Exercício 4.8 Escreva um programa que leia dois números e que pergunte qual operação você deseja realizar. Você deve poder calcular a soma (+), subtração (-), multiplicação (*) e divisão (/). Exiba o resultado da operação solicitada.

Exercício 4.9 Escreva um programa para aprovar o empréstimo bancário para compra de uma casa. O programa deve perguntar o valor da casa a comprar, o salário e a quantidade de anos a pagar. O valor da prestação mensal não pode ser superior a 30% do salário. Calcule o valor da prestação como sendo o valor da casa a comprar dividido pelo número de meses a pagar.

Exercício 4.10 Escreva um programa que calcule o preço a pagar pelo fornecimento de energia elétrica. Pergunte a quantidade de kWh consumida e o tipo de instalação: R para residências, I para indústrias e C para comércios. Calcule o preço a pagar de acordo com a tabela a seguir.

Preço por tipo e faixa de consumo		
Tipo	Faixa (kWh)	Preço
Residencial	Até 500	R$ 0,40
	Acima de 500	R$ 0,65
Comercial	Até 1000	R$ 0,55
	Acima de 1000	R$ 0,60
Industrial	Até 5000	R$ 0,55
	Acima de 5000	R$ 0,60

CAPÍTULO 5

Repetições

Repetições representam a base de vários programas. São utilizadas para executar a mesma parte de um programa várias vezes, normalmente dependendo de uma condição. Por exemplo, para imprimir três números na tela, poderíamos escrever um programa como o apresentado na listagem 5.1.

▶ Listagem 5.1 – Imprimindo de 1 a 3

```
print(1)
print(2)
print(3)
```

Podemos imaginar que para imprimir três números, começando de 1 até o 3, devemos variar `print(x)`, onde `x` varia de 1 a 3. Vejamos outra solução para o problema na listagem 5.2.

▶ Listagem 5.2 – Imprimindo de 1 a 3 usando uma variável

```
x=1
print(x)
x=2
print(x)
x=3
print(x)
```

Outra solução seria incrementar o valor de `x` após cada `print`. Vejamos essa solução na listagem 5.3.

▶ **Listagem 5.3 – Imprimindo de 1 a 3 incrementando**

```
x=1
print(x)
x=x+1
print(x)
x=x+1
print(x)
```

Porém, se o objetivo fosse escrever 100 números, a solução não seria tão agradável, pois teríamos que escrever pelo menos 200 linhas! A estrutura de repetição aparece para nos auxiliar a resolver esse tipo de problema.

Uma das estruturas de repetição do Python é o **while**, que repete um bloco enquanto a condição for verdadeira. Seu formato é apresentado na listagem 5.4, onde `condição` é uma expressão lógica, e `bloco` representa as linhas de programa a repetir enquanto o resultado da condição for verdadeiro.

▶ **Listagem 5.4 – Formato da estrutura de repetição com while**

```
while <condição>:
   bloco
```

Para resolver o problema de escrever três números utilizando o **while**, escreveríamos um programa como o da listagem 5.5.

▶ **Listagem 5.5 – Imprimindo de 1 a 3 com while**

```
x=1 ❶
while x<=3: ❷
   print(x) ❸
   x = x + 1 ❹
```

A execução desse programa seria um pouco diferente do que vimos até agora. Primeiramente, ❶ seria executada inicializando a variável x com o valor 1. A linha ❷ seria uma combinação de estrutura condicional com estrutura de repetição. Podemos entender a condição do **while** da mesma forma que a condição de **if**. A diferença é que, se a condição for verdadeira, repetiremos as linhas ❸ e ❹ (bloco) enquanto a avaliação da condição for verdadeira.

Em ❸ teremos a impressão na tela propriamente dita, onde x é 1. Em ❹ temos que o valor de x é acrescentado de 1. Como x vale 1, `x + 1` valerá 2. Esse novo valor é então atribuído a x. A parte nova é que a execução não termina após ❹ que é o

fim do bloco, mas retorna para ❷. É esse retorno que faz a estrutura de repetição especial.

Agora, x vale 2 e `x <= 3` continua verdadeiro (**True**), logo, o bloco será executado outra vez. ❸ realizará a impressão do valor 2, e ❹ atualizará o valor de x para `x + 1`; nesse caso, 2 + 1 = 3. A execução volta novamente para a linha ❷.

A condição em ❷ é avaliada, e como x vale 3, e `x <= 3` continua verdadeira, fazendo com que as linhas ❸ e ❹ sejam executadas, exibindo 3 e atualizando o valor de x para 4 (3 + 1).

Nesse ponto, ❷, temos que x vale 4 e que a condição `x <= 3` resulta em Falso (**False**), terminando, assim, a repetição do bloco.

Exercício 5.1 Modifique o programa para exibir os números de 1 a 100.

Exercício 5.2 Modifique o programa para exibir os números de 50 a 100.

Exercício 5.3 Faça um programa para escrever a contagem regressiva do lançamento de um foguete. O programa deve imprimir 10, 9, 8, ..., 1, 0 e Fogo! na tela.

5.1 Contadores

O poder das estruturas de repetições é muito interessante, principalmente quando utilizamos condições com mais de uma variável. Imagine um problema onde deveríamos imprimir os números inteiros entre 1 e um valor digitado pelo usuário. Vamos modificar o programa da listagem 5.5 de forma que o último número a imprimir seja informado pelo usuário. O programa já modificado é apresentado na listagem 5.6.

▶ **Listagem 5.6 – Impressão de 1 até um número digitado pelo usuário**

```
fim=int(input("Digite o último número a imprimir:")) ❶
x = 1
while x <= fim: ❷
   print(x) ❸
   x = x + 1 ❹
```

Nesse caso, o programa imprimirá de 1 até o valor digitado em ❶. Em ❷ utilizamos a variável `fim` para representar o limite de nossa repetição.

Agora vamos analisar o que realizamos com a variável `x` dentro da repetição. Em ❸, o valor de `x` é simplesmente impresso. Em ❹ atualizamos o valor de `x` com `x + 1`, ou seja, com o próximo valor inteiro. Quando realizamos esse tipo de operação dentro de uma repetição estamos contando. Logo diremos que `x` é um contador. Um contador é uma variável utilizada para contar o número de ocorrências de um determinado evento; nesse caso, o número de repetições do **`while`**, que satisfaz às necessidades de nosso problema.

Experimente esse programa com vários valores, primeiro digitando 5, depois 500 e, por fim, 0 (zero). Provavelmente, 5 e 500 produzirão os resultados esperados, ou seja, a impressão de 1 até 5, ou de 1 até 500. Porém, quando digitamos zero, nada acontece, e o programa termina logo a seguir, sem impressão.

Analisando nosso programa quando a variável `fim` vale 0, ou seja, quando digitamos 0 em ❶, temos que a condição em ❷ é `x <= fim`. Como `x` é 1 e `fim` é 0, temos que `1 <= 0` é falso desde a primeira execução, fazendo com que o bloco a repetir não seja executado, uma vez que sua condição de entrada é falsa. O mesmo aconteceria na inserção de valores negativos.

Imagine que o problema agora seja um pouco diferente: imprimir apenas os números pares entre 0 e um número digitado pelo usuário, de forma bem similar ao problema anterior. Poderíamos resolver o problema com um **`if`** para testar se `x` é par ou ímpar antes de imprimir. Vale lembrar que um número é par quando é 0 ou múltiplo de 2. Quando é múltiplo de 2, temos que o resto da divisão desse número por 2 é 0, ou seja, o resultado é uma divisão exata, sem resto. Em Python, podemos escrever esse teste com `x % 2 == 0` (resto da divisão de x por 2 é igual a zero), alterando o programa anterior para o da listagem 5.7.

▶ Listagem 5.7 – Impressão de números pares de 0 até um número digitado pelo usuário

```
fim=int(input("Digite o último número a imprimir:"))
x = 0 ❶
while x <= fim:
   if x % 2 == 0: ❷
      print(x) ❸
   x = x + 1
```

Veja que, para começar a imprimir do 0, e não de 1, modificamos ❶. Um detalhe importante é que ❸ é um bloco dentro de **`if`** ❷, sendo para isso deslocado à direita. Execute o programa e verifique seu resultado.

Agora, finalmente, estamos resolvendo o problema, mas poderíamos resolvê-lo de forma ainda mais simples se adicionássemos 2 a `x` a cada repetição. Isso garantiria que `x` sempre fosse par. Vejamos o programa da listagem 5.8.

▶ Listagem 5.8 – Impressão de números pares de 0 até um número digitado pelo usuário, sem if

```
fim=int(input("Digite o último número a imprimir:"))
x = 0
while x <= fim:
   print(x)
   x = x + 2
```

Esses dois exemplos mostram que existe mais de uma solução para o problema, que podemos escrever programas diferentes e obter a mesma solução. Essas soluções podem ser às vezes mais complicadas, às vezes mais simples, mas ainda assim corretas.

Exercício 5.4 Modifique o programa anterior para imprimir de 1 até o número digitado pelo usuário, mas, dessa vez, apenas os números ímpares.

Exercício 5.5 Reescreva o programa anterior para escrever os 10 primeiros múltiplos de 3.

Vejamos outro tipo de problema. Imagine ter que imprimir a tabuada de adição de um número digitado pelo usuário. Essa tabuada deve ser impressa de 1 a 10, sendo `n` o número digitado pelo usuário. Teríamos, assim, n+1, n+2, ... n+10. Confira a solução na listagem 5.9.

▶ Listagem 5.9 – Tabuada simples

```
n = int(input("Tabuada de:"))
x = 1
while x <= 10:
   print(n+x)
   x=x+1
```

Execute o programa anterior e experimente diversos valores.

Exercício 5.6 Altere o programa anterior para exibir os resultados no mesmo formato de uma tabuada: 2x1 = 2, 2x2=4, ...

Exercício 5.7 Modifique o programa anterior de forma que o usuário também digite o início e o fim da tabuada, em vez de começar com 1 e 10.

Exercício 5.8 Escreva um programa que leia dois números. Imprima o resultado da multiplicação do primeiro pelo segundo. Utilize apenas os operadores de soma e subtração para calcular o resultado. Lembre-se de que podemos entender a multiplicação de dois números como somas sucessivas de um deles. Assim, 4 × 5 = 5 + 5 + 5 + 5 = 4 + 4 + 4 + 4 + 4.

Exercício 5.9 Escreva um programa que leia dois números. Imprima a divisão inteira do primeiro pelo segundo, assim como o resto da divisão. Utilize apenas os operadores de soma e subtração para calcular o resultado. Lembre-se de que podemos entender o quociente da divisão de dois números como a quantidade de vezes que podemos retirar o divisor do dividendo. Logo, 20 ÷ 4 = 5, uma vez que podemos subtrair 4 cinco vezes de 20.

Contadores também podem ser úteis quando usados com condições dentro dos programas. Vejamos um programa para corrigir um teste de múltipla escolha com três questões. A resposta da primeira é "b"; da segunda, "a"; e da terceira, "d". O programa da listagem 5.10 conta um ponto a cada resposta correta.

▶ Listagem 5.10 – Contagem de questões corretas

```
pontos = 0
questão = 1
while questão <= 3:
    resposta = input("Resposta da questão %d: " % questão)
    if questão == 1 and resposta == "b":
        pontos = pontos + 1
    if questão == 2 and resposta == "a":
        pontos = pontos + 1
    if questão == 3 and resposta == "d":
        pontos = pontos + 1
    questão +=1
print("O aluno fez %d ponto(s)" % pontos)
```


Execute o programa e digite todas as respostas corretas, depois tente com respostas diferentes. Veja que estamos verificando apenas respostas simples de uma só letra e que consideramos apenas letras minúsculas. Em Python, uma letra minúscula é diferente de uma maiúscula. Se você digitar "A" na segunda questão em vez de "a", o programa não considerará essa resposta correta. Uma solução para esse tipo de problema é utilizar o operador lógico `or` e verificar a resposta maiúscula e minúscula. Por exemplo, `questão == 1 and (resposta == "b" or resposta == "B")`.

Exercício 5.10 Modifique o programa da listagem 5.10 para que aceite respostas com letras maiúsculas e minúsculas em todas as questões.

Embora essa verificação resolva o problema, veremos que, se digitarmos um espaço em branco antes ou depois da resposta, ela também será considerada errada. Sempre que trabalharmos com strings, esse tipo de problema deve ser controlado. Veremos mais sobre o assunto no capítulo 7.

5.2 Acumuladores

Nem só de contadores precisamos. Em programas para calcular o total de uma soma, por exemplo, precisaremos de acumuladores. A diferença entre um contador e um acumulador é que nos contadores o valor adicionado é constante e, nos acumuladores, variável. Vejamos um programa que calcule a soma de 10 números na listagem 5.11. Nesse caso, `soma` ❶ é um acumulador e `n` ❷ é um contador.

▶ Listagem 5.11 – Soma de 10 números

```
n = 1
soma = 0
while n <= 10:
    x = int(input("Digite o %d número:"%n))
    soma = soma + x ❶
    n = n + 1 ❷
print("Soma: %d"%soma)
```

Podemos definir a média aritmética como a soma de vários números divididos pela quantidade de números somados. Assim, se somarmos três números, 4, 5 e 6, teríamos a média aritmética como (4+5+6) / 3, onde 3 é a quantidade de números. Se chamarmos o primeiro número de n1, o segundo de n2, e o terceiro de n3, teremos (n1 + n2 + n3) / 3.

Vejamos um programa que calcula a média de cinco números digitados pelo usuário na listagem 5.12. Se chamarmos o primeiro valor digitado de n1, o segundo de n2, e assim sucessivamente, teremos que:

$$média = (n1+n2+n3+n4+n5)/5 = \frac{n1 + n2 + n3 + n4 + n5}{5}$$

Em vez de utilizarmos cinco variáveis, vamos acumular os valores à medida que são lidos.

▶ Listagem 5.12 – Cálculo de média com acumulador

```
x = 1
soma = 0 ❶
while x <= 5:
    n = int(input("%d Digite o número:" % x))
    soma = soma + n ❷
    x = x + 1
print("Média: %5.2f" % (soma/5)) ❸
```

Nesse caso, temos x sendo um contador e n o valor digitado pelo usuário. A variável soma é criada em ❶ e inicializada com 0. Diferentemente de x, que recebe 1 a cada passagem, a variável soma, em ❷, é adicionada do valor digitado pelo usuário. Podemos dizer que o incremento de soma não é um valor constante, pois varia com o valor digitado pelo usuário. Podemos também dizer que soma acumula os valores de n a cada repetição. Logo, diremos que a variável soma é um acumulador.

Acumuladores são muito interessantes quando não sabemos ou não conseguimos obter o total da soma pela simples multiplicação de dois números. No caso do cálculo da média, o valor de n pode ser diferente cada vez que o usuário digitar um valor.

Exercício 5.11 Escreva um programa que pergunte o depósito inicial e a taxa de juros de uma poupança. Exiba os valores mês a mês para os 24 primeiros meses. Escreva o total ganho com juros no período.

Exercício 5.12 Altere o programa anterior de forma a perguntar também o valor depositado mensalmente. Esse valor será depositado no início de cada mês, e você deve considerá-lo para o cálculo de juros do mês seguinte.

Exercício 5.13 Escreva um programa que pergunte o valor inicial de uma dívida e o juro mensal. Pergunte também o valor mensal que será pago. Imprima o número de meses para que a dívida seja paga, o total pago e o total de juros pago.

5.3 Interrompendo a repetição

Embora muito útil, a estrutura **while** só verifica sua condição de parada no início de cada repetição. Dependendo do problema, a habilidade de terminar **while** dentro do bloco a repetir pode ser interessante.

A instrução **break** é utilizada para interromper a execução de **while** independentemente do valor atual de sua condição. Vejamos o exemplo da leitura de valores até que digitemos 0 (zero) no programa da listagem 5.13.

▶ Listagem 5.13 – Interrompendo a repetição

```
s=0
while True: ❶
    v=int(input("Digite um número a somar ou 0 para sair:"))
    if v==0:
        break ❷
    s = s+v ❸
print(s) ❹
```

Nesse exemplo, substituímos a condição do **while** por **True** em ❶. Dessa forma, o **while** executará para sempre, pois o valor de sua condição de parada (**True**) é constante. Em ❷ temos a instrução **break** sendo ativada dentro de um **if**, especificamente quando v é zero. Porém, enquanto v for diferente de zero, a repetição continuará a somar v a s em ❸. Quando v for igual a zero (0), teremos ❷ sendo executada, terminando a repetição e transferindo a execução para ❹, que, então, exibe o valor de s na tela.

Exercício 5.14 Escreva um programa que leia números inteiros do teclado. O programa deve ler os números até que o usuário digite 0 (zero). No final da execução, exiba a quantidade de números digitados, assim como a soma e a média aritmética.

Exercício 5.15 Escreva um programa para controlar uma pequena máquina registradora. Você deve solicitar ao usuário que digite o código do produto e a quantidade comprada. Utilize a tabela de códigos abaixo para obter o preço de cada produto:

Código	Preço
1	0,50
2	1,00
3	4,00
5	7,00
9	8,00

Seu programa deve exibir o total das compras depois que o usuário digitar 0. Qualquer outro código deve gerar a mensagem de erro "Código inválido".

Vejamos como exemplo um programa que leia um valor e que imprima a quantidade de cédulas necessárias para pagar esse mesmo valor, apresentado na listagem 5.14. Para simplificar, vamos trabalhar apenas com valores inteiros e com cédulas de R$ 50, R$ 20, R$ 10, R$ 5 e R$ 1.

Exercício 5.16 Execute o programa (Listagem 5.14) para os seguintes valores: 501, 745, 384, 2, 7 e 1.

Exercício 5.17 O que acontece se digitarmos 0 (zero) no valor a pagar?

Exercício 5.18 Modifique o programa para também trabalhar com notas de R$ 100.

Exercício 5.19 Modifique o programa para aceitar valores decimais, ou seja, também contar moedas de 0,01, 0,02, 0,05, 0,10 e 0,50.

Exercício 5.20 O que acontece se digitarmos 0,001 no programa anterior? Caso ele não funcione, altere-o de forma a corrigir o problema.

▶ **Listagem 5.14 – Contagem de cédulas**

```
valor=int(input("Digite o valor a pagar:"))
cédulas=0
atual=50
apagar=valor
while True:
    if atual<=apagar:
        apagar-=atual
        cédulas+=1
    else:
        print("%d cédula(s) de R$%d" % (cédulas, atual))
        if apagar == 0:
            break
        if atual == 50:
            atual = 20
        elif atual == 20:
            atual = 10
        elif atual == 10:
            atual = 5
        elif atual == 5:
            atual = 1
        cédulas = 0
```

5.4 Repetições aninhadas

Podemos combinar vários `while` de forma a obter resultados mais interessantes, como a repetição com incremento de duas variáveis. Imagine imprimir as tabuadas de multiplicação de 1 a 10. Vejamos como fazer isso, lendo a listagem do programa 5.15.

▶ **Listagem 5.15 – Impressão de tabuadas**

```
tabuada=1
while tabuada <= 10: ❶
    número = 1 ❷
    while número <= 10: ❸
        print("%d x %d = %d" % (tabuada, número, tabuada * número))
        número+=1 ❹
    tabuada+=1 ❺
```

Em ❶ temos nosso primeiro **while**, criado para repetir seu bloco enquanto o valor de tabuada for menor ou igual a 10. Em ❷ temos a inicialização da variável número dentro do primeiro **while**. Isso é importante porque precisamos voltar a multiplicar por 1 a cada novo valor da variável tabuada. Finalmente, em ❸ temos o segundo **while** com a condição de parada número <= 10. Esse **while** executará suas repetições dentro do primeiro, ou seja, o ponto de execução passa de ❹ para ❸ enquanto a condição for verdadeira. Veja que em ❹ utilizamos o operador += para representar número = número + 1. Quando número valer 11, a condição em ❸ resultará falsa, e a execução do programa continuará a partir da linha ❺. Em ❺ incrementamos o valor de tabuada e voltamos a ❶, onde será verificada a condição do primeiro **while**. Como resulta verdadeiro, voltaremos a executar ❷, reinicializando a variável número com o valor 1. ❷ é muito importante para que possamos novamente executar o segundo **while**, responsável por imprimir a tabuada na tela.

Vejamos o mesmo problema, mas sem utilizar repetições aninhadas, como apresenta a listagem 5.16.

▶ **Listagem 5.16 – Impressão de tabuadas sem repetições aninhadas**

```
tabuada=1
número=1
while tabuada <= 10:
    print("%d x %d = %d" % (tabuada, número, tabuada * número))
    número+=1
    if número == 11:
        número = 1
        tabuada+=1
```

Exercício 5.21 Reescreva o programa da listagem 5.14 de forma a continuar executando até que o valor digitado seja 0. Utilize repetições aninhadas.

Exercício 5.22 Escreva um programa que exiba uma lista de opções (menu): adição, subtração, divisão, multiplicação e sair. Imprima a tabuada da operação escolhida. Repita até que a opção saída seja escolhida.

Exercício 5.23 Escreva um programa que leia um número e verifique se é ou não um número primo. Para fazer essa verificação, calcule o resto da divisão do número por 2 e depois por todos os números ímpares até o número lido. Se o resto de uma dessas divisões for igual a zero, o número não é primo. Observe que 0 e 1 não são primos e que 2 é o único número primo que é par.

Exercício 5.24 Modifique o programa anterior de forma a ler um número n. Imprima os n primeiros números primos.

Exercício 5.25 Escreva um programa que calcule a raiz quadrada de um número. Utilize o método de Newton para obter um resultado aproximado. Sendo n o número a obter a raiz quadrada, considere a base b=2. Calcule p usando a fórmula p=(b+(n/b))/2. Agora, calcule o quadrado de p. A cada passo, faça b=p e recalcule p usando a fórmula apresentada. Pare quando a diferença absoluta entre n e o quadrado de p for menor que 0,0001.

Exercício 5.26 Escreva um programa que calcule o resto da divisão inteira entre dois números. Utilize apenas as operações de soma e subtração para calcular o resultado.

Exercício 5.27 Escreva um programa que verifique se um número é palíndromo. Um número é palíndromo se continua o mesmo caso seus dígitos sejam invertidos. Exemplos: 454, 10501