PROTOTYPING

ANDAR EM CÍRCULOS NÃO É NECESSARIAMENTE RUIM...

3

Andar em círculos não é necessariamente ruim...

LISTA DE FIGURAS

Figura 1 – A montanha-russa não segue seu caminho sem passar pelo loop 5

Andar em círculos não é necessariamente ruim...

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Exemplo de linhas repetidas em pseudocódigo.....6
Código-fonte 2 – Exemplo de laço de repetição em pseudocódigo.....7
Código-fonte 3 – Exemplo de algoritmo com o laço enquanto em pseudocódigo.....8
Código-fonte 4 – Exemplo de laço de repetição em Python.....8
Código-fonte 5 – Exemplo de controle de repetições no laço enquanto em pseudocódigo.....9
Código-fonte 6 – Exemplo de controle de repetições no laço enquanto em Python .9
Código-fonte 7 – Sintaxe do laço para em pseudocódigo.....10
Código-fonte 8 – Exemplo do loop for em Python.....10
Código-fonte 9 – Exemplo do loop for em Python com definição do valor inicial 11
Código-fonte 10 – Exemplo do loop for em Python com definição do incremento 11
Código-fonte 11 – Estrutura de menu feita em Python usando loop while 13
Código-fonte 12 – Cálculo de média com loop *for* feito em Python.....13

Andar em círculos não é necessariamente ruim...

SUMÁRIO

1 ANDAR EM CÍRCULOS NÃO É NECESSARIAMENTE RUIM... 5
1.1 Quer que eu repita?.....5
1.1.1 Por que eu preciso de repetições?.....6
1.1.2 Enquanto houver sol.....7
1.1.3 Para lá e para cá 9
1.2 Aplicando isso tudo! 11
1.2.1 Eu preciso de um menu!.....12
1.2.2 A média de pesos!.....13
HORA DE TREINAR 14
REFERÊNCIAS.....16

1 ANDAR EM CÍRCULOS NÃO É NECESSARIAMENTE RUIM...

1.1 Quer que eu repita?

Já imaginou ter que repetir a mesma informação dezenas de vezes em um único algoritmo? Essa é uma situação comum na vida de um programador!

Seja para exibir uma simples tabuada para um usuário, percorrer centenas de linhas em uma planilha do Excel ou até fazer a extração de dados de uma página web, todas essas tarefas têm uma coisa em comum: elas podem ser resolvidas com o uso de loops!

Figura 1 – A montanha-russa não segue seu caminho sem passar pelo loop
Fonte: Pixabay (2020)

Então aperte os cintos (1), aperte os cintos (2) e aperte os cintos (3), porque neste capítulo vamos conhecer quais são os laços de repetição (loops) existentes e entender as diferenças entre eles!

Andar em círculos não é necessariamente ruim...

1.1.1 POR QUE EU PRECISO DE REPETIÇÕES?

Imagine que todos os alunos do FIAP ON estejam cadastrados em um único arquivo de texto, no formato csv (valores separados por vírgula) e que, munido dessa lista, você deve disparar um e-mail para todos os alunos menores de idade.

Depois de ficar muito bravo com quem decidiu colocar os dados em um arquivo de texto, você precisa cumprir a sua tarefa. O que fazer? Talvez verificar linha por linha o conteúdo desse arquivo?

Em um cálculo rápido, se o arquivo tiver 10.000 linhas, você precisará de 10.000 ifs para verificar se cada aluno em cada uma das linhas é maior ou menor de idade.

```
se <condição para a 1ª linha> então
    //o que ocorrerá se o aluno for menor
fim_se
se <condição para a 2ª linha> então
    //o que ocorrerá se o aluno for menor
fim_se
se <condição para a 2ª linha> então
    //o que ocorrerá se o aluno for menor
fim_se
...
se <condição para a 10.000ª linha> então
    //o que ocorrerá se o aluno for menor
fim_se
```

Código-fonte 1 – Exemplo de linhas repetidas em pseudocódigo
Fonte: Elaborado pelo autor (2020)

Além de ser contraproducente fazer a digitação manual de cada um desses blocos, o programa fica mais suscetível a falhas por conta do desenvolvedor e menos passível de manutenção.

Um loop é uma estrutura que permite indicar a *repetição de uma tarefa* em um determinado número de vezes. Dessa maneira, conseguimos fazer com que o programa repita a tarefa de verificar a idade para cada uma das linhas do arquivo:

Andar em círculos não é necessariamente ruim...

```
      para linha de 1 até 10000 passo 1 faça
          se <condição para linha> então
              //o que acontecerá se o aluno for menor
          fim_se
      fim_para
```

Código-fonte 2 – Exemplo de laço de repetição em pseudocódigo
Fonte: Elaborado pelo autor (2020)

Seguidos dos desvios condicionais, os laços de repetição figuram entre os recursos mais importantes que um programador deve conhecer.

Em linhas gerais, há dois tipos de laços de repetição presentes na maior parte das linguagens de programação: o laço *enquanto* (while) e o laço *para* (for).

Apesar de poderem ser usados nas mesmas situações, os laços com frequência possuem funcionamentos e objetivos diferentes, que vamos explorar agora.

1.1.2 ENQUANTO HOUVER SOL

Que tal prendermos o usuário em uma armadilha? Faremos um algoritmo para que ele seja obrigado a provar que é um verdadeiro *nerd* ao responder: “Qual é a resposta para a vida, o universo e tudo mais?”, e o programa não encerrará até que ele dê a resposta correta (que, segundo o livro *O guia do mochileiro das galáxias*, de Douglas Adams, é “42”).

O que queremos é **repetir** uma pergunta ao usuário **enquanto** ele não acertar a resposta.

O loop while serve exatamente para esse tipo de situação: não temos um número definido de repetições, nem mesmo um limite para isso. Temos apenas uma **condição** que permite ao loop continuar ou parar.

Dessa forma, ele é um *laço de repetição baseado em condição*.

O algoritmo da nossa brincadeira fica mais ou menos assim:

```
      Variáveis:
          resposta: alfanumérico
      Início
          resposta = "0"
          Enquanto (resposta != "42") faça
              Escreva "Qual é a resposta para a vida, o
universo e tudo mais?"
```

Andar em círculos não é necessariamente ruim...

```
            Leia resposta
        Fim_enquanto
        Escreva "Parabéns, você acertou!"
    Fim
```

Código-fonte 3 – Exemplo de algoritmo com o laço enquanto em pseudocódigo
Fonte: Elaborado pelo autor (2020)

Esse mesmo algoritmo, escrito no script exemplo_while.py, fica assim:

```
    #criação da variável com um valor inicial
    resposta = "0"
    #enquanto a resposta não for 42, a repetição ocorre
    while (resposta != "42"):
        #para cada uma das repetições, o usuário vai submeter
uma resposta
        resposta = input("Qual a resposta para a vida, o
universo e tudo mais?")
    #quanto o laço terminar, a mensagem é exibida
    print("Parabéns, você acertou!")
```

Código-fonte 4 – Exemplo de laço de repetição em Python
Fonte: Elaborado pelo autor (2020)

Muitos autores caracterizam o laço de repetição como "potencialmente infinito", pois, se não nos atentarmos à condição criada, o programa ficará preso para sempre nesse ciclo.

Por outro lado, haverá cenários em que o programador desejará criar um loop infinito de forma proposital, como em uma thread que verifica se existem novas mensagens em uma comunicação por sockets.

Mas, antes de chegarmos a esses problemas mais elaborados, vamos continuar explorando o While: o que podemos fazer se quisermos controlar quantas repetições serão realizadas?

É aí que entra em cena a figura da **variável contadora**. Ela nada mais é do que uma variável que será usada como condição do nosso loop, sendo incrementada a cada volta realizada.

Para que tenhamos uma repetição de 10 voltas, podemos pensar em um algoritmo próximo ao seguinte:

Andar em círculos não é necessariamente ruim...

```
Variáveis:
    i: inteiro
Início
    i = 0
    Enquanto (i<10) faça
        Escreva "Mais uma mensagem, com i valendo: ", i
        i = i + 1
    Fim_enquanto
Fim
```

Código-fonte 5 – Exemplo de controle de repetições no laço enquanto em pseudocódigo
Fonte: Elaborado pelo autor (2020)

Perceba que, a cada volta, fazemos o incremento da variável i (i = i + 1), aumentando o seu valor para que, eventualmente, ela atinja o limite estipulado na condição do laço (i<10).

Se transportarmos essa mesma lógica para a linguagem Python, obteremos o seguinte código-fonte:

```
#criação da variável com um valor inicial
i = 0
#enquanto a variável contadora não chegar ao limite
while (i<10):
    #para cada uma das repetições uma mensagem é exibida
    print("Mais uma mensagem, com i valendo: {}".format(i))
    i = i + 1
```

Código-fonte 6 – Exemplo de controle de repetições no laço enquanto em Python
Fonte: Elaborado pelo autor (2020)

Da mesma forma como realizamos o incremento na variável i, poderíamos ter realizado um decremento.

Algumas linguagens de programação suportam notações como "i++" ou "i+=1" para a operação de incremento na variável contadora.

1.1.3 PARA LÁ E PARA CÁ

Já sabemos que um loop serve para repetir a execução de um determinado trecho do nosso programa, e podemos até mesmo utilizar o while aliado a uma variável contadora para controlar o número de vezes que isso vai acontecer.

Andar em círculos não é necessariamente ruim...

Quando estamos diante de um problema que exige um número determinado de repetições, há uma estrutura mais apropriada do que o loop while: o laço de repetição *para*.

A ideia de funcionamento do laço de repetição *para* é baseada em determinarmos um ponto inicial e um ponto final para a repetição, sendo que a própria estrutura se encarregará de controlar o número de voltas.

Analisando a sintaxe em pseudocódigo, conseguimos entender bem esse funcionamento:

```
      para <contadora> de <valor inicial> até <valor final>
passo <incremento> faça
          //instruções que serão repetidas
      fim_para
```

Código-fonte 7 – Sintaxe do laço para em pseudocódigo
Fonte: Elaborado pelo autor (2020)

A estrutura de loop *for* é frequentemente utilizada pelos programadores pela sua facilidade de compreensão e implementação.

Se quisermos uma repetição de 10 vezes em linguagem Python, podemos fazer uso de uma função chamada *range*, que definirá um intervalo de valores para a nossa variável contadora assumir.

Veja que simples:

```
      #a variável contadora deverá assumir valores no intervalo
entre 0 e 9
      for x in range(10):
          #para cada um desses valores, vamos printar o valor
da variável
          print(x)
```

Código-fonte 8 – Exemplo do loop for em Python
Fonte: Elaborado pelo autor (2020)

Se traduzirmos a escrita do código usando rudimentos de inglês instrumental (sem referências a qualquer treinador de futebol), poderemos ler: "*para a variável x no intervalo entre 0 e 9, faça:"*.

Mas vamos supor que você não queira que a sua variável contadora inicie com o valor de 0. Por alguma razão, você deseja controlar o valor inicial dessa variável.

Andar em círculos não é necessariamente ruim...

Isso é possível alterando a função range com a inclusão de outro argumento. Se escrevermos *range(1,15)*, por exemplo, o valor inicial da variável será 1 e o valor final da variável contadora será 14.

```
      #a variável contadora deverá assumir valores no intervalo
entre 1 e 14
      for x in range(1,15):
          #para cada um desses valores, vamos printar o valor
da variável
          print(x)
```

Código-fonte 9 – Exemplo do loop for em Python com definição do valor inicial
Fonte: Elaborado pelo autor (2020)

Ainda é possível que você esteja coçando a cabeça e se perguntando: “Professor! Como podemos fazer para controlar o incremento da variável contadora?”. Essa é fácil! Basta incluirmos mais um argumento na função range.

Se escrevermos *range(1,10,2)*, a variável assumirá o valor inicial como sendo 1, o valor final como sendo 9, e a cada volta ela somará 2. Portanto terá os valores: 1, 3, 5, 7 e 9.

```
      #a variável contadora deverá assumir valores no intervalo
entre 1 e 9 com incremento 2
      for x in range(1,10,2):
          #para cada um desses valores, vamos printar o valor
da variável
          print(x)
```

Código-fonte 10 – Exemplo do loop for em Python com definição do incremento
Fonte: Elaborado pelo autor (2020)

O loop for é extremamente simples de usar, e com o passar do tempo ele estará presente em quase todos os seus programas. Falando nisso, que tal vermos algumas aplicações práticas?

1.2 APLICANDO ISSO TUDO!

Sem conhecer as estruturas de repetição, um programador é um profissional incompleto! Mas... por qual razão, mesmo?

Andar em círculos não é necessariamente ruim...

Depois de termos compreendido o funcionamento dos loops, vamos trabalhar com alguns exemplos para compreendermos melhor como essas estruturas podem nos ajudar.

1.2.1 EU PRECISO DE UM MENU!

Uma das frustrações mais comuns nos nossos primeiros programas é que eles “acabam”. O cenário é quase sempre o mesmo: escrevemos um código, ficamos orgulhosos, testamos e quando o funcionamento previsto termina, o programa se encerra.

Para solucionarmos esse problema, uma excelente solução é criar uma estrutura de menus, na qual o usuário possa escolher se pretende continuar executando o programa ou se quer finalizar o ciclo.

E quem pode nos ajudar nesse caso é o loop while!

Com esse loop, podemos estabelecer a lógica: enquanto o usuário não pressionar uma determinada opção, o menu continua sendo exibido.

Para não perdermos tempo decidindo o que o menu conterá, vamos apenas exibir mensagens para cada uma das opções, OK?

Dessa forma, chegaremos ao seguinte programa em Python:

```
#variável que servirá para receber a opção do usuário
op = -1
#enquanto o usuário não digitar a opção de saída
while op!=4:
    #exibição das opções do menu
    print("SUPER PROGRAMA MARAVILHOSO")
    print("1 - Rodar código 1")
    print("2 - Rodar código 2")
    print("3 - Rodar código 3")
    print("4 - Sair do programa")
    op = int(input("Informe sua opção: "))

    #verificação das opções disponíveis
    if op == 1:
        #O que ocorrerá se a opção 1 for selecionada
        print("CÓDIGO 1 RODANDO!")
    elif op == 2:
        #O que ocorrerá se a opção 2 for selecionada
        print("CÓDIGO 2 RODANDO!")
```

Andar em círculos não é necessariamente ruim...

```
            elif op == 3:
                #O que ocorrerá se a opção 3 for selecionada
                print("CÓDIGO 3 RODANDO!")

        #Quando o looping terminar de rodar, exibir essa mensagem
        print("OK! O programa está encerrado...")
```

Código-fonte 11 – Estrutura de menu feita em Python usando loop while
Fonte: Elaborado pelo autor (2020)

1.2.2 A MÉDIA DE PESOS!

No caminhão de uma empresa de transportes, cabem exatamente 100 caixas de iguais dimensões. O peso dessas caixas, porém, pode variar dependendo do seu conteúdo.

Como alguns caminhões têm se desgastado mais do que outros, você foi convocado para criar um algoritmo que calcule o peso total das caixas que serão colocadas em um caminhão e que exiba o peso médio das caixas.

A solução mais simples para isso é pedir que o usuário vá digitando o peso das caixas à medida que elas são colocadas no caminhão (ou, futuramente, integrar nosso sistema com a balança). Como sabemos que se tratam de 100 caixas, podemos utilizar o loop for para nos ajudar!

O script que soluciona esse problema fica assim:

```
        #variavel para armazenar o peso total das caixas
        peso_total = 0.0
        #loop para repetir por 100 vezes a solicitação de peso
        for x in range(1,101):
            #para cada volta do loop, solicitar o peso da caixa
            peso_atual = float(input("Informe o peso da caixa
atual:"))
            #para cada peso solicitado, somar ao peso total
            peso_total = peso_total + peso_atual

        #ao final do loop, calcular a média aritmética
        media = peso_total/100

        #exibição dos resultados!
        print("O peso total de caixas é {}. \nA média de peso é
{}".format(peso_total, media))
```

Código-fonte 12 – Cálculo de média com loop *for* feito em Python
Fonte: Elaborado pelo autor (2020)

Andar em círculos não é necessariamente ruim...

HORA DE TREINAR

1. Você deve elaborar um algoritmo implementado em Python em que o usuário informe quantos alimentos consumiu naquele dia e depois possa informar o número de calorias de cada um dos alimentos. Como não estudamos listas neste capítulo, você não deve se preocupar em armazenar todas as calorias digitadas, mas deve exibir o total de calorias no final.

2. Observando o mercado de educação infantil, você e sua equipe decidem criar um aplicativo por meio do qual as crianças aprendam a controlar os seus gastos.

Como forma de validar um protótipo, foi solicitado que você crie um script simples, em que o usuário deve informar QUANTAS TRANSAÇÕES financeiras realizou ao longo de um dia e, na sequência, deve informar o VALOR DE CADA UMA das transações que realizou.

Seu programa deverá exibir, ao final, o valor total gasto pelo usuário, bem como a média do valor de cada transação.

3. Uma grande empresa de jogos deseja tornar seus games mais desafiadores. Por isso ela contratou você para desenvolver um algoritmo que será aplicado futuramente em diversos outros games: o algoritmo da sorte de Fibonacci.

A ideia dessa empresa, é claro, é fazer com que seja mais difícil os jogadores terem sucesso nas ações que realizam nos games. Por isso o seu algoritmo deverá funcionar da seguinte forma: o usuário deve digitar um valor numérico inteiro e o algoritmo deverá verificar se esse valor encontra-se na sequência de Fibonacci. Caso o número esteja na sequência, o algoritmo deve exibir a mensagem “Ação bem-sucedida!”, e caso não esteja, deve exibir a mensagem “A ação falhou...”.

A sequência de Fibonacci é muito simples e possui uma lógica de fácil compreensão: ela se inicia em 1 e cada novo elemento da sequência é a soma dos dois elementos anteriores. Portanto: 1, 1, 2, 3, 5, 8, 13, 21, e assim por diante.

Logo, se o usuário digitar o número 55 o programa deverá informar que a ação foi bem-sucedida, afinal 55 está entre os números da sequência.

Andar em círculos não é necessariamente ruim...

Mas, se o usuário digitar o número 6, por exemplo, a ação não será bem-sucedida, pois o número 6 não está na sequência.

4. Um grande cliente seu sofreu um ataque hacker: o servidor foi sequestrado por um software malicioso que criptografou todos os discos e pede a digitação de uma senha para a liberação da máquina. E é claro que os criminosos exigem um pagamento para informar a senha.

Ao analisar o código do programa deles, porém, você descobre que a senha é composta pela palavra “LIBERDADE” seguida do fatorial dos minutos que a máquina estiver marcando no momento da digitação da senha (se a máquina estiver marcando 5 minutos, a senha será LIBERDADE120). Crie um programa que receba do usuário os minutos atuais e exiba na tela a senha necessária para desbloqueio. ATENÇÃO: seu programa não pode utilizar funções prontas para o cálculo do fatorial. Ele deve obrigatoriamente utilizar loop.

CORREÇÕES DOS EXERCÍCIOS TREINO

Andar em círculos não é necessariamente ruim...

REFERÊNCIAS

PIVA JÚNIOR, Dilermando et al. **Algoritmos e programação de computadores**. São Paulo: Elsevier, 2012.

PUGA, Sandra; RISSETTI, Gerson. **Lógica de Programação e Estrutura de Dados**. São Paulo: Pearson Prentice Hall, 2009.

RAMALHO, Luciano. **Python Fluente:** Programação Clara, Concisa e Eficaz. São Paulo: Novatec, 2015.