# 06. Composição de Funções

**Princípios de Engenharia de Software (Texto em Elaboração)**

*Italo S. Vega*
italo@pucsp.br

Faculdade de Estudos Interdisciplinares (FACEI)


Pontifícia Universidade Católica de São Paulo

# Sumário


Leslie Lamport, *Specifying Systems* — Thinking clearly is hard; we can use all the help we can get.

# 6 Composição de Funções

 — Consigo **definir** a composição de funções, **observar** a sua implementação em Python e **construir** soluções compondo funções.

## 6.1 Operador de Composição

Funções constituem um importante conceito na área computacional. Outro conceito fundamental é o de composição. Trata-se de uma operação que produz funções complexas a partir de outras mais simples [Ince (1992); p. 120]. Sejam as funções $f_1$ e $f_2$. Definimos a sua composição, representada pelo símbolo $\circ$, como:

$$(f_2 \circ f_1)(x) \triangleq f_2(f_1(x))$$

 — A notação $f_2 \circ f_1$ pode ser lida como $f_2$ sobre $f_1$ ou, ainda, $f_2$ **depois** de $f_1$.

---

### 6.1.1 Questão: Mínimos Quadrados

*Contexto* Ao estudar modelos de previsão de dados, Fubã deparou-se com o conceito de regressão linear. Segundo o seu entendimento, a ideia central é de "prever dados do futuro com base em dados do passado" usando funções com a forma, considerando $\mathbf{x}$ uma lista de números:

$$f(\mathbf{x}) = a\mathbf{x} + b$$

O cálculo dos coeficientes $a$ e $b$ envolve várias operações, inclusive:

$$g \triangleq \{\ p, q : \mathsf{List} \mid (\#p = \#q) \land (i \in 1..\#p) \bullet i \mapsto p_i - q_i\ \}$$
$$s \triangleq \lambda v : \mathsf{List} \bullet \Sigma v_i^2$$

```
# CENÁRIO de composição funcional
def g(p, q) :
  if len(p) == len(q) :
    return [p[i]-q[i] for i in range(len(p))]
  raise Exception('_|_')

def s(v) :
  return sum([(vi*vi) for vi in v])
# uso da soma de listas
y = [1,2,3]; z = [2,3,4]
w = s(g(y,z))
w
```

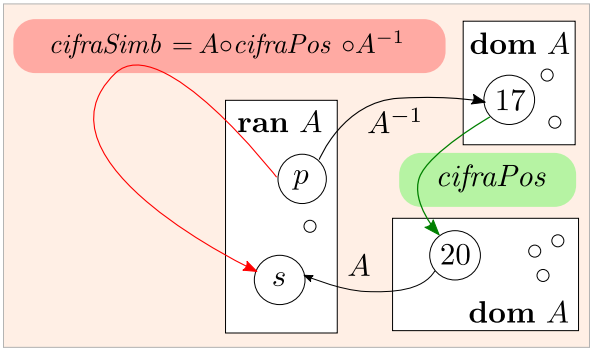
*Enunciado* Assinale a alternativa contendo a afirmação verdadeira:

1. $[\![\mathsf{s(g(y,z))}]\!]_{logica} = 5$
2. $[\![\mathsf{s(g([1,2],z))}]\!]_{logica} \neq \bot$
3. $[\![\mathsf{w}]\!]_{logica} = (s \circ g)(y, z)$
4. $[\![\mathsf{s(g([1,2],[1,2]))}]\!]_{logica} = \langle 1, 2 \rangle$

---

Retornando ao problema da Cifra de César, Espec sugere uma lógica para cifrar cada letra do alfabeto. A sua ideia é de compor três funções: $A$, *cifraPos* e $A^{-1}$, criando outra, chamada *cifraSimb*.

 — Considere o diagrama da função *cifraSimb*:




Uma vez definida esta função, realiza-se a cifragem de símbolos da seguinte maneira:

$$cifraSimb(p) = (A \circ cifraPos \circ A^{-1})\ (p) = s$$

Além disso,

$$cifraSimb(y) = (A \circ cifraPos \circ A^{-1})\ (y) = a$$

 — E qual seria a implementação desta função em Python?

## 6.2 Resumo

1. Composição de funções produz funções mais complexas.
2. Representa-se a composição pelo símbolo $\circ$.

## 6.3 Exercício

### 6.3.1 Sequências em Python

*Contexto* Os seguintes elementos do modelo Cifra de César correspondem ao alfabeto e ao seu comprimento, respectivamente:

$$A \triangleq \langle \square\ , a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, ç\ \rangle$$
$$K \triangleq \#A$$

As seguintes interpretações devem ser respeitadas:

$$[\![A]\!]_{Python} = \texttt{def A(s) : ...}$$
$$[\![K]\!]_{Python} = \texttt{K = ...}$$

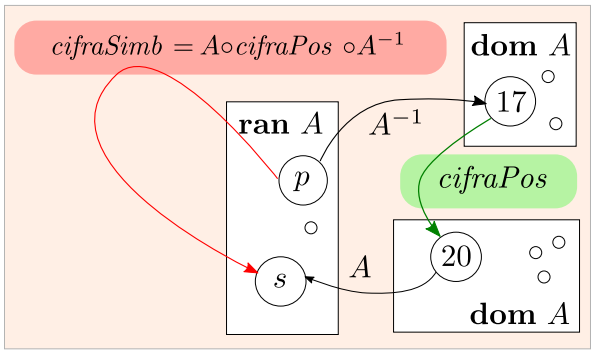
*Enunciado*

1. Implemente $A$, $A^{-1}$ e $K$ em Python.
2. Escreva as seguintes imagens usando o código do item anterior:

a. $A(1)$
b. $A(15)$
c. $A(28)$
d. $A^{-1}(\square)$

### 6.3.2 Dominios e Imagens

*Contexto* Considere as funções do seguinte diagrama:




*Enunciado* De acordo com o diagrama é correto afirmar-se que: **ran** *cifraPos* corresponde à

1. **dom** $A$
2. **ran** $A$
3. $A^{-1}$
4. $A$

### 6.3.3 Função como `dict`

*Contexto* Fubã decide investigar melhor o modelo de cifragem e implementações alternativas. Uma ideia é usar um tipo dicionário[1] para traduzir a função $A$:

```
# CENÁRIO de implementação de função por um "dict"
def f (x) :
    return {
        1 : ' ', 2 : 'a', 3 : 'b', 4 : 'c', 5 : 'd', 6 : 'e',
        7 : 'f', 8 : 'g', 9 : 'h', 10: 'i', 11: 'j', 12: 'k',
        13: 'l', 14: 'm', 15: 'n', 16: 'o', 17: 'p', 18: 'q',
        19: 'r', 20: 's', 21: 't', 22: 'u', 23: 'v', 24: 'w',
        25: 'x', 26: 'y', 27: 'z', 28: 'ç'
    } [x]
```

*Enunciado* A partir desta definição, calcular as imagens:

```
# CENÁRIO de chamada do método "f"
f (2) == ?  # <-- (A)
f (24) == ? # <-- (B)
f (27) == ? # <-- (C)
```

### 6.3.4 Função como `string`

*Contexto* Outra possível tradução da função $A$ segue[2]:

```
# CENÁRIO de implementação de função por uma imagem "String"
def A (x) :
    imagem = ' abcdefghijklmnopqrstuvwxyzç'
    return { i + 1: s for i, s in enumerate (imagem) } [x]
```

Como no caso anterior, também obtenho a mesma imagem para o argumento 2:

*Enunciado* A partir desta definição, calcular as imagens:

```
# CENÁRIO de chamada do método "f"
f (2) == ?  # <-- (A)
f (24) == ? # <-- (B)
f (27) == ? # <-- (C)
```

[1] Código modificado a partir da sugestão do estudante Vitor Couto.
[2] Código modificado a partir da sugestão do estudante Pedro Lucas.

### 6.3.5 Função como `array`

*Contexto* Ao caminhar para casa, Fubã lembrou-se de uma conversa que teve com outro colega[3] a respeito da tradução para Python da função $A$. Nesta versão, ele utilizou-se de dois valores do tipo *array* em Python:

```
# CENÁRIO de implementação de função usando "array"
def f(x) :
  dom = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
    11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
    21, 22, 23, 24, 25, 26, 27, 28]
  ran = [' ','a','b','c','d','e','f',
    'g','h','i','j','k','l','m','n','o',
    'p','q','r','s','t','u','v','w','x','y','z', 'ç']
  return { dom[i]: ran[i] for i in range (len (dom)) } [x]
```

*Enunciado* A partir desta definição, calcule as imagens:

```
# CENÁRIO de chamada do método "f"
f (2) == ?  # <-- (A)
f (24) == ? # <-- (B)
f (27) == ? # <-- (C)
```

### 6.3.6 Implementação Monolítica

*Contexto* Com a ajuda de outro colega[4], Fubã conseguiu o seguinte:

```
# CENÁRIO de implementação da Cifra de César
# Definição do método "cesar"
def cesar(entrada, k) :
  A = 'abcdefghijklmnopqrstuvwyz'
  saida = ''
  for s in entrada :
    p = A.find (s)
    if p == -1 :
      # NÃO cifro o símbolo "s"
      saida += s
    else:
      # cifro o símbolo "s"
      pc = (p + k) % len (A)
      saida += A [pc]
  return saida
# Lógica de CONTROLE e INTERAÇÃO
entrada = 'puc cdia es'
saida = cesar (entrada, k=2)
print (saida)
```

[3] Código modificado a partir da sugestão do estudante Kevin.
[4] Código modificado mas inspirado no raciocínio de Paulo Restaino em 2021.

*Enunciado* Assinale a alternativa contendo um absurdo lógico:

1. `cesar(' ') == 'n'`
2. `cesar('a') == 'c'`
3. `cesar('b') == 'd'`
4. `cesar('c') == 'e'`

# Referências

Ince, D. C. (1992). *An Introduction to Discrete Mathematics, Formal System Specification, and Z* (2º ed). Oxford University Press.