▶ **Listagem 3.25 – Erro de conversão**

```
Digite seu nome: Minduim
Digite sua idade: abc
Traceback (most recent call last):
  File "input/input2.py", line 2, in <module>
    idade = int(input("Digite sua idade: "))
ValueError: invalid literal for int() with base 10: 'abc'
```

A listagem 3.26 mostra outro erro de conversão, mas, dessa vez, durante a conversão para número decimal, usando a função `float`. A entrada de dados é um pouco rústica, parando em caso de erro. Mais adiante, aprenderemos sobre exceções em Python e como tratar esse tipo de erro. Por enquanto, basta saber que não estamos validando a entrada e que nossos programas ainda são frágeis.

▶ **Listagem 3.26 – Erro de conversão: letras no lugar de números**

```
Digite seu nome: Juanito
Digite sua idade: 31
Digite o saldo da sua conta bancária: abc
Traceback (most recent call last):
  File "input/input2.py", line 3, in <module>
    saldo = float(input("Digite o saldo da sua conta bancária: "))
ValueError: could not convert string to float: abc
```

▶ **Listagem 3.27 – Erro de conversão: vírgula no lugar de ponto**

```
Digite seu nome: Mary
Digite sua idade: 25
Digite o saldo da sua conta bancária: 17,4
Traceback (most recent call last):
  File "input/input2.py", line 3, in <module>
    saldo = float(input("Digite o saldo da sua conta bancária: "))
ValueError: invalid literal for float(): 17,4
```

O erro mostrado na listagem 3.27 é muito comum em países onde se usa a vírgula e não o ponto como separador entre a parte inteira e fracionária de um número. Em Python, você deve sempre digitar valores decimais usando o ponto, e não a vírgula como em português. Assim, 17,4 é um valor inválido, pois deveria ter sido digitado como 17.4. Existem recursos em Python para resolver esse tipo de problema, mas ainda é cedo para abordarmos o assunto.

CAPÍTULO 4

Condições

Executar ou não executar? Eis a questão...

Nem sempre todas as linhas dos programas serão executadas. Muitas vezes, será mais interessante decidir que partes do programa devem ser executadas com base no resultado de uma condição. A base dessas decisões consistirá em expressões lógicas que permitam representar escolhas em programas.

4.1 if

As condições servem para selecionar quando uma parte do programa deve ser ativada e quando deve ser simplesmente ignorada. Em Python, a estrutura de decisão é o **`if`**. Seu formato é apresentado na listagem 4.1.

▶ **Listagem 4.1 – Formato da estrutura de condicional if**

```
if <condição>:
   bloco verdadeiro
```

O `if` nada mais é que o nosso "se". Poderemos então entendê-lo em português da seguinte forma: se a condição for verdadeira, faça alguma coisa.

Vejamos um exemplo: ler dois valores e imprimir o maior deles, apresentado na listagem 4.2.

▶ **Listagem 4.2 – Condições**

```
a = int(input("Primeiro valor: "))
b = int(input("Segundo valor: "))
```

```
if a > b: ❶
   print ("O primeiro número é o maior!") ❷
if b > a: ❸
   print ("O segundo número é o maior!") ❹
```

Em ❶, temos a condição `a > b`. Essa expressão será avaliada, e, se o seu resultado for verdadeiro, a linha ❷ será executada. Se for falso, a linha ❷ será ignorada. O mesmo acontece para a condição `b > a` da linha ❸. Se o seu resultado for verdadeiro, a linha ❹ será executada. Se for falso, ignorada.

A sequência de execução do programa é alterada de acordo com os valores digitados como o primeiro e segundo valores. Digite o programa da listagem 4.2 e execute-o duas vezes. Na primeira vez, digite um valor maior primeiro e um menor em segundo. Na segunda vez, inverta esses valores e verifique se a mensagem na tela também mudou.

Quando o primeiro valor é maior que o segundo, temos as seguintes linhas sendo executadas: ❶, ❷, ❸. Quando o primeiro valor é menor que o segundo, temos outra sequência: ❶, ❸, ❹. É importante entender que a linha com a condição em si é executada mesmo se o resultado da expressão for falso.

As linhas ❶ e ❸ foram terminadas com o símbolo dois pontos (:). Quando isso acontece, temos o anúncio de um bloco de linhas a seguir. Em Python, um bloco é representado deslocando-se o início da linha para a direita. O bloco continua até a primeira linha com deslocamento diferente.

Observe que começamos a escrever a linha ❷ alguns caracteres mais à direita da linha anterior ❶ que iniciou o bloco. Como a linha ❸ foi escrita mais à esquerda, dizemos que o bloco da linha ❷ foi terminado.

TRÍVIA

Python é uma das poucas linguagens de programação que utiliza o deslocamento do texto à direita (recuo) para marcar o início e o fim de um bloco. Outras linguagens contam com palavras especiais para isso, como **BEGIN** e **END**, em Pascal; ou as famosas chaves ({ e }), em C e Java.

Exercício 4.1 Analise o programa da listagem 4.2. Responda o que acontece se o primeiro e o segundo valores forem iguais? Explique.

Vejamos outro exemplo, onde solicitaremos a idade do carro do usuário e, em seguida, escreveremos novo se o carro tiver menos de três anos; ou velho, em caso contrário.

▶ Listagem 4.3 – Carro novo ou velho, dependendo da idade

```
idade = int(input("Digite a idade do seu carro: "))
if idade <= 3:
   print("Seu carro é novo") ❶
if idade > 3:
   print("Seu carro é velho") ❷
```

Execute o programa da listagem 4.3 e verifique o que aparece na tela. Você pode executá-lo várias vezes com as seguintes idades: 1, 3 e 5. A primeira condição é `idade <= 3`. Essa condição decide se a linha com função **print** ❶ será ou não executada. Como é uma condição simples, podemos entender que só exibiremos a mensagem do carro novo para as idades 0, 1, 2 e 3. A segunda condição, `idade > 3`, é o inverso da primeira. Se você observar de perto, não há um só número que torne ambas verdadeiras ao mesmo tempo. A segunda decisão é responsável por decidir a impressão da mensagem do carro velho ❷.

Embora óbvio que um carro não poderia ter valores negativos como idade, o programa não trata desse problema. Vamos alterá-lo mais adiante para verificar valores inválidos.

Exercício 4.2 Escreva um programa que pergunte a velocidade do carro de um usuário. Caso ultrapasse 80 km/h, exiba uma mensagem dizendo que o usuário foi multado. Nesse caso, exiba o valor da multa, cobrando R$ 5 por km acima de 80 km/h.

Um bloco de linhas em Python pode ter mais de uma linha, embora o último exemplo mostre apenas dois blocos com uma linha em cada. Se você precisar de duas ou mais no mesmo bloco, escreva essas linhas na mesma direção ou na mesma coluna da primeira linha do bloco. Isso basta para representá-lo.

Um problema comum é quando temos que pagar Imposto de Renda. Normalmente, pagamos o Imposto de Renda por faixa de salário. Imagine que para salários menores que R$ 1.000,00 não teríamos imposto a pagar, ou seja, alíquota 0%. Para salários entre R$ 1.000,00 e R$ 3.000,00 pagaríamos 20%. Acima desses valores, a alíquota seria de 35%. Esse problema se pareceria muito com o anterior, salvo

se o imposto não fosse cobrado diferentemente para cada faixa, ou seja, quem ganha R$ 4.000,00 tem os primeiros R$ 1.000,00 isentos de imposto; com o montante entre R$ 1.000,00 e R$ 3.000,00 pagando 20%, e o restante pagando os 35%. Vejamos a solução na listagem do programa 4.4.

► Listagem 4.4 – Cálculo do Imposto de Renda

```
salário=float(input("Digite o salário para cálculo do imposto: "))
base = salário ❶
imposto = 0
if base > 3000: ❷
   imposto = imposto + ((base - 3000) * 0.35) ❸
   base = 3000 ❹
if base > 1000: ❺
   imposto = imposto + ((base - 1000) * 0.20) ❻
print("Salário: R$%6.2f Imposto a pagar: R$%6.2f" % (salário, imposto))
```

O programa da listagem 4.4 é bem interessante. Tente executá-lo algumas vezes e compare o valor impresso com o valor calculado por você. Rastreie o programa e tente entender o que ele faz antes de ler o parágrafo seguinte. Verifique o que acontece para salários de R$ 500,00, R$ 1.000,00 e R$ 1.500,00.

Em ❶ temos a variável `base` recebendo uma cópia de `salário`. Isso é necessário porque, quando atribuímos um novo valor para uma variável, o valor anterior é substituído (e perdido se não o guardarmos em outro lugar). Como vamos utilizar o valor do salário digitado para exibi-lo na tela, não podemos perdê-lo; por isso, a necessidade de uma variável auxiliar chamada aqui de `base`.

Em ❷ verificamos se a `base` é maior que R$ 3.000,00. Se verdadeiro, executamos as linhas ❸ e ❹. Em ❸, calculamos 35% do valor superior a R$ 3.000,00. O resultado é armazenado na variável `imposto`. Como essa variável contém o valor a pagar para essa quantia, atualizaremos o valor de base para R$ 3.000,00 ❹, pois o que ultrapassa esse valor já foi tarifado.

Em ❺ verificamos se o valor de base é maior que R$ 1.000,00, calculando 20% de imposto em ❻, caso verdadeiro.

Vejamos o rastreamento para um salário de R$ 500,00:

salário	base	imposto
500	500	0

Para um salário de R$ 1.500,00:

salário	base	imposto
1500	1500	0
		100

Para um salário de R$ 3.000,00:

salário	base	imposto
3000	3000	0
		400

Para um salário de R$ 5.000,00:

salário	base	imposto
5000	~~5000~~	0
	3000	~~700~~
		1100

Exercício 4.3 Escreva um programa que leia três números e que imprima o maior e o menor.

Exercício 4.4 Escreva um programa que pergunte o salário do funcionário e calcule o valor do aumento. Para salários superiores a R$ 1.250,00, calcule um aumento de 10%. Para os inferiores ou iguais, de 15%.

4.2 else

Quando há problemas, como a mensagem do carro velho (Listagem 4.3), onde a segunda condição é simplesmente o inverso da primeira, podemos usar outra forma de **`if`** para simplificar os programas. Essa forma é a cláusula **`else`** para especificar o que fazer caso o resultado da avaliação da condição seja falso, sem precisarmos de um novo **`if`**. Vejamos como ficaria o programa reescrito para usar **`else`** na listagem 4.5.

► Listagem 4.5 – Carro novo ou velho, dependendo da idade com else

```
idade = int(input("Digite a idade de seu carro: "))
if idade <= 3:
   print("Seu carro é novo")
else: ❶
   print("Seu carro é velho") ❷
```

Veja que em ❶ utilizamos ":" após **else**. Isso é necessário porque **else** inicia um bloco, da mesma forma que **if**. É importante notar que devemos escrever **else** na mesma coluna do **if**, ou seja, com o mesmo recuo. Assim, o interpretador reconhece que **else** se refere a um determinado **if**. Você obterá um erro caso não alinhe essas duas estruturas na mesma coluna.

A vantagem de usar **else** é deixar os programas mais claros, uma vez que podemos expressar o que fazer caso a condição especificada em **if** seja falsa. A linha ❷ só é executada se a condição `idade <= 3` for falsa.

Exercício 4.5 Execute o programa (Listagem 4.5) e experimente alguns valores. Verifique se os resultados foram os mesmos do programa anterior (Listagem 4.3).

Exercício 4.6 Escreva um programa que pergunte a distância que um passageiro deseja percorrer em km. Calcule o preço da passagem, cobrando R$ 0,50 por km para viagens de até de 200 km, e R$ 0,45 para viagens mais longas.

4.3 Estruturas aninhadas

Nem sempre nossos programas serão tão simples. Muitas vezes, precisaremos aninhar vários **if** para obter o comportamento desejado do programa. Aninhar, nesse caso, é utilizar um **if** dentro de outro.

Vejamos o exemplo de calcular a conta de um telefone celular da empresa Tchau. Os planos da empresa Tchau são bem interessantes e oferecem preços diferenciados de acordo com a quantidade de minutos usados por mês. Abaixo de 200 minutos, a empresa cobra R$ 0,20 por minuto. Entre 200 e 400 minutos, o preço é de R$ 0,18. Acima de 400 minutos, o preço por minuto é de R$ 0,15. O programa da listagem 4.6 resolve esse problema.

► Listagem 4.6 – Conta de telefone com três faixas de preço

```
minutos=int(input("Quantos minutos você utilizou este mês:"))
if minutos < 200: ❶
   preço = 0.20 ❷
else:
   if minutos < 400: ❸
      preço = 0.18 ❹
   else: ❺
      preço = 0.15 ❻
print("Você vai pagar este mês: R$%6.2f" % (minutos * preço))
```

Em ❶, temos a primeira condição: `minutos < 200`. Se a quantidade de minutos for menor que 200, atribuímos 0,20 ao preço em ❷. Até aqui, nada de novo. Observe que **if** de ❸ está dentro de **else** da linha anterior: dizemos que está aninhado dentro de **else**. A condição de ❸, `minutos < 400`, decide se vamos executar a linha de ❹ ou a de ❻. Observe que **else** de ❺ está alinhado com **if** de ❸. No final, calculamos e imprimimos o preço na tela. Lembre-se de que o alinhamento do texto é muito importante em Python.

Vejamos, por exemplo a situação em que cinco categorias são necessárias. Façamos um programa que leia a categoria de um produto e determine o preço pela tabela 4.1.

Tabela 4.1 – Categorias de produto e preço

Categoria	Preço
1	10,00
2	18,00
3	23,00
4	26,00
5	31,00

À esquerda da listagem 4.7, você encontrará os números de linha do programa, numeradas de 1 a 19. Esses números servem apenas para ajudar o entendimento da explicação a seguir: lembre-se de não digitá-los.

▶ **Listagem 4.7 – Categoria x preço**

```
1  categoria = int(input("Digite a categoria do produto:"))
2  if categoria == 1:
3  preço = 10
4  else:
5      if categoria == 2:
6          preço = 18
7      else:
8          if categoria == 3:
9              preço = 23
10         else:
11             if categoria == 4:
12                 preço = 26
13             else:
14                 if categoria == 5:
15                     preço = 31
16                 else:
17                     print("Categoria inválida, digite um valor entre 1 e 5!")
18                     preço = 0
19 print("O preço do produto é: R$%6.2f" % preço)
```

Observe que o alinhamento se tornou um grande problema, uma vez que tivemos que deslocar à direita a cada **else**.

No programa da listagem 4.7, introduzimos o conceito de validação da entrada. Dessa vez, se o usuário digitar um valor inválido, receberá uma mensagem de erro na tela. Nada muito prático ou bonito.

Vejamos a execução das linhas dependendo da categoria digitada na tabela 4.2.

Tabela 4.2 – Linhas executadas

Categoria	Linhas executadas
1	1,2,3,19
2	1,2,4,5,6,19
3	1,2,4,5,7,8,9,19
4	1,2,4,5,7,8,10,11,12,19
5	1,2,4,5,7,8,10,11,13,14,15,19
outras	1,2,4,5,7,8,10,11,13,14,16,17,18,19

Quando lermos um programa com estruturas aninhadas, devemos prestar muita atenção para visualizar corretamente os blocos. Observe como o alinhamento é importante.

Exercício 4.7 Rastreie o programa da listagem 4.7. Compare seu resultado ao apresentado na tabela 4.2.

4.4 elif

Python apresenta uma solução muito interessante ao problema de múltiplos **if**s aninhados. A cláusula **elif** substitui um par **else if**, mas sem criar outro nível de estrutura, evitando problemas de deslocamentos desnecessários à direita.

Vamos revisitar o problema da listagem 4.7, dessa vez usando **elif**. Veja o resultado no programa da listagem 4.8.

▶ **Listagem 4.8 – Categoria x preço, usando elif**

```
categoria = int(input("Digite a categoria do produto:"))
if categoria == 1:
    preço = 10
elif categoria == 2:
    preço = 18
elif categoria == 3:
    preço = 23
elif categoria == 4:
    preço = 26
elif categoria == 5:
    preço = 31
else:
    print("Categoria inválida, digite um valor entre 1 e 5!")
    preço = 0
print("O preço do produto é: R$%6.2f" % preço)
```

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.