

Introducción a la Programación  
Algoritmos y Estructuras de Datos I

Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

Nota 10

Parcial - 1er cuatrimestre 2023



1.1	1.2	2.1	2.2	3	4.1	4.2
B	B	B	B	B	B	B

Apellido OTAZUVA ARCE ..... Nombre MATEO .....

LU 88/23 ..... Turno TARDE .....

Cant. de hojas entregadas (sin contar ésta) 4 .....

El parcial se aprueba con 5 puntos. Entregar cada ejercicio en hoja separada. No se permite consultar ningún material durante el examen.

Ejercicio 1. 2 puntos

1. [1 punto]

Completar en las siguientes especificaciones nombres adecuados para el problema  $a$ , los parámetros  $b$  y  $c$ , y las etiquetas  $x$ ,  $y$ ,  $z$ ,  $u$  y  $w$ .

problema  $a$  (in  $b: seq(Char \times Char)$ , in  $c: seq(Char)$ ) :  $seq(Char)$  {  
 requiere  $x$ : {  $(\forall i, j: \mathbb{Z})((0 \leq i < |b| \wedge 0 \leq j < |b| \wedge i \neq j) \rightarrow b[i]_0 \neq b[j]_0)$  }  
 requiere  $y$ : {  $(\forall i, j: \mathbb{Z})((0 \leq i < |b| \wedge 0 \leq j < |b| \wedge i \neq j) \rightarrow b[i]_1 \neq b[j]_1)$  }  
 requiere  $z$ : {  $(\forall i: \mathbb{Z})(0 \leq i < |c| \rightarrow (\exists j: \mathbb{Z})(0 \leq j < |b| \wedge b[j]_0 = c[i]))$  }  
 asegura  $u$ : {  $|resultado| = |c|$  }  
 asegura  $w$ : {  $(\forall i: \mathbb{Z})(0 \leq i < |c| \rightarrow (\exists j: \mathbb{Z})(0 \leq j < |b| \wedge b[j]_0 = c[i] \wedge b[j]_1 = resultado[i]))$  }  
 }

2. [1 punto]

Especificar el siguiente problema (se puede especificar de manera formal o semi-formal):

Dados los inputs  $b: seq(Char \times Char)$ ,  $m: seq(seq(Char))$  y  $n: seq(seq(Char))$ , retornar verdadero si  $n$  es igual al resultado de aplicar el problema  $a$  (del punto 1.1) a cada elemento de la secuencia  $m$ .

Ejercicio 2. 4 puntos

1. [2 puntos]

Programar en Haskell una función que satisfaga la especificación del problema  $a$  del Ejercicio 1. Recordá escribir los tipos de los parámetros.

2. [2 puntos]

Programar en Python una función que satisfaga la especificación del problema  $a$  del Ejercicio 1. Recordá escribir los tipos de los parámetros y variables que uses en tu implementación.

Ejercicio 3. 2 puntos

Sea la siguiente especificación del problema aprobado y una posible implementación en lenguaje imperativo:

problema aprobado (in  $notas: seq(\mathbb{Z})$ ) :  $\mathbb{Z}$  {  
 requiere: {  $|notas| > 0$  }  
 requiere: {  $(\forall i: \mathbb{Z})(0 \leq i < |notas| \rightarrow 0 \leq notas[i] \leq 10)$  }  
 asegura: {  $result = 1 \leftrightarrow$  todos los elementos de  $notas$  son mayores o iguales a 4 y el promedio es mayor o igual a 7 }  
 asegura: {  $result = 2 \leftrightarrow$  todos los elementos de  $notas$  son mayores o iguales a 4 y el promedio está entre 4 (inclusive) y 7 }  
 asegura: {  $result = 3 \leftrightarrow$  alguno de los elementos de  $notas$  es menor a 4 o el promedio es menor a 4 }  
 }

Ejercicio 1

1. ✓

#"palabra" que puede tener números o símbolos  
(nombre aprobado por Rubinstein)problema  $\text{encriptar} (\text{in clave: seq}\langle \text{Char} \times \text{Char} \rangle, \text{in palabra: seq}\langle \text{Char} \rangle) : \text{seq}\langle \text{Char} \rangle \{$ requiere  $\text{sinRepetidosOrigenClave} : \{(\forall i, j: \mathbb{Z}) ((0 \leq i < |\text{clave}| \wedge 0 \leq j < |\text{clave}| \wedge i \neq j) \rightarrow \text{clave}[i]_0 \neq \text{clave}[j]_0)\}$ requiere  $\text{sinRepetidosDestinoClave} : \{(\forall i, j: \mathbb{Z}) ((0 \leq i < |\text{clave}| \wedge 0 \leq j < |\text{clave}| \wedge i \neq j) \rightarrow \text{clave}[i]_1 \neq \text{clave}[j]_1)\}$ requiere  $\text{todoDePalabraEnClave} : \{(\forall i: \mathbb{Z}) (0 \leq i < |\text{palabra}| \rightarrow (\exists j: \mathbb{Z}) (0 \leq j < |\text{clave}| \wedge \text{clave}[j]_0 = \text{palabra}[i])\}$ asegura  $\text{mismoLargo} : \{|\text{resultado}| = |\text{palabra}|\}$ asegura  $\text{cadaCarácterEncriptado} : \{(\forall i: \mathbb{Z}) (0 \leq i < |\text{palabra}| \rightarrow (\exists j: \mathbb{Z}) (0 \leq j < |\text{clave}| \wedge \text{clave}[j]_0 = \text{palabra}[i] \wedge \text{clave}[j]_1 = \text{resultado}[i])\}$ 

}

2. ✓

problema  $\text{sonPalabrasEncriptadasConClave} (\text{in clave: seq}\langle \text{Char} \times \text{Char} \rangle, \text{in palabras: seq}\langle \text{seq}\langle \text{Char} \rangle \rangle, \text{in palabrasEncriptadas: seq}\langle \text{seq}\langle \text{Char} \rangle \rangle)$ 

Bool {

requiere  $\text{sinRepetidosOrigenClave} : \{(\forall i, j: \mathbb{Z}) ((0 \leq i < |\text{clave}| \wedge 0 \leq j < |\text{clave}| \wedge i \neq j) \rightarrow \text{clave}[i]_0 \neq \text{clave}[j]_0)\}$ requiere  $\text{sinRepetidosDestinoClave} : \{(\forall i, j: \mathbb{Z}) ((0 \leq i < |\text{clave}| \wedge 0 \leq j < |\text{clave}| \wedge i \neq j) \rightarrow \text{clave}[i]_1 \neq \text{clave}[j]_1)\}$ requiere  $\text{todoDePalabrasEnClave} : \{(\forall k: \mathbb{Z}) (0 \leq k < |\text{palabras}| \rightarrow (\forall i: \mathbb{Z}) (0 \leq i < |\text{palabras}[k]| \rightarrow (\exists j: \mathbb{Z}) (0 \leq j < |\text{clave}| \wedge \text{clave}[j]_0 = \text{palabras}[k][i])\}$ requiere  $\text{mismoLargoListas} : \{|\text{palabras}| = |\text{palabrasEncriptadas}|\}$ asegura  $\text{respuestaCorrecta} : \{\text{resultado} = \text{True} \leftrightarrow (\forall i: \mathbb{Z}) (0 \leq i < |\text{palabras}| \rightarrow \text{palabrasEncriptadas}[i] = \text{encriptar}(\text{palabras}[i])\}$ 

↑  
el problema encriptar  
Necesita también recibir  
una clave

# Ejercicio 2

1.

encriptar :: [(Char, Char)] -> [Char] -> [Char] ✓

encriptar clave [] = []

encriptar clave (char:chars) = (encriptarAux clave char) : (encriptar clave chars)

encriptarAux :: [(Char, Char)] -> Char -> Char ✓

encriptarAux (clave:claves) char

| origen == char = destino

| otherwise = encriptarAux claves char

where (origen, destino) = clave

2.

def encriptar (claves: [(str, str)], palabra: str) -> str: ✓

resAux: [str] = []

for char in palabra:

| for clave in claves:

| | if clave[0] == char:

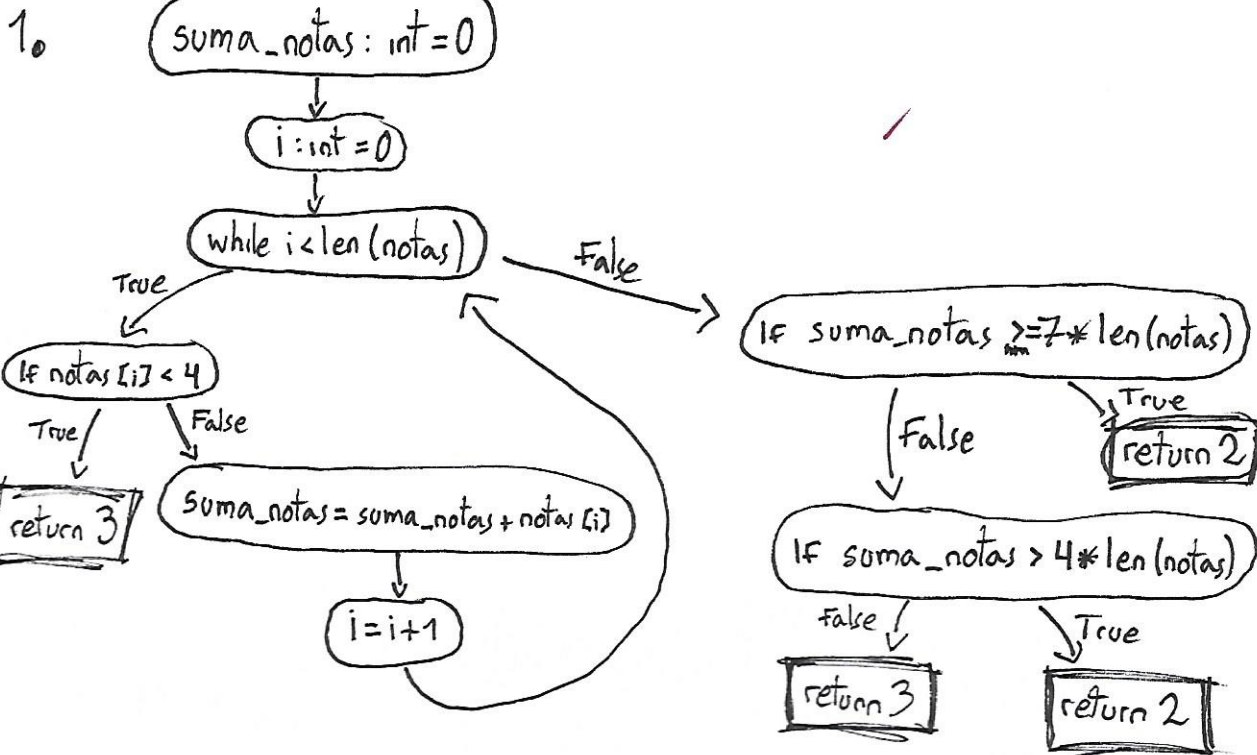
| | resAux.append(clave[1])

res: str = resAux.join()

return res

↑  
" " .join(resAux)

# Ejercicio 3



2. *salidas esperadas*
- caso nota Menor A 4: [2] → 3 ✓
  - caso nota Mayor A 7: [8] → 1 ✓
  - caso nota Entre 4 y 7: [5] → 2 ✓
  - caso nota 4: [4] → 2 ✓

3. idem 2
- caso nota Menor A 4: [2] → 3
  - caso nota Mayor A 7: [8] → 1
  - caso nota Entre 4 y 7: [5] → 2
  - caso nota 4: [4] → 2

4. Hay dos errores en la implementación:

- Nunca devuelve 1, en su lugar devuelve 2 ✓
- Cuando el promedio es 4 debería devolver 2, pero devuelve 3. ✓

Ambos test suites detectan estos defectos. ✓



# Ejercicio 4

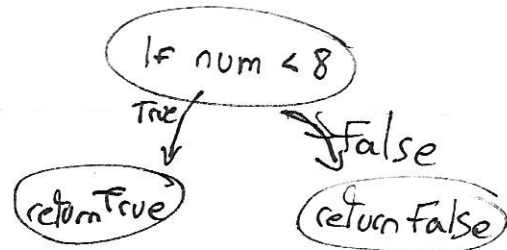
1. Que A sea más fuerte que B significa que  $A \rightarrow B$ , o sea, si A es True, B necesariamente es True también. Pero no siempre que B sea True, necesariamente A lo será.

~~Todo algoritmo que cumpla la especificación de  $p_1$  cumplirá por ende la especificación de  $p_2$ . Pero no todo algoritmo que cumpla la de  $p_2$  cumplirá la de  $p_1$ .~~

Si A y B se encuentran en los asegura, todo algoritmo que cumple con  $p_1$  cumplirá con la de  $p_2$ , pero no todo algoritmo que cumple con la especificación de  $p_2$  cumplirá con la de  $p_1$ . Pero A y B se encuentran en los requiere, por lo tanto,  $p_2$  es una subespecificación de  $p_1$  y todo algoritmo que cumpla con la especificación de  $p_2$  cumplirá la de  $p_1$  pero no necesariamente al revés.

2. Es posible, y se puede demostrar con un ejemplo:

```
def esNegativo (num: int) -> bool:
    | if num < 8: return true
    | else: return false
```



## TEST SUITE

```

{ caso negativo: -1 -> true
{ caso positivo: 9 -> false
```

Este ejemplo recorre todos los nodos sin encontrar el defecto.