[00:00] Fala, galera!
[00:01] Sejam muito bem-vindos a esse curso de Python. Neste curso, vamos dar foco nas strings que o Python tem. Nos métodos strings e como manipular e retirar informações importantes de dentro dessas strings.

[00:14] Vamos começar criando uma classe responsável por extrair argumentos e informações de dentro de URLs. Para quem não sabe, URL é aquela barra de endereço dos sites.

[00:25] Depois disso, de criar a classe e deixar ela bem robusta, com muitos métodos importantes para nós, veremos o que são métodos estáticos e métodos de distância.

[00:35] Vamos começar a utilizar expressões regulares dentro do Python, que são formas muito bacanas de extrair informações de dentro de mensagens de WhatsApp, de postagens de Facebook, de dentro de e-mail, de mensagem de texto, de qualquer coisa, de qualquer texto humano.

[00:51] Além disso, vamos olhar a documentação do Python sempre que tivermos alguma dúvida de algo bem específico da linguagem, lembrando que a documentação de uma linguagem de programação é a principal fonte de conhecimento que você pode ter em relação a essa linguagem.

[01:12] Depois disso, vamos começar a utilizar métodos especiais, que são uma forma bem bacana que o Python tem de dar mais funcionalidades às nossas classes. Nesse processo de criar métodos especiais, vamos aprender um pouco mais sobre o Python, como ele funciona, e ainda mais sobre essa linguagem incrível que é o Python.

[01:32] Quero convidar vocês a aprender mais coisas e colocar mais ferramentas na caixa de ferramentas de desenvolvedores de vocês. Te espero na próxima aula.

[00:00] Fala, pessoal. Bem-vindos ao nosso curso de Python.
[00:01] Quero começar a introduzir a vocês o que é uma string. Dar um conceito bem básico para nas próximas aulas começarmos a criar nosso projeto aplicando alguns métodos mais avançados das strings.

[00:15] Primeiramente, uma string é uma sequência de caracteres um do lado do outro. Isso significa que se eu tenho várias letras uma do lado da outra, tenho uma palavra. Se tenho várias palavras juntas, tenho uma frase. Se tenho várias frases juntas, tenho um texto, uma música, um livro.

[00:37] Uma string, em essência, serve para trabalharmos com textos. Como criamos isso dentro do Python? Vou começar criando uma variável que vai ser uma string e vou mostrá-la na tela.

[00:51] Quero uma variável bem simples, o meu nome, por exemplo: meuNome = Rodrigo

[01:01] E vou dar um print: print (meuNome)

[01:05] Não se preocupe com a estrutura do PyCharm que na aula que vem vamos criar um projeto juntos. Veja só, deu erro, disse que a o nome não está definido, porque o Phyton buscou uma variável com o nome de Rodrigo. Ele não mostrou uma string para nós. Ele tentou mostrar uma variável.

[01:26] Toda string precisa estar entre aspas para poder ser uma string: meuNome = “Rodrigo”

[01:36] Printando de novo, funciona normalmente. Podem ser aspas duplas ou aspas simples, mas tem que ser o mesmo tipo de aspa no início e no fim: meuNome = ‘Rodrigo’

[01:53] Percebe que não precisei dizer para o Phyton o que é essa minha variável? Não precisei colocar antes de tudo um string, ou um str, por exemplo. Ele inclusive acusa um erro se eu tentar fazer isso, porque o Phyton é uma linguagem dinamicamente tipada. Ele sozinho identifica qual o tipo de variável está sendo trabalhado naquele momento.

[02:18] Falando em tipo, o Phyton também possui uma função bem bacana, que é a função type. Com ela, consigo mostrar ao usuário ou para mim mesmo que aquela variável é do tipo que quero que ela seja: print (type(meuNome))

[02:39] Rodando, ele me diz que a classe é str. A classe str é a classe do Python responsável por lidar com strings em geral.

[02:50] Agora quero criar outra variável qualquer e ver o tipo dela: minhaIdade = 26

[02:59] Vou printar essa variável: print (minhaIdade). E vou printar também o tipo: print (type(minhaIdade))

[03:12] Nós sabemos que 26 é um inteiro, e é isso que o Python nos diz. É da classe INT, que no Python é responsável por lidar com números inteiros. Se eu adicionar um .3, ele me diz que é do tipo FLOAT, porque não é mais um inteiro.

[03:38] Vou apagar esses prints antigos e vou criar a variável: sobreMim = ‘Meu nome é Rodrigo e minha idade é 26’. Vou printar isso: print(sobreMim), e também vou ver o tipo: print (type(sobreMim).

[04:10] É um texto com um número dentro. Será que ela é inteira ou uma string? O Phyton acusa como uma string, porque está entre aspas. É um texto, não é mais uma palavra qualquer. É uma coisa maior, é um conjunto de palavras.

[04:29] Na aula que vem vamos ver os métodos que veremos ao longo do curso, e também vamos começar a fazer o fatiamento, a retirar sub-strings de dentro de uma string maior. Por exemplo, se eu quisesse ao invés de pegar a string completa pegar somente minha idade, na aula que vem vamos começar a descobrir como fazer isso.

[00:00] Fala, pessoal. Bem-vindos a mais esse vídeo. Nele, quero começar com vocês do zero um projeto do PyCharm.
[00:08] Clicando no programa, ele já renderizou para mim uma janela com opções de criar projeto, abrir um projeto. Vou escolher criar um novo. O nome do meu projeto vai ser “Python”. Simples e direto.

[00:28] Ele me dá já algumas dicas. Vocês podem ver se quiserem. Tenho do lado esquerdo a pasta que simboliza o meu projeto. Posso clicar com o botão direito nela e escolher criar um arquivo novo. O nome vai ser “main”. Posso também minimizar ou maximizar a aba projeto.

[00:59] Agora, quero começar de onde paramos na aula passada. Lembram que tínhamos uma variável que era sobreMim = ‘Meu nome é Rodrigo e minha idade é 26”. O desafio era conseguir retirar somente o 26 de dentro dessa string, para dizer somente minha idade.

[01:27] Antes disso, vou começar por uma variável mais simples: meuNome = ‘Rodrigo’. Quero dizer para vocês que até certo ponto uma string do Python é muito semelhante a uma lista, porque cada caractere dentro da string possui um índice que posso acessar e mostrar somente esse índice específico.

[01:51] Na linha de baixo, vou colocar: #.....0123456

[01:57] Fiz isso para a linha começar no R de Rodrigo, que tem índice 0, 1, 2, e assim por diante até chegar ao final do nome, com o O no final que tem índice 6.

[02:08] Eu consigo criar uma variável subString= meuNome para acessar um argumento específico. Mas tem uma notação certa para poder acessar esse argumento. Preciso adicionar um colchete: subString= meuNome [ ]

[02:30] Se eu quiser pegar o I de Rodrigo, coloco dentro desse colchete o valor 4, que é o índice desse caractere. Dando um print, o Python nos mostra no console o I, que é representado pelo índice quatro.

[03:02] Mas consigo fazer muito mais coisas. Se eu colocar dentro do colchete dois pontos, consigo começar em algum índice específico e ir até um índice específico, fazendo o fatiamento da string. Por exemplo, se eu quiser pegar o drig, coloco: subString= meuNome [2:5]

[03:36] Rodando isso, ele me dá um dri. Algo deu errado. Eu coloquei os índices corretos, mas não veio tudo que eu queria, porque o índice que inicia o fatiamento é inclusivo, mas o índice que finaliza é exclusivo. Preciso colocar um índice maior do que ele para poder retornar o que eu quero: subString= meuNome [2:6]

[04:02] Tem mais dois detalhes interessantes para vermos. Por exemplo, se eu não colocar o primeiro índice, ele vai pegar do começo, do índice 0, até o índice que indiquei depois. O que faz sentido. Se eu não coloquei nada, o Python entende que quero começar do início da minha string.

[04:23] O contrário é válido também. Se eu quiser colocar o primeiro índice e deixar o segundo índice vazio, o Python sabe que eu quero ir até o final da minha string.

[04:38] Lembram do nosso desafio? O desafio maior agora é conseguir contar quantos índices existem até chegar no 26. Eu vou chutar, colocando: subString= sobreMim [27:]. Ele me mostrou a partir de “idade”. Depois, eu conto quantos faltam até o 26. Somando, dá 35. Se eu colocar agora: subString= sobreMim [35:], ele me dá o 26.

[05:26] Esse sobreMim é uma string interessante, importante, mas existe uma string mais importante ainda que usamos o tempo todo e não damos muita atenção. É a URL de um site. Aquele endereço que fica na barra do navegador que você usa.

[05:50] Acessando o YouTube, por exemplo, vamos supor que eu quero ouvir Wesley Safadão. No momento em que eu der um enter na busca, a URL muda. Depois do endereço principal do YouTube e de “results”, tenho uma interrogação, um caractere especial da URL. Ela indica que estou começando a passar alguma informação. Mas passar o quê e para onde?

[06:34] Tenho que passar um argumento, search_query, que é minha pesquisa, para o back end do YouTube, para a aplicação YouTube, e aí sim ele vai conseguir acessar o banco de dados e renderizar para nós o que queremos.

[06:51] Será que só o YouTube faz isso? Outra coisa mais simples ainda que fazemos no dia a dia é uma pesquisa no Google. Vou pesquisar por Alura. Percebam como novamente a URL mudou. E a interrogação está lá, indicando que estou passando alguma informação para o Google. Temos o q, de query, provavelmente, que separa o nome do valor, que foi Alura.

[07:25] Aqui tenho mais coisas ainda. Tenho o &, que indica que estou passando mais de um argumento para o back end do Google, para ele poder resolver o que ele tem que resolver e fazer as buscas necessárias. Faz sentido, porque talvez o método de uma classe nossa precise de mais de uma coisa para funcionar, mais de um argumento. É aí que esse cara entra em ação, indicando que tem vários argumentos sendo passados.

[07:59] Vamos voltar para o PyCharm. Quero apresentar para vocês um banco chamada ByteBank. Ele não existe, mas já possui uma aplicação web, que faz transformações cambiais. A URL dele completa é a seguinte: url = "https://www.bytebank.com.br/cambio?"
[08:39] Aí começo a passar os argumentos para esse endereço do ByteBank. Então, preciso de uma interrogação. Mas antes, quantos argumentos preciso para conseguir fazer uma transformação cambial?

[08:46] Supondo que eu queira transformar 1,500 de real para dólar. Três argumentos. Preciso do valor, da moeda origem e da moeda destino. Vamos colocar isso na URL: url = "https://www.bytebank.com.br/cambio?moedaorigem=real&moedadestino=dolar&valor=1500"
[09:14] Tenho aqui a URL completa que preciso para fazer essa transformação cambial. Antes disso, vamos dar um zoom em um desses argumentos e vamos fazer um fatiamento com esse argumento usando o que já vimos de string.

[09:34] Ao invés de URL, vou colocar argumento. Vamos dar uma olhada no argumento "moeda origem": argumento = "moedaorigem=real". Supondo que eu já queira pegar o nome desse argumento usando fatiamento, vou comentar: #.....012345678910

[09:59] Depois: subString = argumento [:11]. Preciso colocar até o índice 11 para conseguir capturar até o M. Dando um print na subString, ele me dá o nome desse argumento. Mas e se eu quisesse pegar só origem? Eu teria que colocar o índice do O, que é 5: subString = argumento [5:11]

[10:42] Agora que já vimos que conseguimos retirar o nome desse argumento, vamos supor que quero colocar o endereço completo do ByteBank: argumento = "https://www.bytebank.com.br/cambio?moedaorigem=real&moedadestino=dolar&valva=1500"
[11:05] E se eu rodar o código agora? Apareceu algo sem significado nenhum. O que aconteceu foi que eu coloquei um prefixo na minha string e isso acabou com o meu código. O código que acabamos de fazer para retirar um pedaço do argumento, um nome do argumento, estaria completamente destruído. Eu não conseguiria usar mais ele da forma como meu código está.

[11:32] Será que sempre que eu mudar alguma coisa na minha URL preciso abrir o código fonte do ByteBank e modificar o código fonte? Ou será que existe uma forma mais simples e dinâmica de resolver esse problema? Aula que vem veremos isso.

[00:00] Lembram que o desafio que eu propus na aula passada foi conseguir capturar de forma dinâmica alguma subString dentro da nossa URL? Quando eu digo dinâmica, quero dizer que quero pegar um pedaço específico, independente se tem algo antes ou não, e do que tem antes ou não.
[00:18] Eu falei que o Python possui um método bem bacana para isso. Vou voltar para o argumento = 'moedaorigem=real'. Supondo que eu queira somente a palavra real, somente esse valor do meu argumento, nós sabemos fazer o fatiamento, mas o problema é que se eu colocar qualquer coisa antes, teríamos que ir ao código e mudar os índices do meu fatiamento.

[01:05] Quero apresentar para vocês um método de string chamado find. Como o nome já diz, ele encontra alguma coisa. Posso encontrar algum caractere que eu queira dentro da minha string argumento: index = argumento.find ()

[01:25] Para achar o real, é bem simples. Ele vem logo depois do símbolo de igual. Então, peço para ele encontrar o real: index = argumento.find (“=”). O nome index não foi coincidência. O método find retorna o índice do caractere buscado dentro de uma string. Caso ele não encontre, ele retorna -1, para nos indicar que não existe.

[02:03] Agora, vamos usar esse índice dentro do nosso fatiamento: subString = argumento [index:]

[02:30] Printando isso, ele me mostra =real. Lembram que o primeiro item do fatiamento é inclusivo e o segundo é exclusivo? Preciso colocar um + 1: subString = argumento [index + 1:]

[02:50] Agora consigo pegar simplesmente o real. O M está na posição 11. Quando coloco o +1, começo no índice 12, pegando do R em diante.

[03:12] Assim, eu eliminei aquele problema de ter um prefixo. Se eu colocar o ByteBank de novo, o endereço de câmbio, e rodar, ele somente vai nos dar o real. Meu código agora procura o igual. Retorna o índice para mim e eu passo esse índice para o fatiamento.

[03:35] Mas tem outra forma de resolver esse problema com outra função bem legal do Python, que é o split. Ela transforma uma string em um vetor, recebe de argumento um separador. Em alguns casos, esse método pode ser bem eficaz.

[03:56] Eu não vou mais receber o índice. Vou receber uma lista: listaargumentos = argumento.split(“=”). O que separa o nome do meu argumento do valor do meu argumento, é o sinal de igual. Printando isso, ele cria um vetor e colocou uma string com um nome e uma string com o valor.

[04:40] Essa função é bem bacana. Para casos em que o separador é bem definido e que a string não é tão complexa, ela é muito boa. Mas quando a string fica mais complexa, com mais argumentos, com mais separações não tão específicas, acaba não sendo tão útil.

[05:05] Por enquanto é só. Espero que vocês façam os exercícios, e qualquer dúvida, não hesite em perguntar no fórum. Todo mundo vai estar disposto a responder e te ajudar. Espero vocês na próxima aula.

Nesta aula, você foi apresentado à um padrão comum de criação de URLs com argumentos. O banco ByteBank, por exemplo, possui uma aplicação WEB para o cálculo cambial. Um cliente que deseja calcular o valor de 1.500,00 reais em dólares irá acessar o seguinte endereço:

```
www.bytebank.com.br/cambio?`valor=1500&moedaOrigem=real&moedaDestino=dolar
```

Como nossa aplicação será capaz de separar cada parâmetro? Vamos começar utilizando fatiamento e o método find() em uma string mais simples, visando separar o endereço de uma página dos argumentos passados para a mesma.

Comece criando uma variável com o nome de url como abaixo:

```
url = “pagina?argumentos”
```

Dessa url queremos somente a porção onde os argumentos se encontram, existem diversas formas de fazer isso, vamos fazer utilizando o find().

Lembre-se que o find() precisa receber um separador como argumento e perceba que quem está separando “pagina” de “argumentos” é um sinal de “?”.

Então você precisa fazer um fatiamento que receba como argumento o índice retornado pelo método find() recebendo “?” como parâmetro de busca e além disso somar + 1 à esse índice.

```
url = “pagina?argumentos”
indice = url.find(“?”)
print(url[indice + 1:] )
```

Nessa aula, aprendemos:

- Como retirar substrings de dentro de nossa string utilizando fatiamento.
- A encontrar índices de forma mais dinâmica utilizando o find()

Na próxima aula iremos criar uma classe que será responsável por retirar os argumentos de dentro da url.

AULA 2

[00:00] Sejam bem-vindos a mais essa aula do nosso curso. Espero que estejam gostando e que estejam fazendo os exercícios. Vamos começar.

[00:06] Quero começar a construir a classe que vai ser responsável por extrair os argumentos de dentro da nossa string. É bom sempre ter uma classe, para organizar melhor o código, você cria métodos bem definidos, que fazem coisas bem definidas. Você pode reutilizar essas classes e métodos em algum lugar depois, caso você queira fazer manutenção no seu código é mais fácil, porque vai estar tudo organizado.

[00:36] Chega de papo e vamos criar a classe. Vamos criar um arquivo dentro da mesma pasta com o nome ExtratorArgumentosUrl. E vamos criar uma classe dentro desse arquivo com o mesmo nome: class ExtratorArgumentosUrl

[00:57] Em que momento queremos receber a url dentro de algum lugar? Dentro do nosso código main, no caso, e futuramente dentro da nossa aplicação web do Byte Bank. É bom receber o valor no momento em que o objeto é criado, no momento em que estanciamos essa classe.

[01:15] Precisamos de um construtor para receber esse valor: def init(self, url):
[01:26] Até aqui tudo certo. Posso simplesmente criar um self.url = url. Vai funcionar naturalmente. Mas, e se der algum bug dentro do sistema do Byte Bank ou se o usuário resolver passar algum valor vazio para nossa classe? Temos que deixar nossa classe pronta para lidar com esse tipo de problema.

[01:57] Antes, quero mostrar um conceito através de uma imagem. Existe diferença entre uma coisa vazia e uma coisa nula. Uma coisa vazia de fato existe, só que está vazio. E uma coisa nula nem sequer existe. Para situar melhor, null no Python é igual a none.

[02:25] Vamos fazer alguns testes no código. Vou criar a variável: string = None. Vou dar um print (string is None). Estou perguntando para o Python. Ele diz que é verdade.

[03:07] Mas se eu colocar uma coisa vazia: string = “”. Ele vai dizer que é falso, porque essa pergunta é muito específica. Eu estou perguntando só se é nome. Não perguntei se é vazio ou nome. A boa notícia é que em geral o if espera um booleano, mas no Python, quando você não passa um booleano para o if, ele também consegue tratar essa informação que foi passada da seguinte maneira: if string: print (“Tem algo Aqui”) else: print(“Não tem nada Aqui”)

[04:04] Se eu coloco string = “Rodrigo”, ele me diz que dentro da minha string existe alguma coisa. Mas se eu colocar string = None, ele me diz que não existe nada. Foi para a segunda condição. E se eu não passar nada: string = “”, ele me diz que não tem nada.

[04:39] O Python sabe diferenciar quando uma string é vazia usando o if. Vamos fazer um teste colocando string = 12. Ele indica que tem algo. Se eu passar string = 0, ele diz que não tem nada.

[04:58] A nossa ideia agora é conseguir jogar essa lógica de alguma forma para dentro da nossa classe. É sempre bom criar métodos para organizar bem o código e não embolar muita coisa dentro de um método só. def urlEhValida (self, url): if url: return True else: return False

[05:50] Posso usar isso dentro do meu construtor. Antes de eu atribuir alguma coisa para o meu atributo URL da minha classe, vou fazer essa verificação: If self.urlEhValida(url): self.url = url else: raise LookupError (“Url inválida!!!!!!”)

[06:25] Podemos fazer muita coisa aqui. Podemos printar alguma coisa na tela, podemos dizer ao usuário que ele passou alguma coisa errada. Mas eu prefiro lançar uma exceção, até para aprendermos algo novo, diferente.

[06:40] O raise significa que vou jogar alguma coisa para o meu usuário. Vamos voltar para o código main e tentar usar essa classe.

[07:02] A primeira coisa que temos que fazer é importar ela: from ExtratorArgumentosUrl import ExtratorArgumentosUrl. Quero criar uma instância: url = “https://www.bytebank.com.br/cambio?moedaorigem=real&moedadestino=dolar&valor=700” argumento = ExtratorArgumentosUrl (url) print (argumento)
[07:52] Rodando isso, já tenho o objeto do tipo ExtratorArgumentosUrl alocado em algum lugar da memória do meu computador. E se eu tentar passar outra coisa? url = None

[08:10] Ele procurou o erro e lançou a exceção que nós criamos, url inválida. Mas se eu colocar alguma coisa vazia a mesma coisa acontece.

[08:23] Nossa classe está pronta para trabalhar com URLs que sejam none, que não existam, ou com URLs vazias. Percebam uma coisa interessante no método que acabamos de definir: ele tem o self, mas nós não usamos para nada, certo? Nós usamos ele também no init e precisamos dele para criar nosso atributo. Qual a diferença de uma coisa para a outra?

[08:50] Significa que para usar o init preciso de algum objeto, alguma instância do main, ou de algum outro lugar qualquer. Já o método urlEhValida não precisa dessa instância, porque é um método estático, enquanto o outro é um método de instância.

[09:05] Vou provar para vocês que consigo fazer uma modificação nesse método, chamá-lo diretamente do main. Mas antes, precisamos dizer para o Python que esse método é de fato estático.

Para isso, usamos uma coisa bem bacana chamada decorador. São formas de modificar o comportamento de funções ou classes sem precisar modificar o código.

[09:36] Um decorador é identificado através do @. Se começa com @, é um decorador. O decorador que queremos é o @staticmethod. Eu vou dizer de fato que esse método é estático.

[09:52] Percebe que o PyCharm até me disse que isso aqui não é mais necessário, que estou com o self, mas não tenho ele e não preciso ter ele. Antes, ele estava com outra cor. E minha classe vai continuar funcionando normalmente.

[10:13] Se eu passar o url de novo, vai continuar rodando normalmente. E agora não preciso fazer argumento igual alguma coisa. Posso simplesmente printar e chamar o método diretamente da classe. Funciona perfeitamente. Antes eu não conseguiria, daria erro, porque ele não sabe que é uma classe estática.

[11:10] Até aqui tudo tranquilo, nossa classe está funcionando normalmente. Na próxima aula vamos continuar criando esses métodos que serão responsáveis por extrair os argumentos da url.

[00:00] Vamos voltar à construção dos nossos métodos. Vou criar um método novo: def ExtrairArgumentos(self):

[00:15] Esse método recebe self porque é um método de instância. Eu quero primeiro encontrar os índices que vamos fazer o fatiamento para retirar esses argumentos, depois quero encontrar esses dois argumentos com os índices que nós encontramos.

[01:00] Vamos começar procurando o índice da moeda origem. Dando uma olhada na url que vai receber, eu posso procurar pelo sinal de igual, certo? Se eu der um find no meu url, procurando o igual vou achar o índice da minha moeda origem.

[01:41] O que eu posso procurar para encontrar alguém desse final? O &.

[02:00] Nossa moeda destino está depois de um igual também, mas lembram que o find para na primeira vez que ele encontra o caractere buscado? Então, se eu fizer o find com igual, ele vai parar antes do que precisamos.

[02:21] Lembram que o L do real está na posição 15? O nosso método find recebe mais um argumento além do que ele está buscando. Ele recebe também o ponto de partida, onde ele tem que começar a busca, para não termos o problema de esbarrar no que não queremos. Posso colocar um índice maior do que índice do primeiro igual. Aí, vou ter certeza que meu find vai encontrar o próximo igual. indiceInicialMoedaDestino = self.url.find(“=”, 1) indiceInicialMoedaOrigem = self.url.find(“=”) indiceFinalMoedaOrigem = self.url.find(“&”)

[03:15] Agora, vou criar as variáveis que vão receber: moedaOrigem = self.url[indiceInicialMoedaOrigem:indiceFinalMoedaOrigem] moedaDestino = self.url [indiceInicialMoedaDestino:] return moedaorigem, moedaDestino

[04:10] Agora preciso entrar nessa instância e acessar esse método, que é o ExtraiArgumentos: moedaorigem, moedaDestino = argumentosUrl.extraiArgumentos print (moedaDestino, moedaOrigem)

[04:48] Rodando o main, ele me dá =dólar e =real. Mas está com um sinal de igual. Lembram que preciso colocar o +1 para o sinal de igual não aparecer? Isso porque o fatiamento do índice inicial é inclusivo.

[05:15] Agora, nosso método consegue retornar somente os nomes dos argumentos que estamos buscando. Na próxima aula vamos conhecer o método novo do Python, das strings do Python, para fortalecer mais a nossa classe, deixá-la mais pronta para possíveis problemas.

[00:00] Nossa ideia agora é fortalecer mais nossa classe. Tivemos um problema, que foi encontrar o igual duas vezes. Quero mostrar um pouco melhor como funciona o find, porque a ideia agora é, ao invés de buscar o sinal de igual, buscar o nome do argumento.

[00:22] Antes disso, quero mostrar para vocês como o Python se comporta quando passamos um texto maior, ao invés de um só caractere, porque até agora só passamos o & e o sinal de igual, que são caracteres específicos. Mas e se eu passar “moedaDestino”? Será que ele me retorna o índice no O? No M? Vamos dar uma olhada: index = url.find('”moedaDestino”) print (index)

[01:15] Ele me dá o 17. Se contarmos, é exatamente onde queríamos. Vamos colocar agora a subString. index = url.find('”moedaDestino”) subString = url [index:] print (subString)

[01:50] Isso me gera um problema. Eu preciso começar no índice do D, não no índice do M. Mas perceba que o que tem entre esse índice que encontrei com o find tem a quantidade de caracteres do nome do meu argumento. Depois, é só somar +1. Se eu colocar index = url.find('”moedaDestino”) + 12, ele me dá =dólar. Se eu colocar index = url.find('”moedaDestino”) + 13, ele me dá dólar.

[02:35] Mas e se eu mudar o nome do meu argumento em alguma ocasião? Em algum ponto tenho que mudar esse nome. Será que preciso ficar mexendo na linha de código o tempo todo? Ou será que o Python tem uma função específica que serve para me retornar o tamanho de uma string? Sim, o nome desse método é len.

[02:56] Ele não é um método de string. É uma função do Python index = url.find('”moedaDestino”) +len(“moedaDestino”)

[03:18] Assim, ele chega no =dólar. Se eu colocar +1, ele me dá dólar. Perfeito. Retorno já o valor do argumento que eu quero para dentro do meu código. Agora, o que precisamos fazer é pegar essa lógica e jogar para dentro da nossa classe. Vou criar mais duas variáveis: buscaMoedaOrigem = “moedaorigem” buscaMoedaDestino = “moedaDestino”

[03:56] Vou usar isso para alimentar meu método find e meu método len. Eu preciso pegar a instância, que é o url que estou passando, e preciso usar o método find nela. Perceba que para o moeda origem posso fazer a mesma coisa, só vou mudar a variável. indiceInicialMoedaDestino = self.url.find(buscaMoedaDestino) + len (buscaMoedaDestino) + 1 indiceInicialMoedaOrigem = self.url.find(buscaMoedaOrigem) + len (buscaMoedaOrigem) + 1 indiceFinalMoedaOrigem = self.url.find(“&”)

[05:48] Rodando esse código, ele me dá dólar e real. Funciona normalmente.

[06:09] Agora, o teste é: se eu colocar todo aquele sufixo antes que representa o endereço da minha página, será que vai continuar funcionando normalmente? url = “https://bytebank.com/cambio?moedaorigem=real&moedadestino=dólar”
[06:28] Funciona normalmente. Agora não tenho mais aquele problema de encontrar o igual de um ou de outro. Eu busco direto no nome do argumento.

[06:40] Percebam que eu fiz isso duas vezes, e se eu fiz duas vezes, acho que é uma justificativa para criar um método que vai ser responsável por encontrar esse índice. Eu preciso trazer aquela lógica lá de cima para esse método. def encontraIndiceInicial (self, moedaBuscada): return self.url.find(moedaBuscada) + len (moedaBuscada) + 1

[07:35] Aí, ao invés de passar a lógica inteira, eu chamo minha função: indiceInicialMoedaDestino = self.encontraIndiceInicial (buscaMoedaDestino) indiceInicialMoedaOrigem = self.encontraIndiceInicial (buscaMoedaOrigem)

[08:00] Testando novamente, tudo certo. Retornei minha moeda destino e minha moeda origem.

[08:16] Por enquanto é só. Vejo vocês na próxima aula.

Comece criando um novo arquivo Python com o nome ExtratorArgumentosURL.py dentro do mesmo diretório do arquivo main.py e dentro desse novo arquivo crie uma classe nomeada ExtratorArgumentoURL, bem como um construtor para ela.

```
class ExtratorArgumentoURL:
    def __init__(self,url):
        if self.stringEhValida(url):
            self.url = url
        else:
            raise LookupError(“Url inválida”)
    @staticmethod
    def stringEhValida(url):
        if url:
            return True
        else:
            return False
```

Agora você pode criar o método que retorna os índices dos argumentos e também o método que retorna os argumentos. Lembre-se que é preciso somar + 1 em alguns casos para acessar o índice correto.

```
def retornaMoedas(self):
        buscaMoedaOrigem = "moedaorigem"
          buscaMoedaDestino = "moedadestino"

        inicioSubstringMoedaOrigem = self.encontraIndiceInicioSubstring(buscaMoedaOrigem)
        finalSubstringMoedaOrigem = self.url.find("&")
        moedaOrigem = self.url[inicioSubstringMoedaOrigem:finalSubstringMoedaOrigem]

          inicioSubstringMoedaDestino = self.encontraIndiceInicioSubstring(buscaMoedaDestino)
        finalSubstringMoedaDestino = self.url.find("&valor")
```

```
        moedaDestino = self.url[inicioSubstringMoedaDestino:finalSubstringMoedaDestino]

        return moedaOrigem, moedaDestino

    def encontraIndiceInicioSubstring(self, moedaOuValor):
            return self.url.find(moedaOuValor) + len(moedaOuValor) + 1
```

Até aqui tá tudo tranquilo! Na próxima aula vamos ver mais alguns métodos de string e retornar o argumento que contém o valor da transação de dentro de nossa url.

Nessa aula continuamos aprendendo métodos de string e vimos o seguinte:

•O que é um método estático e um método de instância
•Aprendemos como o if do Python funciona
•Vimos como criar erros e deixar nossa classe mais robusta
•A descobrir o tamanho das strings com a função len()
•O índice retornado pelo find() quando buscamos uma palavra
Na próxima aula começar a criar a classe ExtratorArgumentoURL, espero vocês na próxima aula para continuar a construção dela!

AULA 3

[00:00] Sejam muito bem-vindos a mais essa aula. Espero que estejam aproveitando o conteúdo, fazendo os exercícios e tirando suas dúvidas.

[00:07] Quero propor a vocês agora o seguinte problema. Supondo que a nossa url que chega na classe venha com algum bug do sistema ou algum erro de digitação do usuário e o valor da moeda origem, ao invés de vir real, peso ou dólar, venha com a seguinte string: moeda destino. Vou colocar no código para vocês verem como vai ficar: url = “https://bytebank.com/cambio?moedaorigem=moedadestino&moedadestino=dólar”
[00:40] Será que isso vai dar algum problema na nossa classe? A minha moeda origem já viria com problema com certeza, porque o valor dela está incorreto. Mas tem mais um bug. A minha moeda destino também foi retornada com bug. Por quê?

[00:52] Lembra que o find para na primeira ocorrência que ele encontra? No momento em que ele encontra a ocorrência da moeda destino, ele para e retorna todo o resto, de acordo com o método que definimos.

[01:11] A solução que proponho para esse primeiro problema mais simples é: ao invés de buscar somente o nome do argumento, busquemos o nome do argumento com o sinal de igual. Se rodarmos, passa bem perto de vir certo, ele me dá “olar” ao invés de dólar.

[01:33] Esse problema acontece porque somamos mais um para ele pular o sinal de igual. Como consideramos o sinal de igual agora, não precisamos do + 1. Cortando-o, vem tudo certo.

[01:47] Nós acabamos de resolver metade do problema. Temos que resolver agora o problema do bug mesmo, da entrada incorreta da nossa classe da moeda origem. Para isso, teríamos que ir na nossa url e trocar o valor de “moeda destino” para algum padrão qualquer.

[02:09] Como o ByteBank é BR podemos trocar para real. O Python possui um método específico que busca alguma coisa dentro de uma string e substitui por outro valor qualquer, é o método replace. Vamos ver como ele funciona no main. Depois jogamos essa lógica dentro da classe.

[02:29] Vou criar a string bytebank. Quero que a minha string nova receba um banco novo, e quero procurar por bank e substituir por Rodrigo: stringNova = string.replace(“byte”, “rodrigo”)

[03:02] Foi. Mas se eu tiver mais um byte, o que vai acontecer? Será que o replace substitui tudo? Vamos testar: string = bytebankbytebyte

[03:32] Sim, ele substitui tudo. Ficaria rodrigobankrodrigorodrigo. Na nossa url temos um caso parecido com isso, porque possuímos o moeda destino, o bug que veio, e o nome do argumento moeda destino, que eu não quero perder. Mas, para nossa sorte, o replace do Python possui mais um argumento que ele pode receber: a quantidade de substituições que quero fazer: stringNova = string.replace(“byte”, “rodrigo”, 1)

[04:00] Se eu coloco 1, eu só substituo o primeiro índice que eu encontrar. Se eu colocar 2, ele substitui os dois primeiros que encontrar. O último não é substituído. O que vamos usar na nossa classe vai ser para substituir um. O que temos que fazer é jogar essa lógica para dentro da classe.

[04:15] Vamos abrir nosso arquivo e vamos criar um método novo. O que eu quero fazer é sobrescrever o nosso atributo url por esse atributo novo, que vai ter um valor certo. def trocaMoedaOrigem(self): self.url = self.url.replace (“moedadestino”, “real”, 1)

[05:10] Além disso, preciso encaixar esse método em algum lugar. Preciso chamar esse método no momento em que encontro minha moeda origem. Caso essa moeda origem seja igual a moeda destino, eu chamo esse método. É um if. if moedaOrigem == “moedadestino”: self.trocaMoedaOrigem()

[05:41] Vou tratar primeiro da origem. Depois trato da moeda destino.

[05:51] Ok, eu descobri que a moeda origem é igual moeda destino, o valor dela, e já retornei, já substituí minha url pela url correta. Vamos testar isso: print (self.url)

[06:28] Rodando isso, ele fez a substituição, mas o valor da minha moeda origem ainda não foi trocado. Ainda está como moeda destino, então preciso depois que faço essa troca encontrar a moeda origem novamente.

[06:45] Caso essa moeda origem seja igual moeda destino, fiz a troca, e então chamo tudo aquilo de novo dentro do meu if.

[06:58] Testando novamente, deu certo. Nós substituímos o valor da moeda origem e depois reescrevemos a variável que retornamos para o método main.

[07:20] Estamos deixando nossa classe bem forte, bem robusta. Ela está pronta para receber mais erros vindos do sistema, ou um erro de digitação do usuário. Na próxima aula, vamos continuar melhorando nossa classe e deixando ela mais robusta para esse tipo de coisa. Ela já está retornando moeda origem e moeda destino. Agora, precisamos retornar o valor.

[07:46] Na próxima aula continuamos criando-a então. Obrigado por assistir, e até lá.

[00:00] Até agora, retornamos os argumentos moeda origem e moeda destino. Mas além disso, precisamos também retornar o argumento valor, que nem inserimos na url ainda, mas iremos inserir daqui a pouco.

[00:15] Antes disso, quero propor para vocês um erro novo. A aplicação do ByteBank tem vários bugs, e nossa classe tem que estar pronta para isso. Supondo que algum nome de argumento venha com qualquer caractere maiúsculo, será que vai ter algum problema? Será que o Python sabe a diferença de maiúsculo e minúsculo? Vamos ver.

[00:40] Vou criar duas variáveis: banco1 = “bytebank” e banco2 = “Bytebank”. Vou perguntar para o Python se banco1 é igual a banco2. Ele disse que não. Banco1 não é igual a banco2. O Python sabe a diferença entre letra maiúscula e minúscula.

[01:16] Vamos supor que o r de origem venha maiúsculo na nossa url. Minha moeda destino continua normal, mas minha moeda origem não veio. Então, com certeza, um erro desse tipo gera um problema grande dentro da nossa publicação.

[01:45] O Python possui duas funções bem legais, dois métodos string. O upper e o lower. Um deixa todos os caracteres de uma string em maiúscula e o outro deixa todos em minúscula. Vou fazer o seguinte: banco2 = “Bytebank”.upper () e vou printar. Ele deixou todos os caracteres em maiúsculo. O contrário também é valido se eu colocar lower.

[02:25] Esses dois métodos podem nos ajudar muito a não ter esse tipo de problema na nossa classe. O que temos que fazer é pegar essa lógica e jogar para dentro dela. Onde seria interessante fazer isso e qual dos dois eu faço? Isso é só uma definição. Eu quero que minha função trabalhe só com letras minúsculas ou só com maiúsculas? Acredito que seja uma boa usar somente letras minúsculas. Então, no momento em que recebo esse atributo, já faço a transformação. buscaMoedaOrigem = “moedaorigem”.lower () buscaMoedaDestino = “moedadestino”.lower ()

[03:15] Assim, não tem risco nenhum da minha url ter alguma coisa maiúscula. Dólar e real foram retornados perfeitamente.

[03:38] O que podemos fazer agora é já encontrar o valor, aproveitando que estamos mexendo na classe. Eu vou terminar essa url, vou colocar o último argumento dela: url = “https://bytebank.com/cambio?moedaorigem=moedadestino&moedadestino=dólar&valor=1500”
[03:50] Preciso fazer uma coisa bem parecida com ExtraiArgumentos. Eu vou criar um ExtraiValor. Novamente vou criar uma variável que vai me ajudar a buscar alguma coisa. Eu vou usar isso para me ajudar a localizar o índice inicial. E preciso retornar o índice dele mais o comprimento. Temos um método já bem interessante para isso, que é o encontraIndiceInicial. def extraiValor (self): buscaValor = “valor=” indiceInicialValor = self.encontraIndiceInicial (buscaValor) valor = self.url [indiceInicialValor:] return valor

[05:11] Rodando isso, ele me deu dólar imprimindo todo o resto. Eu quero só o dólar na moeda destino. Esse problema apareceu por quê? Eu estou deixando o último índice do fatiamento da moeda destino vazio. Eu encontro o inicial, mas não encontro o final, porque antes a minha url acabava ali, agora não mais.

[05:47] Eu preciso encontrar o índice final da moeda destino e passá-lo para o fatiamento. Mas qual pode ser esse índice? Posso encontrar o &valor, passar esse índice. Testando, o bug fica corrigido.

[06:20] O que eu preciso agora é criar uma variável nova valor = argumentosUrl.extraiValor (). Printando essa variável, estou retornando meu dólar, meu real e meu valor. Nossa classe está completa, está suportando um bug do valor da moeda origem vir incorreto, o bug de vir algum caractere no nome dos argumentos em maiúsculo, e está retornando tudo o que eu quero.

[07:15] Na próxima aula, quero propor que caso a url não seja do ByteBank, será que minha classe está pronta? Como fazemos para identificar se a url pertence ao Byte Bank ou não? Vejo vocês lá.

[00:00] Até agora, estamos construindo nossa classe sem nos preocuparmos se a url que está sendo enviada para dentro dela é de fato do ByteBank. Mas como definimos se a url vai ser do ByteBank ou não?

[00:15] A minha proposta é: caso a url comece com a string https://bytebank.com/, ela é do ByteBank. Mas será que existe alguma outra forma da url conter essa string específica sem pertencer ao ByteBank? Vou mostrar que sim.
[00:33] Vou fazer uma busca rápida no Google por esse argumento. Lembra que sempre que faço uma busca no Google, ele tem algum argumento ali dentro que vai ficar com o nome do que a gente buscou? Pode ser que isso aconteça, e pode ser que por algum motivo alguma url de algum site de buscas ou alguma outra url maliciosa de um site de piratas ou hackers estejam tentando invadir nosso sistema usando esse tipo de artimanha, de colocar nosso endereço dentro do endereço deles achando que nossa classe não estará pronta para isso.

[01:13] Nós vamos agora preparar a classe para esse tipo de problema. Que método conhecemos que identifica se algum texto está dentro de alguma coisa? O find. A primeira coisa que vamos fazer é começar a usar o método find para buscar se alguma url contém a do ByteBank ou não, e em qual posição ela se encontra, porque isso é muito importante. urlByteBank = “https://bytebank.com” url1 = “https://buscasites.com/busca?q= https://bytebank.com” url2 = “https://bitebank.com.br” url3 = “https://bytebank.com/cambio/teste/teste”
[02:17] Esse é um exemplo parecido com o do Google, mas em escala menor, para deixar a url mais identificável. A url2 tem um nome bem parecido com o do ByteBank, mas não é.

[02:50] Vou começar a utilizar o método find para verificar se nossa url buscada está dentro dessas URLs que acabamos de criar: print (url1.find(urlByteBank)). Ele encontrou na posição 31.

[03:31] Essa url contém a url do ByteBank mesmo sem ser do ByteBank. Nós não queremos que nossa classe aceite esse tipo de coisa. Precisamos que seja do primeiro índice para confirmar que essa url é do ByteBank. Vamos testar de novo com a url2. Ele me retorna -1, o que significa que nem achou. Na url3, ele me dá 0.

[04:08] Se o retorno do método find for 0, posso considerar que essa url de fato é confiável, é do nosso sistema. Mas existe outra forma de fazer esse tipo de busca. Existe um método específico de string que me ajuda bastante com esse tipo de coisa. É o método starts with. Eu verifico se a minha string começa com um caractere específico. Exemplo: print (url3.startswith (urlByteBank)).

[04:52] Ele acusa como verdadeiro. Perfeito. Posso usar ou o find retornando 0 ou posso usar direto esse método starts with, que já me retorna true ou false de cara.

[05:11] Vamos testar com a url1. Lembrando que ela contém de fato a url do ByteBank, mas me retornou false, porque não começa com a url. Novamente, nosso trabalho sempre é colocar essa lógica dentro da classe.

[05:29] Esse método de urlEhValida já parece bem justo para isso. Ele verifica se a url existe. Pode verificar também se além de existir é do ByteBank. Eu posso dizer: if url and url.startswith ("https://bytebank.com"): return true
[06:12] Testando isso, ele funcionou normalmente. Mas e se eu colocar ao invés de Byte, Bite? Ele me mostra um erro e me diz que a url é inválida.

[06:35] Agora nossa classe está pronta para lidar com URLs maliciosas, que não sejam da nossa aplicação e estejam tentando utilizar nossa classe para fazer qualquer coisa. Terminamos nossa classe, ela está pronta, funcional.

[06:54] O que vamos ver na próxima aula é algo um pouco diferente. Nós vamos continuar trabalhando com strings, mas de outra forma. Vamos ver expressões regulares, que são formas de encontrar padrões bem definidos dentro de textos mais humanos, dentro de e-mails, posts de Facebook ou de mensagens de celular. Mas sem spoiler. Façam todas as atividades e tirem as dúvidas que tiverem. Obrigado por assistir e até a próxima aula.

Bora continuar criando nossa classe!

Vamos continuar deixando nossa classe mais forte com o método lower(), caso algum usuário passe para nosso sistema a url com os argumentos em caixa alta nossa classe não dá conta, pois ele busca os argumentos em caixa baixa. Uma solução para isso é a seguinte:

```
def __init__(self, url):
        if self.stringEhValida(url):
            self.url = url.lower()
        else:
            raise LookupError("url inválida!")
```

Para deixar a classe ainda mais preparada para possíveis erros do usuário, vamos pensar no caso em que o argumento de moeda origem é passado da seguinte forma: “moedaorigem=moedadestino”, isso faz com que nossa aplicação retorne valores incorretos. Crie um método responsável por verificar esse caso.

```
def verificaMoedaOrigem(self, buscaMoedaOrigem):
        self.url = self.url.replace("moedadestino", "real", 1)  # primeiro vou fazer sem o 1 e depois vou usa-lo
        inicioSubstringMoedaOrigem = self.encontraIndiceInicioSubstring(buscaMoedaOrigem)
    finalSubstringMoedaOrigem = self.url.find("&")
        return self.url[inicioSubstringMoedaOrigem:finalSubstringMoedaOrigem]
```

Dentro do método retornaMoedas é preciso colocar um condicional, logo abaixo de onde encontramos a moedaOrigem:

```
if moedaOrigem == "moedadestino":
            moedaOrigem = self.verificaMoedaOrigem(buscaMoedaOrigem)
```

Por fim bora criar um método responsável por retornar o valor que deve ser convertido:

```
def retornaValor(self):
    buscaValor = "Valor".lower()
    inicioSubstringValor = self.encontraIndiceInicioSubstring(buscaValor)
    valor = self.url[inicioSubstringValor:]
    return valor
```

Tudo pronto, agora vamos abrir outro arquivo .py no mesmo diretório do arquivo da nossa classe e importa-la.

```
from ExtratorArgumentosURL import ExtratorArgumentoURL
url =" https://www.bytebank.com.br/cambio?moedaorigem=real&moedadestino=dolar&valor=150"
cambio = ExtratorArgumentoURL(url)
moedaOrigem,moedaDestino=cambio.retornaMoedas()
valor = cambio.retornaValor()
print(moedaOrigem,moedaDestino,valor)
Aqui ta tudo bem, teste agora o caso em que "moedaorigem=moedadestino".
url =" https://www.bytebank.com.br/cambio?
moedaorigem=moedadestino&moedadestino=dolar&valor=150"
cambio = ExtratorArgumentoURL(url)
moedaOrigem,moedaDestino=cambio.retornaMoedas()
valor = cambio.retornaValor()
print(moedaOrigem,moedaDestino,valor)
```

Os métodos upper() e lower();

•O método replace()
•O método in
•Os métodos startswith, endswith

Estamos analisando uma URL que é uma string padronizada, mas e se o objetivo for encontrar algo em um texto menos padronizado? É possível extrair um número de telefone que foi enviado dentro do corpo de um e-mail ? Com o que sabemos é possível sim, mas existe uma forma muito mais simples chamada Regex e que vamos aprender na próxima aula!

AULA 4

[00:00] Fala, pessoal! Sejam bem-vindos à mais essa aula do nosso curso.

[00:02] Como eu dei o spoiler na semana passada, vamos começar a falar sobre expressões regulares. Vou começar criando algumas variáveis para vermos em que situação uma expressão regular pode ser bem aplicada.

[00:16] O nome da minha variável vai ser: email1 = "Meu numero é 1234-1234"

[00:26] Isso aqui representa o que seria um e-mail para informar um número de celular. email2 = “Fale comigo em 1234-1234 esse é meu telefone” email3 = “1234-1234 é o meu celular”

[00:52] Perceba que aqui a gente não tem um lugar específico para podermos usar o método find. Não temos um nome de argumento que precede o meu número de celular. Ele pode estar no final do texto, no meio do texto ou no começo do texto. Mas, meu número de celular possui uma estrutura bem padronizada. Ele tem cinco dígitos antes e quatro dígitos depois. No caso, usei quatro dígitos para ser um número de telefone, um telefone fixo. Não vai ser celular.

[01:27] Como podemos representar esse padrão com uma forma muito bem definida? Usando a variável “padrão”, que concorda que o número pode variar de 0 a 9. Então: padrao = “[0123456789]”.

[01:47] Isso pode representar o que vem a ser um número dentro do meu número de telefone. Mas eu preciso de quatro. Então, vou dar ctrl+c ctrl+v, ficando então: padrao = “[0123456789] [0123456789] [0123456789] [0123456789] [-] [0123456789] [0123456789] [0123456789] [0123456789]”.

[02:20] Agora sim. Tenho meu padrão bem representado dentro do Python.

[02:26] Agora, o que precisamos fazer é importar a biblioteca que é responsável por trabalhar com expressões regulares. Lembra que aqui a gente tem o extrator de argumentos do URL? A gente criou a classe e depois importamos dentro do main, com from, extratorArgumentosUrl import e a nossa classe.

[02:46] Aqui, nós podemos simplesmente dar o import. “re” é a biblioteca que já vem junto com o Python que é responsável por lidar com expressões regulares. Já montamos o padrão. Nós temos o e-mail. O que precisamos fazer agora é acessar o método específico dentro dessa biblioteca que é responsável por fazer essa busca.

[03:14] Então, vou criar uma variável com o nome “retorno”. O método “search” recebe alguns argumentos. Ele recebe o padrão que eu estou procurando e recebe o texto em que quero buscar isso. Vou buscar dentro do email1 para começar. retorno = re.search(padrao,email1)

[03:38] E vou printar isso: print (retorno)

[03:51] Ele já encontrou alguma coisa. Só que isso não significa nada para a gente. Está aqui o nosso match. O método que o retorno possui é o “group”. print (retorno.group())

[04:04] Agora sim. Usando esse “.group” eu consigo retornar só o que eu quero, só os resultados que foram encontrados. E encontrei dentro do email1, mas vamos tentar agora com o email2. retorno = re.search(padrao,email2)

[04:18] Continua funcionando normalmente, sendo no meio do texto. Vamos tentar com o email3. retorno = re.search(padrao,email3)

[04:25] Funciona também. Agora, vou criar o email4, só para vermos se está funcionando. email4 = “lalalalala 93543-1254 hsiahsahrueuhdiaeu”

[04:43] Aqui, vou substituir por email4: retorno = re.search(padrao,email4)

[04:47] Vamos rodar novamente. Perceba que eu estou dentro de uma string totalmente aleatória, usei um valor diferente, 1234, 1234, e continuou funcionando. Mas é porque esse padrão representa

qualquer dígito, qualquer valor de 0 a 9. E se eu encontrar quatro dígitos seguidos, seguidos de um hífen, e seguidos de mais quatro dígitos, a minha regex vai identificar com o search e vai retornar para mim. E só para printar de maneira bonitinha eu coloquei o ".group". Está aqui como essa biblioteca é poderosa e como ela possui métodos interessante para nós.

[05:29] Na aula que vem, o que vamos fazer é reduzir um pouco esse padrão. Então, vejo vocês lá.

[00:00] Agora já vimos a força que as expressões regulares têm, sabemos melhor o que elas são e sabemos como usá-las dentro do Python. Um problema que estamos enfrentando agora é que esse padrão está enorme, ocupando nossa tela toda. Queremos saber se existe uma forma de reduzir isso.

[00:22] Vamos olhar a documentação do Python e procurar onde encontramos expressões regulares ali. Vou abrir o Google e simplesmente digitar Python doc. Já é logo o primeiro link. Nele temos as versões do Python, e aí vemos as documentações dessas versões específicas. A última é 3.7. A 3.8 está em desenvolvimento.

[00:56] A documentação é dividida em algumas partes. A primeira é "o que tem de novo no Python?", temos um tutorial, e depois temos referências. Vamos abrir o library reference.

[01:25] Ele descreve bem a sintaxe e a semântica da linguagem Python. Built-in é tudo que vem junto com a instalação do Python. Quando você instala, se alguma coisa vem junto com a instalação do Python, é built-in. Por exemplo, lembram que fizemos o import re? Eu não precisei instalar nada a mais do que o Python, então posso dizer que essa é uma biblioteca built-in do Python.

[01:58] Será que essa biblioteca re está aqui dentro? Está. Faz sentido. Expressões regulares ficam dentro de textos, portanto aqui tenho tudo sobre expressões regulares.

[02:16] Se você quiser ler toda essa documentação, você com certeza vai saber tudo sobre expressões regulares do Python. Vou procurar aqui caracteres especiais, os colchetes, porque estamos usando eles.

[02:38] Ele diz que os caracteres dentro dos colchetes podem ser listados individualmente, que é exatamente o que fizemos. O nosso padrão está pronto para encontrar qualquer elemento dentro dos colchetes.

[03:08] Já se parece um pouco com o que queremos. Se dermos dois caracteres separados com um hífen, ele vai encontrar todos os itens entre aqueles caracteres. Então, se eu colocar [0-9], vou encontrar todos as coisas que estão entre esses dois números.

[03:45] Vamos abrir nosso código e testar isso. Vou começar mudando o primeiro: [0-9]. Caso não dê certo, nós nem tentamos nos outros. Mas deu. Então, vou apagar todo o resto e substituir por esse range. Testando, ele continua funcionando.

[04:29] Vimos a documentação e encontramos uma forma de melhorar nosso código. Mas será que não existe alguma forma de eu dizer que estou procurando a mesma coisa quatro vezes? Se fosse um inteiro, eu poderia colocar [0-9]4 [0-9]4. Não funcionou. Talvez esse asterisco nem seja um caractere especial da minha biblioteca re.
[05:06] Vamos olhar na documentação se tem alguma coisa. Ele me diz que o {m} específica o número de cópias da expressão regular. Por exemplo, a{6} vai ser seis caracteres "a". Ou seja, minha expressão regular vai buscar seis caracteres especificamente.

[05:38] Parece que isso resolve nosso problema. Vamos testar: [0-9]{4}[0-9]{4}. Funcionou. Olhando a documentação, dando uma lida, você acha.

[06:10] Aprendemos agora a olhar a documentação do Python. Sempre que você tiver uma dúvida, jogue no Google. Se você achar rápido, ok. Caso não ache sua dúvida resolvida de forma rápida, o lugar certo para procurar e onde com certeza terá tudo que você precisa, é na documentação.

[06:34] Na aula que vem vamos continuar melhorando a expressão regular e deixando ela mais pronta para pegar um celular que tem cinco dígitos no começo, ao invés de quatro.

[00:00] Mantendo o padrão de na dúvida olhar a documentação do Python, caso eu queira encontrar algum número que seja de celular ou de telefone, quero dizer que antes do hífen, posso ter quatro ou cinco dígitos. Vamos ver se tem alguma coisa que pode nos ajudar com isso.

[00:20] Vamos procurar algo parecido com o que queremos. {m, n}, ele vai encontrar de m a n da expressão regular que antecede ele. Ou seja, se eu colocar 4,5 dentro do nosso código, vou poder encontrar ou um, ou outro. Vamos testar: [0-9]{4, 5}. Funcionou, mesmo com cinco números.

[01:18] Essa sintaxe da expressão regular, que são as chaves, posso usar para buscar intervalos. São expressões regulares com quantificadores e com intervalos. Agora, outra proposição para vocês. Supondo que eu tenha três celulares, e esteja mandando um e-mail para o meu colega de trabalho com todos os três celulares. Esse método não suporta esse tipo de coisa.

[01:50] Se eu testar, ele me retorna somente a primeira ocorrência do padrão que ele encontrou. Novamente, vamos dar uma olhada na documentação e ver se tem algum método sem ser o search que pode nos ajudar.

[03:15] O findall parece promissor. Encontre todos. Ele vai retornar todas as vezes que encontrar o padrão. O próprio nome já explicou o que ele faz. Vamos tentar. Ao invés de search, vamos colocar findall. Vou tirar o group porque quero algo bem específico.

[03:49] Ele retornou uma lista com todas as ocorrências desse padrão encontrado dentro do texto. Vou colocar mais um com quatro dígitos antes do hífen. Ele funciona também. E me retorna na ordem em que encontrou. Vamos tentar com mais um. Continua funcionando. O problema é que eu tinha escrito 45678, mas ele me retornou 4567. Ou seja, os caracteres que ele encontrou antes desse 8. Parece que não faz muito sentido isso.

[04:43] Eu posso colocar então de {4, 5}, ele vai me retornar o 8 daí. Mas, no caso do Brasil, números de celular tem cinco números antes do hífen e quatro depois. Então não quero isso acontecendo mesmo.

[04:58] O que quero agora é que meu padrão esteja pronto para receber o hífen ou não. Olhando a documentação novamente, vamos pesquisar.

[05:39] Parece que encontramos algo que funciona para nós, o asterisco. Vamos testar: [0-9] {4, 5} [-] * [0-9] {4}. E vou também tirar o hífen de cima. Ele continua me retornando, mesmo assim. Colocando o asterisco, ele funciona normalmente. Se eu tirar o hífen de todos, ele vai continuar funcionando.

[06:00] Agora, nossa expressão regular consegue pegar número de celular ou de telefone e consegue identificar tendo o hífen ou não. Vocês já sabem bastante sobre expressões regulares, vocês já

aprenderam muitos métodos de string que já deram muitas ferramentas para a caixa de ferramentas de desenvolvedores de vocês.

[06:41] Na próxima aula, vamos aprender o que são métodos especiais no Python. Com isso, vamos aprender bastante sobre a arquitetura, o funcionamento da linguagem Python. Vejo vocês lá.

Vimos nessa aula que nem todos os textos são tão simples de serem retirados de dentro dos textos utilizando o fatiamento de strings, mas além disso vimos também que com expressões regulares é possível retirar padrões de dentro de qualquer texto.

Vamos identificar números de telefone dentro de um texto:

```
padrao = "[0123456789][0123456789][0123456789][0123456789][-][0123456789][0123456789]
[0123456789][0123456789]"
# Vamos testar esse padrão.
texto =  "Meu número para contato é 2633-5723"
retorno = re.search(padrao,texto)
print(retorno.group())
```

É possível simplificar a representação do padrão utilizando intervalos de caracteres `[0-9]`

```
padrao = "[0-9][0-9][0-9][0-9][-][0-9][0-9][0-9][0-9]"
```

Podemos simplificar ainda mais utilizando quantificadores para os intervalos.

```
padrao = "[0-9]{4}[-][0-9]{4}"
```

Para deixar nosso padrão mais simples podemos retirar `[-]` e deixar somente `-`, pois essa é a única opção para essa posição.

```
padrao = "[0-9]{4}-[0-9]{4}"
```

Agora caso o número de telefone seja de um celular e possua um 9 na frente do número? Podemos dar opções para os quantificadores.

```
padrao = "[0-9]{4,5}-[0-9]{4}"
```

Mas e se quisermos que o `-` seja opcional dentro da expressão regular? Uma forma de fazer isso é utilizando o operador `?`

```
padrao = "[0-9]{4,5}-?[0-9]{4}"
```

Aprendemos a criar expressões regulares para capturar padrões de string dentro de qualquer texto, utilizando os seguintes comandos:

•Grupos de caracteres: [0123456789];
•Intervalos de caracteres: [0-9];
•Quantificadores: {4}, {4,5} e ?;
•E os métodos re.search, re.findall e group()
Na próxima aula iremos discutir sobre métodos especiais e continuar melhorando nossa classe.

AULA 5

[00:00] Sejam bem-vindos a mais essa aula do nosso curso de Python. Nessa aula, quero mostrar o que são métodos especiais. São coisas muito interessantes e que podem dar funcionalidades muito bacanas para as nossas classes.

[00:16] Antes disso, quero fazer uma afirmação. Quero dizer que tudo em Python é uma instância de alguma classe maior. Por exemplo, no momento em que você cria string = “bytebank”, você está criando uma variável do tipo str, que acessa essa classe str. Como consequência, pode usar seus métodos, que foi o que fizemos ao longo do curso. Usamos o método find, lower, replace. Todos dessa classe str.

[00:50] Se você está usando o PyCharm, existe uma forma bem simples de entrar nessa classe. Eu digito str, pressiono ctrl e clico no str. Perceba que já fui jogado para um arquivo totalmente diferente. É um arquivo de built-in.py.

[01:05] Nós já comentamos que tudo que é built-in vem junto com a instalação do Python, como o método re, que usamos na última aula. Aqui tem uma descrição rápida do que é essa classe. E depois temos os métodos.

[01:25] Vamos dar uma olhada no capitalize. Ele diz que faz o primeiro caracteres ser maiúsculo e todo o resto ser minúsculo. É como se fosse um título. Vamos procurar agora o find. Ele diz que o find retorna o valor mais baixo onde uma subString é encontrada. Foi exatamente o que fizemos. Podemos ver até aquele argumento que usamos para conseguir usar o find para chamar a moeda destino, e outro argumento sobre até onde a busca pode ir.

[02:25] Encontramos também o lower aqui. Outro método que nós usamos. Vamos procurar pelos métodos especiais da classe str. Vou dizer que o método especial começa com dois underlines e termina com dois underlines.

[02:39] Esses métodos especiais dão funções para as nossas classes. Vou destacar o método len: print (len(string)). Ele me diz que bytebank possui oito caracteres. Agora, quero criar uma variável do tipo inteiro: x = 200. Vou dar um print nisso: print(len(X)). Será que ele vai dizer que é igual a três, por ter três caracteres? Não funcionou. Ele me diz que objetos do tipo inteiro não possuem a função len.

[03:43] Vamos dar uma olhada nessa classe int e quais métodos ela possui. Quero procurar aqui dentro o método len. Ele não tem. Repare que na classe str existe o método especial len, eu usei função len e o string e funcionou, retornou o tamanho. Já a classe int não tem o método especial len. Eu usei e deu erro.

[04:35] A boa notícia é que posso criar um método especial len dentro da nossa classe, da classe que estamos criando, para dar alguma funcionalidade no momento em que tentamos extrair o comprimento da nossa classe.

[04:52] Vamos voltar para o main. Eu tenho já o argumentosUrl, que é uma instância da nossa classe. Vou tentar printar o tamanho dela: print (len(argumentosUrl)). Rodando isso, ele me fala que objeto do tipo ExtratorArgumentosUrl não possuem len. O mesmo erro que tivemos no inteiro.

[05:18] Vou abrir nossa classe e vou ter que retornar alguma coisa para o Python saber o que eu considero como tamanho da minha classe. def__len__(self): return 10

[05:41] Se eu rodar agora, ele printa 10. Alguma coisa deu erro, porque eu não tinha o tipo len na minha classe. Agora não dá mais erro. Ele retorna alguma coisa. Mas faz sentido o tamanho da minha classe ser 10? O que isso significa? Não tem 10 aqui. Minha url não tem 10 caracteres somente. Tem muito mais.

[06:04] Eu quero que o tamanho da minha classe seja o tamanho do atributo url. Se tivesse algum outro atributo eu poderia decidir entre qual eu quero usar para representar o tamanho, mas como só tem, uso simplesmente: def__len__(self): return len(self.url)

[06:25] Agora ele retornou 74. Dentro da url tem 74 caracteres. Se eu apagar, ele me mostra 73. Está funcionando perfeitamente. Estamos utilizando uma função do Python para fazer alguma coisa com a nossa classe, e tudo isso graças ao método especial.

[06:49] Atentem-se a um detalhe. Olhando para esse init e para esse erro que nós retornamos, eu vejo que o LookupError talvez não seja o mais adequado para identificar o que está acontecendo no momento em que nossa url é inválida.

[07:02] LookupError é um erro de pesquisa. Estamos tendo um erro de passagem de argumento.

[07:09] Quero mostrar uma coisa para vocês na documentação do Python. Vamos em built-in Exceptions. Todas as exceções que o Python possui. Vamos procurar a que estamos usando, LookupError. Esse erro serve para quando estou passando um índice inválido. Não é o que está acontecendo com a gente. Estamos tendo um erro de passagem de argumento.

[07:56] Para essa aula não ficar muito comprida, vou deixar o Lookup, mas quero que vocês saibam que existem muitos erros que estão dentro da biblioteca do Python, da documentação do Python, e que você pode encontrar e aplicar o que for melhor para sua classe, para o seu tipo de erro.

[08:12] Na próxima aula vamos ver uma forma de representar melhor nossa classe. Quando tento dar um print (argumentosUrl), ele me mostra algo bem esquisito. Na próxima aula vamos resolver isso.

[00:00] E se no lugar de printar o len da minha classe, o tamanho que a classe tem, eu quiser printar a minha classe só? Vou dar um print na instância argumentosUrl que foi criada lá em cima.

[00:14] Ele me dá vários caracteres. É alguma coisa específica do meu sistema operacional, mas que para o usuário não faz o menor sentido ver. Para nós que estamos criando já não faz sentido.

[00:35] A boa notícia é que da mesma que tinha um método especial que fazia o len funcionar de forma melhor com a nossa classe, existe também um método especial que faz o print funcionar. Esse método cria uma representação string da nossa classe. O nome dele e é str. Aqui é muito parecido, eu preciso dar um return de alguma coisa. def__str__(self): return “Minha classe”

[01:09] Ele retornou alguma coisa diferente. Retornou “Minha classe”. Não faz muito sentido, porque é muito genérico. Eu quero algo mais específico. Tenho que decidir o que quero que seja mostrado quando printo minha classe. Pode ser a url inteira. def__str__(self): return self.url

[01:30] O que ele me retornou eu podia ter feito a qualquer momento com o get. Acho que não quero criar um método especial str só para isso. Vou começar a criar uma variável que vai ser a representacaoString = self.extraiValor(). Vou retornar essa variável. def__str__(self): representacaoString = self.extraiValor() return representacaoString

[02:01] Rodando o main mais uma vez, ele me retorna 1500. Deu o que eu queria. Mas só isso faz sentido para nós? Acho que não. Vamos concatenar isso. def__str__(self): representacaoString = self.extraiValor() return representacaoString + self.extraiArgumentos()

[02:27] Eu estou tentando passar uma tuple dentro desse meu método especial str, mas não posso. Tenho que passar somente strings. Então vou tentar popular duas variáveis. def__str__(self):

moedaOrigem, moedaDestino = self.extraiArgumentos () representacaoString = self.extraiValor() + “ “ + moedaOrigem + “ “ + moedaDestino return representacaoString + self.extraiArgumentos()

[02:59] Já funcionou. Eu peguei 1500 e o real. Eu consegui agora representar de forma bem mais clara para o meu usuário o que tem dentro dessa instância. Mas posso trazer ainda maior. Posso fazer de uma forma que eu tenha textos indicando o que significam esses nomes. Por exemplo: def__str__(self): moedaOrigem, moedaDestino = self.extraiArgumentos () representacaoString = “Valor:” + self.extraiValor() + “ “ + moedaOrigem + “ “ + moedaDestino return representacaoString + self.extraiArgumentos()

[03:48] Existe outra função do Python que me ajuda com isso. É a format. Dentro do format, eu vou indicar o que vai estar dentro de cada uma das chaves. def__str__(self): moedaOrigem, moedaDestino = self.extraiArgumentos () representacaoString2 = “Valor:” + self.extraiValor() + “ “ + moedaOrigem + “ “ + moedaDestino representacaoString = “Valor: {}\n Moeda Origem: {}\n Moeda Destino: {}\n”.format(self.extraiValor(), moedaOrigem, moedaDestino) return representacaoString + self.extraiArgumentos()

[05:03] Agora temos valor igual a 1500, moeda origem igual a real e moeda destino igual a dólar. Simplesmente por printar a minha instância da classe recebo de forma bem mais estruturada tudo que tem ali dentro. Certo que isso não serve para eu fazer uma conversão de alguma coisa, porque isso é string, mas para uma representação do que tem ali dentro, está ótimo.

[05:31] Por enquanto é isso. Na próxima aula vamos conseguir começar a comparar essas instâncias. Um exemplo: se eu criar um argumentosUrl2 e printar, perguntando se o argumentosUrl é igual ao argumentosUrl2, sendo que não mudei nada, ele vai me dizer que é falso. Na próxima aula vamos ver como resolver esse problema.

[00:00] Nosso objetivo agora é conseguir comparar as instâncias da nossa classe. Lembram que no último vídeo eu tentei simplesmente dar um print perguntando se duas instâncias da mesma classe com a mesma entrada eram iguais e o Python disse que era falso?

[00:20] Eu organizei um pouco o código e comentei algumas coisas para não ter vários prints inúteis. Coloquei nossa string embaixo e instanciei duas vezes com a mesma url. Temos a mesma resposta, falso.

[00:32] Antes de irmos para a parte de métodos especiais, quero mostrar que por default o Python compara nossas classes por um id. Nós podemos acessar uma função id: print.id(argumentoUrl1). Também vou printar uma id de argumentoUrl2.

[00:52] Eles são parecidos, mas são diferentes. É isso que o Python tenta comparar quando não dizemos para ele o que ele precisa comparar de fato. O que temos que fazer é colocar o método especial certo dentro da nossa classe para conseguir fazer a comparação.

[01:12] Nós já temos o init, o len e o str. Vamos criar agora o def__eq. Esse eq é diferente dos outros dois. Ele não recebe só o self. Ele precisa receber outra instância, que vai ser usada para comparação. Eu preciso também dar um return e novamente decidir qual atributo quero que seja comparado neste momento. Por nossa sorte, nossa classe só tem um atributo, posso usar isso como padrão. def__eq__ (self,outraInstancia): return self.url ==outraInstancia.url

[02:02] Caso esses dois atributos sejam iguais nas duas instâncias, o valor retornado será true. Independente agora do id que essas instâncias possuem, o que está sendo comparado será a url.

[02:20] Se eu quiser que a transformação cambial de uma das duas seja diferente, vou criar duas strings. Url1 e Url2. Uma vai ser 1500 e a outra 500. Passo uma string para cada instância. Ele me diz que é falso.

[02:43] Quando eu tiver minhas URLs diferentes, vai ser falso, porque eu disse ao Python quem ele tem que olhar para comparar. É assim que métodos especiais funcionam.

[02:52] Especificamente do método eq tem uma postagem bem interessante no blog da Alura. Eu vou deixar o link na parte escrita do texto.

[03:04] Essa aula foi bem rápida. E é isso. Até logo, galera.

[00:00] Nosso curso chegou ao final. Quero agradecer muito por terem assistido esse curso comigo e espero que tenham gostado, que tenham feito todos os exercícios e que tenham tirado as dúvidas.

[00:13] Vamos fazer uma revisão rápida de tudo que aprendemos. Primeiro, começamos construindo nossa classe, extraindo argumentos, usando só o método find, só para pegar argumentos de dentro da nossa url.

[00:30] Depois disso, aprendemos o método len, que tivemos que criar um método diferente dentro da nossa classe para conseguir usar esse len de forma mais limpa, para não ter uma lógica muito grande dentro de outra classe, que ficaria muito difícil de fazer a manutenção depois.

[00:50] Depois de aprender o método len, aprendemos o método lower, para deixar nossa classe mais forte e pronta para qualquer tipo de problema que possa vir do sistema. A nossa classe vem de uma forma, com maiúsculas, minúsculas, misturado, não importa. Ela vai cair com o atributo em tudo minúsculo para nossa instância. Usamos o lower também para não ter risco de procurar uma coisa que não exista na nossa url.

[01:22] Depois disso, vimos o método replace, para caso tenha algum outro tipo de bug da nossa classe, em que o usuário digitasse errado a moeda origem. Deixamos nossa classe bem robusta.

[01:38] Depois de aprender alguns métodos de string, aprendemos regex, que descobrimos ser formas de retirar padrões de texto bem definidos, como um número de celular, e-mail, ou qualquer coisa desse tipo de dentro de um texto humano, de um post de Facebook, um comentário no Instagram, uma mensagem no WhatsApp, qualquer coisa parecida.

[02:02] Nós olhamos muito a documentação e vimos que a documentação do Python é onde está todo o material que podemos saber da linguagem.

[02:16] Depois, voltamos para a nossa classe e começamos a construir métodos especiais. Vocês viram que métodos especiais são formas de deixarmos nossa classe mais forte e fazer com que ela tenha funções mais adequadas para ela e para coisas que você quer retornar usando funções do Python. Vimos que quando criamos uma variável do tipo int ou str ou float, que não usamos, estamos acessando uma classe built-in do Python, que é a classe responsável por dar esses métodos aos nossos tipos de variáveis, ao string, por exemplo. Vimos que a classe str tem o método replace, lower, len.

[03:03] Vimos que a classe inteiro não tem o método especial len, e por isso quando tentamos usar a função len no tipo inteiro, ele não funciona. Ele retorna para nós que o tipo inteiro não possui len. E vimos que esse tipo de funcionalidade pode ser aplicada dentro das nossas classes.

[03:23] Depois, fizemos o principal. Demos uma representação string para a nossa classe, e a nossa classe começou a não dar vários números aleatórios quando printamos. Ela passou a ser uma coisa visível, uma coisa que faz sentido para quem está vendo.

[03:48] Vimos o método format também, que é um método de uma classe string. Com ele consigo fazer de forma mais bonita essa representação. E por fim, criamos outro método especial, o eq, que posso comparar instâncias diferentes da mesma classe.

[04:14] Lembram que antes de ter isso, ao invés de comparar pela url, o Python comparava pelo id? Porque ele não sabia por onde comparar e os ids eram diferentes uns dos outros.

[04:29] Busquem mais conteúdo de Python, é uma linguagem muito bacana e que está crescendo muito. Vai ser muito importante para a carreira de vocês. Espero que tudo que vimos aqui seja aplicável e que vocês coloquem na caixa de ferramentas de desenvolvedores de vocês.

[04:43] Muito obrigado por assistir e até logo.

O método format() serve para criar strings com chaves que serão substituídas por argumentos dentro do método format(). Veja o exemplo abaixo:

```
texto = “{} é um anime excelente e já possui {} temporada”
print(texto.format(“The Rising of shield Hero”,1))
```

Os campos substituídos estão sempre associados a parâmetros que podem ser nomeados ou não, vamos repetir o exemplo mas dessa vez nomeando os argumentos dentro das chaves e dentro do método format().

```
texto = "{anime} é um anime excelente e já possui {temporada} temporada"
print(texto.format(temporada=1,anime="The Rising of shield Hero"))
```

É possível utilizar o format para alinhar os textos e para determinar uma quantidade máxima de caracteres que podem ser exibidos. Veja a lista abaixo com alguns exemplos e seus resultados para a seguinte string: texto = “Eu gosto de pizza de {}”.format(“Calabresa”).

{0:>20} - Alinha a string à direita com 20 espaços em branco

Resposta : Eu gosto de pizza de Calabresa

{0:#>20} - Alinha a string à direita com 20 símbolos #

Resposta : Eu gosto de pizza de ##########Calabresa

{0:#^20} - A linha a string no centro de 20 símbolos #.

Resposta : Eu gosto de pizza de #####Calabresa######

{0:.5} - Imprime somente os 5 primeiros dígitos

Resposta : Eu gosto de pizza de Calab

Podemos encontrar mais sobre o assunto format(), sendo usado para outros tipos de variáveis na seguinte página: https://cadernoscicomp.com.br/tutorial/introducao-a-programacao-em-python-3/funcoes-print-input-e-o-metodo-format/

Agora vamos construir os métodos especiais que construímos em nossa classe:

O método str() diz ao Python como deve representar nossas classes e a forma de defini-lo é a seguinte:

```
def __str__(self):
    moedaOrigem,moedaDestino = self.extraiArgumentos()
    representacaoString2 = "Valor:" + self.extraiValor() + " " + moedaOrigem + " " + moedaDestino
    representacaoString  = "Valor: {}\n Moeda Origem: {} \n Moeda Destino: {} \
n".format(self.extraiValor(),moedaOrigem,moedaDestino)
    return representacaoString
```

O método len() diz ao Python para qual atributo de nossa classe ele deve olhar para determinar o tamanho de uma instância de nossas classes.

```
def __len__(self):
    return len(self.url)
```

O método eq() determina como o Python deve comprar 2 objetos de uma mesma classe.

```
def __eq__(self,outraInstancia):
    return self.url == outraInstancia.url
```

Nessa aula aprendemos sobre um pouco do funcionamento do Python estudando os seus métodos especiais. E implementamos os seguintes métodos:

•O método especial len();
•O método especial str();
•O método especial eq();
•O método format();