01 – Aprensentado o curso:

**01Introdução**

Olá! Meu nome é **Rodrigo Dias** e venho lhes dar as boas vindas ao curso **Python para Data Science: Funções, Pacotes e Pandas básico**, criado para você que está iniciando no mundo da ciência de dados e ainda não possui experiência com a linguagem Python. Terminamos a primeira parte do treinamento discutindo sobre a biblioteca **Numpy**, bastante importante para data science, e agora continuaremos nossos estudos conhecendo tuplas, dicionários e funções, além de outra biblioteca muito essencial chamada **Pandas**.

Na [próxima atividade](https://cursos.alura.com.br/course/python-funcoes-pacotes-pandas/task/66364) você pode baixar o notebook Python contendo alguns códigos e materiais do curso, pronto para ser preenchido conforme avançarmos nos estudos.

Esperamos que o treinamento seja bastante útil para sua carreira na ciência de dados! No próximo vídeo começaremos a preparar o ambiente em que iremos trabalhar. **Bons estudos!**

**02Projeto inicial do treinamento**

Nesse curso usaremos o **Google Colab** para escrever o código Python e executar os experimentos. Para tal, você precisa de uma conta no Google e acessar [esse link](https://colab.research.google.com/).

Além disso, [baixe aqui o ZIP do projeto inicial deste treinamento, necessário para a continuidade do mesmo.](https://caelum-online-public.s3.amazonaws.com/1480-pythondatascience/01/python-data-science-parte-2-aula-1-inicio.zip)

**03Instalação e ambiente de desenvolvimento**

Antes de começarmos a efetivamente trabalhar os conteúdos do nosso treinamento, é necessário conhecermos o ambiente em que iremos trabalhar, da mesma forma que fizemos na primeira parte. No endereço <https://colab.research.google.com/> (que pode ser encontrado no Google pesquisando pela palavra "colaboratory"),

No [exercício anterior](https://cursos.alura.com.br/course/python-funcoes-pacotes-pandas/task/66364), você deve ter baixado a [pasta zipada contendo os arquivos do curso](https://caelum-online-public.s3.amazonaws.com/1480-pythondatascience/01/python-data-science-parte-2-aula-1-inicio.zip) e que deverá ser adicionada na aba "Upload".

Esse notebook é bem parecido com o que usamos no curso anterior, contendo uma introdução ao Python, alguns passos de instalação do ambiente de desenvolvimento e verificação da versão, e assim por diante. Nesse curso não é necessário instalar o Python na sua própria máquina, já que o Google Colab nos fornece um bom ambiente e disponibiliza as bibliotecas necessárias (que também precisariam ser instaladas no seu computador).

A título de curiosidade, existe a distribuição Anaconda (Anaconda Distribution), muito famosa para data science, que já vem com um conjunto de pacotes/bibliotecas previamente instalados, como Jupyter, Pandas e Numpy.

No Google Colab, podemos verificar a versão do Python que está sendo utilizada para executar os códigos com a instrução !python -V. No momento da gravação deste curso a versão era **Python 3.6.9**, mas as versões posteriores também devem funcionar. Caso você tenha algum problema, pode ser necessário utilizar uma versão mais baixa.

No próximo vídeo vamos explorar brevemente o Pandas para deixá-lo(a) na expectativa a respeito do que trabalharemos ao final do curso.

**04A linguagem Python**

Sobre a linguagem Python, avalie as afirmativas abaixo:

**1)** Python é uma linguagem de programação de alto nível, com suporte a múltiplos paradigmas de programação

**2)** Python é um projeto open source

**3)** Python é uma linguagem de programação interpretada

Quais afirmativas estão corretas?

Todas as afirmativas estão corretas

Alternativa correta! Em nosso notebook temos uma breve definição da linguagem Python.

**05Trabalhando com dados**

É esperado que alguém que está iniciando sua carreira em Ciência de Dados consiga trabalhar com dados. Nesse cenário, o Python tem disponível uma biblioteca bastante importante, chamada **Pandas**, sobre a qual faremos um pequeno curso introdutório ao final deste treinamento.

Inicialmente gostaríamos de apresentar como ele funciona, abrindo um arquivo externo e colocando-o em uma estrutura de dados que é reconhecida pelo Pandas, além de analisarmos um pouco como esses dados funcionam. No futuro entenderemos mais profundamente como funcionam os tratamentos, faremos estatísticas descritivas e assim por diante.

No curso anterior nós trabalhamos com um *dataset* contendo características e valores de automóveis. A ideia é gerar um modelo de regressão voltado à precificação desses automóveis com base em tais características (um modelo de *machine learning*). Para isso, é necessário conhecer e tratar esses dados, algo que é bastante facilitado pelo Pandas.

No notebook Python do Colab, abriremos a aba na lateral esquerda, clicaremos em "Files" e então "Upload" para subirmos o arquivo db.csv (encontrado na pasta "data" dentre os arquivos baixados anteriormente). Como usaremos uma biblioteca que não vem previamente instalada, será necessário importá-la, algo que é feito usando o comando import.

Nesse caso, faremos import pandas as pd de modo a não só realizarmos a importação, mas também atribuirmos à biblioteca o apelido pd (facilitando a utilização dos seus comandos). Após executarmos com "Shift + Enter", prosseguiremos para a importação dos dados no nosso conjunto.

Criaremos uma variável dataset que receberá a chamada de pd.read\_csv() - o apelido da biblioteca, pd, seguido da função read\_csv(), que realiza a leitura de arquivos. Como parâmetro passaremos o arquivo que acabamos de subir, dv.csv.

dataset = pd.read\_csv('db.csv')COPIAR CÓDIGO

Ao executarmos receberemos alguns erros, pois existe um argumento faltando: sep, que define o separador utilizado no conjunto de dados. Nesse caso, o separador é ponto e vírgula (;). Poderemos então visualizar nosso conjunto a partir da variável dataset.

dataset = pd.read\_csv('db.csv', sep = ';')

datasetCOPIAR CÓDIGO

| **Nome** | **Motor** | **Ano** | **Quilometragem** | **Zero\_km** | **Acessórios** | **Valor** |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | Jetta Variant | Motor 4.0 Turbo | 2003 | 44410.0 | False | ['Rodas de liga', 'Travas elétricas', 'Piloto ... | 88078.64 |
| 1 | Passat | Motor Diesel | 1991 | 5712.0 | False | ['Central multimídia', 'Teto panorâmico', 'Fre... | 106161.94 |
| 2 | Crossfox | Motor Diesel V8 | 1990 | 37123.0 | False | ['Piloto automático', 'Controle de estabilidad... | 72832.16 |
| 3 | DS5 | Motor 2.4 Turbo | 2019 | NaN | True | ['Travas elétricas', '4 X 4', 'Vidros elétrico... | 124549.07 |
| 4 | Aston Martin DB4 | Motor 2.4 Turbo | 2006 | 25757.0 | False | ['Rodas de liga', '4 X 4', 'Central multimídia... | 92612.10 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 253 | Phantom 2013 | Motor V8 | 2014 | 27505.0 | False | ['Controle de estabilidade', 'Piloto automátic... | 51759.58 |
| 254 | Cadillac Ciel concept | Motor V8 | 1991 | 29981.0 | False | ['Bancos de couro', 'Painel digital', 'Sensor ... | 51667.06 |
| 255 | Classe GLK | Motor 5.0 V8 Bi-Turbo | 2002 | 52637.0 | False | ['Rodas de liga', 'Controle de tração', 'Câmbi... | 68934.03 |
| 256 | Aston Martin DB5 | Motor Diesel | 1996 | 7685.0 | False | ['Ar condicionado', '4 X 4', 'Câmbio automátic... | 122110.90 |
| 257 | Macan | Motor Diesel V6 | 1992 | 50188.0 | False | ['Central multimídia', 'Teto panorâmico', 'Vid... | 90381.47 |

Perceba que temos o mesmo conteúdo visualizado no arquivo CSV, que também pode ser aberto em um editor de planilhas como o Microsoft Excel. Agora temos um dataframe, a estrutura de dados disponibilizada pelo Pandas para trabalharmos com dados. Dentre as informações, temos o nome dos veículos, tipo de motor, ano, quilometragem, se o veículo é ou não zero quilômetros, os acessórios e o valor.

Repare que nosso retorno só nos mostrou dez linhas, mas é possível visualizar outras. Utilizando a função pd.set\_option() é possível definir alguns parâmetros do dataframe, como o número máximo de linhas retornadas, algo que configuraremos com display.max\_rows seguido do número determinado de linhas, neste exemplo 1000.

import pandas as pd

pd.set\_option('display.max\_rows', 1000)COPIAR CÓDIGO

Se executarmos dataset novamente, receberemos todas as 258 linhas - não foram mostradas 1000 pois nosso conjunto não possui esse total de informações. Também temos uma opção para exibir mais colunas, chamada display.max\_columns. Como gostaríamos de manter a configuração padrão, comentaremos ambas as linhas.

import pandas as pd

# pd.set\_option('display.max\_rows', 10000)

# pd.set\_option('display.max\_columns', 1000)COPIAR CÓDIGO

É possível obtermos os tipos de dados que temos no conjunto, algo que também fizemos no curso anterior, utilizando o comando dataset.dtypes.

dataset.dtypesCOPIAR CÓDIGO

Nome object Motor object Ano int64 Quilometragem float64 Zero\_km bool Acessórios object Valor float64 dtype: object

Aqui temos os tipos de cada coluna do dataframe. No Pandas, todo valor do tipo string é tratado como *object*. Além deste, temos o ano como tipo inteiro (int64), quilometragem como float64, Zero\_km como booleano (bool) e assim por diante. Note que, diferentemente dos arrays Numpy, nos dataframes do Pandas é possível trabalharmos com dados de tipos diferentes sem nenhum problema.

Vamos supor que desejamos calcular uma estatística descritiva com esse conjunto. Em nosso exemplo, usaremos as colunas "quilometragem" e "valor". Usaremos a variável dataset, que contém o nosso conjunto, e passaremos entre dois colchetes (pois estamos passando mais de uma informação) as duas variáveis que possuem os dados desejados - Quilometragem e Valor). Por fim, adicionaremos um ponto (.) e a função describe().

dataset[['Quilometragem', 'Valor']].describe()COPIAR CÓDIGO

O Pandas gerará, de forma simples e rápida, um conjunto de estatísticas descritivas dessas duas variáveis: a quantidade de valores, a média, o desvio padrão, o valor mínimo, os quartis da distribuição e o valor máximo.

|  | **Quilometragem** | **Valor** |
| --- | --- | --- |
| count | 197.000000 | 258.000000 |
| mean | 58278.421320 | 98960.513101 |
| std | 35836.733259 | 29811.932305 |
| min | 107.000000 | 50742.100000 |
| 25% | 27505.000000 | 70743.512500 |
| 50% | 55083.000000 | 97724.380000 |
| 75% | 90495.000000 | 124633.302500 |
| max | 119945.000000 | 149489.920000 |

Nos cursos de estatística da Alura nos aprofundamos no significado de cada uma dessas informações!

Com a função info() também podemos conseguir outras informações importantes. Por exemplo, acabamos de descobrir que temos menos ocorrências de Quilometragem do que de Valor.

dataset.info()COPIAR CÓDIGO

<class 'pandas.core.frame.DataFrame'> RangeIndex: 258 entries, 0 to 257 Data columns (total 7 columns): Nome 258 non-null object Motor 258 non-null object Ano 258 non-null int64 Quilometragem 197 non-null float64 Zero\_km 258 non-null bool Acessórios 258 non-null object Valor 258 non-null float64 dtypes: bool(1), float64(2), int64(1), object(3) memory usage: 12.5+ KB

Isso acontece pois, dentro do conjunto, temos alguns valores faltando (*missing*) ou nulos (*null*). Nos aprofundaremos mais no tratamento desses valores na última aula do curso. No próximo capítulo voltaremos a trabalhar com o Python, agora conhecendo as tuplas!

**06Primeiro contato com o Pandas**

Tivemos o nosso primeiro contato com o pacote **Pandas**, do Python. Conhecemos o dataset do nosso projeto e trabalhamos um pouco com a principal estrutura de dados do Pandas, o **DataFrame**. Das funcionalidades do Pandas que conhecemos até o momento, indique qual retorna um resumo estatístico do conjunto de dados que estamos trabalhando:

.describe()

Alternativa correta! O método describe() gera um conjunto de estatísticas descritivas das colunas selecionadas de um DataFrame.

**08O que aprendemos?**

Nesta aula, aprendemos:

* Os ambientes de desenvolvimento para a linguagem Python
* A criar ***DataFrames*** com o pacote **pandas**, a partir de dados externos
* Como obter informações básicas de um DataFrame
* Como obter estatísticas descritivas dos dados de um DataFrame

02. Listas estáticas: Tuplas

**02Conhecendo as tuplas**

Continuando com o treinamento básico de Python para Data Science que começamos no curso anterior, conheceremos agora uma nova estrutura de dados trabalhada no Python: as tuplas, sequências utilizadas para armanzear coleções de itens, assim como as listas que aprendemos no curso anterior. A diferença básica entre elas é que, diferentemente das listas, as tuplas são imutáveis - ou seja, não é possível modificá-las.

É importante conhecer as tuplas pois, como cientista de dados, você precisará trabalhar com conjuntos de informações. Portanto, faz sentido termos estruturas de dados onde é possível armazenar esse tipo de coleção.

Também é importante ressaltar que a maioria das *built-in functions* (funções nativas) do Python, principalmente as que retornam mais de um valor, têm como retorno uma tupla. Ao final do treinamento passaremos por alguns exemplos disso e, inclusive, criaremos nossas próprias funções que retornam tuplas.

No notebook Python do curso (que pode ser baixado no [exercício anterior](https://cursos.alura.com.br/course/python-funcoes-pacotes-pandas/task/66331)), deixamos alguns exemplos de como é possível criar uma tupla:

- Utilizando um par de parênteses: ( )

- Utilizando uma vírgula à direita: x,

- Utilizando um par de parênteses com itens separados por vírgulas: ( x, y, z )

- Utilizando: tuple() ou tuple(iterador)COPIAR CÓDIGO

Começaremos pelo primeiro. Ao abrirmos e fecharmos parênteses em uma célula (()) e pressionarmos "Shift + Enter", teremos como retorno:

()

Ou seja, uma tupla vazia. Já se digitarmos uma sequência de números separados por vírgula, como 1, 2, 3, receberemos uma tupla populada com esses valores:

(1, 2, 3)

Isso também pode ser feito com variáveis. Nesse exemplo, criaremos as variáveis nome e valor recebendo 'Passat' e 153000, respectivamente. Em seguida, passaremos ambas as variáveis dentro de um par de parênteses.

nome = 'Passat'

valor = 153000

(nome, valor)COPIAR CÓDIGO

Teremos criado uma tupla com as informações desejadas:

('Passat', 153000)

Note que as tuplas, assim como as listas, podem armazenar tipos variados de dados, como strings, dados numéricos, listas ou mesmo outras tuplas. A última forma de criarmos tuplas é usando a função tuple(). Para testarmos, criaremos uma variável nomes\_carros para qual atribuiremos a chamada dessa função, que por sua vez receberá como parâmetro uma lista com nomes de carros.

nomes\_carros = tuple(['Jetta Variant', 'Passat', 'Crossfox', 'DS5'])

nomes\_carrosCOPIAR CÓDIGO

Ao executarmos, receberemos a tupla criada:

('Jetta Variant', 'Passat', 'Crossfox', 'DS5')

Por último, sabemos que a função type() nos permite verificar o tipo de um dado no Python. Faremos isso com a variável nomes\_carros.

type(nomes\_carros)COPIAR CÓDIGO

Logicamente, receberemos como resposta que o dado é do tipo tupla (tuple).

tuple

No próximo vídeo aprenderemos a acessar as informações dentro de uma tupla.

**03Formas de se criar uma tupla**

Tuplas são sequências imutáveis, que são utilizadas para armazenar coleções de itens.

Dentre as alternativas abaixo, quais mostram formas de se criar uma tupla?

 ( 1, 2, 3 )

Alternativa correta! Podemos criar uma tupla utilizando parênteses, com os seus itens separados por vírgulas.

 Alternativa correta



tuple( [ 1, 2, 3 ] )

Alternativa correta! O tipo tuple permite criar tuplas, utilizando como argumento algum tipo de iterador. Neste caso, foi utilizada uma lista do Python.

 Alternativa correta



1, 2, 3

Alternativa correta! Não é preciso utilizar os parênteses para se definir uma tupla, basta definir os elementos e separá-los por vírgulas.

**04Seleções em tuplas**

No treinamento anterior aprendemos a fazer seleções em listas do Python e em arrays Numpy. Uma boa notícia é que o processo com tuplas é exatamente o mesmo, ou seja, se você fixou bem o conteúdo, não terá dificuldades nesse passo. Para nossos testes, inicializaremos novamente a variável nomes\_carros, uma tupla com quatro nomes de veículos.

nomes\_carros = tuple(['Jetta Variant', 'Passat', 'Crossfox', 'DS5'])

nomes\_carrosCOPIAR CÓDIGO

('Jetta Variant', 'Passat', 'Crossfox', 'DS5')

Estamos trabalhando com uma sequência ordenada, da mesma forma que nas listas. Como a indexação se inicia no 0, sabemos que o primeiro item da tupla, "Jetta Variant", recebe esse índice. Por consequência, "Passat" recebe o índice 1; "Crossfox" o índice 2; "DS5" o índice 3; e assim por diante, caso tivéssemos uma tupla com mais itens.

Para acessarmos uma posição da tupla, podemos passar o nome da variável, nesse caso nomes\_carros, seguido de colchetes e a posição que desejamos verificar.

nomes\_carros[0]COPIAR CÓDIGO

'Jetta Variant'

Podemos repetir esse processo com o segundo item.

nomes\_carros[1]COPIAR CÓDIGO

'Passat'

Já aprendemos a acessar facilmente o último elemento de uma lista, e o procedimento é o mesmo para tuplas - ou seja, índices negativos começando pelo -1.

nomes\_carros[-1]COPIAR CÓDIGO

'DS5'

Também aprendemos a fazer fatiamentos ("*slices*") nas listas, lembrando que o ponto de início indicado é incluído no fatiamento (*inclusive*), mas o ponto de parada não (*exclusive*). Isso também ocorre com as tuplas. Em nosso exemplo, queremos que a tupla resultante mostre apenas os carros intermediários, "Passat" e "Crossfox". Para isso passaremos, depois da variável, [1:3] - ou seja, colchetes contendo a posição inicial do nosso fatiamento, o caractere dois pontos (:) e a posição final não-inclusiva (também chamada de exclusiva).

nomes\_carros[1:3]COPIAR CÓDIGO

No retorno teremos exatamente os valores esperados.

('Passat', 'Crossfox')

Como vimos anteriormente, se tivéssemos passado nomes\_carros[1:2], receberíamos somente a posição 1.

nomes\_carros[1:2]COPIAR CÓDIGO

('Passat')

Em resumo, o primeiro item que passamos no fatiamento aparecerá na tupla resultante, mas o último não. Prosseguindo, é possível termos tuplas que contêm, em si, outrasa tuplas.

nomes\_carros = ('Jetta Variant', 'Passat', 'Crossfox', 'DS5', ('Fusca', 'Gol', 'C4'))

nomes\_carrosCOPIAR CÓDIGO

('Jetta Variant', 'Passat', 'Crossfox', 'DS5', ('Fusca', 'Gol', 'C4'))

Se quisermos acessar uma informação dentro da tupla interior, pegaremos o último item da exterior com [-1].

nomes\_carros[-1]COPIAR CÓDIGO

('Fusca', 'Gol', 'C4')

Queremos o segundo item dessa tupla. Para isso, passaremos um novo par de colchetes contendo a sua posição, ou seja, 1.

nomes\_carros[-1][1]COPIAR CÓDIGO

Como retorno, teremos o valor esperado:

'Gol'

No próximo vídeo faremos iterações com tuplas, varrendo informações de forma automática.

**05Selecionando itens em tuplas**

Em nosso primeiro treinamento de Python para Data Science, nós aprendemos como fazer seleções de itens em listas e em arrays Numpy. O procedimento para seleções em tuplas funciona da mesma forma. Considere a seguinte tupla:

carros = (

(

'Jetta Variant',

'Motor 4.0 Turbo',

2003,

False,

('Rodas de liga', 'Travas elétricas', 'Piloto automático')

),

(

'Passat',

'Motor Diesel',

1991,

True,

('Central multimídia', 'Teto panorâmico', 'Freios ABS')

)

)COPIAR CÓDIGO

Considere também os seguintes códigos de seleção:

**1)** carros[0][3]

**2)** carros[-1][-1][-1]

**3)** carros[0][-1][:2]

Assinale a alternativa que mostra o resultado obtido com os códigos de seleção acima.

*Selecione uma alternativa*

**1)** False

**2)** 'Freios ABS'

**3)** ('Rodas de liga', 'Travas elétricas')

Alternativa correta! É a mesma técnica que aprendemos com listas e arrays Numpy.

**06Iterando em tuplas**

Já aprendemos a fazer iterações em listas, e as iterações em tuplas não são diferentes. Nesse vídeo trabalharemos um recurso que nos auxiliará quando estivermos usando funções, principalmente as que retornam mais de um valor, e também com alguns tipos de iterações.

Começaremos inicializando a tupla nomes\_carros.

nomes\_carros = ('Jetta Variant', 'Passat', 'Crossfox', 'DS5')

nomes\_carrosCOPIAR CÓDIGO

Para iterarmos por seus itens, usaremos a instrução for, atribuiremos um nome válido a cada item encontrado (por exemplo item) e continuaremos com in seguido do nome da tupla (nomes\_carros). Não podemos nos esquecer de finalizar a linha com dois pontos (:) e da indentação na linha seguinte. Nesse exemplo simplesmente imprimiremos cada um dos item encontrados usando a função print().

for item in nomes\_carros:

print(item)COPIAR CÓDIGO

Jetta Variant Passat Crossfox DS5

Conseguimos varrer cada um dos itens e imprimi-los. O próximo conceito que iremos abordar é o desempacotamento de tuplas, o recurso que citamos anteriormente. Basicamente, é possível atribuirmos cada valor dentro de uma tupla a uma variável específico. Nesse primeiro exemplo, usaremos as variáveis carro\_1, carro\_2, carro\_3 e carro\_4 separadas por vírgula. Essa sequência receberá o conteúdo de nomes\_carros.

carro\_1, carro\_2, carro\_3, carro\_4 = nomes\_carrosCOPIAR CÓDIGO

Como temos quatro carros e quatro variáveis, cada uma delas receberá um dos valores na tupla, na ordem que definimos. Podemos verificar isso mostrando o conteúdo das nossas variáveis:

carro\_1COPIAR CÓDIGO

'Jetta Variant'

carro\_2COPIAR CÓDIGO

'Passat'

carro\_3COPIAR CÓDIGO

'Crossfox'

carro\_4COPIAR CÓDIGO

'DS5'

Esse é o desempacotamento de tuplas, algo útil, por exemplo, quando uma função nos retorna mais de um valor dentro de uma tupla e queremos utilizá-los de formas variadas no nosso programa, separando-os em variáveis diferentes.

Prosseguiremos para outra situação interessante: imagine que uma função nos retorna quatro valores, mas queremos utilizar somente dois deles. Nesse caso, queremos somente o segundo e o último valor da tupla nomes\_carros (a primeira posição e a terceira). Para isso, podemos usar o recurso do *underscore* (ou "*underline*"), da mesma forma que vimos nas listas.

\_, A, \_, B = nomes\_carrosCOPIAR CÓDIGO

Aqui usamos as variáveis A e B para recebermos os valores das posições desejadas, e os *underscores* para delimitar as posições que não desejamos salvar em variáveis específicas, ignorando os seus valores.

ACOPIAR CÓDIGO

'Passat'

BCOPIAR CÓDIGO

'DS5'

Em outra situação, queremos somente um valor da nossa tupla, mas nenhum outro. Imagine se tivéssemos uma tupla com 60 itens e só gostaríamos de salvar o segundo. Com a sintaxe que acabamos de aprender, seria bastante trabalhoso recuperarmos esse valor.

Para simplificar, podemos ignorar o primeiro valor com *underscore*, receber o segundo com uma variável (por exemplo C) e finalizar com asterisco seguido de *underscore* (\*\_). Dessa forma ignoraremos todos os valores posteriores.

\_, C, \*\_ = nomes\_carrosCOPIAR CÓDIGO

Com isso, recuperaremos somente o valor da primeira posição (ou o segundo item).

CCOPIAR CÓDIGO

Passat

Existe também uma nova função nativa do Python, chamada zip(), que cria um iterador com tuplas (de maneira semelhante à função range()), e vamos entender como ela funciona. Para nossos testes, inicializaremos duas listas: carros, contendo os modelos de alguns veículos; e valores, com seus preços.

carros = ['Jetta Variant', 'Passat', 'Crossfox', 'DS5']

carrosCOPIAR CÓDIGO

valores = [88078.64, 106161.94, 72832.16, 124549.07]

valoresCOPIAR CÓDIGO

Chamaremos então a função zip() passando como parâmetros nossas duas listas.

zip(carros, valores)COPIAR CÓDIGO

Teremos um retorno que significa simplesmente que nosso iterador foi criado com sucesso.

zip at 0x7fb74e671dc8

Com o auxílio da função list(), podemos transformar esse iterador em uma lista de modo a visualizá-lo.

list(zip(carros, valores))COPIAR CÓDIGO

[('Jetta Variant', 88078.64), ('Passat', 106161.94), ('Crossfox', 72832.16), ('DS5', 124549.07)]

Repare que, interessantemente, nosso iterador pegou o primeiro elemento de cada lista e criou uma tupla; depois o segundo elemento de cada lista, criando uma segunda tupla; e assim sucessivamente até o final da lista. Isso nos traz a possibilidade de, por exemplo, iterar o nome e o valor de um veículo ao mesmo tempo.

Faremos isso usando a instrução for que irá iterar pelo resultado de zip(carros, valores). Por enquanto apenas imprimiremos as tuplas resultantes.

for item in zip(carros, valores):

print(item)COPIAR CÓDIGO

('Jetta Variant', 88078.64) ('Passat', 106161.94) ('Crossfox', 72832.16) ('DS5', 124549.07)

Acabamos de aprender a fazer o desempacotamento de tuplas, e podemos fazer isso também com o for. Aqui, ao invés de item, passaremos dois nomes separados por vírgula: carro e valor. Também passaremos a imprimir ambas as variáveis (lembrando que o print() aceita mais de um argumento).

for carro, valor in zip(carros, valores):

print(carro, valor)COPIAR CÓDIGO

Com isso, conseguiremos separar cada carro e valor individualmente em todas as tuplas desse iterador.

Jetta Variant 88078.64 Passat 106161.94 Crossfox 72832.16 DS5 124549.07

Isso nos traz diversas outras possibilidades de análise. Por exemplo, podemos incluir uma condicional if que imprime somente os carros cujo valor é maior do que 100000.

for carro, valor in zip(carros, valores):

if(valor > 100000):

print(carro)COPIAR CÓDIGO

Como resultado, teremos apenas os dois carros que atendem a essa condição.

Passat DS5

No próximo capítulo avançaremos para um novo conteúdo: os dicionários, outro tipo de mapeamento utilizado no Python!

**07Laços for com tuplas**

O procedimento de iteração em tuplas é o mesmo que aprendemos com listas, no treinamento anterior. Utilizamos a tupla como iterador de um laço for simples, ou aninhado, e conseguimos acesso a cada item individualmente.

Para responder esta questão, considere a mesma tupla da atividade anterior:

carros = (

(

'Jetta Variant',

'Motor 4.0 Turbo',

2003,

False,

('Rodas de liga', 'Travas elétricas', 'Piloto automático')

),

(

'Passat',

'Motor Diesel',

1991,

True,

('Central multimídia', 'Teto panorâmico', 'Freios ABS')

)

)COPIAR CÓDIGO

Observe que se trata de uma tupla (1º nível) com duas tuplas, que representam um conjunto de dados de dois veículos (2º nível), e que uma destas informações (acessórios) vêm também dentro de uma tupla (3º nível). O que precisamos é iterar na tupla carros e imprimir todos os acessórios que aparecem. O resultado desejado é o seguinte:

Rodas de liga

Travas elétricas

Piloto automático

Central multimídia

Teto panorâmico

Freios ABSCOPIAR CÓDIGO

Assinale a alternativa com o código que produz este resultado.

*Selecione uma alternativa*

 for tupla in carros:

print(tupla)

Alternativa errada! O laço acima imprime as duas tuplas de informações dos veículos. O que precisamos é somente imprimir os acessórios de cada veículo.

 Alternativa correta



for tupla in carros:

print(tupla[-1])

Alternativa errada! Com o código acima, obtemos todos os acessórios da tupla carros, mas o resultado permanece dentro de tuplas. Não é isso que o problema pede. Talvez seja necessário utilizar um *loop* aninhado.

 Alternativa correta



for tupla in carros:

for item in tupla:

print(item)

Alternativa errada! O código acima imprime todos os itens das duas tuplas de informações dos veículos. Precisamos imprimir somente os acessórios.

 Alternativa correta



for tupla in carros:

for item in tupla[-1]:

print(item)

Alternativa correta! Note que no segundo laço for, utilizamos as listas de acessórios como iterador.

**08A função zip()**

Duas ferramentas bastante utilizadas quando iteramos com tuplas são o **desempacotamento de tuplas** e a ***built-in function* zip()**.

Com o desempacotamento de tuplas, é possível fazer declarações conjuntas de variáveis e utilizar cada variável individualmente. Por exemplo:

nome, valor = ('Passat', 100000.0)COPIAR CÓDIGO

A função zip() permite gerar um iterador de tuplas, como no exemplo abaixo:

**In [1]:**

nomes = ['Passat', 'Crossfox']

valores = [100000.0, 75000.0]

list(zip(nomes, valores))COPIAR CÓDIGO

**Out [1]:**

[('Passat', 100000.0), ('Crossfox', 75000.0)]COPIAR CÓDIGO

Considerando as duas listas abaixo:

nomes = ['Passat', 'Crossfox', 'DS5', 'C4', 'Jetta']

kms = [15000, 12000, 32000, 8000, 50000]COPIAR CÓDIGO

E utilizando o ferramental apresentado acima, marque a alternativa com o código que possibilita a impressão dos nomes dos veículos com quilometragem abaixo de 20.000 km.

*Selecione uma alternativa*

for nome, km in zip(nomes, kms):

if(km < 20000):

print(nome)

Alternativa correta! Observe que na construção do laço for (1ª linha do código), utilizamos o desempacotamento de tuplas e a função zip(). O *output* deste código deve ser o seguinte:

Passat

Crossfox

C4

**10O que aprendemos?**

Nesta aula, aprendemos:

* O que são as tuplas
* Formas de criação de uma tupla
* Técnicas de seleção de itens e fatiamento com tuplas do Python
* Formas de iterar por uma tupla
* A técnica conhecida como **desempacotamento de tuplas**
* A utilizar a *built-in function* zip()