Estruturas de repetição

Para que uma máquina ou processo computacional tenha capacidade de realizar qualquer algoritmo (uma propriedade que chamamos de ser *Turing-completo*), é necessário ter capacidade **equivalente** a:

1. Ler e armazenar valores em espaços de memória;
2. Levar em conta os valores lidos para tomada de decisões;
3. Voltar a um estado anterior para realizar repetições.

No que vimos até agora em Python, a primeira característica equivale à criação e manipulação de variáveis, bem como entrada de dados. A segunda característica equivale às estruturas de execução condicional, isto é, `if/else`.

A terceira característica é o assunto desta parte da matéria. Veremos como realizar repetições na linguagem Python. Existem diversas formas (diretas ou indiretas) de causar repetições em um programa Python. Começaremos vendo a mais flexível e universal delas, o comando `while`. Também veremos o `for`, que será particularmente importante em assuntos futuros como as listas.

A versão atual deste documento está apenas com `while` e futuramente será atualizada para incluir o `for`.

Estrutura while

Uma forma simples de descrever o `while` é em comparação com o `if`. O código a seguir realiza o print caso a condição especificada seja atendida, isto é, se `x` for menor que 3:

```
x = int(input())
if x < 3:
    print('Ok')
```

A leitura em linguagem natural do código acima é: *Leia um número inteiro. Se ele for menor que 3, mostre a mensagem `Ok`*.

Agora, vamos substituir, no programa acima, a palavra `if` por `while`. Observe que, em inglês, "if" significa "se" ("se x for menor que 3"); por sua vez, a palavra "while" significa "enquanto".

```
x = int(input())
while x < 3:
    print('Ok')
```

O programa acima imprime `Ok` *enquanto* `x` for menor que 3. Perceba o problema: se o usuário digitar, por exemplo, o número 2, o programa irá imprimir `Ok` para sempre porque o `x` será menor que 3 para sempre.

Essa situação em que um programa fica em execução para sempre devido a um `while` cuja condição é sempre verdadeira chama-se **loop infinito**. Normalmente é uma situação indesejada, embora existam exceções. (Existem também outras formas de entrar em loop infinito, mas são equivalentes.)

Em vez disso, quase sempre queremos que a condição do `while` possa vir a se tornar falsa em algum momento, para que o programa consiga encerrar sua execução. Dizemos que a variável envolvida (no nosso exemplo, `x`) precisa ser **atualizada**. Por exemplo, podemos fazer assim:

```
x = int(input())
while x < 3:
    print('Ok')
    x = x + 1
```

Nesse exemplo, mesmo que `x` comece com um valor menor que 3, como ele está sendo **incrementado** (tem seu valor aumentado), em algum momento ele irá atingir um valor maior ou igual a 3 e a repetição irá terminar.

Um pouco de vocabulário

Um programa como o último exemplo é bastante típico da forma de usarmos o `while`:

1. Inicializamos uma variável;
2. O `while` terá uma condição que depende dessa variável;
3. Dentro do bloco de código do `while`, precisamos atualizar a variável para que a condição possa ficar falsa em algum momento, permitindo o término da repetição.

Assim, um esquema geral para uso do `while` poderia ser descrito assim:

```
<INICIALIZAÇÃO DA VARIÁVEL>
while <CONDIÇÃO QUE DEPENDE DA VARIÁVEL>:
    <ALGUMA TAREFA>
    <ATUALIZAÇÃO DA VARIÁVEL>
```

Note que deixamos a atualização por último. Isso não é uma regra mas é algo que geralmente torna a lógica mais limpa e evita alguns erros comuns. Usando esse vocabulário, vamos colocar comentários no último exemplo:

```
x = int(input())    # inicialização de x
while x < 3:        # a condição (que depende de x)
    print('Ok')     # tarefa
    x = x + 1       # atualização de x
```

Uma observação: esse esquema é relativamente simples. Existem casos em que a condição do `while` pode depender de diversas variáveis ou valores obtidos externamente.

Veja ainda que uma forma muito comum de atualização da variável é aumentar ou diminuir seu valor em uma quantidade fixa, como por exemplo `x = x + 1`, `n = n + 2`, `x = x - 1` etc.

Uma atualização que aumenta o valor da variável é chamada de **incremento**; uma que o diminui, é chamada de **decremento**.

Alguns outros termos

A própria estrutura de repetição, envolvendo a linha do `while` e todo o bloco de código condicionado a ele, é chamada de **laço de repetição** ou simplesmente **laço** ou **loop** (perceba que "loop" não é necessariamente "loop infinito").

Cada execução do bloco de código condicionado ao `while` é chamado de **um passo**, **uma repetição** ou **uma iteração** do laço.

> *Note que a palavra é "iteração" (sem N) e não "interação". A palavra "iteração" significa "uma ocorrência dentre possivelmente várias", e é a origem de palavras como "reiterar" (que significa repetir, recapitular).*

Resumo (vocabulário)

Podemos resumir a maior parte do vocabulário relevante a laços de repetição na seguinte tabela:

Termo	Significado	Exemplo no código
Laço (ou loop) de repetição	Toda a estrutura de repetição, desde o `while` (ou `for`, mais adiante) até a última linha dentro dele.	
Iteração, repetição, ou passo	Cada execução do bloco de código do laço	
Inicialização	A(s) linha(s) de código que cria(m) a variável e estabelecem seu valor inicial	`i = 1`
Condição	A expressão lógica que, enquanto tem valor True (verdadeiro), determina a continuidade do laço	`i < 10`
Atualização	A(s) linha(s) de código que muda(m) o valor da variável	`i = i + 1`

Exemplos com while

A seguir veremos uma seleção de exemplos usando `while`. Em cada um deles, tente identificar os elementos de vocabulário que citamos acima. Quais são as linhas correspondentes ao laço de repetição? Qual é a atualização da variável? Etc.

Exemplo 1

O exemplo abaixo é uma contagem regressiva de 10 até 1, ao final exibindo uma mensagem de Feliz Ano Novo:

```
contagem = 10
while contagem > 0:
```

```
    print(contagem)
    contagem = contagem - 1
print('Feliz ano novo!')
```

Exemplo 2

O próximo exemplo é um programa que imprime todos os valores ímpares entre 0 e 30.

```
numero = 1
while numero < 30:
    print(numero)
    numero = numero + 2
```

Perguntas para você pensar:

- Por que ele imprime apenas os ímpares?
- Como modificar para imprimir, em vez disso, apenas os pares?

Exemplos 3(a) e 3(b)

Imagine que queremos imprimir a tabuada do número 7. Os dois exemplos a seguir são maneiras diferentes de atingir este mesmo objetivo.

```
valor_atual = 7
while valor_atual <= 7*10:
    print(valor_atual)
    valor_atual = valor_atual + 7
```

```
multiplicador = 1
while multiplicador <= 10:
    print(7*multiplicador)
    multiplicador = multiplicador + 1
```

Observação: forma rápida de fazer incremento

Existe uma sintaxe curta para fazer uma linha do tipo

`variavel = variavel + incremento`

Podemos escrever simplesmente

`variavel += incremento`

e atingir o mesmo efeito.

Assim, por exemplo, a linha

```
valor_atual = valor_atual + 7
```

(do exemplo 3(a)) pode ser reescrita como

```
valor_atual += 7
```

Exemplo 4 (menu)

Uma aplicação recorrente de `while` é a implementação de menu. O exemplo abaixo é um menu bastante simples. São ofertadas duas opções: mostrar uma mensagem ou encerrar o programa. Enquanto o usuário não pedir o encerramento do programa, ele voltará a mostrar as opções e esperar pelo input do usuário para fazer sua escolha.

```
em_execucao = True

while em_execucao:
    print('Escolha uma opção:')
    print('A - mostrar mensagem')
    print('B - encerrar programa')

    opcao = input()

    if opcao == 'A':
        print('Bom dia!')
    elif opcao == 'B':
        em_execucao = False
    else:
        print('Opção inválida, tente de novo.')

print('Encerrando.')
```

Observe que, caso o usuário digite a entrada "B", o efeito será atualizarmos a variável `em_execucao` para o valor `False`. Isso fará com que a condição do `while` fique falsa (pois a condição é simplesmente o valor da variável `em_execucao`) e assim o programa possa terminar.

Veja outros exemplos de menu nos arquivos de aula disponíveis no Teams. Procure fazer o exercício da calculadora interativa.

Exemplo 5

Imagine que estamos dando suporte para um laboratório que está realizando uma cultura de bactérias para diagnóstico. Os processos laboratoriais conseguem fazer um diagnóstico com boa confiança se a cultura tiver uma população de pelo menos 100 mil indivíduos (isto é, 100 mil batérias).

(Na realidade, esse número pode variar dependendo da espécie de bactéria e quais as técnicas usadas para diagnóstico. É apenas um valor de referência.)

O programa abaixo auxilia a equipe do laboratório a calcular quantas horas esperar antes de prosseguir com os testes. Ele supõe que começamos a cultura com 10 indivíduos e calcula quantas horas devemos esperar para atingir pelo menos 100 mil indivíduos.

O programa supõe que a cada hora a população dobra.

```
população_alvo = 100000 # 100 mil

população_atual = 10    # começamos com 10 indivíduos
horas_passadas = 0
while população_atual < população_alvo:
    horas_passadas += 1
    população_atual = 2*população_atual

print('Após', horas_passadas, 'horas, atingimos população de',
população_atual)
print('O processo de diagnóstico pode prosseguir.')
```

Experimente executar o programa com outros valores de população inicial para verificar o impacto disso no tempo necessário para atingir pelo menos 100 mil indivíduos.