

Exercício 2.3 Faça um programa que exiba seu nome na tela.

Exercício 2.4 Escreva um programa que exiba o resultado de $2a \times 3b$, onde a vale 3 e b vale 5.

Exercício 2.5 Modifique o primeiro programa, listagem 2.7, de forma a calcular a soma de três variáveis.

Exercício 2.6 Modifique o programa da listagem 2.11, de forma que ele calcule um aumento de 15% para um salário de R\$ 750.

CAPÍTULO 3

Variáveis e entrada de dados

O capítulo anterior apresentou o conceito de variáveis, mas há mais por descobrir. Já sabemos que variáveis têm nomes que permitem acessar os valores dessas variáveis em outras partes do programa. Neste capítulo, vamos ampliar nosso conhecimento sobre variáveis, estudando novas operações e novos tipos de dados.

3.1 Nomes de variáveis

Em Python, nomes de variáveis devem iniciar obrigatoriamente com uma letra, mas podem conter números e o símbolo sublinha (_). Vejamos exemplos de nomes válidos e inválidos em Python na tabela 3.1.

Tabela 3.1 – Exemplo de nomes válidos e inválidos para variáveis

Nome	Válido	Comentários
a1	Sim	Embora contenha um número, o nome a1 inicia com letra.
velocidade	Sim	Nome formado por letras.
velocidade90	Sim	Nome formado por letras e números, mas iniciado por letra.
salário_médio	Sim	O símbolo sublinha (<u>_</u>) é permitido e facilita a leitura de nomes grandes.
salário médio	Não	Nomes de variáveis não podem conter espaços em branco.
<u>_b</u>	Sim	O sublinha (<u>_</u>) é aceito em nomes de variáveis, mesmo no início.
1a	Não	Nomes de variáveis não podem começar com números.

A versão 3 da linguagem Python permite a utilização de acentos em nomes de variáveis, pois, por padrão, os programas são interpretados utilizando-se um conjunto de caracteres chamado UTF-8 (<http://pt.wikipedia.org/wiki/Utf-8>), capaz de representar praticamente todas as letras dos alfabetos conhecidos.

Variáveis têm outras propriedades além de nome e conteúdo. Uma delas é conhecida como tipo e define a natureza dos dados que a variável armazena. Python tem vários tipos de dados, mas os mais comuns são números inteiros, números de ponto flutuante e strings. Além de poder armazenar números e letras, as variáveis em Python também armazenam valores como verdadeiro ou falso. Dizemos que essas variáveis são do tipo lógico. Veremos mais sobre variáveis do tipo lógico na seção 3.3.

TRÍVIA

A maior parte das linguagens de programação foi desenvolvida nos Estados Unidos, ou considerando apenas nomes escritos na língua inglesa como nomes válidos. Por isso, acentos e letras consideradas especiais não são aceitos como nomes válidos na maior parte das linguagens de programação. Essa restrição é um problema não só para falantes do português, mas de muitas outras línguas que utilizam acentos ou mesmo outros alfabetos. O padrão americano é baseado no conjunto de caracteres ASCII^(*), desenvolvido na década de 1960. Com a globalização da economia e o advento da internet, as aplicações mais modernas são escritas para trabalhar com um conjunto de caracteres dito universal, chamado Unicode^(**).

(*) <http://pt.wikipedia.org/wiki/Ascii>

(**) <http://pt.wikipedia.org/wiki/Unicode>

3.2 Variáveis numéricas

Dizemos que uma variável é numérica quando armazena números inteiros ou de ponto flutuante.

Os números inteiros são aqueles sem parte decimal, como 1, 0, -5, 550, -47, 30000.

Números de ponto flutuante ou decimais são aqueles com parte decimal, como 1.0, 5.478, 10.478, 30000.4. Observe que 1.0, mesmo tendo zero na parte decimal, é um número de ponto flutuante.

Em Python, e na maioria das linguagens de programação, utilizamos o ponto, e não a vírgula, como separador entre a parte inteira e fracionária de um número. Essa é outra herança da língua inglesa. Observe também que não utilizamos nada como separador de milhar. Exemplo: 1.000.000 (um milhão) é escrito 1000000.

Exercício 3.1 Complete a tabela a seguir, marcando inteiro ou ponto flutuante dependendo do número apresentado.

Número	Tipo numérico
5	<input type="radio"/> inteiro <input type="radio"/> ponto flutuante
5.0	<input type="radio"/> inteiro <input type="radio"/> ponto flutuante
4.3	<input type="radio"/> inteiro <input type="radio"/> ponto flutuante
-2	<input type="radio"/> inteiro <input type="radio"/> ponto flutuante
100	<input type="radio"/> inteiro <input type="radio"/> ponto flutuante
1.333	<input type="radio"/> inteiro <input type="radio"/> ponto flutuante

3.2.1 Representação de valores numéricos

Internamente, todos os números são representados utilizando o sistema binário, ou seja, de base 2. Esse sistema permite apenas os dígitos 0 e 1. Para representar números maiores, combinamos vários dígitos, exatamente como fazemos com o sistema decimal, ou de base 10, que utilizamos.

Vejamos primeiro como isso funciona na base 10:

$$\begin{aligned}
 531 &= 5 \times 10^2 + 3 \times 10^1 + 1 \times 10^0 \\
 &= 5 \times 100 + 3 \times 10 + 1 \times 1 \\
 &= 500 + 30 + 1 \\
 &= 531
 \end{aligned}$$

Multiplicamos cada dígito pela base elevada a um expoente igual ao número de casas à direita do dígito em questão. Como em 531 o 5 tem 2 dígitos à direita, multiplicamos 5×10^2 . Para o 3, temos apenas outro dígito à direita, logo, 3×10^1 . Finalmente, para o 1, sem dígitos à direita, temos 1×10^0 . Somando esses componentes, temos o número 531. Fazemos isso tão rápido que é natural ou automático pensarmos desse jeito.

A mudança para o sistema binário segue o mesmo processo, mas a base agora é 2, e não 10. Assim, 1010 em binário representa:

$$\begin{aligned}
 1010 &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\
 &= 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1 \\
 &= 8 + 2 \\
 &= 10
 \end{aligned}$$

A utilização do sistema binário é transparente em Python, ou seja, se você não solicitar explicitamente que esse sistema seja usado, tudo será apresentado na base 10 utilizada no dia a dia. A importância da noção de diferença de base é importante, pois ela explica os limites da representação. O limite de representação é o valor mínimo e máximo que pode ser representado em uma variável numérica. Esse limite é causado pela quantidade de dígitos que foram reservados para armazenar o número em questão. Vejamos como funciona na base 10.

Se você tem apenas 5 dígitos para representar um número, o maior número é 99999. E o menor seria (-99999). O mesmo acontece no sistema binário, sendo que lá reservamos um dígito para registrar os sinais de positivo e negativo.

Para números inteiros, Python utiliza um sistema de precisão ilimitada que permite a representação de números muito grandes. É como se você sempre pudesse escrever novos dígitos à medida que for necessário. Você pode calcular, em Python, valores como $2^{1000000}$ ($2 ** 1000000$) sem problemas de representação, mesmo quando o resultado é um número de 301030 dígitos.

Em ponto flutuante, temos limite e problemas de representação. Um número decimal é representado em ponto flutuante utilizando-se uma mantissa e um expoente ($sinal \times mantissa \times base^{expoente}$). Tanto a mantissa quanto o expoente têm um número de dígitos máximos que limita os números que podem ser representados. Você não precisa se preocupar com isso no momento, pois esses valores são bem grandes e você não terá problemas na maioria de seus programas. Você pode obter mais informações acessando http://pt.wikipedia.org/wiki/Ponto_flutuante.

A versão 3.1.2 do Python, rodando em Mac OS X, tem como limites $2.2250738585072014 \times 10^{-308}$ e $1.7976931348623157 \times 10^{308}$, suficientemente grandes e pequenos para quase qualquer tipo de aplicação. É possível encontrar problemas de representação em função de como os números decimais são convertidos em números de ponto flutuante. Esses problemas são bem conhecidos e afetam todas as linguagens de programação, não sendo um problema específico do Python.

Vejamos um exemplo: o número 0.1 não tem nada de especial no sistema decimal, mas é uma dízima periódica no sistema binário. Você não precisa se preocupar com esses detalhes agora, mas pode investigá-los mais tarde quando precisar (normalmente cursos de computação apresentam uma disciplina, chamada cálculo numérico, para abordar esses tipos de problemas). Digite no interpretador $3 * 0.1$

Você deve ter obtido como resultado 0.3000000000000004, e não 0.3, como esperado. Não se assuste: não é um problema de seu computador, mas de representação. Se for necessário, durante seus estudos, cálculos mais precisos, ou se os resultados em ponto flutuante não satisfizerem os requisitos de precisão esperados,

verifique os módulos `decimals` e `fractions`. A documentação do Python traz uma página específica sobre esse tipo de problema: <http://docs.python.org/py3k/tutorial/floatingpoint.html>.

3.3 Variáveis do tipo Lógico

Muitas vezes, queremos armazenar um conteúdo simples: verdadeiro ou falso em uma variável. Nesse caso, utilizaremos um tipo de variável chamado tipo lógico ou booleano. Em Python, escreveremos `True` para verdadeiro e `False` para falso (Listagem 3.1). Observe que o T e o F são escritos com letras maiúsculas.

► Listagem 3.1 – Exemplo de variáveis do tipo lógico

```
resultado = True
aprovado = False
```

3.3.1 Operadores relacionais

Para realizarmos comparações lógicas, utilizaremos operadores relacionais. A lista de operadores relacionais suportados em Python é apresentada na tabela 3.2.

Tabela 3.2 – Operadores relacionais

Operador	Operação	Símbolo matemático
<code>==</code>	igualdade	<code>=</code>
<code>></code>	maior que	<code>></code>
<code><</code>	menor que	<code><</code>
<code>!=</code>	diferente	<code>≠</code>
<code>>=</code>	maior ou igual	<code>≥</code>
<code><=</code>	menor ou igual	<code>≤</code>

O resultado de uma comparação é um valor do tipo lógico, ou seja, `True` (verdadeiro) ou `False` (falso). Utilizaremos o verbo “avaliar” para indicar a resolução de uma expressão.

► Listagem 3.2 – Exemplo de uso de operadores relacionais

```
>>> a = 1      # a recebe 1
>>> b = 5      # b recebe 5
```

```

>>> c = 2      # c recebe 2
>>> d = 1      # d recebe 1
>>> a == b     # a é igual a b ?
False
>>> b > a     # b é maior que a?
True
>>> a < b     # a é menor que b?
True
>>> a == d     # a é igual a d?
True
>>> b >= a    # b é maior ou igual a a?
True
>>> c <= b    # c é menor ou igual a b?
True
>>> d != a    # d é diferente de a?
False
>>> d != b    # d é diferente de b?
True

```

Esses operadores são utilizados como na matemática. Especial atenção deve ser dada aos operadores `>=` e `<=`. O resultado desses operadores é realmente maior ou igual e menor ou igual, ou seja, `5 >= 5` é verdadeiro, assim como `5 <= 5`.

Observe que, na listagem 3.2, utilizamos o símbolo de cerquilha (#) para escrever comentários na linha de comando. Todo texto à direita do cerquilha é ignorado pelo interpretador Python, ou seja, você pode escrever o que quiser. Leia novamente a listagem 3.2 e veja como é mais fácil entender cada linha quando a comentamos. Comentários não são obrigatórios, mas são muito importantes. Você pode e deve utilizá-los em seus programas para facilitar o entendimento e oferecer uma anotação para você mesmo. É comum leremos um programa alguns meses depois de escrito e termos dificuldade de lembrar o que realmente queríamos fazer. Você também não precisa comentar todas as linhas de seus programas ou escrever o óbvio. Uma dica é identificar os programas com seu nome, a data em que começou a ser escrito e mesmo a listagem ou capítulo do livro onde você o encontrou.

Variáveis de tipo lógico também podem ser utilizadas para armazenar o resultado de expressões e comparações.

► Listagem 3.3 – Exemplo do uso de operadores relacionais com variáveis do tipo lógico

```

nota = 8
média = 7
aprovado = nota > média
print(aprovado)

```

Se uma expressão contém operações aritméticas, estas devem ser calculadas antes que os operadores relacionais sejam avaliados. Quando avaliamos uma expressão, substituímos o nome das variáveis por seu conteúdo e só então verificamos o resultado da comparação.

Exercício 3.2 Complete a tabela abaixo, respondendo True ou False. Considere `a = 4, b = 10, c = 5.0, d = 1 e f = 5`.

Expressão	Resultado
<code>a == c</code>	<input type="radio"/> True <input type="radio"/> False
<code>a < b</code>	<input type="radio"/> True <input type="radio"/> False
<code>d > b</code>	<input type="radio"/> True <input type="radio"/> False
<code>d != f</code>	<input type="radio"/> True <input type="radio"/> False
<code>a == b</code>	<input type="radio"/> True <input type="radio"/> False
<code>c < d</code>	<input type="radio"/> True <input type="radio"/> False
<code>b > a</code>	<input type="radio"/> True <input type="radio"/> False
<code>c >= f</code>	<input type="radio"/> True <input type="radio"/> False
<code>f >= c</code>	<input type="radio"/> True <input type="radio"/> False
<code>c <= c</code>	<input type="radio"/> True <input type="radio"/> False
<code>c <= f</code>	<input type="radio"/> True <input type="radio"/> False

3.3.2 Operadores lógicos

Para agrupar operações com lógica booleana, utilizaremos operadores lógicos. Python suporta três operadores básicos: `not` (não), `and` (e), `or` (ou). Esses operadores podem ser traduzidos como não (\neg negação), e (\wedge conjunção) e ou (\vee disjunção).

Tabela 3.3 – Operadores lógicos

Operador Python	Operação
<code>not</code>	não
<code>and</code>	e
<code>or</code>	ou

Cada operador obedece a um conjunto simples de regras, expresso pela tabela verdade desse operador. A tabela verdade demonstra o resultado de uma operação com um ou dois valores lógicos ou operandos. Quando o operador utiliza apenas um operando, dizemos que é um operador unário. Ao utilizar dois operandos, é chamado operador binário. O operador de negação (`not`) é um operador unário. `or` (ou) e `and` (e) são operadores binários, precisando, assim, de dois operandos.

3.3.2.1 Operador not

O operador `not` (não) é o mais simples, pois precisa apenas de um operador. A operação de negação também é chamada de inversão, pois um valor verdadeiro negado se torna falso e vice-versa. A tabela verdade do operador `not` (não) é apresentada na tabela 3.4.

Tabela 3.4 – Tabela verdade do operador not (não)

V ₁	not V ₁
V	F
F	V

► Listagem 3.4 – Operador not

```
>>> not True
False
>>> not False
True
```

3.3.2.2 Operador and

O operador `and` (e) tem sua tabela verdade representada na tabela 3.5. O operador `and` (e) resulta verdadeiro apenas quando seus dois operadores forem verdadeiros.

Tabela 3.5 – Tabela verdade do operador and (e)

V ₁	V ₂	V ₁ and V ₂
V	V	V
V	F	F
F	V	F
F	F	F

► Listagem 3.5 – Operador and

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
```

3.3.2.3 Operador or

A tabela verdade do operador `or` (ou) é apresentada na tabela 3.6. A regra fundamental do operador `or` (ou) é que ele resulta em falso apenas se seus dois operadores também forem falsos. Se apenas um de seus operadores for verdadeiro, ou se os dois forem, o resultado da operação será verdadeiro.

Tabela 3.6 – Tabela verdade do operador or (ou)

V ₁	V ₂	V ₁ or V ₂
V	V	V
V	F	V
F	V	V
F	F	F

► Listagem 3.6 – Operador or

```
>>> True or True
True
```

```
>>> True or False
True
>>> False or True
True
>>> False or False
False
```

```
True or False and not True
True or False and False
True or False
True
```

Os operadores relacionais também podem ser utilizados em expressões com operadores lógicos.

```
salário > 1000 and idade > 18
```

Nesses casos, os operadores relacionais devem ser avaliados primeiramente. Façamos `salário = 100` e `idade = 20`. Teremos:

```
salário > 1000 and idade > 18
100 > 1000 and 20 > 18
False and True
False
```

A grande vantagem de escrever esse tipo de expressão é representar condições que podem ser avaliadas com valores diferentes. Por exemplo: imagine que `salário > 1000 and idade > 18` seja uma condição para um empréstimo de compra de um carro novo. Quando `salário = 100` e `idade = 20`, sabemos que o resultado da expressão é falso, e podemos interpretar que, nesse caso, a pessoa não receberia o empréstimo. Avaliemos a mesma expressão com `salário = 2000` e `idade = 30`.

```
salário > 1000 and idade > 18
2000 > 1000 and 30 > 18
True and True
True
```

Agora o resultado é `True` (verdadeiro) e poderíamos dizer que a pessoa atende às condições para obter o empréstimo.

Exercício 3.4 Escreva uma expressão para determinar se uma pessoa deve ou não pagar imposto. Considere que pagam imposto pessoas cujo salário é maior que R\$ 1.200,00.

Exercício 3.3

Complete a tabela a seguir utilizando `a = True`, `b = False` e `c = True`.

Expressão	Resultado
<code>a and a</code>	<input type="radio"/> True <input type="radio"/> False
<code>b and b</code>	<input type="radio"/> True <input type="radio"/> False
<code>not c</code>	<input type="radio"/> True <input type="radio"/> False
<code>not b</code>	<input type="radio"/> True <input type="radio"/> False
<code>not a</code>	<input type="radio"/> True <input type="radio"/> False
<code>a and b</code>	<input type="radio"/> True <input type="radio"/> False
<code>b and c</code>	<input type="radio"/> True <input type="radio"/> False
<code>a or c</code>	<input type="radio"/> True <input type="radio"/> False
<code>b or c</code>	<input type="radio"/> True <input type="radio"/> False
<code>a or c</code>	<input type="radio"/> True <input type="radio"/> False
<code>b or c</code>	<input type="radio"/> True <input type="radio"/> False
<code>c or a</code>	<input type="radio"/> True <input type="radio"/> False
<code>c or b</code>	<input type="radio"/> True <input type="radio"/> False
<code>c or c</code>	<input type="radio"/> True <input type="radio"/> False
<code>b or b</code>	<input type="radio"/> True <input type="radio"/> False

3.3.3 Expressões lógicas

Os operadores lógicos podem ser combinados em expressões lógicas mais complexas. Quando uma expressão tiver mais de um operador lógico, avalia-se o operador `not` (não) primeiramente, seguido do operador `and` (e) e, finalmente, `or` (ou). Vejamos a seguir a ordem de avaliação da expressão, onde a operação sendo avaliada é sublinhada; e o resultado, mostrado na linha seguinte.

Exercício 3.5 Calcule o resultado da expressão $A > B \text{ and } C \text{ or } D$, utilizando os valores da tabela a seguir.

A	B	C	D	Resultado
1	2	True	False	
10	3	False	False	
5	1	True	True	

Exercício 3.6 Escreva uma expressão que será utilizada para decidir se um aluno foi ou não aprovado. Para ser aprovado, todas as médias do aluno devem ser maiores que 7. Considere que o aluno cursa apenas três matérias, e que a nota de cada uma está armazenada nas seguintes variáveis: matéria1, matéria2 e matéria3.

3.4 Variáveis string

Variáveis do tipo string armazenam cadeias de caracteres como nomes e textos em geral. Chamamos cadeia de caracteres uma sequência de símbolos como letras, números, sinais de pontuação etc. Exemplo: João e Maria comem pão. Nesse caso, João é uma sequência com as letras J, o, ã, o. Para simplificar o texto, utilizaremos o nome string para mencionar cadeias de caracteres. Podemos imaginar uma string como uma sequência de blocos, onde cada letra, número ou espaço em branco ocupa uma posição, como mostra a figura 3.1.

String									
J	o	ã	o	e	M	a	r	i	a
c	o	m	e	m	p	ã	o		

Figura 3.1 – Representação de uma string.

Para possibilitar a separação entre o texto do seu programa e o conteúdo de uma string, utilizaremos aspas ("") para delimitar o início e o fim da sequência de caracteres.

Voltando ao exemplo anterior, escreveremos "João e Maria comem pão". Veja que nesse caso não há qualquer problema em utilizarmos espaços. Na verdade, o computador ignora praticamente tudo que escrevemos entre aspas, mas veremos mais tarde que não é bem assim.

As variáveis do tipo string são utilizadas para armazenar sequências de caracteres, normalmente utilizadas em textos ou mensagens. O tipo string é muito útil e bastante utilizado para exibir mensagens ou mesmo para gerar outros arquivos.

Uma string em Python tem um tamanho associado, assim como um conteúdo que pode ser acessado caractere a caractere. O tamanho de uma string pode ser obtido utilizando-se a função len. Essa função retorna o número de caracteres na string. Dizemos que uma função retorna um valor quando podemos substituir o texto da função por seu resultado. A função len retorna um valor do tipo inteiro, representando a quantidade de caracteres contidos na string. Se a string é vazia (representada simplesmente por "", ou seja, duas aspas sem nada entre elas, nem mesmo espaços em branco), seu tamanho é igual a zero. Façamos alguns testes, como mostra a listagem 3.7.

■ Listagem 3.7 – A função len

```
>>> print (len("A"))
1
>>> print (len("AB"))
2
>>> print (len(""))
0
>>> print (len("O rato roeu a roupa"))
19
```

Como dito anteriormente, outra característica de strings é poder acessar seu conteúdo caractere a caractere. Sabendo que uma string tem um determinado tamanho, podemos acessar seus caracteres utilizando um número inteiro para representar sua posição. Esse número é chamado de índice, e começamos a contar de zero. Isso quer dizer que o primeiro caractere da string é de posição ou índice 0. Observe a figura 3.2.

String									
0	1	2	3	4	5	6	7	8	
A	B	C	D	E	F	G	H	I	

Figura 3.2 – Índices e conteúdo de uma variável string.

Para acessar os caracteres de uma string, devemos informar o índice ou posição do caractere entre colchetes ([]). Como o primeiro caractere de uma string é o de índice 0, podemos acessar valores de 0 até o tamanho da string menos 1. Logo,

se a string contiver 9 caracteres, poderemos acessar os caracteres de 0 a 8. Veja o resultado de alguns testes com strings na listagem 3.8. Se tentarmos acessar um índice maior que a quantidade de caracteres da string, o interpretador emitirá uma mensagem de erro.

► Listagem 3.8 – Manipulação de strings no interpretador

```
>>> a = "ABCDEF"
>>> print(a[0])
A
>>> print(a[1])
B
>>> print(a[5])
F
>>> print(a[6])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> print(len(a))
6
```

3.4.1 Operações com strings

As variáveis de tipo string suportam operações como fatiamento, concatenação e composição. Por fatiamento, podemos entender a capacidade de utilizar apenas uma parte de uma string, ou uma fatia. A concatenação nada mais é que poder juntar duas ou mais strings em uma nova string maior. A composição é muito utilizada em mensagens que enviamos à tela e consiste em utilizar strings como modelos onde podemos inserir outras strings. Veremos cada uma dessas operações nas seções a seguir.

3.4.1.1 Concatenação

O conteúdo de variáveis string podem ser somados, ou melhor, concatenados. Para concatenar duas strings, utilizamos o operador de adição (+). Assim, "AB" + "C" é igual a "ABC". Um caso especial de concatenação é a repetição de uma string várias vezes. Para isso, utilizamos o operador de multiplicação (*): "A" * 3 é igual a "AAA". Vejamos alguns exemplos na listagem 3.9.

► Listagem 3.9 – Exemplo de concatenação

```
>>> s = "ABC"
>>> print(s + "C")
ABCC
>>> print(s + "D" * 4)
ABCDODD
>>> print("X" + "-"*10 + "X")
X-----X
>>> print(s+"x4 = "+s*4)
ABCx4 = ABCABCABCABC
```

3.4.1.2 Composição

Juntar várias strings para construir uma mensagem nem sempre é prático. Por exemplo, exibir que "João tem X anos", onde X é uma variável numérica.

Usando a composição de strings do Python, podemos escrever de forma simples e clara:

"João tem %d anos" % X

Onde o símbolo de % foi utilizado para indicar a composição da string anterior com o conteúdo da variável X. O %d dentro da primeira string é o que chamamos de marcador de posição. O marcador indica que naquela posição estaremos colocando um valor inteiro, daí o %d.

Python suporta diversas operações com marcadores. Veremos mais sobre marcadores em outras partes do livro. A tabela 3.7 apresenta os principais tipos de marcadores. Veja que eles são diferentes, de acordo com o tipo de variável que vamos utilizar.

Tabela 3.7 – Marcadores

Marcador	Tipo
%d	Números inteiros
%s	Strings
%f	Números decimais

Imagine que precisamos apresentar um número como 001 ou 002, mas que também pode ser algo como 050 ou 561. Nesse caso, estamos querendo apresentar um número com três posições, completando com zeros à esquerda se o número

for menor. Podemos realizar essa operação utilizando "%03d" % x. Observe que adicionamos 03 entre o % e o d. Se você precisar apenas que o número ocupe três posições, mas não desejar zeros à esquerda, basta retirar o zero e utilizar "%3d" % x. Isso é muito importante quando estamos gravando dados em um arquivo ou simplesmente exibindo informações na tela. Vejamos alguns exemplos na listagem 3.10.

► Listagem 3.10 – Exemplo de composição com marcadores

```
>>> idade = 22
>>> print("%d" % idade)
[22]
>>> print("%03d" % idade)
[022]
>>> print("%3d" % idade)
[ 22]
>>> print("%-3d" % idade)
[22 ]
```

Quando formatamos números decimais, podemos utilizar dois valores entre o símbolo de % e a letra f. O primeiro indica o tamanho total em caracteres a reservar; e o segundo, o número de casas decimais. Assim, %5.2f diz que estaremos imprimindo um número decimal utilizando cinco posições, sendo que duas são para a parte decimal. Isso é muito interessante para exibir o resultado de cálculos ou representar dinheiro. Por exemplo, para exibir R\$ 5, você pode utilizar "R\$%f" % 5, mas o resultado não é bem o que esperamos, pois normalmente utilizamos apenas dois dígitos após a vírgula quando falamos de dinheiro. Vejamos alguns exemplos na listagem 3.11.

► Listagem 3.11 – Exemplos de composição com números decimais

```
>>> print("%5f" % 5)
5.000000
>>> print("%5.2f" % 5)
5.00
>>> print("%10.5f" % 5)
5.00000
```

O poder da composição realmente aparece quando precisamos combinar vários valores em uma nova string. Imagine que João tem 22 anos e apenas R\$ 51,34 no bolso. Para exibir essa mensagem, podemos utilizar:

```
"%s tem %d anos e apenas R$%5.2f no bolso" % ("João", 22, 51.34)
```

Python suporta diversas operações com marcadores. Veremos mais sobre marcadores em outras partes do livro. Quando temos mais de um marcador na string, somos obrigados a escrever os valores a substituir entre parênteses. Agora, vejamos exemplos com outros tipos, e utilizando mais de uma variável na composição, na listagem 3.12.

► Listagem 3.12 – Exemplo de composição de string

```
>>> nome = "João"
>>> idade = 22
>>> grana = 51.34
>>> print("%s tem %d anos e R$%f no bolso." % (nome, idade, grana))
João tem 22 anos e R$51.340000 no bolso.
>>> print("%12s tem %3d anos e R$%5.2f no bolso." % (nome, idade, grana))
        João tem 22 anos e R$51.34 no bolso.
>>> print("%12s tem %03d anos e R$%5.2f no bolso." % (nome, idade, grana))
        João tem 022 anos e R$51.34 no bolso.
>>> print("%-12s tem %-3d anos e R$%-5.2f no bolso." % (nome, idade, grana))
João      tem 22   anos e R$51.34 no bolso.
```

3.4.1.3 Fatiamento

O fatiamento em Python é muito poderoso. Imagine nossa string de exemplo da figura 3.2. Podemos fatiá-la de forma a escrever apenas seus dois primeiros caracteres AB utilizando como índice [0:2]. O fatiamento funciona com a utilização de dois pontos no índice da string. O número à esquerda dos dois pontos indica a posição de início da fatia; e o à direita, do fim. No entanto, é preciso ter atenção ao final, pois no exemplo anterior utilizamos 2; e o C, que é o caractere na posição 2; não foi incluído. Dizemos que isso acontece porque o final da fatia não é incluído na mesma, sendo deixado de fora. Entenda [0:2] como a fatia de caracteres da posição 0 até a posição 2, sem incluí-la, ou o intervalo fechado em 0 e aberto em 2.

Vejamos outros exemplos de fatias na listagem 3.13.

► Listagem 3.13 – Exemplo de fatiamento

```
>>> s="ABCDEFGHI"
>>> print(s[0:2])
AB
```

```
>>> print (s[1:2])
B
>>> print (s[2:4])
CD
>>> print (s[0:5])
ABCDE
>>> print (s[1:8])
BCDEFGH
```

Podemos também omitir o número da esquerda ou o da direita para representar do início ou até o final. Assim, `[:2]` indica do início até o segundo caractere (sem incluí-lo), e `[1:]` indica do caractere de posição 1 até o final da string. Observe que, nesse caso, nem precisamos saber quantos caracteres a string contém.

Se omitirmos o início e o fim da fatia, estaremos fazendo apenas uma cópia de todos os caracteres da string para uma nova string.

Podemos também utilizar valores negativos para indicar posições a partir da direita. Assim `-1` é o último caractere; `-2`, o penúltimo; e assim por diante. Veja o resultado de testes com índices negativos na listagem 3.14.

► Listagem 3.14 – Exemplo de fatiamento com omissão de valores e com índices negativos

```
>>> s="ABCDEFGHI"
>>> print (s[:2])
AB
>>> print (s[1:])
BCDEFGHI
>>> print (s[0:-2])
ABCDEFG
>>> print (s[:])
ABCDEFGHI
>>> print (s[-1:])
I
>>> print (s[-5:7])
EFG
>>> print (s[-2:-1])
H
```

Veremos mais sobre strings em Python no capítulo 7.

3.5 Sequências e tempo

Um programa é executado linha por linha pelo computador, executando as operações descritas no programa uma após a outra. Quando trabalhamos com variáveis, devemos nos lembrar de que o conteúdo de uma variável pode mudar com o tempo. Isso porque a cada vez que alterarmos o valor de uma variável, o valor anterior é substituído pelo novo.

Observe o programa da listagem 3.15. A variável `dívida` foi utilizada para registrar quanto alguém estava devendo; e a variável `compra`, o valor de novas despesas dessa pessoa. Como somos justos, a pessoa começou sem dívidas em ①.

► Listagem 3.15 – Exemplo de sequência e tempo

```
dívida = 0 ①
compra = 100 ②
dívida = dívida + compra ③
compra = 200 ④
dívida = dívida + compra ⑤
compra = 300 ⑥
dívida = dívida + compra ⑦
compra = 0 ⑧
print(dívida) ⑨
```

Em ②, temos a primeira compra no valor de R\$ 100. No entanto, o valor da dívida continua sendo 0, pois ainda não alteramos seu valor de forma a adicionar a compra realizada. Isso é feito em ③. Observe que estamos atualizando o valor da dívida com o valor atual mais a compra.

Em ④, registramos uma nova compra no valor de R\$ 200. Nesse ponto, compra tem seu valor substituído por R\$ 200, causando a perda do valor anterior R\$ 100. Como já somamos o valor anterior na variável `dívida`, essa perda não representará um problema.

⑤ é idêntica a ③, mas o resultado é bem diferente. Nesse momento, compra vale R\$ 200; e dívida, R\$ 100.

Em ⑥, alteramos o valor de compra novamente. Dessa vez, a compra foi de R\$ 300.

⑦ é idêntica a ③ e ⑤, mas seu resultado é diferente, pois nesse momento temos compra valendo R\$ 300; e dívida igual a R\$ 300 ($100 + 300$), sendo atualizada para R\$ 600 ($300 + 300$).

Em ❸, simplesmente dizemos que a compra foi 0, representando que a pessoa não comprou mais nada.

❹ exibe o conteúdo da variável dívida na tela (600).

A figura 3.3 mostra a evolução do conteúdo de nossas duas variáveis em função do tempo, representado pelo número da linha.

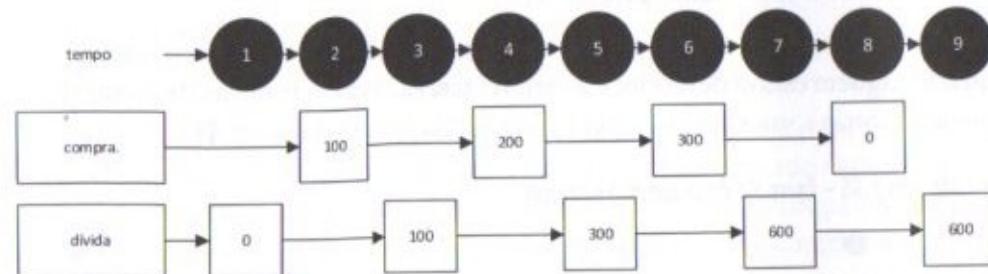


Figura 3.3 – Mudanças no valor de duas variáveis no tempo.

3.6 Rastreamento

Uma das principais diferenças entre ler um texto e um programa é justamente seguir as mudanças de valores de cada variável conforme o programa é executado. Entender que o valor das variáveis pode mudar durante a execução do programa não é tão natural, mas é fundamental para a programação de computadores. Um programa não pode ser lido como um texto, mas cuidadosamente analisado linha a linha. Ao escrever seus programas, verifique linha a linha os efeitos e mudanças causados no valor de cada variável. Para programar corretamente, você deve ser capaz de entender o que cada linha do programa significa e os efeitos que produz. Essa atividade, chamada de rastreamento, é muito importante para entender novos programas e para encontrar erros nos programas que você escreverá.

Para rastrear um programa, utilize lápis, borracha e uma folha de papel. Escreva o nome de suas variáveis na folha de papel, como se fossem títulos de colunas, deixando espaço para ser preenchido embaixo desses nomes. Leia uma linha do programa de cada vez e escreva o valor atribuído a cada variável na outra folha, na mesma coluna em que escreveu o nome da variável. Se o valor da variável mudar, escreva o novo valor e risque o anterior, um embaixo do outro, formando uma coluna. Ao exibir algo na tela, escreva também na outra folha, como se ela fosse a sua tela (você pode desenhar a tela como se fosse uma variável, mas se lembre de deixar um pouco mais de espaço). Um exemplo de como ficaria o resultado do rastreamento do programa da listagem 3.15 é apresentado na figura 3.4. O

rastreamento vai ajudá-lo a entender melhor as mudanças de valores de suas variáveis e a acompanhar a execução do programa, como mais tarde será feito pelo computador. É um processo detalhado que precisa de atenção. Não tente simplificá-lo ou começar a rastrear no meio de um programa. Você deve rastrear linha a linha, do início ao fim do programa. Se você encontrar um erro, pode parar o rastreamento e corrigi-lo, mas lembre-se de recomeçar do início sempre que alterar o programa ou os valores sendo rastreados.

Tela	dívida	compra
600	6	100
100	100	200
300	300	300
600	600	0

Figura 3.4 – Exemplo de rastreamento no papel.

Dominar o rastreamento de um programa é essencial para programar e ajuda muito a entender como os programas realmente funcionam. Lembre-se de que programar é detalhar, e que simplesmente ler o texto de um programa não é suficiente. Você deve rastreá-lo para entendê-lo. Embora pareça óbvio, esse é um dos erros mais comuns quando se começa a programar. Se um programa não funciona ou se você não entendeu exatamente o que ele faz, o rastreamento é a melhor ferramenta.

3.7 Entrada de dados

Até agora nossos programas trabalharam apenas com valores conhecidos, escritos no próprio programa. No entanto, o melhor da programação é poder escrever a solução de um problema e aplicá-la várias vezes. Para isso, precisamos melhorar nossos programas de forma a permitir que novos valores sejam fornecidos durante sua execução, de modo que poderemos executá-los com valores diferentes sem alterar os programas em si.

Chamamos de entrada de dados o momento em que seu programa recebe dados ou valores por um dispositivo de entrada de dados (como o teclado do computador) ou de um arquivo em disco.

A função `input` é utilizada para solicitar dados do usuário. Ela recebe um parâmetro, que é a mensagem a ser exibida, e retorna o valor digitado pelo usuário. Vejamos um exemplo na listagem 3.16.

► Listagem 3.16 – Entrada de dados

```
x = input("Digite um número: ")
print(x)
```

► Listagem 3.17 – Saída na tela, tendo o 5 como exemplo de número digitado pelo usuário

Digite um número: 5

5

Vejamos outro exemplo na listagem 3.18.

► Listagem 3.18 – Exemplo de entrada de dados

```
nome = input("Digite seu nome: ") ❶
print("Você digitou %s" % nome)
print("Olá, %s!" % nome)
```

Em ❶, solicitamos a entrada de dados, no caso, o nome do usuário. A mensagem “Digite seu nome:” é exibida, e o programa para até que o usuário digite ENTER. Só então o resto do programa é executado. Vejamos a saída de dados quando digitamos João como nome na listagem 3.19.

► Listagem 3.19 – Resultado da entrada de dados

```
Digite seu nome:João
Você digitou João
Olá, João!
```

Execute o programa outras vezes digitando, por exemplo, 123 como nome. Observe que o programa não se importa com os valores digitados pelo usuário. Essa verificação deve ser feita por seu programa. Veremos como fazer isso em outro capítulo, como validação de dados.

3.7.1 Conversão da entrada de dados

A função `input` sempre retorna valores do tipo `string`, ou seja, não importa se digitamos apenas números, o resultado sempre é `string`. Para resolver esse pequeno

problema, vamos utilizar a função `int` para converter o valor retornado em um número inteiro, e a função `float` para convertê-lo em número decimal ou de ponto flutuante. Vejamos outro exemplo usando essas funções, no qual devemos calcular o valor de um bônus por tempo de serviço na listagem 3.20.

► Listagem 3.20 – Cálculo de bônus por tempo de serviço

```
anos = int(input("Anos de serviço: "))
valor_por_ano = float(input("Valor por ano: "))
bônus = anos * valor_por_ano
print("Bônus de R$ %.2f" % bônus)
```

Vejamos o resultado se testarmos 10 anos e R\$ 25 por ano na tela da listagem 3.21. Observe que escrevemos apenas 25, e não R\$ 25. Isso porque 25 é um número, e R\$ 25 é uma string. Por enquanto, não vamos misturar dois tipos de dados na mesma entrada de dados, para simplificar nossos programas.

► Listagem 3.21 – Resultado do cálculo para 10 anos e R\$ 25 por ano

```
Anos de serviço: 10
Valor por ano: 25
Bônus de R$ 250.00
```

Execute o programa novamente com outros valores. Experimente digitar uma letra em anos de serviço ou em valor por ano. Você receberá uma mensagem de erro, pois a conversão de letras em números não é automática. A próxima seção explica melhor o problema.

Exercício 3.7 Faça um programa que peça dois números inteiros. Imprima a soma desses dois números na tela.

Exercício 3.8 Escreva um programa que leia um valor em metros e o exiba convertido em milímetros.

Exercício 3.9 Escreva um programa que leia a quantidade de dias, horas, minutos e segundos do usuário. Calcule o total em segundos.

Exercício 3.10 Faça um programa que calcule o aumento de um salário. Ele deve solicitar o valor do salário e a porcentagem do aumento. Exiba o valor do aumento e do novo salário.

Exercício 3.11 Faça um programa que solicite o preço de uma mercadoria e o percentual de desconto. Exiba o valor do desconto e o preço a pagar.

Exercício 3.12 Escreva um programa que calcule o tempo de uma viagem de carro. Pergunte a distância a percorrer e a velocidade média esperada para a viagem.

Exercício 3.13 Escreva um programa que converta uma temperatura digitada em °C em °F. A fórmula para essa conversão é:

$$F = \frac{9 \times C}{5} + 32$$

Exercício 3.14 Escreva um programa que pergunte a quantidade de km percorridos por um carro alugado pelo usuário, assim como a quantidade de dias pelos quais o carro foi alugado. Calcule o preço a pagar, sabendo que o carro custa R\$ 60 por dia e R\$ 0,15 por km rodado.

Exercício 3.15 Escreva um programa para calcular a redução do tempo de vida de um fumante. Pergunte a quantidade de cigarros fumados por dia e quantos anos ele já fumou. Considere que um fumante perde 10 minutos de vida a cada cigarro, calcule quantos dias de vida um fumante perderá. Exiba o total em dias.

3.7.2 Erros comuns

A entrada de dados é um ponto frágil em nossos programas. Como não temos como prever o que o usuário vai digitar, temos que nos preparar para reconhecer os erros mais comuns. Vejamos um programa que lê três valores na listagem 3.22.

► Listagem 3.22 – Entrada de dados com conversão de tipos

```
nome = input("Digite seu nome: ")
idade = int(input("Digite sua idade: "))
saldo = float(input("Digite o saldo da sua conta bancária: "))

print(nome)
print(idade)
print(saldo)
```

A listagem 3.23 mostra o resultado da execução do programa quando todos os valores são digitados corretamente. Correto significa uma ausência de erros durante a função `input` ou durante a conversão do valor retornado pela função.

► Listagem 3.23 – Exemplo de entrada de dados

```
Digite seu nome: João
Digite sua idade: 42
Digite o saldo da sua conta bancária: 15756.34
João
42
15756.34
```

A listagem 3.24 mostra outro exemplo de entrada de dados bem-sucedida. Observe que, como utilizamos a função `float` para converter o saldo, mesmo inserindo 34 o valor foi convertido para 34.0.

► Listagem 3.24 – Exemplo de entrada de dados

```
Digite seu nome: Maria
Digite sua idade: 28
Digite o saldo da sua conta bancária: 34
Maria
28
34.0
```

Já na listagem 3.25, temos um exemplo de erro durante a entrada de dados. No caso, digitamos letras (abc) que não podem ser convertidas em um valor inteiro. Observe que o erro interrompe nosso programa, exibindo a linha em que ocorreu e o nome do erro. Nesse caso, o erro aconteceu na linha 2 e seu nome é `ValueError: invalid literal for int with base 10: 'abc'`.

► 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: "))
```