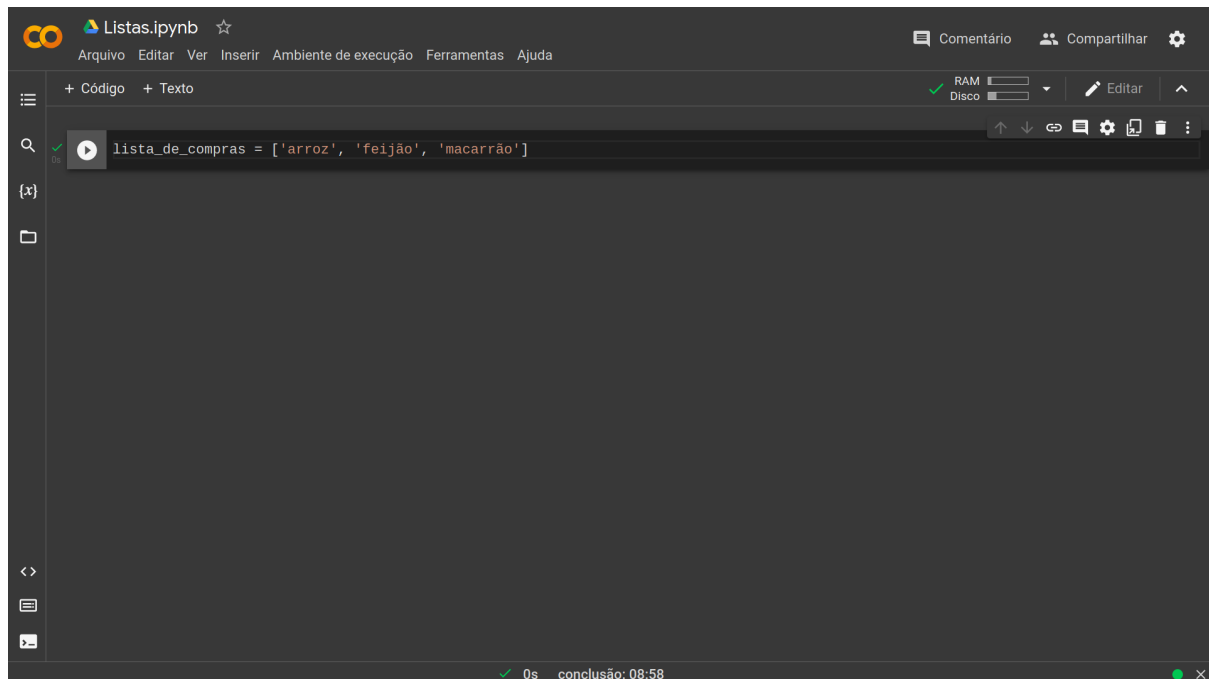


## Aula 30 - Estrutura de dados - listas, tuplas e dicionários

Olá, sejam bem-vindos a mais uma aula, bem nas duas aulas anteriores nós vimos um pouco mais a fundo sobre variáveis. Eu não cheguei a falar, mas uma variável também representa uma estrutura de dados. Mas o que é uma estrutura de dados.

Estrutura de dados em programação, são responsáveis por armazenar dados, igual o que fazíamos com as variáveis, por exemplo, onde a gente guardava um dado, um nome ou um número, ou até mesmo um valor booleano. E existem algumas outras estruturas de dados além dessa, como listas, pilhas, filas, árvores, dicionários e tuplas. No python, a estrutura mais simples dessas é a lista.

Uma lista é uma estrutura de dados que é utilizada para armazenar uma sequência de dados. Por exemplo, uma lista de supermercado, é onde você armazena os dados do que vai comprar. Cada produto que você vai comprar, possui uma posição na lista e em python para representar listas não é muito diferente. Cada posição é denominada índice, vamos ver como se cria uma lista em python:

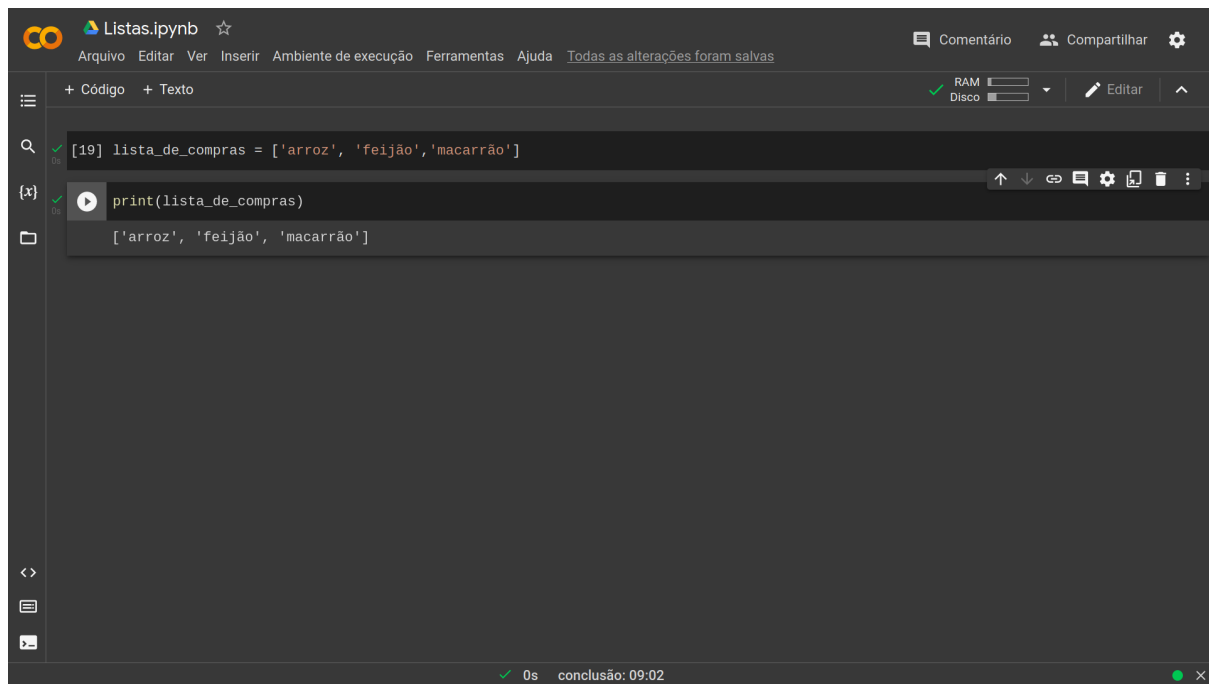


The screenshot shows a Jupyter Notebook window titled 'Listas.ipynb'. The interface includes a top menu bar with options like 'Arquivo', 'Editar', 'Ver', 'Inserir', 'Ambiente de execução', 'Ferramentas', and 'Ajuda'. Below the menu, there are tabs for '+ Código' and '+ Texto'. The main area displays a single code cell with the following Python code: 

```
lista_de_compras = ['arroz', 'feijão', 'macarrão']
```

. To the left of the code cell, there is a search icon and a play button icon. The bottom status bar shows a green checkmark, '0s', and 'conclusão: 08:58'.

Para declarar uma lista é bem simples, basta escolher o nome da variável e dentro de colchetes, passar as variáveis que desejamos.



The screenshot shows a Jupyter Notebook window titled 'Listas.ipynb'. The top menu bar includes 'Arquivo', 'Editar', 'Ver', 'Inserir', 'Ambiente de execução', 'Ferramentas', 'Ajuda', and 'Todas as alterações foram salvas'. On the right, there are buttons for 'Comentário', 'Compartilhar', and a settings icon. Below the menu, there are tabs for '+ Código' and '+ Texto'. The main area contains a code cell with the following Python code: 

```
[19] lista_de_compras = ['arroz', 'feijão', 'macarrão']  
  
print(lista_de_compras)
```

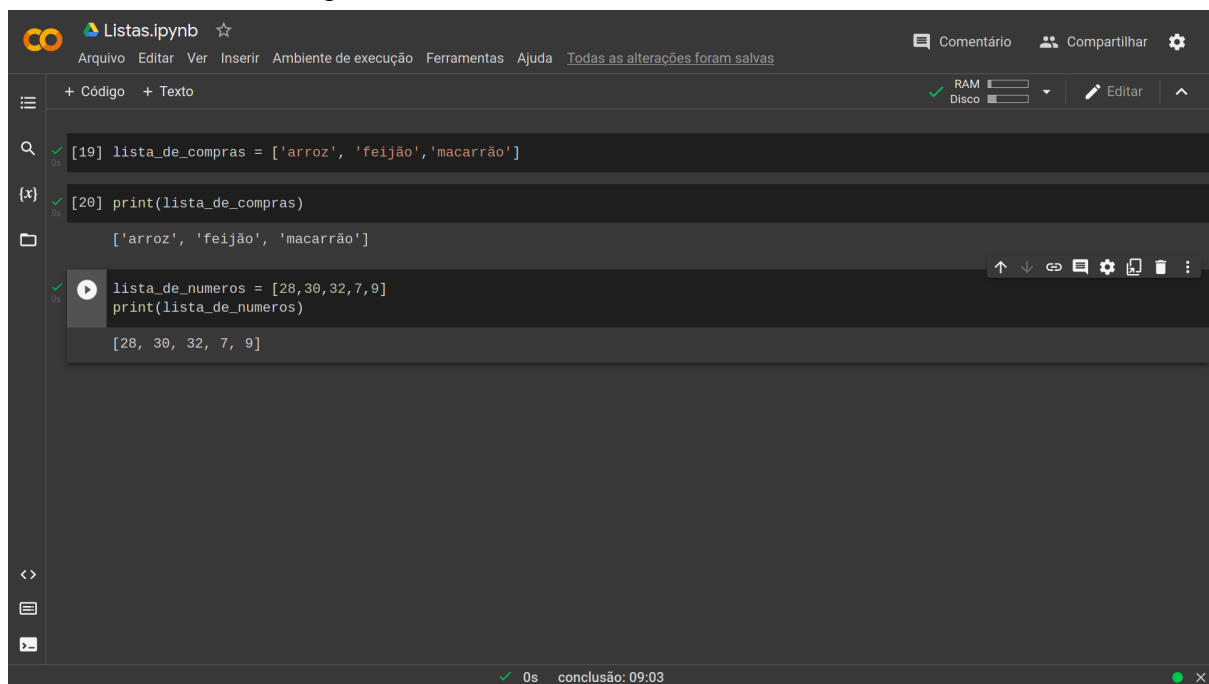
 The output of the cell is displayed below the code: 

```
['arroz', 'feijão', 'macarrão']
```

 The status bar at the bottom indicates '0s' and 'conclusão: 09:02'.

Podemos verificar que a lista está salva.

Podemos fazer o mesmo para uma lista de números:



The screenshot shows a Jupyter Notebook window titled 'Listas.ipynb'. The top menu bar includes 'Arquivo', 'Editar', 'Ver', 'Inserir', 'Ambiente de execução', 'Ferramentas', 'Ajuda', and 'Todas as alterações foram salvas'. On the right, there are buttons for 'Comentário', 'Compartilhar', and a settings icon. Below the menu, there are tabs for '+ Código' and '+ Texto'. The main area contains two code cells. The first cell has the following Python code: 

```
[19] lista_de_compras = ['arroz', 'feijão', 'macarrão']  
  
print(lista_de_compras)
```

 The output of the first cell is displayed below the code: 

```
['arroz', 'feijão', 'macarrão']
```

 The second cell has the following Python code: 

```
[20] lista_de_numeros = [28,30,32,7,9]  
print(lista_de_numeros)
```

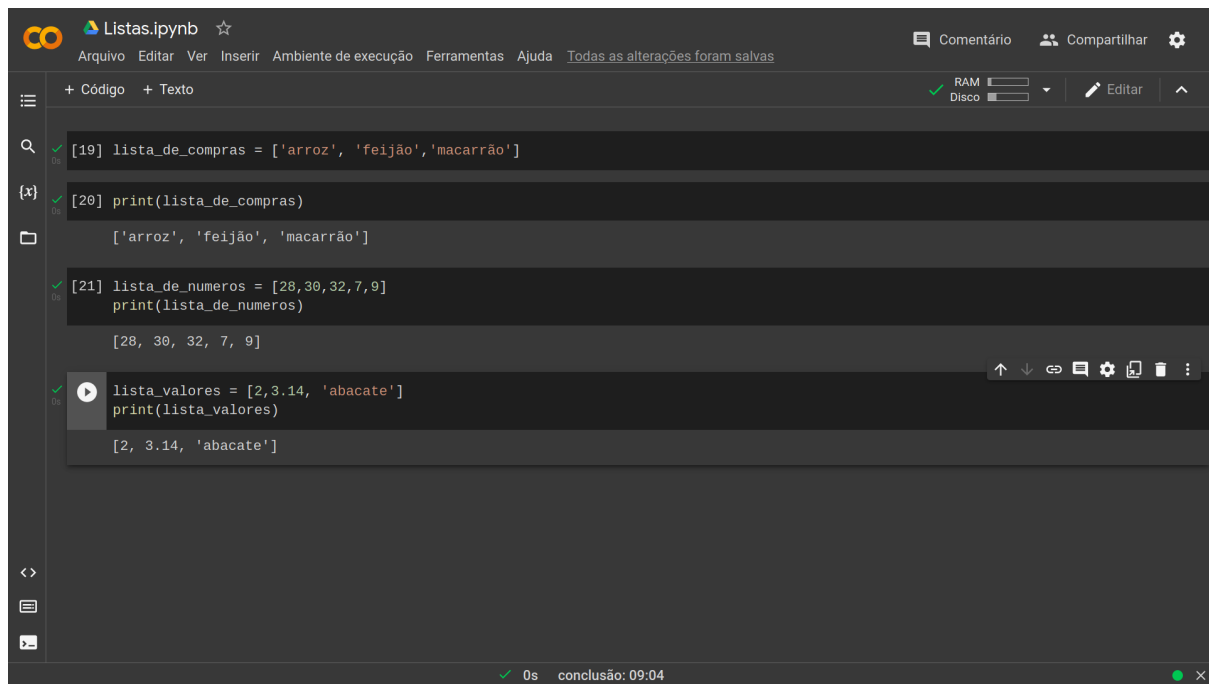
 The output of the second cell is displayed below the code: 

```
[28, 30, 32, 7, 9]
```

 The status bar at the bottom indicates '0s' and 'conclusão: 09:03'.

Dependendo do tipo de dado inserido dentro da lista esse será o tipo de dado que a nossa lista irá armazenar.

No caso de `lista_de_compras` o tipo de dado que estamos adicionando é `String` e no outro caso `int`. O python também nos permite inserir valores de diferentes tipos em uma mesma lista.



The screenshot shows a Jupyter Notebook interface with the title 'Listas.ipynb'. The top bar includes a menu (Arquivo, Editar, Ver, Inserir, Ambiente de execução, Ferramentas, Ajuda) and a status bar indicating 'Todas as alterações foram salvas'. The notebook contains three code cells, each with a green checkmark icon on the left. The first cell (line 19) defines a list of strings: `lista_de_compras = ['arroz', 'feijão', 'macarrão']`. The second cell (line 20) prints this list, showing the output `['arroz', 'feijão', 'macarrão']`. The third cell (line 21) defines a list of integers: `lista_de_numeros = [28,30,32,7,9]` and prints it, showing the output `[28, 30, 32, 7, 9]`. A fourth cell (line 22) defines a list with mixed types: `lista_valores = [2,3.14, 'abacate']` and prints it, showing the output `[2, 3.14, 'abacate']`. The status bar at the bottom shows '0s' and 'conclusão: 09:04'.

```
[19] lista_de_compras = ['arroz', 'feijão', 'macarrão']

[20] print(lista_de_compras)

['arroz', 'feijão', 'macarrão']

[21] lista_de_numeros = [28,30,32,7,9]
print(lista_de_numeros)

[28, 30, 32, 7, 9]

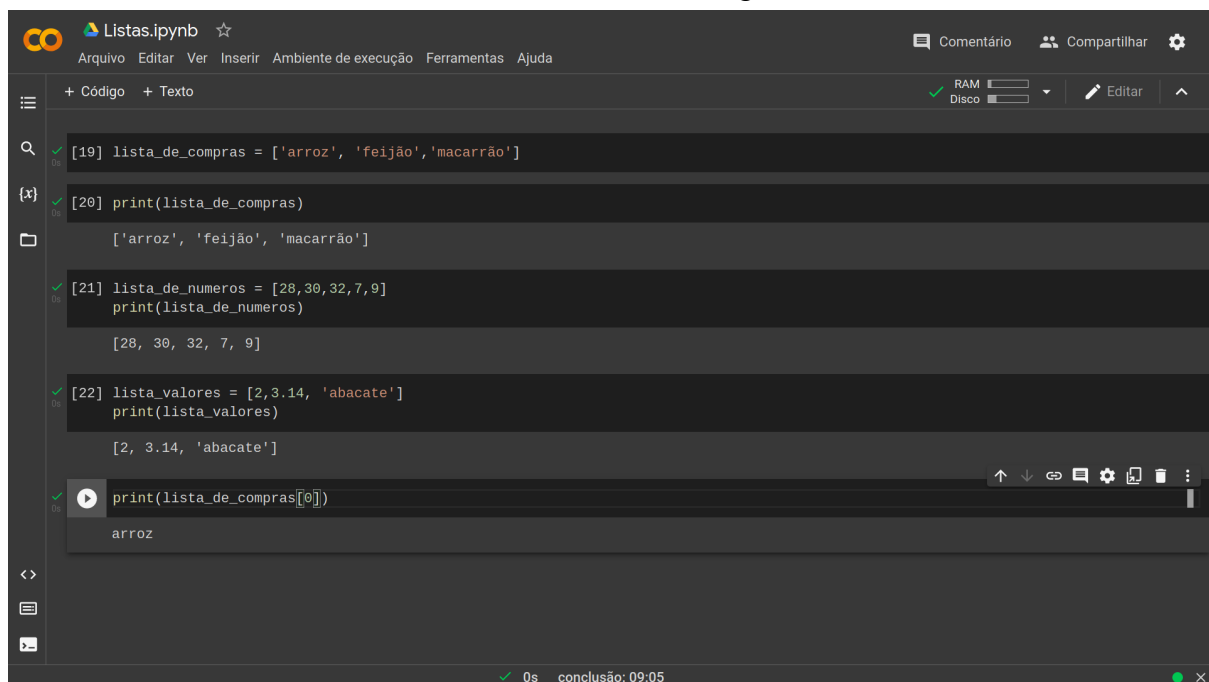
[22] lista_valores = [2,3.14, 'abacate']
print(lista_valores)

[2, 3.14, 'abacate']
```

Note que nessa lista temos três tipos de dados, float, string e int.

Mas se quisermos acessar algum valor da lista, como podemos fazer?

Cada item da lista é representado por um índice iniciando de 0. Ou seja, o primeiro elemento da lista está na lista e o índice 0. Podemos acessá-lo da seguinte maneira:



This screenshot shows the same Jupyter Notebook as the previous one, but with an additional code cell at the bottom. The new cell (line 23) uses indexing to access the first element of the `lista_de_compras` list: `print(lista_de_compras[0])`. The output of this cell is `arroz`. The status bar at the bottom now shows '0s' and 'conclusão: 09:05'.

```
[19] lista_de_compras = ['arroz', 'feijão', 'macarrão']

[20] print(lista_de_compras)

['arroz', 'feijão', 'macarrão']

[21] lista_de_numeros = [28,30,32,7,9]
print(lista_de_numeros)

[28, 30, 32, 7, 9]

[22] lista_valores = [2,3.14, 'abacate']
print(lista_valores)

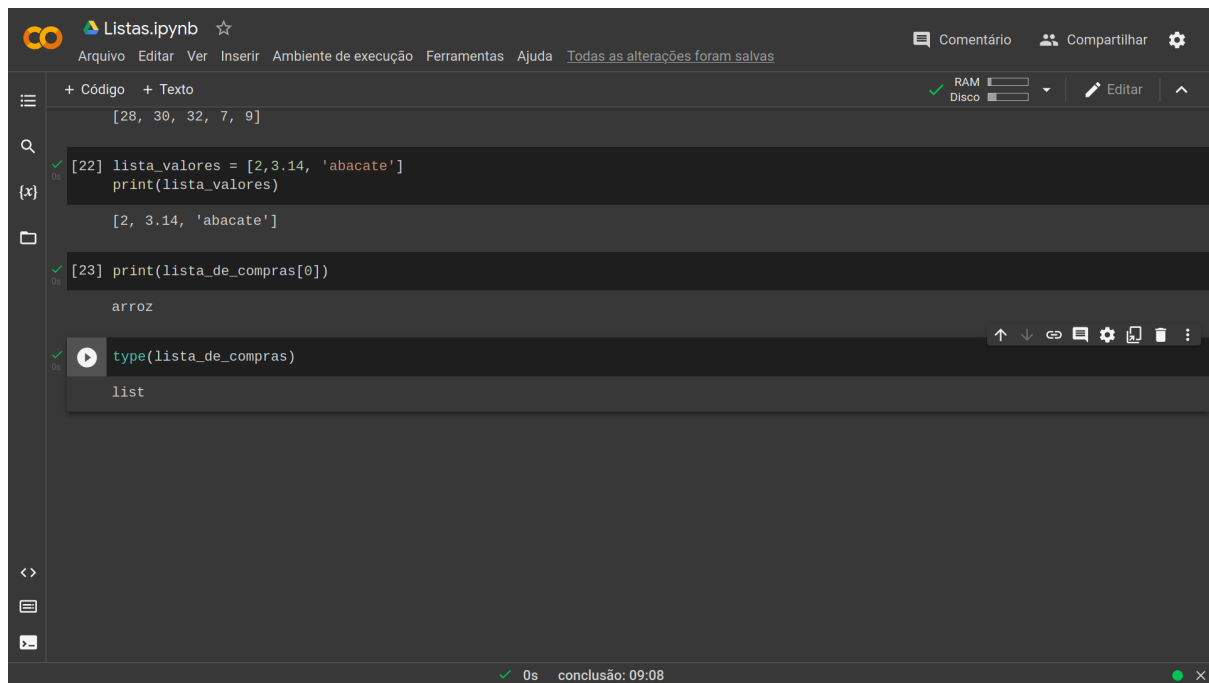
[2, 3.14, 'abacate']

[23] print(lista_de_compras[0])

arroz
```

Basta que coloquemos entre colchetes o índice do item que queremos acessar.

Podemos também verificar o tipo de dados.



The screenshot shows a Jupyter Notebook interface with the title 'Listas.ipynb'. The top bar includes a menu with 'Arquivo', 'Editar', 'Ver', 'Inserir', 'Ambiente de execução', 'Ferramentas', 'Ajuda', and 'Todas as alterações foram salvas'. On the right, there are buttons for 'Comentário', 'Compartilhar', and a settings icon. The left sidebar shows icons for file explorer, search, and other notebook functions. The main area displays the following code cells:

```
[28, 30, 32, 7, 9]
```

```
[22] lista_valores = [2, 3.14, 'abacate']  
print(lista_valores)
```

```
[2, 3.14, 'abacate']
```

```
[23] print(lista_de_compras[0])
```

```
arroz
```

```
type(lista_de_compras)
```

```
list
```

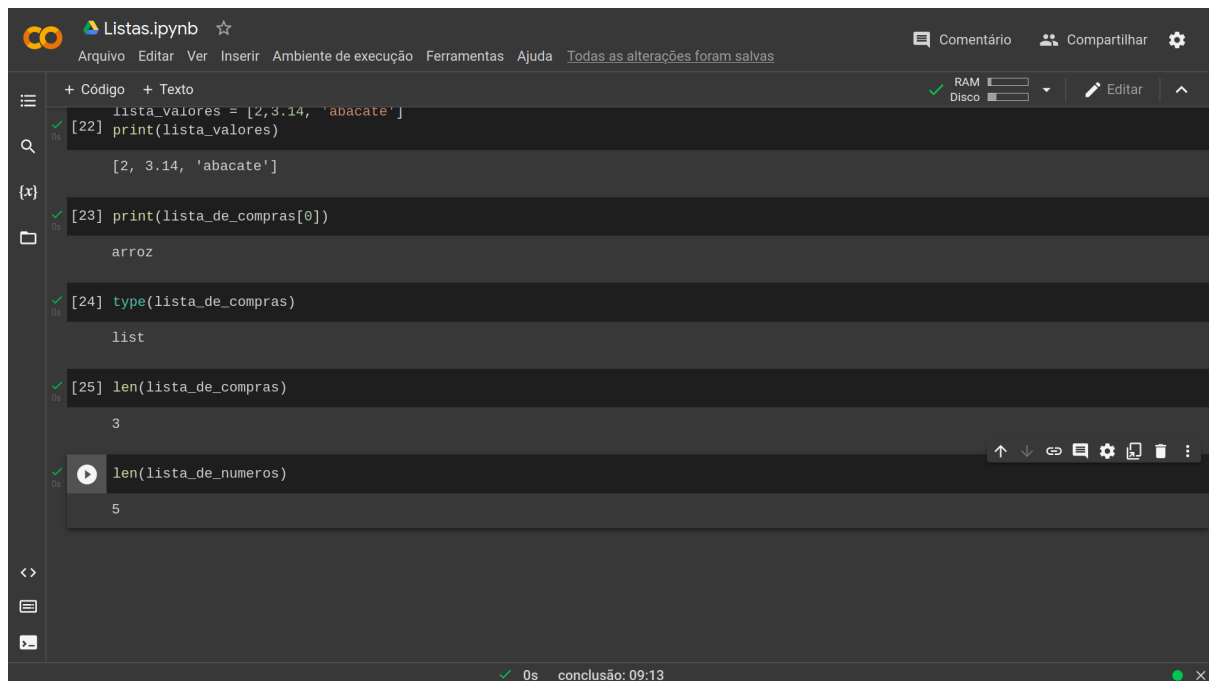
The bottom status bar indicates '0s' execution time and 'conclusão: 09:08'.

Podemos ver que o tipo da lista é list. Que é um tipo de dados do próprio python. Mas uma vantagem do python, em algumas outras linguagens não podemos armazenar mais de um tipo de variável dentro de uma lista.

O próprio python já possui algumas funções que podemos usar para manipular as listas:

Podemos verificar o tamanho de uma lista com a função:

`len(nome_da_lista)`:



This screenshot shows the same Jupyter Notebook with additional code cells demonstrating the use of the `len()` function:

```
lista_valores = [2, 3.14, 'abacate']
```

```
[22] print(lista_valores)
```

```
[2, 3.14, 'abacate']
```

```
[23] print(lista_de_compras[0])
```

```
arroz
```

```
[24] type(lista_de_compras)
```

```
list
```

```
[25] len(lista_de_compras)
```

```
3
```

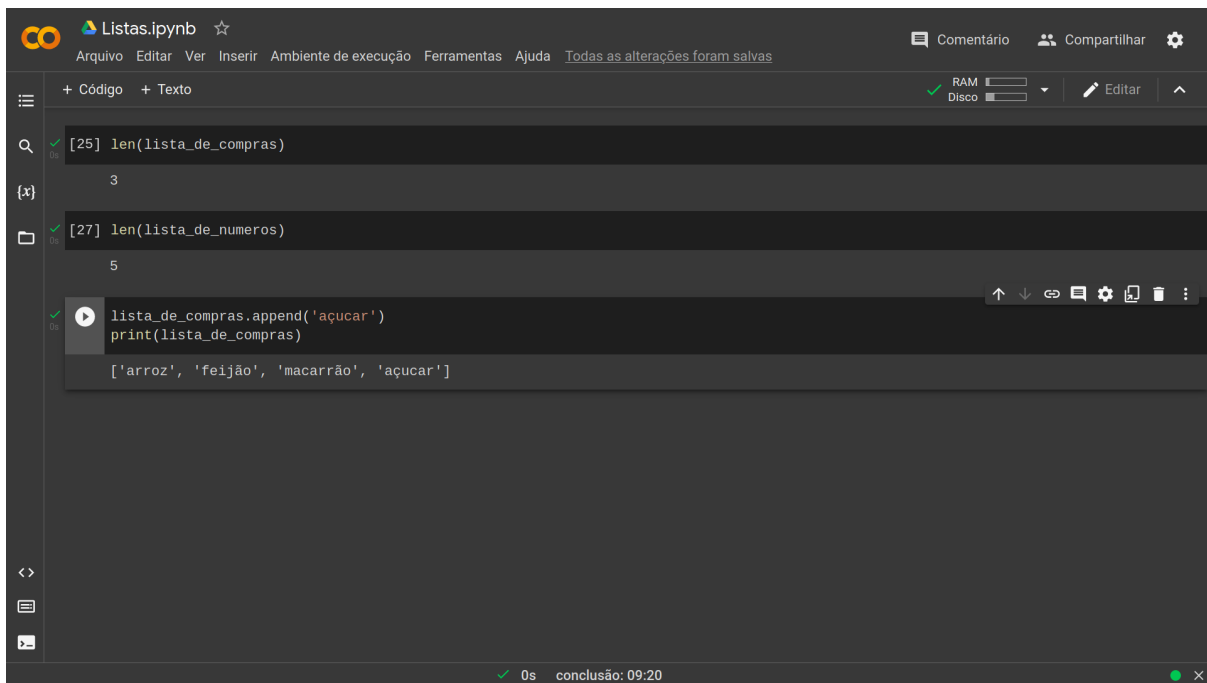
```
len(lista_de_numeros)
```

```
5
```

The bottom status bar now shows '0s' execution time and 'conclusão: 09:13'.

Podemos adicionar um novo elemento na lista com a seguinte função:

`lista.append(nome_da_lista)`:



The screenshot shows a Jupyter Notebook titled 'Listas.ipynb'. The interface includes a top menu bar with options like 'Arquivo', 'Editar', 'Ver', 'Inserir', 'Ambiente de execução', 'Ferramentas', 'Ajuda', and 'Todas as alterações foram salvas'. On the right, there are buttons for 'Comentário', 'Compartilhar', and a settings icon. The left sidebar has icons for file explorer, search, and code execution. The main area displays three code cells:

```
[25] len(lista_de_compras)
3
```

```
[27] len(lista_de_numeros)
5
```

```
lista_de_compras.append('açúcar')
print(lista_de_compras)

['arroz', 'feijão', 'macarrão', 'açúcar']
```

The bottom status bar shows '0s' and 'conclusão: 09:20'.

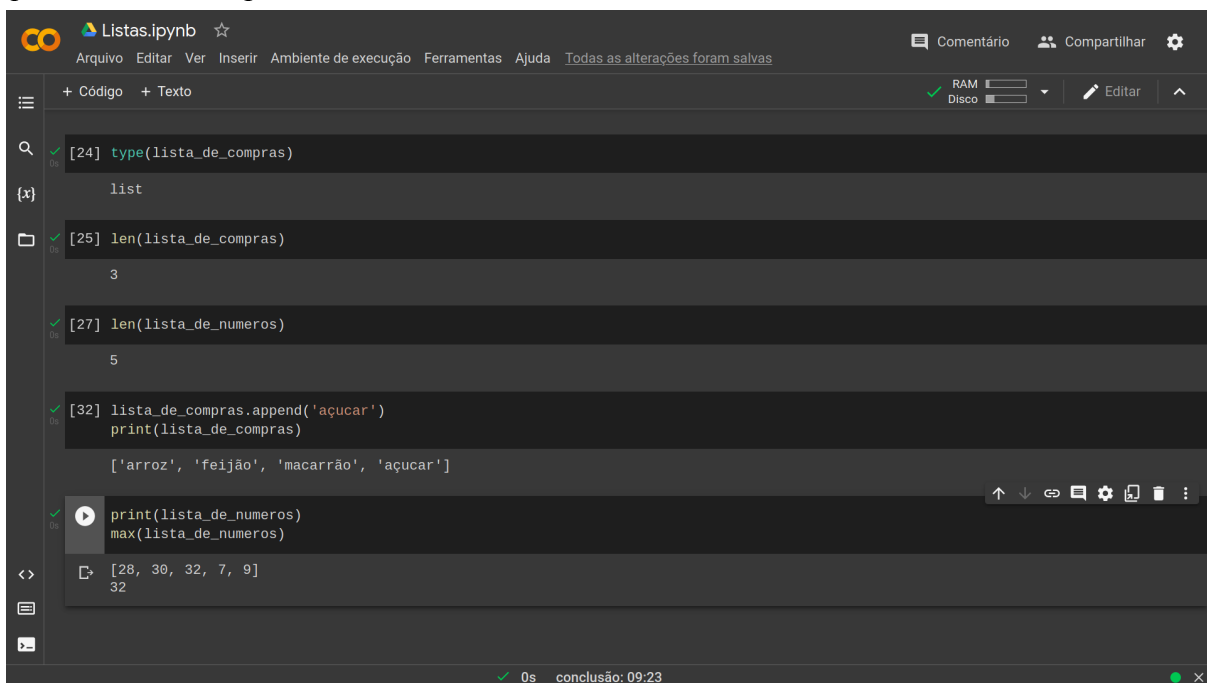
agora a nossa lista possui mais um item. Podemos notar que esse comando insere sempre um item ao final da lista.

Temos também os comandos:

`max(nome_da_lista)`

`min(nome_da_lista)`

que nos retorna respectivamente o maior e o menor elemento da lista.



The screenshot shows the same Jupyter Notebook with additional code cells:

```
[24] type(lista_de_compras)
list
```

```
[25] len(lista_de_compras)
3
```

```
[27] len(lista_de_numeros)
5
```

```
[32] lista_de_compras.append('açúcar')
print(lista_de_compras)

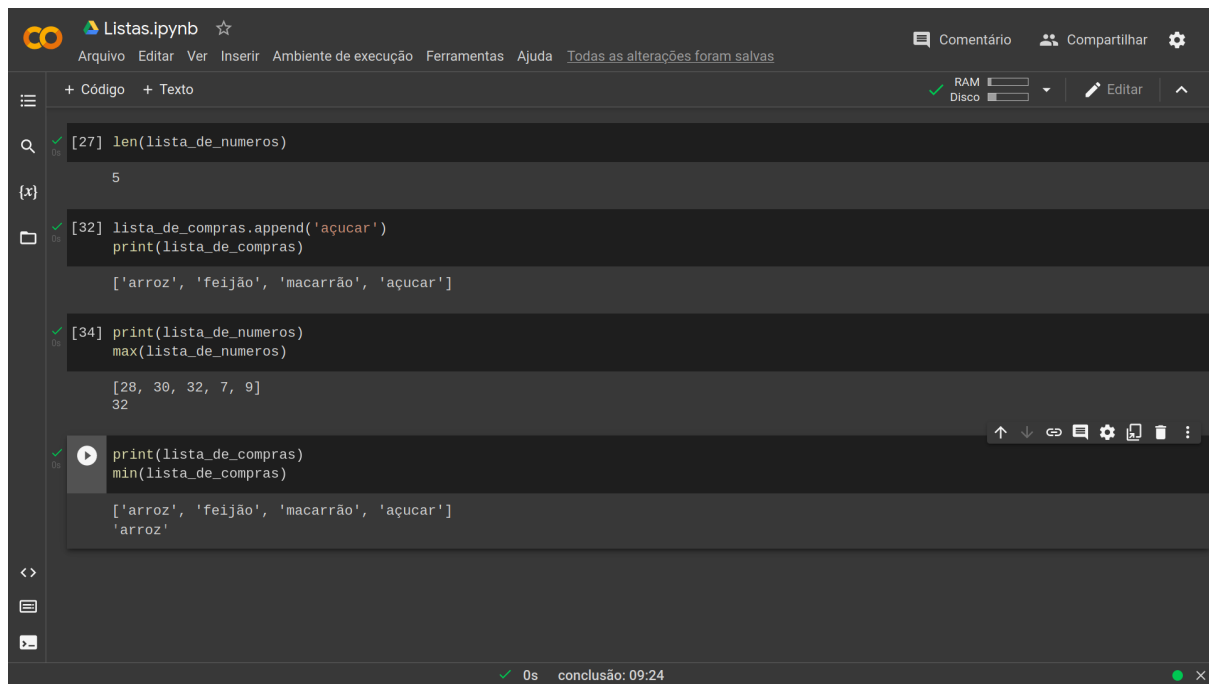
['arroz', 'feijão', 'macarrão', 'açúcar']
```

```
print(lista_de_numeros)
max(lista_de_numeros)
```

```
[28, 30, 32, 7, 9]
32
```

The bottom status bar shows '0s' and 'conclusão: 09:23'.

podemos ver que funciona para strings também, já que palavras podem ser organizadas em ordem alfabética:



The screenshot shows a Jupyter Notebook interface with the title 'Listas.ipynb'. The code is as follows:

```
[27] len(lista_de_numeros)

5

[32] lista_de_compras.append('açucar')
     print(lista_de_compras)

['arroz', 'feijão', 'macarrão', 'açucar']

[34] print(lista_de_numeros)
     max(lista_de_numeros)

[28, 30, 32, 7, 9]
32

print(lista_de_compras)
min(lista_de_compras)

['arroz', 'feijão', 'macarrão', 'açucar']
'arroz'
```

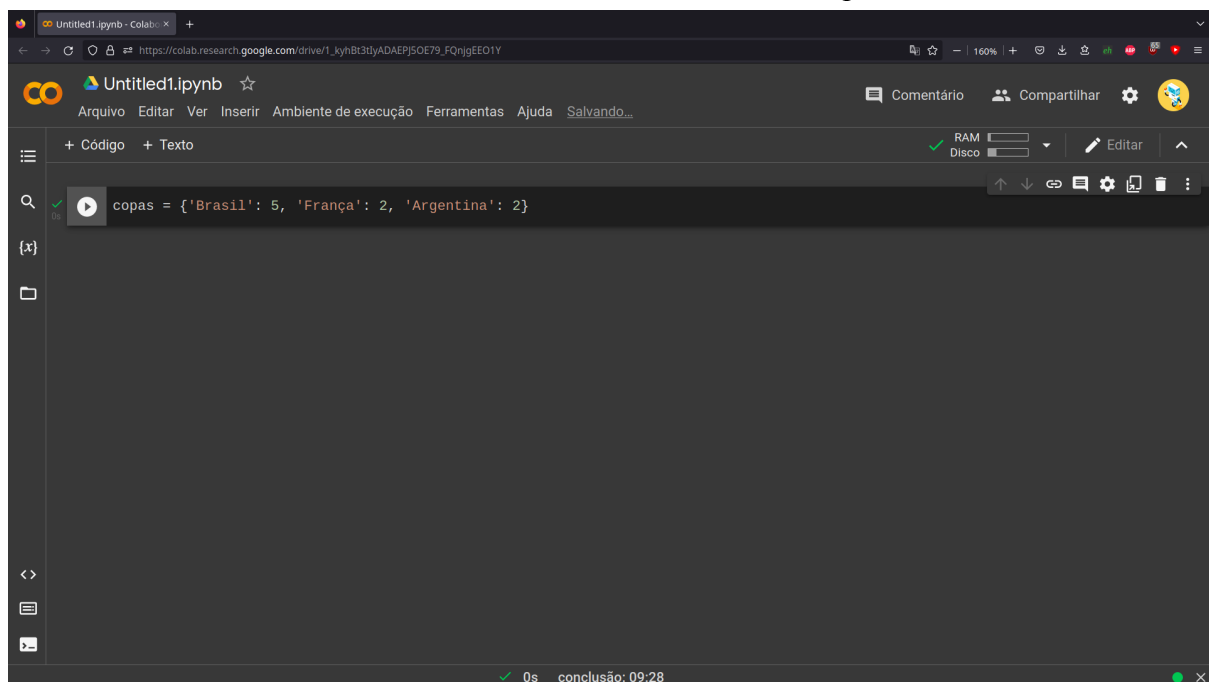
Uma outra estrutura de dados que vamos ter no Python é chamado de dicionário:

### Dicionário.

No python um dicionário é uma estrutura de dados que representa uma coleção de dados. Ou seja, é um objeto que possui diversos valores, a diferença entre um dicionário e uma lista é que para acessar um item de um dicionário, fazemos acesso através das chaves, e não do índice.

Vamos ver como funciona na prática:

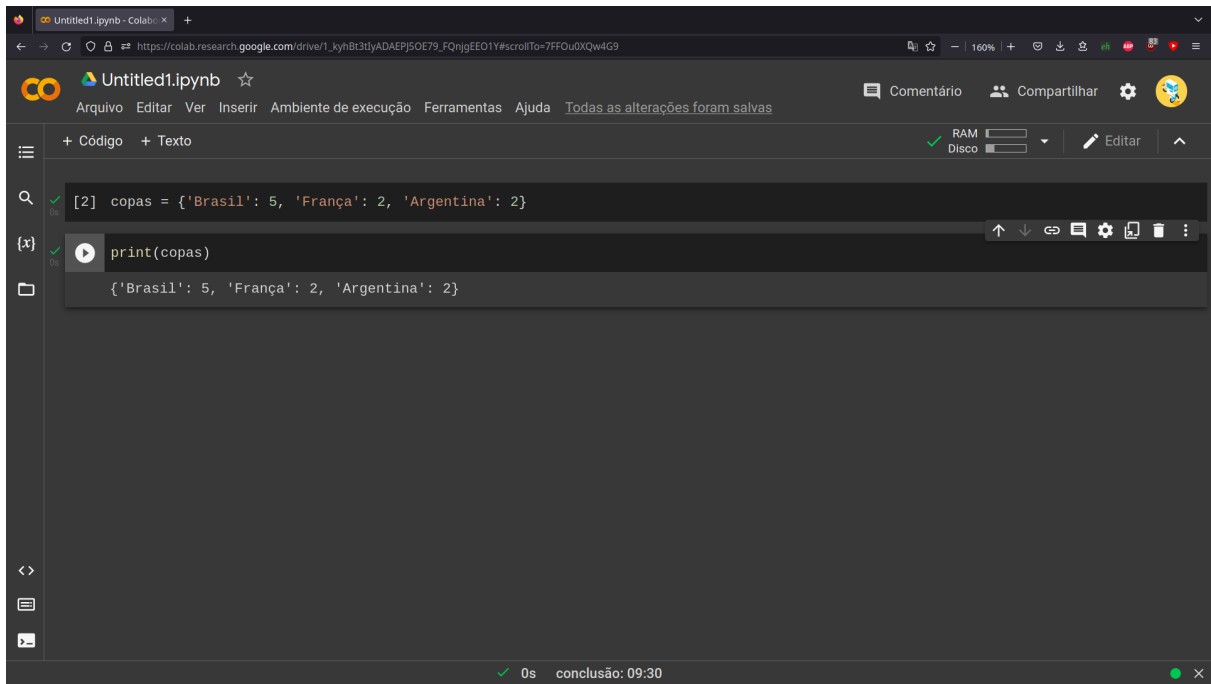
Para declarar um dicionário fazemos o uso de chaves ao invés de parênteses como nas listas.



The screenshot shows a Jupyter Notebook interface with the title 'Untitled1.ipynb'. The code is as follows:

```
copas = {'Brasil': 5, 'França': 2, 'Argentina': 2}
```

Aqui temos uma variável `copas`, que representa os países e a quantidade de títulos que possuem. O que está entre aspas representa as chaves, e o que segue após os dois pontos são os valores.



The screenshot shows a Google Colab notebook titled 'Untitled1.ipynb'. The interface includes a top bar with the Colab logo, file name, and various icons. Below the top bar is a menu bar with options like 'Arquivo', 'Editar', 'Ver', 'Inserir', 'Ambiente de execução', 'Ferramentas', and 'Ajuda'. The main area is divided into a left sidebar with icons for file explorer, search, and other functions, and a central code editor. The code editor contains two cells. The first cell is a code cell with the following Python code: 

```
[2] copas = {'Brasil': 5, 'França': 2, 'Argentina': 2}
```

. The second cell is a code cell with the following Python code: 

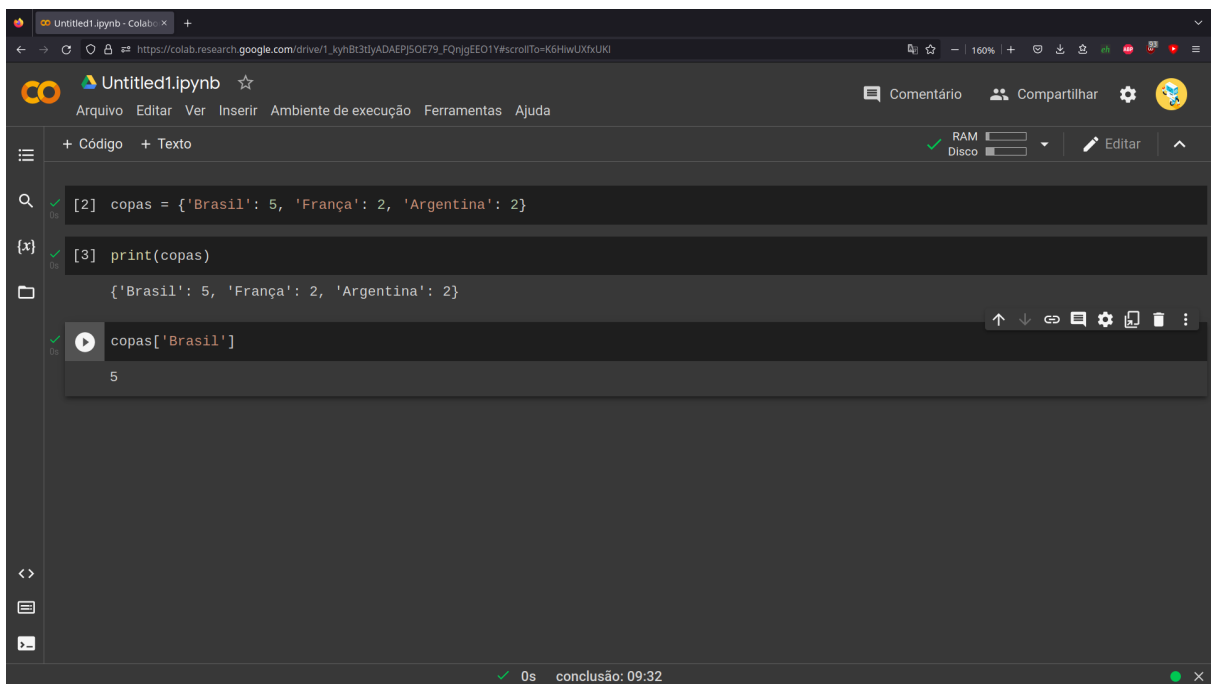
```
print(copas)
```

. The output of the second cell is a dictionary: 

```
{'Brasil': 5, 'França': 2, 'Argentina': 2}
```

. The status bar at the bottom indicates '0s' and 'conclusão: 09:30'.

Para acessar algum valor utilizamos a chave:



The screenshot shows a Google Colab notebook titled 'Untitled1.ipynb'. The interface is similar to the previous one. The code editor contains three cells. The first cell is a code cell with the following Python code: 

```
[2] copas = {'Brasil': 5, 'França': 2, 'Argentina': 2}
```

. The second cell is a code cell with the following Python code: 

```
[3] print(copas)
```

. The output of the second cell is a dictionary: 

```
{'Brasil': 5, 'França': 2, 'Argentina': 2}
```

. The third cell is a code cell with the following Python code: 

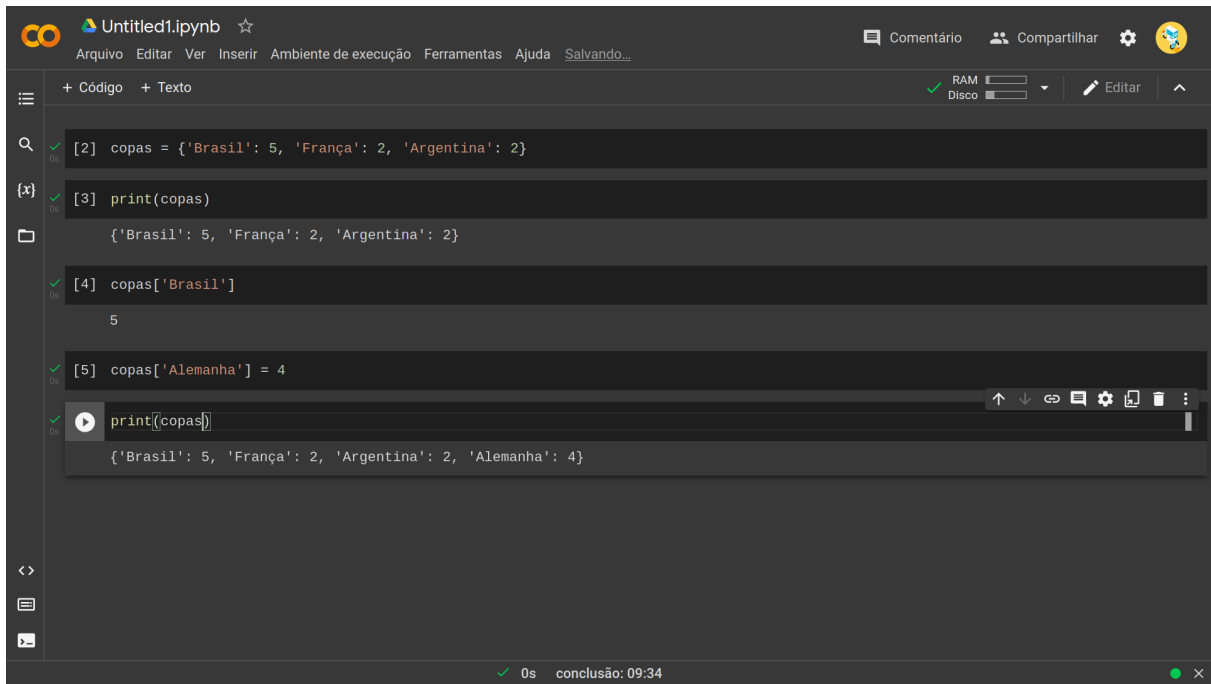
```
copas['Brasil']
```

. The output of the third cell is the value 

```
5
```

. The status bar at the bottom indicates '0s' and 'conclusão: 09:32'.

podemos também adicionar outros valores ao dicionário:



The screenshot shows a Jupyter Notebook interface with the title 'Untitled1.ipynb'. The menu bar includes 'Arquivo', 'Editar', 'Ver', 'Inserir', 'Ambiente de execução', 'Ferramentas', and 'Ajuda'. The toolbar shows 'Comentário', 'Compartilhar', and 'Editar'. The notebook contains the following code cells:

```
[2] copas = {'Brasil': 5, 'França': 2, 'Argentina': 2}

[3] print(copas)

{'Brasil': 5, 'França': 2, 'Argentina': 2}

[4] copas['Brasil']

5

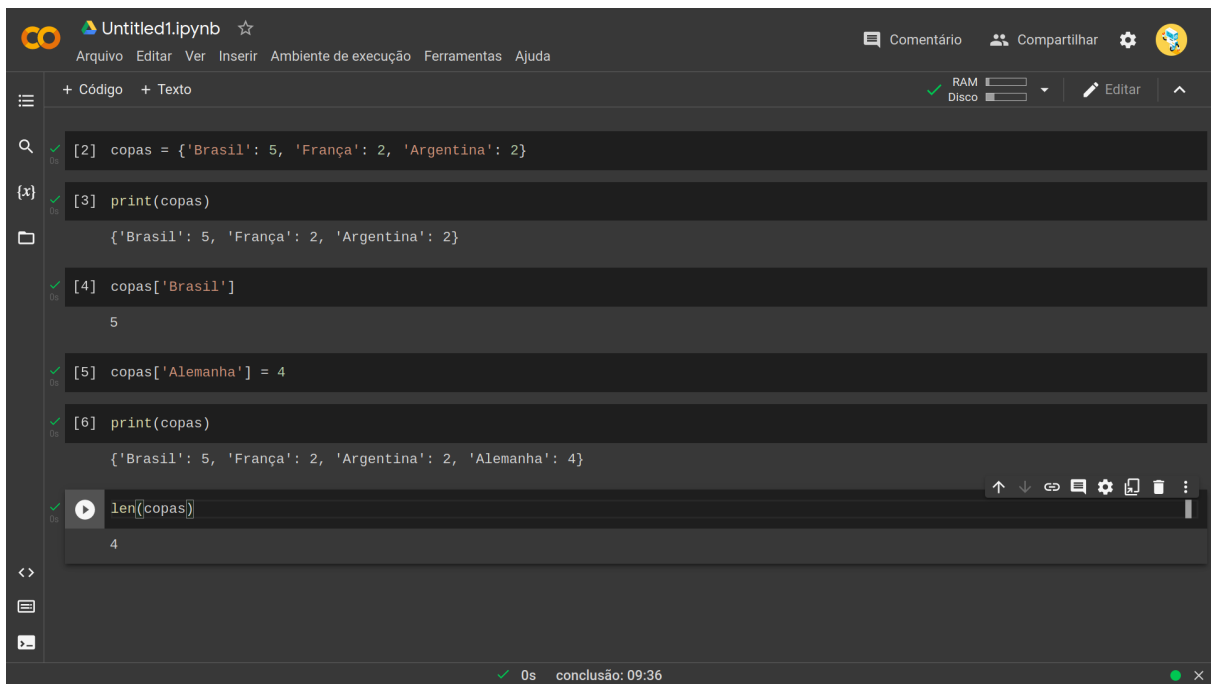
[5] copas['Alemanha'] = 4

[6] print(copas)

{'Brasil': 5, 'França': 2, 'Argentina': 2, 'Alemanha': 4}
```

The status bar at the bottom indicates '0s' and 'conclusão: 09:34'.

E com a mesma função que usamos para ver o tamanho de uma lista podemos verificar o tamanho de um dicionário:



The screenshot shows a Jupyter Notebook interface with the title 'Untitled1.ipynb'. The menu bar includes 'Arquivo', 'Editar', 'Ver', 'Inserir', 'Ambiente de execução', 'Ferramentas', and 'Ajuda'. The toolbar shows 'Comentário', 'Compartilhar', and 'Editar'. The notebook contains the following code cells:

```
[2] copas = {'Brasil': 5, 'França': 2, 'Argentina': 2}

[3] print(copas)

{'Brasil': 5, 'França': 2, 'Argentina': 2}

[4] copas['Brasil']

5

[5] copas['Alemanha'] = 4

[6] print(copas)

{'Brasil': 5, 'França': 2, 'Argentina': 2, 'Alemanha': 4}

[7] len(copas)

4
```

The status bar at the bottom indicates '0s' and 'conclusão: 09:36'.

Podemos usar também as funções `keys()` e `value()` para verificar respectivamente as chaves e os valores dos dicionários.



```
[8] copas.keys()
dict_keys(['Brasil', 'França', 'Argentina', 'Alemanha'])

copas.values()
dict_values([5, 2, 2, 4])
```

E além disso, se usarmos a função `type()` podemos verificar o tipo de variável `dict`, que é dicionário.

```
[8] copas.keys()
dict_keys(['Brasil', 'França', 'Argentina', 'Alemanha'])

[9] copas.values()
dict_values([5, 2, 2, 4])

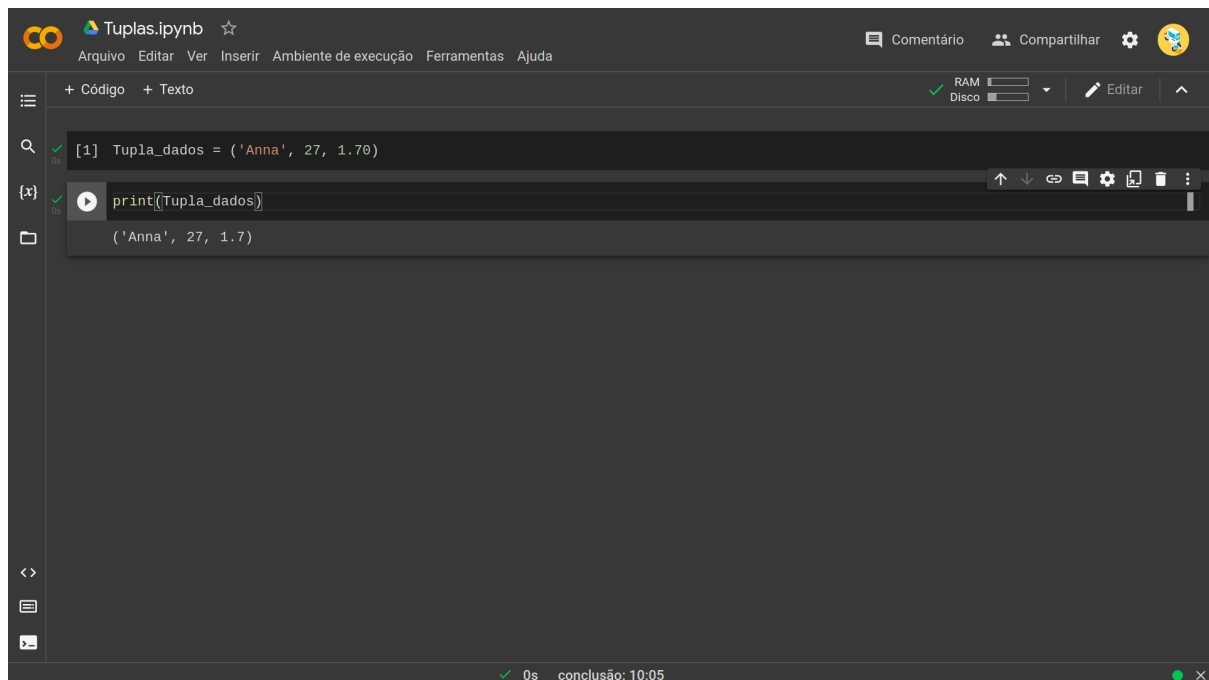
type(copas)
dict
```

Outra estrutura de dados muito utilizada no python são as tuplas.

### Tuplas:

Uma tupla no python é uma estrutura de dados que é parecida com uma lista, no entanto, a principal característica de uma tupla é ser imutável, ou seja, uma vez que declaramos uma tupla e seus valores, não podemos mais adicionar novos valores a esta, assim como nós fazemos na lista através da função `append`, e nem mesmo remover algum elemento. Esse tipo de variável é geralmente utilizado quando queremos garantir que os dados inseridos não poderão ser alterados posteriormente. É interessante dizer que por mais que uma tupla não possa ser alterada, alguns de seus elementos internos podem ser alterados, caso por exemplo

um de seus elementos seja uma lista, esta lista pode ser alterada: Vamos ver um exemplo de como declarar uma tupla. Em lista usamos, [], em dicionários usamos {} e nas tuplas vamos usar ():



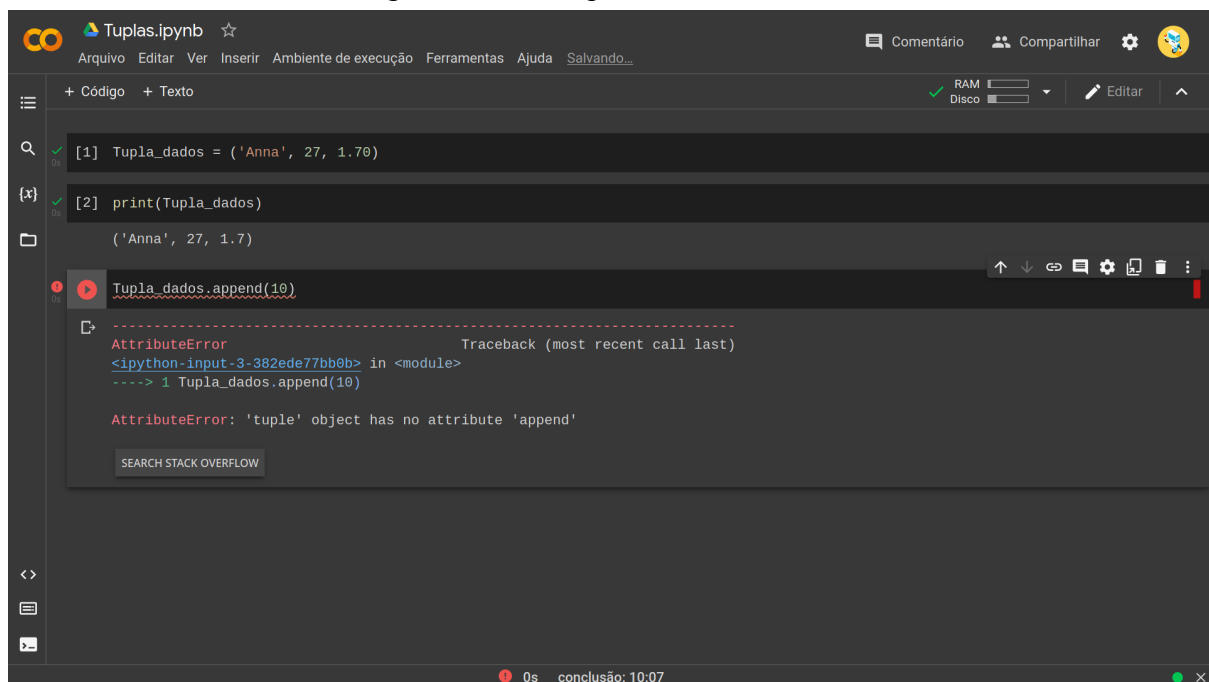
The screenshot shows a Jupyter Notebook interface with the title 'Tuplas.ipynb'. The top bar includes a menu with 'Arquivo', 'Editar', 'Ver', 'Inserir', 'Ambiente de execução', 'Ferramentas', and 'Ajuda'. On the right, there are buttons for 'Comentário', 'Compartilhar', and a settings icon. Below the menu, there are tabs for '+ Código' and '+ Texto'. The main code area contains two cells. The first cell has the code `[1] Tupla_dados = ('Anna', 27, 1.70)`. The second cell has the code `print(Tupla_dados)` and shows the output `('Anna', 27, 1.7)`. The status bar at the bottom indicates '0s' and 'conclusão: 10:05'.

```
[1] Tupla_dados = ('Anna', 27, 1.70)

print(Tupla_dados)

('Anna', 27, 1.7)
```

Mas se tentarmos adicionar algum valor na tupla como fazemos em listas:



The screenshot shows the same Jupyter Notebook interface. The first two cells are identical to the previous one. The third cell contains the code `Tupla_dados.append(10)`. Below this code, a traceback is shown, indicating an `AttributeError` with the message `'tuple' object has no attribute 'append'`. The status bar at the bottom indicates '0s' and 'conclusão: 10:07'.

```
[1] Tupla_dados = ('Anna', 27, 1.70)

[2] print(Tupla_dados)

('Anna', 27, 1.7)

Tupla_dados.append(10)

AttributeError                                Traceback (most recent call last)
<ipython-input-3-382ede77bb0b> in <module>
----> 1 Tupla_dados.append(10)

AttributeError: 'tuple' object has no attribute 'append'
```

Vamos obter um erro dizendo que append não é um atributo. Podemos verificar o tipo da variável tupla com o `type()`:

The screenshot shows a Jupyter Notebook titled 'Tuplas.ipynb'. The code cell contains the following lines:

```
[1] Tupla_dados = ('Anna', 27, 1.70)
[2] print(Tupla_dados)
('Anna', 27, 1.7)
type(Tupla_dados)
tuple
```

The output of the third line is 'tuple'. The status bar at the bottom indicates '0s' execution time and 'conclusão: 10:08'.

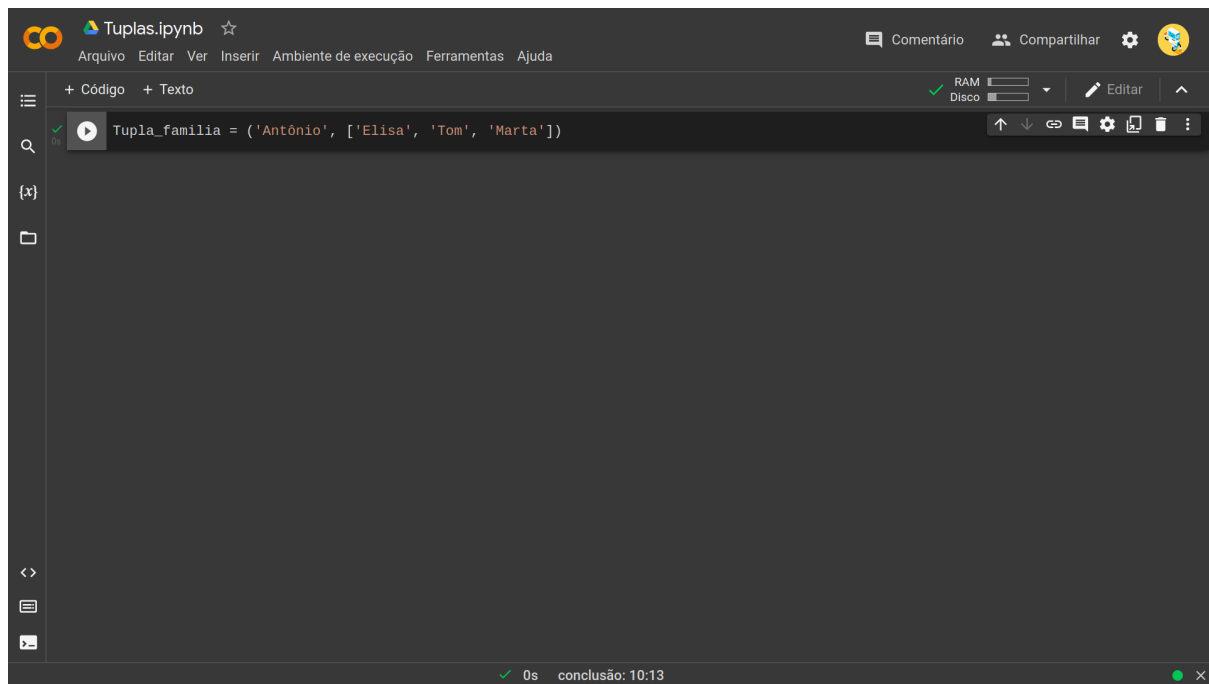
Para acessar uma posição de uma tupla fazemos da mesma maneira que fazemos em listas, através do índice:

The screenshot shows the same Jupyter Notebook with additional code:

```
[4] type(Tupla_dados)
tuple
print(Tupla_dados[0])
Anna
```

The output of the fourth line is 'tuple' and the output of the fifth line is 'Anna'. The status bar at the bottom indicates '0s' execution time and 'conclusão: 10:10'.

Mas como eu falei antes, dependendo do tipo de dado que uma tupla contém, se for uma lista, podemos alterar: Vamos criar uma tupla para guardar as informações de uma pessoa e os nomes de seu ou seus filhos:



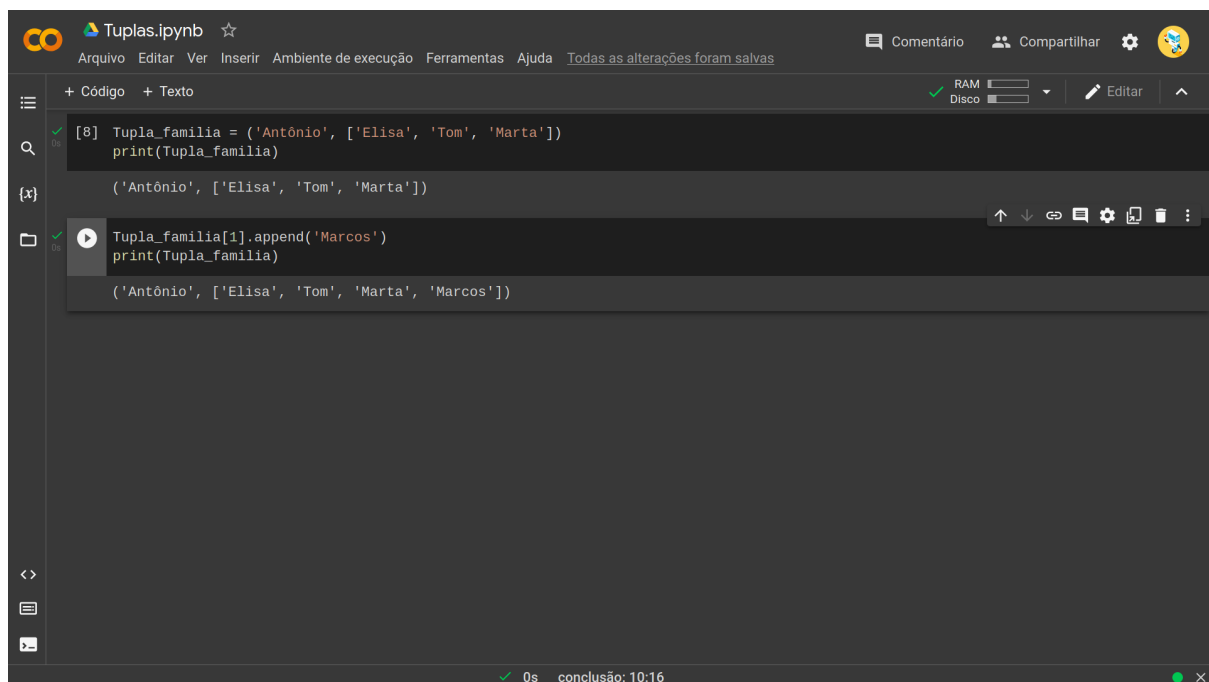
The screenshot shows a Jupyter Notebook window titled 'Tuplas.ipynb'. The top menu bar includes 'Arquivo', 'Editar', 'Ver', 'Inserir', 'Ambiente de execução', 'Ferramentas', and 'Ajuda'. On the right, there are buttons for 'Comentário', 'Compartilhar', and a settings icon. Below the menu, there are tabs for '+ Código' and '+ Texto'. The main area contains a single code cell with the following Python code: 

```
Tupla_familia = ('Antônio', ['Elisa', 'Tom', 'Marta'])
```

 The cell is marked as executed with a green checkmark and '0s'. The bottom status bar shows 'conclusão: 10:13'.

Antônio têm três filhos, mas vamos supor que ele tenha mais um, será que podemos adicionar este filho?

Vamos fazer o seguinte, vamos acessar a posição 1 da tupla, que representa uma lista com os nomes dos filhos de Antônio, e então vamos adicionar o Marcos como filho.



The screenshot shows the same Jupyter Notebook window after two more cells have been added. The first new cell contains: 

```
[8] Tupla_familia = ('Antônio', ['Elisa', 'Tom', 'Marta'])  
print(Tupla_familia)
```

 The output of this cell is: 

```
('Antônio', ['Elisa', 'Tom', 'Marta'])
```

 The second new cell contains: 

```
Tupla_familia[1].append('Marcos')  
print(Tupla_familia)
```

 The output of this cell is: 

```
('Antônio', ['Elisa', 'Tom', 'Marta', 'Marcos'])
```

 The status bar at the bottom now shows 'conclusão: 10:16'. A message 'Todas as alterações foram salvas' is visible in the top right area.

Na tupla, não conseguimos inserir novos elementos, mas em um elemento da tupla, que no caso é uma lista, podemos inserir.

Mas é possível alterar uma tupla, caso queiramos. Mas nesse caso, vamos precisar de uma variável auxiliar que seja uma lista, e através do comando `list`, vamos fazer essa lista receber os valores da tupla. Vejamos como fazer:

The screenshot shows a Jupyter Notebook titled 'Tuplas.ipynb'. The interface includes a menu bar with options like 'Arquivo', 'Editar', 'Ver', 'Inserir', 'Ambiente de execução', 'Ferramentas', 'Ajuda', and 'Todas as alterações foram salvas'. On the right, there are icons for 'Comentário', 'Compartilhar', and settings. The left sidebar shows icons for file explorer, search, and other functions. The main area contains three code cells:

```
[10] Tupla_paises = ('Alemanha', 'Brasil', 'Espanha')
print(Tupla_paises)

('Alemanha', 'Brasil', 'Espanha')
```

```
[12] variavel_auxiliar = list(Tupla_paises)
print(variavel_auxiliar)

['Alemanha', 'Brasil', 'Espanha']
```

```
variavel_auxiliar.append('Canadá')
```

The status bar at the bottom indicates '0s' execution time and 'conclusão: 10:21'.

E agora podemos usar o comando `tuple()` para transformar uma lista numa tupla:

The screenshot shows the same Jupyter Notebook interface, but with an additional code cell that converts the list back to a tuple:

```
[21] Tupla_paises = ('Alemanha', 'Brasil', 'Espanha')
print(Tupla_paises)

('Alemanha', 'Brasil', 'Espanha')
```

```
[22] variavel_auxiliar = list(Tupla_paises)
print(variavel_auxiliar)

['Alemanha', 'Brasil', 'Espanha']
```

```
[23] variavel_auxiliar.append('Canadá')
```

```
Tupla_paises = tuple(variavel_auxiliar)
print(Tupla_paises)

('Alemanha', 'Brasil', 'Espanha', 'Canadá')
```

The status bar at the bottom indicates '0s' execution time and 'conclusão: 10:23'.

Dessa forma, conseguimos manipular as informações dentro de uma tupla.

Bom, por esta aula é isto, nas próximas aulas vamos começar a ver sobre estruturas condicionais, muito obrigado por assistir até aqui, até a próxima aula.

