# 11 - Indentação, comentários e formatação

**Transcrição**

Para encerrarmos essa introdução ao básico de Python, conversaremos um pouco sobre indentação, comentários e formatação de strings. A **indentação**, que é o recuo que utilizamos no código, é uma boa prática em qualquer linguagem de programação, já que facilita a leitura e a manutenção do código, dentre em coisas. Além disso, no Python ela é obrigatória, e sua ausência incorre em comportamentos indesejados ou até mesmo em erros.

Na seção **2.5 Indentação, comentários e formatação de strings**, encontraremos alguns códigos previamente escritos que nos ajudarão a entender esses conceitos,a começar pelo trecho abaixo:

ano\_atual = 2019

ano\_fabricacao = 2019

if (ano\_atual == ano\_fabricacao):

print('Verdadeiro')

else:

print('Falso')COPIAR CÓDIGO

Aqui estamos criando duas variáveis, ano\_atual e ano\_fabricacao, e o valor 2019 está sendo atribuído às duas. Logo após, temos uma cláusula if que verifica se ano\_atual é igual a ano\_fabricacao. Em caso positivo, a mensagem "Verdadeiro" é impressa no console; caso contrário (else), a mensagem "Falso" será exibida. Ao executarmos o código, como esperado, a mensagem "Verdadeiro" é mostrada na tela.

A linha recuada print('Verdadeiro') indica que essa linha de código está dentro da cláusula if acima dela. Mas o que acontece se removermos esse recuo?

ano\_atual = 2019

ano\_fabricacao = 2019

if (ano\_atual == ano\_fabricacao):

print('Verdadeiro')

else:

print('Falso')COPIAR CÓDIGO

Ao executarmos, teremos um *IndentationError*, ou seja, justamente um erro de indentação, pois o Python não consegue interpretar que a linha print('Verdadeiro') está dentro da cláusula if. Ao incluirmos o recuo antes do código, tudo volta a funcionar normalmente.

Outra boa prática em programação é a inclusão de **comentários** no código. O interpretador do Python ignora os comentários, portanto é permitir escrever qualquer coisa que você desejar, como o que aquele código está fazendo, se ele afeta outra parte do programa, se precisa dos dados de uma fonte específica e assim por diante. Para comentários em Python, usamos o símbolo #. No Google Colab, o atalho "Ctrl + ;" permite comentar ou descomentar uma linha de código.

# Isto

# é um

# comentário

ano\_atual = 2019

ano\_atualCOPIAR CÓDIGO

Existe ainda mais uma forma de incluir comentários no Python, que são as aspas simples triplas.

'''Isto é um

comentário'''

ano\_atual = 2019

ano\_atualCOPIAR CÓDIGO

A seguir temos um exemplo do código que rodamos acima, agora com alguns comentários:

# Definindo variáveis

ano\_atual = 2019

ano\_fabricacao = 2019

'''

Estrutura condicional que vamos

aprender na próxima aula

'''

if (ano\_atual == ano\_fabricacao): # Testando se condição é verdadeira

print('Verdadeiro')

else: # Testando se condição é falsa

print('Falso')COPIAR CÓDIGO

A execução desse código ocorrerá normalmente, como se os comentários não existissem. Agora conversaremos um pouco sobre a **formatação de strings**, muitas vezes necessária para melhorar a leitura das saídas do nosso código. Trabalharemos mais especificamente com a funçãostr.format() e o formato f-Strings, mais recente e que vem sido bastante utilizado.

O primeiro passo será incluirmos no código a chamada de print(), uma função embutida do Python que recebe parâmetros e os imprime no console. Fora do Colab, ela é necessária para visualizar os retornos dos códigos que estamos executando. Passaremos para a função o parâmetro 'Olá, {}!' seguido de .format(), um método que pode ser executado a partir de um objeto do tipo string. Esse método receberá "Rodrigo", o nome do nosso instrutor.

print('Olá, {}!'.format('Rodrigo'))COPIAR CÓDIGO

O parâmetro passado para o format() será colocado dentro das chaves ({}) da string anterior. Sendo assim, nosso retorno será:

Olá, Rodrigo!

Podemos ter mais de um valor desse tipo. Como exemplo, vamos supor que estamos trabalhando em um sistema de autenticação e queremos exibir uma mensagem com o nome do usuário e o seu número de acessos. Para isso, passaremos ao print() a string 'Olá, {}! Este é seu acesso de número {}' seguida da chamada de format() contendo os parâmetros Rodrigo e 32. Repare que agora temos duas chaves e elas são posicionais - a primeira delas receberá o primeiro parâmetro, e a segunda o segundo parâmetro.

print('Olá, {}! Este é seu acesso de número {}'.format('Rodrigo', 32))COPIAR CÓDIGO

Olá, Rodrigo! Este é seu acesso de número 32

Também podemos fazer isso nomeando as variáveis. Nesse caso, criaremos, dentro do próprio método format(), a variável nome recebendo 'Rodrigo' e a variável acessos recebendo 32. Dessa forma, não precisaremos mais trabalhar de maneira posicional, podendo passar dentro das chaves as variáveis que acabamos de criar.

print('Olá, {nome}! Este é seu acesso de número {acesso}'.format(nome = 'Rodrigo', acesso = 32))COPIAR CÓDIGO

Olá, Rodrigo! Este é seu acesso de número 32

O formato mais novo, f-Strings, é bem mais simples de entender e aplicar. Por exemplo,

nome = 'Rodrigo'

acessos = 32COPIAR CÓDIGO

Após rodarmos essas variáveis, executaremos o mesmo print() que criamos acima, dessa vez adicionando o caractere f antes da string.

print(f'Olá, {nome}! Este é seu acesso de número {acessos}')COPIAR CÓDIGO

Como retorno, teremos:

Olá, Rodrigo! Este é seu acesso de número 32

Quando adicionamos o f, o Python pega as variáveis armazenadas na memória e passa para a string nos pontos em que determinamos, de um jeito mais elegante e simples que a função format(). No próximo capítulo começaremos a trabalhar com as listas em Python e aprenderemos alguns parâmetros de comparação entre listas.