

PYTHON DEVELOPMENT

ESTRUTURAS LÓGICAS CONDICIONAIS E REPETIÇÕES



LISTA DE FIGURAS

Figura 1 – Resultados da busca por if hadouken	13
Figura 2 – Resultado do programa de teste de conectivos	16
Figura 3 – Use o botão New Project para criar um projeto.....	19



LISTA DE QUADROS

Quadro 1 – Operadores aritméticos em Python	7
Quadro 2 – Conectivos lógicos em Python	15

EMENDAS

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Sintaxe do if simples em Python	8
Código-fonte 2 – Programa do restaurante por quilo	9
Código-fonte 3 – Programa forma de pagamento com if simples.....	10
Código-fonte 4 – Sintaxe do if composto em Python.....	10
Código-fonte 5 – Programa do desconto.....	11
Código-fonte 6 – Sintaxe do if encadeado em Python	12
Código-fonte 7 – Programa da companhia aérea.....	13
Código-fonte 8 – Sintaxe do if encadeado com uso do elif em Python	14
Código-fonte 9 – Programa da companhia aérea com uso do elif.....	15
Código-fonte 10 – Programa para testar os conectivos	16
Código-fonte 11 – Programa de login utilizando o conectivo and.....	17
Código-fonte 12 – Sintaxe do loop while em Python	18
Código-fonte 13 – Exemplo de loop while	18
Código-fonte 14 – Controlando o loop while com uma variável contadora.....	18
Código-fonte 15 – Programa da senha secreta (1)	19
Código-fonte 16 – Programa da senha secreta (2)	20
Código-fonte 17 – Programa da senha secreta (3)	20
Código-fonte 18 – Programa da senha secreta (4)	21
Código-fonte 19 – Estrutura de menus com o loop while	21
Código-fonte 20 – Trecho do código responsável pelo cálculo do fatorial	22
Código-fonte 21 – Programa contendo todas as estruturas do capítulo.....	23

SUMÁRIO

ESTRUTURAS LÓGICAS, CONDICIONAIS E REPETIÇÕES.....	6
1 A LÓGICA QUE O COMPUTADOR ENTENDE	6
1.1 Pensando como um computador.....	6
1.2 Os desvios condicionais	7
1.3 Os conectivos lógicos AND, OR e NOT	15
1.4 Voltas e mais voltas	17
REFERÊNCIAS.....	24

EMANIP

ESTRUTURAS LÓGICAS, CONDICIONAIS E REPETIÇÕES

1 A LÓGICA QUE O COMPUTADOR ENTENDE

1.1 Pensando como um computador

Com certeza agora você já sabe domar algumas das principais capacidades da linguagem de programação Python, não é mesmo? Entre criar variáveis, receber dados do usuário, realizar operações e exibir, já estamos criando uma série de algoritmos. Mas ainda há uma diferença fundamental na forma como pensamos e na forma como o computador “pensa”.

Falando em pensamento, vamos imaginar essa situação hipotética: ao entrar em um cômodo e notar que existem paredes com umidade aparente, você decide fazer uma reforma e aciona profissionais capazes de tocarem a obra. Parece um exemplo simples, mas na origem deste algoritmo está a pergunta “Existem manchas na parede?” e as possibilidades de resposta são apenas “sim” ou “não”.

É justamente neste tipo de pergunta, onde temos respostas dicotômicas, que reside a forma como um computador verifica uma situação, pois ele é capaz de realizar testes booleanos e validar se o resultado desses testes é “verdadeiro” ou “falso”.

Entender o mecanismo básico por trás da lógica aplicada pelo computador, onde os resultados de testes devem ser sempre 0 ou 1, é importantíssimo pois nos auxiliará na construção de algoritmos mais complexos. É esta lógica que nos permitirá criar um sistema de login (“O usuário e senha informados pelo usuário estão corretos?”), um sistema bancário (“Há saldo suficiente na conta para realizar a compra?”) e até mesmo um jogo (“Existe um inimigo na tela na área onde o laser atingiu?”).

Daqui para frente, então, criaremos testes de forma que os resultados possíveis sejam apenas VERDADEIRO ou FALSO, como perguntar “A parede é azul?” em vez de perguntar “Qual é a cor da parede?”.

Todo esse papo sobre “perguntas” deve ter lhe deixado com uma enorme dúvida: como criamos esses testes em linguagem Python? Aperte os cintos, porque é hora de conhecermos os operadores lógicos!

1.2 Os desvios condicionais

A forma de construirmos as nossas “perguntas”, que chamaremos de “testes lógicos” ou “condições” daqui para frente, depende exclusivamente dos operadores que possam ser utilizados na comparação. E, para a nossa sorte, nós já aprendemos esses operadores no capítulo anterior, quando estudamos o tipo de dados bool. Vamos recordar:

Operador	O que retorna?
>	Verdadeiro quando o primeiro valor é maior que o segundo, Falso nos outros casos
<	Verdadeiro quando o primeiro valor é menor que o segundo, Falso nos outros casos
>=	Verdadeiro quando o primeiro valor é maior ou igual que o segundo, Falso nos outros casos
<=	Verdadeiro quando o primeiro valor é menor ou igual que o segundo, Falso nos outros casos
==	Verdadeiro quando o primeiro valor é igual ao segundo, Falso quando são diferentes
!=	Verdadeiro quando o primeiro valor é diferente do segundo, Falso quando são iguais
bool(valor)	Verdadeiro quando o valor é diferente de 0 em variáveis numéricas e diferente de vazio em variáveis de texto, Falso nos outros casos

Quadro 1 – Operadores aritméticos em Python
Fonte: Elaborado pelo autor (2022)

Também já fizemos alguns testes com esses operadores no capítulo anterior, e obtivemos resultados como True (verdadeiro) e False (falso) quando criamos as nossas primeiras condições. Mas, está faltando algo, não é mesmo? Afinal, onde é que essas condições serão úteis?

Existem estruturas de programação que funcionam com base em condições e os desvios condicionais são das mais importantes estruturas deste tipo. Os desvios condicionais, como o nome sugere, são estruturas que permitem ao programa desviar da sua execução normal com base no resultado de um teste lógico.

Imagine a seguinte situação: todos os clientes de um restaurante que cobra por quilo devem receber o valor dos seus pratos com base no peso fornecido e no valor do kg cobrado pelo estabelecimento. Porém, clientes cujos pratos que ultrapassem 1kg podem pagar apenas o valor fixo de R\$80,00 e consumirem à vontade.

Perceba que temos um caminho “normal” para este algoritmo (todos os clientes recebem o valor de seus pratos com base no peso), mas há uma possibilidade de desvio (clientes cujos pratos ultrapassem 1kg podem pagar o valor fixo). Para implementar esse desvio, utilizaremos uma estrutura chamada if dentro do Python, que possui a seguinte estrutura básica:

```
if condição:  
    instruções para realizar caso a condição seja verdadeira
```

Código-fonte 1 – Sintaxe do if simples em Python

Fonte: Elaborado pelo autor (2022)

A estrutura if em Python difere ligeiramente daquela presente em algumas linguagens, portanto é importante observarmos:

- Não há necessidade de envolver as condições entre parênteses em casos nos quais não desejamos priorizar algumas das condições.
- O sinal de dois pontos (:) marca o início das instruções que serão executadas dentro de determinado bloco do desvio condicional.

O Python é capaz de saber quais instruções estão “fora” ou “dentro” do if com base na indentação. É por isso que utilizamos 4 espaços em branco ou 1 tabulação para indicar que uma instrução está dentro do if.

Voltando ao nosso problema do restaurante por quilo, poderíamos propor a seguinte solução:


```
print("Quilão do Andrezão")

preco_quilo = float(input("Informe o valor cobrado por quilo!
"))
peso_prato = float(input("Informe o peso do prato do cliente
(em kg) "))

preco_prato = preco_quilo * peso_prato

print(f"O valor do prato é R${preco_prato:.2f}")

#O desvio abaixo testa se o peso do prato ultrapassa 1kg e
exibe a mensagem apenas
#caso o resultado do teste seja verdadeiro
if peso_prato >= 1:
    print("O prato excedeu 1kg. O cliente pode pagar R$80,00
e consumir à vontade")
```

Código-fonte 2 – Programa do restaurante por quilo
Fonte: Elaborado pelo autor (2022)

O script anterior segue exatamente o raciocínio necessário para resolver o problema: todos os clientes recebem a mesma mensagem, mas há um desvio caso o peso do prato ultrapasse o estabelecido. Academicamente chamamos este tipo de desvio de desvio condicional simples, aquele que só apresenta uma alternativa de execução para o caso de o resultado do teste ser verdadeiro.

Apesar do desvio condicional simples ser excelente e ter se encaixado perfeitamente na solução que propusemos, existem cenários nos quais precisamos realizar uma ação para o caso de o teste ter resultado verdadeiro e outra ação para o caso de o teste ser falso. Veja este cenário:

Uma loja oferece pagamentos por boleto bancário ou por cartão de crédito. Os clientes que pagam por meio de boleto têm direito a 5% de desconto sobre o valor da compra, enquanto os clientes que pagam por intermédio de cartão de crédito podem escolher parcelar a compra em até 12x.

Neste novo caso, existem duas possibilidades de pagamento para os nossos clientes e, como já estudamos anteriormente, precisamos criar as nossas perguntas de forma que o resultado seja apenas verdadeiro ou falso. Podemos optar por criar dois ifs simples, um com a pergunta “A forma de pagamento é boleto?” e outro com a pergunta “A forma de pagamento é cartão?”. Se seguirmos por essa abordagem (que não é adequada, como veremos logo adiante), teríamos o seguinte script:

Estruturas Lógicas Condicionais e Repetições

```
print("Saldão da alegria!")

total_compra = float(input("Por favor, informe o valor total da compra do cliente "))

forma_pagamento = int(input("Selecione a forma de pagamento: 1 - Boleto ou 2 - Cartão "))

if forma_pagamento == 1:
    total_compra_desconto = total_compra * 0.95
    print(f"O total da compra de R${total_compra:.2f} sofreu um desconto pelo pagamento em boleto. O cliente deverá pagar R${total_compra_desconto:.2f}.")

if forma_pagamento == 2:
    parcelas = int(input("Informe a quantidade de parcelas desejadas (entre 1 e 12)"))

    valor_parcela = total_compra / parcelas
    print(f"O total da compra de R${total_compra:.2f} será pago em {parcelas} parcelas de R${valor_parcela:.2f}.")
```

Código-fonte 3 – Programa forma de pagamento com if simples

Fonte: Elaborado pelo autor (2022)

Apesar de o script anterior solucionar o problema proposto e não possuir nenhum demérito aos olhos do usuário, ele contém um grave erro de conceito: ao utilizar dois ifs simples, estamos fazendo duas perguntas ao processador enquanto apenas uma pergunta bastaria (afinal, se a forma de pagamento selecionada não for boleto, só pode ser cartão). Essa sucessão de perguntas desnecessárias pode se traduzir em perda de desempenho ou em consumo excessivo de recursos.

Portanto, precisamos ir além do if simples e utilizar o desvio condicional composto, que também apresenta um fluxo alternativo para o caso de o teste lógico ser falso. Isto é feito através da cláusula else (que significa “senão”, em inglês), que inaugura o bloco de instruções que será executado caso o teste não seja verdadeiro. A sintaxe deste if é:

```
if condição:
    instruções caso a condição seja verdadeira
else:
    instruções caso a condição seja falsa
```

Código-fonte 4 – Sintaxe do if composto em Python

Fonte: Elaborado pelo autor (2022)

Estruturas Lógicas Condicionais e Repetições

Utilizando o if composto, nosso script anterior poderia ser reescrito da seguinte maneira:

```
print("Saldão da alegria!")

total_compra = float(input("Por favor, informe o valor total da compra do cliente "))

forma_pagamento = int(input("Selecione a forma de pagamento: 1 - Boleto ou 2 - Cartão "))

if forma_pagamento == 1:
    total_compra_desconto = total_compra * 0.95
    print(f"O total da compra de R${total_compra:.2f} sofreu um desconto pelo pagamento em boleto. O cliente deverá pagar R${total_compra_desconto:.2f}.")
else:
    parcelas = int(input("Informe a quantidade de parcelas desejadas (entre 1 e 12)"))

    valor_parcela = total_compra / parcelas
    print(f"O total da compra de R${total_compra:.2f} será pago em {parcelas} parcelas de R${valor_parcela:.2f}.")
```

Código-fonte 5 – Programa do desconto

Fonte: Elaborado pelo autor (2022)

Nosso programa agora faz apenas uma pergunta ao processador ("forma_pagamento == 1"), concede o desconto caso o teste tenha resultado verdadeiro e solicita o número de parcelas para o caso de o teste ter resultado falso (o que pode ter desdobramentos que precisemos considerar caso a loja passe a contar com outras formas de pagamento no futuro).

Você poderá se deparar, ainda, com situações em que existe mais de uma condição envolvida, mas em que uma condição depende do resultado das anteriores para ser analisada. O caso a seguir ilustra bem este cenário:

Uma companhia aérea permite que seus clientes do tipo premium despachem bagagens de até 32 kg sem nenhum custo adicional, enquanto os clientes do tipo gold podem levar malas de até 28kg sem nenhum custo adicional e todos os demais devem pagar por qualquer bagagem despachada.

Estruturas Lógicas Condicionais e Repetições

Com base no tipo do cliente e no peso informado da bagagem, podemos informar se ela está ou não nos limites estabelecidos, mas há uma ordem lógica para fazermos essa verificação: após concluirmos que um cliente não é premium, podemos verificar se é gold e, apenas se não pertencer a nenhum dos tipos, podemos concluir que é um cliente normal.

Para implementarmos esta lógica, precisamos de uma estrutura condicional que nos permita encadear os nossos desvios e esse é o caso do desvio condicional encadeado: temos uma estrutura condicional dentro de outra, de forma que a segunda condição só seja avaliada com base no resultado da primeira. Veja abaixo:

```
if condição_1:
    instruções caso a condição_1 seja verdadeira
else:
    if condição_2:
        instruções caso a condição_1 seja falsa e a
        condição_2 seja verdadeira
    else:
        instruções caso a condição_1 seja falsa e a
        condição_2 seja falsa
```

Código-fonte 6 – Sintaxe do if encadeado em Python
Fonte: Elaborado pelo autor (2022)

Aplicando esta mesma estrutura ao problema da companhia aérea, podemos chegar ao seguinte código:

```
tipo_cliente = input("Por favor, informe o tipo do cliente:
PREMIUM, GOLD ou REGULAR ")
peso_bagagem = float(input("Informe o peso da bagagem que o
cliente deseja despachar "))

if tipo_cliente.upper() == "PREMIUM":
    if peso_bagagem <= 32:
        print(f"Cliente {tipo_cliente}, sua bagagem está
dentro do limite permitido! Não é necessário pagar nenhum
valor para despachá-la")
    else:
        excedente = peso_bagagem - 32
        print(f"Os clientes {tipo_cliente} têm direito a
despacharem bagagens de até 32kg. A bagagem atual excede este
peso em {excedente}kg. Dirija-se ao balcão de cobrança para
realizar o pagamento referente ao peso adicional.")
else:
    if tipo_cliente.upper() == "GOLD":
        if peso_bagagem <= 28:
```

Estruturas Lógicas Condicionais e Repetições

```
print(f"Cliente {tipo_cliente}, sua bagagem está dentro do limite permitido! Não é necessário pagar nenhum valor para despachá-la")
else:
    excedente = peso_bagagem - 28
    print(f"Os clientes {tipo_cliente} têm direito a despacharem bagagens de até 28kg. A bagagem atual excede este peso em {excedente}kg. Dirija-se ao balcão de cobrança para realizar o pagamento referente ao peso adicional.")
else:
    print(f"Os clientes {tipo_cliente} não tem direito à bagagem gratuita. Favor dirigir-se ao balcão de cobranças para realizar o pagamento pela bagagem.")
```

Código-fonte 7 – Programa da companhia aérea
Fonte: Elaborado pelo autor (2022)

Apesar de o desvio condicional encadeado ser extremamente útil, pode nos gerar uma situação desagradável que recebeu um nome curioso por parte da comunidade de desenvolvedores: o if hadouken, onde o aninhamento de mais e mais desvios condicionais faz com que seu código acabe se deslocando muito para a direita.

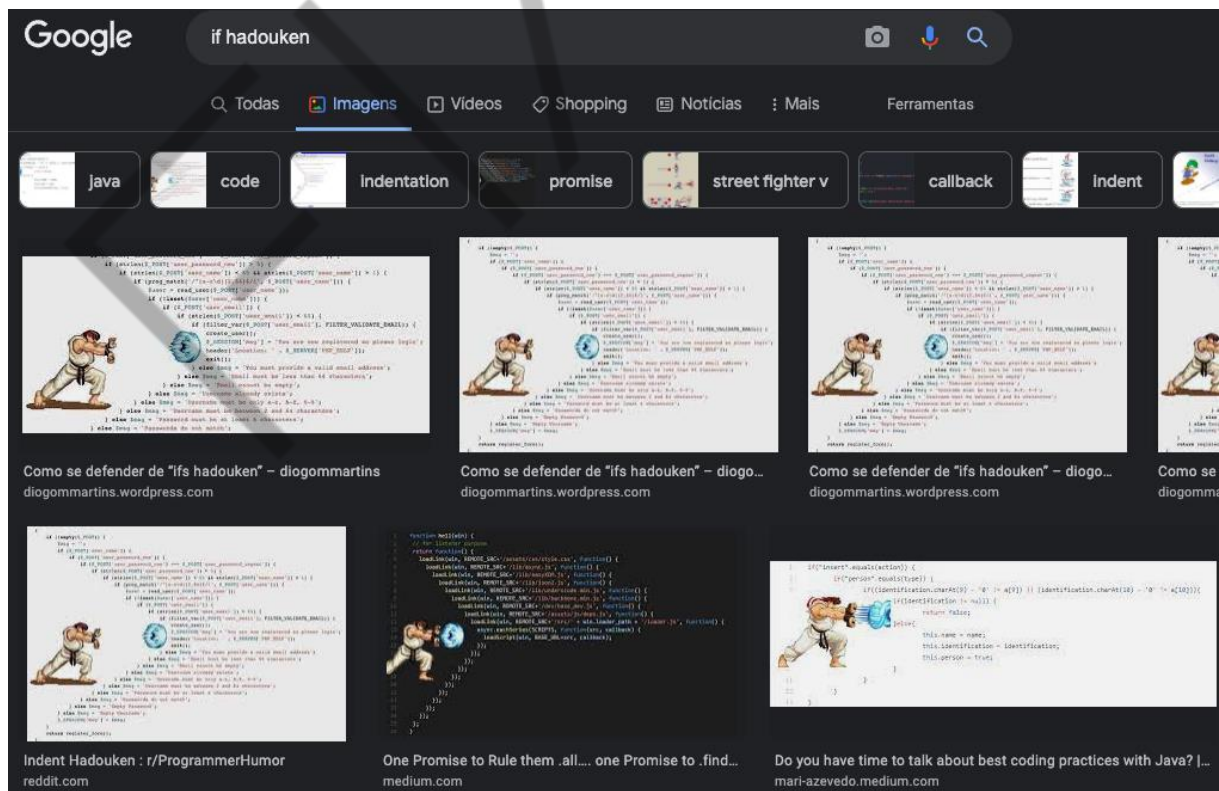


Figura 1 – Resultados da busca por if hadouken
Fonte: Elaborado pelo autor (2022)

Estruturas Lógicas Condicionais e Repetições

Uma ótima forma de se prevenir deste poderoso “golpe”, é utilizar uma forma alternativa da cláusula else dentro da estrutura: o elif. O elif é uma cláusula que desempenha a função de senão-se, ou seja, ela age como um bloco else que é disparado quando a condição do if anterior é falsa, mas também age como um novo if. A estrutura é assim:

```
if condição_1:
    instruções caso a condição_1 seja verdadeira
elif condição_2:
    instruções caso a condição_1 seja falsa e a condição_2
    seja verdadeira
else:
    instruções caso a condição_1 seja falsa e a condição_2
    seja falsa
```

Código-fonte 8 – Sintaxe do if encadeado com uso do elif em Python
Fonte: Elaborado pelo autor (2022)

Se aplicarmos essa mesma estrutura no nosso código anterior, referente ao problema da companhia aérea, vamos chegar ao seguinte script:

```
tipo_cliente = input("Por favor, informe o tipo do
cliente: PREMIUM, GOLD ou REGULAR ")
peso_bagagem = float(input("Informe o peso da bagagem que
o cliente deseja despachar "))

if tipo_cliente.upper() == "PREMIUM":
    if peso_bagagem <= 32:
        print(f"Cliente {tipo_cliente}, sua bagagem está
dentro do limite permitido! Não é necessário pagar nenhum valor
para despachá-la")
    else:
        excedente = peso_bagagem - 32
        print(f"Os clientes {tipo_cliente} têm direito a
despacharem bagagens de até 32kg. A bagagem atual excede este
peso em {excedente}kg. Dirija-se ao balcão de cobrança para
realizar o pagamento referente ao peso adicional.")
elif tipo_cliente.upper() == "GOLD":
    if peso_bagagem <= 28:
        print(f"Cliente {tipo_cliente}, sua bagagem está
dentro do limite permitido! Não é necessário pagar nenhum valor
para despachá-la")
    else:
        excedente = peso_bagagem - 28
        print(f"Os clientes {tipo_cliente} têm direito a
despacharem bagagens de até 28kg. A bagagem atual excede este
peso em {excedente}kg. Dirija-se ao balcão de cobrança para
realizar o pagamento referente ao peso adicional.")
```

```
else:
    print(f"Os clientes {tipo_cliente} não tem direito à  
bagagem gratuita. Favor dirigir-se ao balcão de cobranças para  
realizar o pagamento pela bagagem.")
```

Código-fonte 9 – Programa da companhia aérea com uso do elif
Fonte: Elaborado pelo autor (2022)

Existem algumas situações em que precisaremos dar mais poder às nossas condições, avaliando mais de uma delas para só então o desvio condicional poder seguir seu fluxo normal. Para isso, existem os conectivos lógicos.

1.3 Os conectivos lógicos AND, OR e NOT

Aquele produto que está nos seus sonhos toda noite anda extremamente caro em todas as lojas e você depende de duas condições para adquirir: o produto tem que estar em promoção e é preciso ter dinheiro suficiente para pagar pelo valor mais baixo. Sem perceber você já sabe utilizar os conectivos lógicos!

Os conectivos lógicos são utilizados para avaliar duas condições ou modificar o estado lógico de uma delas, e nós podemos aplicar esse recurso toda vez que precisamos “juntar” condições para os nossos desvios condicionais. Em Python existem três conectivos importantes: AND, OR e NOT. O quadro abaixo mostra como esses conectivos funcionam:

Conectivo	O que retorna?
and	Verdadeiro apenas quando ambas as condições são verdadeiras e falso em todos os outros casos
or	Verdadeiro quando pelo menos uma das condições for verdadeira e falso quando ambas as condições são falsas
not	Verdadeiro quando a condição é falsa e falso quando a condição é verdadeira

Quadro 2 – Conectivos lógicos em Python
Fonte: Elaborado pelo autor (2022)

Estruturas Lógicas Condicionais e Repetições

Vamos escrever primeiro um pequeno script que conterá as variáveis país = “Brasil”, cidade = “São Paulo” e curso= “Python”, para podermos testar o retorno dos nossos conectivos:

```
país = "Brasil"
cidade = "São Paulo"
curso = "Python"

teste = país == "Brasil" and cidade == "São Paulo"
print(f"Ao comparar se o país é Brasil e a cidade é São Paulo com o operador AND obtivemos: {teste}")
teste = país == "Brasil" and cidade == "Rio de Janeiro"
print(f"Ao comparar se o país é Brasil e a cidade é Rio de Janeiro com o operador AND obtivemos: {teste}")

teste = país == "Brasil" or curso == "Python"
print(f"Ao comparar se o país é Brasil e o curso é Python com o operador OR obtivemos: {teste}")
teste = país == "Brasil" or curso == "Java"
print(f"Ao comparar se o país é Brasil e o curso é Java com o operador OR obtivemos: {teste}")

teste = not país == "Brasil"
print(f"Ao aplicar o NOT na comparação país == Brasil obtivemos: {teste}")
teste = not país == "Inglaterra"
print(f"Ao aplicar o NOT na comparação país == Inglaterra obtivemos: {teste}")
```

Código-fonte 10 – Programa para testar os conectivos
Fonte: Elaborado pelo autor (2022)

Ao executarmos este programa devemos obter a seguinte saída:

```
Ao comparar se o país é Brasil e a cidade é São Paulo com o operador AND obtivemos: True
Ao comparar se o país é Brasil e a cidade é Rio de Janeiro com o operador AND obtivemos: False
Ao comparar se o país é Brasil e o curso é Python com o operador OR obtivemos: True
Ao comparar se o país é Brasil e o curso é Java com o operador OR obtivemos: True
Ao aplicar o NOT na comparação país == Brasil obtivemos: False
Ao aplicar o NOT na comparação país == Inglaterra obtivemos: True
```

Figura 2 – Resultado do programa de teste de conectivos
Fonte: Elaborado pelo autor (2022)

Agora que sabemos o que cada um dos conectivos faz, criar um script que valide o usuário “admin” e a senha “123” para o usuário:


```
usuario = input("Informe o usuário que deseja acessar o sistema: ")
senha = input("Informe a senha do usuário que deseja acessar o sistema: ")

if usuario.upper() == "ADMIN" and senha == "123":
    print("Acesso autorizado! ")
else:
    print("Usuário ou senha incorretos. Acesso negado!")
```

Código-fonte 11 – Programa de login utilizando o conectivo and
Fonte: Elaborado pelo autor (2022)

É importante notar que se tivéssemos utilizado o conectivo OR no exemplo anterior, acabaríamos dando acesso ao sistema aos usuários que acertassem apenas o usuário ou a senha.

Deu para notar que os desvios condicionais são poderosíssimos, e que utilizá-los nos permite criar programas cada vez mais complexos. Mas ainda temos um conjunto de estruturas muito útil para conhecermos: os loops!

1.4 Voltas e mais voltas

À medida que vamos aprendendo a programar, aprendemos também a enxergar problemas com um olhar diferente. Passamos a pensar em condições, em variáveis e em cada um dos passos necessários para chegar ao resultado. E, a partir de agora, pensaremos também em repetições.

Imagine que um professor do 1º ano do ensino fundamental vá distribuir ovos de Páscoa para cada um dos seus alunos que, pacientemente, aguardam sentados. Para cada um dos alunos da sala, o professor repetirá exatamente a mesma tarefa: pegar um dos ovos da cesta e entregar ao aluno, avançando para o próximo.

Para tarefas como esta, que exigem a repetição de um mesmo comando, as linguagens de programação contam com estruturas chamadas de laços de repetição ou loops. Essas estruturas são responsáveis por repetir um determinado trecho do código com base nas regras definidas pelo desenvolvedor.

Essas “regras” que o programador vai definir podem ser condicionais ou de intervalo. Vamos primeiro pensar em condicionais e, para isso, o laço de repetição que usaremos é o laço enquanto ou while.

O laço de repetição while é baseado em condição, e ele repete um determinado trecho do programa enquanto a condição continuar sendo verdadeira. Sua sintaxe é:

```
while condição:  
    instruções a serem repetidas enquanto a condição  
    continuar sendo verdadeira
```

Código-fonte 12 – Sintaxe do loop while em Python
Fonte: Elaborado pelo autor (2022)

Para entendermos o funcionamento (e os cuidados que devemos ter) do loop while, vamos criar o script abaixo. Mas fica um alerta: ao executar o script fique preparado para interromper a sua execução (utilizando o atalho CTRL + C no terminal ou apertando o botão de STOP no PyCharm), pois ele rodará infinitamente:

```
while 1 == 1:  
    print("Exibindo a mensagem mais uma vez...")
```

Código-fonte 13 – Exemplo de loop while
Fonte: Elaborado pelo autor (2022)

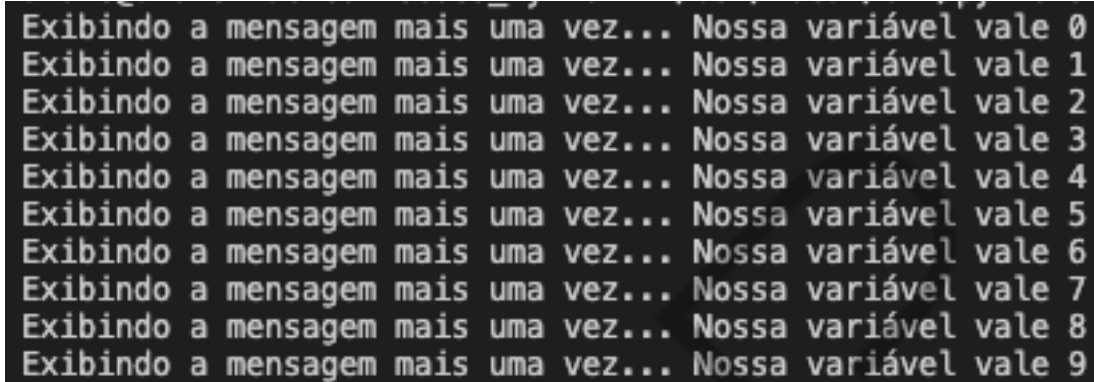
Ao executar este script, sua tela deve ter se enchido da mesma mensagem sendo exibida diversas vezes e, se não parássemos a execução, o script rodaria infinitamente. Isso acontece porque o loop while avalia a condição antes de iniciar uma nova volta e, no nosso caso, criamos uma condição que sempre é verdadeira.

Se quisermos condicionar a execução do loop while a um determinado número de voltas, podemos usar um pouco de lógica de programação: vamos criar uma variável antes do loop e, em cada volta, vamos alterar o valor desta variável. Dessa forma, conseguimos criar uma condição que deixe de ser verdadeira em algum momento. Veja:

```
variavel = 0  
  
while variavel < 10:  
    print(f"Exibindo a mensagem mais uma vez... Nossa  
variável vale {variavel}")  
    variavel = variavel + 1
```

Código-fonte 14 – Controlando o loop while com uma variável contadora
Fonte: Elaborado pelo autor (2022)

Ao executarmos este script, notaremos que a nossa mensagem é exibida 10 vezes, pois na primeira volta do loop ela vale 0 e, ao final dele, passa a valer 1. Dessa forma, na segunda volta a variável valerá 1 e, ao final, 2. Esse fluxo de repetições irá se repetir enquanto a condição continuar sendo verdadeira:



```
Exibindo a mensagem mais uma vez... Nossa variável vale 0
Exibindo a mensagem mais uma vez... Nossa variável vale 1
Exibindo a mensagem mais uma vez... Nossa variável vale 2
Exibindo a mensagem mais uma vez... Nossa variável vale 3
Exibindo a mensagem mais uma vez... Nossa variável vale 4
Exibindo a mensagem mais uma vez... Nossa variável vale 5
Exibindo a mensagem mais uma vez... Nossa variável vale 6
Exibindo a mensagem mais uma vez... Nossa variável vale 7
Exibindo a mensagem mais uma vez... Nossa variável vale 8
Exibindo a mensagem mais uma vez... Nossa variável vale 9
```

Figura 3 – Use o botão New Project para criar um projeto
Fonte: Elaborado pelo autor (2022)

Por mais que possamos utilizar uma variável como contadora com o loop while, a estrutura é perfeita para fazer o que faz de melhor: executar repetições enquanto uma condição é verdadeira. Um desenvolvedor Python pode utilizar esse recurso para criar um loop que não se encerre enquanto uma conexão ainda está ativa, enquanto determinado dado não for recebido, enquanto um arquivo que está sendo lido ainda não terminou... as possibilidades são infinitas!

Uma dessas possibilidades é criar uma pequena brincadeira, em que o usuário ficará preso dentro do nosso programa até que acerte a senha secreta “PYTHON”:

```
resposta = ""

while resposta != "PYTHON":
    resposta = input("Digite a senha secreta: ")

print("Você acertou! Parabéns!")
```

Código-fonte 15 – Programa da senha secreta (1)
Fonte: Elaborado pelo autor (2022)

Se por um lado o loop while é perfeito para situações onde nossa repetição deve basear-se em uma condição, por outro existirão cenários em que será necessário realizar repetições com base em um intervalo.

Lembrando do nosso exemplo do professor que distribuirá ovos de Páscoa, se o raciocínio for “entregar ovos de páscoa enquanto houverem alunos sem terem recebido”, o loop while será adequado. Mas se o raciocínio for: “entregar ovos de Páscoa para cada uma das cadeiras da sala”, não estamos mais pensando em condicionais, mas sim em um intervalo de valores (as cadeiras).

Para situações como essa, existe o loop para que se baseia em contadores ou intervalos. No caso do Python, a sintaxe do loop for é:

```
for contador in intervalo:  
    instruções a serem recebidas
```

Código-fonte 16 – Programa da senha secreta (2)

Fonte: Elaborado pelo autor (2022)

O intervalo deverá ser uma estrutura iterável (estrutura de dados onde é possível percorrer cada um dos itens) e a cada volta do loop for a variável contadora assumirá o valor de um dos itens da estrutura. Como ainda não estudamos as estruturas de dados em Python, vamos criar um loop for que percorra um intervalo que será gerado pela função range:

```
for valor in range(10, 100, 2):  
    print(f"Nesta volta o conteúdo da variável valor é  
{valor}")
```

Código-fonte 17 – Programa da senha secreta (3)

Fonte: Elaborado pelo autor (2022)

Ao executar esse script você vai perceber que a variável valor assume os valores 10, 12, 14, ..., 94, 96, 98. Isso ocorre por que a função range gerou valores em um intervalo entre 10 e 100 (onde o 100 não está incluso e, portanto, o limite superior é 99) com um passo (o intervalo de um valor para o outro) 2.

Se quiséssemos passar por todos os números inteiros entre 1 e 1000, nosso range seria range (1, 1001, 1), pois a sintaxe da estrutura é range (valor_inicial, valor_final -1, passo).

Antes de encerrarmos este capítulo e nos declararmos reis e rainhas das estruturas lógicas e de repetição, vamos conhecer uma última cláusula que pode ser utilizada para interromper a execução de um loop: o break.

Execute o script abaixo:

Estruturas Lógicas Condicionais e Repetições

```
for valor in range(10, 100, 2):  
    print(f"Nesta volta o conteúdo da variável valor é  
{valor}")  
    if valor == 20:  
        break
```

Código-fonte 18 – Programa da senha secreta (4)
Fonte: Elaborado pelo autor (2022)

Você vai reparar que para este script, quase idêntico ao anterior, o último número exibido é o 20. Isso ocorre porque ao passar pela linha “break” o Python entende que deve encerrar o loop onde o comando está contido.

Que tal encerrarmos este capítulo juntando tudo o que aprendemos? Vamos criar uma estrutura de menus usando o loop while e o if encadeado para dar ao usuário 3 escolhas: receber um elogio, calcular o fatorial de um número ou encerrar a execução do programa.

Para criarmos a estrutura de menus precisamos de uma variável que armazene a opção escolhida pelo usuário, uma série de ifs que verifiquem qual opção foi escolhida e um loop while que repita a exibição do menu. Isso pode ser feito da seguinte maneira:

```
opcao = 0  
while opcao != 3:  
    print("1 - Receber um elogio")  
    print("2 - Calcular o fatorial de um número")  
    print("3 - Sair")  
    opcao = int(input("Informe o número da opção desejada  
e depois tecle ENTER: "))  
    if opcao == 1:  
        #instruções para o caso de o usuário ter escolhido  
a opção 1  
    elif opcao == 2:  
        #instruções para o caso de o usuário ter escolhido  
a opção 2  
    elif opcao == 3:  
        #Instruções para o caso de o usuário ter escolhido  
a opção 3  
    else:  
        #Instruções para o caso de o usuário não ter  
escolhido nenhuma opção válida
```

Código-fonte 19 – Estrutura de menus com o loop while
Fonte: Elaborado pelo autor (2022)

Estruturas Lógicas Condicionais e Repetições

Agora que temos a estrutura feita, vamos construir as nossas opções. Para as opções 1 e 3 (Elogio e Sair), basta exibir um print com o elogio ou com a mensagem de saída.

Para o cálculo do fatorial, implementaremos o seguinte trecho de código com o loop for:

```
numero = int(input("Informe o número do qual deseja obter  
o fatorial "))  
fat = numero  
  
for valor in range(1, numero, 1):  
    fat = fat * valor  
  
print(f"O fatorial do número informado é: {fat} ")
```

Código-fonte 20 – Trecho do código responsável pelo cálculo do fatorial
Fonte: Elaborado pelo autor (2022)

Por fim, vamos juntar tudo em um único programa e chegaremos ao seguinte script:

```
opcao = 0  
while opcao != 3:  
    print("1 - Receber um elogio")  
    print("2 - Calcular o fatorial de um número")  
    print("3 - Sair")  
    opcao = int(input("Informe o número da opção desejada  
e depois tecle ENTER: "))  
    if opcao == 1:  
        #instruções para o caso de o usuário ter escolhido  
a opção 1  
        print("Você é uma pessoa muito inteligente!")  
    elif opcao == 2:  
        #instruções para o caso de o usuário ter escolhido  
a opção 2  
        numero = int(input("Informe o número do qual  
deseja obter o fatorial "))  
        fat = numero  
  
        for valor in range(1, numero, 1):  
            fat = fat * valor  
  
        print(f"O fatorial do número informado é: {fat}  
")  
    elif opcao == 3:  
        #Instruções para o caso de o usuário ter escolhido  
a opção 3  
        print("Saindo do sistema...")
```

Estruturas Lógicas Condicionais e Repetições

```
        break
    else:
        #Instruções para o caso de o usuário não ter
        escolhido nenhuma opção válida
        print("Você não selecionou nenhuma opção válida.
        Tente novamente: ")
```

Código-fonte 21 – Programa contendo todas as estruturas do capítulo
Fonte: Elaborado pelo autor (2022)

REFERÊNCIAS

LUTZ, M. **Learning Python**. Sebastopol: O'Reilly Media Inc, 2013.

PUGA, S.; RISSETI, G. **Lógica de programação e estrutura de dados com aplicações em Java**. São Paulo: Pearson Prentice Hall, 2009.

EXEMPLO