Pedro Maciel Xavier

COMPUTAÇÃO I

MMXXI

PRIMEIRA EDIÇÃO

UFRJ

2021

PREFÁCIO

A intenção deste livro é reunir exercícios interessantes para um curso de introdução à computação na forma de pequenos projetos. Conta com enunciados elaborados e desenvolvimentos longos, a fim de explorar alguns dos conceitos básicos de programação.

Este texto é fruto das minhas atividades de monitoria nos cursos de Computação I (MAB125) na Universidade Federal do Rio de Janeiro (UFRJ) entre os anos de 2018 e 2019. Entre as turmas que trabalhei estão os cursos de Engenharia Naval, Engenharia de Produção e o Bacharelado em Ciências Matemáticas e da Terra (BCMT). Por conta disso, você deve encontrar referências a temas bastante diversos, dadas as inúmeras aplicações distintas que foram exercitadas por mim e pelos professores nestes diferentes contextos.

O curso é pensado para alunos que acabaram de ingressar na Graduação e que possivelmente não tiveram contato com computação anteriormente.

Vamos utilizar como referência a linguagem Python. No entanto, este material não pretende ser um curso ou referência da linguagem. Ela nos dará suporte para enunciar os fundamentos de programação, assim como noções de algoritmos e estruturas de dados. De toda forma, as especificidades da linguagem são discutidas com maior profundidade nos tópicos finais de cada capítulo.

A tipografia foi elaborada no sistema LaTeX e fortemente inspirada no *layout* da terceira edição do livro *Linear Algebra and its Applications*[1], de Gilbert Strang, um dos livros que mais gostei de ler durante a universidade.

Petrópolis, 13 de abril de 2021

NOTAÇÃO

Como referência, a notação matemática segue o padrão encontrado na maioria dos livros e artigos do assunto. No entanto, principalmente na hora de falar de código utilizaremos alguns recursos tipográficos para auxiliar na comunicação.

PYTHON ■

Quando falando sobre os elementos da linguagem, estes aparecerão em fonte monoespaçada, tal como `x`, `y` e `math`. As palavras reservadas (comandos) são destacadas como em `if`, `return` e `del`. Funções, tipos e exceções da biblioteca padrão são mostrados como `list`, `range`, `open`, etc. Por fim, temos as *strings*, `'Oi Mundão'`, `"Adeus, Mundinho"`; e os comentários, `# Voltei Mundão`. Trechos de código serão formatados como a seguir:

```
1 import math
2
3 def f(n):
4     if n == 0 or n == 1:
5         return 1
6     else:
7         return f(n - 1) + f(n - 2)
8
9 print("f(10) =", f(10))
```

Sessões do console interativo aparecem de maneira bastante similar:

```
1 >>> L = [2, 3, 5, 7, 11]
2 >>> L[1]
3 3
4 >>> L.append(13)
5 >>> L
6 [2, 3, 5, 7, 11, 13]
```

Pedro Maciel Xavier

COMPUTAÇÃO I

MMXXI

PRIMEIRA EDIÇÃO

UFRJ

2021

Sumário

Referências Bibliográficas **27**

INTRODUÇÃO

1

TIPOS, VARIÁVEIS, OPERADORES E FUNÇÕES

INTRODUÇÃO ■

É interessante, de fato, iniciar o estudo da computação através das funções. As funções trazem consigo os conceitos de tipos e variáveis de maneira muito natural. De maneira simples, uma função pode ser compreendida como a transformação dos dados de entrada nos dados de saída, cada qual com seus respectivos tipos.

A noção de tipo está também relacionada à ideia de conjunto. Ser de um determinado tipo é pertencer a um conjunto, mais especificamente ao conjunto daquele tipo. Isto fica bem claro quando se pensa nos números, embora seja um conceito adequado a uma miríade de situações.

Em textos de matemática a nível superior, é comum que as funções recebam outros nomes a depender do contexto. Uma nomenclatura um tanto elucidativa é chamar funções de *aplicações*, em inglês, *mappings*. É de bom gosto observar que as funções são *mapas* que levam de um ponto em um conjunto noutro ponto de outro conjunto.

Dito isto, é importante pensar por um tempo no significado de uma função, ter em mente que qualquer processo computacional, por mais complicado que seja, pode ser representado pela aplicação de diferentes funções em sequência. Formalmente, uma transformação será descrita pela composição de diversas transformações mais simples. Isso tudo, porém, no mundo das ideias.

Os *operadores*, por outro lado, são a materialização da computação que desejamos realizar. Operações tratam da maneira explícita como uma tarefa lógica ou aritmética deve ser encadeada. Pode-se dizer, portanto, que as operações estão entre as funções mais simples e mais próximas da realidade. O que acabo de dizer aparece de maneira clara no projeto dos processadores digitais. A arquitetura x86 moderna possui algumas centenas de instruções, dentre operações

aritméticas, de acesso à memória e de controle. Um processador RISC, por sua vez, chega a operar com apenas 34 instruções[2]. O importante é ter em mente que as mais complexas atividades realizadas por um computador são compostas a partir de um conjunto reduzido de operações básicas.

TIPOS

A teoria por trás dos tipos é bastante extensa e é tratada com detalhe muito maior no volume de Computação II[3]. No decorrer dos próximos capítulos vamos tratar das especificidades e casos de uso dos tipos básicos. Por enquanto, vamos nos ater à sua apresentação e ao uso prático inicial.

Vamos começar pelos tipos numéricos. Estão disponíveis os tipos `int`, `float` e `complex`, construídos para representar os números inteiros ($\mathbb{Z}$), reais ($\mathbb{R}$) e complexos ($\mathbb{C}$), respectivamente. A primeira vista, podemos distingui-los pela sintaxe: os números representados em ponto-flutuante[1]podem ser escritos adicionando a parte decimal após o ponto, como em `0.5` e `-1.2` ou através de notação científica, como `6.02e+23` ($6,02 \times 10^{23}$). Os números complexos são simplesmente uma extensão do tipo `float`, onde a parte imaginária é indicada acrescentando um `j` ao final do número. A unidade imaginária `i`, por exemplo, escreve-se `1j`. Tendo em mãos estes três tipos, podemos realizar as operações aritméticas básicas de adição `+`, subtração `-`, multiplicação `*` e divisão `/`; além da exponenciação `**` e da divisão inteira, realizada através dos operadores de quociente `//` e resto `%`.

Tome um tempo para assimilar o comportamento destes operadores, procurando explorar tanto os casos simples quanto os mais problemáticos como a divisão por zero ou operações entre números grandes demais. Atente, em cada operação para os tipos dos termos assim como para o tipo do resultado. A própria sintaxe deve mostrar-se suficiente para identificar o tipo mas você pode sempre contar com a função `type`.

```
1 >>> 1 + 1
2 2
3 >>> 1 + 1.0
4 2.0
5 >>> type(1 + 1)
6 <class 'int'>
7 >>> type(1 + 1.0)
8 <class 'float'>
```

[1]Uma discussão mais aprofundada sobre ponto-flutuante e o padrão IEE 754 se encontra no capítulo 2.

Se tem uma coisa fundamental a ser percebida neste primeiro exemplo é a hierarquia existente entre os tipos `int`, `float` e `complex`, relação que vamos relembrar através da adaptação de um velho diagrama (figura 1.1). Pensando

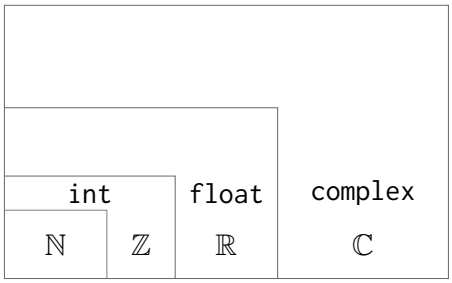


Figura 1.1: Conjuntos numéricos

na correspondência entre os tipos e os conjuntos que representam, podemos afirmar que em uma operação entre números de tipos distintos o resultado será do tipo mais geral, ou seja, do conjunto mais externo. É possível também converter números de um tipo em outro, salvo algumas observações. Números inteiros podem vir a ser representados em ponto-flutuante ou como complexo. Isso ocorre da mesma maneira que um `float`, onde o número original passa a ser a parte real de um novo número, com parte imaginária igual a zero. O caminho contrário apresenta a mesma naturalidade. A conversão de um número decimal para inteiro naturalmente descarta a parte fracionária, de onde restam apenas as unidades. Transformar um número complexo em um tipo simplesmente real seria ambíguo, uma vez que não é óbvio que atitude tomar em relação à parte imaginária do número. Portanto, não é possível converter um número complexo para nenhum dos outros tipos. Resumimos este diálogo na figura 1.2. Por fim, vale ressaltar que o tipo `complex` só é utilizado em aplicações bastante

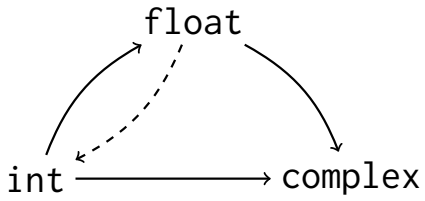


Figura 1.2: Mapa de tipos numéricos

específicas, enquanto o `float` está presente na maioria dos casos de uso. O tipo inteiro estará sempre presente, pois desempenha muitas tarefas importantes

para além da aritmética. Falaremos de maneira mais precisa acerca dos tipos numéricos e suas aplicações no capítulo 2.

Seguindo adiante, vamos tratar de alguns tipos de sequências. Temos disponíveis três tipos básicos de estruturas sequenciais: listas `list`, tuplas `tuple` e *strings* `str`. As duas primeiras são estruturas de uso geral e servem para armazenar de maneira ordenada elementos de qualquer tipo, incluindo os números e até mesmo outras sequências. As *strings* são projetadas para armazenar sequências de caracteres, ou seja, texto.

As listas são delimitadas por colchetes `[]` e seus elementos são separados por vírgulas. As tuplas, por sua vez, utilizam parênteses `()` e tem um comportamento semelhante. As diferenças entre as duas serão esmiuçadas adiante, no capítulo 3. Por fim, as *strings* são compostas pelos caracteres inseridos entre aspas, sejam elas simples `''` ou duplas `""`.

Assim como no caso dos números, é possível transformar listas em tuplas e vice-versa, utilizando as funções `list` e `tuple`. As transformações envolvendo as *strings* e seu tipo `str`, no entanto, possuem um comportamento distinto. Qualquer elemento de qualquer tipo pode ser transformado em *string*, o que consiste em obter uma representação visual para o mesmo. O processo contrário, de transformar *strings* em outros tipos deve ser analisado caso-a-caso.

Uma das principais operações entre sequências é a concatenação `+`. Concatenar, que não é uma palavra muito comum no dia a dia, significa unir duas sequências por justaposição, isto é, respeitando a ordenação inicial. Não é possível, contudo, concatenar cadeias de tipos distintos.

```
>>> [-3, -2, -1] + [1, 2, 3]
[-3, -2, -1, 1, 2, 3]
>>> "Nome" + " " + "Sobrenome"
"Nome Sobrenome"
```

VARIÁVEIS E ATRIBUIÇÃO

```
>>> x = 0
>>> y = 1
>>> x + y
1
```

FUNÇÕES

Para definir uma função vamos utilizar o comando `def`, seguido do nome da função, dos seus parâmetros e, por fim, o código que deve executar.

```
>>> def f(x, y):
...     return x + y
...
>>> z = f(2, 3)
>>> z
5
```

1.1 Cálculo I - Limites

O limite ℓ de uma função $f(x)$ no ponto a é o valor que a função assume conforme x se aproxima de a. Escrevemos:

$$\ell = \lim_{x \to a} f(x)$$

Quando a função não apresentar nenhuma descontinuidade ao redor do ponto a, podemos afirmar com tranquilidade que $\ell = f(a)$. Uma maneira simples de aproximar os limites laterais no computador é calcular

$$\lim_{x \to a^-} f(x) \approx f(a - \epsilon) \text{ e } \lim_{x \to a^+} f(x) \approx f(a + \epsilon)$$

escolhendo um valor suficientemente pequeno para ϵ.

```
def lim(f, a, e=0.01):
    """Limite da função f no ponto a."""
    e = 1E-4 # 0.0001
    return (f(a - e) + f(a + e)) / 2.0
```

Proposta

Faça uma função `lim(f, a)` para calcular o limite da função `f` no ponto `a`. Leve em consideração a existência do limite.

1.2 Música I - As notas e os sons

Cada nota musical corresponde a uma frequência distinta (em Hz). Tomando o lá central (A4) como referência, em 440Hz, podemos calcular a frequência das outras notas com base na distância relativa a essa nota.

$$f(n) = 440 \times 2^{(n/12)}$$

Na tabela abaixo, vemos as notas musicais, seus símbolos, e a distância em semitons[2] para o lá central (A4).

Símbolo		F3	G3	A4	B4	C4	D4	E4	F4	G4	
Nome	...	fá	sol	lá	si	dó	ré	mi	fá	sol	...
Semitons		-4	-2	0	2	3	5	7	8	10	

Você pode notar que a cada 12 semitons, a nota se repete com o dobro da frequência. Chamamos este intervalo entre notas de oitava. Na notação acima, a cada letra indica uma nota diferente, enquanto o número diz a oitava em que ela se encontra.

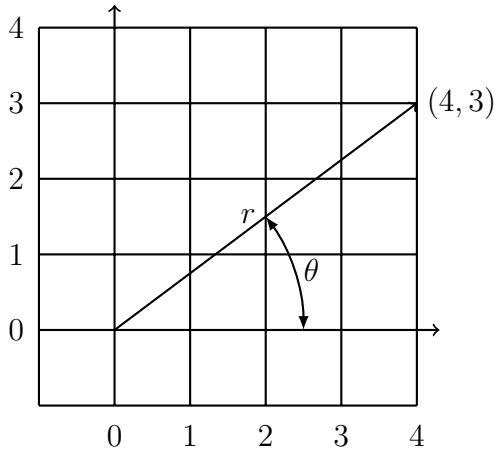
Proposta

Faça uma função `f(n)` que retorne a frequência em Hertz de uma nota que se encontra a n semitons de distância do lá central. Arredonde o resultado para o número inteiro mais próximo usando a função `round(numero, dígitos)`.

```
>>> f(0), f(2), f(3)
(440, 494, 523)
>>> f(12) # frequência dobra ...
880
```

1.3 Coordenadas polares

Estamos acostumados a pensar em coordenadas cartesianas na hora de descrever a geometria de um determinado objeto. No entanto, o sistema de coordenadas deve ser escolhido conforme o cenário em que se está trabalhando.



[2]Dois semitons equivalem a um tom. No violão, cada casa de uma mesma corda está a um semitom da casa adjacente. No piano, quando há uma tecla preta entre as brancas, há uma distância de um tom entre elas. Quando a tecla preta não está, a distância é de meio tom, ou um semitom.

O ponto $(4, 3)$, quando escrito em coordenadas polares, nos dá:

$$r = \sqrt{4^2 + 3^2} = \sqrt{16 + 9} = \sqrt{25} = 5$$
$$\theta = \arctan \frac{3}{4} = 0.6435 \text{ rad} \approx 36.87^\circ$$

Proposta
Construa duas funções: `polar(x, y)` levará um ponto em coordenadas cartesianas (x, y) para a forma polar (r, θ) e `cart(r, theta)`, que fará o caminho contrário.

```
>>> import math
>>> polar(-1, 0)
(1.0, 3.141592653589793)
>>> cart(2, math.pi)
(-2.0, 0.0)
```

CONDICIONAIS ■

Uma das ferramentas mais simples e poderosas dos computadores é a sua capacidade de tomar diferentes atitudes mediante uma condição.

REPETIÇÃO ■

1.1 Sequência de *Collatz*

A sequência de *Collatz* é obtida aplicando sucessivamente a função

$$f(n) = \begin{cases} 3\,n + 1, & \text{se } n \text{ for ímpar} \\ n \div 2, & \text{se } n \text{ for par} \end{cases}$$

Por exemplo, começamos com $n = 26$. Após sucessivas aplicações temos:

$$26 \to 13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

Isso nos dá uma sequência com 11 números. 40 é maior que 26, mas sua sequência só teria 9 números.

Ainda não se sabe se todos os números induzem uma sequência que termina em 1. No entanto, até agora não foi encontrado um número sequer em que isso não tenha acontecido!

Proposta
Faça uma função que calcule o comprimento da sequência gerada a partir de um número natural n qualquer.

```
>>> collatz(26)
11
>>> collatz(40)
9
>>> collatz(1)
1
```

1.2 *Chatbot* I

Existe uma história corrente de que o grande motor da inteligência artificial em escala industrial é o uso indiscriminado do comando `if` e seus derivados. Não é difícil pensar que através de um número suficientemente grande de condições seja possível abarcar uma grande variedade de comportamentos.

Proposta

Construa uma função chamada `chatbot`

RECURSIVIDADE ■

PARÂMETROS ■

DECORADORES ■

2

NÚMEROS

Este capítulo é opcional.

INTEIROS

Além de fazer contas

PONTO-FLUTUANTE

Agora é importante falar um pouco mais sobre o nosso antigo conhecido, o `float`. Existe um padrão industrial, o **IEE 754**, que especifica como deve ser feita a representação numérica em ponto-flutuante.

O padrão afirma que o `float` deve possuir três partes: o *bit* do sinal, o expoente, e a mantissa (ou fração). O número é positivo quando o primeiro *bit* é 0, e negativo quando é 1. O tamanho do expoente e da mantissa depende de diversos fatores, como o projeto do processador, a linguagem de programação e até mesmo das decisões do programador.

```
0 01111000 01000000000000000000000
```

Figura 2.1: O número 0.15625 no padrão IEE 754, precisão simples.

É possível fazer uma analogia com a notação científica. Vamos observar o número de moléculas em um mol de uma substância qualquer:

$$1\text{mol} = 6.02 \times 10^{23}$$

De maneira imediata, podemos separar esse número em três partes também: o sinal (+), o expoente (23), e a mantissa (6.02).

É importante entender que as representações em ponto-flutuante tem suas limitações. Algumas delas são:

- Somar ou subtrair números muito grandes com números muitos pequenos resultará numa *adição ou subtração catastrófica*, isto é, o número menor pode acabar sendo ignorado durante a operação.
- A mantissa é composta pela soma de potências de $\frac{1}{2}$, portanto números que não são resultado de somas finitas dessas potências vão apresentar erros de representação.

COMPLEXOS ■

VETORES ■

3

SEQUÊNCIAS E DICIONÁRIOS

TIPOS MUTÁVEIS E IMUTÁVEIS ■

TUPLAS ■

LISTAS ■

3.1 Crivo

O Eratóstenes foi um cara legal que viveu em 200 a.C., bolou um crivo e calculou a curvatura da terra.

Figura 3.1: Eratóstenes de Cirene

Um *crivo* é uma forma de saber quem são os números primos até um determinado limite n. O Crivo de Eratóstenes funciona de forma bem simples:

1. Escrevemos em uma tabela todos os números de 0 até n.
2. Riscamos o 0 e o 1. Começamos do 2.
3. Se um número não estiver riscado, riscamos todos os múltiplos deste, menos o próprio.
4. Andamos para o próximo número e repetimos a etapa anterior.

Uma maneira interessante de fazer isso é criando uma lista de tamanho $n + 1$, ou seja, cujos índices vão de 0 até n.

Proposta
Implemente o crivo de Eratóstenes e, em seguida, elabore versões melhoradas do algoritmo, até onde conseguir.

DICIONÁRIOS ■

4

STRINGS E TEXTO

De modo geral, *strings* são o principal tipo básico presente nas linguagens de programação para representar textos. Os literais que representam texto em Python são aqueles compreendidos entre aspas simples ' ou duplas ". Não existe preferência absoluta por uma ou outra, fica a seu critério qual utilizar. No entanto, uma opção pode ser feita pela praticidade ao trabalhar com texto que contenha algum destes caracteres.

```
1 >>> historia = "Era uma vez..."
2 >>> print(historia)
3 Era uma vez...
```

CODIFICAÇÃO

Vamos discorrer rapidamente sobre os aspectos fundamentais da codificação de texto em computadores. As cadeias de caracteres ou simplesmente *strings* nada mais são do que sequências que comportam um tipo específico de dado, um número a representar determinado símbolo.

Existem diversos "alfabetos"para a codificação de caracteres, sendo a *Tabela ASCII*[1] desenvolvida na década de 1960 uma das precursoras dentre estas tecnologias. De fato, os padrões de codificação amplamente utilizados na atualidade configuram extensões do seu conjunto inicial de 128 símbolos.

Como um byte possui 8 bits e somente 7 são necessários para representar todas as entradas da tabela, sobra um bit para usos outros, a depender da implementação em voga. O padrão Unicode[4], particularmente em sua especificação *UTF-8*[2], utiliza este bit adicional para indicar se serão necessários bytes adicionais na codificação ou não. Precisamente, o último bit indica se determinado caractere está presente nos 128 itens da *Tabela ASCII*.

[1] *American Standard Code for Information Interchange.*
[2] *Unicode Transformation Format* em 8 bits.

O *UTF-8* se encontra em cerca de 97% das páginas da internet[5]. O padrão utiliza de um a quatro bytes, o que permite representar mais de 2000 símbolos com apenas 2 bytes, o suficiente para cobrir todos os alfabetos latinos, do grego ao cirílico. Com os bytes extras codifica-se o chinês, tradicional e moderno, o coreano, o japonês, o sânscrito e a maioria das línguas orientais. Com quatro bytes é possível descrever símbolos cuneiformes, hieróglifos e sua encarnação contemporânea: os emoji.

4.1 Transformações ASCII

Da maneira como foi construída, a Tabela ASCII nos permite realizar transformações comuns ao texto de uma maneira bem simples. No Python, é proposta uma interface com a tabela através das funções `ord` e `chr`. `ord` recebe um símbolo e diz qual é sua posição na tabela de codificação através de um número inteiro. Já `chr` nos diz qual caractere ocupa a posição dada pelo inteiro que recebe. Ou seja, uma é inversa da outra, de forma que `chr(ord(c)) = c` para qualquer símbolo `c`. Naturalmente, também vale a recíproca.

Proposta

Procure entender a disposição das letras de a a z, maiúsculas e minúsculas, e utilizando as funções `ord` e `chr` defina um par de funções que transforme textos quaisquer em sequências de caixa alta e caixa baixa, respectivamente.

OPERAÇÕES EM TEXTO ■

FORMATAÇÃO E INTERPOLAÇÃO ■

EXPRESSÕES REGULARES ■

5

DICIONÁRIOS E CONJUNTOS

HASH ■

6

ENTRADA E SAÍDA

INTERAÇÃO COM O USUÁRIO ■

ARQUIVOS ■

A LINHA DE COMANDOS ■

1

SÉRIES DE EXERCÍCIOS

CÁLCULO ■

I Limites

II Derivadas

III Derivadas parciais

IV Integrais

MÚSICA ■

2

GLOSSÁRIO

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Gilbert Strang. *Linear Algebra and its Applications*. Harcourt Brace, Massachusetts Institute of Technology, 1988.

[2] Advanced RISC Machines Ltd. *ARM-7TDMI Data Sheet*, 1995.

[3] Pedro M. Xavier. *Computação II*. ?, Universidade Federal do Rio de Janeiro, 2021.

[4] The Unicode Consortium. *The Unicode Standard — Version 13.0.*, 2020.

[5] Usage of character encodings broken down by ranking.