FIAP ON

APRENDENDO TECNOLOGIA COM DESENVOLVIMENTO

ESTRUTURAS AVANÇADAS
DE LÓGICA

ANDRÉ DE FREITAS DAVID

09

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 ESTRUTURAS AVANÇADAS DE LÓGICA

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.

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 inserção 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:

```
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)

Resumidamente, uma estrutura de repetição é utilizada em um algoritmo todas as vezes que um trecho do seu código for redundante, ou seja, quando parte do seu código “clamar” por repetição.

Como saber se o trecho do programa necessita de repetição? Pois é, este é o grande desafio. Interpretar a real necessidade dos subproblemas do problema original é o nosso desafio.

Vamos utilizar um exemplo mais simples para definirmos a necessidade ou não de utilização de laços: Imagine que alguém queira exibir um número na tela. Para tal usaríamos:

```
Escreva n
```

Resolver este problema sugere repetição? Não! Porque desejamos simplesmente exibir um número **n** na tela.

Mudando o contexto deste problema: e se precisássemos exibir os números de 1 a 100 na tela, este novo problema sugere repetição? Agora o problema mudou. Desejamos mostrar 100 números. Sim, esta rotina sugere repetição; então, vamos estruturar um laço que dê 100 voltas e dentro dele colocaremos apenas um comando Escreva:

```
para n de 1 até 100 passo 1 faça
    Escreva n
Fim_para
```

Código-fonte 3 – Exemplo de laço de repetição que exibe 100 números
Fonte: Elaborado por Edson de Oliveira (2022)

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

Em lógica de programação há três estruturas (laços) de repetição:

- Laço contador Para (**for** em Python).
- Laço pré-condicional Enquanto-Faça (**while** em python).
- Laço pós-condicional Faça-Enquanto (Não existe esta estrutura em Python).

O laço pós-condicional Faça-Enquanto não existe no Python? Sim! (sim de 'não existe', olhe a lógica). Dependendo da literatura que trata pseudocódigo (ou portugol), o laço pós-condicional é conhecido como *Faça-Enquanto* ou *Repita até que*. Detalharemos na sequência do capítulo.

Apesar de poderem ser usados nas mesmas situações em quase todos os casos, os laços possuem funcionamentos e objetivos diferentes, que vamos explorar agora.

1.1.2 Enquanto for verdade vamos repetir

A primeira estrutura de repetição que veremos é o laço pré-condicional Enquanto-faça (while). Antes de detalharmos, vamos visualizar a mecânica de seu funcionamento no fluxograma:

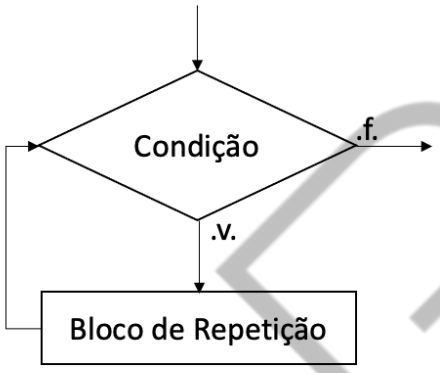

Figura 2 – Exemplo do laço pré-condicional: Enquanto-faça no Fluxograma
Fonte: Elaborado por Edson de Oliveira (2022)

Antes do fluxo do algoritmo chegar no laço, analisamos inicialmente uma *condição* proposta para executar (ou não) o *bloco de repetição*. Caso esta condição resulte *verdade* o *bloco de repetição* será executado e depois o fluxo retorna à *condição* (caracterizando o apelido 'laço': os dois pontos ficam na mesma extremidade); repetindo este processo enquanto a condição resulte *verdade* ou saindo do laço até que ela resulte *falso*.

Veja agora a sintaxe da estrutura Enquanto no Pseudocódigo:

```
Enquanto <condição> Faça
    <Bloco de Repetição>
Fim_enquanto
```

Código-fonte 4 – Sintaxe do laço Enquanto no Pseudocódigo
Fonte: Elaborado pelo autor (2020)

A palavra *Enquanto* inicia o laço e o *Fim_enquanto* é o limite de até onde será repetido o *bloco de repetição*. Sempre que o fluxo chegar ao *Fim_enquanto* ele retornará à linha do *Enquanto* para analisar novamente a condição, executando o *bloco de repetição* se continuar resultando *verdade* ou caso resulte *falso*, prosseguirá o fluxo do algoritmo executando as instruções que eventualmente vierem depois do Fim_Enquanto.

O mecanismo de funcionamento do pseudocódigo e do fluxograma é o mesmo. Contudo, a comparação entre a visualização gráfica (fluxograma) e a codificação (pseudocódigo ou Python) requer algum cuidado.

No fluxograma, as setas indicam qual a próxima instrução a ser executada, mas no pseudocódigo esta visualização não é clara. Por exemplo, no fluxograma, ao executar o *bloco de repetição,* a seta retorna (aponta) à *condição*, agora no pseudocódigo devemos saber que ao chegar ao *Fim_enquanto*, o fluxo deve retornar para a condição que está na linha do Enquanto.

A sintaxe da estrutura pós-condicional Enquanto no Python (while) é:

```
while <condição>:
    <Bloco de Repetição>
```

Código-fonte 5 – Sintaxe do laço while (Enquanto) no Python
Fonte: Elaborado pelo Edson de Oliveira (2022)

A codificação em Python é semelhante ao pseudocódigo, lembrando apenas que a caracterização de blocos em Python é definida pela indentação do código.

A característica mais importante do laço Enquanto, que o diferencia dos demais é o fato de ele ser um laço 0, N. Como assim? Ele tem a característica de executar o bloco de repetição no mínimo nenhuma (zero) vez e no máximo várias (N). O que isso quer dizer? Este laço é mais bem utilizado em situações em que há a possibilidade de um *bloco de repetição* não ser executado.

Exemplificando: imagine que desejamos fazer uma consistência de dados em um algoritmo e obrigar que o usuário digite (somente) S para Sim ou N para Não em resposta a algo; caso o usuário digite uma letra diferente destas duas, será fornecida uma mensagem de advertência “Opção inválida! Digite [S]im ou [N]ão.” Até que ele digite uma letra correta (consistente).

Este caso representa situação 0, N porque se de cara o usuário digitar uma das letras válidas, o *bloco de repetição,* que contém a advertência, não será executado: situação zero; caso o usuário erre a letra, a mensagem de advertência será exibida até que ele acerte. Vejamos a sua implementação no Pseudocódigo (para um melhor entendimento dos comentários posteriores), neste pseudocódigo numeraremos as linhas:

```
1 Programa Exemplo_Enquanto_faça
2 Var
3     resp: caractere
4  Início
5    Escreva "Digite [S]im ou [N]ão"
6    Leia resp
7    Enquanto resp != 'S' .e. resp != 'N' Faça
8        Escreva "Opção inválida! Digite [S]im ou [N]ão."
9       Leia resp
10   Fim_enquanto
11   Escreva "Você digitou a letra válida", resp
12 Fim
```

Código-fonte 6 – Exemplo "Sim ou não" utilizando o laço Enquanto no Pseudocódigo
Fonte: Elaborado por Edson de Oliveira (2022)

Vejamos dois cenários:

Cenário 1 – Se na linha 6 o usuário digitar 'S' na variável *resp*, a condição da linha 7, que está no enquanto, resultará falso. Vamos conferir?

```
resp != 'S' .e. resp != 'N'   // condição do enquanto
'S'  != 'S' .e. 'S' != 'N'    // substituindo resp por S
     .f.    .e.     .v.       // calculando...
            .f.               // resposta final
```

Resultou falso. Como o enquanto prossegue no laço enquanto for verdade, se resultar falso ele pula da linha 6 para a 11, ou seja, não executando nenhuma vez o bloco de repetição que está nas linhas 8 e 9 indo direto para a linha 11 com a mensagem: `Você digitou a letra válida S`.

Cenário 2 – Se na linha 6 o usuário digitar 'H' na variável *resp*, a condição da linha 7, que está no enquanto, resultará verdade. Vamos conferir?

```
resp != 'S' .e. resp != 'N'   // condição do enquanto
'H'  != 'S' .e. 'H' != 'N'    // substituindo resp por H
     .v.    .e.     .v.       // calculando...
            .v.               // resposta final
```

Neste caso resultou verdadeiro. Como o laço *Enquanto* prossegue enquanto for verdade, ele executou o *bloco de repetição* que está nas linhas 8 e 9, com a mensagem de advertência. Este processo se repetirá até que ele digite uma das letras válidas.

Vejamos a implementação deste exemplo no Pyhton:

```
resp = input("Digite [S]im ou [N]ão: ")
 while resp != 'S' and resp != 'N':
     print("Opção inválida! Digite [S]im ou [N]ão.")
     resp = input()
 print("Você digitou a letra válida", resp)
```

Código-fonte 7 – Exemplo “Sim ou não” utilizando o laço while no Python
Fonte: Elaborado por Edson de Oliveira (2022)

O Python, por ser uma linguagem dinâmica e flexível, há outros meios de executar as mesmas ações. Vamos ver como fica o mesmo programa utilizando o operador de associação *not in()*:

```
resp = input("Digite [S]im ou [N]ão: ")
 while resp not in ('S', 'N'):
     print("Opção inválida! Digite [S]im ou [N]ão.")
     resp = input()
 print("Você digitou a letra válida", resp)
```

Código-fonte 8 – Mesmo exemplo utilizando o laço while com operador de associação no Python
Fonte: Elaborado por Edson de Oliveira (2022)

A linha com a instrução:

```
while resp not in ('S', 'N'):
```

Pode ser traduzida como:

```
Enquanto resp não for 'S' ou 'N'
```

Viram a versatilidade do Python? Nem todas as linguagens têm esta versatilidade, sendo assim, o importante é sabemos resolver o problema com artifícios da lógica e utilizarmos a flexibilidade apenas quando a linguagem permitir.

Vamos a um outro exemplo?

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?”
             Leia resposta
         Fim_enquanto
         Escreva “Parabéns, você acertou!”
     Fim
```

Código-fonte 9 – 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 10 – Exemplo de laço de repetição em Python
Fonte: Elaborado pelo autor (2020)

Muitos autores caracterizam o laço de repetição pré ou pós-condicional como “potencialmente infinito”, pois, se não nos atentarmos à condição criada, o programa ficará preso para sempre nesse ciclo.

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

A utilização de thread ou sockets é um assunto adiante.

Mas como funciona o laço de repetição pós-condicional? Veremos a partir de agora.

1.1.3 Vamos repetir enquanto for verdade

Esta é a segunda estrutura de repetição: o laço pós-condicional *Faça-Enquanto*. Este laço não existe no Python, e agora? Antes de respondermos esta pergunta vamos visualizar o seu funcionamento através do fluxograma:

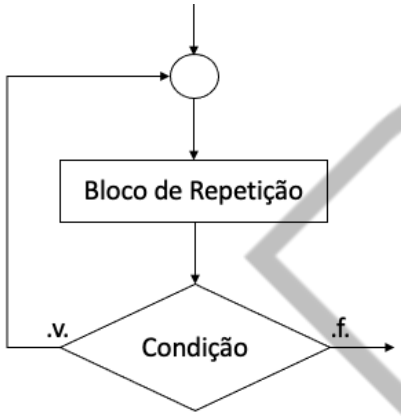

Figura 3 – Estrutura do laço Faça-Enquanto no Fluxograma
Fonte: Elaborado por Edson de Oliveira (2022)

Comparando o laço pós-condicional *Faça-Enquanto* com o laço pré-condicional *Enquanto-Faça* a condição inverte, ou seja, no *Faça-Enquanto* primeiro é executado o *bloco de repetição* e depois é analisada a *condição* para prosseguir no laço ou não, ao contrário do laço pré-condicional.

Representação do laço pós-condicional *Faça-Enquanto* no pseudocódigo:

```
Faça
    Bloco de Repetição
Enquanto Condição
```

Código-fonte 11 – Estrutura do laço Faça-Enquanto no Pseudocódigo
Fonte: Elaborado por Edson de Oliveira (2022)

E esta história dele não existir no Python? Calma, temos um pouco mais a falar sobre esta estrutura antes de respondermos esta pergunta!

Na teoria, para os programadores “raízes”, o laço pós-condicional também é conhecido como *Repita até que* (repeat until):

```
Repita  // Repeat
    Bloco de Repetição
Até que Condição // until
```

Código-fonte 12 – Laço Repita no Pseudocódigo
Fonte: Elaborado por Edson de Oliveira (2022)

A linguagem C e suas derivadas (C++, C#, Java, JavaScript...) utilizam o comando ***do while()*** para representar o Faça-Enquanto, enquanto a linguagem de programação Delphi e a sua nativa Pascal (entre outras), utilizam o *Repeat until* para representar o *Repita até que*.

Então, qual a diferença do *Faça-Enquanto* para o *Repita-até que*?

A estrutura *Faça-Enquanto* executa o laço enquanto a condição resultar **verdade** e a estrutura *Repita-Até que*, executa a repetição **até** que a condição seja verdadeira, ou seja, executa enquanto a condição for **falsa**.

Esta comparação foi feita apenas para que você pudesse perceber que as linguagens têm formas diferentes de tratar o mesmo comando, seja por sintaxe ou detalhes do funcionamento ou até não existir, como neste caso com o Python.

É importante que você saiba dessa teoria, porque com a evolução do seu aprendizado, verá outras linguagens que não são tão versáteis ou dinâmicas como o Python, mas que utilizem este (ou outros) comandos ou recursos para fazer funcionar algo. O importante neste ponto é você saber que tem esta possibilidade.

Falando de conceito. O que o comando *Faça-Enquanto* tem de diferente do comando *Enquanto-faça*? O segundo, como vimos, primeiro analisa a *condição* e depois executa o *bloco de repetição* caso a *condição* resulte verdade, enquanto o primeiro inicialmente executa o *bloco de repetição* e depois analisa através da *condição* se deve continuar.

Isso faz com que o laço Faça-enquanto tenha a característica 1, N; ou seja, executa ao menos uma vez o bloco de repetição (diferente do *Enquanto-faça* que pode executar nenhuma vez o bloco de repetição: 0,N) e executa no máximo várias vezes o bloco de repetição.

Vamos resumir o conceito deste laço com um questionamento:

Em qual situação privilegiamos o laço Faça-Enquanto aos outros? Em situações em que o bloco de repetição tenha que ser executado ao menos uma vez.

Exemplificando: considere que um algoritmo leia números, efetue a somatória e no final exiba o total da soma. Concorda que para este caso ao menos um número deverá ser digitado (e, consequentemente, ao menos uma volta)? Então vamos utilizar este problema para demonstrarmos o funcionamento do laço Faça-enquanto.

Quando estamos aprendendo a usar laços, a maior dificuldade é sabermos “o que fica dentro do laço e o que fica fora”. Para facilitar este entendimento, vamos usar a descrição narrativa do algoritmo:

Primeiro vamos definir os passos importantes:

- Ler os números
- Somar os números
- Exibir a soma dos números

Agora analise cada passo individualmente e pergunte: Este laço requer repetição? Se a resposta for sim, ele estará dentro do laço, senão ficará fora do laço.

Vamos lá:

1. Ler os números requer repetição? Sim!
2. Somar os números requer repetição? Sim!
3. Exibir a soma dos números requer repetição? Não!

Assim, o primeiro e segundo passos formarão o bloco de repetição porque requerem repetição, enquanto o terceiro passo será colocado no algoritmo depois do término do laço.

Vejamos a aplicação no pseudocódigo:

```
Programa Ex_Faca_Enquanto
Var
    somatoria,num: Real
Início
    somatoria = 0
    Escreva “Digite números ou 0 para finalizar...”
    Faça
1.      Leia num
2.      somatoria = somatoria + num
    Enquanto num != 0
3.  Escreva “Somatória = “, Somatória
```

Código-fonte 13 – Pseudocódigo como Faça-Enquanto: Somatória
Fonte: Elaborado por Edson de Oliveira (2022)

Reparem que no pseudocódigo foi colocado o número que representa cada passo da descrição narrativa para a visualização de onde cada passo deve estar. Concluímos que os dois primeiros passos devem estar dentro do laço e o terceiro fora.

Como ficaria esta construção no Python? Opa! Não foi dito que este laço não existe no Python? Verdade, não existe!

Faremos agora uma simulação deste laço no Python:

```
somatoria = 0
 print("Digite números ou 0 para finalizar...")
 while True:
     num = float(input())
     somatoria = somatoria + num
     if num == 0:
         break
 print(f"Somatoria = {somatoria}")
```

Código-fonte 14 – Python simulando o Faça-Enquanto: Somatória
Fonte: Elaborado por Edson de Oliveira (2022)

Nesta simulação, colocamos o comando `while True:` que quer dizer "faça infinitamente"; assim, chegando neste comando entra no laço incondicionalmente. Para simular a condição no final do laço, colocamos o `if num == 0:` para caso o usuário tenha digitado 0, o `break` força a saída do laço.

Assim, contemplamos o conceito do laço Faça-enquanto no Python. Foi executado ao menos uma vez o bloco e a condição de saída ou continuidade do laço foi colocada no final da estrutura de repetição.

Mas, antes de chegarmos a esses problemas mais elaborados, vamos continuar explorando o While. Então, 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:

```
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 15 – 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 16 – 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.

1.1.4 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.

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.

Veja a estrutura no laço contador Para (for):

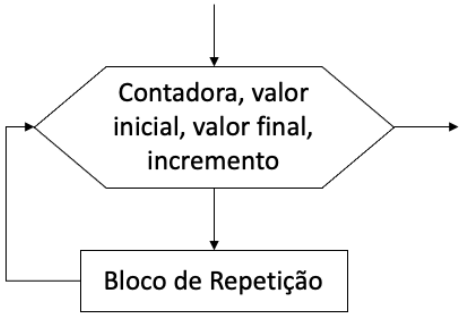

Figura 4 – Laço contador Para no fluxograma
Fonte: Elaborado por Edson de Oliveira (2022)

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 17 – 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 18 – 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 0. Por alguma razão, você deseja controlar o valor inicial dessa variável.

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 19 – 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 20 – 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!

Para todos os algoritmos que envolvam laços, conseguimos resolver todos os estilos de laços, mas sempre há um laço mais apropriado para cada problema:

- **Enquanto-Faça (while):** para situações condicionais (voltas indeterminadas), em que pode ocorrer de não executar nenhuma vez o bloco de repetição.

- **Faça-Enquanto (while True):** para situações condicionais (voltas indeterminadas), em que deve ao menos executar uma vez o bloco de repetição.
- **Para (for):** para situações de contagem determinadas de voltas (finitas), em que o programador tem o maior poder de manipulação do intervalo das voltas através do contador.

Profissionalmente, um programa deve ser escrito para um fácil entendimento, porque outras pessoas darão manutenção em um programa seu ou o contrário. Assim, privilegiar o melhor laço para um problema deixa o seu código mais claro.

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

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 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 sobre 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!")
    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 21 – 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 trata 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 22 – Cálculo de média com loop *for* feito em Python
Fonte: Elaborado pelo autor (2020)

1.2.3 Contagem de alunos aprovados!

Vamos considerar que em uma turma da FIAP tenha 40 alunos. O algoritmo deverá ler todas as notas de todos os alunos de uma determinada disciplina. Os critérios para a aprovação é o seguinte:

- 3 Checkpoints (Tirando a menor nota) – 20%
- 2 sprints (Challenge) – 20%
- 1 prova semestral – 60%

Este critério é para cada semestre. O que diferencia um semestre do outro é que a média do primeiro vale 40% da média final enquanto a média do segundo semestre vale 60%.

Depois de lidas as notas, deve-se efetuar o cálculo das médias de cada aluno, exibi-las com o status ‘Aprovado’ (média de ao menos 6) ou ‘Não aprovado’ e contar quantos alunos foram aprovados direto.

Para resolvermos este problema, codificaremos com os três laços. Começando pelo laço Enquanto-Faça (while):

```
contagemAprovados = 0
qtdAlunos = 40
aluno = 1
while aluno <= qtdAlunos:
    # P R I M E I R O   S E M E S T R E
    print(f"------------------------------ ALUNO: {aluno}")
    print("PRIMEIRO SEMESTRE")
    # Leitura dos checkpoints
    chk1 = float(input("Digite o Checkpoint 1:"))
    chk2 = float(input("Digite o Checkpoint 2:"))
    chk3 = float(input("Digite o Checkpoint 3:"))
    # Verificando o Checkpoint de menor valor
    menor = chk1
    if chk2 < menor:
        menor = chk2
    if chk3 < menor:
        menor = chk3
    # Calculando a média dos Checkpoints
    mediaChk = (chk1 + chk2 + chk3 - menor) / 2
    # Leitura dos Sprints
    sprint1 = float(input("Digite o Sprint 1:"))
    sprint2 = float(input("Digite o Sprint 2:"))
    # Calculando a média dos Sprints
    mediaSprint = (sprint1 + sprint2) / 2
    # Leitura da prova semestral
    semestral = float(input("Digite prova semestral:"))
    # Ponderando os valores das médias
    pontosChk = mediaChk * 0.2
    pontosSprints = mediaSprint * 0.2
    pontosSemestral = semestral * 0.6
    # Cálculo da media do primeiro semestre
    mediaPrimeiroSemestre = (pontosChk + pontosSprints +
pontosSemestral)
    # Pontos obtidos no primeiro semestre
    pontosPrimeiroSemestre = mediaPrimeiroSemestre * 0.4

    # S E G U N D O   S E M E S T R E
    print("SEGUNDO SEMESTRE")
    # Leitura dos checkpoints
    chk1 = float(input("Digite o Checkpoint 1:"))
    chk2 = float(input("Digite o Checkpoint 2:"))
    chk3 = float(input("Digite o Checkpoint 3:"))
    # Verificando o Checkpoint de menor valor
    menor = chk1
    if chk2 < menor:
        menor = chk2
    if chk3 < menor:
        menor = chk3
    # Calculando a média dos Checkpoints
    mediaChk = (chk1 + chk2 + chk3 - menor) / 2
    # Leitura dos Sprints
```

```
    sprint1 = float(input("Digite o Sprint 1:"))
    sprint2 = float(input("Digite o Sprint 2:"))
    # Calculando a média dos Sprints
    mediaSprint = (sprint1 + sprint2) / 2
    # Leitura da prova semestral
    semestral = float(input("Digite prova semestral:"))
    # Ponderando os valores das médias
    pontosChk = mediaChk * 0.2
    pontosSprints = mediaSprint * 0.2
    pontosSemestral = semestral * 0.6
    # Cálculo da media do segundo semestre
    mediaSegundoSemestre = (pontosChk + pontosSprints +
pontosSemestral)
    # Pontos obtidos no segundo semestre
    pontosSegundoSemestre = mediaSegundoSemestre * 0.6

    # Exibiçao da media do primeiro semetre
    print (f"\nMédia do Primeiro Semestre:
{mediaPrimeiroSemestre:.1f}")
    # Exibiçao da media do segundo semetre
    print (f"Média do Segundo Semestre:
{mediaSegundoSemestre:.1f}")
    # Cálculo da média final
    mediaFinal = pontosPrimeiroSemestre +
pontosSegundoSemestre
    print (f"Média Final: {mediaFinal:.1f} - ", end="")
    if mediaFinal >= 6:
        contagemAprovados += 1
        print("Aprovado!")
    else:
        print("Não Aprovado!")
    aluno += 1

# Exibição da Quantidade de Alunos Aprovados
print("Quantidade de aprovados: ", contagemAprovados)
```

Código-fonte 23 – Contagem de aprovados: while (enquanto) em Python
Fonte: Elaborado por Edson de Oliveira (2022)

Feito o algoritmo com o laço Enquanto-faça, agora faremos o mesmo código com o laço Faça-enquanto.

Lembrando que este laço não existe explicitamente no Python, faremos então uma simulação:

```
contagemAprovados = 0
qtdAlunos = 40
aluno = 1
# Simulação do laço Faça-enquanto
while True:
    # P R I M E I R O   S E M E S T R E
    print(f"------------------------------ ALUNO: {aluno}")
    print("PRIMEIRO SEMESTRE")
    # Leitura dos checkpoints
    chk1 = float(input("Digite o Checkpoint 1:"))
    chk2 = float(input("Digite o Checkpoint 2:"))
    chk3 = float(input("Digite o Checkpoint 3:"))
    # Verificando o Checkpoint de menor valor
    menor = chk1
    if chk2 < menor:
        menor = chk2
    if chk3 < menor:
        menor = chk3
    # Calculando a média dos Checkpoints
    mediaChk = (chk1 + chk2 + chk3 - menor) / 2
    # Leitura dos Sprints
    sprint1 = float(input("Digite o Sprint 1:"))
    sprint2 = float(input("Digite o Sprint 2:"))
    # Calculando a média dos Sprints
    mediaSprint = (sprint1 + sprint2) / 2
    # Leitura da prova semestral
    semestral = float(input("Digite prova semestral:"))
    # Ponderando os valores das médias
    pontosChk = mediaChk * 0.2
    pontosSprints = mediaSprint * 0.2
    pontosSemestral = semestral * 0.6
    # Cálculo da media do primeiro semestre
    mediaPrimeiroSemestre = (pontosChk + pontosSprints +
pontosSemestral)
    # Pontos obtidos no primeiro semestre
    pontosPrimeiroSemestre = mediaPrimeiroSemestre * 0.4

    # S E G U N D O   S E M E S T R E
    print("SEGUNDO SEMESTRE")
    # Leitura dos checkpoints
    chk1 = float(input("Digite o Checkpoint 1:"))
    chk2 = float(input("Digite o Checkpoint 2:"))
    chk3 = float(input("Digite o Checkpoint 3:"))
    # Verificando o Checkpoint de menor valor
    menor = chk1
    if chk2 < menor:
        menor = chk2
    if chk3 < menor:
        menor = chk3
    # Calculando a média dos Checkpoints
    mediaChk = (chk1 + chk2 + chk3 - menor) / 2
```

```
    # Leitura dos Sprints
    sprint1 = float(input("Digite o Sprint 1:"))
    sprint2 = float(input("Digite o Sprint 2:"))
    # Calculando a média dos Sprints
    mediaSprint = (sprint1 + sprint2) / 2
    # Leitura da prova semestral
    semestral = float(input("Digite prova semestral:"))
    # Ponderando os valores das médias
    pontosChk = mediaChk * 0.2
    pontosSprints = mediaSprint * 0.2
    pontosSemestral = semestral * 0.6
    # Cálculo da media do segundo semestre
    mediaSegundoSemestre = (pontosChk + pontosSprints +
pontosSemestral)
    # Pontos obtidos no segundo semestre
    pontosSegundoSemestre = mediaSegundoSemestre * 0.6

    # Exibiçao da media do primeiro semetre
    print (f"\nMédia do Primeiro Semestre:
{mediaPrimeiroSemestre:.1f}")
    # Exibiçao da media do segundo semetre
    print (f"Média do Segundo Semestre:
{mediaSegundoSemestre:.1f}")
    # Cálculo da média final
    mediaFinal = pontosPrimeiroSemestre +
pontosSegundoSemestre
    print (f"Média Final: {mediaFinal:.1f} - ", end="")
    if mediaFinal >= 6:
        contagemAprovados += 1
        print("Aprovado!")
    else:
        print("Não Aprovado!")
    aluno += 1

    # Condição de saída do laço "Faça-enquanto"
    if aluno > qtdAlunos:
        break

# Exibição da Quantidade de Alunos Aprovados
print("Quantidade de aprovados: ", contagemAprovados)
```

Código-fonte 24 – Contagem de aprovados: while True (Faça-Enquanto) em Python
Fonte: Elaborado por Edson de Oliveira (2022)

Simulamos o laço pós-condicional Faça-enquanto.

Por fim, mas não menos importante, vamos utilizar o laço contador *for*. Este é o laço mais apropriado para esta situação porque sabemos a quantidade de alunos e, consequentemente, a quantidade de voltas. Vejamos como fica:

```
contagemAprovados = 0
qtdAlunos = 40
aluno = 1
# Simulação do laço Faça-enquanto
for aluno in range(0, qtdAlunos, 1):
    # P R I M E I R O   S E M E S T R E
    print(f"------------------------------ ALUNO: {aluno +
1}")
    print("PRIMEIRO SEMESTRE")
    # Leitura dos checkpoints
    chk1 = float(input("Digite o Checkpoint 1:"))
    chk2 = float(input("Digite o Checkpoint 2:"))
    chk3 = float(input("Digite o Checkpoint 3:"))
    # Verificando o Checkpoint de menor valor
    menor = chk1
    if chk2 < menor:
        menor = chk2
    if chk3 < menor:
        menor = chk3
    # Calculando a média dos Checkpoints
    mediaChk = (chk1 + chk2 + chk3 - menor) / 2
    # Leitura dos Sprints
    sprint1 = float(input("Digite o Sprint 1:"))
    sprint2 = float(input("Digite o Sprint 2:"))
    # Calculando a média dos Sprints
    mediaSprint = (sprint1 + sprint2) / 2
    # Leitura da prova semestral
    semestral = float(input("Digite prova semestral:"))
    # Ponderando os valores das médias
    pontosChk = mediaChk * 0.2
    pontosSprints = mediaSprint * 0.2
    pontosSemestral = semestral * 0.6
    # Cálculo da media do primeiro semestre
    mediaPrimeiroSemestre = (pontosChk + pontosSprints +
pontosSemestral)
    # Pontos obtidos no primeiro semestre
    pontosPrimeiroSemestre = mediaPrimeiroSemestre * 0.4

    # S E G U N D O   S E M E S T R E
    print("SEGUNDO SEMESTRE")
    # Leitura dos checkpoints
    chk1 = float(input("Digite o Checkpoint 1:"))
    chk2 = float(input("Digite o Checkpoint 2:"))
    chk3 = float(input("Digite o Checkpoint 3:"))
    # Verificando o Checkpoint de menor valor
    menor = chk1
    if chk2 < menor:
        menor = chk2
    if chk3 < menor:
        menor = chk3
    # Calculando a média dos Checkpoints
```

```
    mediaChk = (chk1 + chk2 + chk3 - menor) / 2
    # Leitura dos Sprints
    sprint1 = float(input("Digite o Sprint 1:"))
    sprint2 = float(input("Digite o Sprint 2:"))
    # Calculando a média dos Sprints
    mediaSprint = (sprint1 + sprint2) / 2
    # Leitura da prova semestral
    semestral = float(input("Digite prova semestral:"))
    # Ponderando os valores das médias
    pontosChk = mediaChk * 0.2
    pontosSprints = mediaSprint * 0.2
    pontosSemestral = semestral * 0.6
    # Cálculo da media do segundo semestre
    mediaSegundoSemestre = (pontosChk + pontosSprints +
pontosSemestral)
    # Pontos obtidos no segundo semestre
    pontosSegundoSemestre = mediaSegundoSemestre * 0.6

    # Exibiçao da media do primeiro semetre
    print (f"\nMédia do Primeiro Semestre:
{mediaPrimeiroSemestre:.1f}")
    # Exibiçao da media do segundo semetre
    print (f"Média do Segundo Semestre:
{mediaSegundoSemestre:.1f}")
    # Cálculo da média final
    mediaFinal = pontosPrimeiroSemestre +
pontosSegundoSemestre
    print (f"Média Final: {mediaFinal:.1f} - ", end="")
    if mediaFinal >= 6:
        contagemAprovados += 1
        print("Aprovado!")
    else:
        print("Não Aprovado!")

# Exibição da Quantidade de Alunos Aprovados
print("Quantidade de aprovados: ", contagemAprovados)
```

Código-fonte 25 – Contagem de aprovados: while True (Faça-Enquanto) em Python
Fonte: Elaborado por Edson de Oliveira (2022)

Viram? Conseguimos utilizar os três laços para resolver o mesmo problema, mas lembrem-se: sempre há um laço mais apropriado.

2 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, você deve criar 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, obviamente, 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 se encontra 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. Mas, se o usuário digitar o número 6, por exemplo, a ação não será bem-sucedida, pois o número 6 não está na sequência.

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.