


## Condiciones de aprobación

Para aprobar es necesario simultáneamente: <ul style="list-style-type: none"> <li>• completar el 60% del examen, y</li> <li>• obtener al menos la mitad de los puntos en cada paradigma.</li> </ul>	<b>En todas tus respuestas sé puntual</b> , no pierdas el foco de lo que se pregunta. Respuestas en exceso generales son tan malas como respuestas incompletas. 
---	---

## Parte A

De un alumno se conocen su nombre, su dedicación (medida en horas) y los conceptos que aprendió hasta el momento en la carrera. Por ejemplo, PdeP es una materia en la cual se enseñan conceptos nuevos (como orden superior, polimorfismo, etc.) y *compliqueti101* es una materia que suma 100 horas de dedicación por cada reentrega de TP realizada.

Se tiene a los alumnos modelados en Haskell de la siguiente forma:

```
data Alumno = Alumno { nombre :: String , dedicacion :: Int , conceptos :: [String] }
jonSnow = Alumno "Jon Snow" 800 ["nothing"]
```

Y las materias modeladas con estas funciones:

```
pdep conceptosNuevos (Alumno n d cs) = Alumno n d (cs ++ conceptosNuevos)
compliqueti101 reentregas (Alumno n d cs) = Alumno n (d + reentregas * 100) cs
```

- Muestre un ejemplo de uso (una única consulta) para enseñar a jonSnow "polimorfismo" e "invertibilidad" y sumarle 500 horas de dedicación mediante estas materias.
  - Luego de la consulta del punto a**, explicar conceptualmente cuál será el resultado de consultar:
 

```
> dedicacion jonSnow
```
- ¿Puede modelarse **una cursada** en forma de una lista de materias? Explique CON EJEMPLOS qué concepto principal permite o impide su implementación.
- Existe una materia recursividad que, dado un alumno, le enseña una lista infinita de conceptos "Recursividad 1", "Recursividad 2", "Recursividad 3", etc. Dada la siguiente expresión:
 

```
> (elem "nothing" . conceptos . recursividad) jonSnow
```

 Mostrar **ejemplos de implementación**<sup>1</sup> de la función recursividad para los cuales:
  - La expresión indicada termina de evaluarse. Explicar qué lo permite.
  - La expresión indicada NO termina de evaluarse. Explicar qué lo impide.

## Parte B

Dada la siguiente base de conocimiento:

<pre>%libro(Titulo, Soporte) %% digital(Formato, PesoEnKB). %% papel(CantidadDePaginas, Editorial). libro(estudioEnEscarlata, digital(mobi, 532)). libro(elSabuesoDeLosBaskerville, papel(354, zeta)). libro(fundacion, papel(546, planeta)). libro(segundaFundacion, digital(epub, 880)). libro(juegoDeTronos, digital(amz, 1046)).</pre>	<pre>%compro(Usuario, NombreLibro) compro(george, estudioEnEscarlata). compro(george, fundacion). compro(martina, elSabuesoDeLosBaskerville). compro(martina, juegoDeTronos).</pre>
--	---

Y la siguiente solución al cálculo de la deuda de una compra:

```
deudaTotal(Usuario, Deuda):-
    findall(P, (compro(Usuario, Titulo), libro(Titulo, digital(_, KB)),
                P is (KB / 1024) * 10), DeudasDigitales),
    findall(P, (compro(Usuario, Titulo), libro(Titulo, papel(Paginas, _)),
                P is Paginas * 0.05), DeudasEnPapel),
    sum_list(DeudasDigitales, TotalDeudasDigitales),
    sum_list(DeudasEnPapel, TotalDeudasEnPapel),
    Deuda is TotalDeudasDigitales + TotalDeudasEnPapel.
```

1. Escribir una consulta individual y una existencial para deudaTotal1/2 justificando el resultado de ambas.

<sup>1</sup> Se puede usar una función `conceptoRecurso :: Int -> String` para generar cada concepto a incorporar, no hace falta implementar esa función.

2. Es posible afirmar que debido a que la solución planteada es capaz de calcular los precios para cualquier tipo de soporte, se está aprovechando el polimorfismo. Justifique su respuesta.
3. En caso de ser necesario plantee una solución que mejore los puntos anteriores.

## Parte C

De un alumno se conoce las cursadas que hizo, para las cuales registramos qué materia es, dedicación medida en horas y las notas, y queremos saber si se recibió en base a si aprobó las cursadas sabiendo que:

- Todas las materias requieren que hayas aprobado al menos 2 parciales
- Para aprobar "Introducción a la complejidad" además le tenés que dedicar al menos 100 horas.
- "Complicueti101" es una materia que además requiere que alguna de tus notas haya sido 10.
- Para aprobar "Zaraza total" alguna de las notas debe ser al menos 6, ignorando la definición anterior.

La solución propuesta por una consultora fue la siguiente:

<pre> class Alumno {   const cursadas = []   method seRecibio() = cursadas.all({ cursada =&gt;     cursada.fueAprobada()   }) } class Cursada {   var materia; var dedicacion; const notas = []   method fueAprobada() {     if (materia.esZaraza()) {       notas.forEach({nota =&gt;         if (nota &gt; 6) return true })       throw new Exception("No aprobó")     }     var acum = 0     notas.forEach({nota =&gt; if (nota &gt;= 6)       acum++})     if (acum &lt; 2) throw new Exception("No       aprobó")     if (materia.esIntroComple()) {       if (dedicacion &gt;= 100) return true       else throw new Exception("No aprobó")     }     if (materia.esComplicueti()) {       var bool = false       notas.forEach({nota =&gt;         if (nota == 10) bool = true })       if (bool) return bool       else throw new Exception("No aprobó")     }   } } class Materia {   method esIntroComple() = false   method esComplicueti() = false   method esZaraza() = false } class IntroComplejidad inherits Materia {   override method esIntroComple() = true } class Complicueti inherits Materia {   override method esComplicueti() = true } class ZarazaTotal inherits Materia {   override method esZaraza() = true } </pre>	<pre> # Alumno (vi: cursadas) &gt;&gt; seRecibio   ^cursadas allSatisfy:     [:cursada cursada fueAprobada]  # Cursada (vi: materia, dedicacion, notas) &gt;&gt; fueAprobada    acum booleana    materia esZaraza ifTrue: [     notas do: [:nota nota &gt; 6 ifTrue:       [^true]].     self error: 'No aprobó'   ].   acum := 0.   notas do: [:nota   nota &gt;= 6 ifTrue:     [acum := acum + 1]].   acum &lt; 2 ifTrue: [self error: 'No aprobó'].   materia esIntroComple ifTrue: [     dedicacion &gt;= 100 ifTrue: [^true]     ifFalse: [ self error: 'No aprobó' ] ].   materia esComplicueti ifTrue: [     bool := false.     notas do: [:nota   nota = 10 ifTrue:       [bool := true]].     bool ifTrue: [^ bool].     self error: 'No aprobó'.   ].  # Materia &gt;&gt; esIntroComple   ^ false &gt;&gt; esComplicueti   ^ false &gt;&gt; esZaraza   ^ false # IntroComplejidad &gt;&gt; esIntroComple   ^ true # Complicueti &gt;&gt; esComplicueti   ^ true # ZarazaTotal &gt;&gt; esZaraza   ^ true </pre>
--	---

- 1) Justifique los errores conceptuales de la solución, indicando en el código dónde se producen, en base a:
  - a) delegación y polimorfismo; b) manejo de errores; c) declaratividad.
- 2) Modele una solución alternativa en base a estos puntos.
- 3) Indique por qué a su criterio su solución supera los inconvenientes expuestos en el punto 1.