Programação básica em Python

Fundamentos de Python: estruturas de dados
(Kenneth A. Lambert)
Adaptado por Lucília Ribeiro

Elementos básicos do programa

- O Python possui uma ampla gama de recursos e construções
  - Está entre as poucas linguagens cujos elementos básicos são bastante simples.

Programas e módulos

- Python consiste em um ou mais módulos.
- Módulo é apenas um arquivo de código.
  - Pode incluir instruções, definições de função e definições de classe.
- Script: pequeno programa.
  - Pode estar contido em um módulo.
- Programas longos e complexos geralmente incluem:
  - Um módulo principal e um ou mais módulos de suporte.
- Módulo principal:
  - Contém o ponto de partida da execução do programa.
- Módulos de suporte:
  - Contêm definições de funções e classes.

Um exemplo de programa Python: adivinhando um número (1 de 2)

```
"""Fornece os limites de um intervalo de números
e deixa o usuário adivinhar o número do computador até
a suposição estiver correta."""

import random

menor = int(input("Entre com o limite inferior: "))
maior = int(input("Entre com o limite superior: "))
meuNumero = random.randint(menor, maior)
count = 0
while True:
    count += 1
    numero = int(input("Adivinha? "))
    if numero < meuNumero:
        print("Muito pequeno")
    elif numero > meuNumero:
        print("Muito grande")
    else:
        print("Adivinhou em ", count, " tentativas!")
        break
```

Editando, compilando e executando programas Python (1 de 2)

- Para executar o programa contido no arquivo Jogo.py, digite o seguinte comando na maioria das janelas de terminal:
  - `python Jogo.py`
- Para criar ou editar um módulo Python, use o IDLE (abreviação de Integrated DeveLopment Environment) do Python:
  - Digite o idle ou comando idle3 at a terminal em um prompt de terminal ou clique no seu ícone.
  - Você também pode iniciar o IDLE:
    - Clicando duas vezes em um arquivo de código-fonte Python.
    - Clicando com o botão direito do mouse no arquivo e selecionando Open ou Edit com IDLE.
- O IDLE fornece uma janela de shell para a execução interativa de expressões e instruções Python:
  - IDLE também formata o código e o codifica por cores.

Editando, compilando e executando programas Python (2 de 2)

- Para executar um programa:
  - Mova o cursor até a janela do editor e pressione a tecla F5 (função-5).
- O Python compila o código na janela do editor e o executa na janela do shell.
- Se um programa Python travar ou não fechar da maneira normal,
  - você pode fechá-lo pressionando Ctrl+C ou fechando a janela do shell.

Comentando programas

- Comentário de programa
  - Texto ignorado pelo compilador Python.
  - Importante para o leitor como documentação.
- Comentário de fim de linha
  - Começa com um símbolo **#** e se estende até o final da linha atual.
- Comentário multilinha
  - Uma string entre aspas simples triplas ou aspas duplas triplas.
  - Também são chamados **docstrings**, para indicar que podem documentar as principais construções de um programa.

Elementos lexicais

- Elementos lexicais
  - Tipos de palavras ou símbolos usados para construir frases.
- Também incluídos entre os itens lexicais:
  - Identificadores (nomes).
  - Literais (números, strings e outras estruturas de dados incorporadas).
  - Operadores.
  - Delimitadores (aspas, vírgulas, parênteses, colchetes e chaves).

Convenções de ortografia e nomenclatura (1 de 2)

- Palavras-chave e nomes Python diferenciam maiúsculas de minúsculas:
  - As palavras-chave do Python são escritas em letras minúsculas e codificadas em laranja.
  - Todos os nomes Python, exceto os das funções incorporadas, são codificados em preto.
- Um nome pode começar com uma letra ou um sublinhado (_)
  - Seguido por qualquer número de letras, sublinhados ou dígitos.
- Por definição:
  - nomes dos módulos, variáveis, funções e métodos são escritos em letras minúsculas.
  - Os nomes das classes seguem as mesmas convenções, mas começam com letra maiúscula.
  - Quando uma variável nomeia uma constante, todas as letras são maiúsculas e um sublinhado separa todas as palavras incorporadas.

Convenções de ortografia e nomenclatura (2 de 2)

- Tabela 1.1 Exemplos de convenções de nomenclatura do Python

Tipo de nome	Exemplos
Variável	`salario, horasTrabalhadas, isento`
Constante	`ZERO_ABSOLUTO, TAXA_INTERESSE`
Função ou método	`imprimeResultados, raizCubica, entrada`
Classe	`ContaCorrente, ConjuntoEscolhido`

Elementos sintáticos

- Elementos sintáticos
  - Tipos de frases (expressões, instruções, definições e outras construções) compostas a partir dos elementos lexicais.
- O Python usa espaços em branco para marcar a sintaxe de muitos tipos de frases:
  - Recuos e quebras de linha.
- Este livro utiliza indentação na largura de quatro espaços em todas os códigos Phyton.

Literais

- Os números (inteiros ou de ponto flutuante) são escritos como em outras linguagens de programação.
- valores booleanos `True` e `False` são palavras-chave.
- Algumas estruturas de dados também possuem literais.
  - Como strings, tuplas, listas e dicionários.

Literais de string (1 de 2)

- Você pode colocar strings entre:
  - aspas simples;
  - aspas duplas;
  - conjuntos de três aspas duplas ou três aspas simples.
- Os valores dos caracteres são strings de um único caractere.
- O caractere \ é usado para “escapar” de caracteres, como
  - a nova linha (\n) e a tabulação (\t), ou o caractere \ em si.

Literais de string (2 de 2)

- Exemplo:
  - `print("Usando aspas duplas")`
  - `print('Usando aspas simples')`
  - `print("Escrevendo a palavra 'Python' entre aspas")`
  - `print("Incorporando uma\nquebra de linha com \\n")`
  - `print("""Incorporando uma quebra de`
  - `de linha com três aspas duplas""")`
- Saída:

```
Usando aspas duplas
Usando aspas simples
Escrevendo a palavra 'Python' entre aspas
Incorporando uma
quebra de linha com \n
Incorporando uma quebra de
• de linha com três aspas duplas
```

Operadores e expressões (1 de 2)

- As expressões aritméticas usam
  - Os operadores +, -, *, e %
- O operador / produz um resultado de ponto flutuante com quaisquer operandos numéricos.
- O operador // produz um quociente de número inteiro.
- O operador + significa concatenação quando usado com coleções.
- O operador ** é usado para exponenciação.

Operadores e expressões (2 de 2)

- Estes operadores de comparação funcionam com números e strings
  - **<**, **<=**, **>**, **>=**, **==** e **!=**
- O operador **==** compara o conteúdo interno das estruturas de dados quanto à equivalência estrutural.
- o operador is compara dois valores para a identidade do objeto.
- Comparações retornam **`True`** ou **`False`**.
- Os operadores lógicos **`and`**, **`ou`** e **`not`** tratam vários valores, como 0, **`None`**, a string vazia e a lista vazia, como **`False`**.
  - A maioria dos outros valores Python contam como **`True`**.

Chamadas de função

- As funções são chamadas
  - Com o nome da função seguido por uma lista de argumentos entre parênteses
- Exemplo:
  - `min(5,2)    #Retorna 2`

A função `print`

- **`print`**
  - Exibe seus argumentos no console.
- O Python executa automaticamente a função **`str`** em cada argumento para obter a representação de string:
  - Separa cada string com um espaço antes da saída.
- Por padrão
  - **`print`** termina a saída com uma nova linha.

A função `input`

- **`input`**
  - Espera que o usuário insira texto no teclado.

- Quando o usuário pressiona a tecla Enter,
  - A função retorna uma string contendo os caracteres inseridos.

Funções de conversão de tipo e operações de modo misto

- Você pode usar alguns nomes de tipos de dados como funções de conversão de tipo
- Exemplo:
  - Quando o usuário insere um número no teclado, a função **`input`** retorna uma sequência de dígitos, não um valor numérico.
  - O programa deve converter a string em um **`int`** ou um **`float`** antes do processamento numérico.
- O próximo segmento de código insere o raio de um círculo, converte essa string em um float e calcula e gera a área do círculo :

```
raio = float(input("Raio: "))

print("A area = ", 3.14 * raio ** 2)
```

Argumentos opcionais e de função de palavra-chave

- As funções podem permitir argumentos opcionais
  - Podem ser nomeados com palavras-chave quando a função é chamada.
- Exemplo:
  - Para evitar que a função print por padrão gere uma nova linha depois que os argumentos são exibidos, você pode fornecer ao argumento opcional end um valor da string vazia:

```
print("O cursor permanece no final da linha", end="")
```

- Os argumentos necessários não têm valores padrão.
  - Os argumentos opcionais têm valores padrão.
- O número de argumentos transmitidos a uma função quando ela é chamada deve ser pelo menos igual aos argumentos necessários.

Variáveis e instruções de atribuição

- Uma variável Python é introduzida com uma instrução de atribuição:
  - `PI = 3.1416`
- Sintaxe:
  - `<identificador> = <expressão>`
- Muitas variáveis podem ser introduzidas na mesma instrução de atribuição:
  - `minValue, maxValue = 1, 100`
- As instruções de atribuição devem aparecer em uma única linha de código
  - A menos que a linha seja quebrada após uma vírgula, parêntese, chave ou colchete.
  - Outro meio de quebrar uma linha em uma instrução é terminá-la com o símbolo de escape \.

Tipo de dados do Python

- Qualquer variável pode nomear um valor de qualquer tipo.
- As variáveis não são declaradas:
  - Elas simplesmente recebem um valor.
- os nomes dos tipos de dados quase nunca aparecem nos programas Python:
  - No entanto, todos os valores ou objetos têm tipos.

Instruções `import`

- A instrução import torna visíveis para um programa os identificadores de outro módulo.
- Existem várias maneiras de expressar uma instrução import:
  - Simplesmente importar o nome do módulo:

```
import math
```

  - Um segundo estilo de importação incorpora um nome próprio:

```
from math import sqrt
print (sqrt (2))
```

- Você pode importar vários nomes individuais, listando-os:

```
from math import pi, sqrt
print (sqrt (2) * pi)
```

Obtendo ajuda sobre os componentes do programa

- Para acessar a ajuda,
  - basta inserir a chamada de função **`help (<component>)`** no prompt do shell.
- Por exemplo,
  - **`help (abs)`** e **`help (math.sqrt)`** exibem documentação para as funções abs e math.sqrt, respectivamente.
- Chamadas de **`dir(int)`** e **`dir(math)`** listam todas as operações no tipo **`int`** e módulo **`math`**:
  - Você pode então correr help para obter ajuda sobre uma dessas operações.

Instruções de controle

- Uma sequência de instruções é um conjunto de instruções escritas uma atrás da outra.
- Cada instrução em uma sequência deve começar na mesma coluna.

Instruções condicionais (1 de 2)

- As palavras-chave **`if`**, **`elif`**, and **`else`** são significativas
  - assim como o caractere de dois pontos e o recuo.
- A sintaxe da instrução if unidirecional é:

```
if <expressão booleana>:
     <sequência de instruções>
```

- A sintaxe da instrução **`if`** unidirecional é:

```
if <expressão booleana>:
    <sequência de instruções>
else:
    <sequência de instruções>
```

Instruções condicionais (2 de 2)

- A sintaxe da instrução `if` unidirecional é:

```
if <expressão booleana>:
    <sequência de instruções>
elif <expressão booleana>:
    <sequência de instruções>
else:
    <sequência de instruções>
```

- Exemplo:

```
if x > y:
    print ("x is greater than y")
elif x < y:
    print ("x is less than y")
else:
    print ("x is equal to y")
```

Usando `if _name_ == "_main_"`

- O programa discutido anteriormente inclui a definição de uma função **`main`** e a seguinte instrução **`if`**:

```
if _name_ == "_main_":
    main ()
```

- O objetivo dessa instrução i**f** é permitir que o programador execute o módulo como um programa autônomo ou importe-o do shell ou de outro módulo.

Instruções de laços de repetição

- Sintaxe da estrutura da instrução iterativa `while` do Python:

```
while <expressão booleana>:
    <sequência de instruções>
```

- Sintaxe da instrução de laço `for`:

```
for <variável> in <objeto iterável>:
    <sequência de instruções>
```

- Programadores Python geralmente preferem um laço **`for`** para iterar por intervalos definidos ou sequências de valores:
  - Eles usam um laço **`while`** quando a condição de continuação é uma expressão booleana arbitrária.

Strings e suas operações

- Uma string Python é um objeto composto que inclui outros objetos:
  - Seus caracteres.
  - Cada caractere em uma string Python é por si só uma string de um único caractere.
- O tipo de string denominado **`str`**
  - Inclui um grande conjunto de operações.

Operadores

- Operadores de comparação:
  - Quando as strings são comparadas, os pares de caracteres em cada posição nas duas strings são comparados
- O operador + constrói e retorna uma nova string que contém os caracteres dos dois operandos.
- O operador subscrito em sua forma mais simples espera um inteiro no intervalo de 0 ao comprimento da string menos 1:

```
"greater"[0]   # Retorna 'g'
```

- O **operador de fatia** é uma variação do subscrito:

```
"greater"[:] # Retorna "greater"
"greater"[2:]     # Retorna "eater"
"greater"[:2]     # Retorna "gr"
"greater"[2:5]    # Retorna "eat"
```

Formatando strings para saída (1 de 2)

- O Python inclui um mecanismo de formatação geral que permite ao programador especificar larguras de campo para diferentes tipos de dados.
- Exemplo de como justificar à direita e à esquerda a string "four" dentro de uma largura de campo de 6:

```
>>>"%6s" % "four"   # Justifica a direita
'   four'
>>>"%-6s" % "four"  # Justifica a esquerda
'four   '
```

Formatando strings para saída (2 de 2)

- As informações de formato para um valor de dados do tipo **`float`** tem a forma:

  `%<largura do campo>.<precisão>f`

- Esse exemplo mostra a saída de um número de ponto flutuante sem e depois com uma string de formato:

```
>>>salary = 100.00
>>>print("Your salary is $" + str(salary))
Your salary is $100.0
>>>print("Your salary is $%0.2f" % salary)
Your salary is $100.00
```

Objetos e chamadas de método

- A sintaxe de uma chamada de método é:
  - `<objeto>.<nome do método>(<lista de argumentos>)`
- Eis alguns exemplos de chamadas de método em string:

```
"greater".isupper()                 # Retorna False
"greater".upper()                   # Retorna "GREATER"
"greater".startswith("great") # Retorna True
```

- Se você tentar executar um método que um objeto não reconhece,
  - o Python apresentará uma exceção e interromperá o programa
- Para descobrir o conjunto de métodos que um objeto reconhece,
  - você executa a função Python **dir**, no shell Python, com o tipo do objeto como um argumento.

Coleções Python incorporadas e suas operações

- As linguagens de programação modernas incluem vários tipos de coleções, como listas,
  - que permitem ao programador organizar e manipular vários valores de dados de uma vez.
- Esta seção explora as coleções incorporadas no Python.

Listas (1 de 2)

- Lista
  - Uma sequência de zero ou mais objetos Python.
  - Comumente chamados **itens**.
  - Tem uma representação literal.
  - Usa colchetes para incluir itens separados por vírgulas.
- Exemplos:

```
[]                          # Uma lista vazia
["greater"]                 # A list of one string
["greater", "less"]         # A list of two strings
["greater", "less", 10]     # A list of two strings and an int
["greater", ["less", 10]]   # A list with a nested list
```

Lists (2 de 2)

- Os métodos de modificação de lista mais comumente usados:
  - **`append, insert, pop, remove e sort`**
- Examples:

```
testList = []                  # testList é []
testList.append(34)            # testList é [34]
testList.append(22)            # testList é [34, 22]
testList.sort()                # testList é [22, 34]
testList.pop()                 # Retorna 22; testList é [34]
testList.insert(0, 22)         # testList é [22, 34]
testList.insert(1, 55)         # testList é [22, 55, 34]
testList.pop(1)               # Retorna 55; testList é [22, 34]
testList.remove(22)            # testList é [34]
testList.remove(55)            # captura ValueError
```

Tuplas

- Tupla
  - Uma sequência imutável de itens.
  - Literais de tupla colocam os itens entre parênteses.
  - É essencialmente como uma lista sem métodos modificadores.
- Uma tupla com um item ainda deve incluir uma vírgula:

```
>>> (34)
34

>>> (34,)
(34)
```

Laços sobre sequências

- O laço **`for`** é usado para iterar por itens em uma sequência :

```
testList = [67, 100, 22]
for item in testList:
    print(item)
```

- Isso é equivalente a, mas mais simples do que, um laço baseado em índice na lista:

```
testList = [67, 100, 22]
for index in range(len(testList)):
    print(testList[index])
```

Dicionários

- Dicionário
  - Contém zero ou mais entradas.
  - Cada entrada associa uma chave única a um valor
    - As chaves são normalmente strings ou inteiros.
    - Os valores são quaisquer objetos Python.
- Exemplos:

```
{}                                          # Um dicionário vazio
{"name":"Ken"}                               # Uma entrada
{"name":"Ken", "age":67}                     # Duas entradas
{"hobbies":["reading", "running"]}           # Uma entrada, o valor
                                               é uma lista
```

```
>>> for key in {"name":"Ken", "age":67}:
    print(key)
name
age
```

Procurando um valor

- O programador pode pesquisar nas strings, listas, tuplas ou dicionários um determinado valor executando o operador in
  - com o valor e a coleção.
  - Esse operador retorna **`True`** ou **`False`**
- Para dicionários
  - Os métodos **`get`** e **`pop`** podem receber dois argumentos: uma chave e um valor padrão.
  - Uma pesquisa falha retorna o valor padrão.

Correspondência de padrões com coleções

- O subscrito possa ser usado para acessar itens em listas, tuplas e dicionários.
  - Geralmente é mais conveniente acessar vários itens de uma vez por meio de correspondência de padrões.
- Exemplo usando o operador subscrito **`colorTuple`**:

```
rgbTuple = colorTuple[0]
hexString = colorTuple[1]
r = rgbTuple[0]
g = rgbTuple[1]
b = rgbTuple[2]
```