Apellidos y nombre: _____ ID: _____

**Calificación del examen**

C0. La calificación máxima del examen es de 7 puntos, y en cada ejercicio se parte de la puntuación ($P$) asociada al mismo.

C1. Por cada error detectado por
`pylint --disable=invalid-name,redefined-outer-name,missing-docstring <archivo.py>`
se restan $P/4$ puntos.

C2. Por cada test no superado: se restan $(2P)/(3T)$ puntos, donde $T$ = cantidad de tests del ejercicio.

C3. Por cada test superado por un algoritmo *ad-hoc*[1]: se restan $P/T$ puntos.

C4. Por cada algoritmo incorrecto utilizado: se restan $P/2$ puntos.

C5. Por cada detalle algorítmico innecesariamente complicado o ineficiente: se restan $P/4$ puntos.

## 1. Evaluación de expresiones ($P = 3$ puntos)

Programar en Python un modelo orientado a objetos del problema de representar y evaluar expresiones aritméticas. Para ello, escribir una clase abstracta llamada `Expresion` y tres subclases que representen diferentes tipos de expresiones.

Por ejemplo, la expresión:

```
2 + 3 * 5
```

Se podría representar de la siguiente forma:

```
Suma(Numero(2), Producto(Numero(3), Numero(5)))
```

donde `Suma`, `Producto` y `Numero` son subclases de `Expresion`:

- `Numero(`$x$`)` crea un objeto que representa al número entero $x$.
- `Suma(`$x$`, `$y$`)` crea un objeto que representa la suma de dos expresiones $x$ e $y$.
- `Producto(`$x$`, `$y$`)` crea un objeto que representa el producto de dos expresiones $x$ e $y$.

La clase `Expresion` dispone de un método abstracto llamado `valor` (sin parámetros) que habrá que redefinir en cada subclase y que devolverá el valor de la expresión correspondiente (un entero). Por ejemplo:

```
>>> Numero(2).valor()
2
>>> Producto(Numero(3), Numero(5)).valor()
15
```

**Tests:**

```
Numero(2).valor() == 2
Suma(Numero(2), Producto(Numero(3), Numero(5))).valor() == 17
Producto(Numero(2), Suma(Numero(3), Numero(5))).valor() == 16
```

## 2. **Instituto** ($P = 4$ puntos)

Programar en Python un modelo orientado a objetos de un Instituto de Secundaria basado en las clases `Alumno`, `Grupo` y `Asignatura`:

- Cada alumno tiene un nombre.
- Los alumnos pertenecen a un grupo.
- Los alumnos pueden matricularse de varias asignaturas.
- Cada asignatura tiene una denominación y un número de trimestres (que puede ser más de tres).
- Cada alumno puede tener, en cada asignatura, tantas notas como trimestres tenga esa asignatura.
- Un alumno tiene una asignatura aprobada si su nota media en esa asignatura es igual o superior a cinco.
- En la clase `Alumno` se deben crear, al menos, los siguientes métodos:
  - `matricular(`*asignatura*`)`: matricula al alumno en una asignatura; devuelve el propio objeto `Alumno`. *Indicación*: asociar al alumno con la asignatura y viceversa.
  - `set_nota(`*asignatura*`,` *trimestre*`,` *nota*`)`: asigna la nota (un número real) que el alumno ha obtenido en un trimestre en una asignatura; devuelve el propio objeto `Alumno`.
  - `media(`*asignatura*`)`: devuelve la nota media del alumno en una asignatura (un número real).
  - `nota(`*asignatura*`,` *trimestre*`)`: devuelve la nota que el alumno ha obtenido en un trimestre en una asignatura; devuelve `None` si el alumno no está matriculado de esa asignatura o no tiene nota en ese trimestre.
  - `aprobada(`*asignatura*`)`: devuelve `True` si el alumno tiene aprobada la asignatura indicada, o `False` en caso contrario.

**Tests:** Partiendo de:

```
ingles = Asignatura("Inglés", 3)
mates = Asignatura("Matemáticas", 2)
juan = Alumno("Juan Pérez").matricular(ingles).matricular(mates)
juan.set_nota(ingles, 1, 4.0).set_nota(ingles, 2, 6.0).set_nota(ingles, 3, 8.0)
antonio = Alumno("Antonio García").matricular(mates)
antonio.set_nota(mates, 1, 2.5)
```

los tests son:

```
juan.media(ingles) == 6.0
antonio.nota(mates, 2) == None
antonio.nota(mates, 1) == antonio.media(mates)
juan.aprobada(ingles) == True
```

## Nota

[1]Un algoritmo *ad-hoc* es aquel que está diseñado maliciosamente para superar un test, devolviendo directamente la salida esperada por el test ante los valores de entrada adecuados. Por ejemplo, una función que debería calcular la suma de dos números pero que lo único que hace es devolver directamente 7 ante los argumentos 4 y 3 para superar el test `suma(4, 3) == 7`.