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

```
# ---------------------------------------------------------------------
# Introducción                                                       --
# ---------------------------------------------------------------------

# 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]]

# 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()))
```

```python
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

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

1.2. Definiciones con condicionales, guardas o patrones

```
# ---------------------------------------------------------------------
# Introducción                                                      --
# ---------------------------------------------------------------------
```

```python
# En esta relación se presentan ejercicios con definiciones elementales
# (no recursivas) de funciones que usan condicionales, guardas o
# patrones.
#
# Estos ejercicios se corresponden con el tema 4 del curso cuyas apuntes
# se encuentran en https://bit.ly/3x1ze0u

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

from math import gcd, sqrt
from typing import TypeVar
from hypothesis import assume, given, strategies as st
A = TypeVar('A')
B = TypeVar('B')

# ---------------------------------------------------------------------
# Ejercicio 1. Definir la función
#    divisionSegura : (float, float) -> float
# tal que divisionSegura(x, y) es x/y si y no es cero y 9999 en caso
# contrario. Por ejemplo,
#    divisionSegura(7, 2)  ==  3.5
#    divisionSegura(7, 0)  ==  9999.0
# ---------------------------------------------------------------------

# 1ª definición
def divisionSegura1(x: float, y: float) -> float:
    if y == 0:
        return 9999.0
    return x/y

# 2ª definición
def divisionSegura2(x: float, y: float) -> float:
    match y:
        case 0:
            return 9999.0
        case _:
            return x/y
```

```
# La propiedad de equivalencia es
@given(st.floats(allow_nan=False, allow_infinity=False),
       st.floats(allow_nan=False, allow_infinity=False))
def test_equiv_divisionSegura(x, y):
    assert divisionSegura1(x, y) == divisionSegura2(x, y)

# La comprobación está al final de la relación.

# ---------------------------------------------------------------------
# Ejercicio 2. La disyunción excluyente de dos fórmulas se verifica si
# una es verdadera y la otra es falsa. Su tabla de verdad es
#    x     | y     | xor x y
#    ------+-------+---------
#    True  | True  | False
#    True  | False | True
#    False | True  | True
#    False | False | False
#
# Definir la función
#    xor : (bool, bool) -> bool
# tal que xor(x, y) es la disyunción excluyente de x e y. Por ejemplo,
#    xor(True, True) == False
#    xor(True, False) == True
#    xor(False, True) == True
#    xor(False, False) == False
# ---------------------------------------------------------------------

# 1ª solución
def xor1(x, y):
    match x, y:
        case True,  True: return False
        case True,  False: return True
        case False, True: return True
        case False, False: return False

# 2ª solución
def xor2(x: bool, y: bool) -> bool:
    if x:
        return not y
    return y
```

```python
# 3ª solución
def xor3(x: bool, y: bool) -> bool:
    return (x or y) and not(x and y)

# 4ª solución
def xor4(x: bool, y: bool) -> bool:
    return (x and not y) or (y and not x)

# 5ª solución
def xor5(x: bool, y: bool) -> bool:
    return x != y

# La propiedad de equivalencia es
@given(st.booleans(), st.booleans())
def test_equiv_xor(x, y):
    assert xor1(x, y) == xor2(x, y) == xor3(x, y) == xor4(x, y) == xor5(x, y)

# La comprobación está al final de la relación.

# -----------------------------------------------------------------
# Ejercicio 4. Las dimensiones de los rectángulos puede representarse
# por pares; por ejemplo, (5,3) representa a un rectángulo de base 5 y
# altura 3.
#
# Definir la función
#    mayorRectangulo : (tuple[float, float], tuple[float, float])
#                      -> tuple[float, float]
# tal que mayorRectangulo(r1, r2) es el rectángulo de mayor área entre
# r1 y r2. Por ejemplo,
#    mayorRectangulo((4, 6), (3, 7))  ==  (4, 6)
#    mayorRectangulo((4, 6), (3, 8))  ==  (4, 6)
#    mayorRectangulo((4, 6), (3, 9))  ==  (3, 9)
# -----------------------------------------------------------------

def mayorRectangulo(r1: tuple[float, float],
                    r2: tuple[float, float]) -> tuple[float, float]:
    (a, b) = r1
    (c, d) = r2
    if a*b >= c*d:
```

```
            return (a, b)
        return (c, d)

# ---------------------------------------------------------------------
# Ejercicio 5. Definir la función
#    intercambia : (tuple[A, B]) -> tuple[B, A]
# tal que intercambia(p)  es el punto obtenido intercambiando las
# coordenadas del punto p. Por ejemplo,
#    intercambia((2,5))  ==  (5,2)
#    intercambia((5,2))  ==  (2,5)
#
# Comprobar con Hypothesis que la función intercambia esidempotente; es
# decir, si se aplica dos veces es lo mismo que no aplicarla ninguna.
# ---------------------------------------------------------------------

def intercambia(p: tuple[A, B]) -> tuple[B, A]:
    (x, y) = p
    return (y, x)

# La propiedad de es
@given(st.tuples(st.integers(), st.integers()))
def test_equiv_intercambia(p):
    assert intercambia(intercambia(p)) == p

# La comprobación está al final de la relación.

# ---------------------------------------------------------------------
# Ejercicio 5. Definir la función
#    distancia : (tuple[float, float], tuple[float, float]) -> float
# tal que distancia(p1, p2) es la distancia entre los puntos p1 y
# p2. Por ejemplo,
#    distancia((1, 2), (4, 6)) == 5.0
#
# Comprobar con Hypothesis que se verifica la propiedad triangular de
# la distancia; es decir, dados tres puntos p1, p2 y p3, la distancia
# de p1 a p3 es menor o igual que la suma de la distancia de p1 a p2 y
# la de p2 a p3.
# ---------------------------------------------------------------------

def distancia(p1: tuple[float, float],
```

```
                p2: tuple[float, float]) -> float:
    (x1, y1) = p1
    (x2, y2) = p2
    return sqrt((x1-x2)**2+(y1-y2)**2)

# La propiedad es
cota = 2 ** 30

@given(st.tuples(st.integers(min_value=0, max_value=cota),
                 st.integers(min_value=0, max_value=cota)),
       st.tuples(st.integers(min_value=0, max_value=cota),
                 st.integers(min_value=0, max_value=cota)),
       st.tuples(st.integers(min_value=0, max_value=cota),
                 st.integers(min_value=0, max_value=cota)))
def test_triangular(p1, p2, p3):
    assert distancia(p1, p3) <= distancia(p1, p2) + distancia(p2, p3)

# La comprobación está al final de la relación.

# Nota: Por problemas de redondeo, la propiedad no se cumple en
# general. Por ejemplo,
#    λ> p1 = (0, 9147936743096483)
#    λ> p2 = (0, 3)
#    λ> p3 = (0, 2)
#    λ> distancia(p1, p3) <= distancia(p1, p2) + distancia (p2. p3)
#    False
#    λ> distancia(p1, p3)
#    9147936743096482.0
#    λ> distancia(p1, p2) + distancia(p2, p3)
#    9147936743096480.05

# ---------------------------------------------------------------------
# Ejercicio 6. Definir una función
#    ciclo : (list[A]) -> list[A]
# tal que ciclo(xs) es la lista obtenida permutando cíclicamente los
# elementos de la lista xs, pasando el último elemento al principio de
# la lista. Por ejemplo,
#    ciclo([2, 5, 7, 9])  == [9, 2, 5, 7]
#    ciclo([])            == []
#    ciclo([2])           == [2]
```

```
#
# Comprobar que la longitud es un invariante de la función ciclo; es
# decir, la longitud de (ciclo xs) es la misma que la de xs.
# ---------------------------------------------------------------------

def ciclo(xs: list[A]) -> list[A]:
    if xs:
        return [xs[-1]] + xs[:-1]
    return []

# La propiedad de es
@given(st.lists(st.integers()))
def test_equiv_ciclo(xs):
    assert len(ciclo(xs)) == len(xs)

# La comprobación está al final de la relación.

# ---------------------------------------------------------------------
# Ejercicio 7. Definir la función
#    numeroMayor : (int, int) -> int
# tal que numeroMayor(x, y) es el mayor número de dos cifras que puede
# construirse con los dígitos x e y. Por ejemplo,
#    numeroMayor(2, 5) == 52
#    numeroMayor(5, 2) == 52
# ---------------------------------------------------------------------

# 1ª definición
def numeroMayor1(x: int, y: int) -> int:
    return 10 * max(x, y) + min(x, y)

# 2ª definición
def numeroMayor2(x: int, y: int) -> int:
    if x > y:
        return 10 * x + y
    return 10 * y + x

# La propiedad de equivalencia de las definiciones es
def test_equiv_numeroMayor():
    # type: () -> bool
    return all(numeroMayor1(x, y) == numeroMayor2(x, y)
```

```
                for x in range(10) for y in range(10))

# La comprobación está al final de la relación.

# ---------------------------------------------------------------------
# Ejercicio 8. Definir la función
#    numeroDeRaices : (float, float, float) -> float
# tal que numeroDeRaices(a, b, c) es el número de raíces reales de la
# ecuación a*x^2 + b*x + c = 0. Por ejemplo,
#    numeroDeRaices(2, 0, 3)    ==  0
#    numeroDeRaices(4, 4, 1)    ==  1
#    numeroDeRaices(5, 23, 12)  ==  2
# ---------------------------------------------------------------------

def numeroDeRaices(a: float, b: float, c: float) -> float:
    d = b**2-4*a*c
    if d < 0:
        return 0
    if d == 0:
        return 1
    return 2

# ---------------------------------------------------------------------
# Ejercicio 9. Definir la función
#    raices : (float, float, float) -> list[float]
# tal que raices(a, b, c) es la lista de las raíces reales de la
# ecuación ax^2 + bx + c = 0. Por ejemplo,
#    raices(1, 3, 2)    ==  [-1.0,-2.0]
#    raices(1, (-2), 1) ==  [1.0,1.0]
#    raices(1, 0, 1)    ==  []
#
# Comprobar con Hypothesis que la suma de las raíces de la ecuación
# ax^2 + bx + c = 0 (con a no nulo) es -b/a y su  producto es c/a.
# ---------------------------------------------------------------------

def raices(a: float, b: float, c: float) -> list[float]:
    d = b**2 - 4*a*c
    if d >= 0:
        e = sqrt(d)
        t = 2*a
```

```
        return [(-b+e)/t, (-b-e)/t]
    return []

# Para comprobar la propiedad se usará la función
#    casiIguales : (float, float) -> bool
# tal que casiIguales(x, y) se verifica si x e y son casi iguales; es
# decir si el valor absoluto de su diferencia es menor que una
# milésima. Por  ejemplo,
#    casiIguales(12.3457, 12.3459)  ==  True
#    casiIguales(12.3457, 12.3479)  ==  False
def casiIguales(x: float, y: float) -> bool:
    return abs(x - y) < 0.001

# La propiedad es
@given(st.floats(min_value=-100, max_value=100),
       st.floats(min_value=-100, max_value=100),
       st.floats(min_value=-100, max_value=100))
def test_prop_raices(a, b, c):
    assume(abs(a) > 0.1)
    xs = raices(a, b, c)
    assume(xs)
    [x1, x2] = xs
    assert casiIguales(x1 + x2, -b / a)
    assert casiIguales(x1 * x2, c / a)

# La comprobación está al final de la relación.

# ---------------------------------------------------------------------
# Ejercicio 10. La fórmula de Herón, descubierta por Herón de
# Alejandría, dice que el área de un triángulo cuyo lados miden a, b y c
# es la raíz cuadrada de s(s-a)(s-b)(s-c) donde s es el semiperímetro
#    s = (a+b+c)/2
#
# Definir la función
#    area : (float, float, float) -> float
# tal que area(a, b, c) es el área del triángulo de lados a, b y c. Por
# ejemplo,
#    area(3, 4, 5) == 6.0
# ---------------------------------------------------------------------
```

```
def area(a: float, b: float, c: float) -> float:
    s = (a+b+c)/2
    return sqrt(s*(s-a)*(s-b)*(s-c))

# ---------------------------------------------------------------------
# Ejercicio 11. Los intervalos cerrados se pueden representar mediante
# una lista de dos números (el primero es el extremo inferior del
# intervalo y el segundo el superior).
#
# Definir la función
#    interseccion : (list[float], list[float]) -> list[float]
# tal que interseccion(i1, i2) es la intersección de los intervalos i1 e
# i2. Por ejemplo,
#    interseccion([],     [3, 5])  ==  []
#    interseccion([3, 5], [])      ==  []
#    interseccion([2, 4], [6, 9])  ==  []
#    interseccion([2, 6], [6, 9])  ==  [6, 6]
#    interseccion([2, 6], [0, 9])  ==  [2, 6]
#    interseccion([2, 6], [0, 4])  ==  [2, 4]
#    interseccion([4, 6], [0, 4])  ==  [4, 4]
#    interseccion([5, 6], [0, 4])  ==  []
#
# Comprobar con Hypothesis que la intersección de intervalos es
# conmutativa.
# ---------------------------------------------------------------------

Rectangulo = list[float]

def interseccion(i1: Rectangulo,
                 i2: Rectangulo) -> Rectangulo:
    if i1 and i2:
        [a1, b1] = i1
        [a2, b2] = i2
        a = max(a1, a2)
        b = min(b1, b2)
        if a <= b:
            return [a, b]
        return []
    return []
```

```
# La propiedad es
@given(st.floats(), st.floats(), st.floats(), st.floats())
def test_prop_raices2(a1, b1, a2, b2):
    assume(a1 <= b1 and a2 <= b2)
    assert interseccion([a1, b1], [a2, b2]) == interseccion([a2, b2], [a1, b1])

# La comprobación está al final de la relación.

# ---------------------------------------------------------------------
# Ejercicio 12.1. Los números racionales pueden representarse mediante
# pares de números enteros. Por ejemplo, el número 2/5 puede
# representarse mediante el par (2,5).
#
# El tipo de los racionales se define por
#    Racional = tuple[int, int]
#
# Definir la función
#    formaReducida    : (Racional) -> Racional
# tal que formaReducida(x) es la forma reducida del número racional
# x. Por ejemplo,
#    formaReducida((4, 10))  ==  (2, 5)
#    formaReducida((0, 5))   ==  (0, 1)
# ---------------------------------------------------------------------

Racional = tuple[int, int]

def formaReducida(x: Racional) -> Racional:
    (a, b) = x
    if a == 0:
        return (0, 1)
    c = gcd(a, b)
    return (a // c, b // c)

# ---------------------------------------------------------------------
# Ejercicio 12.2. Definir la función
#    sumaRacional     : (Racional, Racional) -> Racional
# tal que sumaRacional(x, y) es la suma de los números racionales x e y,
# expresada en forma reducida. Por ejemplo,
#    sumaRacional((2, 3), (5, 6))  ==  (3, 2)
#    sumaRacional((3, 5), (-3, 5)) ==  (0, 1)
```

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

def sumaRacional(x: Racional,
                 y: Racional) -> Racional:
    (a, b) = x
    (c, d) = y
    return formaReducida((a*d+b*c, b*d))

# ---------------------------------------------------------------------
# Ejercicio 12.3. Definir la función
#    productoRacional : (Racional, Racional) -> Racional
# tal que productoRacional(x, y) es el producto de los números
# racionales x e  y, expresada en forma reducida. Por ejemplo,
#    productoRacional((2, 3), (5, 6))  ==  (5, 9)
# ---------------------------------------------------------------------

def productoRacional(x: Racional,
                     y: Racional) -> Racional:
    (a, b) = x
    (c, d) = y
    return formaReducida((a*c, b*d))

# ---------------------------------------------------------------------
# Ejercicio 12.4. Definir la función
#    igualdadRacional : (Racional, Racional) -> bool
# tal que igualdadRacional(x, y) se verifica si los números racionales x
# e y son iguales. Por ejemplo,
#    igualdadRacional((6, 9), (10, 15))  ==  True
#    igualdadRacional((6, 9), (11, 15))  ==  False
#    igualdadRacional((0, 2), (0, -5))   ==  True
# ---------------------------------------------------------------------

def igualdadRacional(x: Racional,
                     y: Racional) -> bool:
    (a, b) = x
    (c, d) = y
    return a*d == b*c

# ---------------------------------------------------------------------
# Ejercicio 12.5. Comprobar con Hypothesis la propiedad distributiva del
```

```
# producto racional respecto de la suma.
# ----------------------------------------------------------------------

# La propiedad es
@given(st.tuples(st.integers(), st.integers()),
       st.tuples(st.integers(), st.integers()),
       st.tuples(st.integers(), st.integers()))
def test_prop_distributiva(x, y, z):
    (_, x2) = x
    (_, y2) = y
    (_, z2) = z
    assume(x2 != 0 and y2 != 0 and z2 != 0)
    assert igualdadRacional(productoRacional(x, sumaRacional(y, z)),
                            sumaRacional(productoRacional(x, y),
                                         productoRacional(x, z)))

# La comprobación está al final de la relación

# ----------------------------------------------------------------------
# Comprobación de propiedades.
# ----------------------------------------------------------------------

# La comprobación de las propiedades es
#    src> poetry run pytest -q condicionales_guardas_y_patrones.py
#    9 passed in 1.85s
```