CAPÍTULO 03

3. Uma introdução informal ao Python

Nos exemplos seguintes, pode-se distinguir entrada e saída pela presença ou ausência dos prompts (>>> e ...): para repetir o exemplo, você deve digitar tudo após o prompt, quando o mesmo aparece; linhas que não começarem com um prompt são na verdade as saídas geradas pelo interpretador. Observe que quando aparece uma linha contendo apenas o prompt secundário você deve digitar uma linha em branco; é assim que se encerra um comando de múltiplas linhas.

You can toggle the display of prompts and output by clicking on >>> in the upper-right corner of an example box. If you hide the prompts and output for an example, then you can easily copy and paste the input lines into your interpreter.

Muitos exemplos neste manual, mesmo aqueles inscritos na linha de comando interativa, incluem comentários. Comentários em Python começam com o caractere cerquilha '#' e estende até o final da linha. Um comentário pode aparecer no inicio da linha ou após espaço em branco ou código, mas não dentro de uma string literal. O caracterer cerquilha dentro de uma string literal é apenas uma cerquilha. Como os comentários são para esclarecer o código e não são interpretados pelo Python, eles podem ser omitidos ao digitar exemplos.

Alguns exemplos:

```
# this is the first comment
spam = 1  # and this is the second comment
          # ... and now a third!
text = "# This is not a comment because it's inside quotes."
```

3.1. Usando Python como uma calculadora

Vamos experimentar alguns comandos simples em Python. Inicie o interpretador e aguarde o prompt primário, >>>. (Não deve demorar muito.)

3.1.1. Números

O interpretador funciona como uma calculadora bem simples: você pode digitar uma expressão e o resultado será apresentado. A sintaxe de expressões é a usual: operadores +, -, * e / funcionam da mesma forma que em outras linguagens tradicionais (por exemplo, Pascal ou C); parênteses (()) podem ser usados para agrupar expressões. Por exemplo:

```
>>> 2 + 2
4
```

```
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # division always returns a floating point number
1.6
```

Os números inteiros (ex. `2`, `4`, `20`) são do tipo `int`, aqueles com parte fracionária (ex. `5.0`, `1.6`) são do tipo `float`. Veremos mais sobre tipos numéricos posteriormente neste tutorial.

Divisão (`/`) sempre retorna ponto flutuante (float). Para fazer uma divisão pelo piso e receber um inteiro como resultado (descartando a parte fracionária) você pode usar o operador `//`; para calcular o resto você pode usar o `%`:

```
>>> 17 / 3  # classic division returns a float
5.666666666666667
>>>
>>> 17 // 3  # floor division discards the fractional part
5
>>> 17 % 3  # the % operator returns the remainder of the
division
2
>>> 5 * 3 + 2  # floored quotient * divisor + remainder
17
```

Com Python, é possível usar o operador `**` para calcular potências [1]:

```
>>> 5 ** 2  # 5 squared
25
>>> 2 ** 7  # 2 to the power of 7
128
```

O sinal de igual (`'='`) é usado para atribuir um valor a uma variável. Depois de uma atribuição, nenhum resultado é exibido antes do próximo prompt:

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

Se uma variável não é “definida” (não tem um valor atribuído), tentar utilizá-la gerará um erro:

```
>>> n  # try to access an undefined variable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

Há suporte completo para ponto flutuante (*float*); operadores com operandos de diferentes tipos convertem o inteiro para ponto flutuante:

```
>>> 4 * 3.75 - 1
14.0
```

No modo interativo, o valor da última expressão exibida é atribuída a variável `_`. Assim, ao utilizar Python como uma calculadora, fica mais fácil prosseguir com os cálculos, por exemplo:

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
>>> round(_, 2)
113.06
```

Essa variável especial deve ser tratada como *somente para leitura* pelo usuário. Nunca lhe atribua explicitamente um valor — do contrário, estaria criando uma outra variável (homônima) independente, que mascararia a variável especial com seu comportamento mágico.

Além de `int` e `float`, o Python suporta outros tipos de números, tais como `Decimal` e `Fraction`. O Python também possui suporte nativo a números complexos, e usa os sufixos `j` ou `J` para indicar a parte imaginária (por exemplo, `3+5j`).

3.1.2. Strings

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples (`'...'`) ou duplas (`"..."`) e teremos o mesmo resultado 2. `\` pode ser usada para escapar aspas:

```
>>> 'spam eggs'  # single quotes
'spam eggs'
>>> 'doesn\'t'  # use \' to escape the single quote...
"doesn't"
>>> "doesn't"  # ...or use double quotes instead
```

```
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```

Na interpretação interativa, a string de saída é delimitada com aspas e caracteres especiais são escapados com barras invertidas. Embora isso possa às vezes parecer diferente da entrada (as aspas podem mudar), as duas strings são equivalentes. A string é delimitada com aspas duplas se a string contiver uma única aspa simples e nenhuma aspa dupla, caso contrário, ela é delimitada com aspas simples. A função `print()` produz uma saída mais legível, ao omitir as aspas e ao imprimir caracteres escapados e especiais:

```
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
>>> print('"Isn\'t," they said.')
"Isn't," they said.
>>> s = 'First line.\nSecond line.'  # \n means newline
>>> s  # without print(), \n is included in the output
'First line.\nSecond line.'
>>> print(s)  # with print(), \n produces a new line
First line.
Second line.
```

Se não quiseres que os caracteres sejam precedidos por `\` para serem interpretados como caracteres especiais, poderás usar *strings raw* (N.d.T: “crua” ou sem processamento de caracteres de escape) adicionando um `r` antes da primeira aspa:

```
>>> print('C:\some\name')  # here \n means newline!
C:\some
ame
>>> print(r'C:\some\name')  # note the r before the quote
C:\some\name
```

As strings literais podem abranger várias linhas. Uma maneira é usar as aspas triplas: `"""..."""` ou `'''...'''`. O fim das linhas é incluído automaticamente na string, mas é possível evitar isso adicionando uma `\` no final. O seguinte exemplo:

```
print("""\
Usage: thingy [OPTIONS]
     -h                        Display this usage message
```

```
     -H hostname               Hostname to connect to
""")
```

produz a seguinte saída (observe que a linha inicial não está incluída):

```
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
```

Strings podem ser concatenadas (coladas) com o operador `+`, e repetidas com `*`:

```
>>> # 3 times 'un', followed by 'ium'
>>> 3 * 'un' + 'ium'
'unununium'
```

Duas ou mais *strings literais* (ou seja, entre aspas) ao lado da outra são automaticamente concatenados.

```
>>> 'Py' 'thon'
'Python'
```

Esse recurso é particularmente útil quando você quer quebrar strings longas:

```
>>> text = ('Put several strings within parentheses '
...         'to have them joined together.')
>>> text
'Put several strings within parentheses to have them joined
together.'
```

Isso só funciona com duas strings literais, não com variáveis ou expressões:

```
>>> prefix = 'Py'
>>> prefix 'thon'  # can't concatenate a variable and a string
literal
  File "<stdin>", line 1
    prefix 'thon'
                ^
SyntaxError: invalid syntax
>>> ('un' * 3) 'ium'
  File "<stdin>", line 1
    ('un' * 3) 'ium'
                   ^
SyntaxError: invalid syntax
```

Se você quiser concatenar variáveis ou uma variável e uma literal, use `+`:

```
>>> prefix + 'thon'
'Python'
```

As strings podem ser *indexadas* (subscritas), com o primeiro caractere como índice 0. Não existe um tipo específico para caracteres; um caractere é simplesmente uma string cujo tamanho é 1:

```
>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'
```

Índices também podem ser números negativos para iniciar a contagem pela direita:

```
>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
>>> word[-6]
'P'
```

Note que dado que -0 é o mesmo que 0, índices negativos começam em -1.

Além da indexação, o *fatiamento* também é permitido. Embora a indexação seja usada para obter caracteres individuais, *fatiar* permite que você obtenha substring:

```
>>> word[0:2]  # characters from position 0 (included) to 2
(excluded)
'Py'
>>> word[2:5]  # characters from position 2 (included) to 5
(excluded)
'tho'
```

Os índices do fatiamento possuem padrões úteis; um primeiro índice omitido padrão é zero, um segundo índice omitido é por padrão o tamanho da string sendo fatiada:

```
>>> word[:2]    # character from the beginning to position 2
(excluded)
'Py'
```

```
>>> word[4:]    # characters from position 4 (included) to the end
'on'
>>> word[-2:]  # characters from the second-last (included) to the end
'on'
```

Observe como o início sempre está incluído, e o fim sempre é excluído. Isso garante que `s[:i] + s[i:]` seja sempre igual a `s`:

```
>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

Uma maneira de lembrar como fatias funcionam é pensar que os índices indicam posições *entre* caracteres, onde a borda esquerda do primeiro caractere é 0. Assim, a borda direita do último caractere de uma string de comprimento *n* tem índice *n*, por exemplo:

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

A primeira fileira de números indica a posição dos índices 0…6 na string; a segunda fileira indica a posição dos respectivos índices negativos. Uma fatia de *i* a *j* consiste em todos os caracteres entre as bordas *i* e *j*, respectivamente.

Para índices positivos, o comprimento da fatia é a diferença entre os índices, se ambos estão dentro dos limites da string. Por exemplo, o comprimento de `word[1:3]` é 2.

A tentativa de usar um índice que seja muito grande resultará em um erro:

```
>>> word[42]  # the word only has 6 characters
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

No entanto, os índices de fatiamento fora do alcance são tratados graciosamente (N.d.T: o termo original “gracefully” indica robustez no tratamento de erros) quando usados para fatiamento. Um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia:

```
>>> word[4:42]
'on'
>>> word[42:]
''
```

As strings do Python não podem ser alteradas — uma string é imutável. Portanto, atribuir a uma posição indexada na sequência resulta em um erro:

```
>>> word[0] = 'J'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> word[2:] = 'py'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Se você precisar de uma string diferente, deverá criar uma nova:

```
>>> 'J' + word[1:]
'Jython'
>>> word[:2] + 'py'
'Pypy'
```

A função embutida `len()` devolve o comprimento de uma string:

```
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)
34
```

Ver também

Tipo sequência de texto — str

As strings são exemplos de *tipos de sequências* e suportam as operações comumente suportadas por esses tipos.

Métodos de string

As strings suportam uma grande quantidade de métodos para transformações básicas e busca.

Literais de string formatados

Strings literais que possuem expressões embutidas.

Sintaxe das strings de formato

Informações sobre formatação de string com o método `str.format()`.

Formatação de String no Formato printf-style

As antigas operações de formatação invocadas quando as strings são o operando esquerdo do operador `%` são descritas com mais detalhes aqui.

ATIVIDADES

1. Faça um Programa que mostre a mensagem "Alo mundo" na tela

2. Faça um Programa que peça um número e então mostre a mensagem *O número informado foi [número]*.

3. Faça um Programa que peça as 4 notas bimestrais e mostre a média.

4. Faça um Programa que peça o raio de um círculo, calcule e mostre sua área.

5. Faça um Programa que peça a temperatura em graus Fahrenheit, transforme e mostre a temperatura em graus Celsius.

a. C = 5 * ((F-32) / 9).

3.1.3. Listas

Python inclui diversas estruturas de dados *compostas*, usadas para agrupar outros valores. A mais versátil é *list* (lista), que pode ser escrita como uma lista de valores (itens) separados por vírgula, entre colchetes. Os valores contidos na lista não precisam ser todos do mesmo tipo.

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
```

Como strings (e todos os tipos embutidos de sequência), listas pode ser indexados e fatiados:

```
>>> squares[0]  # indexing returns the item
1
>>> squares[-1]
25
>>> squares[-3:]  # slicing returns a new list
[9, 16, 25]
```

Todas as operações de fatiamento devolvem uma nova lista contendo os elementos solicitados. Isso significa que o seguinte fatiamento devolve uma cópia rasa da lista:

```
>>> squares[:]
[1, 4, 9, 16, 25]
```

As listas também suportam operações como concatenação:

```
>>> squares + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Diferentemente de strings, que são imutáveis, listas são mutáveis, ou seja, é possível alterar elementos individuais de uma lista:

```
>>> cubes = [1, 8, 27, 65, 125]  # something's wrong here
>>> 4 ** 3  # the cube of 4 is 64, not 65!
64
>>> cubes[3] = 64  # replace the wrong value
>>> cubes
[1, 8, 27, 64, 125]
```

Você também pode adicionar novos itens no final da lista, usando o *método* `append()` (estudaremos mais a respeito dos métodos posteriormente):

```
>>> cubes.append(216)  # add the cube of 6
>>> cubes.append(7 ** 3)  # and the cube of 7
>>> cubes
[1, 8, 27, 64, 125, 216, 343]
```

Atribuição a fatias também é possível, e isso pode até alterar o tamanho da lista ou remover todos os itens dela:

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # replace some values
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # now remove them
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
```

```
>>> # clear the list by replacing all the elements with an
empty list
>>> letters[:] = []
>>> letters
[]
```

A função embutida `len()` também se aplica a listas:

```
>>> letters = ['a', 'b', 'c', 'd']
>>> len(letters)
4
```

É possível aninhar listas (criar listas contendo outras listas), por exemplo:

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

3.2. Primeiros passos para a programação

Claro, podemos usar o Python para tarefas mais complicadas do que somar 2+2. Por exemplo, podemos escrever o início da sequência de Fibonacci assim:

```
>>> # Fibonacci series:
... # the sum of two elements defines the next
... a, b = 0, 1
>>> while a < 10:
...     print(a)
...     a, b = b, a+b
...
0
1
1
2
3
5
8
```

Este exemplo introduz diversas características ainda não mencionadas.

- A primeira linha contém uma atribuição múltipla: as variáveis `a` e `b` recebem simultaneamente os novos valores 0 e 1. Na última linha há outro exemplo de atribuição múltipla demonstrando que expressões do lado direito são sempre avaliadas primeiro, antes da atribuição. As expressões do lado direito são avaliadas da esquerda para a direita.
- O laço de repetição `while` executa enquanto a condição (aqui: `a < 10`) permanece verdadeira. Em Python, como em C, qualquer valor inteiro que não seja zero é considerado verdadeiro; zero é considerado falso. A condição pode também ser uma cadeia de caracteres ou uma lista, ou qualquer sequência; qualquer coisa com um tamanho maior que zero é verdadeiro, enquanto sequências vazias são falsas. O teste usado no exemplo é uma comparação simples. Os operadores padrões de comparação são os mesmos de C: `<` (menor que), `>` (maior que), `==` (igual), `<=` (menor ou igual), `>=` (maior ou igual) e `!=` (diferente).
- O *corpo* do laço é *indentado*: indentação em Python é a maneira de agrupar comandos em blocos. No console interativo padrão você terá que digitar tab ou espaços para indentar cada linha. Na prática você vai preparar scripts Python mais complicados em um editor de texto; a maioria dos editores de texto tem facilidades de indentação automática. Quando um comando composto é digitado interativamente, deve ser finalizado por uma linha em branco (já que o interpretador não tem como adivinhar qual é a última linha do comando). Observe que toda linha de um mesmo bloco de comandos deve ter a mesma indentação.
- A função `print()` escreve o valor dos argumentos fornecidos. É diferente de apenas escrever a expressão no interpretador (como fizemos anteriormente nos exemplos da calculadora) pela forma como lida com múltiplos argumentos, quantidades de ponto flutuante e strings. As strings são impressas sem aspas, e um espaço é inserido entre os itens, assim você pode formatar bem o resultado, dessa forma:

```
>>> i = 256*256
>>> print('The value of i is', i)
The value of i is 65536
```

O argumento *end* pode ser usado para evitar uma nova linha após a saída ou finalizar a saída com uma string diferente:

```
>>> a, b = 0, 1
>>> while a < 1000:
...     print(a, end=',')
...     a, b = b, a+b
...
0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,
```

ATIVIDADES

1. Faça um Programa que leia um vetor de 5 números inteiros e mostre-os.
2. Utilizando listas faça um programa que faça 5 perguntas para uma pessoa sobre um crime. As perguntas são:
    1. "Telefonou para a vítima?"
    2. "Esteve no local do crime?"
    3. "Mora perto da vítima?"
    4. "Devia para a vítima?"
    5. "Já trabalhou com a vítima?" O programa deve no final emitir uma classificação sobre a participação da pessoa no crime. Se a pessoa responder positivamente a 2 questões ela deve ser classificada como "Suspeita", entre 3 e 4 como "Cúmplice" e 5 como "Assassino". Caso contrário, ele será classificado como "Inocente".
3. Faça um Programa que leia 4 notas, mostre as notas e a média na tela.