Aula 4: tuplas, listas, conjuntos e dicionários

This is an example of a data structure — a mechanism for grouping and organizing data to make it easier to use.

Agenda

- Revisão exercícios `while`
- tuple, list, set, dict
- `for element in iterable`
- `range(...)`

Revisão

Exercícios:
1. FizzBuzz (até N)

2. Número primo + números primos

3. Fatorial

Tuplas (`tuple`)

Exemplo 1: Login

```
###
# Sem usar tuplas
###
email_usuário_1 = "gabriel@saldanha.dev"
senha_usuário_1 = "123456"

email = input("Digite o email: ")
senha = input("Digite a senha: ")

if email == email_usuário_1 and senha == senha_usuário_1:
    print("Login bem sucedido!")
elif email == email_usuário_1 and senha != senha_usuário_1:
    print("Senha incorreta.")
else:
    print("Usuário não existe.")

###
# Com tuplas
###
```

```
usuário_1 = ("gabriel.saldanha@dev", "123456")

email = input("Digite o email: ")
senha = input("Digite a senha: ")

if email, senha == usuário1:
    print("Login bem sucedido!")
elif email == usuário_1[0] and senha != usuário_1[1]:  # [0] é o primeiro índice
    print("Senha incorreta.")
else:
    print("Usuário não existe.")
```

Exemplo 2: `type(tupla)`

```
nome = ("Gabriel")
print(type(nome))

pessoa = ("Gabriel", 25)  # nome, idade
print(type(pessoa))

pessoa = "Gabriel", 25
print(type(pessoa))

nome_tupla = "Gabriel",
tupla_vazia = ()
```


Atenção para o uso da vírgula, somente parêntesis não vai "gerar" uma tupla

Exemplo 3: swap

```
# Inverta o valor de duas variáveis
###
# Sem tupla
###
a = 10
b = 20

# Qual o problema?
b = a
a = b

# variável temporária
c = a  # c = 10
a = b  # a = 20
b = c  # b = 10

###
# Com tupla
###
a = 10
b = 20

b, a = a, b

# Então....
a, b = 10, 20  # Também é válido!
```

Exemplo 4: unpacking

```
aula = ("Aula 4", "Estruturas de dados", 20)
título, tópico, horário = aula

título, tópico = aula  # Erro!
```

Listas (`list`)

```
numeros = [1, 5, 9, 15]  # [ ] ao invés de ( )
primeiro = numeros[0]  # sempre [i] para acessar um índice, não (i)
```

```
type(numeros)  # list
```

Listas vs tuplas

Imutável vs mutável

> 💡 Uma estrutura de dados **mutável** pode ter seu valor alterado. Uma imutável não.

```
usuário_list = ["gabriel@saldanha.dev", "123456")
usuário_tuple = ("gabriel@saldanha.dev", "123456")
# Ou usuário_tuple = usuário_list[0], usuário_list[1]

# Atualizar a senha
usuário_list[1] = "654321"  # OK
usuário_tuple[1] = "654321"  # Ops...

# Entretanto, você pode alterar a variável que referenciava o valor
# mas não o valor em si.
usuário_tuple = "Qualquer outra coisa"  # É válido!
```

Referência (consequência da mutabilidade)

```
a = [1]
b = a
b[0] = 1000
print(b)  # ?
```

> 💡 Mais cuidado ainda com funções

```
def alterar_primeiro_elemento(lista):
    lista[0] = 1

minha_lista = [10, 100, 1000]
alterar_primeiro_elemento(minha_lista)

print(minha_lista)
```

Operações com *lists*

```
# len(minha_lista) => retorna o tamanho da lista
minha_lista = [1, 2, 3]
print(len(minha_lista))  # 3

# minha_lista.append(elemento) => adiciona elemento à lista
minha_lista.append(4)
print(minha_lista)  # [1, 2, 3, 4]

# minha_lista.remove(elemento) => remove elemento da lista
minha_lista.remove(4)
print(minha_lista)  # [1, 2, 3]

# ordenar uma lista
minha_lista = [4, 3, 2, 1]
print(sort(minha_lista))  # [1, 2, 3, 4]
```

Conjuntos (`set`)

💡 São mutáveis assim como listas, mas não podem ter elementos repetidos e **não garantem ordem de elementos**.

```
minha_lista = [1, 1, 1, 1, 1]
meu_conjunto = {1, 1, 1, 1, 1}

print(minha_lista)  # [1, 1, 1, 1, 1]
print(meu_conjunto)  # {1}
```

💡 Para declarar um conjunto vazio: `x = set()` e não `{}`

Operações com *sets*

```
A = {1, 2, 3}
B = {3, 4, 5}

# A - B = elementos em A que não estão em B
print(A - B)  # {1, 2}
print(B - A)  # {4, 5}

# A.union(B) => união de conjuntos
print(A.union(B))  # {1, 2, 3, 4, 5}
```

Dicionários (`dict`)

💡 Uma estrutura de dados onde uma **chave** 🔑 (ou índice) aponta para algum valor

```
# Criar um dicionário
usuários = {
    "gabriel@saldanha.dev": "123456",
    "hacker@saldanha.dev": "654321",
}

# Adicionar chave/valor ao dicionário
usuários["novo@saldanha.dev"] = "111111"
print(usuários)

# Remover chave/valor de dicionário
del usuários["novo@saldanha.dev"]
print(usuários)
```

`for in range(...)`

💡 Todas as estruturas de dados que abordamos são **iteráveis**, ou seja, podem ser **iteradas** (ou percorridas). Para isso, geralmente utilizamos o `for elemento in iterable`

```
for <nome da variável> in <estrutura iterável>:
    <instrução 1>
    <instrução 2>
    ...
```

O `for` é um outro tipo de *loop* (repetição) do Python. Geralmente ele é utilizado para percorrer os elementos de uma lista, dicionário ou tupla.

```
lista_de_usuários = ["gabriel@saldanha.dev", "hacker@saldanha.dev"]

def usuário_existe(nome_usuário):
```

```
    existe = False
    for usuário in lista_de_usuários:
        if nome_usuário == usuário:
            existe = True
    return existe
```

💡 A palavra-chave `in` pode ser utilizada para retornar se um elemento existe em uma estrutura de dados

```
lista_de_usuários = ["gabriel@saldanha.dev", "hacker@saldanha.dev"]
existe = "gabriel@saldanha.dev" in lista_de_usuários
print(existe)
```

Misturando...

💡 Podemos misturar diferentes estruturas de dados. Uma lista pode conter elementos de diferentes **tipos**, inclusive dicionários.

```
lista_de_usuários = [
    # dicionário dentro da lista
    {"email": "gabriel@saldanha.dev", "senha": "123456"},
    {"email": "hacker@saldanha.dev", "senha": "654321"},
    123,  # Um número inteiro na lista (não é uma boa prática)
]

for usuário in lista_de_usuários:
    print(usuário)  # OK

# Imprima somente os emails
for usuário in lista_de_usuários:
    print(usuário["email"])
```

Exercícios

Simple Programming Problems

Whenever I'm TA for a introductory CS class where students learn some programming language, I have trouble coming up with good exercises. Problems from Project Euler and the like are usually much too difficult for beginners, especially if they don't have a strong background in mathematics.

https://adriann.github.io/programming_problems.html

1. Escreva uma função que recebe uma lista de números inteiros como argumento e retorna o maior elemento dessa lista.

```
def maior_elemento(lista):
    ...

lista = [2, 3, 1]
x = maior_elemento(lista)
print(x)
# 3
```

2. Escreva uma função que receba uma lista (de qualquer tipo) e retorne essa lista invertida.

```
def inverte_lista(lista):
    ...

lista = ["a", 5, {1}]
lista_invertida = inverte_lista(lista)
print(lista_invertida)
# [{1}, 5, "a"]
```

3. **DESAFIO**. Um **ponto** pode ser representado por duas coordenadas: `x` e `y`. Em Python, podemos representá-lo utilizando uma tupla. Escreva um programa que solicite ao usuário os valores `x` e `y` para **dois pontos** (ou seja, serão solicitados 4 valores ao todo) e depois imprima a **distância entre esses dois pontos**.
4. **DESAFIO**. Uma **string** também é **iterável**, ou seja, pode ser percorrida. Quando fazemos `for x in "abc"`, para cada iteração, `x` vai receber o valor de um dos caracteres de "abc"

```
for x in "abc":
    print(x)
# Vai imprimir:
# a
# b
# c
```

Escreva um programa que solicite uma string para o usuário e imprima quantas vezes cada letra aparece na palavra. Por exemplo:

```
"banana"
# O resultado deve ser
{
    'a': 3,
    'b': 1,
    'n': 2
}
```