



Esse Material é Parte do Curso:

DATA SCIENCE COM PYTHON AVANÇADO

do: Prof. M.Sc. Howard Roatti

contato: howardcruzroatti@gmail.com

Figuras

Figura 1: Processo Computacional. Fonte: O Autor	11
Figura 2: Visão do Jupyter Notebook (Célula com Prompt In []:). Fonte: O Autor	13
Figura 3: Visualizando a Assinatura do Método Print. Fonte: O Autor	13
Figura 4: Anatomia do Método Print. Fonte: O Autor	14
Figura 5: Execução do Método Print. Fonte: O Autor	14
Figura 6: Declaração de Variáveis. Fonte: O Autor	15
Figura 7: Exemplos de Nomes de Variáveis Corretas e Incorretas. Fonte: https://aprendendoprogramacao.com.br/2018/08/19/laboratorio-de-algoritmos-aula-03a/	16
Figura 8: Entrada de Dados com Input. Fonte: O Autor	16
Figura 9: Definição de Variáveis e Verificação de Tipos. Fonte: O Autor	17
Figura 10: (a) Preenchendo as Variáveis Utilizando Input (b) Exibindo os Tipos das Variáveis. Fonte: O Autor	18
Figura 11: cast Aplicado aos Métodos input e Verificação de Tipagem. Fonte: O Autor	18
Figura 12 - Estruturas Condicionais. Fonte: https://medium.com/@tiwarigaurav2512/python-if-if-else-elif-nested-if-statements-d443cf0bb2e1	29
Figura 13: Matriz Bidimensional – Indexação de Células. Fonte: O Autor	61
Figura 14: Métodos de Leitura de Dados Externos. Fonte: O Autor	75
Figura 15: Anatomia de um Gráfico. Fonte: https://matplotlib.org/2.0.2/faq/usage_faq.html	86

Tabelas

Tabela 1: Símbolos de Operações Matemáticas. Fonte: O Autor	15
Tabela 2: Operadores Aritméticos. Fonte: O Autor	19
Tabela 3: Operadores de Atribuição. Fonte: O Autor	20
Tabela 4: Operadores de Comparação. Fonte: O Autor	20
Tabela 5: Operadores Lógicos. Fonte: O Autor	21
Tabela 6: Operadores de Membros. Fonte: O Autor	21
Tabela 7: Fatiamento de Lista. Fonte: O Autor	23
Tabela 8: Tabela com os Modos de abertura de arquivo. Fonte: O Autor	38

Sumário

Módulo I – Programação em Python.....	6
VISÃO GERAL.....	7
A Linguagem de Programação Python	8
O Ambiente de Desenvolvimento da Linguagem Python	9
INTRODUÇÃO À PROGRAMAÇÃO COM PYTHON	11
Entrada, Processamento e Saída	12
Operadores	18
Estruturas de Dados	21
Listas	22
Tuplas.....	25
Conjuntos.....	26
Dicionários	26
Estruturas Condicionais e Estruturas de Repetição	28
Estruturas Condicionais	28
Estruturas de Repetição	30
Métodos e Pacotes	33
Métodos.....	33
Pacotes	35
Arquivos.....	37
Web Scraping.....	41
Módulo II – A Biblioteca Numpy.....	46
VISÃO GERAL.....	58
Introdução à Biblioteca Numpy.....	58
Criação de ndarray.....	58
Operações Matemáticas.....	60
Indexação e Fatiamento	60
Funções Universais	63
Métodos Estatísticos	64
Ordenação e Unicidade de Elementos	64
Módulo III – A Biblioteca Pandas.....	67
VISÃO GERAL.....	68
Estrutura de Dados da Biblioteca Pandas.....	68
Reindexação.....	70
Indexação, Seleção e Filtragem	71

Ordenação	71
Estatísticas Descritivas.....	72
Módulo IV – A Biblioteca Pandas – Arquivos e Preparação de Dados	74
Carregamento de Arquivos.....	75
Arquivos CSV.....	75
Arquivo JSON	77
Arquivo XML	78
Arquivo HTML.....	80
Banco de Dados	80
Preparação de Dados.....	81
Remoção de Registros Duplicados	81
Remoção Registros com Dados Ausentes	82
Preenchendo Registros com Dados Ausentes	82
Substituindo Valores.....	83
Conversão de Tipos de Dados	83
Módulo V – Visualização de Dados.....	85
Introdução a Biblioteca Matplotlib	86
Introdução ao Seaborn	91
Referências	93

Módulo I – Programação em Python

Nesse módulo você irá aprender o básico da programação na linguagem Python que lhe dará base para ser um Cientista de Dados, além de te permitir criar programas simples e atuar com Inteligência Artificial.

VISÃO GERAL

É impossível tratar de Ciência de Dados, sem comentar sobre os dados. Afinal, o que é um dado? Um dado trata de um fato, uma observação documentada, um resultado de uma medição ou, de maneira mais abrangente, uma característica do mundo em sua forma mais bruta, por exemplo: uma cor, a matrícula de um funcionário, o valor de um imóvel, o número do CPF, o nome de um país, a resposta de uma pergunta com sim ou não. Um dos maiores desafios da Ciência de Dados é transformar essa observação bruta da vida real em uma oportunidade de informação.

A humanidade, desde o seu surgimento, tem a necessidade de armazenar e manipular os dados por meio da comunicação, para que ela prospere e alcance o maior número de indivíduos possível. Os homens das cavernas armazenavam as mensagens nas paredes, os egípcios nos hieróglifos, os romanos nos pergaminhos, os livros surgiram dos manuscritos que perduraram por milênios mantendo os dados armazenados e sendo disseminado pela humanidade, até que o homem inventou o computador, o meio de comunicação que mais revolucionou a sociedade atual, onde milhões de milhões de dados são armazenados e processados diuturnamente.

Embora atualmente tenhamos diversas origens para os dados, a sua organização é importante para que o cientista de dados possa realizar suas análises e preparação para que sua utilização seja eficaz. Dessa forma é necessário que os dados passem por um processamento para que fiquem como tabelas, com formalidade para cada tipo de dados que será apresentado ao domínio das características das entidades presentes nos negócios.

A Ciência de Dados é uma área multidisciplinar que une disciplinas como Ciência da Computação, Estatística, Matemática e *Business*. A Ciência da Computação é responsável pelas linguagens de programação, armazenamento e processamento de dados juntos dos algoritmos desenvolvidos com o auxílio da matemática. A estatística dá uma visão resumida dos dados, além de apoiar os algoritmos nas inferências e descobertas da informação, essas advindas dos negócios que são responsáveis pela produção dos dados através da sua área de atuação.

Muitas das evoluções tecnológicas foram pensadas com o objetivo de diminuir o esforço humano e mantê-lo seguro em atividades de alto risco. Uma dessas evoluções tecnológicas teve o seu ápice no início do século XX com nomes como: McCulloch e Pitts, Shannon e Alan Turing, foi a Inteligência Artificial que surgiu como um braço da matemática e logo foi absorvida pela Ciência da Computação.

Hoje em dia a Ciência da Computação tem sob ela a Ciência de Dados, se apoia na Matemática e na Estatística com estudos de algoritmos para o desenvolvimento de soluções e softwares, a Eletrônica e a Física para o desenvolvimento de Hardware. E segue com diversas evoluções.

Para iniciar os primeiros passos na Ciência de Dados, um dos marcos inicial é aprender sobre programação de computadores, onde terá habilidades lógicas e matemáticas para o desenvolvimento de algoritmos dos mais simples até as Inteligências Artificiais e Robótica. Em seguida deverá aprender como processar matematicamente os dados, aplicando operações, criando estruturas que facilitem o seu manuseio e armazenamento, extraíndo de diversas origens e visualizando esses dados em forma de gráficos e relatórios.

Com essas habilidades será possível adentrar ao mundo da Ciência de Dados e então iniciar os primeiros passos com Inteligência Artificial, através do aprendizado de máquina criando modelos que consigam prever e prescrever ações para o mundo real, identificando padrões e auxiliando nos *insights* para tomadas de decisão.

A Linguagem de Programação Python

Python é uma linguagem de programação de alto nível, ou seja, facilmente entendível pelas pessoas. É uma linguagem interpretada, isso significa que o processo de transformação da linguagem de alto nível em linguagem de máquina é feita por um mecanismo que analisa e converte linha a linha do seu Script, validando e verificando a existência de alguma falha, informando instantaneamente para o programador. Imperativa, orientada à objetos e funcional, Python traz o melhor dos mundos para uma linguagem de simples aprendizado com tipagem dinâmica e forte.

Concebida em 1989 por Guido van Rossum baseada na linguagem ABC com parte de sua sintaxe derivada do C e Haskell, foi lançada em 1991 com a versão 0.9.0. Em 1994 foi criado o primeiro fórum e mais importante sobre discussão da linguagem, mantendo a comunidade em torno da linguagem unida desde então fazendo evoluções e melhorias. Sendo uma linguagem de código aberto, o Python tem na comunidade o maior pilar da sua evolução e se tornou espelho para o surgimento de outras linguagens multiparadigmas. Podemos Python sendo aplicada em diversos ambientes distintos, desde operações matemáticas simples, passando por computação gráfica, servidores de aplicação, linguagem de script para o pacote de escritórios Libre Office, chegando até *Machine Learning*, robótica e Computação Quântica.

A linguagem possui duas distintas versões que é de suma importância conhecer. A versão 2.x, que apresentou as maiores evoluções desde o seu surgimento se adaptando melhor ao paradigma orientado a objetos e funcional. E a versão 3.x, removendo alguns excessos que foram feitos na versão anterior e então fazendo com houvesse perda de compatibilidade entre as versões. Dessa forma é necessário no planejamento de um novo programa escrito nessa linguagem ter em mente que instruções criadas na versão 2.x podem não ser compatíveis completamente pela versão 3.x, fazendo com que os analistas de sistemas decidam por uma ou outra.

A pesar de todo esse contexto, a linguagem apenas passou a ficar famosa após a criação das bibliotecas de funcionalidades criadas que permitiu a linguagem ser utilizada para o desenvolvimento Web, desenvolvimento de software em geral e aprendizado de máquina. Antes disso, era uma linguagem mais conhecida pelos usuários da plataforma Linux para processamento de textos. Devido a essa característica, foi facilmente trazida para a computação científica. E pela sua facilidade foi levada para criação de ambientes gráficos e páginas Web.

O Ambiente de Desenvolvimento da Linguagem Python

A linguagem possui um console próprio para execução de comandos e scripts diretamente no Terminal Linux/macOS ou Prompt de Comandos/Power Shell do Windows. É um interpretador compatível com ambientes como Visual Studio Code, PyCharm, Eclipse, PyDev,

Atom, Win IDE, IntelliJ IDEA, NetBeans e diversos outros. Cada um desses ambientes possui suas características e facilidades que permitem o desenvolvedor ter melhor desempenho na criação de seus códigos.

Além de todas essas opções, existe uma biblioteca chamada IPython que permite um console um pouco mais avançado que o original com alguns comandos mágicos que facilitam a vida do cientista de dados. Dessa biblioteca surgiu a tão famosa IDE Jupyter Notebook que alia as melhorias do IPython com um ambiente Web muito atrativo para criação e documentação de scripts Python para os mais distintos objetivos, porém mais utilizado pelos cientistas para desenvolvimento no formato de relatórios, com uma apresentação digna de um trabalho de faculdade, além de possibilidades de geração de outros formatos de documentos apresentado todas as análises feitas em forma de história.

Serão utilizados principalmente dois ambientes de código aberto: Visual Studio Code (VS Code para os íntimos) e Jupyter Notebook. Para tal feito sugere-se a instalação de um pacote de aplicações mais completo conhecido como Anaconda, onde em seu navegador há a facilidade de instalação e configuração dos ambientes citados. Para instalação basta acessar o link <https://www.anaconda.com/products/individual#download-section> e na seção Anaconda Installers realizar o download da versão compatível com sua plataforma. A versão atual conta com o interpretador Python 3.8. Basta seguir os passos de instalação que o ambiente estará pronto para sua utilização. No decorrer do material outras dicas e sugestões de utilização irão surgir, paralelamente ensinando a utilizá-lo junto ao aprendizado de todo o conteúdo.

INTRODUÇÃO À PROGRAMAÇÃO COM PYTHON

Computadores foram criados para manipulação de dados, essa manipulação permite a extração de informações. O processo de computação é dividido em três partes: entrada (onde o usuário a partir de periféricos inserem os dados), o processamento (onde o computador utilizando de algoritmos trata e transforma os dados) e saída (onde os dados processados se tornam informações dotadas de relevância com a capacidade humana em análise), Figura 1.



Figura 1: Processo Computacional. Fonte: O Autor

Para que o computador realize o processamento é necessário que alguém implemente um algoritmo. Esses algoritmos são roteiros escritos em uma gramática de alto nível (entendível por humanos) e transformada em linguagem de baixo nível (entendível pelos computadores, sabendo que o computador atualmente apenas compreende uma linguagem com 0s e 1s).

Para exemplificar os algoritmos, pense na vida real, existem diversas rotinas que às vezes são automatizadas pelas pessoas que nem percebem que estão executando um conjunto instruções finitas com objetivo final, por exemplo: atravessar a rua, a rotina da manhã ao acordar, acessar as redes sociais, seguir uma receita de uma comida que gosta e se arrumar para sair, entre outras. Todas essas tarefas detalhadas indicam passos que são feitos de maneira simples e objetiva transmitem passo a passo para o alcance de um objetivo.

Para o computador é necessário que as instruções sejam descritas de maneira que ele as compreenda, por isso foram criadas as linguagens de programação onde as pessoas podem, sabendo como se expressar no idioma selecionado, instruir o computador a realizar ações. As linguagens de programação possuem dois processos distintos para tradução das

instruções escritas pelo humano para a linguagem compreendida pelos computadores. Interpretação e compilação.

A compilação é um processo complexo que analisa todo o algoritmo verificando se há algum elemento que não foi declarado corretamente ou não faz parte da linguagem, emitindo um relatório de falhas, caso haja. Não havendo falhas o compilador transforma esse algoritmo em um programa passível de execução já em uma linguagem que as máquinas compreendem.

A interpretação é um processo mais simples que analisa o algoritmo linha a linha analisando possíveis falhas ou falta de declarações, porém, cada vez que encontra algo que possa estar incorreto o processo para até que seja corrigido. Por isso esse processo é um pouco mais lento do que a compilação, entretanto tem sido o processo mais utilizado pelas linguagens de programação atuais. E esse é o processo utilizado pela linguagem Python.

Então, se seu objetivo é criar programas que tenham uma execução mais rápida, não precisa deixar de usar o Python. Algumas bibliotecas foram criadas baseadas em linguagens compiladas e quando incorporadas fazem com que o desempenho do Python seja superior às linguagens compiladas. Além disso, é possível acelerar a execução de algoritmos em Python utilizando técnicas como *Multithreading* (assunto não abordado aqui).

As bibliotecas são conjuntos de algoritmos disponibilizados como funcionalidades implementadas por terceiros que permitem incorporação em outros scripts, para que não seja necessário recriar funções existentes. Isso facilita a vida dos cientistas de dados e desenvolvedores de software.

Entrada, Processamento e Saída

Após a execução do Jupyter Notebook e selecionar a criação de um Notebook do Python, tudo que for digitado no prompt (In []:) será salvo no arquivo que foi gerado. Para executar as instruções incluídas em cada uma das células basta pressionar Shift+Enter que também insere uma célula nova, caso não seja interesse executar e criar uma nova célula basta pressionar Ctrl+Enter (Figura 2).

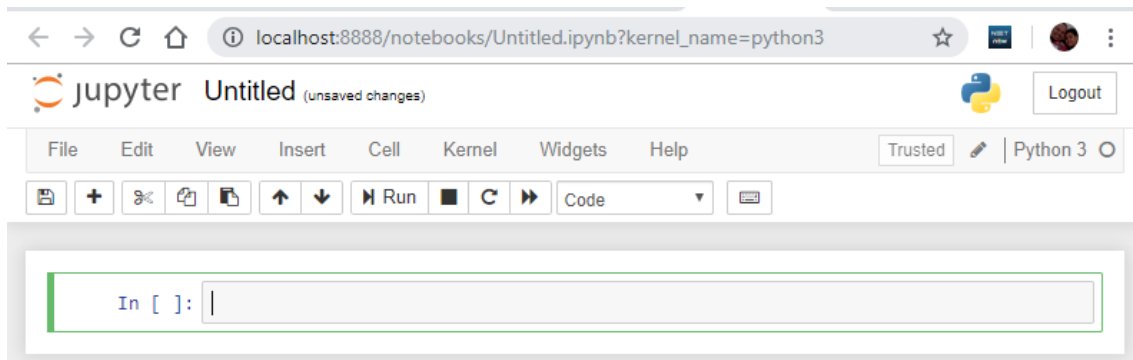


Figura 2: Visão do Jupyter Notebook (Célula com Prompt In []:). Fonte: O Autor

Inicie com o “Hello, World!”. Para isso insira o método utilizado para exibição de um conteúdo, o `print`. Os métodos além do nome possuem os atributos parametrizáveis necessários para sua execução, a isso se denomina assinatura do método. É possível verificar a assinatura do método e suas funcionalidades através da utilização do símbolo de interrogação (?) ou dupla interrogações (??) após o método que se deseja analisar (Figura 3).

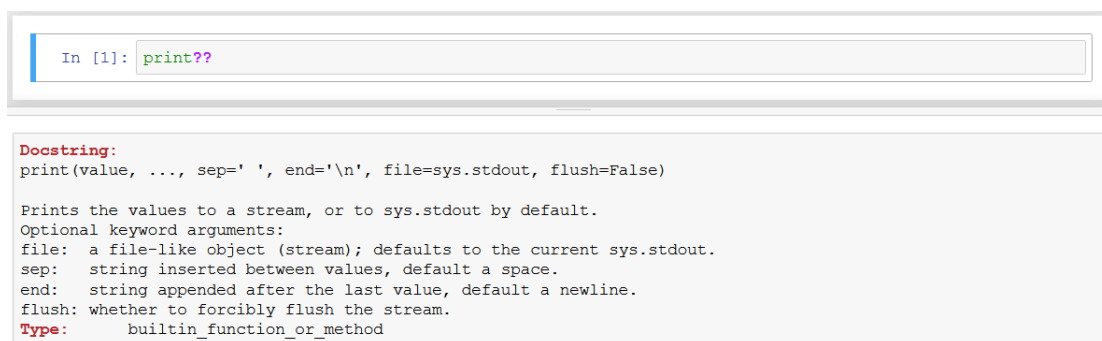


Figura 3: Visualizando a Assinatura do Método Print. Fonte: O Autor

Como é possível ver, a maior parte dos parâmetros do método `print` não são obrigatórios, pois já possuem valor padrão pré-definidos na assinatura. Assim, o único parâmetro obrigatório é o `value`, aquele que será utilizado para impressão/exibição do que se deseja. Esse método será o mais utilizado, pois ele permite, além de exibição de textos delimitados por aspas simples ou dupla (Figura 4), a exibição dos valores armazenados nas variáveis (assunto tratado mais a frente). No método ainda podemos definir o caractere que irá separar os conteúdos exibidos através do parâmetro `sep`, podemos definir o caractere de finalização de linha com `end`, entre outras coisas.

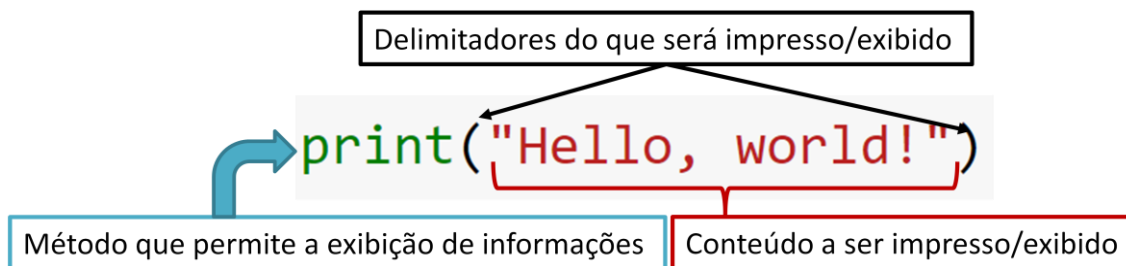


Figura 4: Anatomia do Método Print. Fonte: O Autor

Você poderá executar o método da Figura 5 e verificar a sua saída logo abaixo sendo executada pressionando os atalhos ditos anteriormente, o Ctrl+Enter ou Ctrl+Shift.

```
In [1]: print("Hello, World!")
Hello, World!
```

Figura 5: Execução do Método Print. Fonte: O Autor

Para executar operações matemáticas, primeiro devemos conhecer os símbolos que representam as operações na linguagem Python (outras operações são possíveis através da utilização de bibliotecas):

Operadores	Operação	Exemplo
+	Soma	<pre>In []: 2+2</pre>
-	Subtração	<pre>In []: 3-1</pre>
*	Multiplicação	<pre>In []: 4*2</pre>
/	Divisão	<pre>In []: 8/2</pre>
**	Potência	<pre>In []: 3**2</pre>

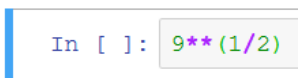
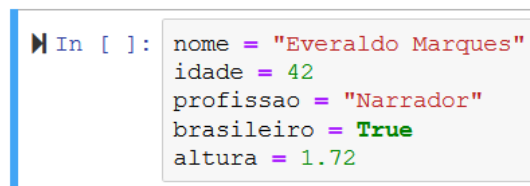
**(1/2)	Raiz Quadrada (Existe outra forma, porém necessita da biblioteca Math ou similar)	
---------	--	---

Tabela 1: Símbolos de Operações Matemáticas. Fonte: O Autor

Perceba que para executar operações como as descritas na Tabela 1, basta digitar os valores, a operação e executar. Nesse caso não há a obrigação de utilizar o método `print`. Haverá casos que os valores estarão armazenados e então o método `print` será o responsável por exibir esses valores, ou caso você possua muitas instruções em uma mesma célula, o método será útil.

Um conceito muito importante na programação é o da variável, assim como na matemática com seus Xs, Ys e Zs. A programação utiliza o conceito de variáveis para armazenar um valor em memória e permitir que futuramente esse valor seja utilizado apenas informando o nome da variável a qual o valor foi armazenado.



```
In [ ]: nome = "Everaldo Marques"
        idade = 42
        profissao = "Narrador"
        brasileiro = True
        altura = 1.72
```

Figura 6: Declaração de Variáveis. Fonte: O Autor

As variáveis seguem um padrão de definição onde um nome será indicado a ela que seja relevante e que permita facilmente a identificação de seu objetivo. Os nomes das variáveis devem obrigatoriamente iniciar com letras, mas podem ter números e alguns caracteres especiais são permitidos. Geralmente utiliza-se apenas o caractere especial *underline* (`_`). Acentuação gráfica é permitida, mas é fortemente desaconselhada. Alguns exemplos de nomes de variáveis podem ser vistos na Figura 6 onde se observa que para a linguagem Python o símbolo de igual (`=`) é utilizado para atribuição de um valor à variável, conseqüentemente sua criação e definição de tipo. Vejam alguns exemplos de nomes de variáveis possíveis e não permitidas na Figura 7.

idade	😊	_12345	😊
dt_nasc	😊	_1_	😊
data_nasc	❌	local?	❌
nota2	😊	sua/altura	❌
nota_2_	😊	nota-1	❌
3peso	❌	_____	😊
n1,aluno	❌	nome1	😊

Figura 7: Exemplos de Nomes de Variáveis Corretas e Incorretas. Fonte: <https://aprendendoprogramacao.com.br/2018/08/19/laboratorio-de-algoritmos-aula-03a/>

Para o processo de entrada, processamento e saída ficar completo falta o processo de entrada que possui um método próprio para interação com usuário e algumas outras formas avançadas de recuperação de dados que serão apresentadas mais para o final desse conteúdo. Para interagir com o usuário utiliza-se o método `input`, esse método tem como parâmetro o texto que será exibido para o usuário informando o que ele precisa digitar para ser atribuída à variável. Veja os exemplos na Figura 8 a seguir:

```
In [ ]: nome = input("Informe o seu nome: ")
        idade = input("Informe a sua idade: ")
        profissao = input("Qual a sua profissão? ")
        brasileiro = input("Você é brasileiro? ")
        altura = input("Qual a sua altura? ")
```

Figura 8: Entrada de Dados com Input. Fonte: O Autor

É possível verificar que o resultado obtido pelo método de entrada é atribuído diretamente a variável. Entretanto, é preciso ter em mente os tipos de variáveis que existem em Python, para que se faça a conversão do resultado do `input` para o formato correto de cada variável, devido o `input` apenas recuperar texto (`str`). Os tipos de dados permitem as linguagens tratarem de forma eficiente cada domínio armazenado, avaliando corretamente as operações possíveis e otimizando a execução dos algoritmos. O domínio de uma variável define os valores possíveis que podem ser armazenados nas variáveis.

Em Python, todos os tipos de dados são objetos (assunto não tratado aqui, mas relevante para quem pretende adotar a linguagem Python como principal para desenvolvimento de softwares), portanto possuem métodos pré-definidos que podem ser utilizados, além de novos métodos que podem ser criados. Esses objetos possuem tipos que representam o mundo real na menor unidade possível como para nulo (*Null*) o `None`, para texto (*strings*) o

str. Para números racionais, aqueles com casas decimais, o float. Números inteiros são representados por int. Já valores como verdadeiro (True) ou falso (False) o bool é o tipo atribuído ao domínio da variável.

Ao contrário das linguagens fortemente tipadas, que após sua definição o tipo não pode ser alterado, no Python os tipos são dinamicamente tipados, ou seja, o tipo definido na declaração pode ser alterado. Entretanto, diferente de outras linguagens dinamicamente tipadas, no Python, após a declaração e definição do tipo, apenas operações relacionadas àquele tipo podem ser aplicadas.

Caso, durante o desenvolvimento não se recorde o tipo do dado de uma variável, o Python dispõe de um método chamado type que permite a exibição da tipagem atual. Vejamos o exemplo na Figura 9:

```
In [1]: nome = "Everaldo Marques"
        idade = 42
        profissao = "Narrador"
        brasileiro = True
        altura = 1.72

In [2]: type(nome)
Out[2]: str

In [3]: type(idade)
Out[3]: int

In [4]: type(profissao)
Out[4]: str

In [5]: type(brasileiro)
Out[5]: bool

In [6]: type(altura)
Out[6]: float
```

Figura 9: Definição de Variáveis e Verificação de Tipos. Fonte: O Autor

Na Figura 9 vemos que na primeira célula foram definidas as variáveis: nome, idade, profissao, brasileiro e altura. Nas células seguintes, utilizando o type, foi possível verificar o tipo associado a cada uma das variáveis. No prompt de saída Out []: podemos visualizar os tipos pertinentes a cada uma das variáveis.

Retornando ao método input, experimente executar e preencher os valores das variáveis dinamicamente. Em seguida utilize o método type para verificar a tipagem das variáveis (Figura 10). Você irá perceber que todas elas estarão definidas como str (texto). O que fazer?

```

In [7]: nome = input("Informe o seu nome: ")
        idade = input("Informe a sua idade: ")
        profissao = input("Qual a sua profissão? ")
        brasileiro = input("Você é brasileiro? ")
        altura = input("Qual a sua altura? ")

Informe o seu nome: Everaldo Marques
Informe a sua idade: 42
Qual a sua profissão? Narrador
Você é brasileiro? True
Qual a sua altura? 1.72

In [8]: type(nome)
Out[8]: str

In [9]: type(idade)
Out[9]: str

In [10]: type(profissao)
Out[10]: str

In [11]: type(brasileiro)
Out[11]: str

In [12]: type(altura)
Out[12]: str

```

Figura 10: (a) Preenchendo as Variáveis Utilizando Input (b) Exibindo os Tipos das Variáveis. Fonte: O Autor

Existe um conceito muito importante na programação que define a possibilidade de alteração do tipo de uma variável para outra, o nome desse conceito é cast. O cast permite que se faça esse tipo de mudança e para isso é importante ter em mente os nomes dos tipos de variáveis que o Python possui, pois através deles utilizando-os como métodos a mudança será possível.

```

In [13]: nome = str(input("Informe o seu nome: "))
        idade = int(input("Informe a sua idade: "))
        profissao = str(input("Qual a sua profissão? "))
        brasileiro = bool(input("Você é brasileiro? "))
        altura = float(input("Qual a sua altura? "))

Informe o seu nome: Everaldo Marques
Informe a sua idade: 42
Qual a sua profissão? Narrador
Você é brasileiro? True
Qual a sua altura? 1.72

In [14]: type(nome)
Out[14]: str

In [15]: type(idade)
Out[15]: int

In [16]: type(profissao)
Out[16]: str

In [17]: type(brasileiro)
Out[17]: bool

In [18]: type(altura)
Out[18]: float

```

Figura 11: cast Aplicado aos Métodos input e Verificação de Tipagem. Fonte: O Autor

Na Figura 11 é possível verificar a aplicação do cast para cada um dos tipos existentes na linguagem, permitindo a alteração do tipo str que originalmente é definido pelo método input para os tipos pertinentes ao dado que deverá ser utilizado: str, int, bool e float. Em seguida tem-se a verificação de tipagem de cada uma das variáveis criadas.

Operadores

Além dos operadores matemáticos vistos anteriormente, existem outros operadores que podem ser aplicados e que são importantes no desenvolvimento de algoritmos e aplicações, são eles: aritméticos, de atribuição, de comparação, lógicos e membros.

Os operadores aritméticos são utilizados em números para a realização de operações matemáticas, conforme Tabela 2 e exemplos a seguir:

Operador	Descrição	Exemplo
+	Calcula a soma de duas parcelas	7+42
-	Calcula a diferença do subtraendo do minuendo	36-19
*	Calcula o produto de dois fatores	3*4
/	Calcula o quociente do dividendo pelo divisor	30/2
%	Calcula o resto do quociente do dividendo pelo divisor, também conhecido como módulo.	31%2
//	Calcula o quociente do dividendo pelo divisor tendo como resultado a parte inteira do quociente	31//2
**	Calcula a potência da base pelo expoente	2**16

Tabela 2: Operadores Aritméticos. Fonte: O Autor

Exemplos em Python:

```
x = 13
y = 3
z = 3.0
```

```
print("x + y", (x + y))
print("x - y", (x - y))
print("x * y", (x * y))
print("x / y", (x / y))
print("x / z", (x / z))
print("x // y", (x // y))
print("x ** y", (x ** y))
```

Operadores de atribuição são utilizados para atribuir valores as variáveis, alguns podem realizar a atribuição e aplicar uma operação matemática ao mesmo tempo. Esse tipo de operador visto na Tabela 3 é utilizado para facilitar a escrita de operações matemáticas, permitindo aplicar a operação ao valor previamente atribuído a variável como o novo valor à direita aplicando a operação.

Operador	Exemplo	Forma Comum
=	a = 7	a = 7
+=	a += 7	a = a + 7
-=	a -= 7	a = a - 7
*=	a *= 7	a = a * 7
/=	a /= 7	a = a / 7

%=	a %= 7	a = a % 7
//=	a //= 7	a = a // 7
**=	a **= 7	a = a ** 7

Tabela 3: Operadores de Atribuição. Fonte: O Autor

Vejamos alguns exemplos:

```
x = 1
print(x)
x += 2
print(x)
x **= 2
print(x)
x /= 3
print(x)
x //= 2
print(x)
x -= 1
print(x)
```

Operadores de comparação são utilizados para comparar valores e verificar se a condição avaliada é True (verdadeira) ou False (falsa) (Tabela 4).

Operador	Descrição	Exemplo
>	Retorna True caso o operando da esquerda seja maior que o da direita	a > b
<	Retorna True caso o operando da esquerda seja menor que o da direita	a < b
==	Retorna True se ambos possuem o mesmo valor	a == b
!=	Retorna True se os operados possuem valores distintos	a != b
>=	Retorna True caso o operado da esquerda seja maior ou igual ao da direita	a >= b
<=	Retorna True caso o operado da esquerda seja menor ou igual ao da direita	a <= b

Tabela 4: Operadores de Comparação. Fonte: O Autor

Esses operadores serão muito úteis quando estivermos tratando de Estruturas Condicionais, junto aos operadores lógicos permitirão algoritmos mais sofisticados. Vejamos alguns exemplos mais simples a seguir:

```
x = 73
y = 42
print(x > y)
print(x < y)
print(x == y)
print(x != y)
print(x >= y)
print(x <= y)
```

Operadores lógicos permitem combinar operadores de comparação para que situações que precisam ocorrer ao mesmo tempo sejam validadas, obtendo como resultado True ou False (Tabela 5).

Operador	Descrição	Exemplo
and	Retorna True caso o resultado de ambas as operações sejam True, para qualquer outro caso o resultado será False	<code>a > b and a == c</code>
or	Retorna False caso o resultado de ambas as operações sejam False, para qualquer outro caso o resultado será True	<code>a > b or a == c</code>
not	Retorna o resultado revertido de uma operação de comparação	<code>not (a > b)</code>

Tabela 5: Operadores Lógicos. Fonte: O Autor

Esses operadores são úteis quando existem duas situações a serem avaliadas que precisam ocorrer simultaneamente, como por exemplo, para que uma pessoa possa possuir a carteira de motorista ela precisa ter maior idade (`idade >= 18`) e ter sido aprovada nas avaliações do DETRAN (`passouNasAvaliacoes == True`). Vejamos alguns exemplos em Python:

```
idadeCNH = 18
passouNasAvaliacoes = True
print(idadeCNH == 18 and passouNasAvaliacoes == True)
idadeBrincar = 12
altura = 1.45
print(idadeBrincar > 12 or altura >= 1.4)
print(not(idadeBrincar != idadeCNH))
```

Operadores de membros permitem verificar a ocorrência de um valor dentro de uma sequência, `str` ou lista de valores (Tabela 6).

Operador	Descrição	Exemplo
in	Retorna True se o valor ocorre ao menos uma vez em uma lista, sequência ou <code>str</code> . Caso contrário retorna False.	<code>'a' in 'banana'</code>
not in	Retorna True se o valor não ocorre ao menos uma vez em uma lista, sequência ou <code>str</code> . Caso contrário retorna False.	<code>1 not in [1, 2, 3]</code>

Tabela 6: Operadores de Membros. Fonte: O Autor

Os operadores de membros são úteis para verificar a existência de um elemento dentro de uma sequência de textos ou numérica, representada muitas das vezes com uma Estrutura de Dados. Vejamos alguns exemplos:

```
a = [1,2,3]
b = "banana"
print(1 in a)
print(1 not in a)
print("a" in b)
print("a" not in b)
```

Estruturas de Dados

As variáveis são boas maneiras de se armazenar um conteúdo e poder utilizá-lo futuramente, entretanto, haverá vezes em que será necessária a criação de muitas deles com mesmo significado para armazenar diversos valores distintos com mesmo significado,

por exemplo: notas de uma disciplina, membros de uma equipe, times de futebol, salários de funcionários, entre outros.

Para que se possa colecionar um conjunto de valores com mesmo significado, as linguagens de programação possuem um tipo de estrutura especial que permite o armazenamento e manuseio desses dados de modo a facilitar e melhorar o desempenho na construção de algoritmos específicos para essas estruturas. O Python possui nativamente quatro tipos de estruturas, são elas: listas, tuplas, sets (conjuntos) e dicionários. Ainda é possível criar outros tipos de estruturas, mas isso é tratado em disciplinas avançadas de Ciência da Computação. Manteremos o foco principalmente nas listas e dicionários, bases para estruturas futuras que serão tratadas em momentos oportunos.

As listas são um tipo de coleção ordenada que permita alteração de seus membros, movimentação e principalmente, permite a duplicação deles. Geralmente criam-se listas de elementos do mesmo tipo, mas essa estrutura tão versátil do Python permite o armazenamento de tipos diferentes. As tuplas também são um tipo de coleção em que a ordem prevalece e os membros podem ser duplicados, porém não é possível alterá-los após sua definição. Os sets são estruturas criadas para que a álgebra relacional e a teoria dos conjuntos pudessem ser utilizadas de maneira mais simples no Python, não é uma coleção ordenada e também não indexada, mesmo assim não permite que haja membros duplicados. Por fim, os dicionários são tipos de estruturas polivalentes, não possuem ordenação, é indexada e permite alterações.

Listas

As listas são representadas por elementos separados por vírgula envolta de um par de colchetes. A definição irá elucidar a definição:

```
lista = ['D', 'a', 't', 'a', ' ', 'S', 'c', 'i', 'e', 'n', 'c', 'e']
```

É possível exibir uma lista utilizando o método `print()`:

```
print(lista)
```

Uma informação importante é o tamanho que pode ser verificado através do método `len()`:

```
len(lista)
```

Por ser uma estrutura indexada, o acesso aos elementos é feita de forma simples e intuitiva através da utilização do número que indica o índice de cada elemento apresentado entre um par de colchetes. É preciso salientar que os índices no Python iniciam em 0 (zero), portanto o último índice de uma lista será o tamanho da lista menos 1. Veja a seguir:

```
lista = ['D', 'a', 't', 'a', ' ', 'S', 'c', 'i', 'e', 'n', 'c', 'e']
print(lista[0])
print(lista[1])
print(lista[2])
print(lista[3])
print(lista[4])
print(lista[5])
print(lista[6])
print(lista[7])
print(lista[8])
print(lista[9])
print(lista[10])
print(lista[11])
```

Caso tente executar o `print(lista[12])` irá receber do Python uma mensagem de erro informando que você está tentando acessar uma área fora da variação de índices possíveis da lista, pois a lista criada anteriormente possui 12 (doze) elementos e, portanto índices de 0 (zero) a 11 (onze), conforme mencionado.

Além de acessar diretamente cada um dos índices, é possível fatiar a lista original em trechos definidos por conjuntos específicos de índices contíguos. Para isso há uma notação que é adotada para realizar o fatiamento, a notação envolve a identificação do índice inicial, separado por dois pontos (:) do índice final. Ainda é possível informar um intervalo a ser utilizado entre o índice inicial e o final incluindo mais um símbolo de dois pontos e valor do passo (Tabela 7).

Notação	Descrição	Exemplo
<code>lista[inicio:fim]</code>	Retorna parte da lista original a partir do índice inicio até o índice fim -1	<code>lista[2:8]</code>
<code>lista[inicio:]</code>	Retorna parte da lista original a partir do índice inicio até o fim da lista	<code>lista[2:]</code>
<code>lista[:fim]</code>	Retorna parte da lista original desde seu primeiro índice até o índice fim-1	<code>lista[:8]</code>
<code>lista[:]</code>	Retorna a lista original	<code>lista[:]</code>
<code>lista[inicio:fim:passo]</code>	Retorna lista original desde o índice inicio até o índice fim-1, criando intervalos com o tamanho de passo	<code>lista[2:8:2]</code> <code>lista[2:8:3]</code>

Tabela 7: Fatiamento de Lista. Fonte: O Autor

Por ser uma estrutura mutável, itens da lista podem ser alterados a partir de simples atribuições como:

```
listaNumerica = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
listaNumerica[0] = 11
print(listaNumerica)
```

Ou atribuindo um pequeno conjunto em outra lista assim:

```
listaNumerica += [10, 11]
print(listaNumerica)
```

Os itens de uma lista podem ser acessados iterativamente utilizando uma estrutura de repetição, como esta:

```
for letra in lista:
    print(letra)
```

```
for numero in listaNumerica:
    print(numero)
```

Além da atribuição com operadores de atribuição, as listas possuem métodos que permitem a inclusão de novos elementos, veja os exemplos:

```
listaNumerica.append(12) #acrescenta o número 12 ao final da lista
print(listaNumerica)
listaNumerica.extend([13, 14]) #acrescenta vários elementos ao final da lista
print(listaNumerica)
listaNumerica.insert(0, 1) #insere o número 1 (um) na posição do índice 0 (zero)
empurrando os elementos para frente e alterando seus índices
print(listaNumerica)
```

Para remover elementos de uma lista existem três formas distintas:

1 – Excluir um elemento ao encontrar na lista:

```
novaLista = [2, 5, 4, 6, 4, 8, 10, 0]
novaLista.remove(2)
print(novaLista)
novaLista.remove(4)
print(novaLista)
```

2 – Remover elementos do final da lista:

```
novaLista.pop()
print(novaLista)
```

3 – Remover um elemento específico no índice indicado:

```
del(novaLista[3])
print(novaLista)
```

Caso sua lista esteja desordenada e deseje colocar em uma ordem específica (crescente ou decrescente), os métodos `sort()` e `sorted()` podem ser utilizados. Veja:


```

listaDesordenada1 = list((17, 14, 2, 7, 5, 11, 8, 3))
listaDesordenada2 = list((-42, 73, -18, -33, 55, 18, 4, -2))
listaDesordenada1.sort() #Ordenada e altera os registros da lista
print(listaDesordenada1)
sorted(listaDesordenada2) #Ordena e não altera os registros da lista
listaOrdenadaReversa = sorted(listaDesordenada2, reverse=True)
print(listaOrdenadaReversa)

```

Lista são estruturas poderosas que são utilizadas em diversos contextos, existem várias possibilidades de utilização e aplicação. Além de outros métodos que podem ser encontrados em diversos fóruns, blogs e tutoriais pela Internet, além, é claro, da documentação oficial <https://docs.python.org/pt-br/3.7/>.

Tuplas

Tuplas são sequências imutáveis, ou seja, uma vez criadas não podem ser modificadas. Elas são definidas com seus elementos envoltos de pares de parênteses. A diferença dessa estrutura para a lista é que seus elementos não podem ser alterados, dessa forma existem bibliotecas que a utilizam como parte de parametrizações que não devem ser modificados, como uma string de conexão com banco de dados.

Para criar uma tupla, basta atribuir a uma variável uma lista de valores separados por vírgula entre uma parte de parentes:

```

tupla1 = ("Iron Man", "Captain America", "Hulk", "Hawkeye", "Black Panther")
print(tupla1)

```

Também é possível criar uma tupla com o método `tuple()`, assim como para lista há o método `list()`:

```

tupla2 = tuple(x for x in range(1, 20, 3)) #O mesmo que tuple((1, 4, 7, 10, 13, 16, 19))
print(tupla2)
lista = list(x for x in range(1, 20, 3))
print(tupla2)
print(lista)

```

O acesso aos itens de uma tupla é realizado da mesma forma que a lista, utilizando o índice dos elementos entre colchetes (isso, não é entre parênteses!). Veja:

```

print(tupla1[0])
print(tupla1[2])
print(tupla1[-1])
print(tupla1[:-2])

```

Também é possível acessar um índice específico de uma tupla utilizando o método `index()`, passando entre os parênteses o índice desejado.

Conjuntos

Sets são coleções ordenadas que possui elementos únicos, são definidos por valores separados por vírgula envoltos de um par de chaves. São muito utilizados para operações matemáticas de conjuntos de dados, como: união, interseção e diferença. Essas operações permitem um melhor trato com os dados de maneira a aperfeiçoar o trabalho do cientista. A criação é simples como as outras estruturas:

```
conjunto1 = {1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 7}
print(conjunto1)
conjunto2 = set([-10, 18, -3, 6, 4, 3])
print(conjunto2)
conjunto3 = set('Data Science')
print(conjunto3)
```

Para adicionar um novo elemento ao conjunto, utiliza-se o método `add()` informando como parâmetro o valor a ser acrescido. Para remover o método é o `discard()`. Experimente:

```
conjunto2.add(8)
conjunto2.add(11)
print(conjunto2)
conjunto2.discard(8)
print(conjunto2)
```

Para utilizar as operações de conjuntos na estrutura, basta invocar os métodos pré-existent: `intersection`, `difference` e `union`.

```
a = {1, 2, 3, 4}
b = {2, 3, 4, 5}
print(a.intersection(b))
print(a.difference(b))
print(a.union(b))
```

Dicionários

Hoje em dia é comum ouvirmos falar de arquivos json, um arquivo hierárquico que auxilia no armazenamento de dados semi-estruturados com capacidade gigantesca de abstração do mundo real, permitindo a criação de programas de alta complexidade. No Python há diversas bibliotecas que implementam o modelo json, além disso, embutido na linguagem existe a estrutura de dados chamada dicionário. Essa estrutura funciona exatamente como um arquivo json, cada registro é identificado por uma chave – um valor único que identifica o registro – e possui um valor após o símbolo de dois pontos (`:`) – o valor pode ser qualquer tipo de objeto Python, desde variáveis primitivas até outros dicionários. Um cuidado que se

deve ter é não confundir o dicionário com o sets, pois ambos são identificados por um par de chaves que envolve os dados colecionados. O que difere é a identificação dos registros.

Vejamos mais claramente como é um dicionário e como acessar suas chaves:

```
car = {  
    "brand": "Ford",  
    "electric": False,  
    "year": 1964,  
    "colors": ["red", "white", "blue"]  
}  
print(car)  
print(car["brand"])  
print(car["electric"])  
print(car.get("brand"))  
print(car.get("colors"))
```

É possível navegar, utilizando um estrutura de repetição, por todas as chaves ou valores acessando o que está armazenado associado:

```
for valor in dicionario.keys():  
    print(car[valor])
```

```
for valor in dicionario.values():  
    print(valor)
```

```
for chave, valor in dicionario.items():  
    print(f'{chave} : {valor}')
```

Acima é possível ver três formas de iterar sobre o dicionário e recuperar o seu valor, na primeira estrutura se itera modificando o dado na variável valor e associando como um índice do dicionário, na segunda estrutura se itera nos valores e então não há a necessidade de indexar o dicionário explicitamente, já na terceira estrutura são recuperados paralelamente a chave e o valor e para sua exibição é necessário a inclusão no método print da letra f que permite a substituição de uma string pelas variáveis do escopo.

Novas chaves podem ser facilmente adicionadas ao dicionário, a remoção de itens também é simples com algumas opções simples e intuitivas baseado no que já se viu:

```
dicionario['preco'] = 17000.0 #adiciona a chave preco com valor associado de 17k  
print(dicionario)  
dicionario.pop('preco') #retorna e remove a chave-valor associada  
dicionario.popitem() #remove o último par chave-valores
```

Caso necessite realizar uma cópia do dicionário para algumas alterações, não crie um outro dicionário apenas associando a nova variável a variável do dicionário antigo, assim:

```
novoDicionario = dicionario
```

Isso fará com que ambas apontem para o mesmo objeto em memória e, portanto uma alteração realizada em novoDicionario irá afetar o dicionário original. Para fazer corretamente uma cópia do dicionário e não ocorrer esse tipo de situação, você deverá executar a seguinte instrução:

```
novoDicionario = dicionario.copy()
```

É possível gerar um dicionário a partir de duas listas, onde uma delas é o conjunto de chaves e a outra o conjunto de valores. Para isso devemos utilizar o método zip que recebe como parâmetro as duas listas e permite que navegue por ambas de acordo com o índice retornando uma tupla, perceba que esse resultado é temporário:

```
lista_chaves = ["Nome", "Email", "Telefone"]
lista_valores = ["José Carlos", "jose.carlos@email.com", "+5527999999999"]
zipped_list = zip(lista_chaves, lista_valores)
zipped_list
dicionario_cadastro = dict(zipped_list)
list(zipped_list)
print(dicionario_cadastro)
```

Estruturas Condicionais e Estruturas de Repetição

A partir de agora serão abordadas as estruturas que permitem a aplicação lógica de situações reais no desenvolvimento de algoritmos computacionais, as condicionais auxiliar na determinação de ações a serem adotadas observando o resultado obtido por um teste lógico que resulta em True ou False. As estruturas de repetição permitem iterar sobre valores de coleções ou repetir ações por vezes determinado por um número ou determinando sua pausa por uma condição alcançada.

Estruturas Condicionais

Sempre que o objetivo for executar uma instrução determinada pela satisfação de uma condição, as estruturas condicionais entram em ação e palavras reservadas são utilizadas para representar, são elas: if, elif, else. Esses são os termos utilizados para realizar uma comparação lógica condicional. Vejam algumas ilustrações:

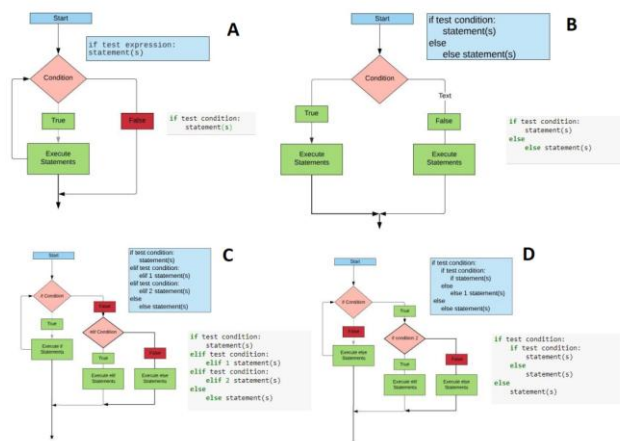


Figura 12 - Estruturas Condicionais. Fonte: <https://medium.com/@tiwarigaurav2512/python-if-if-else-elif-nested-if-statements-d443cf0bb2e1>

Na Figura 12A temos uma estrutura condicional simples contendo apenas instrução `if` a condição a ser verificada e a declaração a ser executada caso a condição seja verdadeira. Caso a condição seja falsa, o fluxo muda o percurso saltando a declaração executada no caso verdadeiro do teste. Veja um exemplo:

```
if idade >= 18:
    print("Você possui idade para ter CNH!")
print("Essa instrução sempre será executada.")
```

É importante salientar que o Python preza pela organização do código, dessa forma é necessário falar sobre escopo. O escopo determina o limite da declaração das instruções a ser executado dentro de um escopo, ele se inicia a partir do símbolo de dois pontos (:). As declarações a partir da linha de baixo precisam seguir uma organização chamada de indentação que, visualmente, permite ao programador e o interpretador da linguagem saberem quais são as declarações que seguem o escopo e, portanto, quais devem ser executadas a partir do teste executado. Essa indentação no Python é definida pelo número de 4 (quatro) espaços alinhadas a instrução da linha superior, nesse caso a estrutura condicional `if`.

Na Figura 12 B, temos a estrutura condicional composta das instruções `if` e `else`. A primeira validando se a condição é verdadeira executa a declaração de seu escopo e pula a instrução `else`. A segunda, validando que a condição do `if` é falsa, executa a instrução do `else`. Vejamos:

```
if idade >= 18:
    print("Você possui idade para ter CNH!")
else:
    print("Você não possui idade suficiente para ter CNH.")
print("Essa instrução sempre será executada.")
```

Na Figura 12 C, temos a estrutura condicional composta pelas instruções if, elif e else. Elas permitem que mais testes possam ser realizados e declarações específicas sejam executadas atendendo a alguma regra de negócio específica. Vejamos um exemplo:

```
if carro == "Mustang":
    print("Parabéns, você possui um Mustang!")
elif carro == "Ferrari":
    print("Parabéns, você possui uma Ferrari!")
elif carro == "Lamborghini":
    print("Parabéns, você possui uma Lamborghini!")
else:
    print("Desculpe, não consegui identificar seu automóvel!")
print("Essa instrução sempre será executada.")
```

Na Figura 12 D, temos uma estrutura condicional composta de todas as instruções que definem uma condicional e, além disso, possuem estruturas aninhadas, isso significa que novos testes podem ser aplicados como parte das instruções declaradas na condição testada. E não somente novas condicionais podem ser aplicadas, qualquer outro tipo de instrução pode ser incluído, lembrando-se de respeitar a indentação do código para que o interpretador entenda como parte do escopo.

```
if idade >= 18:
    if pagamento == True:
        print("Você possui idade e efetuou o pagamento para ter CNH!")
    else:
        print("Você precisa realizar o pagamento para poder ter uma CNH")
else:
    print("Você não possui idade suficiente para ter CNH.")
print("Essa instrução sempre será executada.")
```

É preciso ter em mente que muitas das vezes será possível reescrever uma estrutura condicional aninhada na forma de uma estrutura condicional com operadores lógicos. Veja:

```
if idade >= 18 and pagamento == True:
    print("Você possui idade para ter CNH!")
else:
    print("Você não possui idade para ter CNH ou não realizou o pagamento.")
print("Essa instrução sempre será executada.")
```

Estruturas de Repetição

A estrutura de repetição é uma estrutura que permite executar diversas vezes a mesma instrução, além de ser muito utilizada para iterar sobre objetos conforme visto em exemplos anteriores com as estruturas de dados. Essa repetição é realizada enquanto uma condição não for satisfeita, as condições mais comuns são o fim de um objeto iterável, um número

mínimo de repetições efetuadas e há condições específicas como alcançar um objetivo específico.

A estrutura de repetição mais utilizada é o for, ela possui uma estrutura simples onde identifica a variável iteradora e o objeto iterável seguido do símbolo de dois pontos (:), vejamos:

```
for variavel in objetoIteravel:  
    <instruções a serem repetidas>
```

```
listaCarros = list(["Mercedes", "BMW", "Toyota", "Ford", "Chevrolet", "Fiat",  
"Ferrari", "Porsche"])  
for carro in listaCarros:  
    print(carro)
```

Para navegar em uma lista e utilizar o índice dela para recuperar um dado de uma outra lista de mesmo tamanho, podemos utilizar o método enumerate para transformar a lista em um conjunto com índice e valor, por exemplo: vamos criar uma lista com marcas de celular e uma outra lista com valores fictícios dos mesmos. Então iremos exibir a lista de marcas e preços das marcas que possuem a letra i em seus nomes. Vejamos:

```
lista_marcas = ['Samsung', 'iPhone', 'Motorola', 'LG', 'Xiaomi', 'Huawei',  
'Nokia', 'Positivo']  
lista_precos = [2500, 6000, 1500, 1900, 1000, 1200, 990, 600]  
for idx, marca in enumerate(lista_marcas):  
    print(idx, marca)
```

Se não fizéssemos isso, precisaríamos criar uma variável de controle e ir atualizando ela a cada passo para podermos recuperar um dado de uma segunda lista, assim:

```
indice = 0  
for marca in lista_marcas:  
    if 'i' in marca:  
        print(marca, lista_precos[indice])  
    indice += 1
```

O objeto iterável por ser qualquer uma das estruturas vistas e, caso queira iterar sobre um número mínimo de repetições, o Python possui o método range que define uma variação numérica. Vejamos:

```
range(inicio, fim[, passo])
```

Dado que o índice inicial sempre será 0 (zero), então o último valor gerado será o fim-1, sempre! Veja alguns exemplos de geração de variações numéricas utilizando o range e a utilização do mesmo em uma estrutura de repetição:

```
range(10) #range com o parâmetro final igual a 10
range(5, 18) #range com parâmetro inicial em 5 e final em 18
range(3, 19, 3) #range com parâmetro inicial=3, final=19 e com passo=3
for i in range(10):
    print(i)
```

Ainda é possível colocar um loop dentro do outro para operações mais complexas, como iterar sobre uma matriz bidimensional, veja:

```
matriz = [[1,2,3],
          [4,5,6],
          [7,8,9]]
for i in range(3):
    for j in range(3):
        print(matriz[i][j])

for i in range(3):
    for j in range(3):
        print('i =', i, 'j = ', j)
```

O Python traz uma novidade que é a utilização da instrução else que indica uma instrução que sempre será executada, mesmo que a condição da estrutura de repetição não seja atendida:

```
for i in range(10):
    print(i)
else:
    print("Fim!")
```

Além do for, existe também a estrutura de repetição while. Trata-se de um estruturação de repetição indeterminada, a pesar de muitas das vezes o desenvolvedor saber o número de iterações que serão executadas. Vejamos alguns exemplos:

```
while <condição>:
    <declarações>

i = 1
while i <= 6:
    print(i**2)

j = 2
while j >= 0:
    print(j//2)
    j -= 1
```


Existem algumas instruções que podem dar um pouco mais de flexibilidade na utilização das estruturas de repetição, são elas: break e continue. O break é utilizado para realizar a quebra da iteração da estrutura para dado alguma condição satisfatória. Já o continue é utilizado para parar o fluxo de iteração e pular para o próximo elemento iterável. Vejamos:

```
while True:
    opcao = input("Deseja sair? S ou N")
    if opcao == "S":
        break

numero = 0
while numero < 100:
    numero += 1
    if numero % 3 != 0:
        print("Não divisível por 3", numero)
        continue
```

Dessa forma, é possível criar estruturas mais robustas que atendam a regras que possam vir a surgir no desenvolvimento de algoritmos. Um cuidado que se deve ter é ao criar os famosos loops infinitos, são estruturas de repetição que não possui uma instrução de parada. Dessa forma para finalizar a execução você deverá finalizar o programa. Exemplo:

```
while True:
    print("Olá!")
```

Métodos e Pacotes

Métodos

Já vimos até aqui alguns métodos, porém não os definimos. Métodos são blocos de subprogramas que permitem a disponibilização de alguma função a ser executada. O método que mais utilizamos até aqui foi o print, responsável por exibir na saída padrão uma informação inserida em seu parâmetro. Ou método visto foi o input, o qual exibe uma informação na saída padrão e aguarda uma entrada de dados do usuário.

Os métodos são criados para permitir que uma mesma função que tenha a necessidade de ser utilizada diversas vezes não precise ser recriada, fazendo com que o bloco de código criado seja reutilizável, permitindo criar algoritmos mais elegantes e fáceis de ler. A definição de uma função utiliza a palavra reservada def seguido do nome do método e, caso seja necessário, são criados argumentos que serão utilizados internamente ao método. Esses

argumentos são os parâmetros que os usuários passam na utilização do método. Vejamos alguns exemplos:

```
def soma(a, b):  
    """Esse método irá realizar  
    a soma do argumento a ao argumento b"""  
    return a + b
```

```
def exibeMensagem():  
    """Esse método irá exibir uma mensagem de boas-vindas"""  
    print("Seja bem-vindo!")
```

```
def uniao(conjuntoA, conjuntoB):  
    """Esse método irá realizar a união  
    entre os conjuntos A e B"""  
    return conjuntoA.union(conjuntoB)
```

Os métodos podem ser tão complexos quanto à imaginação do criador, basta ter em mente que dentro do método você poderá criar um algoritmo inteiro que aplica diversas operações, outras funções, estruturas e etc. O que se pede é que o método não fuja do escopo de sua definição, ou seja, ele terá um objetivo e não deverá realizar mais do que o seu objetivo de criação.

Para utilizar o método basta invocá-lo pelo seu nome, passando os parâmetros para os argumentos quando houver:

```
soma(10, 12)  
exibeMensagem()  
exibeMensagem()  
uniao(conjunto1, conjunto2)
```

É possível ainda criar métodos que possuem argumentos com valores padrão, dessa maneira o usuário não será obrigado a informar um parâmetro, deixando que a execução do método siga a execução padrão:

```
def adiciona(a, valor = 1):  
    return a + valor
```

```
print(adiciona(13, 18))  
print(adiciona(14, 20))
```

Isso permite que os argumentos sejam utilizados, inclusive, informando o nome para que facilite sua utilização:

```
def menorIdade(idade1 = 0, idade2 = 0, idade3 = 0):  
    """Esse método irá verificar qual a menor entre três idades informadas"""  
    if idade1 == idade2 and idade2 == idade3:  
        return 0
```

```

elif idade1 > idade2 and idade2 > idade3:
    return idade3
elif idade2 > idade3 and idade3 > idade1:
    return idade1
else:
    return idade2

```

```

print(menorIdade(idade1 = 10, idade2 = 11, idade3 = 18))

```

Pacotes

Pacotes ou packages são maneiras de se organizar os scripts criados em menores arquivos, permitindo melhor manutenção e utilização/reutilização. Segundo a documentação oficial da linguagem, sempre que acessamos o interpretador utilizado, as definições de métodos e variáveis são perdidas. Isso faz com que você tenha que reescrever seus códigos novamente, causando retrabalho. É possível então criar arquivos contendo tudo que foi feito, os famosos scripts. Eles conterão toda a ideia lógica construída anteriormente, permitindo que você apenas o execute conforme a necessidade.

Imagine que criou um script com diversas funções com objetivos distintos, uma boa maneira de se manter organizado é reunião em scripts distintos (arquivo com extensão .py) métodos que fazem parte de uma mesma regra de negócio, a esses damos o nome de módulos. Assim, você poderá facilmente reutilizar em seus scripts futuro.

Crie um arquivo no bloco de notas com métodos e o salve como fibonacci.py, ele irá conter as seguintes funções:

```

# Fibonacci numbers module
def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result

```

a partir do arquivo criado com essas funções, estando o arquivo no mesmo diretório ou direcionando corretamente o novo script para localizar esse arquivo, você poderá utilizá-lo apenas realizando uma importação do módulo, logo, seu novo script ficará assim:

```
import fibonacci
fibonacci.fib(1000)
print(fibonacci.fib2(100))
```

Dessa maneira você terá métodos reutilizáveis e um novo script mais enxuto, sem a necessidade de reescrever os métodos. É possível melhorar ainda mais esse algoritmo tornando o mais legível, para isso basta indicar para o interpretador que, a partir desse módulo, você deseja importar os módulos:

```
from fibonacci import fib, fib2
fib(1000)
print(fib2(100))
```

Caso queira importar todos os módulos disponíveis e os invocar pelo nome, você poderá utilizar o caractere coringa * (asterisco) no lugar dos nomes dos métodos:

```
from fibonacci import *
fib(1000)
print(fib2(100))
```

Ainda é possível renomear o módulo ou os métodos, o módulo:

```
import fibonacci as fibo
fibo.fib(1000)
```

Os métodos:

```
from fibonacci import fib as fibonacci
fibonacci(1000)
```

Para que você possa executar o script através do prompt de comando ou de um terminal, invocando o python e o arquivo criado, é necessário editar o seu script fibonacci.py incluindo no final o seguinte trecho de código:

```
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

Isso fará com que seu arquivo seja utilizável tanto como script, quanto como um módulo importável. Faz-se necessário esse trecho, pois o interpretador só executa o módulo se ele for um arquivo “principal”. Então você conseguirá executar seu script assim:

```
python fibonacci.py 50
```

Um pacote é uma coleção de módulos em um diretório, o nome do pacote é o nome do diretório onde estão armazenados os módulos criados. Isso permite que você reúna em um único diretório, diversos módulos criados, fazendo com que a utilização desses seja feita

através do nome do pacote seguido de ponto e o nome do módulo a ser utilizado. Dessa forma não precisará se preocupar com nomes de módulos repetidos por outros criadores.

Para criar um pacote, primeiro organize o diretório com um nome sugestivo para o pacote, então crie um script vazio chamado `__init__.py`. Esse arquivo será o responsável por indicar ao interpretador que o diretório trata-se um pacote de módulos Python reutilizáveis. O seu diretório terá uma aparência assim:

```
nomeDoPacote/  
    __init__.py  
    modulo1.py  
    modulo2.py  
    modulo3.py
```

Em seu novo script você irá importar da seguinte forma para utilização:

```
import nomeDoPacote.modulo1
```

ou

```
from nomeDoPacote import modulo1, modulo2
```

ou

```
from nomeDoPacote.modulo1 import metodo1
```

Arquivos

Arquivos são localizações nomeadas no disco que armazenam informações relacionadas. São utilizados para armazenamento permanente de dados em uma memória não volátil, geralmente o HDD. Quando há a necessidade de ler ou escrever um conteúdo no arquivo, a primeira operação a ser feita é a de abertura do arquivo. Após realizar uma das operações de leitura ou escrita, devemos salvar o conteúdo e fechar o arquivo. Dessa forma o arquivo estará livre para que outros softwares ou pessoas possam utilizar o arquivo.

Assim, no Python, essas operações também podem ser programadas através das operações de abertura, leitura ou escrita e fechamento do arquivo. Existem formas mais elegantes de se tratar arquivos, com bibliotecas mais sofisticadas, mas antes é necessário conhecer o básico.

Para abrir um arquivo no Python há um método embutido chamado `open()` que realiza a abertura de um arquivo, deixando-o disponível para as operações.

```
arquivo = open("teste.txt")  
arquivo = open("C:/Usuários/nomeDoUsuario/teste.txt")
```

Ainda é possível especificar os modos de abertura do arquivo, cada modo indica o tipo de operação que poderá ser efetuado no arquivo. Para leitura r, escrita w, adição de conteúdo ao já existente a. Ainda é possível especificar se queremos abrir o arquivo no formato texto ou binário. O padrão é ler o arquivo no formato texto, dessa forma é possível recuperar os conteúdos linha a linha. Arquivos do tipo imagem ou vídeo devem ser abertos no formato binário e tratado o conjunto bytes recuperados. Outros modos de abertura de arquivo podem ser vistos na Tabela 8.

Modo	Descrição
r	Abre um arquivo para leitura
w	Abre um arquivo para escrita. Cria um novo arquivo, caso ele não exista ou apaga o arquivo existente criando um novo
x	Abre um arquivo para criação exclusiva. Caso o arquivo já exista, a operação falha
a	Abre um arquivo para adicionar um novo conteúdo ao final do conteúdo existente, sem que os dados sejam apagados. Cria um novo arquivo caso não exista
t	Abre um arquivo em modo texto
b	Abre um arquivo em modo binário
+	Abre um arquivo para atualização (leitura e escrita)

Tabela 8: Tabela com os Modos de abertura de arquivo. Fonte: O Autor

```
f = open("teste.txt") # equivalente a 'r' ou 'rt'  
f = open("teste.txt", 'w') # escrita em modo texto  
f = open("imagem.bmp", 'r+b') # leitura e escrita em modo binário
```

Uma preocupação que se deve ter ao lidar com arquivos texto é codificação a qual ele foi criado, pois cada uma indica uma forma de mapear os caracteres dentro do Sistema Operacional. As mais comuns são cp1252 do Windows e utf-8 do Linux. Portanto, não devemos depender também da codificação padrão ou nosso código se comportará de maneira diferente em diferentes plataformas, ao trabalhar com arquivos em modo de texto, é altamente recomendável especificar o tipo de codificação.

```
f = open("test.txt", mode='r', encoding='utf-8')
```

Quando terminarmos de executar as operações no arquivo, precisamos fechar o arquivo corretamente. Fechar um arquivo liberará os recursos vinculados a ele. Isso é feito usando o método `close()` disponível em Python. Python tem um coletor de lixo para limpar objetos não referenciados, mas não devemos confiar nele para fechar o arquivo.

```
f = open("teste.txt", encoding = 'utf-8')
# execute suas operações
f.close()
```

Este método não é totalmente seguro. Se ocorrer uma exceção quando estivermos executando alguma operação com o arquivo, o código será encerrado sem fechar o arquivo. Uma maneira mais segura é usar um bloco `try ... finally`.

```
try:
    f = open("teste.txt", encoding = 'utf-8')
    # execute suas operações
finally:
    f.close()
```

Assim há a garantia de que o arquivo será fechado corretamente, mesmo que seja gerada uma exceção que faça com que o fluxo do programa pare. A melhor maneira de fechar um arquivo é usando a instrução `with`. Isso garante que o arquivo seja fechado quando o bloco dentro da instrução `with` for encerrado. Não precisamos chamar explicitamente o método `close()`, é feito internamente.

```
with open("teste.txt", encoding = 'utf-8') as f:
    # execute suas operações
```

Com o objetivo de escrever em um arquivo em Python, precisamos abri-lo em modo escrita `w`, modo de adicionar conteúdo a ou modo de criação exclusivo `x`. É necessário ter cuidado com o modo `w`, pois ele sobrescreverá o arquivo caso exista. A escrita de uma string ou sequência de bytes (para arquivos binários) é feita usando o método `write()`.

```
with open("teste.txt", 'w', encoding = 'utf-8') as f:
    f.write("Esse é o meu primeiro arquivo\n")
    f.write("Esse arquivo\n")
    f.write("possui três linhas\n")
```

Este programa criará um novo arquivo chamado `teste.txt` no diretório atual se ele não existir. Se existir, será sobrescrito. Devemos incluir os próprios caracteres de nova linha para distinguir as diferentes linhas.

Para ler um arquivo em Python, devemos abri-lo no modo leitura r. Usaremos o método read(tamanho) para ler, tamanho indica a quantidade de dados que serão lidos. Caso não especifique o atributo tamanho, o método irá ler e retornar todo o conteúdo.

```
f = open("teste.txt", 'r', encoding = 'utf-8')
f.read(4)    # lê os primeiros 4 dados disponíveis
f.read(4)    # lê os próximos 4 dados disponíveis
f.read()     # lê o restante do documento a partir da leitura anterior
```

Você verá que o método read() retorna o caractere '\n'. Uma vez que o final do arquivo é alcançado, obtemos uma string vazia em outras leituras. É possível mudar o cursor atual, ou seja, a posição de leitura dentro do arquivo usando o método seek() com um parâmetro numérico indicando a posição que se deseja estar. Também é possível verificar a posição atual do arquivo utilizando o método tell() que retorna a posição do cursor em números de bytes (equivalente a um caractere).

```
f.tell()
f.seek(0)
print(f.read())
```

É possível realizar a leitura linha a linha utilizando uma estrutura de repetição, é rápido e eficiente.

```
f.seek(0)
for line in f:
    print(line, end = ' ')
```

Neste programa, as linhas no próprio arquivo incluem um caractere de nova linha \n. Portanto, usamos o parâmetro end da função print() para evitar duas novas linhas ao imprimir. De modo alternativo, podemos usar o método readline() para ler linhas individuais de um arquivo. Este método lê um arquivo até a nova linha, incluindo o caractere de nova linha.

```
f.seek(0)
f.readline()
f.readline()
f.readline()
```

Por fim, o método readlines() retorna uma lista contendo as linhas do documento inteiro a partir do cursor atual. Sempre que o cursor estiver no final do arquivo, esses métodos de leitura irão retornar um caractere vazio. Por isso é importante verificar sempre a posição do cursor e, se necessário, retorná-lo ao início com o método seek().

```
f.seek(0)
```



```
f.readlines()
```

Outros métodos podem ser encontrados na documentação oficial da linguagem: <https://docs.python.org/pt-br/3.7/>.

Web Scraping

Web Scraping é o nome dado ao processo de coleta e tratamento de dados puros vindos da web. Esse processo é importante para recuperação da informação, dando o primeiro passo para criação de um conjunto de dados para análises futuras. Alguns sites proíbem explicitamente os usuários de extrair seus dados com ferramentas automatizadas. Os sites fazem isso por dois motivos principais:

- (1) O site tem um bom motivo para proteger seus dados. Por exemplo, o Google Maps não permite que você solicite muitos resultados muito rapidamente.
- (2) Fazer muitas solicitações repetidas ao servidor de um site pode consumir largura de banda, tornando o site mais lento para outros usuários e potencialmente sobrecarregando o servidor de forma que o site pare de responder completamente.

IMPORTANTE: antes de pensar criar um *Web Scraping*, verifique as políticas de utilização dos dados do site alvo do trabalho. A violação dos termos de uso pode acarretar em ações penais. Alguns portais são *Web Scraping Friendly*, outros permitem acesso apenas em parte de conteúdo, já os demais não permitem acesso algum. Você conseguirá essas informações nos arquivo robot.txt na raiz dos sites ou através das tags HTML meta name="robots". Fique atento!

A comunidade Pythonista se empenhou bastante para construir pacotes que permitam facilmente realizar o acesso a sites e extrair dados relevantes. A principal biblioteca para acesso é `urllib` e junto dela utiliza-se com frequência a `beautifulsoup`. Serão realizados alguns testes em sites criados para tal atividade no portal Real Python.

Primeiro devemos importar da biblioteca o método que irá realizar a abertura do portal, em seguida o endereço do portal será armazenado em uma variável para facilitar a utilização no decorrer da atividade, por fim o método `urlopen()` será utilizado para carregar a página para um objeto que possa ser utilizado:

```
from urllib.request import urlopen
```

```
url = "http://olympus.realpython.org/profiles/aphrodite"
page = urlopen(url)
page
```

Ao examinar o conteúdo do objeto Page você verá que ele é um objeto do HTTPResponse. Para extrair o conteúdo HTML da página você precisará utilizar o método `read()` a partir do objeto Page criado, esse método irá retornar uma sequência de bytes. Então será necessário decodificar o conteúdo utilizando o `decode()` indicando a codificação desejada, UTF-8, por exemplo:

```
html_bytes = page.read()
html = html_bytes.decode("utf-8")
print(html)
```

Assim você conseguirá investigar o conteúdo html recuperado e então poderá decidir o que deseja explorar no site alvo da análise. Uma maneira de explorar é utilizando métodos string, pois agora o conteúdo que você possui é um grande conjunto de caracteres. Então será possível encontrar trechos específicos e então fatiar o texto para recuperar o conteúdo dentro das tags. O método `find()` irá retornar o índice da primeira ocorrência do conteúdo pesquisado. No exemplo a seguir iremos encontrar a tag `<title>` que contém o título da página, mas para recuperar o conteúdo é necessário encontrar o seu par `</title>`. Em seguida é preciso ajustar os índices para então extrair conteúdo:

```
title_index = html.find("<title>")
print(title_index)
start_index = title_index + len("<title>")
print(start_index)
end_index = html.find("</title>")
print(end_index)
title = html[start_index:end_index]
print(title)
```

Esse é o cenário perfeito, mas no mundo real as coisas podem ser bem diferentes do que se imagina. Utilizando outro endereço do Real Python, vejamos o que ocorre ao utilizar os mesmos passos anteriormente vistos:

```
url = "http://olympus.realpython.org/profiles/poseidon"
page = urlopen(url)
html = page.read().decode("utf-8")
start_index = html.find("<title>") + len("<title>")
end_index = html.find("</title>")
title = html[start_index:end_index]
title
```

Você perceberá um retorno incorreto do conteúdo esperado, pois não examinou anteriormente o conteúdo e a tag `<title>` possui uma pequena diferença do endereço

anterior, um espaço a mais. Para o navegador de internet, não faz a mínima diferença, porém para a extração de conteúdo faz, devido o método find() buscar exatamente o conteúdo informado. Então, o que resta é utilizar uma forma que seja genérica e que permita encontrar as tags independente de como elas estejam escritas.

As expressões regulares, ou regex, são padrões que podem ser utilizados para busca por texto dentro de uma sequência de caracteres (string). Para utilizar no Python iremos incorporar ao projeto a biblioteca re. O regex utiliza caracteres coringas para identificar diferentes padrões de conteúdo, por exemplo, o asterisco (*) serve para informar que um determinado caractere pode ou não ocorrer na sequência analisada. O método utilizado será findall que recupera o conteúdo que combina com o resultado da busca, vejamos alguns exemplos:

```
import re
print(re.findall("ab*c", "ac"))
print(re.findall("ab*c", "abcd"))
print(re.findall("ab*c", "acc"))
print(re.findall("ab*c", "abcac"))
print(re.findall("ab*c", "abdc"))
print(re.findall("ab*c", "ABC"))
#Devido a diferença de maiúsculo e minúsculo, o retorno será vazio

print(re.findall("ab*c", "ABC", re.IGNORECASE))
#Para resolver, informamos que é para ignorar a diferença
```

Outro caractere que pode ser utilizado para ajudar a identificar um padrão que possui um único caractere qualquer separando, é o ponto (.). A combinação de ponto com asterisco permite encontrar padrões que iniciam por determinado conteúdo, finalizando por outro e que no intervalo pode haver qualquer caractere não informado no padrão de busca. Vejamos:

```
print(re.findall("a.c", "abc"))
print(re.findall("a.c", "abbc"))
print(re.findall("a.c", "ac"))
print(re.findall("a.c", "acc"))
print(re.findall("a.*c", "abc"))
print(re.findall("a.*c", "abbc"))
print(re.findall("a.*c", "ac"))
print(re.findall("a.*c", "acc"))
```

Para uma busca mais robusta, podemos utilizar o método search() que busca diversos padrões no conteúdo e retorna um objeto com os agrupamentos encontrados. Então basta

no objeto recuperado utilizar o método `group()` que ele irá retornar o primeiro resultado, que geralmente é o que está se buscando.

```
match_results = re.search("ab*c", "ABC", re.IGNORECASE)
match_results.group()
```

Algumas vezes você pode querer realizar substituições dentro do conteúdo, então o método `sub()` será útil. E o caractere de interrogação (?) permitirá dentro de um conteúdo muito extenso buscar padrões mais próximos e isso será muito útil futuramente. Vejamos:

```
texto = "Todos somos <replaced> mesmo em pequenas experiências de <tags>."
texto = re.sub("<.*>", "CIENTISTAS", texto)
print(texto)
```

Nesse primeiro caso, a busca substituiu todo o conteúdo "<replaced> mesmo em pequenas experiências de<tags>." Por "CIENTISTAS", quando na verdade o resultado esperado deveria ser esse "Todos somos CIENTISTAS mesmo em pequenas experiências de CIENTISTAS.". Nesse momento a interrogação atua para auxiliar, veja:

```
texto = "Todos somos <replaced> mesmo em pequenas experiências de <tags>."
texto = re.sub("<.*?>", "CIENTISTAS", texto)
print(texto)
```

Retornando ao Scraping, veja como podemos utilizar o regex o que aprendemos até aqui para recuperar o título de outra página do Real Python:

```
import re
from urllib.request import urlopen
url = "http://olympus.realpython.org/profiles/dionysus"
page = urlopen(url)
html = page.read().decode("utf-8")
pattern = "<title.*?>.*?</title.*?>"
match_results = re.search(pattern, html, re.IGNORECASE)
title = match_results.group()
title = re.sub("<.*?>", "", title) # Remove as tags HTML
print(title)
```

Embora o regex resolva alguns problemas na hora de extrair dados de sites com sua busca por padrões, é mais fácil utilizar um analisador de páginas HTML para buscar pelos conteúdos disponíveis. Nesse sentido o BeautifulSoup é uma boa biblioteca para auxiliar com essas atividades. Primeiro, é necessário instalar e fazemos isso executando a linha de comando a seguir, ela pode ser executada no próprio Jupyter Notebook e também pode ser executada no Terminal/Prompt de comando de sua preferência.

```
#Jupyter Notebook
!pip install beautifulsoup4
```

```
#Terminal
python3 -m pip install beautifulsoup4
```

Primeiro precisamos criar um objeto para então poder usar toda sua facilidade. O trecho a seguir realiza a abertura uma outra página do Real Python, então decodifica e cria um objeto BeautifulSoup. Vejamos:

```
from bs4 import BeautifulSoup
from urllib.request import urlopen
url = "http://olympus.realpython.org/profiles/dionysus"
page = urlopen(url)
html = page.read().decode("utf-8")
soup = BeautifulSoup(html, "html.parser")
```

A partir desse ponto, podemos utilizar o soup para poder acessar os elementos da página facilmente, por exemplo para recuperar todo texto disponível:

```
print(soup.get_text())
```

Para encontrar tags específicas:

```
soup.find_all("img") #retornará todas as imagens disponíveis
```

Para armazenar as imagens em objetos para futura utilização, podemos fazer:

```
img1, img2 = soup.find_all("img")
```

Então podemos acessar as propriedades de cada uma das imagens, por exemplo o nome delas:

```
print(img1.name)
```

Para acessar o título da página, fazemos assim:

```
print(soup.title)
```

Se quisermos apenas o conteúdo sem as tags:

```
print(soup.title.string)
```

Para mais informações sobre a biblioteca, acesse a documentação oficial: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/> .

Exercícios

Observação: a maioria dos exercícios foram extraídos de <https://wiki.python.org.br/ListaDeExercicios>

[Entrada Processamento e Saída]

1. Faça um Programa que peça um número e então mostre a mensagem O número informado foi [número].
2. Faça um Programa que peça as 4 notas bimestrais e mostre a média.
3. Faça um Programa que converta metros para centímetros.
4. Faça um Programa que peça o raio de um círculo, calcule e mostre sua área.
5. Faça um Programa que calcule a área de um quadrado
6. Faça um Programa que pergunte quanto você ganha por hora e o número de horas trabalhadas no mês. Calcule e mostre o total do seu salário no referido mês.
7. Tendo como dado de entrada a altura (h) de uma pessoa, construa um algoritmo que calcule seu peso ideal, utilizando as seguintes fórmulas:
 - a. Para homens: $(72.7 * h) - 58$
 - b. Para mulheres: $(62.1 * h) - 44.7$
8. Faça um Programa que pergunte quanto você ganha por hora e o número de horas trabalhadas no mês. Calcule e mostre o total do seu salário no referido mês, sabendo-se que são descontados 11% para o Imposto de Renda, 8% para o INSS e 5% para o sindicato, faça um programa que nos dê:
 - a. salário bruto.
 - b. quanto pagou ao INSS.
 - c. quanto pagou ao sindicato.
 - d. o salário líquido.
 - e. calcule os descontos e o salário líquido, conforme a tabela abaixo:

```
+ Salário Bruto : R$  
- IR (11%) : R$  
- INSS (8%) : R$  
- Sindicato ( 5%) : R$  
= Salário Líquido : R$
```

9. Calcule e exiba a quantidade de salários mínimos que um determinado funcionário ganha. Para isto, peça o valor do seu salário e o valor do salário mínimo atual.
10. Calcule e exiba o valor final de uma dívida. Para isto pergunte ao usuário o valor inicial do débito, a quantidade de meses e os juros mensais. Use o cálculo de juros simples.

[Estruturas de Dados]

Observação: exercícios retirados/adaptados de:

- (1) <http://www2.ic.uff.br/~vanessa/material/prog-python/11-TuplasDicionarios.pdf>
- (2) https://www.ufsm.br/app/uploads/sites/679/2019/08/aula_2_Python.pdf
- (3) <https://github.com/andreinaoliveira/Exercicios-Python>
- (4) <https://wiki.python.org.br/ListaDeExercicios>

1. Crie uma **tupla** preenchida com os 20 primeiros colocados da Tabela do Campeonato Brasileiro de Futebol 2020, na ordem de colocação. Depois mostre:

- a. Os 5 primeiros times.
 - b. Os últimos 4 colocados.
 - c. Times em ordem alfabética.
 - d. Em que posição está o São Paulo Futebol Clube.
2. Desenvolva um programa que leia quatro valores pelo teclado e guarde-os em uma **tupla**. No final, mostre:
 - a. Quantas vezes apareceu o valor 9.
 - b. Em que posição foi digitado o primeiro valor 3.
 - c. Quais foram os números pares.
3. Crie um programa onde o usuário possa digitar vários valores numéricos e cadastre-os em uma **lista**. Caso o número já exista lá dentro, ele não será adicionado. No final, serão exibidos todos os valores únicos digitados, em ordem crescente.
4. Crie um programa que vai ler vários números e colocar em uma **lista**. Depois disso, mostre:
 - a. Quantos números foram digitados.
 - b. A lista de valores, ordenada de forma decrescente.
 - c. Se o valor 5 foi digitado e está ou não na lista.
5. Faça um programa que leia nome e média de um aluno, guardando também a situação em um **dicionário**. No final, mostre o conteúdo da estrutura na tela.
6. Faça um programa que pode receber várias notas de alunos e vai retornar um dicionário com as seguintes informações:
 - a. Quantidade de notas
 - b. A maior nota
 - c. A menor nota
 - d. A média da turma
7. Faça um programa que remova todos os números pares de uma lista de 0 até 10.
8. Crie um programa que armazena um grupo de 3 pessoas contendo o número de telefone de cada uma e a idade de cada uma e depois mostre todas as informações e a informação apenas de uma pessoa.
9. Uma pista de Kart permite 10 voltas para cada um de 6 corredores. Escreva um programa que leia todos os tempos em segundos e os guarde em um dicionário, onde a chave é o nome do corredor. Ao final diga de quem foi a melhor volta da prova e em que volta; e ainda a classificação final em ordem (1o o campeão). O campeão é o que tem a menor média de tempos.
10. Escreva um programa para armazenar uma agenda de telefones em um dicionário. Cada pessoa pode ter um ou mais telefones e a chave do dicionário é o nome da pessoa.

11. Faça um Programa que leia 5 números inteiros e mostre-os. Escolha uma estrutura para colecionar seus dados.
12. Faça um Programa que leia 10 números reais e mostre-os na ordem inversa. Escolha uma estrutura para colecionar seus dados.
13. Foram anotadas as idades e alturas de 30 alunos. Faça um Programa que determine quantos alunos com mais de 13 anos possuem altura inferior à média de altura desses alunos. Escolha uma estrutura para colecionar seus dados.
14. Faça um programa que receba a temperatura média de cada mês do ano e armazene-as em uma lista. Após isto, calcule a média anual das temperaturas e mostre todas as temperaturas acima da média anual, e em que mês elas ocorreram (mostrar o mês por extenso: 1 – Janeiro, 2 – Fevereiro, . . .). Escolha uma estrutura para colecionar seus dados.
15. Utilizando listas faça um programa que faça 5 perguntas para uma pessoa sobre um crime. As perguntas são:
 - i. "Telefonou para a vítima?"
 - ii. "Esteve no local do crime?"
 - iii. "Mora perto da vítima?"
 - iv. "Devia para a vítima?"
 - v. "Já trabalhou com a vítima?"
 - a. O programa deve no final emitir uma classificação sobre a participação da pessoa no crime. Se a pessoa responder positivamente a 2 questões ela deve ser classificada como "Suspeita", entre 3 e 4 como "Cúmplice" e 5 como "Assassino". Caso contrário, ele será classificado como "Inocente".
16. Faça um programa que carregue uma lista com os modelos de cinco carros (exemplo de modelos: FUSCA, GOL, VECTRA etc). Carregue uma outra lista com o consumo desses carros, isto é, quantos quilômetros cada um desses carros faz com um litro de combustível. Calcule e mostre:
 - a. O modelo do carro mais econômico;
 - b. Quantos litros de combustível cada um dos carros cadastrados consome para percorrer uma distância de 1000 quilômetros e quanto isto custará, considerando um que a gasolina custe R\$ 6,25 o litro.
17. Faça um programa que simule um lançamento de dados. Lance o dado 100 vezes e armazene os resultados em um vetor
 - a. Depois, mostre quantas vezes cada valor foi conseguido. Dica: use uma lista de contadores (1-6) e uma função para gerar números aleatórios (random), simulando os lançamentos dos dados.

[Estruturas Condicionais e Estruturas de Repetição]

Observação: a maioria dos exercícios foram extraídos de <https://wiki.python.org.br/ListaDeExercicios>

1. Faça um Programa que peça dois números e imprima o maior deles.
2. Faça um Programa que peça um valor e mostre na tela se o valor é positivo ou negativo.
3. Faça um Programa que verifique se uma letra digitada é "F" ou "M". Conforme a letra escrever: F - Feminino, M - Masculino, Sexo Inválido.
4. Faça um Programa que verifique se uma letra digitada é vogal ou consoante.
5. Faça um programa para a leitura de duas notas parciais de um aluno. O programa deve calcular a média alcançada por aluno e apresentar:
 - a) A mensagem "Aprovado", se a média alcançada for maior ou igual a sete;
 - b) A mensagem "Reprovado", se a média for menor do que sete;
 - c) A mensagem "Aprovado com Distinção", se a média for igual a dez.
6. Faça um Programa que leia três números e mostre o maior deles.
7. Faça um Programa que leia três números e mostre o maior e o menor deles.
8. Faça um programa que pergunte o preço de três produtos e informe qual produto você deve comprar, sabendo que a decisão é sempre pelo mais barato.
9. Faça um Programa que leia três números e mostre-os em ordem decrescente.
10. Faça um Programa que pergunte em que turno você estuda. Peça para digitar M-matutino ou V-Vespertino ou N- Noturno. Imprima a mensagem "Bom Dia!", "Boa Tarde!" ou "Boa Noite!" ou "Valor Inválido!", conforme o caso.
11. As Organizações Tabajara resolveram dar um aumento de salário aos seus colaboradores e lhe contrataram para desenvolver o programa que calculará os reajustes.
 - a) Faça um programa que recebe o salário de um colaborador e o reajuste segundo o seguinte critério, baseado no salário atual:
 - b) salários até R\$ 280,00 (incluindo) : aumento de 20%
 - c) salários entre R\$ 280,00 e R\$ 700,00 : aumento de 15%
 - d) salários entre R\$ 700,00 e R\$ 1500,00 : aumento de 10%
 - e) salários de R\$ 1500,00 em diante : aumento de 5% Após o aumento ser realizado, informe na tela:
 - f) o salário antes do reajuste;
 - g) o percentual de aumento aplicado;
 - h) o valor do aumento;
 - i) o novo salário, após o aumento.
12. Faça um programa para o cálculo de uma folha de pagamento, sabendo que os descontos são do Imposto de Renda, que depende do salário bruto (conforme tabela

abaixo) e 3% para o Sindicato e que o FGTS corresponde a 11% do Salário Bruto, mas não é descontado (é a empresa que deposita). O Salário Líquido corresponde ao Salário Bruto menos os descontos. O programa deverá pedir ao usuário o valor da sua hora e a quantidade de horas trabalhadas no mês.

Desconto do IR:

- a) Salário Bruto até 900 (inclusive) - isento
 - b) Salário Bruto até 1500 (inclusive) - desconto de 5%
 - c) Salário Bruto até 2500 (inclusive) - desconto de 10%
 - d) Salário Bruto acima de 2500 - desconto de 20%
- Imprima na tela as informações, dispostas conforme o exemplo abaixo. No exemplo o valor da hora é 5 e a quantidade de hora é 220.

Salário Bruto: (5 * 220)	: R\$ 1100,00
(-) IR (5%)	: R\$ 55,00
(-) INSS (10%)	: R\$ 110,00
FGTS (11%)	: R\$ 121,00
Total de descontos	: R\$ 165,00
Salário Líquido	: R\$ 935,00

13. Faça um Programa que leia um número e exiba o dia correspondente da semana. (1- Domingo, 2- Segunda, etc.), se digitar outro valor deve aparecer valor inválido.
14. Faça um programa que lê as duas notas parciais obtidas por um aluno numa disciplina ao longo de um semestre, e calcule a sua média. A atribuição de conceitos obedece à tabela abaixo:

Média de Aproveitamento	Conceito
Entre 9.0 e 10.0	A
Entre 7.5 e 9.0	B
Entre 6.0 e 7.5	C
Entre 4.0 e 6.0	D
Entre 4.0 e zero	E

- a) O algoritmo deve mostrar na tela as notas, a média, o conceito correspondente e a mensagem “APROVADO” se o conceito for A, B ou C ou “REPROVADO” se o conceito for D ou E.
15. Faça um Programa que peça os 3 lados de um triângulo. O programa deverá informar se os valores podem ser um triângulo. Indique, caso os lados formem um triângulo, se o mesmo é: equilátero, isósceles ou escaleno.

Dicas:

- a) Três lados formam um triângulo quando a soma de quaisquer dois lados for maior que o terceiro;
- b) Triângulo Equilátero: três lados iguais;
- c) Triângulo Isósceles: quaisquer dois lados iguais;
- d) Triângulo Escaleno: três lados diferentes;

16. Faça um programa que calcule as raízes de uma equação do segundo grau, na forma $ax^2 + bx + c$. O programa deverá pedir os valores de a, b e c e fazer as consistências, informando ao usuário nas seguintes situações:
- a) Se o usuário informar o valor de A igual a zero, a equação não é do segundo grau e o programa não deve pedir os demais valores, sendo encerrado;
 - b) Se o delta calculado for negativo, a equação não possui raízes reais. Informe ao usuário e encerre o programa;
 - c) Se o delta calculado for igual a zero a equação possui apenas uma raiz real; informe-a ao usuário;
 - d) Se o delta for positivo, a equação possui duas raízes reais; informe-as ao usuário;
17. Faça um Programa que peça um número correspondente a um determinado ano e em seguida informe se este ano é ou não bissexto.
18. Faça um Programa que peça uma data no formato dd/mm/aaaa e determine se a mesma é uma data válida.
19. Faça um Programa que leia um número inteiro menor que 1000 e imprima a quantidade de centenas, dezenas e unidades do mesmo.

Observando os termos no plural a colocação do "e", da vírgula entre outros. Exemplo:

- a) 326 = 3 centenas, 2 dezenas e 6 unidades
 - b) 12 = 1 dezena e 2 unidades Testar com: 326, 300, 100, 320, 310, 305, 301, 101, 311, 111, 25, 20, 10, 21, 11, 1, 7 e 16
20. Faça um Programa para leitura de três notas parciais de um aluno. O programa deve calcular a média alcançada por aluno e apresentar:
- a) A mensagem "Aprovado", se a média for maior ou igual a 7, com a respectiva média alcançada;
 - b) A mensagem "Reprovado", se a média for menor do que 7, com a respectiva média alcançada;
 - c) A mensagem "Aprovado com Distinção", se a média for igual a 10.
21. Faça um Programa para um caixa eletrônico. O programa deverá perguntar ao usuário a valor do saque e depois informar quantas notas de cada valor serão fornecidas. As notas disponíveis serão as de 1, 5, 10, 50 e 100 reais. O valor mínimo é de 10 reais e o máximo de 600 reais. O programa não deve se preocupar com a quantidade de notas existentes na máquina.
- a) Exemplo 1: Para sacar a quantia de 256 reais, o programa fornece duas notas de 100, uma nota de 50, uma nota de 5 e uma nota de 1;
 - b) Exemplo 2: Para sacar a quantia de 399 reais, o programa fornece três notas de 100, uma nota de 50, quatro notas de 10, uma nota de 5 e quatro notas de 1.

22. Faça um Programa que peça um número inteiro e determine se ele é par ou impar.
Dica: utilize o operador módulo (resto da divisão).
23. Faça um Programa que peça um número e informe se o número é inteiro ou decimal.
Dica: utilize uma função de arredondamento.
24. Faça um Programa que leia 2 números e em seguida pergunte ao usuário qual operação ele deseja realizar. O resultado da operação deve ser acompanhado de uma frase que diga se o número é:
- par ou ímpar;
 - positivo ou negativo;
 - inteiro ou decimal.

25. Faça um programa que faça 5 perguntas para uma pessoa sobre um crime. As perguntas são:

- "Telefonou para a vítima?"
- "Esteve no local do crime?"
- "Mora perto da vítima?"
- "Devia para a vítima?"
- "Já trabalhou com a vítima?"

O programa deve no final emitir uma classificação sobre a participação da pessoa no crime. Se a pessoa responder positivamente a 2 questões ela deve ser classificada como "Suspeita", entre 3 e 4 como "Cúmplice" e 5 como "Assassino". Caso contrário, ele será classificado como "Inocente".

26. Um posto está vendendo combustíveis com a seguinte tabela de descontos:

- Álcool:
 - até 20 litros, desconto de 3% por litro
 - acima de 20 litros, desconto de 5% por litro
 - Gasolina:
 - até 20 litros, desconto de 4% por litro
 - acima de 20 litros, desconto de 6% por litro
- Escreva um algoritmo que leia o número de litros vendidos, o tipo de combustível (codificado da seguinte forma: A-álcool, G-gasolina), calcule e imprima o valor a ser pago pelo cliente sabendo-se que o preço do litro da gasolina é R\$ 2,50 o preço do litro do álcool é R\$ 1,90.

27. Uma fruteira está vendendo frutas com a seguinte tabela de preços:

	Até 5 Kg	Acima de 5 Kg
Morango	R\$ 2,50 por Kg	R\$ 2,20 por Kg
Maçã	R\$ 1,80 por Kg	R\$ 1,50 por Kg

Se o cliente comprar mais de 8 Kg em frutas ou o valor total da compra ultrapassar R\$ 25,00, receberá ainda um desconto de 10% sobre este total. Escreva um algoritmo

para ler a quantidade (em Kg) de morangos e a quantidade (em Kg) de maçãs adquiridas e escreva o valor a ser pago pelo cliente.

28. O Hipermercado Tabajara está com uma promoção de carnes que é imperdível. Confira:

	Até 5 Kg	Acima de 5 Kg
File Duplo	R\$ 4,90 por Kg	R\$ 5,80 por Kg
Alcatra	R\$ 5,90 por Kg	R\$ 6,80 por Kg
Picanha	R\$ 6,90 por Kg	R\$ 7,80 por Kg

Para atender a todos os clientes, cada cliente poderá levar apenas um dos tipos de carne da promoção, porém não há limites para a quantidade de carne por cliente. Se compra for feita no cartão Tabajara o cliente receberá ainda um desconto de 5% sobre o total a compra. Escreva um programa que peça o tipo e a quantidade de carne comprada pelo usuário e gere um cupom fiscal, contendo as informações da compra: tipo e quantidade de carne, preço total, tipo de pagamento, valor do desconto e valor a pagar.

29. Faça um programa que leia um nome de usuário e a sua senha e não aceite a senha igual ao nome do usuário, mostrando uma mensagem de erro e voltando a pedir as informações.
30. Supondo que a população de um país A seja da ordem de 80000 habitantes com uma taxa anual de crescimento de 3% e que a população de B seja 200000 habitantes com uma taxa de crescimento de 1.5%. Faça um programa que calcule e escreva o número de anos necessários para que a população do país A ultrapasse ou iguale a população do país B, mantidas as taxas de crescimento.
31. Altere o programa anterior permitindo ao usuário informar as populações e as taxas de crescimento iniciais. Valide a entrada e permita repetir a operação.
32. Faça um programa que imprima na tela os números de 1 a 20, um abaixo do outro. Depois modifique o programa para que ele mostre os números um ao lado do outro.
33. Faça um programa que leia 5 números e informe o maior número.
34. Modifique o programa anterior para que ele informe a soma e a média dos números.
35. Faça um programa que imprima na tela apenas os números ímpares entre 1 e 50.
36. Faça um programa que receba dois números inteiros e gere os números inteiros que estão no intervalo compreendido por eles.
37. Altere o programa anterior para mostrar no final a soma dos números.
38. Faça um programa que calcule o fatorial de um número inteiro fornecido pelo usuário. Ex.: $5!=5.4.3.2.1=120$

39. Faça um programa que peça um número inteiro e determine se ele é ou não um número primo. Um número primo é aquele que é divisível somente por ele mesmo e por 1.
40. Faça um programa que mostre todos os primos entre 1 e N sendo N um número inteiro fornecido pelo usuário.
41. Faça um programa que peça para n pessoas a sua idade, ao final o programa deverá verificar se a média de idade da turma varia entre 0 e 25, 26 e 60 ou maior que 60. Então, dizer se a turma é jovem, adulta ou idosa, conforme a média calculada.
42. Numa eleição existem três candidatos. Faça um programa que peça o número total de eleitores. Peça para cada eleitor votar e ao final mostrar o número de votos de cada candidato. Exiba o vencedor!
43. Foi feita uma estatística em cinco cidades brasileiras para coletar dados sobre acidentes de trânsito. Foram obtidos os seguintes dados:
 - a) Códigos das cidades;
 - b) Número de veículos de passeio (em 1999);
 - c) Número de acidentes de trânsito com vítimas (em 1999). Deseja-se saber:
 - (1) Qual o maior e menor índice de acidentes de transito e a que cidade pertence;
 - (2) Qual a média de veículos nas cinco cidades juntas;
 - (3) Qual a média de acidentes de trânsito nas cidades com menos de 2.000 veículos de passeio.

[Métodos e Pacotes]

Observação: a maioria dos exercícios foram extraídos de:

- (1) <https://wiki.python.org.br/ListaDeExercicios>
- (2) <https://panda.ime.usp.br/cc110/static/cc110/10-funcoes-listas.html>

1. Faça um programa, com uma função que necessite de três argumentos, e que forneça a soma desses três argumentos.
2. Faça um programa, com uma função que necessite de um argumento. A função retorna o valor de caractere 'P', se seu argumento for positivo, e 'N', se seu argumento for zero ou negativo.
3. Faça um programa com uma função chamada somalmposto. A função possui dois parâmetros formais: taxalmposto, que é a quantia de imposto sobre vendas expressa

em porcentagem e custo, que é o custo de um item antes do imposto. A função “altera” o valor de custo para incluir o imposto sobre vendas.

4. Faça um programa que converta da notação de 24 horas para a notação de 12 horas. Por exemplo, o programa deve converter 14:25 em 2:25 P.M. A entrada é dada em dois inteiros. Deve haver pelo menos duas funções: uma para fazer a conversão e uma para a saída. Registre a informação A.M./P.M. como um valor ‘A’ para A.M. e ‘P’ para P.M. Assim, a função para efetuar as conversões terá um parâmetro formal para registrar se é A.M. ou P.M. Inclua um loop que permita que o usuário repita esse cálculo para novos valores de entrada todas as vezes que desejar.
5. Faça um programa que use a função valorPagamento para determinar o valor a ser pago por uma prestação de uma conta. O programa deverá solicitar ao usuário o valor da prestação e o número de dias em atraso e passar estes valores para a função valorPagamento, que calculará o valor a ser pago e devolverá este valor ao programa que a chamou. O programa deverá então exibir o valor a ser pago na tela. Após a execução o programa deverá voltar a pedir outro valor de prestação e assim continuar até que seja informado um valor igual a zero para a prestação. Neste momento o programa deverá ser encerrado, exibindo o relatório do dia, que conterá a quantidade e o valor total de prestações pagas no dia. O cálculo do valor a ser pago é feito da seguinte forma. Para pagamentos sem atraso, cobrar o valor da prestação. Quando houver atraso, cobrar 3% de multa, mais 0,1% de juros por dia de atraso.
6. Faça uma função que informe a quantidade de dígitos de um determinado número inteiro informado.
7. **Reverso do número.** Faça uma função que retorne o reverso de um número inteiro informado. Por exemplo: 127 -> 721.
8. A probabilidade de dar um valor em um dado é 1/6 (uma em 6). Faça um script em Python que simule 1 milhão de lançamentos de dados e mostre a frequência que deu para cada número.
9. Faça uma função que receba uma lista de itens e um item. Então deverá retornar True se o item é um elemento da lista e False em caso contrario.
10. Recebe um objeto 'item' e uma lista 'lista' e retorna o índice da posição em que item ocorre na lista. Caso item não ocorra na lista a função retorna None.
11. Dados n e uma sequencia com n números inteiros, conta e imprime o numero de vezes que cada numero ocorre na sequencia.
12. Recebe uma lista de números e dois índices ini e fim, retorna a soma da sublista (= fatia) formada pelos elementos de índices ini, ini+1,...,fim-1.

a. Pré-condição: a função supõe que: $0 \leq ini \leq fim \leq len(lista)$

13. Crie uma função que receba quatro valores que indicam dois pontos no eixo cartesiano, você deverá retornar a distância entre os dois pontos como resultado.

Crie um pacote incluindo todas as funções criadas. Crie um script com um menu de opções para utilização de cada função que criou.

Módulo II – A Biblioteca Numpy

Nesse módulo você irá aprender a utilizar a biblioteca Numpy, muito útil para desenvolver algoritmos que utilizam estruturas de dados de alto desempenho, além de possibilitar o trabalho com operações algébricas.

VISÃO GERAL

Numpy é a abreviatura de *Numerical Python*, segundo o site oficial do Numpy “traz o poder computacional de linguagens como C e Fortran para Python[...]”. Possui uma estrutura de vetorização rápida e versátil que utiliza os padrões utilizados pela computação atualmente, tem suporte uma variedade de rotinas e operações rápidas em matrizes, incluindo matemática, lógica, manipulação de forma, classificação, seleção, I/O, transformadas discretas de Fourier, álgebra linear básica, operações estatísticas básicas, simulação, entre outras diversas operações.

O ponto principal dessa biblioteca é a sua estrutura multidimensional, o `ndarray`. Ele encapsula matrizes multidimensionais homogêneas, com operações compiladas que aumentam o desempenho na sua utilização. Uma matriz multidimensional trata de uma estrutura de coleção como uma lista, conjunto ou dicionário. A diferença, nesse caso, é que os objetos armazenados necessariamente possuem o mesmo tipo. Outra vantagem que o `ndarray` traz é a possibilidade de aplicação de operações com menor quantidade de código necessário, já que a maioria das operações possíveis está disponível na biblioteca Numpy. A importância do Numpy é tanta, que diversas outras bibliotecas vem surgindo baseando-se nela.

Introdução à Biblioteca Numpy

O primeiro passo para a utilização da biblioteca é importar para dentro de seu projeto. E existe uma convenção adotada mundialmente da maneira como se deve ser importada, para que possa facilitar a descoberta de conhecimento ao analisar scripts de terceiros e diminuir a quantidade de conteúdo a ser escrito. Vejamos:

```
import numpy as np
```

Criação de ndarray

Um exemplo simples que mostra o poder o Numpy, a seguir, gera uma matriz de duas dimensões e três atributos. Então a essa matriz são executadas simples operações matemáticas que, pela característica da estrutura, são aplicados a todos os elementos de uma única vez. A isso dá-se o nome de broadcast de operações:

```
data = np.random.randn(2, 3)  
print(data)
```

```
print("\nMultiplica todo o conteúdo por 14")
print(data * 14)
print("\nRealiza a soma elemento a elemento da matriz")
print(data + data)
```

O ndarray possui alguns atributos que nos permitem conhecer melhor o conteúdo armazenado, o formato e o número de dimensões:

```
print(data.shape) #Exibe a forma do ndarray. Nº de linhas e Nº de colunas,
respectivamente
print(data.dtype) #Exibe o tipo dos dados armazenados
print(data.ndim) #Exibe o número de dimensões da estrutura
```

É possível utilizar qualquer tipo de objeto de sequências para gerar um novo ndarray Numpy contendo os dados recebidos:

```
lista1 = [6, 7.5, 8, 0, 1] #Uma lista do Python
array1 = np.array(lista1) #O método array da biblioteca Numpy é responsável pela
criação
print(array1)
print(type(array1))
print(array1.ndim)
print(array1.shape)
print(array1.dtype)
lista2 = [[1, 2, 3, 4], [5, 6, 7, 8]] #Lista Python com duas dimensões
array2 = np.array(lista2)
#Analise os atributos da nova lista criada
```

Existem diversas formas de se criar um ndarray, as mais comuns utilizam os métodos zeros(), que gera um ndarray somente com 0s, ones(), que gera um ndarray somente com 1s, eye(), que cria uma matriz identidade quadrada com 1s na diagonal principal e 0s nas demais posições. Outros métodos podem ser encontrados na documentação da biblioteca disponível em <https://numpy.org/doc/stable/reference/routines.array-creation.html?highlight=create>.

Uma operação que será muito útil é mudança de formato, tanto de uma dimensão para N como de N para uma dimensão. No exemplo a seguir criaremos um ndarray de uma dimensão apenas, então transformaremos em 2 dimensões, 3 dimensões e retornaremos para uma dimensão.

```
g = np.arange(24)
print(g)
g.reshape(4,6)
print(g)
g.reshape(2, 3, 4)
print(g)
g.ravel()
```

Operações Matemáticas

A aplicação de operações matemáticas é uma constante na utilização do Numpy, portanto é importante saber as possibilidades existentes para que se saiba utilizar no momento oportuno. Vejamos mais alguns exemplos:

```
array3 = np.array([[1., 2., 3.], [4., 5., 6.]])
print(array3)
print(array3 * array3)
print(array3 - array3)
print(1 / array3)
print(array3 ** 0.5) #Uma forma de aplicar a raiz quadrada
```

Outras operações que são fundamentais é a comparação entre elementos que retorna um tipo binário booleano (True ou False) e que posteriormente pode ser aplicado como filtro em uma visualização de dados específicos. Essa é a base para estruturas mais robustas. Vejamos:

```
array4 = np.array([[0., 4., 1.], [7., 2., 12.]])
print(array4 > array3)
print(array4[array4 > array3]) #Os elementos com resultado True são exibidos
```

Indexação e Fatiamento

Assim como os strings, as listas, os conjuntos e os dicionários, o ndarray permite a indexação e o fatiamento utilizando os índices entre um par de colchetes:

```
array5 = np.arange(10)
print(array5)
print(array5 [5])
print(array5 [9])
print(array5 [5:8])
array5 [5:8] = 12 #Altera dos os elementos entre 5 e 7 para 12
print(array5)
arr_slice = array5[5:8] #Cria um novo ndarray a partir do subconjunto filtrado
print(arr_slice)
arr_slice[1] = 12345 #Altera o conteúdo do segundo elemento
print(array5)
arr_slice[:] = 64 #Altera todos os elementos de uma vez
print(array5)
```

Como é perceptível, a alteração feita em arr_slice implica na alteração de array5 também, isso porque ao efetuar a criação de arr_slice fizemos com que ambas as variáveis endereçassem as mesmas posições de memória. Se essa não for a sua intenção, ao criar uma cópia de um ndarray ou de um subconjunto dele você deverá utilizar o método copy(), esse será o responsável pela cópia do conteúdo e criação de uma nova estrutura endereçando um local diferente e, assim, não causando alterações na estrutura original.

```
arr_copia = array5[5:8].copy()
print(arr_copia)
arr_copia[2] = 32
print(array5)
print(arr_copia)
```

Para o caso de estar trabalhando com ndarray de múltiplas dimensões, você precisará ter certo cuidado na indexação, por não estar tratando de uma única dimensão é importante indicar os índices aderentes a outra dimensão. Na Figura 13 temos uma matriz bidimensional, perceba que cada um dos elementos são indexados pelo número correspondente a linha e a coluna. Isso fará com que uma célula seja acessada corretamente, permitindo que operações sejam feitas exatamente para ela.

		eixo 1		
		0	1	2
eixo 0	0	0,0	0,1	0,2
	1	1,0	1,1	1,2
	2	2,0	2,1	2,2

Figura 13: Matriz Bidimensional – Indexação de Células. Fonte: O Autor

Vejamos um exemplo real de indexação:

```
matriz_bidimensional = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
#Acessando uma linha
print(matriz_bidimensional [2])
```

```
#Acessando uma coluna
print(matriz_bidimensional[:, 1])
```

```
#Acessando uma célula específica
print(matriz_bidimensional[0][2])
print(matriz_bidimensional[0, 2])
```

Para selecionar alguns subconjuntos, o fatiamento permite diversas possibilidades:

```
#Selecionando a 2ª linha, somente as 2 primeiras colunas
print(matriz_bidimensional[1, :2])
print(matriz_bidimensional[0, 2])
```

```
#Selecionando a 3ª coluna, somente as 2 primeiras linhas
print(matriz_bidimensional[:2, 2])
```

```
#Selecionando todo um eixo de determinada dimensão
print(matriz_bidimensional[:, :1])
```

```
print(matriz_bidimensional[:, 1:2])
```

Como visto anteriormente, a comparação de elementos retorna resultado binário do tipo True ou False e é muito útil para filtragem, a essa filtragem damos o nome de indexação booleana, devido ao tipo retornado. Vejamos um exemplo concreto com nomes de pessoas e números associados a essas pessoas, podendo ser nota, salário, entre outras características do mundo real. Primeiro será criado um ndarray de nomes, em seguida uma matriz com sete linhas e quatro colunas, cada linha representa um nome e as colunas os valores associados a eles. Em seguida será realizado um filtro e aplicado a matriz para exibir apenas os números associados ao nome filtrado. Veja:

```
nomes = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
print(nomes)
dados = np.random.randn(7, 4)
print(nomes == 'Bob') #Filtro que será aplicado
print(dados[nomes == 'Bob']) #Linhas contendo os números associados ao Bob
```

```
#Exibindo apenas os registros de dados que não possuem Bob como nome associado
print(nomes != 'Bob')
print(dados[~(nomes == 'Bob')])
```

```
#Criando uma variável para armazenar o resultado do filtro e então aplicá-lo como
indexador
condicao = nomes == 'Will'
print(dados[condicao])
condicao = nomes == 'Bob'
print(dados[~condicao])
```

Se quisermos aplicar mais de uma condição, podemos utilizar operadores lógicos para aplicar. Os operadores de junção (and) e desjunção (or) são representados por & e |, respectivamente. Lembrando que na junção, sempre que ambas as condições forem True o resultado final será True, qualquer outra combinação retorna False. Já a desjunção, somente terá um retorno False quando ambas as condições forem False, do contrário o resultado sempre será True. Exemplificando:

```
maskara = (nomes == 'Bob') | (nomes == 'Will')
print(maskara)
print(dados[maskara])
```

Lembre-se que podemos utilizar os operadores relacionais para comparação numérica e isso se aplica aqui também:

```
print(dados[dados > 0])
dados[dados < 0] = 0
print(dados)
```

Uma maneira interessante de se realizar um filtro é indicado os diversos índices que almeja exibir de uma única vez:

```
#Criando um ndarray dinamicamente
array6 = np.empty((8,4))

#Alterando os valores de cada linha
for i in range(8):
    array6[i] = i

#Filtrando os índices na ordem que deseja-se exibir
print(array6[[4, 3, 0, 6]])

#Contando os índices trás para frente
print(array6[[-3, -5, -7]])
```

Funções Universais

Uma função universal é aquela que é aplicada a todos os elementos de ndarray. A documentação traz diversos métodos implementados que podem ser utilizados quando se tratam de álgebra linear, métodos estatísticos e tratamento de dados. Vejamos alguns exemplos:

```
matriz1 = np.random.randn(8,3)
matriz2 = np.random.randn(8,3)
print(np.add(matriz1, matriz2)) #Soma elemento a elemento das matrizes
print(np.matmul(matriz1, matriz2.T)) #Realiza multiplicação de matrizes
print(np.negative(matriz1)) #Multiplica todos os elementos por -1
print(np.mod(matriz1, matriz2)) #Retorna o resto da divisão de elemento por elemento
print(np.absolute(matriz1)) #Retorna o absoluto de cada elemento
print(np rint(matriz1)) #Arredonda cada elemento para o inteiro mais próximo
```

Ainda existem funções trigonométricas que podem ser aplicadas:

```
print(np.sin(matriz1)) #Calcula o seno de cada elemento
print(np.cos(matriz1)) #Calcula o cosseno de cada elemento
print(np.tan(matriz1)) #Calcula a tangente de cada elemento
```

Além, é claro de funções de comparação:

```
#Retorna True nos elementos correspondente a matriz1 > matriz2
print(np.greater(matriz1, matriz2))

#Retorna True nos elementos correspondente a matriz1 >= matriz2
print(np.greater_equal(matriz1, matriz2))

#Retorna True nos elementos correspondente a matriz1 < matriz2
print(np.less(matriz1, matriz2))

#Retorna True nos elementos correspondente a matriz1 <= matriz2
print(np.less_equal(matriz1, matriz2))
```

```
#Retorna True nos elementos correspondente a matriz1 != matriz2
print(np.not_equal(matriz1, matriz2))
```

```
#Retorna True nos elementos correspondente a matriz1 == matriz2
print(np.equal(matriz1, matriz2))
```

Outros métodos da biblioteca Numpy podem ser consultados na documentação oficial:
<https://numpy.org/doc/stable/reference/ufuncs.html>.

Métodos Estatísticos

A estrutura ndarray possui métodos estatísticos que permite realizar uma análise descritiva dos dados armazenados na estrutura. Por exemplo, podemos calcular estimativas de localização e estimativas de variabilidade:

```
renda_ingressantes = np.array([2.90, 2.90, 2.95, 2.95, 3.10, 3.10, 3.15, 3.20,
3.20, 3.25, 3.30, 3.40, 3.45, 3.45, 3.50, 3.65, 3.65, 3.80, 3.90, 3.90, 4.00,
5.00, 5.20, 5.50, 6.40])
```

```
#Estimativas de Localização
print(np.mean(renda_ingressantes)) #Média
print(np.median(renda_ingressantes)) #Mediana
```

```
#Estimativas de Variabilidade
print(np.var(renda_ingressantes)) #Variância
print(np.std(renda_ingressantes)) #Desvio-Padrão
```

```
#Distribuição de Dados
print(np.percentile(renda_ingressantes, 25)) #Retorna o percentil indicado pelo
número após a vírgula
print(np.quantile(renda_ingressantes, 0.25)) #Retorna o quantil indicado pelo
número após a vírgula
print(np.quantile(renda_ingressantes, 0.50)) #Retorna o quantil indicado pelo
número após a vírgula
print(np.quantile(renda_ingressantes, 0.75)) #Retorna o quantil indicado pelo
número após a vírgula
```

```
X = np.array([1, 2, 3])
Y = np.array([0, 1, 0.5])
#Correlação
print(np.corrcoef(X, Y)) #Calcula o coeficiente de correlação de Pearson
print(np.correlate(X, Y)) #Calcula a correlação dos atributos
```

Outros métodos podem ser analisados em:
<https://numpy.org/doc/stable/reference/routines.statistics.html?highlight=statistical>.

Ordenação e Unicidade de Elementos

Assim como as estruturas embutidas no Python, o Numpy possui método para ordenação e para recuperar apenas uma única ocorrência por vez de dados armazenados no ndarray. Observe:


```
array7 = np.array([2.52, 3.42, 1.85, 2.95, 3.95, 3.78, 3.17, 3.96, 2.97, 2.98,
3.35, 4.23, 3.42, 4.60, 5.24, 3.64, 4.54, 4.81, 1.77, 1.51, 3.08, 2.00])
#Colocando em ordem
print(np.sort(array7))
```

No caso de um ndarray de mais dimensões, é possível indicar se a ordenação será nas colunas ou nas linhas:

```
matriz = np.array([[1, 4], [3, 1]])
print(matriz)
print(np.sort(matriz, axis = 1)) #Ordena as colunas
print(np.sort(matriz, axis = 0)) #Ordena as linhas
```

O método de unicidade permite recuperar uma única ocorrência dos dados armazenados e ordená-los:

```
nomes = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
print(np.unique(nomes))
numeros = np.array([3, 3, 2, 2, 1, 1, 1, 4, 4])
print(np.unique(numeros))
```

Talvez em algum momento você precise unificar ndarrays, existem algumas formas de se realizar essa tarefa. Podemos utilizar métodos de empilhamento (*stacking*) ou concatenação (*concatenate*). Vejamos:

```
# Criando nossos ndarray de exemplo
q1 = np.full((3,4), 1.0)
q1
q2 = np.full((4,4), 2.0)
q2
q3 = np.full((3,4), 3.0)
q3

# Empilhando verticalmente
q4 = np.vstack((q1, q2, q3))
q4
q4.shape

# Empilhando horizontalmente
q5 = np.hstack((q1, q3)) # q1 e q3 possuem mesma quantidade de linhas
q5
q5.shape

# Utilizando concatenate para replicar o vstack
q7 = np.concatenate((q1, q2, q3), axis=0)
q7

# Utilizando concatenate para replicar o hstack
q8 = np.concatenate((q1, q3), axis=1)
q8
```

Ainda existe muito a se explorar na biblioteca Numpy. Você pode conhecer mais e ver outros exemplos na [documentação oficial](#) e [nesse notebook escrito por Aurélien Géron](#), autor do livro Hands-On: Machine Learning with Scikit-Learn, Keras & TensorFlow.

Módulo III – A Biblioteca Pandas

Nesse modulo iremos tratar sobre a biblioteca Pandas, ela possui estruturas e operações de manipulação de dados, projetadas para agilizar a análise de dados em Python. Frequentemente ela é utilizada com outras bibliotecas como Numpy, Scipy, Matplotlib e Scikit Learn. A maneira de se trabalhar se assemelha muito ao Numpy, inclusive algumas operações foram inspiradas nessa biblioteca. A principal diferença da biblioteca Pandas para a biblioteca Numpy é que a Pandas estará mais ligado a dados tabulares e heterogêneos, enquanto a Numpy será mais adequada em realizar operações com dados lineares e homogêneos.

VISÃO GERAL

Pandas é uma biblioteca do Python que permite manipular grandes quantidades de dados de maneira prática e ágil. Podemos carregar planilhas e tabelas, operando da mesma forma que um banco de dados SQL, é possível realizar todo tipo de análise estatística, inclusive a criação de gráficos a partir da biblioteca gera belos gráficos.

Estrutura de Dados da Biblioteca Pandas

A biblioteca possui duas principais estruturas de dados, no entanto apenas uma é mais utilizada pelo fato de representar muito bem internamente uma estrutura formal de armazenamento, uma tabela. Assim como Numpy, Pandas também adota um apelido para facilitar a identificação e utilização da biblioteca. Essa padronização é adotada por toda a comunidade de Ciência de Dados e, portanto, facilitar a compreensão de scripts de terceiros.

```
import numpy as np
import pandas as pd
```

Portanto, sempre que ver instruções iniciadas por np, associe a biblioteca Numpy, quando tiver instruções iniciadas por pd, associe a biblioteca Pandas.

A principal estrutura da biblioteca se chama Data Frame (Quadro de Dados em português). A estrutura secundária que não iremos tratar aqui, se chama Series. Essa estrutura traz uma representação indexada de um conjunto de dados homogêneo que pode ser trabalhado assim como um ndarray da biblioteca Numpy, a pesar de a biblioteca Pandas trazer outras funcionalidades que você pode encontrar em [link](#).

Data Frame é uma estrutura indexada com dados de diversos tipos sendo permitidos de serem armazenados e manipulados. A criação de um *Data Frame* remete a uma estrutura como um dicionário, porém com mais possibilidades de manipulação. Além da possibilidade de criação a partir de um dicionário, o pandas possui métodos capazes de realizar a leitura e o carregamento de dados a partir de diversas fontes distintas como: arquivo csv, planilhas, arquivos de textos, conexão direta com banco de dados, entre outros. Vejamos a criação de um simples *Data Frame* a partir de um dicionário de dados:

```
dados = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002, 2003],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
}
```

```
populacao = pd.DataFrame(dados)
populacao
```

Podemos analisar as colunas existentes através do atributo `columns` do *Data Frame* e os valores através do atributo `values`:

```
populacao.columns
populacao.values
```

Caso queira modificar a ordem em que as colunas serão exibidas, você pode indicar a ordem de leitura com atributo `columns`:

```
populacao = pd.DataFrame(dados, columns=['year', 'state', 'pop'])
populacao
```

Se em alguma análise for de interesse possuir índices de linha mais amigáveis do que números, você pode indicar através de uma lista os índices que deseja ter:

```
populacao_novo = pd.DataFrame(dados, columns=['year', 'state', 'pop', 'debt'],
index=['I', 'II', 'III', 'IV', 'V', 'VI'])
```

A exibição de um atributo específico pode ser feito de duas maneiras:

```
populacao_novo['state']
populacao_novo.year
```

Valores podem ser facilmente alterados utilizando a notação de acima associando o valor que se deseja armazenar:

```
populacao_novo['debt'] = 16.5
populacao_novo
populacao_novo['debt'] = np.arange(6.)
populacao_novo
```

A criação de um novo atributo também é facilitada com a seguinte instrução indicando um atributo inexistente:

```
populacao_novo['eastern'] = populacao_novo.state == 'Ohio'
populacao_novo
```

Em alguns momentos pode ser necessário realizar a transposição de um *Data Frame* para melhor representar os dados carregados:

```
populacao_anual = pd.DataFrame({'Nevada': {2001: 2.4, 2002: 2.9}, 'Ohio': {2000:
1.5, 2001: 1.7, 2002: 3.6}})
populacao_anual
populacao_anual.T
```

Quando a origem dos dados possui um número de registros muito grande, para não demorar o carregamento dos dados para exibição, é interessante utilizar métodos como `head()` e

tail()). Esses métodos permitem inspecionar os dados contidos no *Data Frame* de maneira a exibir apenas os primeiros e os últimos registros. Por padrão eles exibem 5 (cinco) registros. Se tiver interesse de exibir mais, basta incluir o parâmetro *n* associado ao número de registros que deseja exibir. Vejamos

```
iris = pd.read_csv("./DataSets/iris.csv")
iris.head()
iris.tail()
iris.head(n=10)
iris.tail(n=15)
```

Reindexação

A reindexação é um método que precisa ter muito cuidado ao utilizá-la, devido a sua capacidade de criar valores indicativos de ausência caso algum novo índice não for preenchido previamente. O método *reindex* possui dois atributos que podem auxiliar no preenchimento de valores ausentes, os atributos *method* que indica um método de preenchimento e *fill_value* que indica um valor fixo a ser utilizado. Vejamos alguns exemplos:

```
idades = pd.DataFrame(np.arange(9).reshape((3,3)), index=['a', 'c', 'd'],
columns=['Ohio', 'Texas', 'California'])
idades
idades_novo = idades.reindex(['a', 'b', 'c', 'd']) #Reindexando as linhas
idades_novo
estados = ['Texas', 'Utah', 'California']
idades.reindex(columns=estados)
```

Preenchendo valores com indicativo de ausência:

```
index = ['Firefox', 'Chrome', 'Safari', 'IE10', 'Konqueror']
navegadores = pd.DataFrame({'http_status': [200, 200, 404, 404, 301],
                             'response time': [0.04, 0.02, 0.07, 0.08, 1.0]},
                             index=index)
navegadores
new_index = ['Safari', 'Iceweasel', 'Comodo Dragon', 'IE10', 'Chrome']
navegadores.reindex(new_index, fill_value=0)
```

Utilizando os métodos de preenchimento para frente (*ffill*) e para trás (*bfill*):

```
date_index = pd.date_range('1/1/2010', periods=6, freq='D')
precos_perodo = pd.DataFrame({"prices": [100, 101, np.nan, 100, 89, 88]},
                              index=date_index)
precos_perodo
```

Assim serão gerados muitos valores não preenchidos que precisam ser tratados

```
date_index2 = pd.date_range('12/29/2009', periods=10, freq='D')
precos_perodo.reindex(date_index2)
```

O ideal para esse caso é utilizar o preenchimento para trás:

```
precos_perodo.reindex(date_index2, method='bfill')
```

Mas você pode optar por utilizar o preenchimento para frente:

```
precos_perodo.reindex(date_index2, method='ffill')
```

Indexação, Seleção e Filtragem

Em Data Frame indexação permite recuperar uma ou mais colunas, ainda é possível selecionar apenas trechos da estrutura para manipulação, até mesmo filtrações condicionais ou mais avançadas são possíveis de serem feitas. Vejamos:

```
titanic = pd.read_csv("./DataSets/titanic_treino.csv")
titanic.head()
titanic.loc[1] #Dessa forma estaremos recuperando os dados do primeiro passageiro
titanic[:2] #Recupera todos os dados dos dois primeiros passageiros
titanic[['Sex', 'Age', 'Survived']] #Recupera todos os dados das colunas na ordem
em que aparecem
titanic[titanic['Pclass'] == 3] #Recupera todos os passageiros da terceira classe
titanic.loc[[1,2], ['Name', 'Sex', 'Age']] #Recupera o segundo e o terceiro
passageiros com os atributos listados
titanic.loc[10:20] #Recupera do décimo ao vigésimo passageiro
titanic.loc[10:30:2] #Recupera do décimo ao trigésimo passageiro intercalando de
dois em dois
```

Além do método loc e da sintaxe mostrada acima, podemos utilizar o método query que tem uma sintaxe semelhante as queries do SQL, utilizando operadores lógicos AND (&) e OR (|) e operadores relacionais (==, !=, >, <, >= e <=). Vejamos alguns exemplos:

```
titanic.query('Age > 20').head() #Recupera os passageiros com idade maior que 20
titanic.query('Age > 20 & Sex=="male"').head(10) #Recupera os passageiros com
idade maior que 20 e do sexo masculino
df.query('Age > 20 | Sex=="male"').head(10) #Recupera os passageiros com idade
maior que 20 ou do sexo masculino
titanic.query('Embarked in ["C","Q"]', inplace=True) #Recupera alterando o Data
Frame, os passageiros que embarcaram em C e Q
titanic
```

Ordenação

Com Data Frame podemos ordenar quaisquer dos eixos existentes, isso permite diversas possibilidades de manuseio dos dados para análises exploratórias.

```
frame = pd.DataFrame(np.arange(8).reshape((2, 4)), index=['three', 'one'],
columns=['d', 'a', 'b', 'c'])
frame.sort_index(axis=0)
frame.sort_index(axis=1)
```

Por padrão a ordenação será em ordem crescente, para ordem decrescente basta alterar o atributo ascending para False:

```
frame.sort_index(axis=1, ascending=False)
```

Há ainda a possibilidade de ordenação pelo valor, possibilidade que decida por qual atributo deseja manter a ordem dos seus dados:

```
frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})  
frame
```

```
frame.sort_values(by=['b'])  
frame.sort_values(by=['a', 'b'])
```

Estatísticas Descritivas

Assim a biblioteca Numpy, a biblioteca Pandas possui métodos matemáticos e estatísticos implementados, tornando fácil a análise exploratória dos dados pertinentes ao conjunto de informações.

```
titanic['Age'].mean() #Calcula a média das idades dos passageiros  
titanic[["Age", "Fare"]].median() #Retorna a mediana da idade e da taxa paga
```

Muitas das vezes será necessário ter conhecimento dos tipos de dados de cada atributos para que operações específicas possam ser realizadas, para isso basta invocar o método info() que recupera os tipos de dados e a quantidade de valores preenchidos por atributo:

```
titanic.info()
```

Outro método que pode ajudar trazendo algumas estatísticas é o método describe() que traz com ele informações como contagem de valores, média, desvio-padrão, valor mínimo, 1º quartil, 2º quartil, 3º quartil, mínimo e máximo:

```
titanic.describe()  
titanic[["Age", "Fare"]].describe()
```

Ainda há possibilidades como aplicar métodos estatísticos agrupando campos específicos, como:

```
titanic.groupby(by='Sex').size() #Conta a quantidade de registros por sexo  
titanic.groupby(by='Sex')['Age'].mean() #Calcula a média de idades por sexo
```

Com o método agg é possível aplicar diversos métodos ao mesmo tempo:

```
titanic.groupby(['Sex', 'Survived']).agg({'Age': np.mean, 'PassengerId': np.size})  
#Calcula a média e contabiliza os passageiros das idades por sexo e situação de sobrevivente ou não
```


Ainda existem muitas possibilidades com a biblioteca Pandas, no módulo seguinte iremos tratar de carregamento e armazenamento de dados, leitura de dados no formato texto, interação com API Web, interação com Banco de Dados e preparação de dados para utilização em pesquisas futuras.

Módulo IV – A Biblioteca Pandas – Arquivos e Preparação de Dados

Agora que sabemos como utilizar o Data Frame da biblioteca Pandas, iremos avançar os estudos da biblioteca aprendendo outras utilidades como acesso arquivos de dados, acesso a banco de dados e tratamento de dados.

Carregamento de Arquivos

A biblioteca Pandas possui diversos métodos capazes de realizar leitura de dados tabulares de diversas fontes e transformá-los em Data Frame. Na Figura 14 podemos ver que temos leitura para arquivos delimitados, arquivos tabulados, planilhas do Excel, HDF, arquivos HTML, arquivos JSON e consulta direta ao banco de dados.

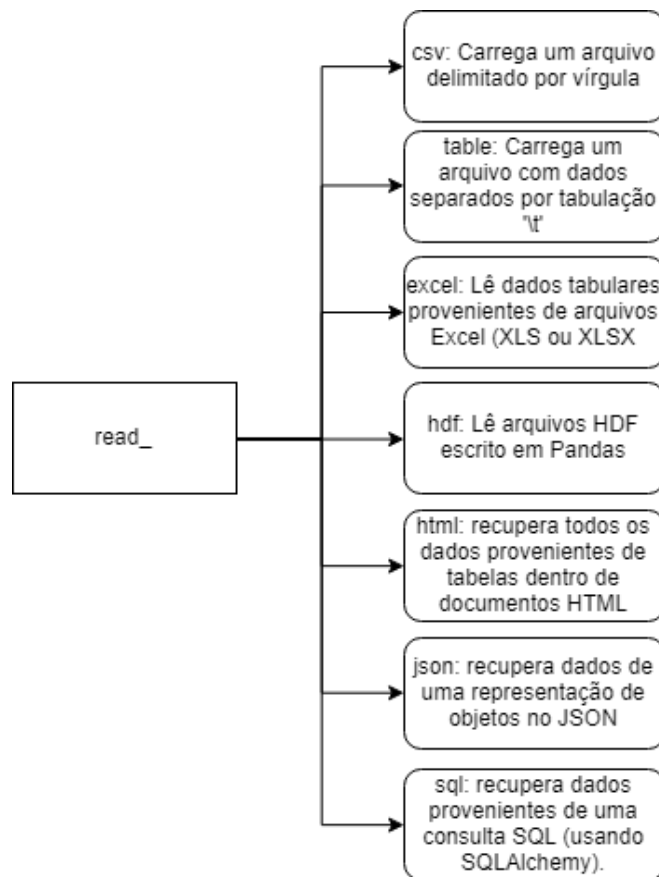


Figura 14: Métodos de Leitura de Dados Externos. Fonte: O Autor

Arquivos CSV

Arquivos CSV são documentos padronizados que possuem os atributos de cada registro separados, geralmente, por vírgula. Por isso o nome CSV de *Comma Separated Value*, valores separados por vírgula em tradução própria. Porém, é possível que se encontre pelos repositórios de Internet arquivos com extensão CSV que possuem outros separadores de atributos, como ponto-e-vírgula (;) ou tabulação (\t). Saber como seus dados estão separados é trivial para realizar a leitura correta do arquivo e poder utilizá-lo carregado em um Data Frame.

O método da biblioteca Pandas utilizado para leitura de arquivos CSV é o `read_csv()`, por sua vez ele possui diversos atributos que podem ser utilizados para facilitar a recuperação das informações armazenadas no arquivo. Os atributos mais utilizados são:

- (1) `filepath_or_buffer`: indica o caminho origem do arquivo no computador;
- (2) `sep/delimiter`: indica qual é o caractere que está sendo utilizado para separação dos dados;
- (3) `header`: indica onde está o cabeçalho dos dados caso existe;
- (4) `names`: indica uma lista de nomes para serem utilizados pelas colunas;
- (5) `index_col`: indica qual coluna poderá ser utilizada como índice do Data Frame;
- (6) `usecols`: indica quais colunas serão utilizadas;
- (7) `skiprows`: indica o número das linhas que a biblioteca deverá deixar de ler ao carregar os dados
- (8) `nrows`: indica o número de linhas que serão lidas do arquivo;
- (9) `encoding`: indica qual a codificação o arquivo está utilizando e portanto realizar a leitura correta dos caracteres; e
- (10) E diversos outros que podem ser encontrados na [documentação oficial](#).

Percebam que, exceto o primeiro atributo, todo o restante é opcional que internamente possuem valores padrão a ser utilizado. Cada caso irá te levar a utilização de um ou vários atributos a mais para a leitura dos arquivos.

Já vimos no módulo anterior algumas leituras de arquivos CSV, iremos exemplificar mais algumas formas de utilização do método de leitura:

```
#Arquivo sem cabeçalho
dados1 = pd.read_csv('exemplo1.csv', header=None)
#Leitura de arquivo .csv com indicação de nomes para as colunas e qual será
utilizada como índice dos dados
dados1 = pd.read_csv('exemplo1.csv', names=['nome', 'telefone', 'email',
'dataNascimento'], index_col='dataNascimento')
#Leitura de arquivo .csv com dados separados por tabulação
dados2 = pd.read_csv('exemplo2.csv', sep='\t')
# Leitura de arquivo .csv com indicação de quais colunas serão lidas do arquivo
dados2 = pd.read_csv('exemplo2.csv', usecols=[1,3,4,5,8,10])
dados2 = pd.read_csv('exemplo2.csv', usecols=['valor1', 'valor3', 'valor8',
'valorTotal'])
# Leitura de arquivo .csv com indicação de quais linhas serão deixadas de ler
dados3 = pd.read_csv('exemplo3.csv', skiprows=[0, 2, 3])
#Leitura de arquivo .csv com indicação do número de linhas a serem lidas
dados3 = pd.read_csv('exemplo3.csv', nrows=18)
```

Muitas das vezes os projetos de análise de dados envolverão a recuperação dos dados, a higienização dos dados e aplicação de algum estudo sobre eles. Pode ocorrer de surgir à necessidade de armazenar os dados alterados em um novo arquivo que possa ser utilizado futuramente, para isso a biblioteca possui um método atrelado ao Data Frame que nos permite salvar em um novo arquivo do tipo CSV, muito simples de ser utilizado. Basta indicar o local que o arquivo será salvo e um nome para ele:

```
dados3.to_csv('caminhoCompleto/diretorio/dados3.csv')
```

Você poderá conferir se o arquivo foi gerado corretamente realizando a leitura do mesmo, conforme vimos nos exemplos anteriores ou utilizando algum outro atributo ainda não utilizado. Experimente!

Arquivo JSON

Um arquivo JSON é um formato livre para representação de objetos e formatos hierárquicos, leve e intercambiável entre ambientes sistêmicos através da Internet. A formatação desse tipo de arquivo se assemelha ao dicionário existente no Python, com a orientação por pares de chaves ({, }), utilização de chave e valor para indicar os atributos e os dados armazenados. A natureza desse arquivo é textual, assim como o CSV. Primeiro criaremos um arquivo JSON e em seguida iremos ler o seu conteúdo:

```
dados = pd.DataFrame([{"a": 1, "b": 2, "c": 3},
                      {"a": 4, "b": 5, "c": 6},
                      {"a": 7, "b": 8, "c": 9}])
dados
dados.to_json('dados.json')
dadosJSON = pd.read_json('dados.json')
dadosJSON
```

Um cuidado que é necessário ter para ler esse tipo de arquivo é conhecer sua hierarquia para aplicar corretamente os parâmetros necessários para sua leitura, geralmente espera-se que o arquivo possua diversos registros por linhas que se repetem ao longo do arquivo com dados distintos. Entretanto, você pode inspecioná-lo antes de carregar para uma estrutura do tipo Data Frame. Para isso você precisará da biblioteca json. No exemplo a seguir, a variável obj é atribuída um texto no formato json. Em seguida o método loads é responsável por carregar esse conteúdo para a variável resultado. Por fim, após exibir o conteúdo armazenado em resultado. O dicionário é convertido em um json. Apesar desse processo

estar sendo feito todo no Script, você pode considerar a leitura de um arquivo e essas conversões irão emitir erros caso a estrutura esteja com algo incorreto.:

```
import json
obj = """
{"name": "Wes",
 "places_lived": ["United States", "Spain", "Germany"],
 "pet": null,
 "siblings": [{"name": "Scott", "age": 30, "pets": ["Zeus", "Zuko"]},
               {"name": "Katie", "age": 38,
                "pets": ["Sixes", "Stache", "Cisco"]}]}
"""
resultado = json.loads(obj)
resultado
arquivo_json = json.dumps(resultado)
arquivo_json
```

Arquivo XML

O arquivo XML é um arquivo de marcação extensível com tags personalizáveis que permitem organizar um conjunto de dados de entidades através de uma hierarquia. É um tipo de arquivo comumente utilizado para comunicação de dados distribuídos com o protocolo SOAP. A estrutura se comporta de forma parecida ao JSON, porém o JSON utiliza o protocolo HTTP para comunicação distribuída, se fazendo utilizar dos métodos HTTP de requisição e resposta. Um exemplo clássico no Brasil de utilização do XML é a geração de Nota Fiscal Eletrônica, que possui um arquivo complexo com dezenas de tags que discriminam os elementos de uma nota fiscal: cabeçalho, dados do cliente, dados do fornecedor, dados da transportadora, dados dos produtos, dados sobre impostos e dados diversos.

No Python há diversas bibliotecas existentes que permite realizar a leitura e conversão dos arquivos XML para um Data Frame, você poderá encontrar exemplos espalhados pela Internet. Aqui irei demonstrar uma, vejamos:

O conteúdo do arquivo estará disposto da seguinte maneira:

```
<data>
  <student name="John">
    <email>john@mail.com</email>
    <grade>A</grade>
    <age>16</age>
  </student>
  <student name="Alice">
    <email>alice@mail.com</email>
    <grade>B</grade>
    <age>17</age>
```

```

</student>
<student name="Bob">
  <email>bob@mail.com</email>
  <grade>C</grade>
  <age>16</age>
</student>
<student name="Hannah">
  <email>hannah@mail.com</email>
  <grade>A</grade>
  <age>17</age>
</student>
</data>

```

É um arquivo com dados de estudantes que será lido e convertido em um Data Frame. Utilizaremos a biblioteca XML, a partir dela o objeto ElementTree que permite converter a árvore de elementos em uma estrutura rastreável. Primeiro realiza-se a leitura do arquivo, em seguida recupera-se a raiz do arquivo, a tag <data>. Em seguida iremos navegar por cada par de tags <student> para recuperar os dados, incluindo em um dicionário, para então criar o Data Frame. Para deixar o processo mais dinâmico, iremos criar um método que irá receber o caminho do arquivo e a lista de atributos que deveremos utilizar para criação do Data Frame:

```

import xml.etree.ElementTree as et
def parse_XML(xml_file, df_cols):
    xtree = et.parse(xml_file)
    xroot = xtree.getroot()
    rows = []

    for node in xroot:
        res = []
        res.append(node.attrib.get(df_cols[0]))
        for el in df_cols[1:]:
            if node is not None and node.find(el) is not None:
                res.append(node.find(el).text)
            else:
                res.append(None)
        rows.append({df_cols[i]: res[i]
                     for i, _ in enumerate(df_cols)})

    out_df = pd.DataFrame(rows, columns=df_cols)

    return out_df

estudantes = parse_XML("students.xml", ["name", "email", "grade", "age"])
estudantes

```

Perceba que da forma que está construída, você conseguirá apenas descer a um nível de dados. Se tiverem mais níveis, o método precisa ser ajustado.

Arquivo HTML

Para arquivos HTML a biblioteca pandas possui um método capaz de compreender em uma página de Internet uma tabela existente pelos tags de tabela. Para outras tags, você precisará aplicar [Web Scraping](#). Outro detalhe importante do `read_html` é que ele irá retornar para o objeto uma lista com um único elemento na primeira posição, esse elemento é o Data Frame. Vejamos um simples exemplo:

```
url = ("https://raw.githubusercontent.com/pandas-  
dev/pandas/master/pandas/tests/io/data/html/spam.html")  
dfs = pd.read_html(url)  
dfs  
type(dfs)  
dfs[0]  
type(dfs[0])
```

Essa uma forma simples que pode auxiliar em problemas em que as páginas tenham bem definidas uma tabela existente.

Banco de Dados

Para leitura de dados a partir de um banco de dados estruturado, o `read_sql` auxilia nessa atividade onde você pode realizar a leitura de uma tabela específica ou de uma consulta que gera os dados necessários. A biblioteca Pandas trabalha junto da biblioteca SQLALCHEMY para realizar a consulta com diversos bancos e permitir consultas SQL. Vejamos um exemplo simples utilizando o SQLITE.

Primeiro veremos como realizar uma conexão e uma consulta. A biblioteca para conexão ao SQLITE é a `sqlite3`, veja:

```
import sqlite3  
conexao = sqlite3.connect("./DataSets/flights.db")  
#É necessário cria um objeto chamado cursor para poder navegar pelos registros  
cursor = conexao.cursor()  
consulta = cursor.execute("select * from airlines")  
resultados = cursor.fetchall()  
print(resultados)  
#É importante fechar as conexões para não corromper os dados.  
cursor.close()  
conexao.close()
```

Certo, agora vamos ver como realizar a consulta e converter diretamente em um Data Frame:

```
import sqlite3  
conexao = sqlite3.connect("./DataSets/flights.db")
```



```
dados = pd.read_sql("select * from airlines", conexao)
dados.head()
cursor.close()
conexao.close()
```

Com esses passos simples e a biblioteca correta para conectar ao SGBD, você conseguirá recuperar os dados de qualquer conjunto de tabelas, com consultas simples e complexas.

A biblioteca ainda possui diversas possibilidades de recuperação de dados de fontes diversas, você pode verificar na [documentação oficial](#).

Preparação de Dados

Oitenta por cento do tempo o cientista passa realizando transformação e preparação de dados para poder seguir com a pesquisa. Dados ausentes podem ser removidos ou substituídos por valores significantes, dados duplicados podem ser removidos, textos podem ser transformados em maiúsculo ou minúsculo e valores podem ser filtrados e removidos. A preparação não se resume a isso, outros problemas podem surgir e serem tratados de acordo com o que ocorrer, mas no geral essas técnicas atendem diversos problemas que ocorrem em diversos conjuntos de dados.

Remoção de Registros Duplicados

Algumas vezes, por falha na leitura e geração dos dados, os registros podem duplicar e isso não é interessante para análises de dados. Os registros precisam ser únicos para representar a entidade do mundo real a qual os dados estão associados. Vamos criar um Data Frame com dados duplicados e em seguida iremos utilizar o método `drop_duplicates` para remover as linhas que possuem os mesmos atributos:

```
dados = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'], 'k2': [1, 1, 2, 3, 3, 4, 4]})
dados
```

Podemos remover as linhas duplicadas analisando todo o registro de uma única vez, ou decidir remover as linhas duplicadas analisando apenas algumas colunas do filtro.

```
dados.drop_duplicates()
#ou
dados.drop_duplicates(['k1'])
```

Remoção Registros com Dados Ausentes

No Python a ausência de informações pode ser representada de diversas formas, as mais comuns são: NaN, NaT e None. Independente da forma a qual será representada, a biblioteca Pandas remove os registros com dados ausentes utilizando o método `dropna()`. Por padrão ele irá remover todas as linhas que possuam em algum atributo (coluna) um dado ausente. Você pode definir as colunas que deseja analisar, o número de registros mínimos que não possuem dados ausentes para manter e remover as colunas que possuem algum dado ausente.

```
df = pd.DataFrame({"name": ['Alfred', 'Batman', 'Catwoman'],  
                  "toy": [np.nan, 'Batmobile', 'Bullwhip'],  
                  "born": [pd.NaT, pd.Timestamp("1940-04-25"),  
                           pd.NaT]})
```

O método `dropna` do Data Frame irá retornar uma nova estrutura com a remoção conforme solicitado. Possivelmente você irá querer que a alteração seja aplicada, para tal informe no atributo `inplace` o parâmetro `True`.

```
#Removendo todas as linhas com ausência de dados  
df.dropna()  
#Removendo as linhas onde todos os atributos estão ausentes  
df.dropna(how='all')  
#Removendo as colunas que possuem algum dado ausente  
df.dropna(axis='column')  
#Mantendo linhas que possuem ao menos um número mínimo de valores presentes  
df.dropna(thresh=2)  
#Definindo um conjunto de atributos a serem verificados  
df.dropna(subset=['name', 'toy'])  
#Aplicando a alteração diretamente no Data Frame  
df.dropna(inplace=True)
```

Esses atributos podem ser combinados para aplicar ao mesmo tempo e obter o resultado esperado.

Preenchendo Registros com Dados Ausentes

Da mesma forma que remover pode ser a solução para os dados ausentes, você pode preferir preenchê-los com algum valor existente, uma fórmula ou um valor padrão que dê significado aos seus dados. O método `fillna()` da biblioteca Pandas permite preencher os dados ausentes. Vamos criar um Data Frame e em seguida preencher os dados:

```
df = pd.DataFrame([[np.nan, 2, np.nan, 0],  
                  [3, 4, np.nan, 1],  
                  [np.nan, np.nan, np.nan, 5],  
                  [np.nan, 3, np.nan, 4]])
```

```

columns=list('ABCD'))
df
#Preenchendo todos os valores com 0 (zero)
df.fillna(0)
#Propagando valores não nulos para frente
df.fillna(method='ffill')
#Propagando valores não nulos para trás
df.fillna(method='bfill')
#Preenchendo valores de acordo com o nome dos atributos
df.fillna(value= {'A': 0, 'B': 1, 'C': 2, 'D': 3})

```

Substituindo Valores

É possível ao invés de um valor utilizar uma função matemática para preenchimento, como a média, a soma, o desvio-padrão, entre outros. Na disciplina de estatística há estudos que podem auxiliar na seleção de como preencher valores ausentes de acordo com os dados existentes. Então você pode utilizar o método `replace()` para modificar os valores existentes.

```

data = pd.Series([1,2,-99,4,5,-99,7,8,-99])
data
data.replace(-99, np.nan)

```

Ao invés de preencher com um valor ausente, você pode querer aplicar a média de valores absolutos:

```

data.replace(-99, np.mean(np.abs(data)))

```

Conversão de Tipos de Dados

É muito importante ter dados com os tipos de corretos, isso irá aumentar o desempenho da execução dos seus scripts. Para converter dados, podemos usar o método `astype()` em uma coluna específica. Também tenha em mente o tipo 'category' - ele reduz o tamanho de um Data Frame e torna os cálculos mais rápidos. Podemos converter para qualquer valor que possa ser usado como categoria - dias da semana, gênero, abreviações de continentes - dependem de um contexto. O atributo `errors='coerce'` fará com que possíveis erros sejam deixados de lado.

```

#Convertendo para string
df['column1'] = df['column1'].astype(str)
#Convertendo em categoria
df['column1'] = df['column1'].astype('category')
#Convertendo em tipo numérico
df['column1'] = pd.to_numeric(df['column1'], errors='coerce')
#Convertendo em data
df['column1'] = pd.to_datetime(df['column1'], errors='coerce')

```

Esses são os primeiros passos para se ter um conjunto de dados prontos para utilização em uma pesquisas de Ciência de Dados e Aprendizado de Máquina, ainda há algumas

possibilidades de tratamento, além de tratamentos específicos que iremos tratar na Engenharia de Atributos no [Módulo de Machine Learning](#).

Módulo V – Visualização de Dados

Gerar informações visuais é uma das tarefas mais importantes na análise exploratória de dados. Muitas das vezes ela será o objetivo final da pesquisa em uma visualização interativa na Web. Em outros momentos ela será um suporte a compreensão dos dados, identificação de discrepância ou até mesmo transformação dos dados para que ideias sejam geradas. Nesse módulo iremos explorar as bibliotecas Matplotlib e sua subjacente Seaborn para criação de visualizações.

Introdução a Biblioteca Matplotlib

A melhor maneira de se acompanhar o desenvolvimento de visualizações dinamicamente é utilizando Jupyter Notebook, para isso basta importar a biblioteca apelidando ela como plt (padronização de utilização do método de plotagem) e incluindo o magic command adequado para visualização. Magics commands, são marcações incluídas no IPython e no Jupyter Notebook que permitem execução de códigos automatizados, você pode consultá-los [nesse link](#). No caso da visualização, ele irá gerar as imagens dos gráficos que queremos, logo estamos interessados no magic command `%matplotlib [gui]` onde `[gui]` será substituído pelo meio utilizado para visualizar. Geralmente utiliza-se `inline` e `notebook`, porém, nada lhe impede de utilizar as outras possibilidades: `osx`, `qt4`, `qt5`, `gtk3`, `notebook`, `wx`, `qt`, `nbagg`, `gtk`, `tk`, `inline`.

Antes de começar a gerar visualizações, é importante compreender a estrutura de um gráfico, a documentação oficial de Matplotlib traz essa imagem que retrata bem cada item presente em um gráfico, vejamos:

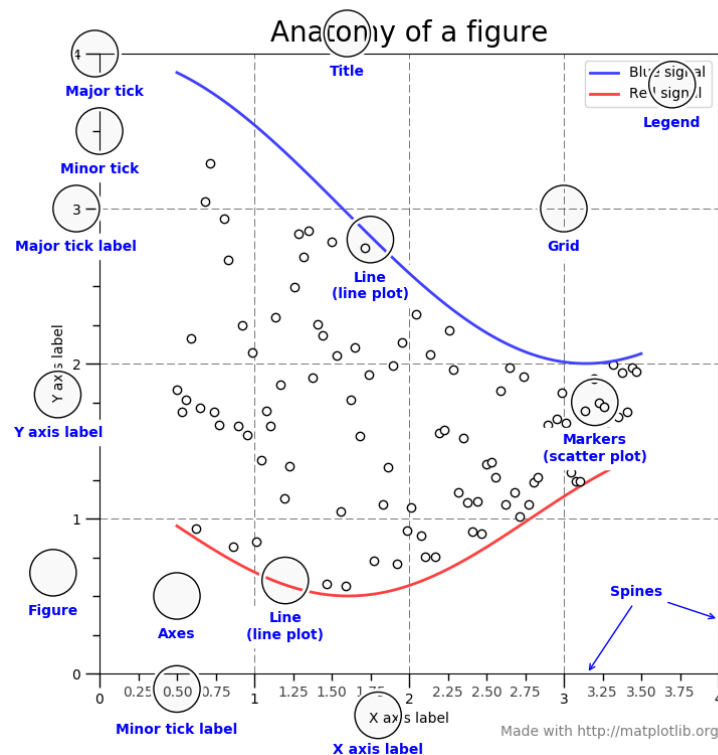


Figura 15: Anatomia de um Gráfico. Fonte: https://matplotlib.org/2.0.2/faq/usage_faq.html

Na Figura 15 podemos identificar os rótulos de eixo, legenda, marcação numérica, marcadores de dados, título, bordas. E isso lhe auxiliará na compreensão dos exemplos que veremos.

Vamos iniciar importando a biblioteca para o script e em seguida incluiremos o magic command. O objeto da biblioteca que será utilizado é o PyPlot, uma versão Python para gerar visualizações tão robustas quanto o MATLAB®:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

Criaremos um gráfico de linha com um ndarray gerado pela biblioteca Numpy em uma variação de 0 (zero) a 9 (nove).

```
import numpy as np
dados = np.arange(10)
plt.plot(dados)
plt.plot(dados, dados**2)
```

Embora internamente Pandas possua métodos de plotagem, para personalização dos gráficos é necessário que se conheça um pouco sobre Matplotlib. Aprofundar sobre Matplotlib renderia um curso inteiro sobre ela, então iremos focar no que é mais utilizado em projetos de Data Science.

O método plot é bastante robusto e suporta diversos atributos de personalização ([vide documentação](#)), vamos começar a explorar com o que há de mais simples, um gráfico em linha 2D com informações categóricas no eixo x e numéricas no eixo y. Apenas por inserir as informações que deseja visualizar, você já obterá um gráfico descente:

```
meses = ['Janeiro', 'Fevereiro', 'Março', 'Abril', 'Maio', 'Junho']
valores = [105235, 107697, 110256, 109236, 108859, 109986]
plt.plot(meses, valores)
```

Podemos alterar a visão do gráfico aplicando limites superior e inferior através do método ylim(valorMinimo, valorMaximo) para melhor adequar os dados na visualização:

```
plt.ylim(100000, 120000)
plt.plot(meses, valores)
```

Dessa forma, centralizaremos mais os dados para perceber o comportamento dos dados. Para dar mais características que permitam o entendimento do gráfico, iremos adicionar um título (title) ao gráfico, ao eixo x (xlabel) e ao eixo y (ylabel):

```
plt.ylim(100000, 120000)
plt.title('Faturamento no primeiro semestre de 2017')
plt.xlabel('Meses')
plt.ylabel('Faturamento em R$')
plt.plot(meses, valores)
```

Vamos experimentar algumas cores e formas com o plot:

```
dados1 = [10,5,2,4,6,8]
dados2 = [ 1,2,4,8,7,4]
x = 10 * np.array(range(len(dados1)))

plt.plot( x, dados1, 'go') # green bolinha
plt.plot( x, dados1, 'k:', color='orange') # linha pontilhada orange

plt.plot( x, dados2, 'r^') # red triangulo
plt.plot( x, dados2, 'k--', color='blue') # linha tracejada azul

plt.axis([-10, 60, 0, 11]) #Os limites da caixa
plt.title("Um Título Elegante")

plt.grid(True) #Exibição do grid de fundo
plt.xlabel("eixo horizontal(x)")
plt.ylabel("eixo vertical(y)")
plt.show()
```

Histogramas são gráficos que permitem visualizar o comportamento de determinado atributo de um conjunto de dados de acordo com a frequência de cada elemento, a partir dele é possível modelar a famosa curva de Gauss (curva normal, se preferir) e calcular probabilidades de eventos ocorrerem, calcular intervalos de confiança, entre outras coisas que a Estatística pode nos beneficiar. O método para histogramas é o `hist()`, você indica a quantidade de barras com o atributo `bins`. Vejamos um exemplo com dados aleatórios:

```
x = np.random.randn(10000)
plt.hist(x, bins=100, color='#FFC0CB')
```

Um gráfico de barras é suficiente bom para exibir categorias e número relacionados, o método `bar()` atribuindo as categorias e os valores gera essa visualização para você:

```
grupos = ['Produto A', 'Produto B', 'Produto C']
valores = [1, 10, 100]
plt.bar(grupos, valores)
```

Talvez o seu projeto necessite de gráficos de barras distintas na mesma plotagem, então você poderia dividir o quadro (figure) em dois com o método `subplot(número de linhas, número de colunas)`, para exibir as relações de atributos e valores. A utilização do método `show()` irá limpar a área de impressão, mantendo apenas as figuras geradas pelos dados. Os objetos `ax1` e `ax2` irão te direcionar para cada um dos gráficos que irá gerar. O atributo `figsize(linhas, colunas)` limitará o tamanho da imagem produzida. Vejamos:


```
# Define as configurações dos plots
# Cada plot terá o mesmo tamanho de figuras (10,5)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))

# Dados para cada subplot
ax1.bar([1,2,3],[3,4,5])
ax2.barh([0.5,1,2.5],[0,1,2])#barras horizontais

ax1.set(title="Gráfico de Barras Verticais", xlabel="Eixo X", ylabel=" Eixo Y")
ax2.set(title="Gráfico de Barras Horizontais", xlabel="Eixo X", ylabel="Eixo Y")

plt.show()
```

O número de plotagens em uma mesma figura pode ser incrementado e caracterizados conforme os projetos exigirem. Você pode saber um pouco mais no [link](#).

Se você está habituado com as cores em RGB, poderia alterar o projeto anterior dando-lhes cores diferentes incluindo o atributo color com a numeração em hexadecimal das cores desejadas, por exemplo: #D2691E para chocolate e #0000FF para azul Royal.

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))
ax1.bar([1,2,3],[3,4,5], color='#D2691E')
ax2.barh([0.5,1,2.5],[0,1,2], color='#0000FF')
ax1.set(title=" Barras Verticais", xlabel="Eixo X", ylabel=" Eixo Y")
ax2.set(title=" Barras Horizontais", xlabel="Eixo X", ylabel="Eixo Y")
```

Gráficos de pizza (pie) são bons para exibir à proporção que cada categoria possui diante do todo:

```
vendas = [3000, 2300, 1000, 500]
labels = ['E-commerce', 'Loja Física', 'e-mail', 'Marketplace']

plt.pie(vendas, labels=labels)
plt.legend(labels, loc=3)#Adiciona as legendas na posição (loc) indicada
plt.show()
```

Muitas vezes você estará interessado em gerar gráficos que permita analisar o comportamento de atributos, analisando os valores máximos, mínimos, mediana, quartis e outliers. Esse gráfico é o boxplot, onde você poderá analisar uma única variável ou mais de uma para comparação e análise de comportamento:

```
import pandas as pd
df = pd.read_csv("./DataSets/Clientes_Cartao_Credito.csv", sep=";")

plt.title("Limite de Créditos x Gênero")
plt.boxplot([df[df.Gender == 'M'].Credit_Limit,
             df[df.Gender == 'F'].Credit_Limit],
            labels=['Masculino', 'Feminino'])
plt.show()
```

Nesse boxplot podemos analisar os limites de créditos concedidos a dois grupos de pessoas, de gêneros distintos, para poder analisar a homogeneidade dos dados e as dispersões que porventura surjam. Caso seja interessante analisar apenas um dos grupos, basta modificar a lista de valores:

```
plt.title("Limite de Créditos x Gênero")
plt.boxplot([df[df.Gender == 'F'].Credit_Limit],
            labels=['Feminino'], flierprops=dict(markerfacecolor='g', marker='D'))
plt.show()
```

A correlação dos atributos de um conjunto de dados indica se ambos se comportam da mesma forma à proporção que mudam os valores, pode exibir essa relação através de um gráfico de calor. Nesse exemplo, quanto maior for a correlação positiva entre os atributos, um tom de azul mais forte será exibido, caso seja uma correlação negativa, tons de vermelho serão exibidos. A não correlação é indicada por zero, portanto, será exibida por um tom mais próximo de branco. Para calcular a correlação dos valores, utilizaremos o método `corr()` da biblioteca Pandas que retorna um Data Frame com a correlação de Pearson calculada para todos os atributos existentes. Nesse exemplo selecionamos apenas algumas colunas do conjunto de dados. Vejamos como ficou o código:

```
#Leitura dos dados, selecionando apenas alguns atributos de interesse para análise
de correlação
vendas_casas = pd.read_csv("./DataSets/house_sales.csv",
                           sep="\t",
                           usecols=['SalePrice', 'zhvi_px', 'zhvi_idx',
                                    'AdjSalePrice', 'NbrLivingUnits', 'SqFtLot',
                                    'SqFtTotLiving', 'SqFtFinBasement',
                                    'Bathrooms',
                                    'Bedrooms', 'BldgGrade', 'YrBuilt',
                                    'YrRenovated',
                                    'TrafficNoise', 'LandVal', 'ImpsVal'])
#Criando um Data Frame com os atributos correlacionados
correlacao = vendas_casas.corr()

#Criando uma figura de 20x15
fig, ax = plt.subplots(figsize=(20,15))

#Exibindo o gráfico com as correlações sendo pintadas de acordo com o valor
im = ax.imshow(correlacao, cmap='RdBu')

#Configurando a figura para exibir todos os rótulos
ax.set_xticks(np.arange(len(vendas_casas.columns)))
ax.set_yticks(np.arange(len(vendas_casas.columns)))

#Adicionando os mesmo rótulos para os eixos x e y
ax.set_xticklabels(vendas_casas.columns)
ax.set_yticklabels(vendas_casas.columns)
```

```
# Rotacionando os rótulos de x em 45 graus para melhorar a exibição
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")
```

```
plt.show()
```

Como é de praxe, você poderá encontrar mais informações sobre a biblioteca na [documentação oficial](#).

Introdução ao Seaborn

Seaborn é uma biblioteca gráfica que age sobre o Matplotlib para gerar gráficos com designs mais interessantes. Só de importar para o projeto você perceberá a diferença nas visualizações geradas. Primeiro iremos plotar um gráfico utilizando apenas Matplotlib, em seguida importaremos para o projeto Seaborn, então realizaremos a mesma visualização. Veja e perceba a diferença:

```
#Gerando os dados de exemplo
rng = np.random.RandomState(0)
x = np.linspace(0, 10, 500)
y = np.cumsum(rng.randn(500, 6), 0)
```

```
#Gerando a visualização apenas com Matplotlib
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left');
```

```
#Importando Seaborn e aplicando a configuração no projeto
import seaborn as sns
sns.set()
```

```
#Gerando a mesma visualização após importar Seaborn
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left')
```

Vamos explorar alguns gráficos de Seaborn, por exemplo, visualizando duas distribuições em histogramas sobrepostos:

```
data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]], size=2000)
data = pd.DataFrame(data, columns=['x', 'y'])
```

```
for col in 'xy':
    plt.hist(data[col], alpha=0.5)
```

Seaborn traz consigo alguns conjuntos de dados para que você possa experimentar visualizações diferentes e aplicar em seus projetos. Vejamos um gráfico de dispersão no conjunto de dados de voos:

```
flights_data = sns.load_dataset("flights")
flights_data.head()
sns.scatterplot(data=flights_data, x="year", y="passengers")
```

Agora, um gráfico de linhas:

```
sns.lineplot(data=flights_data, x="year", y="passengers")
```

Em seguida, um gráfico de barras com linha indicando outliers:

```
sns.barplot(data=flights_data, x="year", y="passengers")
```

Usando o conjunto de dados de gorjeta, vamos gerar um gráfico de calor:

```
tips_df = sns.load_dataset('tips')
tips_df.head()
#Adiciona o atributo porcentagem da gorjeta
tips_df["tip_percentage"] = tips_df["tip"] / tips_df["total_bill"]
#Cria uma tabela pivot para gerar a visualização
pivot = tips_df.pivot_table(
    index=["day"],
    columns=["size"],
    values="tip_percentage",
    aggfunc=np.average)
sns.heatmap(pivot)
```

A [documentação oficial](#) traz diversos exemplos e aplicações que você poderá explorar em seus projetos.

Referências

Amos, D.; **A Practical Introduction to Web Scraping in Python**; Disponível em: <https://realpython.com/python-web-scraping-practical-introduction/> Acessado em: 19/03/2021

Brink, H; Richards, J. W.; Fetherolf, M. **Real World Machine Learning**; 2017; Manning Publications; NY.

Baeza-Yates, R.; Ribeiro-Neto, B.; **Information Retrieval: the concepts and technology behind search**; 2ed; 2011; Addison Wesley.

Bruce, P.; Bruce, A.; **Estatística Prática para Cientistas de Dados: 50 Conceitos Essenciais**; 2019; Alta Books; Rio de Janeiro

Dataaspirant; **Five most popular similarity measures implementation in python**; 2015; Disponível em: <https://dataaspirant.com/five-most-popular-similarity-measures-implementation-in-python/>; Acessado em: 07/06/2021

Deep Learning Book; Disponível em: <https://www.deeplearningbook.com.br/> Acessado em: 13/06/2021

Documentação Oficial do Python; Disponível em: <https://docs.python.org/3/> Acessado em: 19/03/2021

Documentação Oficial da Biblioteca Numpy; Disponível em: <https://numpy.org/doc/stable/> Acessado em: 21/03/2021

Documentação Oficial da Biblioteca Matplotlib; Disponível em: <https://matplotlib.org/> Acessado em: 30/03/2021

Documentação Oficial da Biblioteca Seaborn; Disponível em: <https://seaborn.pydata.org/> Acessado em: 30/03/2021

Galarnyk, M.; **Understanding Decision Trees for Classification in Python**; 2019; Disponível em: <https://www.kdnuggets.com/2019/08/understanding-decision-trees-classification-python.html>; Acessado em: 10/06/2021

Géron, A.; **Hands-On Machine Learn with Scikit-Learn, Keras & TensorFlow**: Concepts, Tools, and Techniques to Build Intelligent Systems; 2019; 2ed; O'Reilly Media; CA.

Martinez, J. C.; How To Build Beatiful Plot With Python and Seaborn; 2020; Disponível em: <https://livecodestream.dev/post/how-to-build-beautiful-plots-with-python-and-seaborn/>; Acessado em: 30/03/2021

Matheus, Y.; **Matplotlib uma Biblioteca Python para Gerar Gráficos Interessantes**; 2018; Disponível em: <https://www.alura.com.br/artigos/criando-graficos-no-python-com-a-matplotlib>; Acesso em: 30/03/2021

Preste, R.; **From XML to Pandas dataframes**; 2019; Disponível em: <https://medium.com/@robertopreste/from-xml-to-pandas-dataframes-9292980b1c1c> Acessado em: 27/03/2021

Python File I/O; Disponível em: <https://www.programiz.com/python-programming/file-operation> Acessado em: 19/03/2021

Python Iluminado; Disponível em: <https://pythoniluminado.netlify.app/> Acessado em: 19/03/2021

Python Tutorial; Disponível em: <https://www.w3schools.com/python/default.asp> Acessado em: 19/03/2021

Reif, R.; **Limitations of Applying Dimensionality Reduction using PCA**; 2018; Disponível em: <https://www.robertoreif.com/blog/2018/1/9/pca> Acessado em: 03/06/2021

Reinforcement Learning Applications; Disponível em: <https://neptune.ai/blog/reinforcement-learning-applications> Acessado em: 10/05/2021

Rodrigues, V.; **Métricas de Avaliação**: acurácia, precisão, recall... quais as diferenças?; 2019; Disponível em: <https://vitorborbarodrigues.medium.com/m%C3%A9tricas-de->

[avalia%C3%A7%C3%A3o-acur%C3%A1cia-precis%C3%A3o-recall-quais-as-diferen%C3%A7as-c8f05e0a513c](#) Acessado em: 13/06/2021

Santana, R.; **Plotando Gráficos de um Jeito Fácil com Python**; 2020; Disponível em: <https://minerandodados.com.br/plotando-graficos-de-forma-facil-com-python/>; Acessado em: 30/03/2021

Santana, R.; **Análise de Dados com Python usando Pandas**; 2019; Disponível em: <https://minerandodados.com.br/analise-de-dados-com-python-usando-pandas/>; Acessado em: 21/03/2021

The Marketing Research; **Select a Clustering Procedure**; 2015; Disponível em: <https://themarketingresearch.com/select-a-clustering-procedure-12089>; Acesso em: 06/06/2021.

Tiwari, G.; **Python IF, IF ELSE, ELIF & Nested IF Statements**; Disponível em: <https://medium.com/@tiwarigaurav2512/python-if-if-else-elif-nested-if-statements-d443cf0bb2e1> Acesso em: 19/03/2021

Trifonova, O. P., Lokhov, P. G., Archakov, A. I.; **Metabolic profiling of human blood**; 2014; *Biomeditsinskaya khimiya*, 60(3), 281-294.

Tsanas, A.; Arora, S.; **Large-scale Clustering of People Diagnosed with Parkinson's Disease using Acoustic Analysis of Sustained Vowels: Findings in the Parkinson's Voice Initiative Study**; In [Proceedings of the 13th International Joint Conference on Biomedical Engineering Systems and Technologies - Volume 4: SERPICO](#), 369-376, 2020 , Valletta, Malta

Tutorial Python em Português; Disponível em <https://docs.python.org/pt-br/3/tutorial/index.html> Acessado em: 19/03/2021

Unsupervised Machine Learning; Disponível em: <https://www.guru99.com/unsupervised-machine-learning.html> Acesso em: 10/05/2021

Ven, F.; **The Neural Network Zoo**; Disponível em: <https://www.asimovinstitute.org/neural-network-zoo/>; Acessado em: 25/08/2021

Visualização de Gráficos 2D Usando Matplotlib; Disponível em:
<https://panda.ime.usp.br/algoritmos/static/algoritmos/10-matplotlib.html>; Acessado em:
30/03/2021