Ejercicios de programación con Python

José A. Alonso Jiménez

Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, 7 de septiembre de 2022

Índice general

Introducción

Este libro es una colección de relaciones de ejercicios de programación con Python. Está basada en la de Ejercicios de programación funcional con Haskell que se ha usado en el curso de Informática (de 1º del Grado en Matemáticas de la Universidad de Sevilla).

Las relaciones están ordenadas según los temas del curso.

El código de los ejercicios de encuentra en el repositorio I1M–Ejercicios–Python [1] de GitHub.

[1] https://github.com/jaalonso/Ejercicios-Python

Parte I

Introducción a la programación con Python

Capítulo 1

Definiciones elementales de funciones

1.1. Definiciones por composición sobre números, listas y booleanos

```
# En esta relación se plantean ejercicios con definiciones de funciones
# por composición sobre números, listas y booleanos.

# ---------------------------------------------------------------------
# Cabecera
# ---------------------------------------------------------------------

from math import pi
from typing import TypeVar
from hypothesis import given, strategies as st
A = TypeVar('A')

# ---------------------------------------------------------------------
# Ejercicio 1. Definir la función
#    media3 : (float, float, float) -> float
# tal que (media3 x y z) es la media aritmética de los números x, y y
# z. Por ejemplo,
#    media3(1, 3, 8)   ==  4.0
#    media3(-1, 0, 7)  ==  2.0
#    media3(-3, 0, 3)  ==  0.0
# ---------------------------------------------------------------------
```

```
def media3(x: float, y: float, z: float) -> float:
    return (x + y + z)/3

# ---------------------------------------------------------------------
# Ejercicio 2. Definir la función
#    sumaMonedas : (int, int, int, int, int) -> int
# tal que sumaMonedas(a, b, c, d, e) es la suma de los euros
# correspondientes a a monedas de 1 euro, b de 2 euros, c de 5 euros, d
# 10 euros y e de 20 euros. Por ejemplo,
#    sumaMonedas(0, 0, 0, 0, 1)  ==  20
#    sumaMonedas(0, 0, 8, 0, 3)  == 100
#    sumaMonedas(1, 1, 1, 1, 1)  ==  38
# ---------------------------------------------------------------------

def sumaMonedas(a: int, b: int, c: int, d: int, e: int) -> int:
    return 1 * a + 2 * b + 5 * c + 10 * d + 20 * e

# ---------------------------------------------------------------------
# Ejercicio 3. Definir la función
#    volumenEsfera : (float) -> float
# tal que volumenEsfera(r) es el volumen de la esfera de radio r. Por
# ejemplo,
#    volumenEsfera(10)  ==  4188.790204786391
# ---------------------------------------------------------------------

def volumenEsfera(r: float) -> float:
    return (4 / 3) * pi * r ** 3

# ---------------------------------------------------------------------
# Ejercicio 4. Definir la función
#    areaDeCoronaCircular : (float, float) -> float
# tal que areaDeCoronaCircular(r1, r2) es el área de una corona
# circular de radio interior r1 y radio exterior r2. Por ejemplo,
#    areaDeCoronaCircular(1, 2) == 9.42477796076938
#    areaDeCoronaCircular(2, 5) == 65.97344572538566
#    areaDeCoronaCircular(3, 5) == 50.26548245743669
# ---------------------------------------------------------------------

def areaDeCoronaCircular(r1: float, r2: float) -> float:
```

```
    return pi * (r2 ** 2 - r1 ** 2)

# ---------------------------------------------------------------------
# Ejercicio 5. Definir la función
#    ultimoDigito : (int) -> int
# tal que ultimoDigito(x) es el último dígito del número x. Por
# ejemplo,
#    ultimoDigito(325) == 5
# ---------------------------------------------------------------------

def ultimoDigito(x: int) -> int:
    return x % 10

# ---------------------------------------------------------------------
# Ejercicio 6. Definir la función
#    maxTres : (int, int, int) -> int
# tal que maxTres(x, y, z) es el máximo de x, y y z. Por ejemplo,
#    maxTres(6, 2, 4)  ==  6
#    maxTres(6, 7, 4)  ==  7
#    maxTres(6, 7, 9)  ==  9
# ---------------------------------------------------------------------

def maxTres(x: int, y: int, z: int) -> int:
    return max(x, max(y, z))

# ---------------------------------------------------------------------
# Ejercicio 7. Definir la función
#    rota1 : (List[A]) -> List[A]
# tal que rota1(xs) es la lista obtenida poniendo el primer elemento de
# xs al final de la lista. Por ejemplo,
#    rota1([3, 2, 5, 7])    == [2, 5, 7, 3]
#    rota1(['a', 'b', 'c']) == ['b', 'c', 'a']
# ---------------------------------------------------------------------

# 1ª solución
def rota1a(xs: list[A]) -> list[A]:
    if xs == []:
        return []
    return xs[1:] + [xs[0]]
```

```python
# 2ª solución
def rota1b(xs: list[A]) -> list[A]:
    if xs == []:
        return []
    ys = xs[1:]
    ys.append(xs[0])
    return ys

# 3ª solución
def rota1c(xs: list[A]) -> list[A]:
    if xs == []:
        return []
    y, *ys = xs
    return ys + [y]

# La equivalencia de las definiciones es
@given(st.lists(st.integers()))
def test_rota1(xs: list[int]) -> None:
    assert rota1a(xs) == rota1b(xs) == rota1c(xs)

# La comprobación está al final

# ---------------------------------------------------------------------
# Ejercicio 8. Definir la función
#    rota : (int, List[A]) -> List[A]
# tal que rota(n, xs) es la lista obtenida poniendo los n primeros
# elementos de xs al final de la lista. Por ejemplo,
#    rota(1, [3, 2, 5, 7]) == [2, 5, 7, 3]
#    rota(2, [3, 2, 5, 7]) == [5, 7, 3, 2]
#    rota(3, [3, 2, 5, 7]) == [7, 3, 2, 5]
# ---------------------------------------------------------------------

def rota(n: int, xs: list[A]) -> list[A]:
    return xs[n:] + xs[:n]

# ---------------------------------------------------------------------
# Ejercicio 9. Definir la función
#    rango : (List[int]) -> List[int]
# tal que rango(xs) es la lista formada por el menor y mayor elemento
# de xs.
```

```
#    rango([3, 2, 7, 5]) == [2, 7]
# ---------------------------------------------------------------------

def rango(xs: list[int]) -> list[int]:
    return [min(xs), max(xs)]

# ---------------------------------------------------------------------
# Ejercicio 10. Definir la función
#    palindromo : (List[A]) -> bool
# tal que palindromo(xs) se verifica si xs es un palíndromo; es decir,
# es lo mismo leer xs de izquierda a derecha que de derecha a
# izquierda. Por ejemplo,
#    palindromo([3, 2, 5, 2, 3])     ==  True
#    palindromo([3, 2, 5, 6, 2, 3])  ==  False
# ---------------------------------------------------------------------

def palindromo(xs: list[A]) -> bool:
    return xs == list(reversed(xs))

# ---------------------------------------------------------------------
# Ejercicio 11. Definir la función
#    interior : (list[A]) -> list[A]
# tal que interior(xs) es la lista obtenida eliminando los extremos de
# la lista xs. Por ejemplo,
#    interior([2, 5, 3, 7, 3])  ==  [5, 3, 7]
# ---------------------------------------------------------------------

# 1ª solución
def interior1(xs: list[A]) -> list[A]:
    return xs[1:][:-1]

# 2ª solución
def interior2(xs: list[A]) -> list[A]:
    return xs[1:-1]

# La propiedad de equivalencia es
@given(st.lists(st.integers()))
def test_interior(xs):
    assert interior1(xs) == interior2(xs)
```

```
# La comprobación está al final

# ---------------------------------------------------------------------
# Definir la función
#    finales : (int, list[A]) -> list[A]
# tal que finales(n, xs) es la lista formada por los n finales
# elementos de xs. Por ejemplo,
#    finales(3, [2, 5, 4, 7, 9, 6])  ==  [7, 9, 6]
# ---------------------------------------------------------------------

# 1ª definición
def finales1(n: int, xs: list[A]) -> list[A]:
    if len(xs) <= n:
        return xs
    return xs[len(xs) - n:]

# 2ª definición
def finales2(n: int, xs: list[A]) -> list[A]:
    if n == 0:
        return []
    return xs[-n:]

# 3ª definición
def finales3(n: int, xs: list[A]) -> list[A]:
    ys = list(reversed(xs))
    return list(reversed(ys[:n]))

# La propiedad de equivalencia es
@given(st.integers(min_value=0), st.lists(st.integers()))
def test_equiv_finales(n, xs):
    assert finales1(n, xs) == finales2(n, xs) == finales3(n, xs)

# La comprobación está al final.

# ---------------------------------------------------------------------
# Ejercicio 13. Definir la función
#    segmento : (int, int, list[A]) -> list[A]
# tal que segmento(m, n, xs) es la lista de los elementos de xs
# comprendidos entre las posiciones m y n. Por ejemplo,
#    segmento(3, 4, [3, 4, 1, 2, 7, 9, 0])  ==  [1, 2]
```

```
#    segmento(3, 5, [3, 4, 1, 2, 7, 9, 0])  ==  [1, 2, 7]
#    segmento(5, 3, [3, 4, 1, 2, 7, 9, 0])  ==  []
# ---------------------------------------------------------------------

# 1ª definición
def segmento1(m: int, n: int, xs: list[A]) -> list[A]:
    ys = xs[:n]
    return ys[m - 1:]

# 2ª definición
def segmento2(m: int, n: int, xs: list[A]) -> list[A]:
    return xs[m-1:n]

# La propiedad de equivalencia es
@given(st.integers(), st.integers(), st.lists(st.integers()))
def test_equiv_segmento(m, n, xs):
    assert segmento1(m, n, xs) == segmento2(m, n, xs)

# La comprobación está al final.

# ---------------------------------------------------------------------
# Ejercicio 14. Definir la función
#    extremos : (int, list[A]) -> list[A]
# tal que extremos(n, xs) es la lista formada por los n primeros
# elementos de xs y los n finales elementos de xs. Por ejemplo,
#    extremos(3, [2, 6, 7, 1, 2, 4, 5, 8, 9, 2, 3])  ==  [2, 6, 7, 9, 2, 3]
# ---------------------------------------------------------------------

def extremos(n: int, xs: list[A]) -> list[A]:
    return xs[:n] + xs[-n:]

# ---------------------------------------------------------------------
# Ejercicio 15. Definir la función
#    mediano : (int, int, int) -> int
# tal que mediano(x, y, z) es el número mediano de los tres números x, y
# y z. Por ejemplo,
#    mediano(3, 2, 5) == 3
#    mediano(2, 4, 5) == 4
#    mediano(2, 6, 5) == 5
#    mediano(2, 6, 6) == 6
```

```
# ---------------------------------------------------------------------

def mediano(x: int, y: int, z: int) -> int:
    return x + y + z - min([x, y, z]) - max([x, y, z])

# ---------------------------------------------------------------------
# Ejercicio 16. Definir la función
#    tresIguales : (int, int, int) -> bool
# tal que tresIguales(x, y, z) se verifica si los elementos x, y y z son
# iguales. Por ejemplo,
#    tresIguales(4, 4, 4)  ==  True
#    tresIguales(4, 3, 4)  ==  False
# ---------------------------------------------------------------------

# 1ª solución
def tresIguales1(x: int, y: int, z: int) -> bool:
    return x == y and y == z

# 2ª solución
def tresIguales2(x: int, y: int, z: int) -> bool:
    return x == y == z

# La propiedad de equivalencia es
@given(st.integers(), st.integers(), st.integers())
def test_equiv_tresIguales(x, y, z):
    assert tresIguales1(x, y, z) == tresIguales2(x, y, z)

# La comprobación está al final.

# ---------------------------------------------------------------------
# Ejercicio 17. Definir la función
#    tresDiferentes : (int, int, int) -> bool
# tal que tresDiferentes(x, y, z) se verifica si los elementos x, y y z
# son distintos. Por ejemplo,
#    tresDiferentes(3, 5, 2)  ==  True
#    tresDiferentes(3, 5, 3)  ==  False
# ---------------------------------------------------------------------

def tresDiferentes(x: int, y: int, z: int) -> bool:
    return x != y and x != z and y != z
```

```python
# ---------------------------------------------------------------------
# Ejercicio 18. Definir la función
#    cuatroIguales : (int, int, int, int) -> bool
# tal que cuatroIguales(x,y,z,u) se verifica si los elementos x, y, z y
# u son iguales. Por ejemplo,
#    cuatroIguales(5, 5, 5, 5) == True
#    cuatroIguales(5, 5, 4, 5) == False
# ---------------------------------------------------------------------

# 1ª solución
def cuatroIguales1(x: int, y: int, z: int, u: int) -> bool:
    return x == y and tresIguales1(y, z, u)

# 2ª solución
def cuatroIguales2(x: int, y: int, z: int, u: int) -> bool:
    return x == y == z == u

# La propiedad de equivalencia es
@given(st.integers(), st.integers(), st.integers(), st.integers())
def test_equiv_cuatroIguales(x, y, z, u):
    assert cuatroIguales1(x, y, z, u) == cuatroIguales2(x, y, z, u)

# La comprobación está al final.

# La comprobación de las propiedades es
#    src> poetry run pytest -q definiciones_por_composicion.py
#    6 passed in 0.81s
```