Datos Compuestos (Cap.8)

Hasta el momento hemos visto únicamente cómo operar con valores simples (esto es, con números, con strings, y con valores lógicos). Sin embargo, en computación es recurrente el tener que manipular valores **compuestos**, esto es, formados por una combinación de uno o más valores simples.

Problema: suma entre dos fracciones

Fórmula:

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}$$

Solución 1:

Función que recibe los valores **a**, **b**, **c**, **d**, y calcula el resultado usando la fórmula.

```
# sumaFracciones:  int int int int -> float
# calcula la suma entre dos fracciones a/b y c/d
# ejemplo:  sumaFracciones(1, 2, 3, 4) devuelve 1.25

def sumaFracciones(a,b,c,d):

  return (a*d+b*c)/(b*d)
```

```
sumaFracciones(1, 2, 3, 4)
```

```
sumaFracciones(4,5,7,16)   #suma 4/5 + 7/16
```

Problema:

Lo que devuelve la función NO es una fracción, sino un número real que corresponde a la **representación decimal** del resultado

Estructura (struct)

- Tipo de dato que permite encapsular un conjunto fijo de valores (de uno o más tipos)
- Representado por atributos, para conformar un único valor compuesto

    - Se le denomina ***dato compuesto***
- Ocuparemos el **módulo** `estructura` que contiene la **función** `crear` para trabajar con datos compuestos

En efecto, podemos representar una *fracción* como una estructura formada por dos atributos: un *numerador* y un *denominador*.

```
import estructura
estructura.crear("nombre", "atributo1 atributo2 ...  atributoN")
```

Definiendo fracciones como una estructura

```
import estructura
estructura.crear("fraccion", "numerador denominador")
```

Fijarse que la función `estructura.crear` crea un *molde* para crear una o más fracciones.

```
a = fraccion(1,2)   # crea la primera fracción y la guarda en a
```

```
a      # consultamos para ver que cosa es 'a'
```

```
print(a)      # que sucede si intentamos mostrar a en pantalla?
```

```
a.numerador  # vemos el numerador de 'a'
```

```
a.denominador # vemos el denominador de 'a'
```

```
b = a    # se pueden copiar
print(a.numerador)
print(a.denominador)
```

No mutable

¡No se pueden modificar los valores de los atributos de una estructura!

```
b = fraccion(3,a.denominador)
print(a)
print(b)
```

Receta de diseño con datos compuestos

- Reconocer desde el planteamiento del problema las estructuras de datos que se requieran
- Diseñar las estructuras de datos especificando atributos y sus tipos, por ej:

```
# fraccion: numerador (int) denominador(int)
estructura.crear("fraccion", "numerador denominador")
```

Esto incluye *describir* qué partes tiene la estructura que van a crear, y de qué tipos son dichas partes. Esto es justamente lo que está haciendo en la línea con el "comentario" justo antes de `estructura.crear`.

Además debe:

- Seguir la receta usual para las funciones que ocupen las estructuras:
  - Contrato: tipos parámetros -> tipo resultado
  - Objetivo (propósito) de la *función*
  - Ejemplo(s)
  - Cuerpo de la función
  - Pruebas (test)

Ejemplo: módulo `fraccion`

Se guarda en un archivo llamado `fraccion.py`

```
import estructura
# Diseno de la estructura
# fraccion: numerador (int) denominador(int)
estructura.crear("fraccion", "numerador denominador")



# Contrato
# sumaFracciones: fraccion fraccion -> fraccion

# Proposito
# crear una nueva fraccion que corresponda a la suma de dos fracciones f1 y f2

# Ejemplo:
# sumaFracciones(fraccion(1,2), fraccion(3,4))
# devuelve fraccion(10,8)

# Cuerpo de la funcion
def sumaFracciones(f1,f2):
```

```
    assert type(f1) == fraccion and type(f2)== fraccion
    assert f1.denominador != 0 and f2.denominador != 0

    num = f1.numerador*f2.denominador + f2.numerador*f1.denominador
    den = f1.denominador*f2.denominador
    return fraccion(num,den)

# Test
f12=fraccion(1,2)
f34=fraccion(3,4)
assert sumaFracciones(f12,f34) == fraccion(10,8)
```

```
def pruebaSuma():
    print("suma de fracciones a/b y c/d")

    a=int(input("a?"))   # Ojo, si hacen a = input("a?"), denominador sera str pero debiera s
    b=int(input("b?"))
    f1=fraccion(a,b)

    f2=fraccion(int(input("c?")),int(input("d?")))
    f3=sumaFracciones(f1,f2)

    print("suma=" + str(f3.numerador) + "/" + str(f3.denominador))
```

```
pruebaSuma()
```

```
# Contrato
# restaFracciones: fraccion fraccion -> fraccion

# Proposito
# resta dos fracciones y retorna otra fraccion con el resultado

# Ejemplo:
# restaFracciones(fraccion(1,2), fraccion(3,6))
# devuelve fraccion(-2,8)

# Cuerpo de la funcion
def restaFracciones(f1,f2):
    assert type(f1)==fraccion and type(f2)==fraccion
    assert f1.denominador != 0 and f2.denominador !=0
    num = f1.numerador*f2.denominador - f1.denominador*f2.numerador
    den = f1.denominador * f2.denominador
```

```
    return fraccion(num,den)

# Test
f12=fraccion(1,2)
f34=fraccion(3,4)
assert restaFracciones(f12,f34) == fraccion(-2,8)
```

```
import math

# Contrato
# simplificaFracciones: fraccion -> fraccion

# Proposito
# entrega una fraccion nueva que es la version simplificada de f

# Ejemplo:
# simplificaFracciones(fraccion(10,30)) -> fraccion(1,3)


# Cuerpo de la funcion
def simplificaFracciones(f):
    m = math.gcd(f.numerador,f.denominador)
    return fraccion(f.numerador/m, f.denominador/m)

# Test
assert simplificaFracciones(fraccion(10,30)) == fraccion(1,3)
```

```
# Contrato
# igualdadFracciones: fraccion fraccion -> bool

# Proposito
# Indica si las fracciones f1 y f2 son iguales (equivalentes)

# Ejemplo:
# igualdadFracciones(fraccion(1,2), fraccion(3,6))
# devuelve True

# Cuerpo de la funcion
def igualdadFracciones(f1,f2):
    return simplificaFracciones(f1) == simplificaFracciones(f2)

# Test
f12=fraccion(1,2)
f36=fraccion(3,6)
assert igualdadFracciones(f12,f36)
```

```
def aString(f):
    return str(f.numerador)+"/"+str(f.denominador)
```

```
assert aString(fraccion(1,2)) == "1/2"
```

```
aString(fraccion(2,3))
```

Ejemplo: creamos una estructura para guardar números de a pares

```
import estructura
estructura.crear("par","num1 num2")
```

```
par
```

```
par1 = par(10,23)
```

```
par1
```

```
par1.num1
```

```
par1.num2
```

Propuesto *(opcional, no es ejercicio)*

En lo que sigue, recuerde seguir la receta de diseño.

1. Basado en la estructura de números a pares, cree una estructura `tiempo`, donde cada tiempo consiste en minutos y segundos.
2. Escriba la función `convertirASegundos(t)` que reciba una variable de tipo tiempo y retorne el equivalente en segundos. Por ejemplo, si t1 representa 22 minutos y 57 segundos, entonces `convertirASegundos(t1)` debe retornar 1377 pues 22*60+57=1377.

3. Escriba la función `suma(tpo1,tpo2)` que reciba dos variable tiempos y retorne un tiempo con la suma de los tiempos tpo1 y tpo2.

   Por ejemplo: si un t1 representa 28 min y 30 segundos, y t2 representa 36 min y 42 segundos, entonces `suma(t1,t2)` debiera retornar un tiempo que represente 65 minutos y 12 segundos.