

```
ano = 2020
nascimento = 1987

idade = ano - nascimento

print('Sua idade é: ', idade)

```

```
Sua idade é:  33
```

R

```
ano = 2020
nascimento = 1987

idade = ano - nascimento

paste('Sua idade é: ', idade)

```

```
'Sua idade é:  33'
```

Em resumo, para combinar diferentes tipos de dados em um retorno, em Python usaremos da função print( ) enquanto em R usaremos da função paste( ).

Operadores

Anteriormente vimos que toda e qualquer linguagem de programação possui uma sintaxe própria, composta por meios permitidos ou não permitidos de se escrever o código e entendemos também que existem certas palavras que são reservadas ao sistema representando funções internas.

Quando estamos trabalhando com toda a simbologia acessória que existe para fazer a comunicação entre variáveis e funções estamos falando de diferentes tipos de operadores.

Operadores de Atribuição

A atribuição padrão de um determinado dado/valor para uma variável, pela sintaxe do Python, é feita através do operador " = ", repare que o uso do símbolo de igual (=) usado uma vez tem a função de atribuidor, já quando realmente queremos usar o símbolo de igual para sua função de igualar operandos na aritmética, usaremos ele duplicado (==).

Em R podemos perfeitamente usar do operador " = " para atribuição de dados/valores a variáveis, porém, é bastante comum encontrar códigos defasados em repositórios que usam do operador " <- " para realizar este tipo de atribuição.

Como dito anteriormente, a medida que as linguagens evoluíram, foram implementadas novas funcionalidades assim como foram simplificadas outras. No caso do R é importante salientar que ambas formas são permitidas, cabendo novamente ao desenvolvedor fazer uso da qual achar mais confortável.

Python / R

```
1  salario = 995
2  aumento = 37.65
3
4  print(salario + aumento)
5
6  salario_atual = salario + aumento
7
```

Usando de um exemplo parecido com o anterior, simplesmente perceba que para as variáveis salario e aumento existem valores atribuídos e isso se dá por meio do operador = .

Internamente, o interpretador irá ler o conteúdo dessas variáveis e trabalhar com os mesmos normalmente. O nome em si das variáveis é só uma forma de criar um objeto que guarda uma informação e o nome em si só faz sentido para o desenvolvedor, por parte do interpretador o que importa para o mesmo são os dados/valores.

Aproveitando este tópico para entender mais uma particularidade, repare que quando realizamos as operações entre as variáveis diretamente dentro da função print( ), este valor será descartado logo em seguida.

A partir do momento que realizamos uma operação e atribuímos a mesma a uma variável, esta variável guardará esse dado para reutilização.

R

```r
salario <- 995
aumento <- 37.65

```

Apenas demonstrando o operador de atribuição alternativo em R.

Operadores Lógicos

Em resumo, operadores lógicos são aqueles que nos retornarão uma expressão de validação, normalmente True para verdadeiro / certo / ligado / positivo, 1 / etc… e False para falso / errado / desligado / negativo / 0 / etc… dependendo seu contexto, normalmente comparando ou equivalendo certos dados.

Python	**R**
> Maior que	> Maior que
< Menor que	< Menor que
>= Igual ou maior a	>= Igual ou maior a
<= Igual ou menor a	<= Igual ou menor a
== Igual a	== Igual a
!= Diferente de	!= Diferente de
True Verdadeiro	T Verdadeiro
False Falso	F Falso

Fazendo o uso correto dos operadores podemos realizar praticamente qualquer verificação entre dados de variáveis.

Python / R

```
1  num1 = 12
2  num2 = 12
3
4  print(num1 == num2)
5
```

Apenas como exemplo, aqui são declaradas duas variáveis de nomes num1 e num2, cada uma delas com um valor atribuído para si.

Na sequência, por meio da função print( ) estamos perguntando se num1 é igual a num2. o interpretador fará a comparação entre os valores atribuídos a essa variável e retornará uma expressão lógica.

```
True
```

Como esperado, haja visto que os valores das variáveis são iguais, perguntado se de fato eram, temos como retorno True.

Python / R

```
4  print(num1 != num2)
5

False
```

Perguntado se num1 é diferente de num2, o retorno é False.

Python / R

```
1  print(30 > 29)
2

True
```

30 é maior que 29? True.

Python / R

```
1  var1 = 100
2  var2 = 28
3
4  print(var2 <= var1)
5
```

```
True
```

O conteúdo de var2 é menor ou igual ao de var 1? True.

Operadores Lógicos Complexos:

Python

```
1  var1 = 100
2  var2 = 200
3
4  print(var1 > var2 and var2 > 55)
5
```

```
False
```

O valor de var1 é maior que o de var2? E o valor de var2 é maior que 55? Nesse caso o retorno é False pois uma das proposições é falsa. Para um retorno True as duas proposições deveriam ser verdadeiras.

Operadores Aritméticos

Operadores aritméticos, como o nome sugere, são aqueles que usaremos para realizar operações matemáticas em meio a nossos blocos de código.

Tanto Python quanto R por padrão já vem com bibliotecas pré-alocadas que nos permitem a qualquer

momento fazer operações matemáticas simples como soma, subtração, multiplicação e divisão.

Para operações de maior complexidade também é possível importar bibliotecas externas que irão implementar tais funções.

Por hora, vamos começar do início, entendendo quais são os operadores que usaremos para realizarmos pequenas operações matemáticas.

Operador	Função
+	Realiza a soma de dois números
-	Realiza a subtração de dois números
*	Realiza a multiplicação de dois números
/	Realiza a divisão de dois números

Como mencionado anteriormente, a biblioteca que nos permite realizar tais operações já vem carregada quando iniciamos nosso IDE, nos permitindo a qualquer momento realizar os cálculos básicos que forem necessários.

Por exemplo:

Soma:

Python / R

```
1  print(8 + 15)
2
```

```
23
```

Subtração:

Python / R

```
print(8 - 15)

```

```
-7
```

Multiplicação:

Python / R

```
print(30 * 4)

```

```
120
```

Divisão:

Python / R

```
print(30 / 4)

```

```
7.5
```

Operações com mais de 2 operandos:

Python / R

```
print(5 + 2 * 7)

```

```
19
```

O resultado será 19 porque pela regra matemática primeiro se fazem as multiplicações e divisões para depois efetuar as somas ou subtrações, logo 2 x 7 são 14 que somados a 5 se tornam 19.

Operações Dentro de Operações:

Python / R

```
1  print((5 + 2) * 7)
2
```

```
49
```

O resultado será 49 porque inicialmente é realizada a operação dentro dos parênteses (5 + 2) que resulta 7 e aí sim este valor é multiplicado por 7 fora dos parênteses.

Exponenciação:

Python / R

```
1  print(4 ** 5)
2
```

```
1024
```

Elevando 4 a 5ª potência (4 x 4 x 4 x 4 x 4) o resultado é 1024.

Divisão Inteira:

Python

```
1  print(9.4 / 3)
2  print(9.4 // 3)
3
```

```
3.1333333333333333
3.0
```

R

```
print(9.4 / 3)
print(9.4 %/% 3)
```

```
[1] 3.133333
[1] 3
```

O resultado para a expressão da primeira linha é uma divisão normal, enquanto para a segunda linha será retornado um valor arredondado.

Resto da Divisão:

Python

```
print(10 % 3)
```

```
1
```

R

```
print(10 %% 3)
```

```
[1] 1
```

Dividindo 10 por 3 temos 9, restando 1.

Python	**R**
+ Soma	+ Soma
- Subtração	- Subtração
* Multiplicação	* Multiplicação
/ Divisão	/ Divisão
// Divisão exata	%/% Divisão exata
% Módulo da divisão	%% Módulo da divisão

Operadores Relacionais

Operadores relacionais basicamente são aqueles que fazem a comparação de dois ou mais elementos, dados/valores de duas ou mais variáveis, possuindo uma sintaxe própria que deve ser respeitada para que não haja erros de interpretação.

O retorno obtido no uso desses operadores também será booleano, True para verdadeiro e False para falso.

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

Usando como referência o console, operando diretamente nele, ou por meio de nossa função print( ) em meio ao código, podemos fazer alguns experimentos.

Python / R

```
1  print(3 > 4)
2
```

```
False
```

O resultado é False, pois 3 não é maior que 4.

Python / R

```
1  print(7 >= 4)
2
```

```
True
```

O resultado é True, pois 7 é igual ou maior que 4.

Python / R

```
1  print(9 >= 9)
2
```

```
True
```

True, pois 9 não é maior, mas é igual a 9.

Operadores de Membro

Ainda dentro do contexto de operadores podemos realizar a consulta dentro de uma lista obtendo a confirmação True ou a negação False quanto a um determinado elemento pertencer ou não na mesma.

Python

```
1  lista = [1, 2, 3, 'Ana', 'Maria']
2
3  print(2 in lista)
4
```

```
True
```

Por meio do operador in estamos verificando se, nesse caso, o número 2 é um elemento da lista. Nesse caso o retorno é True uma vez que 2 é o segundo elemento ordenado dessa lista.

Python

```
1  lista = [1, 2, 3, 'Ana', 'Maria']
2
3  print(2 not in lista)
4
```

```
False
```

Dentro da mesma lógica, por meio do operador not in podemos realizar a chamada negação lógica. Nesse caso, estamos perguntando se 2 não pertence aos elementos de lista, e como esperado o retorno é False, pois 2 é um elemento da variável lista.

Estruturas Condicionais

Quando aprendemos sobre lógica de programação e algoritmos, era fundamental entendermos que toda ação tem uma reação (mesmo que apenas interna ao sistema), dessa forma, conforme abstraíamos ideias para código, a coisa mais comum era nos depararmos com tomadas de decisão, que iriam influenciar os rumos da execução de nosso programa.

Muitos tipos de programas se baseiam em metodologias de estruturas condicionais, são programadas todas possíveis tomadas de decisão que o usuário pode ter e o programa executa e retorna certos aspectos conforme o usuário vai aderindo a certas opções.

Lembre-se das suas aulas de algoritmos, digamos que, apenas por exemplo o algoritmo ir_ate_o_mercado está sendo executado, e em determinada seção do mesmo existam as opções: SE estiver chovendo vá pela rua nº1, SE NÃO estiver chovendo, continue na rua nº2.

Esta é uma tomada de decisão onde o usuário irá aderir a um rumo ou outro, mudando as vezes totalmente a execução do programa, desde que essas possibilidades estejam programadas dentro do mesmo.

Não existe como o usuário tomar uma decisão que não está condicionada ao código, logo, todas possíveis tomadas

de decisão dever ser programadas de forma lógica e responsiva, prevendo todas as possíveis alternativas.

If

Uma das partes mais legais de programação, sem sombra de dúvidas, é quando finalmente começamos a lidar com estruturas condicionais. Uma coisa é você ter um programa linear, que apenas executa uma tarefa após a outra, sem grandes interações e desfecho sempre linear, como um script passo-a-passo sequencial. Já outra coisa é você colocar condições, onde de acordo com as variáveis o programa pode tomar um rumo ou outro.

Como sempre, começando pelo básico, tanto em Python quanto em R a sintaxe para trabalhar com estruturas condicionais é bastante simples se comparado a outras linguagens de programação, basicamente temos os comandos if (se), elif (o mesmo que else if / mas se) e else (se não) e os usaremos de acordo com nosso algoritmo demandar tomadas de decisão.

A lógica de execução sempre se dará dessa forma, o interpretador estará executando o código linha por linha até que ele encontrará uma das palavras reservadas mencionadas anteriormente que sinaliza que naquele ponto existe uma tomada de decisão, de acordo com a decisão que o usuário indicar, ou de acordo com a validação de algum parâmetro, o código executará uma instrução, ou não executará nada, ignorando esta condição e pulando para o bloco de código seguinte.

Python

```
1  a = 33
2  b = 34
3  c = 35
4
5  if b > a:
6  |  print('b é maior que a')
7
```

```
b é maior que a
```

R

```
1  a = 33
2  b = 34
3  c = 35
4
5  if (b > a){
6  |  print('b é maior que a')
7  }
```

```
[1] "b é maior que a"
```

Inicialmente repare na sintaxe, aqui temos uma grande diferença em como escrevemos nossas estruturas condicionais se comparado Python e R.

Declaradas três variáveis de nomes a, b e c, respectivamente, com seus devidos valores atribuídos, em seguida temos o bloco de código de nossa estrutura condicional.

Basicamente criamos uma estrutura com a palavra reservada ao sistema if, onde em seguida é colocada a condição que será testada, caso essa condição seja alcançada, é executada uma simples mensagem por meio da função print( ). Caso essa condição não fosse alcançada, nesse exemplo o interpretador simplesmente iria ignorar essa sentença e pular para o próximo bloco de código.

Nesse caso, a expressão pode ser lida como “se o valor de b for maior que o valor de a, então mostre em tela a mensagem da função print( )”.

Else

Python

```python
a = 33
b = 30
c = 35

if b > a:
    print('b é maior que a')
else:
    print('b é menor que a')

```

```
b é menor que a
```

R

```r
a = 33
b = 30
c = 35

if (b > a){
  print('b é maior que a')
}else{
  print('b é menor que a')
}
```

```
[1] "b é menor que a"
```

Apenas para fins de exemplo, agora alteramos o valor de b, e implementamos uma estrutura condicional um pouco mais elaborada.

Agora, caso a primeira sentença seja verdadeira (b maior que a) é impresso a referente mensagem, caso essa condição não seja verdadeira, então o interpretador irá pular para a linha onde consta o else, exibindo a referente mensagem.

Nesse caso, como agora b não é maior que a, a mensagem exibida é a referente ao else.

Elif

Python

```python
a = 30
b = 20
c = 50

if b > a:
  print('b é maior que a')
elif b == 20:
  print('b é igual a 20...')
else:
  print('b é menor que a')

```

```
b é igual a 20...
```

R

```r
a = 30
b = 20
c = 50

if (b > a){
  print('b é maior que a')
}else if(b == 20){
  print('b é igual a 20')
}else{
  print('b é menor que a')
}
```

```
[1] "b é igual a 20"
```

Até então vimos estruturas onde a condição era simples, em outras palavras, ou uma coisa ou outra, porém em situações reais nada mais normal do que, dependendo do contexto, termos mais de duas alternativas para nossas condições.

Por parte sintática, isso é feito por meio de elif em Python e por else if em R.

Nesse exemplo, temos três possíveis desfechos para nossa estrutura condicional, inicialmente se b for maior que a, logo após se b for igual a 20 e finalmente se b for menor que a.

Repare que, em estruturas condicionais que usam elif / else if, o interpretador irá sequencialmente testar cada uma das possibilidades, quando encontrar a verdadeira ele para a execução do código naquela linha.

Em outras palavras, note que ao ser feita a validação de que o valor de b era igual a 20, não foi gerado o retorno de else, mesmo b sendo menor que a de acordo com o mesmo. Nesse caso o retorno foi b é igual a 20.

Python

```python
idade = int(input('Digite a sua idade: '))

if idade < 12:
    print('criança')
elif idade < 18:
    print('adolescente')
elif idade >= 18:
    print('adulto')
elif idade > 70:
    print('idoso')
else:
    print('idade inválida')

```

```
Digite a sua idade: 33
adulto
```

R

```
idade = readline('Digite sua idade: ')

if (idade < 12){
  print('criança')
}else if(idade < 18){
  print('adolescente')
}else if(idade >= 18){
  print('adulto')
}else if(idade > 70){
  print('idoso')
}else{
  print('idade inválida')
}
```

```
Digite sua idade: 33
[1] "adulto"
```

Apenas como exemplo, em meio ao if e else (primeira condição e retorno caso nenhuma condição anterior tenha sido validada) podemos usar quantos elifs / else if forem necessários.

Estruturas condicionais compostas

Também é possível combinar o uso de operadores AND e OR para elaborar estruturas condicionais mais complexas (de duas ou mais condições válidas).

Python

```python
nome = input('Digite o seu nome: ')
idade = int(input('Digite a sua idade: '))

if nome == 'Maria' and idade < 18:
    print('Maria é de menor')
elif nome == 'Maria' and idade >= 18:
    print('Maria é adulta')
elif nome == 'Carlos' and idade < 18:
    print('Carlos é de menor')
elif nome == 'Carlos' and idade >= 18:
    print('Carlos é adulto')
else:
    print('Pessoa desconhecida')

```

```
Digite o seu nome: Carlos
Digite a sua idade: 33
Carlos é adulto
```

Para esse exemplo, simplesmente pedimos que o usuário digite um nome e uma idade, caso seu nome seja Maria ou Carlos, será feita a verificação da idade e exibida a mensagem referente. Caso o usuário digite um nome diferente de Maria ou Carlos, o retorno será a mensagem em else.

Nesses casos de condicionais compostas pelo operador AND, as duas expressões condicionais devem ser True para que ocorra a validação daquela linha de código.

Python

```python
nome = input('Digite o seu nome: ')

if nome == 'Maria' or nome == 'Carlos':
    print('Usuário reconhecido')
else:
    print('Usuário desconhecido')

```

```
Digite o seu nome: Maria
Usuário reconhecido
```

Outra possibilidade é por meio do operador OR realizar uma condicional composta, diferentemente do operador AND, onde era necessário que as duas proposições fossem verdadeiras para ocorrer a validação, pelo operador OR, como o nome já sugere, basta uma ou outra expressão ser verdadeira para que se valide a sentença.

Nesse exemplo, caso o usuário digite Maria ou digite Carlos, é gerado o retorno para isso, caso digite qualquer outro nome diferente destes, é retornado o que está indentado para else.

Nesse exemplo, usuário digitando Maria, o retorno é Usuário reconhecido.

Condicionais dentro de condicionais

Python

```python
nome = input('Digite o seu nome: ')
idade = int(input('Digite sua idade: '))

if nome == 'Maria' or nome == 'Carlos':
    print('Usuário reconhecido')
    if nome == 'Maria' and idade < 18:
        print('Maria é de menor')
    elif nome == 'Maria' and idade >= 18:
        print('Maria é de maior')
    elif nome == 'Carlos' and idade < 18:
        print('Carlos é de menor')
    elif nome == 'Carlos' and idade >= 18:
        print('Carlos é de maior')
else:
    print('Usuário desconhecido')

```

```
Digite o seu nome: Carlos
Digite sua idade: 33
Usuário reconhecido
Carlos é de maior
```

Avançando com nossos exemplos, em Python é perfeitamente possível criar estruturas condicionais compostas aninhadas umas dentro de outras.

R

```r
nome = readline('Digite o seu nome: ')
idade = readline('Digite sua idade: ')

if (nome == 'Maria'){
  print('Olá Maria!!!')
  if (idade < 18){
    print('Maria é de menor')
  }else if (idade >= 18){
    print('Maria é de maior')
  }
}else{
  print('Usuário desconhecido')
}
```

```
Digite o seu nome: Maria
Digite sua idade: 33
[1] "Olá Maria!!!"
[1] "Maria é de maior"
```

A linguagem R suporta estruturas condicionais dentro de estruturas condicionais, porém não oferece suporte a operadores como AND e OR dentro de estruturas condicionais, sendo assim, temos alguns limites em relação ao exemplo via Python.

Estruturas de Repetição

While

Nas linguagens Python e R temos dois tipos de comandos para executar comandos em loop (executar repetidas

vezes uma mesma instrução) o while e o for, inicialmente vamos entender como funciona o while.

While do inglês, em tradução livre significa enquanto, ou seja, enquanto uma determinada condição for válida, a ação continuará sendo repetida.

Python

```python
num = 1

while num < 10:
    print(num)
    num += 1
```

```
1
2
3
4
5
6
7
8
9
```

R

```r
num = 1

while(num < 10){
  print(num)
  num = num + 1
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

Declarada uma variável de nome num, inicialmente com valor 1 atribuído para si, criaremos um laço de repetição por meio de while.

Preste atenção na sintaxe, while é uma palavra reservada ao sistema e quando escrevemos a mesma devemos colocar logo a seguir uma estrutura condicional para a mesma.

Nesse caso, simplesmente definimos que enquanto o valor de num for menor que 10 será executado o bloco de código indentado a while. Neste bloco temos nossa função print( ) mostrando os valores processados e repare que existe uma estrutura para atualizar o valor de num cada vez que essa estrutura de repetição for executada.

A função em si repetirá esse processo até o momento em que o último valor de num seja o mais próximo e ao mesmo tempo inferior a 10.

Toda estrutura de repetição em while terá uma condição a ser alcançada e dentro de si um mecanismo que atualiza o valor de uma variável.

For

O comando for será muito utilizado quando quisermos trabalhar com um laço de repetição onde conhecemos os seus limites, ou seja, quando temos um objeto, normalmente em forma de lista ou dicionário e queremos que uma variável percorra cada elemento dessa lista/dicionário interagindo com o mesmo.

Python

```
1  numeros = [1,2,3,4,5,6,7,8]
2
3  for numero in numeros:
4      print(numero)
5
```

```
1
2
3
4
5
6
7
8
```

R

```
1  numeros = c(1,2,3,4,5,6,7,8)
2
3  for(numero in numeros){
4      print(numero)
5  }
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
```

Como exemplo temos uma variável de nome numeros que recebe uma lista de números como atributo.

Em seguida é criado um laço de repetição for que cria uma variável temporária numero, percorre todos os dados/valores de nossa lista numeros e a cada laço de repetição mostra em tela qual era o número lido e associado a variável temporária numero.

O resultado, como esperado, é que a cada execução um novo número é exibido em tela, sequencialmente,

Listas / Arrays

Listas em Python são o equivalente a Arrays em outras linguagens de programação, inclusive R, mas calma que se você está começando do zero, e começando com Python, esse tipo de conceito é bastante simples de entender.

Listas são um dos tipos de dados aos quais iremos trabalhar com frequência, uma lista será um objeto que permite guardar diversos dados dentro dele, de forma organizada e indexada.

É como se você pegasse várias variáveis de vários tipos e colocasse em um espaço só da memória do seu computador, dessa maneira, com a indexação correta, o interpretador consegue buscar e ler esses dados de forma muito mais rápida do que trabalhar com eles individualmente.

Python

```
1  minhalista = [ ]
2
```

R

```
1  minhalista = c( )
2
```

Repare na sintaxe, aqui temos outro exemplo de onde as linguagens Python e R se diferem. Basicamente em Python tudo o que estiver dentro do símbolo de colchetes [ ] será lido pelo interpretador como tipo de dado lista, já em R, o marcador que identifica para o interpretador que tal dado está em formato de lista é o c( ).

Python

```python
lista2 = [1987, 'Maria', 19.90, ['arroz','feijão','carne'] ]

print(lista2)

```

```
[1987, 'Maria', 19.9, ['arroz', 'feijão', 'carne']]
```

R

```r
lista2 = c(1987, 'Maria', 19.90, c('arroz','feijão','carne'))

print(lista2)

```

```
[1] "1987"   "Maria"  "19.9"   "arroz"  "feijão" "carne"
```

Como dito anteriormente, uma das principais funcionalidades de uma lista é guardar diversos dados associados a apenas uma variável.

Nesse contexto, uma lista suporta que seja guardado em si qualquer tipo de dado, inclusive listas dentro de listas.

Um ponto a destacar aqui é que, existem formas equivalentes de se criar listas tanto em Python quanto em R, porém a maneira com que esses dados serão tratados por parte de interpretador é muito diferente,

Raciocine que em Python a partir do momento que temos uma lista composta de vários dados, podemos trabalhar com os mesmos sempre dentro do contexto de que são objetos de uma lista, elementos a serem usados como parte de uma lista.

Em R, é possível dispor elementos em uma lista, porém uma simples lista é lida como um vetor, onde cada elemento da mesma é um elemento independente, se comportando diferente dos elementos de uma lista em Python.

O ponto que devemos observar aqui é que como Python é uma linguagem mais generalista, a estrutura de dados de uma lista basicamente serve para guardar múltiplos dados de forma indexada em uma variável, enquanto em R, uma vez que seu foco é computação científica, usar de listas/arrays em R tem abordagens diferentes por parte do interpretador,

pois o mesmo em primeira instância espera que esses dados sejam numéricos e para aplicação de funções matemáticas.

Dicionários

Enquanto listas e tuplas são estruturas indexadas, um dicionário, além da sintaxe também diferente, se resume em organizar dados em formato chave : valor. Assim como em um dicionário normal você tem uma palavra e seu respectivo significado, aqui a estrutura lógica será a mesma.

A sintaxe de um dicionário é definida por chaves { }, deve seguir a ordem chave:valor e usar vírgula como separador, para não gerar um erro de sintaxe.

Python

```
1  dicionario1 = {'nome':'Fernando', 'idade':33}
2
3  print(dicionario1)
4
```

```
{'nome': 'Fernando', 'idade': 33}
```

Inicialmente criamos uma variável de nome dicionario1, atribuído para a mesma estão alguns dados em formato de dicionário, não só pela conotação de colchetes, mas os dados internos estão dispostos em formato chave : valor.

O retorno, como esperado, são os dados em suas respectivas posições.

*Em R, a criação de estruturas nos mesmos moldes é feita pela biblioteca externa dict.

Conjuntos Numéricos

Se você lembrar das aulas do ensino médio certamente lembrará que em alguma etapa lhe foi ensinado sobre conjuntos numéricos, que nada mais era do que uma forma de categorizarmos os números quanto suas características (reais, inteiros, etc...).

Na programação isto se repete de forma parecida, uma vez que quando queremos trabalhar com conjuntos numéricos normalmente estamos organizando números para poder aplicar funções matemáticas sobre os mesmos.

Um conjunto é ainda outra possibilidade de armazenamento de dados que temos, de forma que ele parece uma lista com sintaxe de dicionário, mas não indexável e que não aceita valores repetidos, confuso não? A questão é que como aqui podemos ter valores repetidos, assim como conjuntos numéricos que usávamos no ensino médio, aqui podemos unir dois conjuntos, fazer a intersecção entre eles, etc…

Python

```
1  num_pares = {2,4,6,8,10,12,14,16,18,20}
2
3  print(num_pares)
4

{2, 4, 6, 8, 10, 12, 14, 16, 18, 20}
```

Repare na sintaxe, para esse exemplo é criada a variável num_pares que por sua vez recebe um conjunto numérico de números pares de 2 a 20.

Por parte sintática, note que tais dados estão delimitados por chaves igual se faz em um dicionário, porém, como aqui os dados não estão dispostos em chave : valor, o interpretador do Python automaticamente identificará tais dados como elementos de um conjunto.

Por meio da função print( ) é possível exibir o conteúdo de num_pares, como demonstrado na imagem acima.

Em R, basicamente temos dados em forma equivalente a uma lista ou vetor, uma vez que temos dados guardados de forma ordenada e sequencial.

Funções

Já vimos anteriormente as funções print( ) e input( ), duas das mais utilizadas quando estamos criando estruturas básicas para nossos programas. Agora que você já passou por outros tópicos e já acumula uma certa bagagem de conhecimento, seria interessante nos aprofundarmos mais no assunto funções, uma vez que existem muitas possibilidades novas quando dominarmos o uso das mesmas, já que o principal propósito é executar certas ações por meio de funções.

Uma função, independentemente da linguagem de programação, é um bloco de códigos que podem ser executados e reutilizados livremente, quantas vezes forem necessárias. Uma função pode conter ou não parâmetros, que nada mais são do que instruções a serem executadas cada vez que a referenciamos associando as mesmas com variáveis, sem a necessidade de escrever repetidas vezes o mesmo bloco de código da mesma funcionalidade.

Para ficar mais claro imagine que no corpo de nosso código serão feitas diversas vezes a operação de somar dois valores e atribuir seu resultado a uma variável, é possível criar uma vez esta calculadora de soma e definir ela como uma função, de forma que eu posso posteriormente no mesmo código reutilizar ela apenas mudando seus parâmetros.

Seguindo o nosso raciocínio, temos funções prontas pré carregadas, funções prontas que simplesmente podemos importar para nosso código e certamente haverão situações onde o meio mais rápido ou prático será criar uma função própria personalizada para uma determinada situação.

Basicamente, quando necessitamos criar uma função personalizada, em Python ela começará com a declaração do prefixo def, já em R, a palavra reservada function vem como atributo da variável, antes dos parâmetros. Dessa forma temos um meio para que o interpretador assuma que a partir desses marcadores está sendo criada uma função.

Uma função personalizada pode ou não receber parâmetros (variáveis instanciadas dentro de parênteses), pode retornar ou não algum dado ou valor ao usuário e por fim, terá um bloco de código indentado para receber suas instruções.

Python

```
1  def Soma(a, b):
2      soma = a + b
3      return soma
4
5  a = int(input('Digite um Número: '))
6  b = int(input('Digite outro Número: '))
7
8  resultado = Soma(a, b)
9
10 print(resultado)
11
```

```
Digite um Número: 9
Digite outro Número: 13
22
```

R

```r
soma = function(x, y){
  soma = x + y
  return(soma)
}

a = as.integer(readline('Digite um número: '))
b = as.integer(readline('Digite outro número: '))

resultado = soma(a, b)

print(resultado)

```

```
Digite um número: 9
Digite outro número: 13
[1] 22
```

Usando de um exemplo bastante básico, vamos supor que temos um simples programa que recebe dois números digitados pelo usuário e então retorna a soma destes números.

Para isso inicialmente criamos a função soma, que recebe dois valores e internamente realiza a soma dos mesmos.

Por fim é criada todas as demais estruturas para que haja a devida interação com o usuário, ao final, exibindo o valor esperado.

Bibliotecas, Módulos e Pacotes

Em Python, assim como em R, de acordo com a nossa necessidade, existe a possibilidade de trabalharmos com as bibliotecas básicas já inclusas nas linguagens, ou importarmos outras bibliotecas que nos tragam novas funcionalidades.

Python como já dissemos anteriormente algumas vezes, é uma linguagem de programação "com pilhas inclusas", ou

seja, as bibliotecas básicas que você necessita para grande parte das funções básicas já vem incluídas e pré-alocadas, de forma que basta chamarmos a função que queremos para de fato usá-la, o mesmo ocorre em R, mas com algumas particularidades, haja visto que R é uma linguagem mais especializada se comparado com Python.

Dependendo do caso e seu contexto, podemos precisar fazer o uso de funções que não são carregadas por padrão ou que até mesmo são de bibliotecas externas a ambas linguagens, sendo assim necessário que importemos tais bibliotecas para termos acesso às suas funções.

Python

```
1  pip install matplotlib
2
3  import matplotlib
4
```

R

```
1  Install.packages("ggplot2", dependencies=T)
2
3  library(ggplot2)
4
```

Apenas como exemplo, supondo que para uma determinada aplicação nosso código necessitará de ferramentas para geração de gráficos, podemos importar uma biblioteca específica para isso.

Em Python importaremos a biblioteca MatPlotLib, em R, a biblioteca GGPlot, equivalente a MatPlotLib.

Repare na sintaxe, há uma grande diferença na forma como realizamos tais importações em Python e em R.

Tanto Python quanto R em todas suas versões já contam com funções matemáticas pré-etabelecidass, porém para usarmos algumas funções específicas ou constantes matemáticas, é necessário importá-las.

Também é perfeitamente possível criar manualmente nossas próprias bibliotecas/módulos e pacotes, porém

dependendo muito das funcionalidades das mesmas, é um re-serviço criar algo manualmente quando já se existe tal estrutura de código pronta.

APLICAÇÃO COM PYTHON / R

Partindo para um exemplo real, vamos implementar um perceptron (estrutura mais básica de uma rede neural artificial) realizando a equivalência em ambos os códigos.

Para esse perceptron, estaremos sequencialmente criando suas estruturas de dados de entrada, processamento dos mesmos por uma função matemática e o desfecho esperado por meio de uma função de ativação.

```python
# Python

entradas = [1, 7, 5]
pesos = [0.8, 0.1, 0]

```

```r
# R

entradas = c(1, 7, 5)
pesos = c(0.8, 0.1, 0)

```

Inicialmente criamos as variáveis entradas e pesos com seus respectivos atributos, nesse caso, como estamos atribuindo vários dados para uma variável o processo é feito colocando tais dados em forma de lista. Prestar a atenção no que diz respeito a sintaxe para ambas linguagens, uma vez que uma lista em Python é identificada por [ ] enquanto em R o mesmo tipo de dado é identificado por c( ).

```python
# Python

def soma(entradas, pesos):
    s = 0
    for i in range (3):
        #print(entradas[i])
        #print(pesos[i])
        s += (entradas[i] * pesos[i])
    return s
```

```r
# R

soma = function(entradas, pesos) {
    s = 0
    for (i in 1:3) {
        #print(entradas[i])
        #print(pesos[i])
        s = s + (entradas[i] * pesos[i])
    }
    return (s)
}
```

Em seguida criamos a função que irá receber os dados das variáveis entradas e pesos, respectivamente, retornando o resultado da soma de cada elemento dos mesmos.

Lembrando que em Python para termos um resultado permanente devemos instanciar tal função por uma variável.

Por parte de sintaxe também existem algumas pequenas diferenças de notação quanto a simbologia usada para separar as estruturas de código, assim como algumas equivalências que deverão ser realizadas.

Como o objetivo é extrair cada elemento da lista e o usar individualmente para função de soma, o método mais fácil é desempacotar o mesmo por meio de um laço for.

Note que em Python usamos da notação in range ( ) para percorrer todos elementos da lista, já em R o equivalente a isso é feito diretamente definindo a posição de índice a ser lida.

Em R quando estamos lendo elementos de uma lista via índice, assim como em Python, o último elemento serve apenas como marcador para identificar o último elemento e o final da lista, sendo assim, como aqui temos dados referentes a entradas (índice 1) e pesos (índice 2), definimos que o laço for percorra do índice 1 ao 3.

Lembrando que em Python a leitura de índice de uma lista começa em 0, enquanto em R o primeiro elemento de índice é 1.

Ao final do processo a variável s tem seu valor atualizado, retornando esse valor atribuído a s.

```python
# Python

produto_escalar = soma(entradas, pesos)

```

```r
# R

produto_escalar = function(entradas, pesos)

```

Na sequência declaramos uma variável de nome produto escalar que chama a função codificada anteriormente, parametrizando a mesma com os respectivos dados de entrada e pesos.

```python
# Python

resultado_soma = produto_escalar(entradas, pesos)

```

```r
# R

resultado_soma = produto_escalar(entradas, pesos)

```

De forma parecida criamos uma variável de nome resultado_soma que por sua vez instancia nossa variável produto_escalar.

```python
# Python

def funcao_degrau(soma):
  if (soma >= 1):
    return 1
  return 0
```

```r
# R

funcao_degrau = function(soma) {
  if (soma >= 1) {
    return (1)
  }
  return (0)
}
```

Na sequência criamos nossa função degrau, função essa responsável por receber os dados processados anteriormente e com base neles definir o desfecho a ser tomado, podendo ser 0 ou 1, True ou False, etc… pois como aqui estamos trabalhando sobre um problema computacional de classificação binária, o retorno será “ou uma coisa ou outra”.

Note que a estrutura é bastante simples, usando de uma estrutura condicional if é feita a verificação de “se o valor atribuído para soma é igual ou maior que 1, retorne 1, caso contrário, retorne 0”.

```python
# Python

resultado_final = funcao_degrau(resultado_soma)

```

```r
# R

resultado_final = funcao_degrau(resultado_soma)

```

Por fim, criamos uma variável de nome resultado_final que terá atribuído para si o retorno gerado por nossa funcao_degrau( ).

```
1.7777777777777777
1
```

O resultado, como esperado, para nossa função soma( ) é 1.7, assim como para nossa função funcao_degrau( ) é 1, pois dentro da probabilidade definida (soma >= 1) o retorno esperado é a ativação deste perceptron.

TABELA DE EQUIVALÊNCIA PYTHON / R

Declaração de Variáveis	
Python nome_da_variavel = atributo num1 = 25 nome = ‘Fernando’	R nome_da_variavel = atributo num1 = 25 nome = ‘Fernando’ nome_da_variavel <- atributo num1 <- 25 nome <- ‘Fernando’
Tipos de Dados	

Python

int / float
```
num1=1987           #int
num1=1987           # int
num1=int(1987)      # int
num2=19.90          # float
num2 = float(19.90) # float
```

String
```
nome = 'Fernando'
frase = 'Estudando a linguagem R'
```

Lista
```
lista1 = [1, 2, 3, 4, 5]
```

Dicionário
```
Dicionario1 = {'Nome': 'Fernando',
                       'idade': 33}
```

Booleano
```
estado = True
estado = False
```

Em Python DataFrames e Matrizes são criados a partir de bibliotecas externas como Numpy ou Pandas.

Vetor / Lista
```
vetor1 = [1,2,3,4,5,6,7,8]
# Se comporta exatamente como uma lista
#  vetor1[4]  retornará  5  pois  o índice começa sempre em 0
```

R

numeric
```
num1=1987         #numeric
num1=1987L        # int
num1=as.integer(1987) # int
num2=19.90        # float
num3=19           # internamente é float
```

Character
```
nome = 'Fernando'  # ou "Fernando"
frase = 'Estudando a linguagem R'
```

Lista / Vetor
```
lista1 = c(1, 2, 3, 4, 5)
```

Em R a estrutura de dado equivalente a um dicionário com chaves/valores deve ser feita a partir da biblioteca externa dict ou usando de lista dentro de lista, uma camada para chaves, outra para valores.

Logical
```
estado = T     # ou TRUE
estado = F     # ou FALSE
```

DataFrame
```
data = data.frame(x = 1:3, y = c(1,2,3,4))
# Tipo de dado em forma de tabela com linhas e colunas indexadas.
```

Matriz
```
matriz1  =  matrix(1:6,  ncol=3, nrol=2)
# Matriz de linhas e colunas, seus dados devem ser todos de mesmo tipo.
```

Vetor
```
vetor1 = c(1,2,3,4,5,6,7,8)
#  Unidimensional e composto por apenas um tipo de dado
#  vetor1[4]  retornará  4  pois  o índice começa sempre em 1
```

# Verificando o tipo 	
type(nome_do_objeto)
	Array
	array1 = array(1:10, c(1,2,3,4,5))
	# Parecido com uma matriz, porém com suporte a mais dimensões.
	
	Complex
	Var1 = 5 + 2i #5 é a parte inteira, 2 é a imaginária
	
	# Verificando o tipo
	typeof(nome_do_objeto)
	ou
	class(nome_do_objeto)

Funções Básicas de Entrada e Saída	
Python **print( )** print(‘Olá Mundo!!!’) # suporta todo e qualquer tipo de dado	**R** **print( )** print(‘Olá Mundo!!!’) # suporta apenas um tipo de dado paste(‘Bem-Vindo’, variável) # suporta múltiplos tipos de dados
Python **input( )** nome = input(‘Digite o seu nome: ‘)	**R** **readline( )** nome = readline(‘Digite o seu nome: ‘)
Operador de Atribuição	

Python	**R**
= nome = ‘João’	= ou <- nome = ‘João’ nome <- ‘João’

Operadores Lógicos

Python	**R**
> Maior que < Menor que >= Igual ou maior a <= Igual ou menor a == Igual a != Diferente de True Verdadeiro False Falso	> Maior que < Menor que >= Igual ou maior a <= Igual ou menor a == Igual a != Diferente de T Verdadeiro F Falso

Operadores Aritméticos

Python	**R**
+ Soma - Subtração * Multiplicação / Divisão // Divisão exata % Módulo da divisão	+ Soma - Subtração * Multiplicação / Divisão %/% Divisão exata %% Módulo da divisão

Estruturas Condicionais

Python

if

```
If condição:
        comando


if 50 < 100:
        print('50  é  menor  que
100')
```

R

if

```
If(condição){
        comando
}

If(50 < 100){
        Print('50 é menor que 100')
}
```

else

```
n = input('Digite um número: ')

if n < 100:
   print('O  número  é  menor  que
100')
else:
   print('O  número  é  maior  que
100')
```

else

```
n = readline('Digite um número: ')

if(n < 100){
  print('O número é menor que 100')
}else{
  print('O número é maior que 100')
}
```

while

```
num1 = 10

while num1 <= 10:
        print(num1)
        num1 = num1 + 1
```

while

```
num1 = 10

while(num1 <= 10){
        print(num1)
        num1 = num1 + 1
}
```

Laços de Repetição

Python

for

```
numeros = [1, 2, 3, 4, 5]

for i in numeros:
        print(i)
```

R

for

```
numeros = c(1, 2, 3, 4, 5)

for(i in numeros){
        print(i)
}
```

Funções

Python

```
def Soma(a, b):
        soma = a + b
        return soma

a = int(input(‘Digite um Número: ‘))
b = int(input(‘Digite outro Número: ‘))

resultado = Soma(a, b)

print(resultado)
```

R

```
soma = function(x, y = 0) {
  soma = x + y
  return(soma)
}

a = as.integer(readline('Digite um número: '))
b = as.integer(readline('Digite outro número: '))

resultado = soma(a, b)

print(resultado)
```

Importação de Bibliotecas, Módulos e Pacotes

Python

```
pip install matplotlib

import matplotlib
```

R

```
Install.packages(“ggplot2”, dependencies=T)

library(ggplot2)
```

CONCLUSÃO

Disponível apenas na versão completa do livro.

BÔNUS

Disponível apenas na versão completa do livro.