**String em Python**

**Extraindo informações de uma URL**

1. **Aula 1 – Introdução e Fatiamento de Strings:**
   1. Em uma URL existem parâmetros no final que funcionam como variáveis passadas para o nosso programa.
      1. Temos como exemplo essa imagem:



* + 1. Começa com https:// e o site em que estamos, em seguida, temos /Search? Que indica que estamos na página de pesquisa.
    2. O *q=* é a nossa primeira variável que indica query, ou seja, indica qual a busca/pesquisa que eu fiz. Nesse caso indicando alura.
    3. &: é o que faz a separação das variáveis.
    4. Src: source = indica origem, que nesse caso é typed\_query, ou seja, pesquisa digitada.
    5. Isso tudo indica para o twitter que fizemos uma pesquisa digitada por alura na página de busca do site twitter.
  1. Criando um exemplo: Precisamos fazer o cambio de moedas no site bytebank. O URL ficaria mais ou menos assim:
     1. <https://bytebank.com/cambio?moedaOrigem=real&moedaDestino&quantidade=100>
     2. Na URL acima temos que: queremos fazer o cambio de 100 unidades de dinheiro do real para dólar no site bytebank.
  2. Podemos fazer o fatiamento de strings.
     1. Quando setamos uma string em uma variável, podemos pedir para o python imprimir o que está em alguma posição, como na posição 0, por exemplo.
     2. O que python irá exibir é a letra que se encontra naquela posição. E.:



* + 1. Lembrando que a contagem de posições do python é de 0 até o -1 o len() da string.
    2. Também podemos pedir para que ele devolva o que está em mais de uma posição de uma vez, ou seja, conseguimos ver o que está da posição 1 desejada até a posição 2, separada por “:”. Ex.:



* + 1. Porém, no fatiamento, temos que a posição/argumento 1 é inclusivo e o 2 é exclusivo, ou seja, quando pedimos para exibir o que está da posição 0 até a 1, o que ele exibirá será somente a primeira letra, pois excluiu o que está na posição 1. Exs.:



* + 1. Porém, apesar de a segunda posição escolhida é exclusiva, podemos acrescentar um somador na hora de pedir a devolução do valor, desse modo tornando a segunda posição inclusiva também. Ex.:



* 1. Strings uma vez que colocadas se tornam imutáveis. Tudo o que fazemos com elas depois disso é apenas cópia da original.
     1. Para saber mais sobre imutabilidade: <https://blog.saldanha.dev/imutabilidade-de-strings-no-python>
  2. O que aprendemos:
     1. URLs e seus formatos: como as URLs funcionam e o que cada parte de uma URL significa - base e parâmetros;
     2. O operador de fatiamento [a:b], utilizado para obter uma substring desde o índice a até o índice b - 1 da string original. Lembrando que b - 1 pois o segundo argumento do fatiamento é exclusivo;
     3. A string original não é alterada ao ser fatiada devido à sua imutabilidade.

1. **Aula 2 – Utilizando Métodos de Strings:**
   1. O problema de usar o fatiamento para separar o que é site/domínio do que é parâmetro é que estamos considerando que em todas as url’s o “?” que faz essa separação está na mesma posição, o que não é verdade.
      1. Levando isso em consideração, o que nós temos que fazer é utilizar o método *find()*. Ex.:



* + 1. Repare que ele retornou 2, sendo essa a posição da letra na string. Lembrando que a contagem sempre começa em 0.
    2. Utilizando esse método, podemos fazer com que a separação sempre seja feita através da posição do “?” independentemente de onde seja. Ex.:



* + 1. No url\_parametros nós colocamos +1 justamente para retirar o “?” do fatiamento. Lembrando que a primeira posição é inclusiva e a segunda exclusiva, por isso na url\_base não precisamos colocar -1, por exemplo. Essa matemática já é feita por padrão.
  1. Um outro problema que temos é que o nsso final do parâmetros ainda está fixo, e nós não sabemos qual é o tamanho de todas as URL’s e ficar contando e alterando o nosso programa para cada uma delas não é muito prático.
     1. A nossa solução para isso é simples: Só deixar o segundo argumento de posição vazio, dessa forma ele irá pegar todo o resto que estiver na URL. Ex.:



* + 1. Dessa forma o python entende que você quer ir daquele início passado até o final da string.
    2. O mesmo vale para o início das strings, estávamos colocando o 0 até agora, mas podemos simplificar e deixar sem nada que o resultado será o mesmo. Ex.:

