Programação em Python

Roteiro de Estudos

Autor: Me. Leandro da Conceição Cardoso
Revisor: Jaime Gross

Olá! Seja bem-vindo(a) à nossa disciplina de Programação em Python. A partir dela você passará a interligar alguns assuntos das principais características de programação utilizando Python, conhecendo as suas estruturas de dados e, também, de controle. Isso permitirá a você consolidar seus conhecimentos fundamentais associando Programação em Python ao uso na área de *Data Science*.

Caro(a) estudante, ao ler este roteiro você vai:

- compreender os fundamentos de programação utilizando Python;
- estudar estruturas de dados (tipos, listas, *strings*, tuplas);
- aprender estruturas de controle condicionais;
- compreender as estruturas e laços de repetição;
- refletir sobre estruturas de dados e de controle associadas à utilização em Data Science.

Introdução

Ao adotar uma linguagem de programação como Python, por exemplo, para um determinado projeto, é importante conhecer a sua escrita e como se dá o seu entendimento, se realmente é uma linguagem robusta que atenderá todas as especificações do projeto. Logicamente, é

necessário saber se possui qualidade de execução nas aplicações que são desenvolvidas, se elas permitem fácil atualização e estão preparadas caso precisem lidar com alto volume de dados, prevendo o crescimento da demanda de uso que uma aplicação possa ter.

Para ter bons resultados, a linguagem deve prover ferramentas que facilitam a lógica de programação, assim como a tradução do raciocínio em um algoritmo, por exemplo. Estas e outras questões serão abordadas ao longo deste roteiro de estudos.

Fundamentos de Programação utilizando Python

A linguagem de programação Python foi desenvolvida com objetivo de proporcionar legibilidade e produtividade, ou seja, criar códigos eficientes de maneira fácil e rápida, sem deixar de manter o seu alto nível. É uma linguagem interpretada, de *script*, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte, mas, mesmo assim, não é de difícil aprendizado, pelo fato de sua sintaxe ser clara e possuir tipos predefinidos robustos, mas simples de usar. É possível testar, rapidamente, trechos de código utilizando o interpretador Python, uma linguagem expressiva com abstrações de alto nível que usa código curto e de rápido desenvolvimento; desta forma, traduzir o raciocínio em um algoritmo passa a ser tarefa fácil.

As características que tornam a programação por Python simples fazem com que os profissionais foquem seus conhecimentos, estudos e esforços no que realmente importa, ou seja, na lógica de programação; pois o desenvolvimento de aplicações é a etapa que necessita de maior trabalho, sendo uma das mais importantes na criação de quaisquer projetos. Além de possuir volumes grandes de documentação e comunidade bastante ativa para sanar dúvidas que possam aparecer, essa linguagem também é utilizada por grandes empresas. A maneira de codificar a escrita de extensões para Python em C e C++ é uma tarefa simples, principalmente para projetos de aplicações que exigem o máximo de desempenho, ou quando for necessário interfacear alguma ferramenta nessas linguagens. A escrita de extensões para Python em C e C+ permite que a aplicação desenvolvida nessa linguagem de programação possa ser executada em múltiplas plataformas, sem a necessidade de desenvolver alterações no código para cada plataforma que será executada a aplicação.

Python é uma linguagem de programação livre, mas, mesmo assim, não deixa de disponibilizar recursos para tratamento de exceções, como o moderno mecanismo de tratamento de erros, que

inclui herança múltipla e também é orientado a objetos (MENEZES, 2014). Ao contrário das linguagens compiladas que convertem a escrita do código dos programas para uma determinada plataforma, como Linux ou Windows, por exemplo, Python converte o código do programa em *bytecodes*. Estes são processados por meio de um interpretador, fazendo com que o aplicativo seja executado em diferentes plataformas utilizando quase nenhuma ou pouca alteração. Além disso, grande parte do sistema operacional Linux distribuído já vem com o interpretador do Python instalado. Já para outros sistemas operacionais, como Windows, caso não possua o interpretador, se faz necessário um instalador, no qual é incluída a documentação, o interpretador e um ambiente de desenvolvimento integrado (IDE) – o IDLE, com recursos como autocompletar (autocomplete), coloração de sintaxe (*syntax highlighting*) e um *debugger*. A programação que utiliza Python permite o uso de bibliotecas GUI – a tabela a seguir apresenta alguns exemplos disponíveis.

Biblioteca	Descrição
Tkinter	Módulo padrão para GUI no Python
PyGTK	Interface para a biblioteca GTK+
PyQt	Interface para a biblioteca Qt
wxPython	Interface para a biblioteca wxWidgets
Etk	Interface para a biblioteca EFL
Wax	Construído para simplificar o uso do wxPython
Kivy	Toolkit multiplataforma

Tabela 1 - Bibliotecas de GUI disponíveis para Python
Fonte: Beazley (2013, p. 31).

A programação com Python é bem difundida e usada em grandes sistemas, como YouTube; por grandes conglomerados da área de tecnologia e *internet*, como Google (parte dos *crawlers*); e por sites com grupos de usuários, como Yahoo!. A NASA também utiliza alguns competentes.

Os *scripts* Python são utilizados em *softwares* de edição de imagem, como o GIMP, além de, para alguns sistemas operacionais, ser um componente padrão. Curiosidade: também é utilizado como sistema de gerenciamento de conteúdo. Por exemplo, Python é extensivamente utilizado na computação gráfica da produção dos filmes da série *Star Wars*, pois é uma linguagem de fácil tradução e raciocínio em um algoritmo.

LIVRO

Consumindo a API do Zabbix com Python

Autor: Janssen dos Reis Lima

Editora: Brasport

Ano: 2016

Comentário: este livro apresenta exemplos práticos de como extrair dados do ambiente de monitoramento e como recuperar e modificar informações via programação. Fornece acesso a dados históricos de um API utilizando a linguagem de programação Python. Recomendo a leitura do Capítulo 2 relacionado na biblioteca.

Onde encontrar?

Biblioteca Virtual da Laureate.

ARTIGO

Python: História e Ascendência.

Autor: Danilo Morais da Silva

Ano: 2018

Comentário: este artigo apresenta a história da linguagem de programação Python e, de certa forma, a sua popularização, que ocorreu porque os fundamentos dessa linguagem são uma proposta simplificada que colabora significativamente para os recursos tecnológicos disponíveis na atualidade. Recomendo a leitura da página 96, sobre o tópico “Fundamentos da Linguagem Python”.

ACESSAR

Estruturas de Dados (Tipos, Listas, *Strings*, Tuplas)

A programação em Python permite utilizar vários tipos de estruturas de dados que podem resolver um determinado problema, sendo úteis em diferentes situações (como listas, *sets*, dicionários, tuplas etc.). A lista é considerada a estrutura de dados básica do Python, pois armazena os dados de maneira sequencial, para que cada elemento tenha sua posição na lista, que recebe o nome de índice. O primeiro elemento deve ser o índice zero, e a sequência segue a partir daí. A cada elemento inserido na lista, um valor é incrementado, podendo fazer o armazenamento de quaisquer tipos de dados primitivos, sejam eles *strings*, inteiros, *floats* etc. Para declarar uma lista na programação em Python, utilize a seguinte sintaxe:

```
nome_da_lista = [] # Criação de uma lista vazia
```

```
nome_da_lista = [1, 2, 3] # Criação de uma lista de inteiros

nome_da_lista = [1, "Olá, estudante!", 1.1] # Criação de uma lista com vários tipos diferentes
```

O Python permite a criação de listas dentro de outras listas já existentes, que recebem o nome de *nested lists* e sua sintaxe é a seguinte:

```
nome_da_lista = ["Pós Graduação EAD Laureate",  [1, 2, 3], ["outra_lista"]]
```

A estrutura de uma tupla é similar à de uma lista, se diferenciando apenas em relação aos elementos inseridos, que na tupla não podem ser alterados e na lista, podem. Embora ambas as estruturas sejam muito semelhantes, os usos são diferentes, tendo em vista que as listas devem ser empregadas quando é necessário que as sequências sejam homogêneas; enquanto as tuplas são estruturas de dados heterogêneas. Dessa forma, a utilização da tupla é para o armazenamento de dados de tipos diferentes, e a da lista é para dados do mesmo tipo. Assim, as tuplas possuem algumas vantagens em comparação às listas, que são destacadas a seguir:

- são imutáveis, a iteração sobre elas é mais rápida, ganhando desempenho, se comparado com as listas;
- podem ser utilizadas como chave para um dicionário, pois seus elementos são imutáveis, não sendo possível utilizar listas;
- caso seja necessário armazenar dados que não serão alterados, é recomendado usar uma tupla, pelo fato de garantir a proteção dos dados em relação a alterações posteriores.

A declaração de uma tupla é similar à sintaxe da criação de uma lista. A simplicidade continua a mesma e o uso dos colchetes (listas) é alterado, pois na tupla são utilizados parênteses, como no exemplo a seguir:

```
nome_da_tupla = (1, 2, 3) #tupla de inteiros

nome_da_tupla = (1, "olá", 1.5) #tupla heterogênea
```

Os *sets* na programação em Python são uma coleção de itens não ordenada, parcialmente imutável e que, por isso, não pode conter elementos duplicados. Os *sets* possuem permissão de adição e remoção de elementos; e normalmente são utilizados com operações matemáticas de união, interseção e diferença simétrica. Existem duas maneiras de declarar *set*, sendo a primeira semelhante às tuplas e às listas, mas diferenciando com o uso das chaves {} para indicar os elementos do *set*, conforme a sintaxe a seguir:

```
nome_do_set = {1, 2, 3, 4}
```

A segunda maneira de declaração de *set* é utilizar o método *set* presente no Python, como apresentado no exemplo a seguir:

```
nome_do_set = set([1, 2, 3, 4])
```

Já os dicionários se referem às coleções de itens desordenados; seu principal ponto de diferenciação, em comparação a listas, *sets* e tuplas, é o fato de um elemento dentro de um dicionário possuir uma chave atrelada a ele, como se fosse uma espécie de identificador. Dessa forma, o seu uso é recomendado no momento em que é necessário o armazenamento de dados de maneira organizada e que possui identificação única, que é o que acontece em banco de dados. Veja a seguir dois exemplos de sintaxe:

```
nome_do_dicionario = {1: 'Ingrid', 2: 'Mônica'}
```

```
nome_do_dicionario = {'nome': 'Ingrid', 'sobrenome': 'Bitencourt'}
```

ou

```
nome_do_dicionario = dict({1: 'Ingrid', 2: 'Mônica'})
```

```
nome_do_dicionario = dict({'nome': 'Ingrid', 'sobrenome': 'Bitencourt'})
```

Na programação em Python existem diversas estruturas prontas que podem ser utilizadas. Cada uma é a solução para um determinado tipo problema. Dessa maneira, o seu desenvolvimento possui fácil tradução ao raciocínio em um algoritmo.

LEITURA

Programação em Python: Fundamentos e Resolução de Problemas

Autor: Ernesto Costa

Editora: Editora de Informática Ltda. (FCA)

Ano: 2015

Comentário: este livro apresenta os conceitos teóricos e exercícios que proporcionam uma consolidação do conhecimento e aprofundamento dos temas aqui abordados. Indicamos a leitura do tópico 1.3, iniciado na página 16, que aborda o assunto sobre a linguagem de programação Python.

ACESSAR

Estruturas de Controle Condicionais

Ao programar, em muitas situações é necessário que blocos específicos de códigos sejam executados apenas se uma determinada condição for verdadeira, é nesse momento que utilizamos estruturas de controle condicionais, como *if-else* e *elif*. Sendo o *if* uma estrutura de controle condicional que permite a avaliação de uma expressão, dependendo do resultado, ele executará uma ação específica. Veja o exemplo da utilização do *if* na programação em Python, a seguir, para verificar se a variável idade é menor que 22; se for positivo, será apresentada a mensagem na tela de “Idade permitida”, e se for negativo, o código deve seguir normalmente (e, para isso, deve desconsiderar a linha 3).

```
1.  idade = 18
```

```
2. if idade < 22:
3. print('Idade permitida')
```

Analisando o código, essa estrutura é constituída pela palavra reservada *if*, após uma condição e dois pontos (:), e as linhas logo abaixo formam o bloco que instruirá as ações que devem ser tomadas, caso a condição seja satisfatória. Para que isso aconteça, é necessário que elas sejam identadas corretamente, de acordo com a linguagem de programação Python, e somente a linha 3 é executada. Caso seja necessário executar as demais linhas, se idade for menor do que 22, elas também devem estar no mesmo nível de identação da linha 3.

O exemplo anterior é de uma estrutura de controle condicional para situação positiva — ou seja, se a condição foi atendida. No entanto, caso a condição seja insatisfatória e não foi especificado nenhum comportamento no código, é preciso usar a reservada *else*, como representado a seguir:

```
1. idade = 18
2. if idade >= 22:
3. print(Idade permitida')
4. else:
5. print('Idade não permitida')
```

Nessa situação, se a condição avaliada que está na linha 3 não for atendida, o fluxo alternativo que o código deve seguir é definido. Ou seja, se a idade não for maior ou igual a 22, o bloco abaixo da palavra reservada *else* deverá ser executado, assim, terá somente a instrução de mensagem da linha 5.

Caso ainda exista mais que uma condição alternativa que será necessária para a verificação, utilize o *elif* para avaliar as expressões intermediárias antes de usar o *else*, conforme o exemplo:

```
1. idade = 18
2. if idade < 12:
3. print('criança')
4. elif idade < 18:
5. print('adolescente')
6. elif idade < 60:
7. print('adulto')
8. else:
```

9. print('idoso')

No exemplo anterior, na segunda linha é definida a primeira condição (idade < 12), se for atendida, a aplicação deve ser para a linha 4 e passará para avaliação da próxima condição *elif*. Se esta for positiva, fará com que o bloco logo abaixo, identificado na quinta linha nesse caso, seja executado. Mas, se a condição ainda não for atendida, terá outra alternativa identificada na linha 6, que deve ser avaliada. Isso fará o bloco abaixo ser executado, se ela for atendida. Assim, se nenhuma das condições for satisfeita, a aplicação partirá para a linha 8, executando o que é definido pelo *else*.

LIVRO

Lógica de Programação e Estruturas de Dados: Com Aplicações em Java.

Autor: Sandra Puga e Gerson Rissetti

Editora: Pearson

Ano: 2003

Comentário: embora o livro aborde a lógica de programação e estruturas de dados com aplicações em Java, é indicada a leitura do Capítulo 1, denominado “Introdução à lógica”, (página 17 a 24). Esse capítulo apresenta conceitos de lógica que são empregados em todas as linguagens de programação, inclusive a Python.

Onde encontra?

Biblioteca Virtual da Laureate.

Estruturas e Laços de Repetição

Quando é necessário utilizar um conjunto de instruções, ou apenas uma instrução que precise ser executada várias vezes, é indicado o uso de estruturas e laços de repetição, ou simplesmente *loop*, o que permitirá a execução de um determinado bloco de codificação no momento que uma condição é atendida. Na programação em Python, os laços de repetição são codificados com as estruturas de repetição *for* e *while*. O laço denominado *for* tem a função de permitir que a codificação percorra os itens de uma coleção, para que cada um faça a execução do bloco de código que foi declarado no laço da repetição. Possui a seguinte sintaxe:

```
1. for variavel in lista
2. comandos
```

No momento que a lista de valores é percorrida, a variável indicada no *for* receberá, a cada iteração, um item da coleção. Desta maneira, a execução de algum processamento com esse elemento é permitida. Observe a codificação que percorre a lista nomes e a mensagem de cada elemento.

```
1. nomes = ['Klessia', 'Virginia', 'Adriana']
2. for n in nomes:
3. print(n)
```

Observe que a variável definida na linha 1 se refere a uma lista que iniciou com uma sequência de valores do tipo *string*, e a instrução *for* percorre todos esses elementos, um por vez e, em cada caso, atribuindo o valor do item à variável n, que apresenta a mensagem em seguida. Dessa forma, o resultado é a mensagem de todos os nomes contidos na lista. Também é permitida a adição de instrução *else* no final do *for*, fazendo com que a execução de um bloco de codificação seja feita no final da iteração, como apresentado na sintaxe a seguir:

```
1. nomes = ['Klessia', 'Virginia', 'Adriana']
2. for n in nomes:
3. print(n)
4. else:
5. print("Todos os nomes foram listados com sucesso")
```

Nesse exemplo, na linha 1, é definida uma variável que armazenará uma lista de nomes; em seguida, a instrução *for* percorre todos esses elementos e atribui um a um à variável n, que

apresentará a mensagem, como pode ser visto na terceira linha. Depois do laço de repetição terminar, o bloco de codificação contida na instrução *else* será acionado, apresentando a mensagem na tela.

As estruturas e laços de repetição que utilizam o comando *while* permitem que uma série de instruções faça a sua execução durante uma condição atendida. No momento que o resultado dessa condição passa a ser falso, a execução do laço de repetição é interrompida, como apresentado na sintaxe a seguir:

1. contador = 0
2. while (contador < 5):
3. print(contador)
4. contador = contador + 1

Observando esse código, é possível constatar que, no momento em que a variável contador, inicializada com 0, for menor do que 5, as instruções das terceira e quarta linhas serão executadas. Ao analisar a quarta linha, é incrementado o valor da variável contador, de modo que, em algum momento, seu valor igualará o número 5. No momento em que acontece a verificação da segunda linha, o laço será interrompido. Se faltar essa codificação, a condição de parada não será alcançada, o que gerará o chamado de *loop* infinito. Evite que isso aconteça, pois leva ao congelamento e finalização da aplicação.

É importante salientar que, na programação em Python, para informar o bloco de codificação pertencente ao *while*, é necessário somente indentar a codificação de acordo como o mostrado no exemplo, mesmo quando utilizada a estrutura *While-else*.

LIVRO

Lógica de Programação Algorítmica

Autor: Sérgio Guedes

Editora: Pearson

Ano: 2014

Comentário: este livro aborda o conteúdo relacionado a processamento de dados, sistemas algébricos e relacionais e conceitos de algoritmo com exemplos de estudo de caso. Recomendo a leitura da Unidade 2, denominada “Estrutura e dados e repetição”, na página 42, que é um aprofundamento deste tópico.

Onde encontrar?

Biblioteca Virtual da Laureate.

Estruturas de Dados e de Controle Associados à Utilização em *Data Science*

Na programação em Python, existem muitas bibliotecas disponíveis, o que é um grande facilitador para o *Data Science*, em que podem ser utilizadas as estruturas de dados e de controle associadas. Assim, é importante conhecer algumas bibliotecas, pois também ajudam no desenvolvimento de aplicações de fácil tradução do raciocínio em um algoritmo. Uma das bibliotecas é a denominada SciPy, que permite o *download* de cada pacote de maneira individual. Essas bibliotecas

disponibilizam e fornecem apoio para matemática, ciência, estatística e engenharia, e as principais são: Matplotlib, IPython, NumPy, SimPy, Pandas, Scikit-learn e Beautiful Soup.

A NumPy se caracteriza por ser pacote fundamental para computação científica com Python, pois permite manipular matriz n-dimensional, ou seja, uma matriz multidimensional eficiente e veloz que autoriza a vetorização de operações aritméticas, portanto, é fundamental para o trabalho em Data Science. Possui ferramentas que integram código C / C++ e Fortran, permite a transferência de dados para bibliotecas externas escritas nessas linguagens, assim como utilitários de álgebra linear, e tem capacidade de gerar números aleatórios. Pode ser usada como um recipiente multidimensional de dados genéricos, os tipos de dados arbitrários podem ser definidos, permitindo assim que NumPy, de maneira transparente e rápida, faça uma integração ampla de variedade de bancos de dados. Essa biblioteca não fornece a funcionalidade de análise de dados de alto nível, mas fornece operações com matrizes, tornando os processos de análise de dados bem eficiente.

A biblioteca SciPy fornece manipulação de matriz n-dimensional de maneira veloz e prática. Foi constituída com objetivo de trabalhar com matrizes e fornecer diversas rotinas numéricas de uso eficiente e simples, como rotinas de integração e otimização numérica. Disponibiliza módulos para otimização, álgebra linear, integração e outras tarefas comuns em Data Science. Já Pandas é uma biblioteca de alto desempenho que dispõe de suporte para estruturas de dados e ferramentas de análise de dados; dessa forma, é otimizada para executar tarefas de *Data Science* de maneira eficiente e veloz. O princípio básico do Pandas é dispor de análise de dados e suporte à modelagem para Python de forma semelhante a outras línguas, como o R.

A Scikit-learn é um módulo Python para Machine Learning, ele fornece conjunto de algoritmos de aprendizagem de máquina comum aos usuários por meio de interface consistente e é um facilitador de implementação rápida de algoritmos em seu conjunto de dados. A sua lista de algoritmos inclui ferramentas para diversas tarefas de aprendizagem de máquina padrão, como *clustering*, classificação, regressão etc. A Matlplotlib é um módulo Python para visualizar dados, ela proporciona a criação de gráfico, histogramas e outras figuras profissionais de maneira fácil e permite a configuração de personalizar cada aspecto de uma figura. No momento em que é utilizada no IPython, a Matplotlib possui recursos interativos, como *zoom* e visão panorâmica, tem suporte em todos os sistemas operacionais e permite salvar gráficos para vetor comum e formatos gráficos, como pdf, jpg, png, bmp, gif, svg etc.

A SymPy é uma biblioteca Python destinada à matemática, o seu principal objetivo é se tornar um sistema de álgebra computacional *full-featured*, mantendo a codificação simples quanto possível, desta maneira, é mais compreensível e facilmente extensível. É escrita em Python por completo, não requerendo nenhuma biblioteca externa. Já a biblioteca Beautiful Soup dispõe de formas para

análise de dados HTML ou XML, de forma que Python possa compreender. Assim, é permitido trabalhar com dados com base em *tags*, como os encontrados em arquivos html e xml.

A IPython-Notebook executa múltiplas linhas/blocos de codificação em diferentes células, permitindo movê-los para cima ou para baixo e obter resultados em tempo real logo abaixo da célula. De certa forma, é um organizador para os profissionais de *Data Science*, permite a escrita em R, SQL, Scala e outras linguagens em IPython-Notebook, o que faz com que o fluxo de trabalho seja muito mais fácil e eficiente.

ARTIGO

Difusão da linguagem Python no desenvolvimento de sistemas web: pesquisa exploratória em empresas brasileiras

Autoras: Anatália Saraiva Martins Ramos, Idelmárcia Dantas de Oliveira e Chiara Ângela de Carvalho Sales

Ano: 2006

Comentário: recomendamos a leitura do Tópico 2, denominado “Sistemas web e a Linguagem Python”, que apresenta a visão de alguns atores que apontam que a linguagem Python se distancia de Perl e de outras linguagens dinâmicas pela sua facilidade de manutenção, é reconhecida como uma linguagem limpa em termos de legibilidade e também é muito modular, como Java e C.

ACESSAR

Conclusão

A programação em Python tem escrita de fácil entendimento, mas não deixa de ser robusta e possuir qualidade de execução nas aplicações que são desenvolvidas, sendo utilizada por grandes empresas que trabalham com tecnologia. Dessa maneira, vem se tornando cada vez mais usada pelos profissionais da área ao desenvolver soluções, pois eles podem direcionar mais a sua energia para a lógica de programação, ao invés de se preocupar com a tradução do raciocínio em um algoritmo, por exemplo. Essas e outras questões foram apresentadas ao longo deste roteiro de estudos.

Referências Bibliográficas

BEAZLEY, D. B. K. P**ython Cookbook**. 3. ed. Sebastopol: O'Reilly, 2013.

COSTA, E. **Programação em Python**: fundamentos e resolução de problemas. Lisboa: FCA, 2015.

GUEDES, S. **Lógica de programação algorítmica**. São Paulo: Pearson, 2014.

LIMA. J. R. **Consumindo a API do Zabbix com Python**. Rio de Janeiro: Brasport 2016.

MENEZES N. L. **Introdução à programação com Python**: algoritmos e lógica de programação para iniciantes. São Paulo: Novatec. 2. ed. 2014.

PUGA, S; RISSETTI, G. **Lógica de programação e estruturas de dados**: com aplicações em Java. Pearson. 2003.

RAMOS, A. S. M.; OLIVEIRA, I. D.; SALES, C. A. C. Difusão da linguagem Python no desenvolvimento de sistemas web: pesquisa exploratória em empresas brasileiras. *In*: SIMPEP, 8., 2006, Bauru. **Anais** [...]. Bauru, 2006. Disponível em: https://simpep.feb.unesp.br/anais/anais_13/artigos/1053.pdf. Acesso em: 20 ago. 2020.

SILVA, D. M. Python: História e ascendência. **Programar**: revista portuguesa de programação, ed 59, p. 96-98, fev. 2018. Disponível em: https://www.revista-programar.info/static/downloads/download.php?t=site&e=59. Acesso em: 20 ago. 2020.