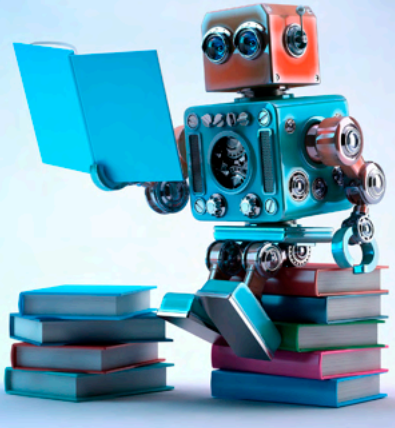
Gabriel Di Santis Sylvestre Pires
Guilherme Cruz Ferreira
Joao Vitor Fernandes Cavalcanti
Anna Cristina Barbosa Dias de Carvalho

APRENDIZADO DE MÁQUINA

COM PYTHON 3

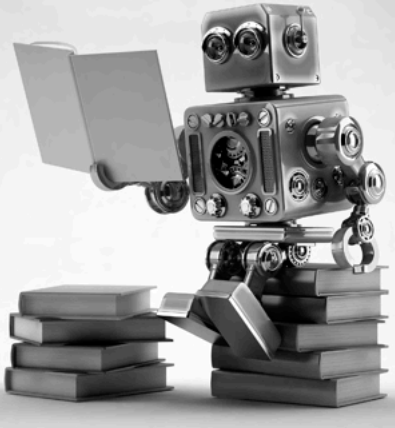


Atena Editora
Ano 2022

Gabriel Di Santis Sylvestre Pires
Guilherme Cruz Ferreira
Joao Vitor Fernandes Cavalcanti
Anna Cristina Barbosa Dias de Carvalho

APRENDIZADO DE MÁQUINA

COM PYTHON 3



Atena Editora
Ano 2022

Editora chefe
Profª Drª Antonella Carvalho de Oliveira
Editora executiva
Natalia Oliveira
Assistente editorial
Flávia Roberta Barão
Bibliotecária
Janaina Ramos
Projeto gráfico
Bruno Oliveira
Camila Alves de Cremo
Daphynny Pamplona
Luiza Alves Batista
Natália Sandrini de Azevedo
Imagens da capa
iStock
Edição de arte
Luiza Alves Batista

Atena Editora
Ano 2022

Atena
Editora
Ano 2022

Diagramação: Gabriel Motomu Teshima
Correção: Bruno Oliveira
Indexação: Amanda Kelly da Costa Veiga
Revisão: Os autores
Autores: Gabriel Di Santis Sylvestre Pires
Guilherme Cruz Ferreira
Joao Vitor Fernandes Cavalcanti
Anna Cristina Barbosa Dias de Carvalho

Dados Internacionais de Catalogação na Publicação (CIP)

A654 Aprendizado de máquina com python 3 / Gabriel Di Santis Sylvestre Pires, Guilherme Cruz Ferreira, Joao Vitor Fernandes Cavalcanti, et al. - Ponta Grossa - PR, 2022.

Outra autora
Anna Cristina Barbosa Dias de Carvalho

Formato: PDF
Requisitos de sistema: Adobe Acrobat Reader
Modo de acesso: World Wide Web
Inclui bibliografia
ISBN 978-65-258-0069-1
DOI: https://doi.org/10.22533/at.ed.691222604

1. Engenharia de máquinas. 2. Python. 3. Aprendizado de máquina. 4. Tecnologia. I. Pires, Gabriel Di Santis Sylvestre. II. Ferreira, Guilherme Cruz. III. Cavalcanti, Joao Vitor Fernandes. IV. Título.

CDD 621.8

Elaborado por Bibliotecária Janaina Ramos – CRB-8/9166

Atena Editora
Ponta Grossa – Paraná – Brasil
Telefone: +55 (42) 3323-5493
www.atenaeditora.com.br
contato@atenaeditora.com.br

Atena Editora
Ano 2022

DECLARAÇÃO DOS AUTORES

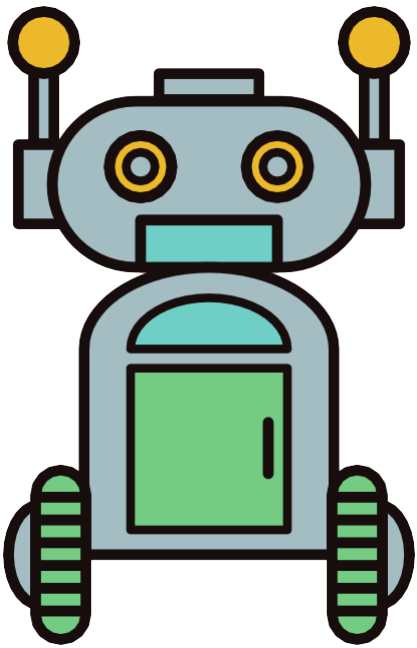
Os autores desta obra: 1. Atestam não possuir qualquer interesse comercial que constitua um conflito de interesses em relação ao artigo científico publicado; 2. Declaram que participaram ativamente da construção dos respectivos manuscritos, preferencialmente na: a) Concepção do estudo, e/ou aquisição de dados, e/ou análise e interpretação de dados; b) Elaboração do artigo ou revisão com vistas a tornar o material intelectualmente relevante; c) Aprovação final do manuscrito para submissão.; 3. Certificam que o texto publicado está completamente isento de dados e/ou resultados fraudulentos; 4. Confirmam a citação e a referência correta de todos os dados e de interpretações de dados de outras pesquisas; 5. Reconhecem terem informado todas as fontes de financiamento recebidas para a consecução da pesquisa; 6. Autorizam a edição da obra, que incluem os registros de ficha catalográfica, ISBN, DOI e demais indexadores, projeto visual e criação de capa, diagramação de miolo, assim como lançamento e divulgação da mesma conforme critérios da Atena Editora.

DECLARAÇÃO DA EDITORA

A Atena Editora declara, para os devidos fins de direito, que: 1. A presente publicação constitui apenas transferência temporária dos direitos autorais, direito sobre a publicação, inclusive não constitui responsabilidade solidária na criação dos manuscritos publicados, nos termos previstos na Lei sobre direitos autorais (Lei 9610/98), no art. 184 do Código Penal e no art. 927 do Código Civil; 2. Autoriza e incentiva os autores a assinarem contratos com repositórios institucionais, com fins exclusivos de divulgação da obra, desde que com o devido reconhecimento de autoria e edição e sem qualquer finalidade comercial; 3. Todos os e-book são *open access, desta forma* não os comercializa em seu site, sites parceiros, plataformas de *e-commerce,* ou qualquer outro meio virtual ou físico, portanto, está isenta de repasses de direitos autorais aos autores; 4. Todos os membros do conselho editorial são doutores e vinculados a instituições de ensino superior públicas, conforme recomendação da CAPES para obtenção do Qualis livro; 5. Não cede, comercializa ou autoriza a utilização dos nomes e e-mails dos autores, bem como nenhum outro dado dos mesmos, para qualquer finalidade que não o escopo da divulgação desta obra.

INTRODUÇÃO

Esse *ebook* tem por objetivo ampliar o alcance e quantidade de pessoasinseridas no universo de aprendizado de máquina e suas possibilidades,trazendo a tona conceitos básicos de uma linguagem de fácil acesso epropiciando ao leitor um conjunto de ferramentas que possibilite ao mesmoanalisar novos modelos e entrar cada vez mais nesse mundo que está emcrescimento. Com isso em mente esse livro digital utilizou de linguagemvisual em conjunto com a plataforma do Google Colab para apresentar deforma dinâmica os códigos em funcionamento e permitir a interação do leitorcom o que ele está visualizando na tela através de uma pasta do GoogleDrive, além de fundamentos da linguagem em uso esse ebook aborda aindauma aplicação de aprendizado de máquina apresentando os pontos queestruturam o código e os resultados obtidos.



SUMÁRIO

INTRODUÇÃO À LINGUAGEM PYTHON

O Python é uma linguagem de programação criada no final da década de 1980 e tem como principal objetivo tornar a leitura e entendimento do código mais simples de ser compreendida tendo também como resultado escrever programas complexos com menos linhas, além dessas características, um atributo que diferencia ela da maioria das outras linguagens de programação é o de possuir código aberto.

Código aberto significa que é possível alterar seu código fonte, trazendo customização e possibilitando a criação de atalhos para certas aplicações que são chamadas de módulos.

Os módulos são grande parte do Python para facilitar e tornar possível interações, por exemplo, o módulo “MatPlotLib” permite a criação de gráficos para uma análise mais crítica dos dados de determinado programa, por meio de uma simples sintaxe e módulos personalizados, o Python consegue simplificar o estudo da inteligência artificial que é o principal foco nesse e-book [1].

O Python pode ser considerado multiplataforma, englobando Windows, macOS, Linux, Android, IOS e muitas outras, porém sua performance em dispositivos móveis é significativamente menor do que em sistemas operacionais mais robustos [1].

A tabela abaixo apresenta as principais vantagens e desvantagens do Python:

Vantagens	Desvantagens
Sintaxe simples	Alto uso de memória
Código aberto	Impopular em desenvolvimento de software na indústria
Módulos personalizados	Difícil aplicação em plataformas móveis
Multiplataforma	
Multiuso	

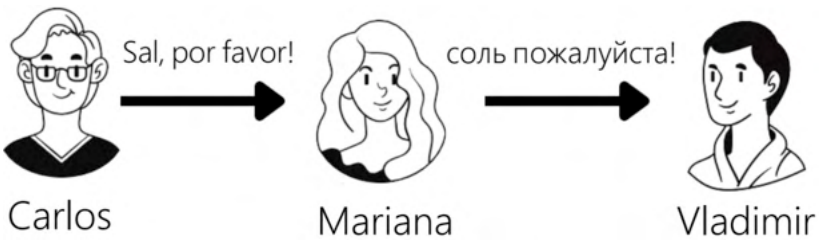
Tabela 1 - Vantagens e desvantagens do Python

Fonte: *Python Pocket Reference*

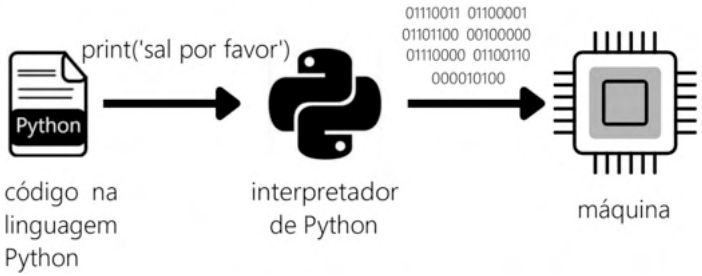
CONHECENDO O INTERPRETADOR

As linguagens de programação permitem a comunicação de forma lógica com um computador. Mas como se comunicar com pedaços de metal e plástico que só falam em 1s e 0s?

Para responder essa pergunta, vamos entender um exemplo prático: Carlos e Mariana são irmãos brasileiros e ambos dominam o Português, porém, Mariana possui fluência no idioma russo. Vladimir, amigo de Mariana chegou da capital Russa para passar o verão no Brasil, mas ele só fala russo. Naturalmente, quando Carlos deseja se comunicar com Vladimir, ele não procura um dicionário português-russo para traduzir cada palavra para o idioma natal da visita, mas pede para Mariana traduzir, o que poupa muito tempo quando se deseja apenas pedir para que passe o sal.



O Python é o nome da linguagem, assim como é o nome de seu interpretador, portanto, para fazer códigos funcionarem ela necessita, primeiramente, do código escrito na linguagem Python e seu interpretador, que traduzirá o código escrito para código de máquina [2]. Então, a representação será a seguinte:



INSTALANDO INTERPRETADOR PYTHON

Neste e-book iremos trabalhar com o "Google Collaboratory" que é uma plataforma onde podemos escrever algoritmos de programação e mantê-los públicos para que qualquer pessoa consiga utilizar e testar no próprio navegador, porém para darmos inicio e introduzir as principais ferramentas do Python, utilizaremos a máquina local (seu próprio computador).

Agora que conhecemos um pouco mais como o Python e seu interpretador funcionam, é a hora de acessar o site da organização oficial do Python e baixar a versão mais atualizada do interpretador.

PARTE 1 - INSTALANDO O INTERPRETADOR NO WINDOWS

Passo 1

Para instalar o interpretador de Python é necessário entrar no site oficial da organização Python através desse link ou digitando o endereço https://www.python.org/ no seu navegador de internet. Após isso clicar na seção "Downloads".

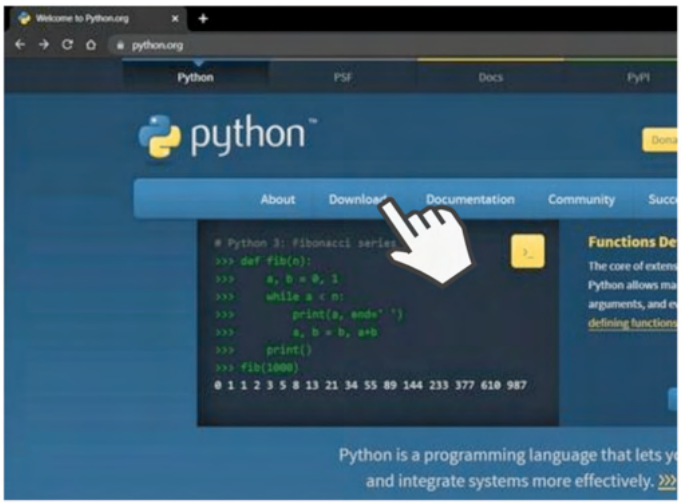


Imagem 1 - site Python.org

Fonte: Autoria própria

Agora é só clicar no botão amarelo que diz “Download Python”. Neste e-book vamos trabalhar como Python na versão 3 e esse botão baixará a versão mais atualizada.

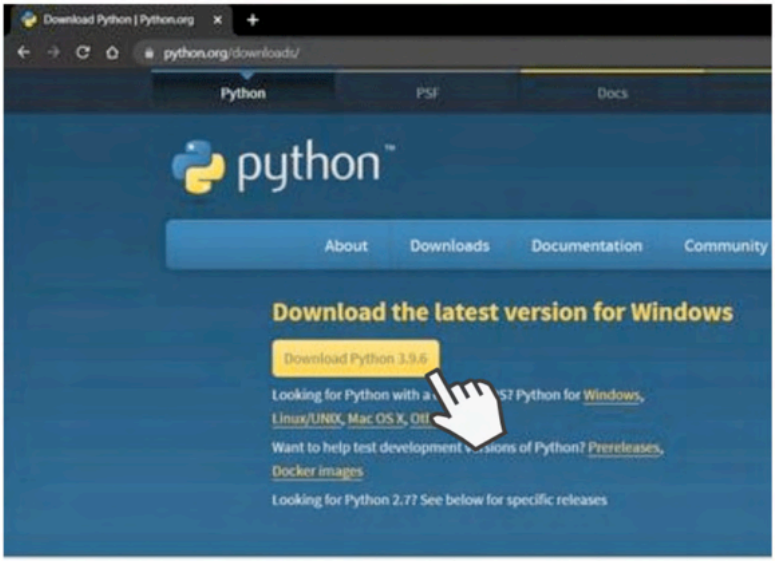
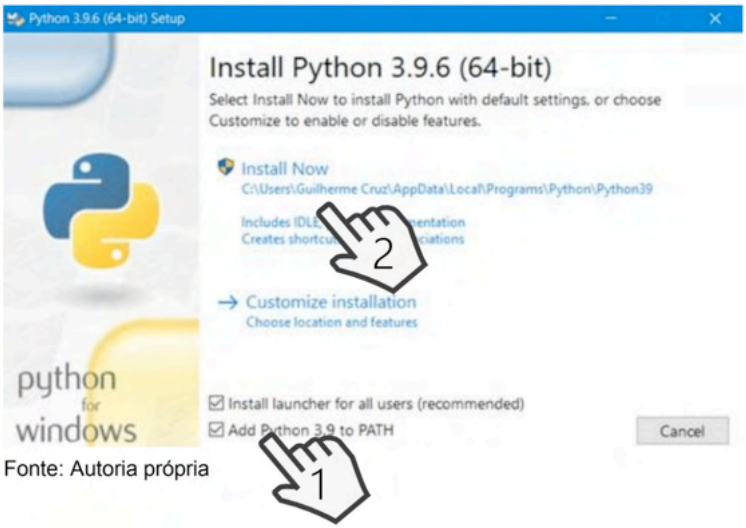


Imagem 2 -Download Python.org

Fonte: Autoria própria

Passo 2

Após a conclusão do download abrir o arquivo e selecionar a opção "Add Python 3.9 to PATH", que fará iniciar o Python pelo CMD (*Command Prompt* ou *Prompt de comando*) mais simples e não necessitará do endereço específico onde o Python será instalado e depois clicar em "*Install now*".



Fonte: Autoria própria

Imagem 3 - setup Python.

Após a instalação dele aparecerá uma janela semelhante com a abaixo, e fim! Instalamos o Python na sua versão mais recente com sucesso.

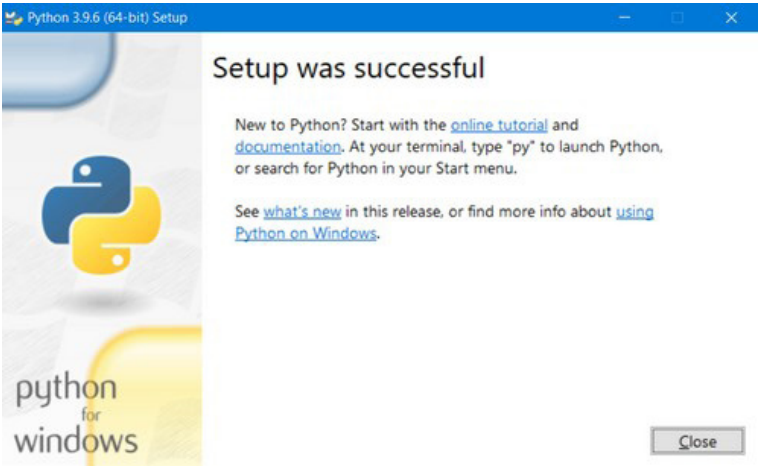


Imagem 4 -conclusão setup Python

Fonte: Autoria própria

PARTE II - CHECANDO A VERSÃO DO PYTHON NO WINDOWS

Para checar a versão do Python que foi instalada no sua máquina, basta abrir seu terminal seguindo os seguintes passos:

Passo A - Apertar tecla Windows

Passo B - Digitar "Prompt de comando"

Passo C - Apertar tecla "Enter"

E por fim digitar o seguinte comando:

```
python --version
```

Esse comando retornará a versão instalada como no exemplo abaixo:

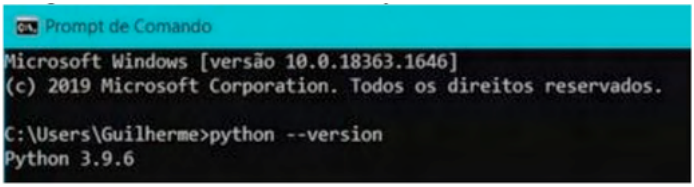


Imagem 5 -checando versão do Python

Fonte: Autoria própria

PARTE III - ATUALIZANDO O INTERPRETADOR DE PYTHON NO LINUX

Se você usa Linux, você não está desamparado. Mesmo que provavelmente você saiba mais sobre computadores do que a gente (o que é muito provável) vamos também colocar como instalar ou atualizar seu interpretador Python para a versão mais recente através do terminal, pois ele vem instalado de fábrica no seu sistema operacional.

Passo 1

Acessar o terminal do sistema operacional seguindo os seguintes passos:

Passo A - Acessar seu pesquisador de aplicativos ou apertar tecla Windows

Passo B - Digitar "terminal"

Passo C - Apertar tecla "Enter"

E em seguida digitar o seguinte comando:

```
sudo apt-get install python3
```

Por possuir diversas variações, vamos cobrimos apenas o comando das versões Ubuntu e Debian por serem as mais utilizadas, se você estiver utilizando outra versão ou

se o comando não funcionou, o Python também pode ser instalado de forma semelhante ao Windows utilizando seu navegador através desse link ou acessando https://python.org.br/instalacao-linux/.

PARTE IV - CHECANDO A VERSÃO DO PYTHON NO LINUX

Para checar a versão do Python que foi instalada no sua máquina, basta abrir seu terminal e digitar o seguinte comando:

```
python --version
```

ROTEIRO PYTHON

Para poder mexer com Inteligência artificial com **PYTHON**, é necessário saber usar o Python, portanto, neste capítulo, vamos nos colocar em sintonia nessa questão, para que possamos aproveitar melhor nosso concorrido tempo.

Para nos ajudar a entender e a demonstrar as características da linguagem, nós, autores utilizamos a biblioteca oficial do Python que se encontra nesse link ou inserindo

https://docs.python.org/pt-br/3.6/reference/index.html#reference- index

Na barra de pesquisa do seu navegador note que, apesar da página estar na língua portuguesa (BR), as palavras chave do Python são imutáveis em questão da língua.

PARTE 1 - PYTHON INTERATIVO E OPERADORES BÁSICOS

Para iniciar o Python no seu computador primeiro acesse seu terminal e digite "python"

Imagem 6 - Acionando o Python pelo terminal.

Fonte: Autoria própria

A presença do ">>>" no início da linha indica que já estamos no Python interativo, esse modo permite que façamos operações e obtenhamos retorno imediato do sistema.

Vamos observar os exemplos de algoritmos abaixo e tente reproduzi-los no seu terminal, lembrando que é preciso pressionar a tecla "Enter" para obter o retorno.

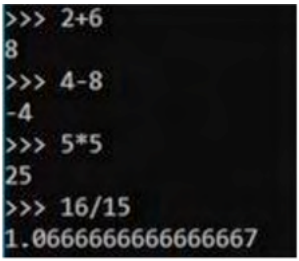


Imagem 7 - operações no terminal

Fonte: Autoria própria

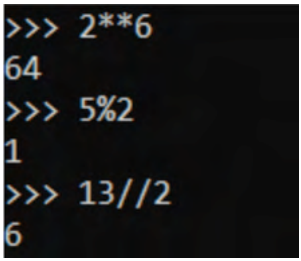


Imagem 8 - operações no terminal

Fonte: Autoria própria

Estes são alguns exemplos de operações, mas também é possível realizá-las em conjunto, com a opção de utilizar agrupamento com os parênteses. Abaixo está a descrição das operações e como utilizá-las.

+	2+2 → 4	Soma dois números
-	4-3 → 1	Subtrai dois números
*	6*4 → 24	Multiplica dois números
/	16/4 → 4.0	Divide dois números
**	4**3 → 64	Retorna potência de um número
%	7%3 → 1	Retorna resto de uma divisão
//	10//3 → 3	Divide e arredonda para baixo

Tabela 2 - Operações no Python

Fonte: Autoria própria

Mais exemplos do que vimos nesse capítulo:

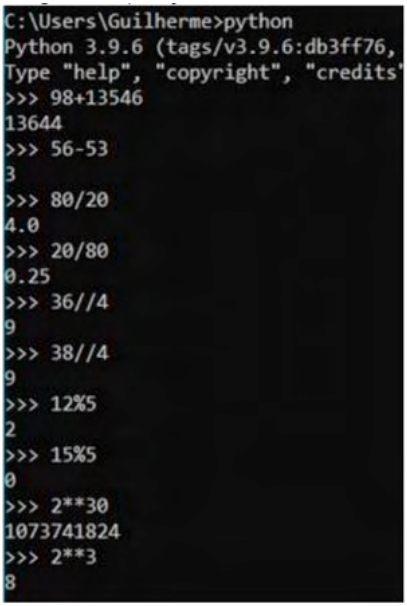


Imagem 9 -operações no terminal

Fonte: Autoria própria

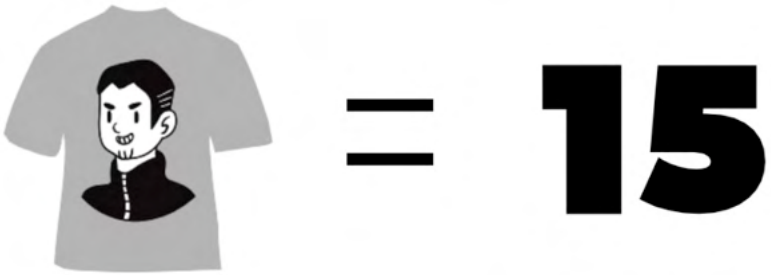
Imagem 10 -operações no terminal

Fonte: Autoria própria

Agora, é sua vez! Aproveite esse momento para aplicar essas ferramentas de forma criativa no seu Python.

PARTE 2 - VARIÁVEL

Magali e Joaquim vão abrir uma loja de camisetas estampadas como rosto de artistas famosos. Na hora de decidir os preços, Magali eJoaquim definiram que a camiseta como rosto do cantor local Ernesto-Bom-De-Baixo seria vendida por 15 reais.



A camiseta - que é a variável - agora possui o valor de 15 reais e quando ela precisar checar o valor desta camiseta no sistema, ou quando for somá-la com outra camiseta ela terá esse valor.

No Python você pode criar variáveis com qualquer nome e tamanho, desde que obedeçam algumas regras:

- Para criar uma variável é necessário criar o nome dela e inserir o símbolo "=" e em seguida o valor a ser atribuído.
- Variáveis em Python podem conter números, letras acentuadas ou não e o caractere "_".
- Variáveis só podem começar com letras de A-Z não importando se é maiúscula ou minúscula e o caractere "_".
- A atribuição não é fixa: a qualquer momento pode-se atribuir um novo valor à variável.
- Variáveis não podem ser palavras reservadas que possuem alguma função atribuída no Python. (serão apresentadas nos tópicos a seguir).

```
C:\Users\Guilherme>python
Python 3.9.6 (tags/v3.9.6:db
Type "help", "copyright", "c
>>> Variável_1 = 5
>>> Variável_2 = 4
>>> Variável_1 + Variável_2
9
```

Imagem 11 - operações no terminal.

Fonte: Autoria própria.

Mais exemplos do que vimos nesse capítulo:

```
>>> camiseta_xadrez = 46
>>> camiseta_xadrez + 12
58
>>> camiseta_branca = 5
>>> camiseta_preta = 5
>>> camiseta_colorida = 6
>>> camiseta_branca * 2
10
>>> (camiseta_branca + camiseta_preta) * camiseta_colorida
60
>>>
```

Imagem 12 -operações no terminal.

Fonte: Autoria própria.

```
>>> var1 = 10
>>> var2 = 20
>>> var3 = 30
>>> var1 * var2 + var3
230
>>> var1 * (var2 + var3)
500
>>> var2 - var1
10
```

Imagem 13 -operações no terminal.

Fonte: Autoria própria.

Agora, é sua vez! Aproveite esse momento para aplicar essas ferramentas de forma criativa no seu Python.

PARTE 3 - TIPOS PRIMITIVOS

As variáveis no Python possuem duas características principais de armazenamento: O tipo de dado, e como esse dado está organizado. Nessa parte vamos cobrir quais são os tipos de dados e quais suas principais características.

PARTE I - TIPOS DE DADOS NUMÉRICOS

Números se apresentam de duas formas nessa linguagem de programação, na forma do conjunto de números **INTEIROS**, onde serão classificados como **int** (do inglês "integer", que significa inteiro), ou na forma do conjunto de números **REAIS**, que por sua vez serão os **float** (ou flutuantes, no português).

De um ponto de vista prático, os int e os float funcionam da mesma forma que ocorre na matemática clássica, mas então onde ocorrem as diferenças?

> Pelas regras da matemática, quando dividimos dois números que possuem uma divisão exata, temos como resultado um número inteiro. Mas no Python é outra história, pois sempre que dividimos um número por outro, mesmo sendo um resultado exato, o tipo da variável resultante é um **float.**

Para nos ajudar a comprovar esse fato, além da parte decimal mostrada no número, podemos conferir o tipo da variável com a função abaixo:

```
type(var)
```

As funções são palavras que fazem algo no código chamando elas onde se deseja que ocorra a função, e podem ou não necessitar de uma variável para funcionar, que será declarada dentro do parênteses, sendo que determinadas funções só funcionam com tipos específicos de variáveis. Mas não se preocupe pois as funções serão abordadas com mais profundidade nos próximos capítulos.

Vamos pular no terminal do computador e iniciar o Python interativo para descobrir quais são essas diferenças!

Segundo a imagem abaixo tirado do Python interativo, podemos ver que os números 90 e -78 são da classe dos int e 78.00 e 73.98 são da classe dos float.

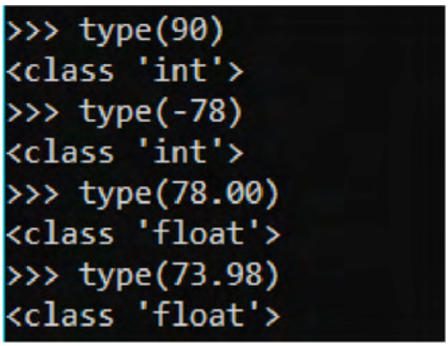


Imagem 14 - operações no terminal.

Fonte: Autoria própria.

E podemos constatar também que mesmo estando dentro de uma variável podemos checar o tipo dela.

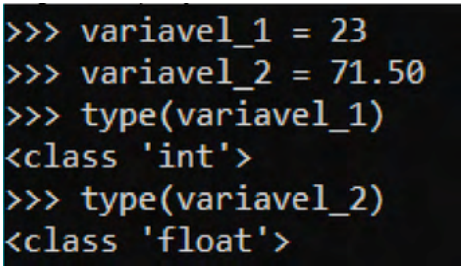


Imagem 15 - operações no terminal.

Fonte: Autoria própria.

PARTE II - TIPOS DE DADOS DE TEXTO

Além dos números, podemos manipular texto dentro do Python! Seu uso pode ser desde gerar senhas aleatórias, até criar um bot para conversar com você.

Todos os textos que podemos manipular com o Python participa da classe **string**.

As **strings** vão aparecer apenas entre aspas ou aspas duplas, porém devemos fechar com a mesma que começamos, como no exemplo abaixo:

'olá, mundo!'

"Hello, world!"

'Привет мир!'

PROPRIEDADE ESPECIAL

Quando criamos strings, por exemplo com aspa simples, devemos "fechar" ela com aspa simples. Mas e se precisarmos usar aspas no sentido de destacar alguma informação dentro dessa string, como fazemos?

Como você pode ter imaginado, nós utilizaremos a aspa que não utilizamos para denotar o início e o final da string.

Complicado, né? mas é mais simples do que parece, vejamos alguns exemplos:

```
"Muito 'bonito' o que você está fazendo!"

'O que você está fazendo é muito
"cringe", amigo'

'Olá "flor"'
```

PARTE 4 - GOOGLE COLAB

Quando trabalhamos com um código maior, que precisa de informações de entrada de um usuário, além de salvar o código já desenvolvido, fazemos uso de um editor de texto, que irá nos prover uma maior dinâmica com o código escrito.

A partir desse capítulo vamos utilizar a plataforma do Google Colab para desenvolver os códigos do tutorial de Python, tanto para apresentar novas funções de um código editável e para nos acostumar com o ambiente onde vamos criar os algoritmos de inteligência artificial.

No Colab o código é executado em células. Esse recurso permite a execução de apenas algumas partes de código sem comprometer as outras. A outra principal característica é a facilidade de criar, comentar e compartilhar algoritmos.

Para que seja possível acompanhar os próximos exemplos, basta entrar no link a seguir que redirecionará para a pasta onde se encontram os códigos.

https://drive.google.com/drive/folders/1ggcHVrf7fqXBz9yixeDQZWKYjJXZGMld?usp=sharing

A visão que se tem ao abrir o link do Google Drive será a da pasta principal onde encontramos as pastas referentes aos capítulos, e dentro delas teremos os arquivos do Google Colab.

Imagem 16 - Pasta Google Colab

Fonte: Google Colab

Acessando a pasta do capítulo 1 por exemplo, teremos acesso aos códigos dos exemplos dados em cada parte do capítulo 1 a partir de agora.

Imagem 17 - Capítulo 1 Google Colab

Fonte: Google Colab

Ao acessar o arquivo do Google Colab podemos executar os códigos e fazer modificações, porém essas modificações feitas não serão salvas, visto que esse arquivo é "público" e outras pessoas vão usar, mas é possível salvar uma cópia do arquivo no seu próprio google drive, onde as alterações feitas serão mantidas.

Para criar uma cópia onde suas modificações ficam salvas, basta abrir o arquivo desejado, ir na aba "Arquivo" e em seguida clicar em "Salvar uma cópia no Drive, conforme mostra a imagem 18.

Imagem 18 - Salvar Cópia

Fonte: Google Colab

Agora com o arquivo já aberto vamos ver algumas informações importantes, o Google Colab trabalha com células de execução, então ele permite que você execute etapa por etapa do código, na imagem X podemos ver um exemplo de uma célula, para executar a célula individualmente, basta clicar no símbolo de play.

Imagem 19 - Salvar Cópia

Fonte: Autoria própria

Uma parte importante do Google Colab é a aba "Ambiente de execução", onde temos opções como executar todas nossas células e reiniciar ambiente de execução, que é responsável por fazer o código voltar do zero.

Imagem 20 - Aba Ambiente de Execução

Fonte: Google Colab

Com isso em mente já podemos continuar, agora com a facilidade de poder ver e mexer no código de forma simples, vale lembrar também que as partes que estão em

verde e que tem o sinal de cerquilha ("#") no seu início, são os comentários feitos para um melhor entendimento do código.

A página inicial do Google Collab está apresentada na imagem abaixo:

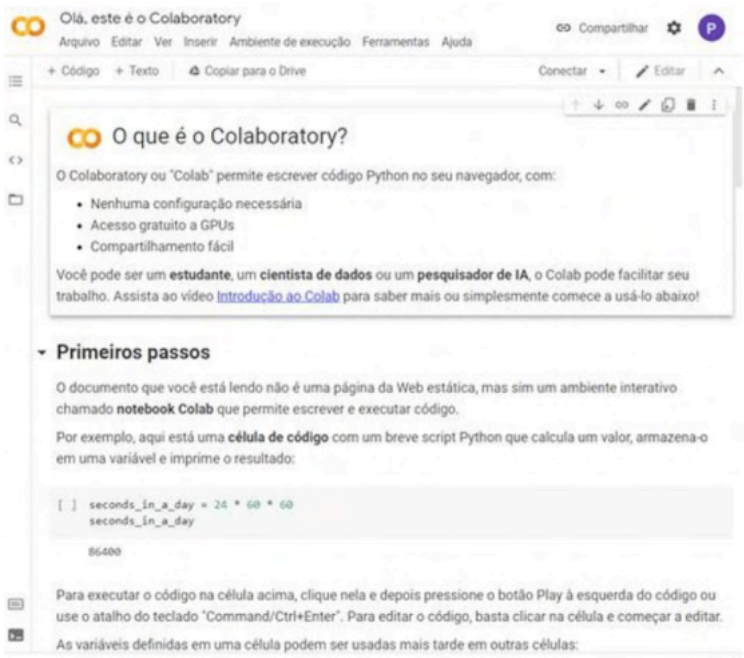


Imagem 20 - Aba Ambiente de Execução

Fonte: Google Colab

Para criar uma nova página para programar, clique em "Arquivo" e após "Novo notebook".

Para acompanhar os próximos exemplos, basta entrar no link a seguir que redirecionará para a pasta onde se encontram os códigos que irão aparecer.

https://drive.google.com/drive/folders/1ggcHVrf7fqXBz9yixeDQZW KYjJXZGMld?usp=sharing

PARTE 5 - COMPARADORES

No nosso período letivo nos acostumamos com sinais para comparação de valores, alguns deles sendo a igualdade, dizer se um item possui maior ou menor valor do que outro. No Python fazemos operações semelhantes utilizando esses mesmos símbolos e conceitos.

==	igual a
>	maior que
<	menor que
>=	maior ou igual
<=	menor ou igual
!=	diferente de

Tabela 3 - Comparadores do Python

Fonte: Autoria própria

Vamos a um exemplo prático!

No seu terminal abra o Python e faça uma comparação simples:

1.- Crie duas variáveis e atribua valores à elas;

2.- Compare ambas utilizando um dos operadores comparadores da tabela acima.

Se não houve nenhum erro, seu teste resultou em algo similar a isso:

```
[1]  var1 = 8
     var2 = 10
     var1 > var2

     False
```

Imagem 21 - operações no Colab

Fonte: Autoria própria

Para sabermos o resultado da comparação o Python nos dará um feedback como True ou False.

Esse tipo de resultado também é considerado variável como a variável String, int ou float, e nós a catalogamos como variável do tipo Bool, de Boole, onde o resultado é 1 ou 0, sim ou não.

Para sabermos o resultado da comparação o Python nos dará um feedback como True ou False.

Esse tipo de resultado também é considerado variável como a variável String, int ou float, e nós a catalogamos como variável do tipo Bool, de Boole, onde o resultado é 1 ou 0, sim ou não.

True	Verdadeiro
False	Falso

Tabela 3 - Comparadores do Python

Fonte: Autoria Própria

Também é possível atribuir a variáveis. Sua peculiaridade é que a primeira letra de True ou False deve ser maiúscula.

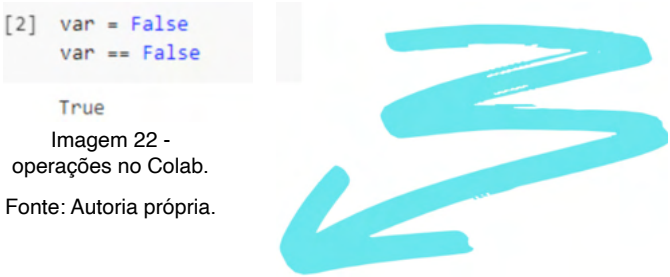


Imagem 22 - operações no Colab.

Fonte: Autoria própria.

- False foi atribuído a 'var'
- Comparação de 'var' com 'False'

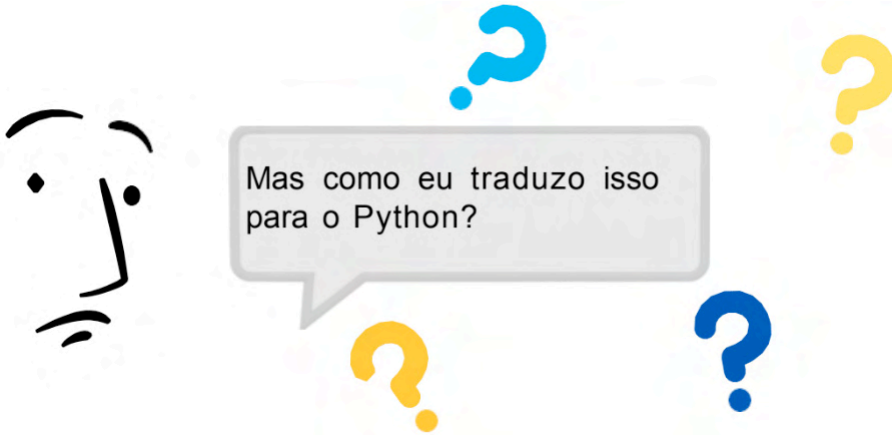
Retorno Verdadeiro da comparação.

PARTE 6 - ESTRUTURAS DE DECISÃO

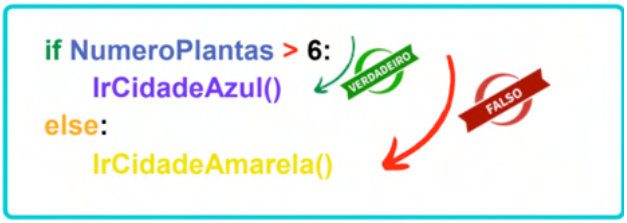
Desenvolver linhas de códigos e lógicas de programação requerem, em alguns casos, a utilização de ferramentas que possibilitem a tomada de diferentes rumos dentro do mesmo código. Pode-se imaginar as estruturas condicionais ou de decisão como uma estrada com uma bifurcação, onde dependendo das condições que se tem, pode-se tomar um rumo ou o outro.

Olhando a pergunta feita pelos personagens acima podemos notar a utilização de um comparador, um dos elementos chaves quando necessitamos avaliar alguma condição presente.

No caso acima, observamos que a condição é falsa, visto que eles possuem exatas 6 plantas dispostas, então eles seguiriam o código que os levaria para a cidade amarela, a qual podem ocorrer novas decisões, alterações na quantidade de plantas, etc.



Para realizarmos o mesmo teste que os personagens da página anterior dentro de um código, tudo que precisamos fazer é:



Parece um pouco intimidador a primeiro momento, mas calma, é mais simples do que parece, primeiramente:

- Em VERDE nós temos a palavra responsável por levantar o questionamento quanto a quantidade de plantas (if corresponde a "se" em português);
- Em AZUL observamos a variável responsável por conter a informação de quantas plantas ali estão presentes, por via de demonstração essa variável pode ter sido preenchida por algum observador bem perspicaz o qual armazenou a informação nela;
- O sinal em VERMELHO é o nosso comparador como comentado na QUARTA PARTE desse ebook;
- O 6 corresponde ao valor o qual a variável está sendo comparada;
- Em VIOLETA nós observamos a ação que será tomada caso a condição seja VERDADEIRA, que no nosso caso seria ir até a cidade azul, mas poderia ser

uma operação matemática, como por exemplo perder uma planta, ou qualquer outra atividade que se adeque ao seu código;

- Em LARANJA nós temos a palavra responsável por trazer a condição negativa da pergunta feita, ou seja, SENÃO for verdadeiro execute isso. (else corresponde a "senão" em português);

- Por fim, nos temos uma ação na cor AMARELO que ocorrerá caso a condição seja FALSA, o que nos resultaria a uma viagem a cidade amarela, o que pode fazer com que uma nova planta seja obtida.

De forma resumida, a indentação consiste na margem existente antes das instruções "IrCidadeAzul" e "IrCidadeAmarela".

Para ficar mais claro, reproduziremos o código na plataforma Google Colab:

```
[1] NumeroPlantas = 6
    #Aqui nós atribuímos o valor de 6 plantas, que é o que consta na imagem
```

Iniciamos declarando a quantidade de plantas presentes:

```
[2] def IrCidadeAzul(val):
      return val - 1
    #Aqui temos o evento que ocorreria Indo para a cidade Azul, que no caso seria a perda de uma planta.

    def IrCidadeAmarela(val):
      return val + 1
    #O sortudo indo para Cidade Amarela ganharia uma nova planta
```

Nessa etapa temos as funções responsáveis pela tomada de decisão seguinte, que no caso são os destinos dos nossos personagens.

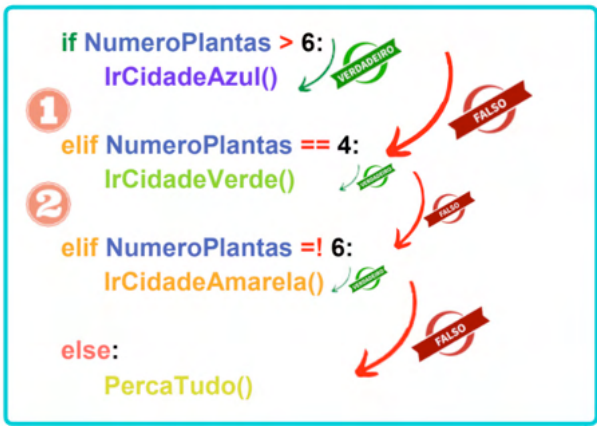
```
[3] if NumeroPlantas>6:
    #Aqui temos o teste condicional, que checa se temos MAIS que seis plantas
      NumeroPlantas = IrCidadeAzul(NumeroPlantas)
    #Uma nova atribuição de variáveis, agora com o valor recalculado de acordo com o destino.

    else:
    #Caso a condição testada seja falsa:
      NumeroPlantas = IrCidadeAmarela(NumeroPlantas)
    #Uma nova atribuição de variáveis, agora com o valor recalculado de acordo com o destino.
```

Aqui são os testes sendo realizados e a devida atribuição dos novos valores obtidos após a viagem dos personagens.

```
[4]  print(NumeroPlantas) #Exibição do resultado final do numero de plantas para o novo destino
```

Por fim temos a exibição do resultado da viagem, que no caso resultou na obtenção de uma nova planta.

Nas páginas anteriores pudemos observar como o "if" e "else" funcionam em Python, entretanto, caso tenhamos mais do que duas condições, como faríamos? Para isso adotaremos o "elif" que funciona como exemplificado a seguir:



O principio de funcionamento é bem semelhante, com o acréscimo de novos testes condicionais, vale ressaltar que a sequência a qual o código está escrito influencia diretamente no resultado, visto que caso trocássemos a posição entre o "elif" 1 e 2, o valor 4 sendo diferente de 6 executaria a função IrCidadeAmarela().

Outra forma de solucionar o problema acima pode ser visto abaixo, assim como exemplos utilizando if para mais de uma condição.

```
[1]  #Outras alternativas de solução:
     NumeroPlantas = 6

     if NumeroPlantas>6:
       NumeroPlantas = NumeroPlantas - 1

     elif NumeroPlantas == 4:
       NumeroPlantas = NumeroPlantas - 2

     elif NumeroPlantas =! 6:
       NumeroPlantas = NumeroPlantas + 1

     else:
       NumeroPlantas = 0

     print(NumeroPlantas)
```

Para utilizarmos mais do que uma condição para funcionamento do código, como podemos observar abaixo:

```
[7]  Carros = 8
     Pilotos = 9

     if Carros>6:
       if Pilotos == Carros:
         print('Podemos ter a corrida')
       elif Pilotos>Carros:
         print('Falta(m) carro(s) o(s) piloto(s)')
       else:
         print('Faltam Pilotos')
     else:
       print('O numero mínimo de carros não foi atingido')
```

Indentações →

A indentação é essencial para o funcionamento dos comandos if, elif, else.

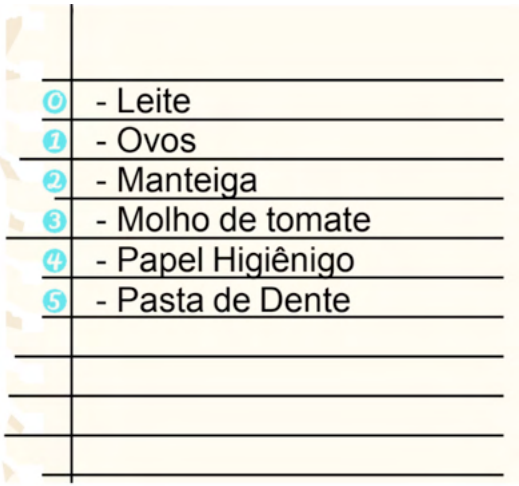
PARTE 7 - VARIÁVEIS COMPOSTAS

Quando nos deparamos com uma grande quantidade de dados, pode-se surgir a dúvida de como armazená-los, e dentro do python existem as coleções, que são recursos usados para agrupar dados.

Podemos ver as coleções como uma espécie de caixa onde serão guardados esses dados, agora abordaremos três tipos de coleções que são as listas, os dicionários e as tuplas.

PARTE I - [LISTAS]

Para entendermos a lista vamos pensar em uma lista de supermercado, onde anotamos o que precisamos comprar.



Se quisermos transferir essa lista para linguagem Python, primeiro precisamos dar um nome para essa lista que no caso será "listadecompras", lembrando de não utilizar espaços no nome, e temos que entender também que o que define essa coleção como uma lista é o uso dos colchetes ("[]").

```
listadecompras = ["Leite", "Ovos", "Manteiga", "Molho de Tomate", "Papel Higiênico", "Pasta de dente"]
```

Ao analisar a lista podemos perceber que existem números antes dos produtos,

que são os números correspondentes a sua posição, na lista no Python esse número não aparece explicitamente, mas funciona da mesma forma. Lembrando que tudo que está dentro das aspas ('simples' ou "dupla") funcionará como uma string.

Se precisarmos adicionar um item da lista para uma variável para realizar alguma outra função basta usarmos o comando:

```
produto1 = listadecompras[0]
produto2 = listadecompras[1]
produto3 = listadecompras[2]
produto4 = listadecompras[3]
```

Neste caso estamos atribuindo a posição 0 (Leite) da lista que foi nomeada como "listadecompras" à uma variável nomeada produto1 e assim por diante.

Muitas vezes quando estamos fazendo a lista de compras do mercardo esquecemos de algum item e precisamos adiciona-lo depois, no python fazemos isso através do comando .append(), que é mostrado a seguir:

```
listadecompras.append("Chocolate")
listadecompras.append("Rocambole")
```

O função **append** permite que seja adicionado apenas 1 elemento a lista por vez, e para usá-lo basta entrar com o nome da lista a ser usada ".append" e como parâmetro o item a ser adicionado. Agora nossa lista de compras ficou assim:

Também temos os comandos para substituir ou remover algum item indesejado.

```
listadecompras[7] = "Bolacha"
```

Esse comando é responsável por substituir o item que está na posição 7 da lista (Rocambole) por um novo item (Bolacha).

```
del listadecompras[3]
```

Aqui estamos tirando da lista o item que está na posição 3 , no caso o molho de tomate.

Nossa lista de compras ficará assim:

Como resultado teremos a Bolacha no Lugar do Rocambole, e como deletamos um item da nossa lista, ela se tornará menor, e os itens que estavam depois do molho de tomate descerão uma posição.

Vamos imaginar que você e seu cônjuge tenham feito listas separadas

Para juntar essas duas listas em apenas uma, precisamos primeiro tê-las criado como vimos anteriorente, nesse caso usaremos lista1 e lista2, basta usar o seguinte comando:

Neste caso temos lista 3 como a lista resultante da junção da lista 1 com a lista 2

PARTE II - {DICIONÁRIOS}

Dicionários são parecidos com as listas, mas com a diferença que os elementos não são localizados por sua posição, mas sim com uma chave específica, veja o exemplo a seguir: Vamos supor que você more em uma mansão e tenha várias chaves, uma para cada cômodo, e como sua casa é muito grande você colocou etiquetas para não se esquecer.

Nesse caso temos as letras como chaves, e o seu valor correspondente é um cômodo. Quando se trata de dicionários em Python, para cria-lo precisamos dar um nome, nesse caso será "comodosmansao", e o que define que nossa coleção será um dicionário é o uso das chaves ("{}").

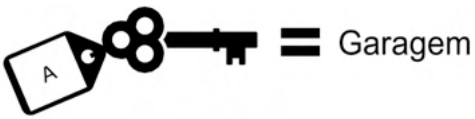
```
comodosmansao = { "A" : "Garagem", "B" : "Quarto 1", "C" : "Quarto 2",
"D" : "Adega" }
```

Quando passamos para Python as chaves ficarão antes dos dois pontos (":") e o seu respectivo valor ficará depois.

Uma vez que temos nosso dicionário criado, podemos usar as chaves para descobrir o seu valor, no nosso caso entraremos com as letras que estão nos chaveiros para descobrir qual cômodo a chave abre.

```
comodosmansao["A"]
```

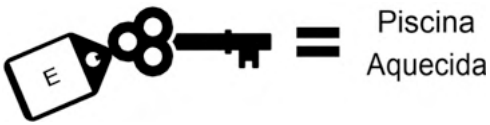
Então descobriremos que a chave "A" abre a garagem.



Podemos também estar adicionando novas chaves e valores ao dicionário usando o seguinte comando:

```
comodosmansao["E"] = "Piscina Aquecida"
```

Teremos então uma nova chave.



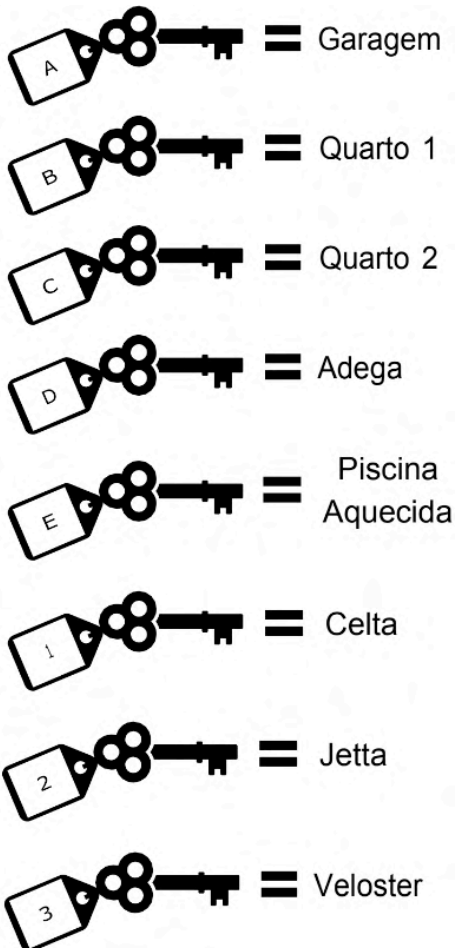
Como nas listas também podemos juntar dois dicionários diferentes. Vamos criar um dicionário novo e junta-lo com o que já temos.

```
carros = {1 : "Celta", 2 : "Jetta", 3 : "Veloster"}
```

Criamos um novo dicionário agora com os carros, e para juntar os dois usaremos o seguinte comando:

```
comodosmansao.update(carros)
```

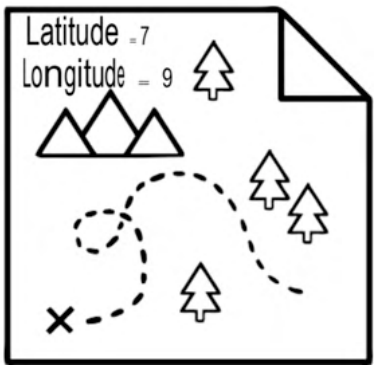
Então nosso dicionário final ficará assim:



PARTE III - (TUPLAS)

Vamos supor que você enterrou um tesouro e precisa guardar essas coordenadas para nunca se esquecer do local, se simplesmente usássemos listas ou dicionários, essas coordenadas poderiam ser alteradas ou até mesmo apagadas utilizando os comandos que vimos. A solução para isso é o uso das tuplas, que funcionam de forma similar as listas, mas com o detalhe que tuplas são imutáveis, o que quer dizer que uma vez criada, a tupla não pode ser alterada, não se pode adicionar nem remover itens.

Esse seu tesouro está enterrado na latitude 7 e na longitude 9 conforme vemos no mapa:



Para criarmos uma tupla e armazenar essas coordenadas super importantes devemos seguir o modelo que usamos para as listas, porém agora usando os parênteses.

```
tesouro = (7, 9)
```

Podemos por fim também transformar uma tupla em uma lista, permitindo assim sua modificação, através do seguinte comando:

```
lista_tesouro = list(tesouro)
```

O contrário também pode acontecer

```
tesouro1 = tuple(lista_tesouro)
```

PARTE 8 - ESTRUTURA DE REPETIÇÃO

PARTE I - WHILE

As estruturas de repetição nos permitem repetir um conjunto de instruções até que se atinja uma certa condição desejada, a primeira que veremos será o WHILE.

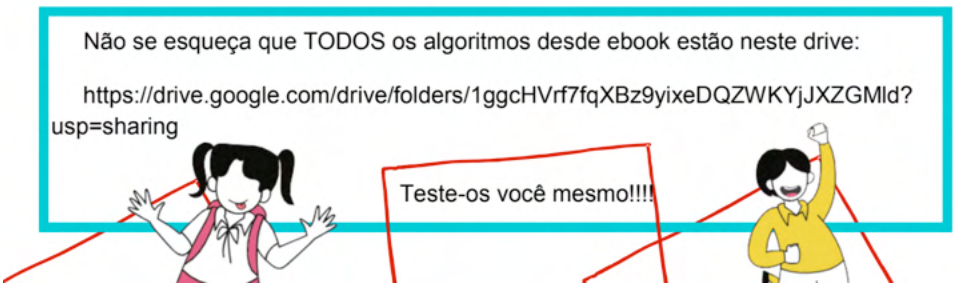
Se quisermos fazer um programa para contar até 5, para não precisarmos ficar escrevendo várias linhas de código usando print e colocando os números em ordem podemos usar o while para facilitar essa contagem.

Para entendermos o while, podemos traduzir a palavra para o português que seria referente a palavra enquanto, então essa nossa estrutura de repetição continuará funcionando enquanto sua variável não atingir o valor definido.

```
contador = 0
while (contador <= 5):
    print(contador)
    contador = contador + 1
else:
    print("contagem concluida")
```

O código acima é referente a nossa contagem de 0 a 5, e para isso precisamos seguir as seguintes etapas:

1. Definir a variável;
2. Uso do while, enquanto contador for menor ou igual a 5 a repetição continuará;
3. Mostra o valor;
4. Incrementa nossa variável;
5. Quando atingir o valor, a repetição cessará e será mostrada essa mensagem.



PARTE II - FOR

O comando For é a outra estrutura de repetição do Python. Sua principal característica é percorrer variáveis, geralmente, variáveis compostas.

O grande druida Fábio (membro do Conselho dos Magos Independentes, Registro nº43142) partiu em busca de elementos novos para suas poções de crescimento florestal.

Sua pesquisa extensiva resultou em 4 ingredientes vitais: 'sementes_macieira' 'água' 'adubo' 'minhocas'

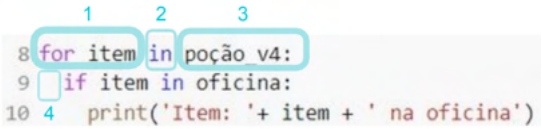
Fábio, à procura dos itens, percebe sua oficina extremamente desorganizada:

```
oficina = ('semente_macieira','cajado quebrado', 'espada_cega', 'água', 'roupão',
           'Ferrari_que_com_certeza_não_foi_roubada', 'tesouro', 'pote', 'fornalha',
           'meio_metro_de_corda,', 'chapéu','minhocas', 'sacola_de_papel',
           'sapato_biodegradável', 'adubo')
```

O mago então lança um feitiço na linguagem Python das máquinas para encontrar os itens para ele:

```
poção_v4 = ['semente_macieira', 'água', 'adubo', 'minhocas']

oficina = ('semente_macieira','cajado quebrado', 'espada_cega', 'água', 'roupão',
           'Ferrari_que_com_certeza_não_foi_roubada', 'tesouro', 'pote', 'fornalha',
           'meio_metro_de_corda,', 'chapéu','minhocas', 'sacola_de_papel',
           'sapato_biodegradável', 'adubo')

for item in poção_v4:
  if item in oficina:
    print('Item: '+ item + ' na oficina')
```

Essa estrutura do Python funciona atribuindo um valor da variável, no caso do druida uma lista, a uma variável auxiliar temporária e a cada varredura, o for atualizará essa temporária com o próximo item da lista.



1 - Variável auxiliar temporária

2 - Palavra chave in

3 - Variável onde o for fará varredura

4 - Indentação

De forma simplificada, o for funciona da seguinte forma:

No primeiro momento teremos a variável "item" sendo atribuída com o primeiro item da variável "poção_v4".

```
8 for item in poção_v4:
9   if item in oficina:
```

```
1 poção_v4 = ['semente_macieira', 'água', 'adubo', 'minhocas']
```

Item = 'semente_macieira'

Então, após todos os comandos que estiverem dentro da indentação do for forem realizados, ele atribuirá o próximo item da lista à variável auxiliar:

Item = 'água'

Após a realização do último item da lista o looping acabará automaticamente, seguindo para o próximo comando do algoritmo.

O for permite a utilização de variáveis simples e compostas para checar, porém permite apenas variáveis que contenham STRING. Variável INT, FLOAT e BOOL não são operadas pelo for, resultando em erro na execução do algoritmo.

Apesar dessa infeliz característica, podemos "Burlar" essa regra colocando essa variável em forma de STRING, portanto sua característica será STRING e sua aparência

será de outro tipo de variável.

Variável simples

Quando utilizamos variáveis simples, o for considerará um caractere por vez até o último da palavra.

```
1 var = 'banana'
2 for item in var:
3   print('variável simples: '+ item)
```

```
variável simples: b
variável simples: a
variável simples: n
variável simples: a
variável simples: n
variável simples: a
```

Variável simples com string

```
1 var = '154'
2 for item in var:
3   print('variável simples: '+ item)
```

```
variável simples: 1
variável simples: 5
variável simples: 4
```

Variável simples com string e sendo número

(Tuplas) e [Listas]

As tuplas e listas não possuem nenhuma diferença prática quanto ao for, dependendo apenas da aplicação do usuário:

```
1 tupla = ('Sal', 'True', 'pires')
2 for item in tupla:
3   print('tupla: '+item)
```

```
tupla: Sal
tupla: True
tupla: pires
```

(Tupla)

```
1 lista = ['Silêncio', 'Configuração', 'Marácas']
2 for item in lista:
3   print('tupla: '+item)
```

```
tupla: Silêncio
tupla: Configuração
tupla: Marácas
```

(Lista)

{Dicionários}

Com os dicionários temos mais variações de uso com o uso de seus métodos internos. Podemos fazer com que o for interaja apenas com as chaves ou apenas com os valores ou ambos:

Utilizando apenas com as chaves

Para o for trabalhar apenas com as chaves do dicionário utilizamos o método `.keys()` após o dicionário.

```
[42] 1 dicionário = {'nome':'Ervaldo',
     2              'profissão':'professor',
     3              'local_de_trabalho':'FATEC Itaquera'}
     4
     5 for chave in dicionário.keys():
     6   print(chave)
```

```
nome
profissão
local_de_trabalho
```

Utilizando apenas com os valores

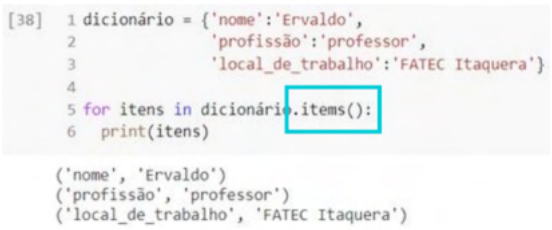
Para o for trabalhar apenas com os valores do dicionário utilizamos o método `.values()` após o dicionário.

```
[41] 1 dicionário = {'nome':'Ervaldo',
     2              'profissão':'professor',
     3              'local_de_trabalho':'FATEC Itaquera'}
     4
     5 for valor in dicionário.values():
     6   print(valor)
```

```
Ervaldo
professor
FATEC Itaquera
```

Utilizando chaves e valores

Para o for trabalhar apenas com as chaves e os valores do dicionário utilizamos o método `.items()` após o dicionário.

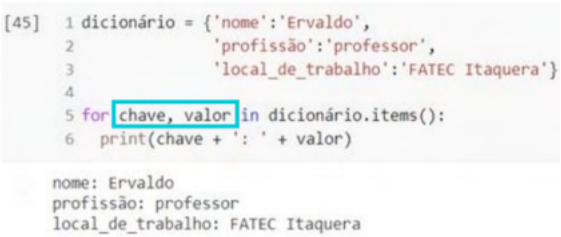


```
[38]  1 dicionário = {'nome':'Ervaldo',
      2               'profissão':'professor',
      3               'local_de_trabalho':'FATEC Itaquera'}
      4
      5 for itens in dicionário.items():
      6   print(itens)

('nome', 'Ervaldo')
('profissão', 'professor')
('local_de_trabalho', 'FATEC Itaquera')
```

Utilizando chaves e valores de forma bonita

Dicionários são úteis, mas não tão bonitos quando damos print() nos itens dele (imagem acima dirá tudo), portanto, para dar uma melhor aparência a seu código podemos utilizar duas variáveis diferentes, cada uma representando o item e seu valor respectivo.

É necessário apenas atribuir para chave e valor respectivamente na variável auxiliar temporária:

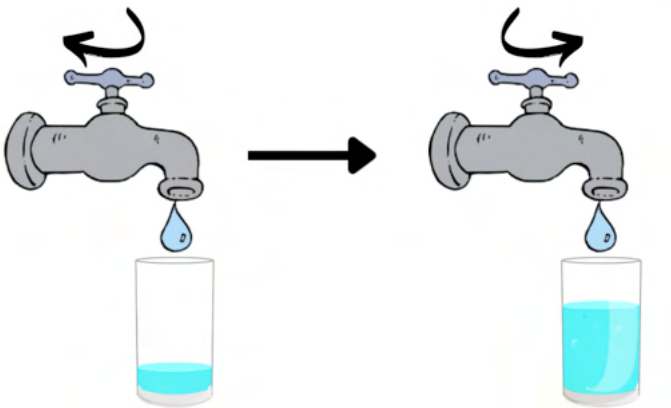


```
[45]  1 dicionário = {'nome':'Ervaldo',
      2               'profissão':'professor',
      3               'local_de_trabalho':'FATEC Itaquera'}
      4
      5 for chave, valor in dicionário.items():
      6   print(chave + ': ' + valor)

nome: Ervaldo
profissão: professor
local_de_trabalho: FATEC Itaquera
```

PARTE 9 - FUNÇÕES

As funções são blocos de excução em que uma sequência de eventos ocorre, entretanto a função possui a funcionalidade de agir apenas quando é solicitada.

Em um exemplo, quando necessitamos beber água devemos seguir uma sequência de procedimentos, como por exemplo abrir a torneira para dar vazão a água e fechá-la quando atingirmos a quantidade adequada. Uma tarefa simples, que pode ser separada em uma função devido a quantidade de vezes que realizamos essa tarefa.



Em funções podemos fornecer argumentos a serem utilizadas durante a execução dessa função e receber resultados.

Mas parando de conversa, vamos a estrutura de uma função, e como sempre, utilizando a plataforma google collab.

```
def PrimeiraFuncao():
  print('Isso eh uma funcao')

PrimeiraFuncao()
```

```
Isso eh uma funcao
```

Pontos a serem destacados: "def" indica a criação de uma função, "PrimeiraFuncao()" corresponde ao nome da função, como ela deverá ser chamada na linha de código para ser utilizada. Após os dois pontos e na linha de baixo, com a utilização da indentação temos as instruções a serem realizadas.

Exemplo do copo de água e torneira:

O código acima apresenta uma função hipotética para beber água, no caso, a etapa de encher o copo, o código apresenta uma condição de teste quanto a condição atual do copo, se ele já estiver cheio o código retorna ao usuário que o copo já estava cheio, caso contrário retorna a mensagem de que o copo foi enchido.

Alguns pontos a serem reforçados:

1. Declaração do inicio de uma função
2. Nome da função
3. Argumento
4. Retorna o valor da variável copo como resultado da função
5. Parâmetro "vazio" sendo enviado para dentro função.

É de suma importância compreender que a variável trabalhada dentro da função é uma variável LOCAL, o que significa que ela não retornará o valor em nenhum outro ponto do código sem a utilização do método return e armazenamento adequado em outra variável.

```
[2]  def Contas(val1,val2):
       soma = val1+val2
       subtracao = val1-val2
       divisao = val1/val2
       multiplicacao = val1*val2
       return soma, subtracao, divisao, multiplicacao

     som, sub, divi, mult = Contas(1,2)
     print(som,sub,divi,mult)

     3 -1 0.5 2
```

Número de argumentos desconhecidos, para isso devemos modificar um pouco a estrutura da declaração, com o acréscimo de um '*' antes do nome do argumento na função.

```
[1]  def MuitosArgumentos(*nomes):
       val = str(len(nomes))
       print('Foram inseridos ' + val + ' Nomes')
       print('O primeiro nome inserido foi ' + nomes[0] )

     MuitosArgumentos('Carlos', 'Maria', 'Luis', 'Pedro')

     Foram inseridos 4 Nomes
     O primeiro nome inserido foi Carlos
```

A função "len(x)" é uma função nativa do Python e o que ela faz é contar a quantidade de itens dentro da lista nomes, ferramenta importante para controlar a quantidade de valores inseridos.

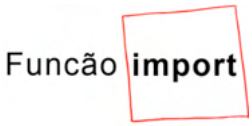
Além dessa função existem inúmeras outras, que podem ser visualizadas e estudadas a fundo no link abaixo.

https://docs.python.org/pt-br/3.6/library/functions.html

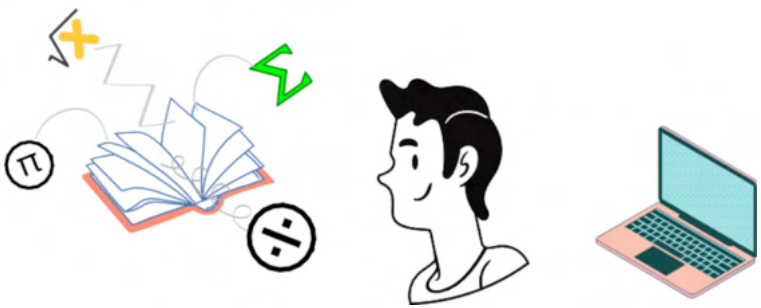
PARTE 10 - MÓDULOS

Vimos em capítulos anteriores que para se ter maior dinâmica com um algoritmo escrito, podemos salvá-lo em um arquivo (com extensão ".py" no final), que pode ser útil para desenvolvimento de projetos que demandarão muito tempo para conclusão.

Python é o melhor amigo do desenvolvedor de códigos longos, pois permite que o programador reutilize FUNCÕES já escritas anteriormente em códigos diferentes!



A ideia central por trás dessa função é exatamente a de "Chamar" o conjunto de funções desejadas.



Roberto é um professor de matemática e tem o costume de dar 10 minutos para que cada aluno realize as tarefas propostas.

Roberto passou uma atividade, porém percebeu que não estava com seu relógio, portanto, em um momento de iluminação, ele lembrou que poderia importar um MÓDULO que o permitiria escrever rapidamente um código para marcar o tempo dos alunos. Após alguns minutos olhando a documentação oficial do MÓDULO Time (https://docs.python.org/pt-br/3/library/time.html) encontrou uma FUNÇÂO que poderia ajudar e desenvolveu o seguinte código:

```
1 import time
2
3 qntd_tempo = int(input('Digite o tempo desejado em minutos: '))
4 segundos = qntd_tempo * 60
5
6 time.sleep(segundos)
7
8
9 print('O tempo acabou!!!!')
10
```

Analisando o código de Roberto, que, em primeiro momento na linha 1, ele fez a importação do módulo Time para seu código utilizando a palavra chave import e, após isso, o nome do módulo.

```
1 import time
```

Depois do programa receber o tempo que o usuário deseja contar, houve a contagem propriamente dita:

```
5 time.sleep(segundos) # conta o tempo
```

Para utilizar as funções que estão no módulo utiliza-se um ponto após o nome do módulo, por isso é importante conhecer as funções do módulo através da documentação oficial.

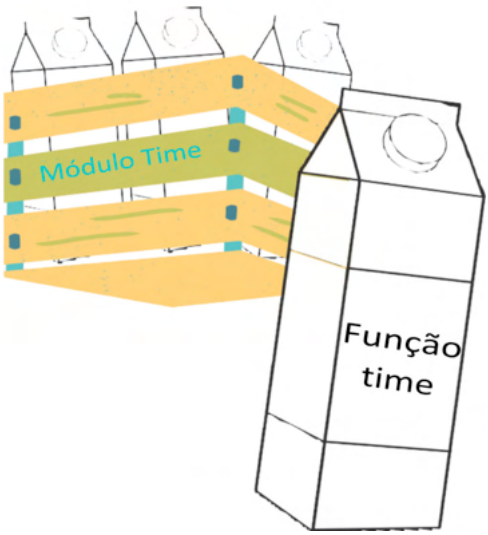
Para acessar, testar e modificar o teste de Roberto no Google Colab acesse: https://colab.research.google.com/drive/10fguzd1OHBmByntZ5WQFJRHe2_gNsfAX?usp=sharing

Quando importamos um módulo, habilitamos no nosso código todas as funções presentes nele, o que pode sobrecarregar o processamento e aumentar tempo do código. Essa noção é importante quando trabalhamos com módulos de inteligência artificial como o Scikit-learn pois apresentam diversas funções diferentes que nem sempre serão usadas nos algoritmos.

Para importar funções específicas de um módulo, utiliza-se a palavra chave FROM em conjunto com o IMPORT.

```
[ ]  1 from time import sleep, clock_gettime
```

Quando importarmos mais de uma função, faz-se o uso de vírgula para a separação.



Também podemos mudar o nome do módulo e da função para algo mais conveniente para nós, desenvolvedores. Conseguimos fazer isso utilizando a palavra chave AS.

```
[ ]  1 import time as tm
```

Assim como podemos, também, importar as funções com nome personalizado.

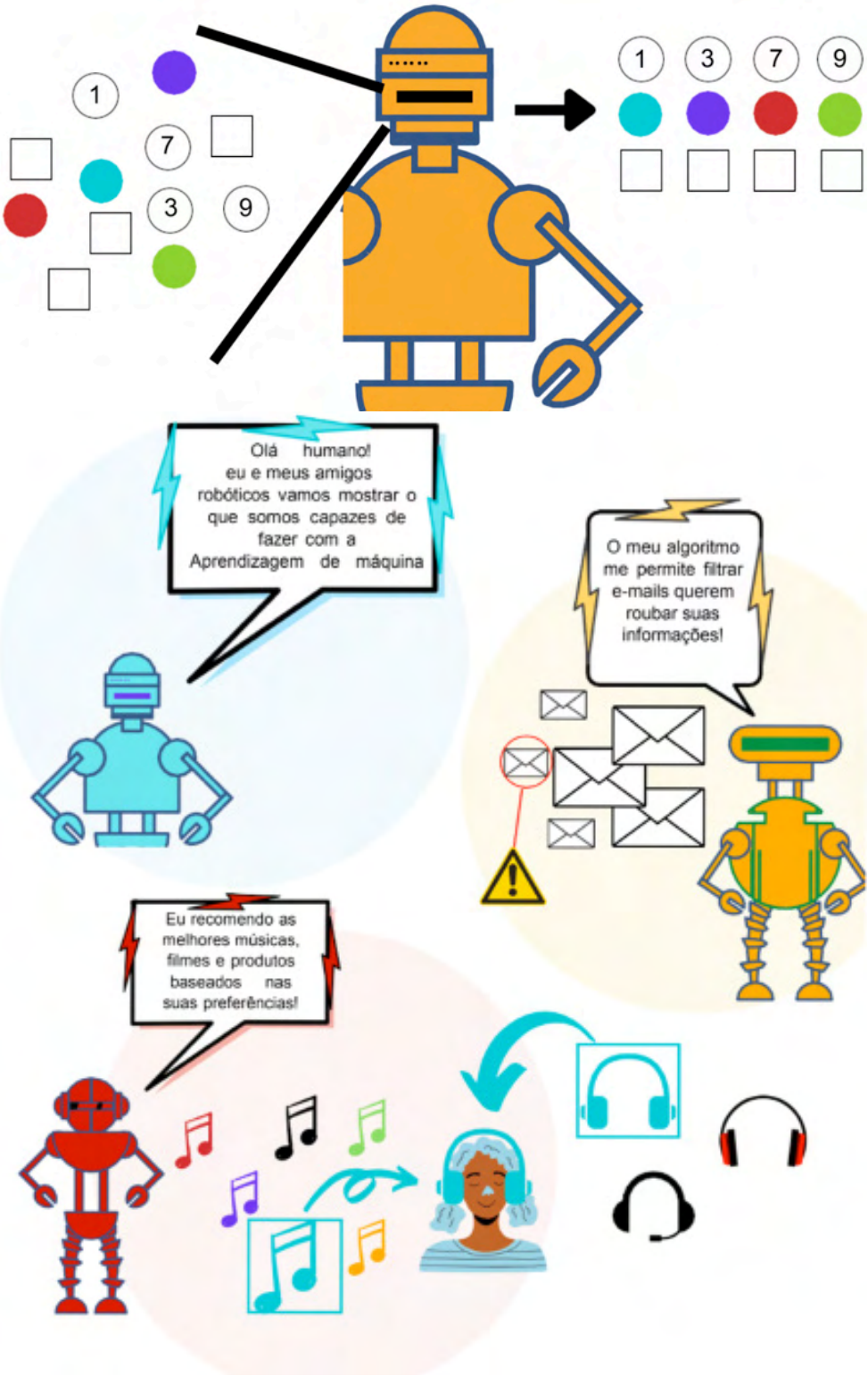
```
[ ]  1 from time import sleep as sp
```

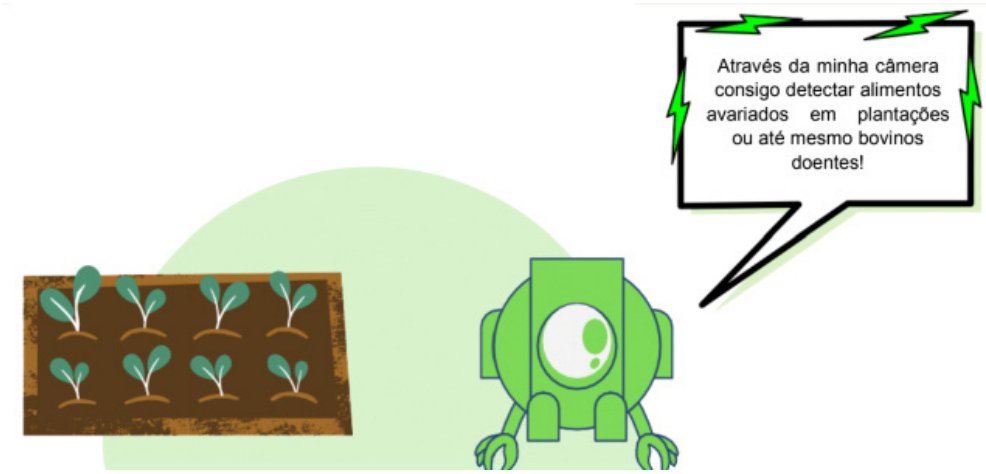

o que é aprendizado?
A professora Tamires fez a pergunta acima para sua classe e recebeu diversas respostas:
Executar determinada tarefa com repetição
Quando ganhamos conhecimentos ou habilidades a partir de estudos
Analisar padrões e extrair conclusões
Estão corretos, portanto, podemos definir como "Aprendizagem é o processo de extração de informações de experiências e é incorporada ao nosso repertório em forma de conhecimentos e habilidades"

A professora Tamires perguntou à Jéssica o que vem à mente dela quando mencionamos o seguinte termo:

Ela fez menção de algo sobre Robôs inteligentes do futuro. Para surpresa dela, ela estava parcialmente certa:

> Aprendizado de máquina é um ramo que parte do campo de inteligência artificial que permite a análise de um conjunto de dados ou informações e o seu tratamento personalizado sem ser necessária a programação explícita de seus atos [3].


1
7
3
9
1
3
7
9
Olá humano! eu e meus amigos robóticos vamos mostrar o que somos capazes de fazer com a Aprendizagem de máquina
O meu algoritmo me permite filtrar e-mails querem roubar suas informações!
Eu recomendo as melhores músicas, filmes e produtos baseados nas suas preferências!


Através da minha câmera consigo detectar alimentos avariados em plantações ou até mesmo bovinos doentes!

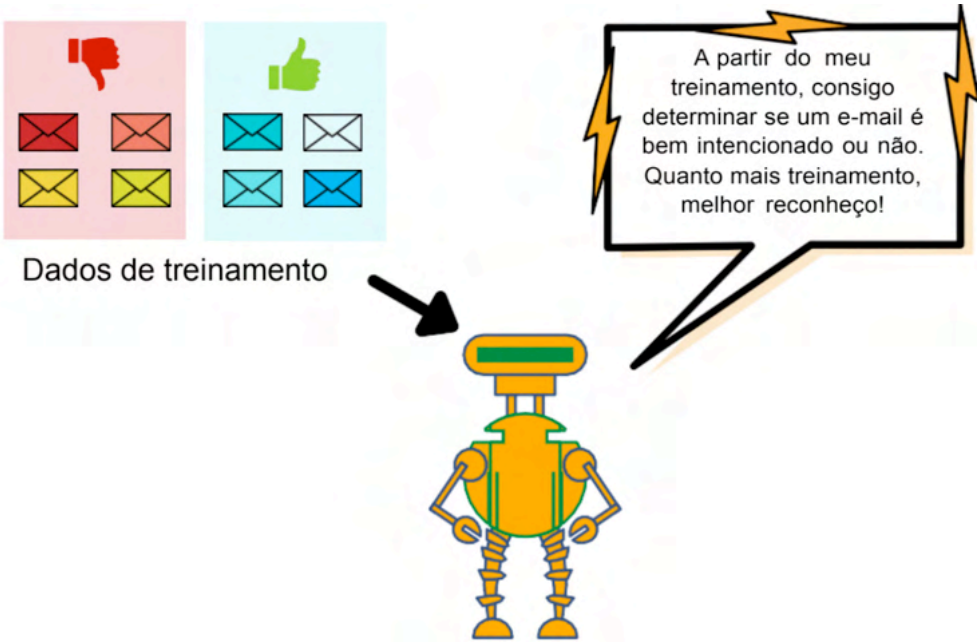
TIPOS DE APRENDIZADO DE MÁQUINA

O aprendizado de máquina é um tema muito abrangente e que possui diversas aplicações nas mais diversas áreas. Mediante isso, existem técnicas diferentes que são utilizadas dependendo do tipo de problema a ser solucionado, sendo assim, elas foram agrupadas em métodos de aprendizagem juntamente com seus modelos de programação.

APRENDIZADO SUPERVISIONADO

O aprendizado supervisionado tem como principal característica a necessidade de um modelo base para que possa retornar uma previsão [3], precisamos fornecer um "dataset", que possui os dados de entrada e seu respectivos dados de saída, ou seja precisamos fazer a pergunta e dizer a resposta desejada, para que a partir disso o modelo escolhido possa realizar suas previsões [4].

Classificação: A classificação tem como função prever um item a partir de uma lista de possibilidades, retornando um resultado discreto [5]. Um exemplo de classificação é a separação de e- mails que são SPAM, uma vez que o algoritmo aprendeu as características que definem um e-mail como SPAM, ele já será que capaz de definir um novo e-mail como SPAM ou não.

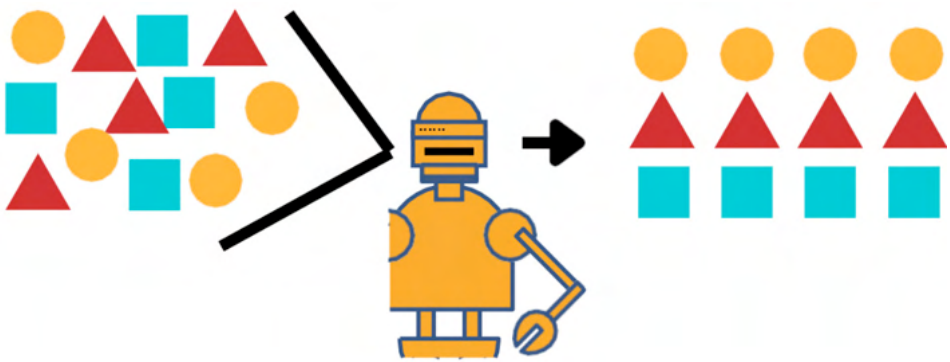


Regressão: A regressão funciona de forma similar a classificação para determinar uma resolução, porém ela não se limita a resultados binários, abrangendo assim uma infinidade de números. Um exemplo é a predição do salário de uma certa área de trabalho

levando em conta, variáveis como idade e tempo de trabalho [6].

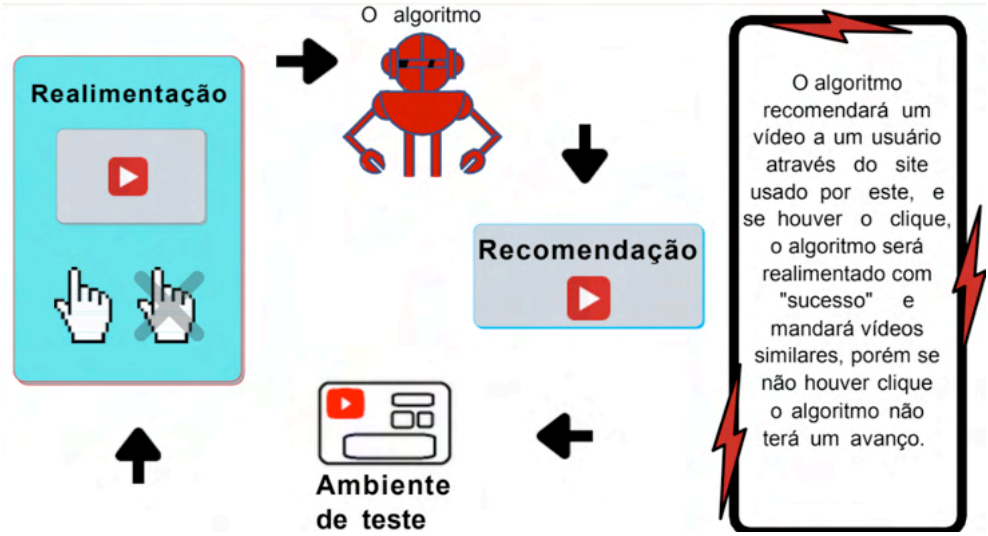
APRENDIZADO NÃO SUPERVISIONADO

O aprendizado não supervisionado não tem uma resposta definida, então o algoritmo analisa os dados a fim de encontrar padrões entre eles fazendo separações [6]. Esse tipo de modelo é chamado de agrupamento e pode ser usado por exemplo para separar frutas conforme seu formato. Como nesse tipo de aprendizado o algoritmo trabalha sozinho, dependendo do tamanho do conjunto de dados, ele pode se tornar um pouco mais demorado [5].



APRENDIZADO REFORÇADO

O aprendizado reforçado faz com que o modelo faça decisões por si próprio, e é recompensado de acordo com o progresso que faz na atividade em que foi designado. Em outras palavras podemos dizer que o aprendizado reforçado funciona através da tentativa e erro e a medida que vai ganhando experiência e eficiência quando tem sucesso nas escolhas [3]. O aprendizado reforçado pode ser usado para recomendação de músicas e filmes, o algoritmo irá recomendar uma música ou filme relacionado ao que você costuma consumir, e caso você ouça a música ou assista o filme, o algoritmo entenderá que foi uma boa recomendação.


O algoritmo
Realimentação
Recomendação
Ambiente de teste
O algoritmo recomendará um vídeo a um usuário através do site usado por este, e se houver o clique, o algoritmo será realimentado com "sucesso" e mandará vídeos similares, porém se não houver clique o algoritmo não terá um avanço.

PRINCIPAIS BIBLIOTECAS

Para construirmos soluções utilizando o aprendizado de máquina por meio da linguagem python faz-se necessário a utilização de bibliotecas que possibilitem trabalharmos com os dados presentes em um dataset ou até mesmo bibliotecas capazes de apresentar ao usuário uma representação gráfica do que foi executado ou dos resultados obtidos.

Criado em 2007 como um projeto do Google Summer of Code o scikit-learn consiste em uma biblioteca que encontra-se em constante melhoramento, disponível para estudo e utilização em aplicações de aprendizado de máquina, dentro desse pacote podemos encontrar uma vasta quantidade de algoritmos de aprendizado de máquina, assim como uma documentação muito bem desenvolvida sobre cada algoritmo. Abaixo está o link para o site da biblioteca.

https://scikit-learn.org/stable/tutorial/index.html

Numpy é uma biblioteca muito importante para a área científica e matemática presente no Python, criada em 2005 esse conjunto de pacotes possibilita a construção de matrizes e arrays, bem como operações matemáticas, modelagem da estrutura, ordenação, seleção, álgebra linear básica, operações estatísticas básicas, simulações aleatórias e muito mais. O numpy é vastamente utilizado devido ao formato de array criado pela biblioteca ser de fácil utilização em outras bibliotecas, o famoso ndarray.

https://numpy.org/doc/stable/index.html

Com objetivo de construir gráficos, a biblioteca matplotlib é capaz de fornecer ao

usuário ferramentas como a construção de gráficos de linha, histogramas, gráficos de dispersão e diversas outras ferramentas visuais que podem ser utilizadas para obtenção de conclusões quanto ao desempenho ou melhor apresentação de um determinado conjunto de dados.

https://matplotlib.org

pandas

Pandas é uma biblioteca que serve para tratamento de dados e análise, de forma resumida o pandas possui um estruturamento de dados que possibilita a utilização de diversos tipos de dados em uma mesma tabela a ser trabalhada por outras bibliotecas, o pandas apresenta uma diferença quanto ao numpy, visto que a lista construída pelo numpy se delimita a apenas um tipo de dado. O pandas possibilita ainda a utilização de diversos formatos de banco de dados, como SQL, Excel e arquivos CSV. https://pandas.pydata.org

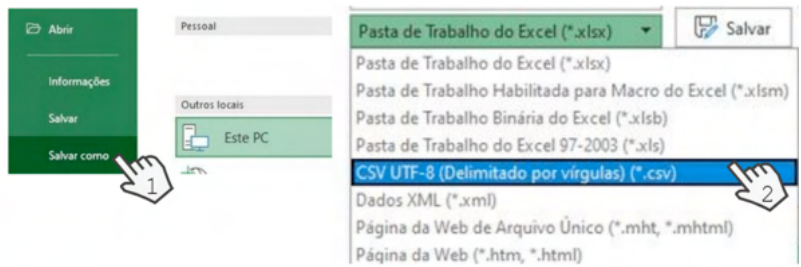
CONJUNTO DE DADOS

Quando estamos trabalhando com a biblioteca Pandas para usar nosso dataset usamos os arquivos com extensão ".csv" que vem do inglês "comma separated values", o que em português significa valores separados por vírgulas, vamos ver como podemos gerar e manipular esse tipo de arquivo, e aproveitando que estamos entrando nesse assunto já vamos ensinar a importar o dataset para o google colab.

Para gerar um arquivo com a extensão ".csv", basta abrir o excel ou o software utilizado para formatação de tabelas e ao invés de separar os dados por coluna, separa-los por virgulas como mostra a imagem abaixo.

```
idade,genero,estilo_musical
20,1,Rap
23,1,Rap
25,1,Rap
26,1,Jazz
29,1,Jazz
```

Quando os dados já estiverem devidamente separados, basta ir em "salvar como" e selecionar a extensão ".csv" como mostra a figura abaixo



Para importar o dataset para o Google Colab basta entrar no arquivo desejado do Google Colab, ir no canto esquerdo e clicar no ícone de uma pasta, abrindo assim a aba arquivos, e basta arrastar o arquivo ".csv" para essa aba.

Com os arquivos já importados, o próprio google colab nos permite ver como as tabelas ficaram, e com isso é importante se atentar à um detalhe. As imagens abaixo demonstram dois arquivos com extensão ".csv" formatados de formas diferentes, e seus respectivos resultados quando abrimos no Google Colab.

idade	genero	estilo_musical
20	1	Rap
23	1	Rap
25	1	Rap
26	1	Jazz
29	1	Jazz
30	1	Jazz
31	1	MPB
33	1	MPB
37	1	MPB
20	0	Funk
21	0	Funk
25	0	Funk
26	0	Sertanejo
27	0	Sertanejo
30	0	Sertanejo
31	0	MPB
34	0	MPB
35	0	MPB

Excel

idade,genero,estilo_musical
20,1,Rap
23,1,Rap
25,1,Rap
26,1,Jazz
29,1,Jazz
30,1,Jazz
31,1,MPB
33,1,MPB
37,1,MPB
20,0,Funk
21,0,Funk
25,0,Funk
26,0,Sertanejo
27,0,Sertanejo
30,0,Sertanejo
31,0,MPB
34,0,MPB
35,0,MPB

x

idade;genero;estilo_musical
20;1;Rap
23;1;Rap
25;1;Rap
26;1;Jazz
29;1;Jazz
30;1;Jazz
31;1;MPB
33;1;MPB
37;1;MPB
20;0;Funk
21;0;Funk
25;0;Funk
26;0;Sertanejo
27;0;Sertanejo
30;0;Sertanejo
31;0;MPB
34;0;MPB
35;0;MPB

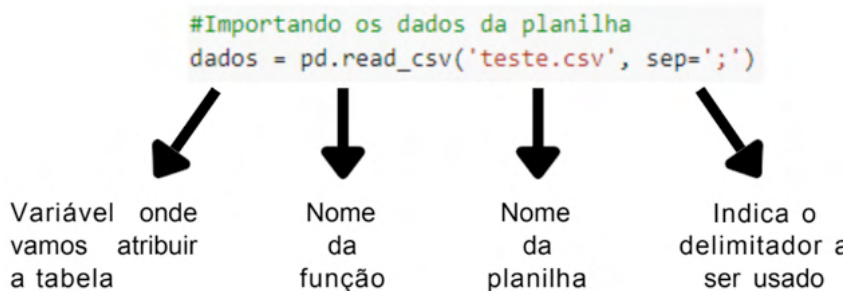
Google Colab

idade	genero	estilo_musical
20	1	Rap
23	1	Rap
25	1	Rap
26	1	Jazz
29	1	Jazz
30	1	Jazz
31	1	MPB
33	1	MPB
37	1	MPB
20	0	Funk
21	0	Funk
25	0	Funk
26	0	Sertanejo
27	0	Sertanejo
30	0	Sertanejo
31	0	MPB
34	0	MPB
35	0	MPB

Se pararmos para observar a tabela que antes estava formatada de forma "normal"

em diferentes colunas do Excel, na hora de importar para o Google Colab ela se tornou dividida por ponto e vírgula (";"), enquanto a que respeitou a divisão por vírgulas se mostrou como uma tabela formatada normalmente no Google Colab.

É possível ajustar ambos os arquivos com os dados para que sejam mais fáceis de ler e de tratar no algoritmo. Para tanto, necessita-se fazer uma configuração inicial no código na hora de fazer o "dataset" se transformar em "strings" tratáveis pelo Python, o comando utilizado para fazer isso é o "pd.read_csv".



O que precisamos nos atentar é quanto ao "sep", que define o delimitador das colunas, ou seja, o símbolo que é usado para separar uma coluna da outra.

O padrão é o uso da vírgula, então quando nosso arquivo é feito originalmente com vírgulas não precisamos declarar o "sep", já no caso da figura XX onde podemos ver a separação feita por ponto e vírgula, deveremos declarar o "sep" sendo ";".

ALGORITMO - ÁRVORE DE DECISÃO

O algoritmo que foi desenvolvido e será retratado a seguir, tem a função de predizer qual o estilo musical uma pessoa gosta, baseado em sua idade e seu gênero. Para isso foi utilizado o modelo de classificação, utilizando a técnica de "árvore de decisão".

Em pesquisas científicas, o ideal é sempre trabalhar com um número elevado de amostras para uma análise que represente uma maioria. Para o exemplo a seguir utilizaremos um conjunto de dados reduzido para tornar mais fácil o entendimento.

A tabela abaixo representa nosso conjunto de dados utilizado para desenvolver o algoritmo.

idade	genero	estilo_musical
20	1	Rap
23	1	Rap
25	1	Rap
26	1	Jazz
29	1	Jazz
30	1	Jazz
31	1	MPB
33	1	MPB
37	1	MPB
20	0	Funk
21	0	Funk
25	0	Funk
26	0	Sertanejo
27	0	Sertanejo
30	0	Sertanejo
31	0	MPB
34	0	MPB
35	0	MPB

Dados de entrada

Dados que serão utilizados como parâmetros e irão ser relacionados com os dados de saída e utilizados como base para fazer a predição.

Dados de saída

Valores que serão retornados ao usuário quando for alimentado com um valor de entrada.

Temos como dados de entrada a idade e o gênero, sendo o número "1" representando o gênero masculino e o "0" o feminino, e como dados de saída o estilo musical.

O algoritmo e o dataset se encontram no link abaixo:

https://drive.google.com/drive/u/0/folders/1qaaxcyzSOX3JwFPhnU g2AQ6A5RqWJf5y

Na primeira etapa do algoritmo ocorre a declaração das bibliotecas usadas no decorrer do código.

```
[1]  #Importando Bibliotecas
     import pandas as pd
     import matplotlib.pyplot as mpl
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import accuracy_score
     from sklearn import tree
```

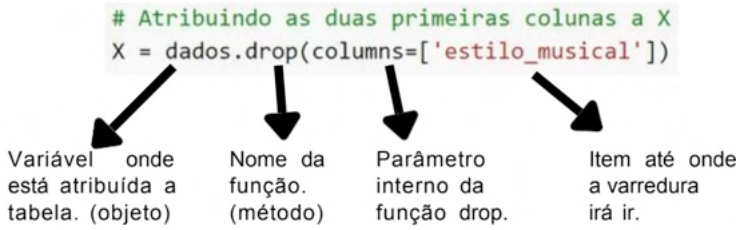
Após definir as bibliotecas, é importado o dataset (conjunto de dados) para o Google Colab, e fazer o algoritmo reconhecê-lo. O comando usado para permitir o tratamento dos dados no algoritmo é o "pd.read_csv()" e como parâmetro, dentro dos parênteses, inserimos o nome da planilha.

No nosso caso, atribuímos esse comando à variável "dados", esse comando é responsável por transformar os dados em Strings tratáveis no Python.

```
#Importando os dados da planilha
dados = pd.read_csv('musicas.csv')
```

Se tivéssemos usado todos os dados que temos para treinar o nosso algoritmo não saberíamos a precisão de predição dele, então é importante dividirmos um pouco dos dados para treinar o algoritmo e o resto para testar e verificar sua acurácia, portanto, existe a necessidade de se dividir os dados brutos em partes tratáveis, sendo elas os dados de entrada e de saída.

Para separar os dados de entrada, que são formados pelas duas primeiras colunas da tabela do conjunto de dados usamos a função abaixo:



Separar os dados de saída será mais simples, visto que é formado por apenas uma coluna.

Tudo precisou ser feito foi atribuir a coluna inteira a uma variável:

```
#Atribuindo a última coluna  a y
y = dados['estilo_musical']
```

Após esse primeiro tratamento de dados, obtemos as seguintes atribuições às variáveis X e y:

Dados de entrada (X)

```
1 #Mostra X
2 print(X)
```

	idade	genero
0	20	1
1	23	1
2	25	1
3	26	1
4	29	1
5	30	1
6	31	1
7	33	1
8	37	1
9	20	0
10	21	0
11	25	0
12	26	0
13	27	0
14	30	0
15	31	0
16	34	0
17	35	0

Dados de saída (y)

```
1 #Mostra y
2 print(y)
```

0	Rap
1	Rap
2	Rap
3	Jazz
4	Jazz
5	Jazz
6	MPB
7	MPB
8	MPB
9	Funk
10	Funk
11	Funk
12	Sertanejo
13	Sertanejo
14	Sertanejo
15	MPB
16	MPB
17	MPB

Para mais informações sobre a função "drop", acesse a documentação a partir do link abaixo:

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop.html

Utilizando os dados que foram separados em entrada e saída, faremos o treinamento do algoritmo usando a seguinte função do módulo Sklearn:

```
train_test_split(X, y, test_size = 0.2)
```

Essa função é responsável por alimentar o algoritmo com esses dados alterando

a ordem aleatoriamente.

Dentro do parênteses atribuímos as variáveis que queremos dividir (X, y), o "test_size" que define qual a porcentagem desses dados serão usados para treino e para teste. Atribuímos o que a função retorna para quatro variáveis, duas de teste e duas de treino, serão elas X_treino e X_teste para dados de entrada, e y_treino e y_teste para os de saída.

```
#Divindo os dados da planilha para váriaveis de treino e de teste
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size = 0.2)
```

Com os dados de treino e teste em mãos, enviamos esses dados para a árvore de decisão realizar a predição dos dados.

O primeiro passo é armazenar a função DecisionTreeClassifier em uma variável para tratamento:

```
#Define o modelo como árvore de decisão
modelo = DecisionTreeClassifier()
```

Com o modelo definido podemos começar a treina-lo, o comando usado será o .fit(), onde devemos preencher os parênteses com os dados de treino, tanto entrada quanto saída.

```
#Treina o modelo
modelo.fit(X_treino,y_treino)
```

Com o algoritmo treinado já podemos começar a fazer predições, e as primeiras que vamos fazer vão ser em relação as variáveis de entrada responsáveis pelo teste, para podermos calcular a acurácia, a função responsável por fazer a predição é o ".predict()", onde dentro dos parênteses devemos inserir os dados de entrada.

```
# Faz a predição usando os dados de entrada de X_teste
predição=modelo.predict(X_teste)
```

Para calcular a acurácia vamos usar o comando "accuracy_score()", onde dentro dos parênteses vamos colocar o resultado da nossa predição, e os reais valores que foram armazenados em y_teste.

```
#Compara os dados feitos pela predição com os dados reais de saída de y_teste
acurácia=accuracy_score(y_teste, predição)
print(acurácia)
```

A acurácia se trata da proximidade entre o resultado que foi obtido pelo nosso algoritmo e o valor "real", que no caso refere-se aos nossos valores de teste, onde o valor da nossa acurácia varia de 0 a 1, onde mais próximo de 1 melhor a acurácia.

Para analisarmos nosso algoritmo vamos pedimos para fazer uma predição de um homem de 21 anos e uma mulher de 22 anos, para isso usamos de novo o comando "modelo.predict", mas agora inserindo respectivamente idade e gênero, sendo 1 para homem e 0 para mulher.

```
#Faz a predição para um homem de 21 anos e para
#uma mulher de 22 anos
prediçãonova = modelo.predict ([[21,1],[22,0]])
print(prediçãonova)
```

```
['Rap' 'Funk']
```

A predição do nosso algoritmo foi que o homem de 21 anos provavelmente gosta de Rap e a mulher de 22 anos provavelmente gosta de Funk, se nós analisarmos nossa tabela veremos que não temos nenhum registro para homem de 21 anos nem para mulher de 22 anos, já para homens entre 20 e 25 anos o estilo musical é o Rap e para mulheres entre 20 e 25 o estilo musical e o Funk, o que demonstra que a predição feita pelo nosso algoritmo através da árvore de decisão faz sentido.

idade	genero	estilo_musical
20	1	Rap
23	1	Rap
25	1	Rap
26	1	Jazz
29	1	Jazz
30	1	Jazz
31	1	MPB
33	1	MPB
37	1	MPB
20	0	Funk
21	0	Funk
25	0	Funk
26	0	Sertanejo
27	0	Sertanejo
30	0	Sertanejo
31	0	MPB
34	0	MPB
35	0	MPB

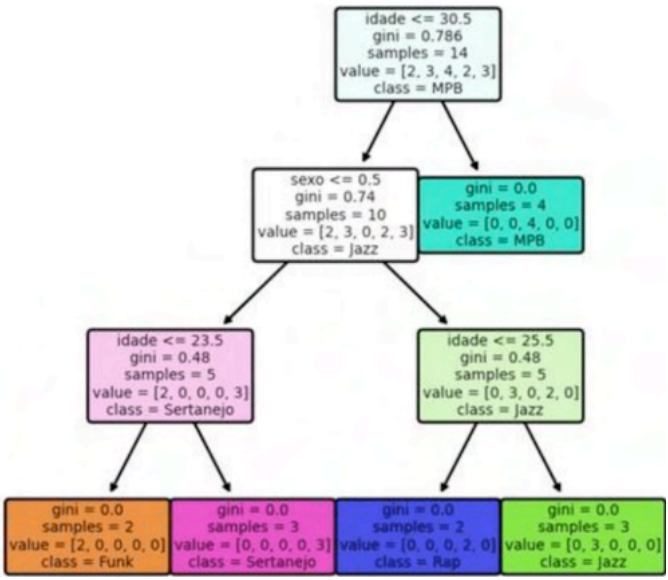
Para gerar uma visualização gráfica das tomadas de decisão do algoritmo utilizamos a função "tree.plot_tree()", onde como parâmetro inserimos a variável onde nosso modelo se encontra.

- "Feature_names": é responsável por nomear os recursos;
- "class_names": nomeia os estilos musicais, o comando sorted é o responsável por dividir a nossa váriavel y onde se encontram
- os estilos musicais, enquanto "unique" garante que não haverão nomes repetidos;
- "rounded": trata apenas do visual, onde ao ser habilitado ele tornará as caixas de nó arredondadas e mudará a fonte da letra;
- "filled": também trata apenas da parte estética, onde quando habilitado colore as caixas de nó.

O comando "mpl.subplots" é responsável por ajustar o tamanho da figura, onde "figzise" determina o tamanho da figura em polegadas, e o "dpi" que significa Dots per inches, determina quantos pixels a figura possui.

```
#Ajusta o tamanho da figura
mpl.subplots(figsize = (4,4), dpi=200)

# Gera a visualização gráfica da árvore de decisão
tree.plot_tree(modelo,
               feature_names=['idade','sexo'],
               class_names =sorted(y.unique()),
               rounded=True,
               filled=True)
```

Executando essa célula do código nós obteremos a árvore de decisão que é usada pelo algoritmo, lembrando que toda vez que você rodar o código a árvore pode se tornar diferente visto que os dados de treino e de teste mudam.



A primeira linha presente na árvore de decisão representa uma comparação entre a idade fornecida e um valor que nosso algoritmo gerou, dessa forma se a informação se mostrar verdadeira o fluxo da árvore continua para a direita, enquanto se for falsa vai para a esquerda, e o mesmo acontece para as próximas células, até encontrar o valor ideal para tal predição.

Temos também "gini" que consiste no parâmetro responsável por mensurar a impureza da célula de acordo com a progressão das etapas, onde a divisão das células e separação de classes se encerra apenas quando o valor do gini atingir 0, o que significa que a célula retém apenas objetos de mesma classe.

"Sample" mostra a quantidade de amostras presentes naquela etapa, enquanto "value" demonstra através de uma lista as quantidades de amostras das classes restantes na célula, e por final "class" demonstra a classe a qual o valor pertence.

REFERÊNCIAS

LUTZ, Mark. ***Python Pocket Reference***. 5. ed. EUA: O'Reilly, 2014.

BARRY, Paul. ***Head First Python***. 2. ed. [S. l.]: O'Reilly, 2015.

BUKOV, Andriy. ***The hundred-page Machine Learning***. [S.l]. Leanpub, 2019.

HARRINGTON, Peter. ***Machine Learning in Action***. Nova Iorque: MANNING, 2012.

RAVI, Ajay; REBALA, Gopinath; CHURIWALA, Sanjay. ***An Introduction to Machine Learning***. Suíça: Springer, 2019.

SHALEV-SWARTZ, S.;BEN-DAVID, S. ***Understanding Machine Learning:*** From Theory to Algorithms. New York, EUA: Cambridge University Press, 2014.

PERSONAGENS AUXILIARES

Carlos

Mariana

Vladimir

Cátia e Márcia

Casal que foi ao mercado

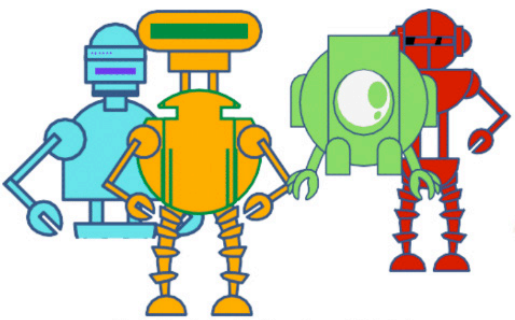
Ernesto Bom-De-Baixo

Josiel

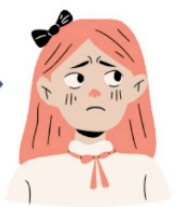
Tamires

Classe da professora Tamires

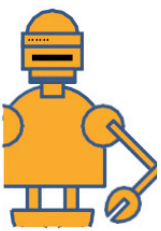
Roberto



Robôs inteligência artificial



Jéssica



Robô organizador

Helena

Damares

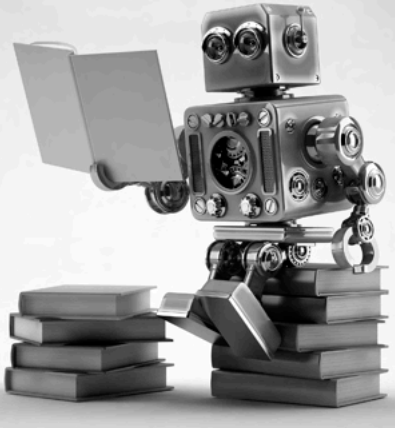
Francisco

Druida Fábio

www.atenaeditora.com.br
contato@atenaeditora.com.br
@atenaeditora
www.facebook.com/atenaeditora.com.br

APRENDIZADO DE MÁQUINA

COM PYTHON 3

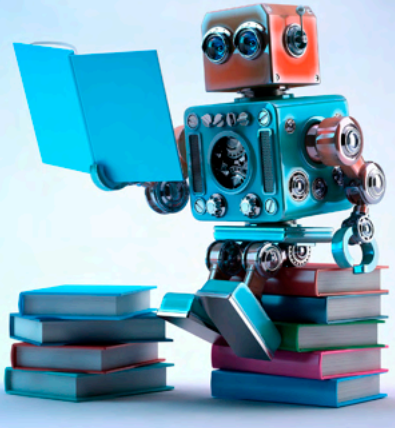


Atena Editora
Ano 2022

www.atenaeditora.com.br
contato@atenaeditora.com.br
@atenaeditora
www.facebook.com/atenaeditora.com.br

APRENDIZADO DE MÁQUINA

COM PYTHON 3



Atena Editora
Ano 2022