Introducción a la Programación

Guía Práctica 1
Lógica

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

1. Lógica binaria (Verdadero o Falso)

Ejercicio 1. ★ Sean p y q variables proposicionales. ¿Cuáles de las siguientes expresiones son *fórmulas bien formadas*?

a) $(p \neg q)$
b) $p \vee q \wedge True$
c) $(p \to \neg p \to q)$
d) $\neg(p)$
e) $(p \vee \neg p \wedge q)$
f) $(True \wedge True \wedge True)$
g) $(\neg p)$
h) $(p \vee False)$
i) $(p = q)$

Ejercicio 2. ★ Sean $x : \mathbb{Z}$, $y : \mathbb{Z}$ y $z : \mathsf{Bool}$ tres variables. Indique cuáles de las siguientes expresiones están bien definidas, teniendo en cuenta que estén bien tipadas las subexpresiones que correspondan.

a) $(1 = 0) \vee (x = y)$
b) $(x + 10) = y$
c) $(x \vee y)$
d) $z = \text{true} \leftrightarrow (y = x)$
e) $(z = 0) \vee (z = 1)$
f) $y + (y < 0)$

Ejercicio 3. La fórmula $(3 + 7 = \pi - 8) \wedge True$ es una fórmula bien formada. ¿Por qué? Justifique informal, pero detalladamente, su respuesta.

Ejercicio 4. ★ Determinar el valor de verdad de las siguientes proposiciones:

a) $(\neg a \vee b)$
b) $(c \vee (y \wedge x) \vee b)$
c) $\neg(c \vee y)$
d) $(\neg(c \vee y) \leftrightarrow (\neg c \wedge \neg y))$
e) $((c \vee y) \wedge (x \vee b))$
f) $(((c \vee y) \wedge (x \vee b)) \leftrightarrow (c \vee (y \wedge x) \vee b))$
g) $(\neg c \wedge \neg y)$

cuando el valor de verdad de a, b y c es *verdadero*, mientras que el de x e y es *falso*.

Ejercicio 5. Determinar, utilizando tablas de verdad, si las siguientes fórmulas son tautologías, contradicciones o contingencias.

a) $(p \vee \neg p)$
b) $(p \wedge \neg p)$
c) $((\neg p \vee q) \leftrightarrow (p \to q))$
d) $((p \vee q) \to p)$
e) $(\neg(p \wedge q) \leftrightarrow (\neg p \vee \neg q))$
f) $(p \to p)$
g) $((p \wedge q) \to p)$
h) $\big((p \wedge (q \vee r)) \leftrightarrow ((p \wedge q) \vee (p \wedge r))\big)$
i) $\big((p \to (q \to r)) \to ((p \to q) \to (p \to r))\big)$

Ejercicio 6. ★ Dadas las proposiciones lógicas α y β, se dice que α es más fuerte que β si y sólo si $\alpha \to \beta$ es una tautología. En este caso, también decimos que β es más débil que α. Determinar la relación de fuerza de los siguientes pares de fórmulas:

a) $True$, $False$
b) $(p \wedge q)$, $(p \vee q)$
c) $True$, $True$
d) p, $(p \wedge q)$
e) $False$, $False$
f) p, $(p \vee q)$
g) p, q
h) p, $(p \to q)$

¿Cuál es la proposición más fuerte y cuál la más débil de las que aparecen en este ejercicio?

Ejercicio 7. ★ Usando reglas de equivalencia (conmutatividad, asociatividad, De Morgan, etc) determinar si los siguientes pares de fórmulas son equivalencias. Indicar en cada paso qué regla se utilizó.

a)
- $((\neg p \vee \neg q) \vee (p \wedge q)) \rightarrow (p \wedge q)$
- $(p \wedge q)$

b)
- $(p \vee q) \wedge (p \vee r)$
- $\neg p \rightarrow (q \wedge r)$

c)
- $\neg(\neg p) \rightarrow (\neg(\neg p \wedge \neg q))$
- q

d)
- $((True \wedge p) \wedge (\neg p \vee False)) \rightarrow \neg(\neg p \vee q)$
- $p \wedge \neg q$

e)
- $(p \vee (\neg p \wedge q))$
- $\neg p \rightarrow q$

f)
- $\neg(p \wedge (q \wedge s))$
- $s \rightarrow (\neg p \vee \neg q)$

g)
- $p \rightarrow (q \wedge \neg(q \rightarrow r))$
- $(\neg p \vee q) \wedge (\neg p \vee (q \wedge \neg r))$

Ejercicio 8. Decimos que un conectivo es *expresable* mediante otros si es posible escribir una fórmula utilizando exclusivamente estos últimos y que tenga la misma tabla de verdad que el primero (es decir, son equivalentes). Por ejemplo, la disyunción es expresable mediante la conjunción más la negación, ya que $(p \vee q)$ tiene la misma tabla de verdad que $\neg(\neg p \wedge \neg q)$.

Mostrar que cualquier fórmula de la lógica proposicional que utilice los conectivos $\neg$ (negación), $\wedge$ (conjunción), $\vee$ (disyunción), $\rightarrow$ (implicación), $\leftrightarrow$ (equivalencia) puede reescribirse utilizando sólo los conectivos $\neg$ y $\vee$.

Ejercicio 9. ★ Sean las variables proposicionales f, e y m con los siguientes significados:

$f \equiv$"es fin de semana" $\quad e \equiv$"Juan estudia" $\quad m \equiv$"Juan escucha música"

a) Escribir usando lógica proposicional las siguientes oraciones:
- "Si es fin de semana, Juan estudia o escucha música, pero no ambas cosas"
- "Si no es fin de semana entonces Juan no estudia"
- "Cuando Juan estudia los fines de semana, lo hace escuchando música"

b) Asumiendo que valen las tres proposiciones anteriores ¿se puede deducir que Juan no estudia? Justificar usando argumentos de la lógica proposicional.

Ejercicio 10. En la salita verde de un jardín se sabe que las siguientes circunstancias son ciertas:

a) Si todos conocen a Juan entonces todos conocen a Camila (podemos pensar que esto se debe a que siempre caminan juntos).

b) Si todos conocen a Juan, entonces que todos conozcan a Camila implica que todos conocen a Gonzalo.

La pregunta entonces es: ¿Es cierto que si todos conocen a Juan entonces todos conocen a Gonzalo? Justificar.

Ejercicio 11. Siempre que Haroldo se pelea con sus compañeritos, vuelve a casa con un ojo morado. Si un día lo viéramos llegar con el ojo destrozado, podríamos sentirnos inclinados a concluir que se ha tomado a golpes de puño y cabezazos con los otros niñitos. ¿Puede identificar el error en el razonamiento anterior? *Pista:* Es conocido como *falacia de afirmar el consecuente.*

2. Lógica ternaria (Verdadero, Falso o Indefinido)

Ejercicio 12. ★ Asignar un *valor de verdad* (*verdadero*, *falso* o *indefinido*) a cada una de las siguientes *expresiones aritméticas* en los reales.

a) $5 > 0$

b) $1 \leq 1$

c) $(5 + 3 - 8)^{-1} \neq 2$

d) $\frac{1}{0} = \frac{1}{0}$

e) $0 \cdot \sqrt{-1} = 0$

f) $\sqrt{-1} \cdot 0 = 0$

Ejercicio 13. ★ ¿Cuál es la diferencia entre el operador $\wedge$ y el operador $\wedge_L$? Describir la tabla de verdad de ambos operadores.

Ejercicio 14. ★ ¿Cuál es la diferencia entre el operador $\vee$ y el operador $\vee_L$? Describir la tabla de verdad de ambos operadores.

Ejercicio 15. ★ ¿Cuál es la diferencia entre el operador $\rightarrow$ y el operador $\longrightarrow_L$? Describir la tabla de verdad de ambos operadores.

Ejercicio 16. ★ Determinar los valores de verdad de las siguientes proposiciones cuando el valor de verdad de b y c es *verdadero*, el de a es *falso* y el de x e y es *indefinido*:

a) $(\neg x \vee_L b)$

b) $((c \vee_L (y \wedge_L a)) \vee b)$

c) $\neg(c \vee_L y)$

d) $(\neg(c \vee_L y) \leftrightarrow (\neg c \wedge_L \neg y))$

e) $((c \vee_L y) \wedge_L (a \vee_L b))$

f) $(((c \vee_L y) \wedge_L (a \vee_L b)) \leftrightarrow (c \vee_L (y \wedge_L a) \vee_L b))$

g) $(\neg c \wedge_L \neg y)$

Ejercicio 17. Sean p, q y r tres variables de las que se sabe que:

- p y q nunca están indefinidas,
- r se indefine sii q es *verdadera*

Proponer una fórmula que nunca se indefina, utilizando siempre las tres variables y que sea verdadera si y solo si se cumple que:

a) Al menos una es verdadera.

b) Ninguna es verdadera.

c) Exactamente una de las tres es verdadera.

d) Sólo p y q son verdaderas.

e) No todas al mismo tiempo son verdaderas.

f) r es verdadera.

3. Cuantificadores

Ejercicio 18.

a) ★ Determinar para cada aparición de variables, si dicha aparición se encuentra libre o ligada. En caso de estar ligada, aclarar a qué cuantificador lo está.

i) $(\forall x : \mathbb{Z})(0 \leq x < n \rightarrow\ x + y = z)$

ii) $(\forall x : \mathbb{Z})((\forall y : \mathbb{Z})((0 \leq x < n \wedge 0 \leq y < m) \rightarrow x + y = z))$

iii) $(\forall j : \mathbb{Z})(0 \leq j < 10 \rightarrow j < 0)$

iv) $s \wedge a < b - 1 \wedge ((\forall j : \mathbb{Z})(a \leq j < b \rightarrow_L 2 * j < b \vee s))$

v) $(\forall j : \mathbb{Z})(j \leq 0 \rightarrow (\forall j : \mathbb{Z})(j > 0 \rightarrow j \neq 0))$

vi) $(\forall j : \mathbb{Z})(j \leq 0 \rightarrow P(j))$

vii) $(\forall j : \mathbb{Z})(j \leq 0 \rightarrow P(j)) \wedge P(j)$

b) ★ En los casos en que sea posible, proponer valores para las variables libres del item anterior de modo tal que las expresiones sean verdaderas.

Ejercicio 19. Sean $P(x : \mathbb{Z})$ y $Q(x : \mathbb{Z})$ dos predicados cualesquiera que nunca se indefinen. Considerar los siguientes enunciados y su predicado asociado. Determinar, en cada caso, por qué el predicado no refleja correctamente el enunciado. Corregir los errores.

a) *"Todos los naturales menores a 10 que cumplen P, cumplen Q":*
pred $a()\{(\forall x : \mathbb{Z})((0 \leq x < 10) \rightarrow_L (P(x) \wedge Q(x)))\}$

b) *"No hay ningún natural menor a 10 que cumpla P y Q":*
pred $b()\{\neg((\exists x : \mathbb{Z})(0 \leq x < 10 \wedge P(x) \wedge \neg((\exists x : \mathbb{Z})(0 \leq x < 10 \wedge Q(x)))))\}$

Introducción a la Programación

Guía Práctica 2
Especificación de problemas

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

Ejercicio 1. Escribir semiformalmente los siguientes predicados.

a) `esPrimo`: que dado un número verifica si cumple las propiedades de ser un número primo.

b) `esPosicionValida`: que dado un entero i y una secuencia l, verifica si i es un índice válido para l.

c) `esMinimo`: que dado una secuencia de enteros l y un entero *elem*, verifica que *elem* sea el mínimo.

d) `esMaximo`: que dado una secuencia de enteros l y un entero *elem*, verifica que *elem* sea el máximo.

Ejercicio 2. ★ Especificar semiformalmente los siguientes problemas. Recuerde que es recomendable descomponer un problema en otros problemas más sencillos (en caso de ya haber especificado los subproblemas no es necesario especificarlos nuevamente).

a) `min`: que dado dos enteros devuelve el menor entre ellos.

b) `max`: que dado dos enteros devuelve el mayor entre ellos.

c) `elMayorPrimo`: que dado dos números primos devuelve el mayor entre ambos.

d) `buscar`: que dado un entero *elem* y una secuencia de enteros l que incluye a *elem*, devuelva una posición donde esté *elem*.

e) `buscarMinimo`: que dado una secuencia de enteros devuelva la posición donde se encuentra el mínimo.

f) `#apariciones`: que dado un entero n y una secuencia de enteros l devuelva la cantidad de veces que aparece n en l.

g) `ordenadaCrecientemente`: que dada una secuencia de enteros sin repetidos, verifique si la lista está ordenada crecientemente.

h) `elMásRepetido`: que dada una secuencia de enteros devuelva el valor que más apariciones tiene.

i) `borrar`: que dada una secuencia de enteros sin repetidos y un elemento *elem*, devuelva el resultado de borrar *elem* (preservando el resto de posiciones intactas).

Ejercicio 3. ★ Las siguientes especificaciones formales no son correctas. Indicar por qué, y corregirlas para que describan correctamente el problema.

a) `buscar:` Dada una secuencia y un elemento de ésta, devuelve en *resultado* alguna posición de la secuencia en la cual se encuentre el elemento.

`problema buscar` (l: $seq\langle\mathbb{R}\rangle$, elem: $\mathbb{R}$) : $\mathbb{Z}$ {
  `requiere:` $\{elem \in l\}$
  `asegura:` $\{l[resultado] = elem\}$
}

b) `minimo:` Dada una secuencia de enteros l, devuelve en *result* el menor elemento de l.

`problema minimo` (l: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {
  `requiere:` $\{True\}$
  `asegura:` $\{(\forall y : \mathbb{Z})((y \in l \land y \neq x) \rightarrow y > result)\}$
}

c) `progresionGeometricaFactor2:` Indica si la secuencia l representa una progresión geométrica con factor 2. Es decir, si cada elemento de la secuencia es el doble del elemento anterior.

`problema progresionGeometricaFactor2` (l: $seq\langle\mathbb{Z}\rangle$) : Bool {

`requiere:` $\{True\}$

`asegura:` $\{resultado = True \leftrightarrow ((\forall i : \mathbb{Z})(0 \leq i < |l| \rightarrow_L l[i] = 2 * l[i-1]))\}$

}

Ejercicio 4. La siguiente especificación informal es una resolución válida del Ejercicio 2 item (d) `buscar`.

`problema buscar` (l: $seq\langle\mathbb{R}\rangle$, elem: $\mathbb{R}$) : $\mathbb{Z}$ {
`requiere:` {$elem$ pertenece a l}
`asegura:` {$resultado$ es una posición de l donde esta el elemento $elem$}
}

Compare esta especificación con la versión formal corregida del ejercicio 3 (a). Considerando la definición de especificación, ¿qué diferencia hay entre una especficicación semiformal y una formal? Ejemplifique con el ejercicio `buscar`.

Ejercicio 5. La siguiente no es una especificación válida, ya que para ciertos valores de entrada que cumplen la precondición, no existe una salida que cumpla con la postcondición.

`problema elementosQueSumen` (l: $seq\langle\mathbb{Z}\rangle$, suma:$\mathbb{Z}$) : $seq\langle\mathbb{Z}\rangle$ {
`requiere:` $\{True\}$
`asegura:` {
/* La secuencia result está incluída en la secuencia l*/
$(\forall x : \mathbb{Z})(x \in result \rightarrow \#apariciones(x, result) \leq \#apariciones(x, l))$
/* La suma de la lista result coincide con el valor suma */
$\wedge\ suma = \sum_{i=0}^{|result|-1} result[i]$
}
}

a) Mostrar valores para l y $suma$ que hagan verdadera la precondición, pero tales que no exista $result$ que cumpla la postcondición.

b) Supongamos que agregamos a la especificación la siguiente cláusula y las especificaciones de los subproblemas que usa la cláusula:

`requiere:` $\{min\_suma(l) \leq suma \leq max\_suma(l)\}$

`problema min_suma` (l :$seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {

`requiere:` $\{True\}$

`asegura:` $\{resultado = \sum_{i=0}^{|l|-1}$ if $l[i] < 0$ then $l[i]$ else 0 fi$\}$

}

`problema max_suma` (l :$seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {

`requiere:` $\{True\}$

`asegura:` $\{resultado = \sum_{i=0}^{|l|-1}$ if $l[i] > 0$ then $l[i]$ else 0 fi$\}$

}

¿Ahora es una especificación válida? Si no lo es, justificarlo con un ejemplo como en el punto anterior.

c) Dar una precondición en lengaje semiformal que haga correcta la especificación.

Ejercicio 6. ★ Para los siguientes problemas, dar todas las soluciones posibles a las entradas dadas:

a) `problema raizCuadrada` (x: $\mathbb{R}$) : $\mathbb{R}$ {

`requiere:` $\{x \geq 0\}$

`asegura:` $\{resultado^2 = x\}$

}

i) $x = 0$

ii) $x = 1$

iii) $x = 27$

b) `problema indiceDelMaximo` (l: $seq\langle\mathbb{R}\rangle$) : $\mathbb{Z}$ {

`requiere:` $\{|l| > 0\}$

`asegura:` $\{0 \leq resultado < |l| \wedge_L ((\forall i : \mathbb{Z})(0 \leq i < |l| \rightarrow_L l[i] \leq l[resultado])\}$

}

i) $l = \langle 1, 2, 3, 4\rangle$

ii) $l = \langle 15.5, -18, 4.215, 15.5, -1\rangle$

iii) $l = \langle 0, 0, 0, 0, 0, 0\rangle$

c) `problema indiceDelPrimerMaximo` (l: $seq\langle\mathbb{R}\rangle$) : $\mathbb{Z}$ {

`requiere:` $\{|l| > 0\}$

`asegura:` {
$0 \leq result < |l|$
$\wedge_L ((\forall i : \mathbb{Z})((0 \leq i < |l| \wedge i \neq resultado) \rightarrow_L (l[i] < l[resultado] \vee (l[i] = l[resultado] \wedge i \geq resultado))))$
}

}

i) $l = \langle 1, 2, 3, 4\rangle$

ii) $l = \langle 15.5, -18, 4.215, 15.5, -1\rangle$

iii) $l = \langle 0, 0, 0, 0, 0, 0\rangle$

d) ¿Para qué valores de entrada `indiceDelPrimerMaximo` y `indiceDelMaximo` tienen necesariamente la misma salida?

Ejercicio 7. ★ Sea $f : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ definida como:

$$f(a, b) = \begin{cases} 2b & \text{si } a < 0 \\ b - 1 & \text{en otro caso} \end{cases}$$

¿Cuáles de las siguientes especificaciones son correctas para el problema de calcular $f(a, b)$?
Para las que no lo son, indicar por qué.

a) `problema f` (a, b: $\mathbb{R}$) : $\mathbb{R}$ {

`requiere:` $\{True\}$

`asegura:` $\{(a < 0 \wedge resultado = 2 * b) \wedge (a \geq 0 \wedge resultado = b - 1)\}$

}

b) `problema f` (a, b: $\mathbb{R}$) : $\mathbb{R}$ {

`requiere:` $\{True\}$

`asegura:` $\{(a < 0 \wedge resultado = 2 * b) \vee (a > 0 \wedge resultado = b - 1)\}$

}

c) `problema f` (a, b: $\mathbb{R}$) : $\mathbb{R}$ {
`requiere:` $\{True\}$
`asegura:` $\{(a < 0 \land resultado = 2 * b) \lor (a \geq 0 \land resultado = b - 1)\}$
}

d) `problema f` (a, b: $\mathbb{R}$) : $\mathbb{R}$ {
`requiere:` $\{True\}$
`asegura:` $\{(a < 0 \rightarrow resultado = 2 * b) \land (a \geq 0 \rightarrow resultado = b - 1)\}$
}

e) `problema f` (a, b: $\mathbb{R}$) : $\mathbb{R}$ {
`requiere:` $\{True\}$
`asegura:` $\{(a < 0 \rightarrow resultado = 2 * b) \lor (a \geq 0 \rightarrow resultado = b - 1)\}$
}

f) `problema f` (a, b: $\mathbb{R}$) : $\mathbb{R}$ {
`requiere:` $\{True\}$
`asegura:` $\{resultado = (\text{if } a < 0 \text{ then } 2 * b \text{ else } b - 1 \text{ fi})\}$
}

Ejercicio 8. ★ Considerar la siguiente especificación, junto con un algoritmo que dado x devuelve x^2.

`problema unoMasGrande` (x: $\mathbb{R}$) : $\mathbb{R}$ {
`requiere:` $\{True\}$
`asegura:` $\{resultado > x\}$
}

a) ¿Qué devuelve el algoritmo si recibe $x = 3$? ¿El resultado hace verdadera la postcondición de `unoMasGrande`?

b) ¿Qué sucede para las entradas x = 0.5, x = 1, x = -0.2 y x = -7?

c) Teniendo en cuenta lo respondido en los puntos anteriores, escribir una **precondición** para `unoMasGrande`, de manera tal que el algoritmo cumpla con la especificación.

Ejercicio 9. ★ Sean x y r variables de tipo $\mathbb{R}$. Considerar los siguientes predicados:

P1: $\{x \leq 0\}$
P2: $\{x \leq 10\}$
P3: $\{x \leq -10\}$

Q1: $\{r \geq x^2\}$
Q2: $\{r \geq 0\}$
Q3: $\{r = x^2\}$

a) Indicar la relación de fuerza entre P1, P2 y P3.

b) Indicar la relación de fuerza enrte Q1, Q2 y Q3.

c) Escribir 2 programas que cumplan con la siguiente especificación E1:

`problema hagoAlgo` (x: $\mathbb{R}$) : $\mathbb{R}$ {
`requiere:` $\{x \leq 0\}$
`asegura:` $\{res \geq x^2\}$
}

d) Sea A un algoritmo que cumple con la especificación E1 del ítem anterior. Decidir si necesariamente cumple las siguientes especificaciones:

a) Pre: $\{x \leq -10\}$, Post: $\{res \geq x^2\}$

b) Pre: $\{x \leq 10\}$, Post: $\{res \geq x^2\}$

c) Pre: $\{x \leq 0\}$, Post: $\{res \geq 0\}$

d) Pre: $\{x \leq 0\}$, Post: $\{res = x^2\}$

e) Pre: $\{x \leq -10\}$, Post: $\{res \geq 0\}$

f) Pre: $\{x \leq 10\}$, Post: $\{res \geq 0\}$

g) Pre: $\{x \leq -10\}$, Post: $\{res = x^2\}$

h) Pre: $\{x \leq 10\}$, Post: $\{res = x^2\}$

e) ¿Qué conclusión pueden sacar? ¿Qué debe cumplirse con respecto a las precondiciones y postcondiciones para que sea seguro reemplazar la especificación?

Ejercicio 10. ★ Considerar las siguientes dos especificaciones, junto con un algoritmo a que satisface la especificación de `p2`.

`problema p1` (x: $\mathbb{R}$, n: $\mathbb{Z}$) : $\mathbb{Z}$ {
  `requiere:` $\{x \neq 0\}$
  `asegura:` $\{x^n - 1 < resultado \leq x^n\}$
}

`problema p2` (x: $\mathbb{R}$, n: $\mathbb{Z}$) : $\mathbb{Z}$ {
  `requiere:` $\{n \leq 0 \rightarrow x \neq 0\}$
  `asegura:` $\{resultado = \lfloor x^n \rfloor\}$
}

a) Dados valores de x y n que hacen verdadera la precondición de `p1`, demostrar que hacen también verdadera la precondición de $p2$.

b) Ahora, dados estos valores de x y n, supongamos que se ejecuta a: llegamos a un valor de res que hace verdadera la postcondición de `p2`. ¿Será también verdadera la postcondición de `p1` con este valor de res?

c) ¿Podemos concluir que a satisface la especificación de `p1`?

Ejercicio 11. Considerar las siguientes especificaciones:

`problema n-esimo1` (l: $seq\langle\mathbb{Z}\rangle$, n: $\mathbb{Z}$) : $\mathbb{Z}$ {
  `requiere:` {La longitud de la secuencia sea mayor a 0}
  `requiere:` {Los elementos están ordenados crecientemente}
  `requiere:` {n es mayor o igual a cero y menor que la longitud de l}
  `asegura:` {$resultado$ es el valor n-esimo de la lista l}
}

`problema n-esimo2` (l: $seq\langle\mathbb{Z}\rangle$, n: $\mathbb{Z}$) : $\mathbb{Z}$ {
  `requiere:` {La longitud de la secuencia sea mayor a 0}
  `requiere:` {Los elementos son distintos entre sí}
  `requiere:` {n es mayor o igual a cero y menor que longitud de l}
  `asegura:` {$resultado$ pertenece a l}
  `asegura:` {La cantidad de elementos de l que son menores a resultado es igual a n}
}

¿Es cierto que todo algoritmo que cumple con n-esimo1 cumple también con n-esimo2? ¿Y al revés?
Sugerencia: Razonar de manera análoga a la del ejercicio anterior.

Ejercicio 12. Especificar los siguientes problemas semiformalmente:

a) ★Dado un entero positivo, obtener su descomposición en factores primos. Devolver una secuencia de tuplas (p, e), donde p es un factor primo y e es su exponente, ordenada en forma creciente con respecto a p.

b) Dada una secuencia de números reales, obtener la diferencia máxima entre dos de sus elementos.

Ejercicio 13. Especificar semiformalmente los siguientes problemas sobre secuencias:

a) Dadas dos secuencias s y t, decidir si s es una subcadena de t.

b) ★ Dadas dos secuencias s y t, decidir si s está *incluida* en t, es decir, si todos los elementos de s aparecen en t en igual o mayor cantidad.

c) problema mezclarOrdenado$(s, t : seq\langle\mathbb{Z}\rangle) : seq\langle\mathbb{Z}\rangle$, que recibe dos secuencias ordenadas y devuelve el resultado de intercalar sus elementos de manera ordenada.

d) ★Dado una secuencia l y un entero n, devolver la secuencia resultante de multiplicar solamente los valores pares por n.

e) Dada una secuencia de números enteros, devolver la secuencia que resulta de borrar los valores múltiplos de 3.

Introducción a la Programación

Guía Práctica 3
Introducción a Haskell

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

1. Definición de funciones básicas

Ejercicio 1. a) Implentar la función parcial `f :: Integer ->Integer` definida por extensión de la siguiente manera:

$$f(1) = 8$$
$$f(4) = 131$$
$$f(16) = 16$$

cuya especificación es la siguiente:

`problema f` (n: $\mathbb{Z}$) : $\mathbb{Z}$ {
`requiere:` $\{n = 1 \vee n = 4 \vee n = 16\}$
`asegura:` $\{(n = 1 \rightarrow result = 8) \wedge (n = 4 \rightarrow result = 131) \wedge (n = 16 \rightarrow result = 16)\}$
}

b) Análogamente, especificar e implementar la función parcial `g :: Integer ->Integer`

$$g(8) = 16$$
$$g(16) = 4$$
$$g(131) = 1$$

c) A partir de las funciones definidas en los ítems 1 y 2, implementar las funciones parciales $h = f \circ g$ y $k = g \circ f$

Ejercicio 2. ★ Especificar e implementar las siguientes funciones, incluyendo su signatura.

a) `absoluto`: calcula el valor absoluto de un número entero.

b) `maximoabsoluto`: devuelve el máximo entre el valor absoluto de dos números enteros.

c) `maximo3`: devuelve el máximo entre tres números enteros.

d) `algunoEs0`: dados dos números racionales, decide si alguno de los dos es igual a 0 (hacerlo dos veces, una usando *pattern matching* y otra no).

e) `ambosSon0`: dados dos números racionales, decide si ambos son iguales a 0 (hacerlo dos veces, una usando *pattern matching* y otra no).

f) `mismoIntervalo`: dados dos números reales, indica si están relacionados considerando la relación de equivalencia en $\mathbb{R}$ cuyas clases de equivalencia son: $(-\infty, 3]$, $(3, 7]$ y $(7, \infty)$, o dicho de otra forma, si pertenecen al mismo intervalo.

g) `sumaDistintos`: que dados tres números enteros calcule la suma sin sumar repetidos (si los hubiera).

h) `esMultiploDe`: dados dos números naturales, decidir si el primero es múltiplo del segundo.

i) `digitoUnidades`: dado un número natural, extrae su dígito de las unidades.

j) `digitoDecenas`: dado un número natural, extrae su dígito de las decenas.

Ejercicio 3. Implementar una función `estanRelacionados :: Integer ->Integer ->Bool`

`problema estanRelacionados` (a:$\mathbb{Z}$, b:$\mathbb{Z}$) : Bool {
`requiere:` $\{a \neq 0 \wedge b \neq 0\}$
`asegura:` $\{(res = true) \leftrightarrow (\exists k : \mathbb{Z})((k \neq 0) \wedge (a * a + a * b * k = 0))\}$
}

Por ejemplo:

`estanRelacionados 8 2` $\rightsquigarrow$ `True` porque existe un $k = -4$ tal que $8^2 + 8 \times 2 \times (-4) = 0$.
`estanRelacionados 7 3` $\rightsquigarrow$ `False` porque no existe un k entero tal que $7^2 + 7 \times 3 \times k = 0$.

Ejercicio 4. ★

Especificar e implementar las siguientes funciones utilizando tuplas para representar pares, ternas de números.

a) `prodInt`: calcula el producto interno entre dos tuplas $\mathbb{R} \times \mathbb{R}$.

b) `todoMenor`: dadas dos tuplas $\mathbb{R} \times \mathbb{R}$, decide si es cierto que cada coordenada de la primera tupla es menor a la coordenada correspondiente de la segunda tupla.

c) `distanciaPuntos`: calcula la distancia entre dos puntos de $\mathbb{R}^2$.

d) `sumaTerna`: dada una terna de enteros, calcula la suma de sus tres elementos.

e) `sumarSoloMultiplos`: dada una terna de números enteros y un natural, calcula la suma de los elementos de la terna que son múltiplos del número natural. *Por ejemplo:*
`sumarSoloMultiplos (10,-8,-5) 2` $\rightsquigarrow$ `2`
`sumarSoloMultiplos (66,21,4) 5` $\rightsquigarrow$ `0`
`sumarSoloMultiplos (-30,2,12) 3` $\rightsquigarrow$ `-18`

f) `posPrimerPar`: dada una terna de enteros, devuelve la posición del primer número par si es que hay alguno, y devuelve 4 si son todos impares.

g) `crearPar :: a ->b ->(a, b)`: crea un par a partir de sus dos componentes dadas por separado (debe funcionar para elementos de cualquier tipo).

h) `invertir :: (a, b) ->(b, a)`: invierte los elementos del par pasado como parámetro (debe funcionar para elementos de cualquier tipo).

Ejercicio 5. Implementar la función `todosMenores :: (Integer, Integer, Integer) ->Bool`

`problema todosMenores` $((n_1,n_2,n_3) : \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z})$: Bool {
  `requiere:` $\{True\}$
  `asegura:` $\{(res = true) \leftrightarrow ((f(n_1) > g(n_1)) \wedge (f(n_2) > g(n_2)) \wedge (f(n_3) > g(n_3))))\}$
}

`problema f` (n: $\mathbb{Z}$) : $\mathbb{Z}$ {
  `requiere:` $\{True\}$
  `asegura:` $\{(n \leq 7 \rightarrow res = n^2) \wedge (n > 7 \rightarrow res = 2n - 1)\}$
}

`problema g` (n: $\mathbb{Z}$) : $\mathbb{Z}$ {
  `requiere:` $\{True\}$
  `asegura:` $\{res =$ if $esPar(n)$ then $n/2$ else $3n + 1$ fi$\}$
}

pred $esPar(n : \mathbb{Z})\{$ (n mod 2) $= 0$ $\}$

Ejercicio 6. Programar una función `bisiesto :: Integer ->Bool`

pred $EsMultiplo(x : \mathbb{Z}, y : \mathbb{Z})\{x \ mod \ y = 0\}$

`problema bisiesto` (año: $\mathbb{Z}$) : Bool {
  `requiere:` $\{True\}$
  `asegura:` $\{res = false \leftrightarrow (\neg EsMultiplo(año, 4) \vee (EsMultiplo(año, 100) \wedge \neg EsMultiplo(año, 400)))\}$
}

Por ejemplo:
`bisiesto 1901` $\rightsquigarrow$ `False`, `bisiesto 1904` $\rightsquigarrow$ `True`,
`bisiesto 1900` $\rightsquigarrow$ `False`, `bisiesto 2000` $\rightsquigarrow$ `True`.

Ejercicio 7. Implementar una función:
`distanciaManhattan:: (Float, Float, Float) ->(Float, Float, Float) ->Float`

`problema distanciaManhattan` $(p : \mathbb{R} \times \mathbb{R} \times \mathbb{R}, q : \mathbb{R} \times \mathbb{R} \times \mathbb{R})$: $\mathbb{R}$ {
  `requiere:` $\{True\}$
  `asegura:` $\{res = \sum_{i=0}^{2} |p_i - q_i|\}$
}

Por ejemplo:
`distanciaManhattan (2, 3, 4) (7, 3, 8)` $\rightsquigarrow$ `9`
`distanciaManhattan ((-1), 0, (-8.5)) (3.3, 4, (-4))` $\rightsquigarrow$ `12.8`

Ejercicio 8. Implementar una función `comparar :: Integer ->Integer ->Integer`

`problema comparar` (a:$\mathbb{Z}$, b:$\mathbb{Z}$) : $\mathbb{Z}$ {
    `requiere:` $\{True\}$
    `asegura:` $\{(res = 1 \leftrightarrow sumaUltimosDosDigitos(a) < sumaUltimosDosDigitos(b))\}$
    `asegura:` $\{(res = -1 \leftrightarrow sumaUltimosDosDigitos(a) > sumaUltimosDosDigitos(b))\}$
    `asegura:` $\{(res = 0 \leftrightarrow sumaUltimosDosDigitos(a) = sumaUltimosDosDigitos(b)))\}$
}

`problema sumaUltimosDosDigitos` (x: $\mathbb{Z}$) : $\mathbb{Z}$ {
    `requiere:` $\{True\}$
    `asegura:` $\{res = (x \mod 10) + (\lfloor (x/10) \rfloor \mod 10)\}$
}

Por ejemplo:
`comparar 45 312` $\rightsquigarrow$ `-1` porque $312 \prec 45$ y $1 + 2 < 4 + 5$.
`comparar 2312 7` $\rightsquigarrow$ `1` porque $2312 \prec 7$ y $1 + 2 < 0 + 7$.
`comparar 45 172` $\rightsquigarrow$ `0` porque no vale $45 \prec 172$ ni tampoco $172 \prec 45$.

Ejercicio 9. A partir de las siguientes implementaciones en Haskell, describir en lenguaje natural qué hacen y especificarlas semiformalmente.

a)
```
f1 :: Float -> Float
f1 n | n == 0 = 1
     | otherwise = 0
```

b)
```
f2 :: Float -> Float
f2 n | n == 1  =  15
     | n == -1 = -15
```

c)
```
f3 :: Float -> Float
f3 n | n <= 9 = 7
     | n >= 3 = 5
```

d)
```
f4 :: Float -> Float -> Float
f4 x y = (x+y)/2
```

e)
```
f5 :: (Float, Float) -> Float
f5 (x, y) = (x+y)/2
```

f)
```
f6 :: Float -> Int -> Bool
f6 a b = truncate a == b
```

Introducción a la Programación

Guía Práctica 4
Recursión sobre números enteros

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

Ejercicio 1. Implementar la función `fibonacci: Integer ->Integer` que devuelve el i-ésimo número de Fibonacci. Recordar que la secuencia de Fibonacci se define como:

$$fib(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ fib(n-1) + fib(n-2) & \text{en otro caso} \end{cases}$$

`problema fibonacci` (n: $\mathbb{Z}$) : $\mathbb{Z}$ {
`requiere:` { $n \geq 0$ }
`asegura:` { $resultado = fib(n)$ }
}

Ejercicio 2. Implementar una función `parteEntera :: Float ->Integer` según la siguiente especificación:

`problema parteEntera` (x: $\mathbb{R}$) : $\mathbb{Z}$ {
`requiere:` {$True$ }
`asegura:` { $resultado \leq x < resultado + 1$ }
}

Ejercicio 3. Implementar la función `esDivisible :: Integer ->Integer ->Bool` que dados dos números naturales determinar si el primero es divisible por el segundo. No está permitido utilizar las funciones `mod` ni `div`.

Ejercicio 4. Implementar la función `sumaImpares :: Integer ->Integer` que dado $n \in \mathbb{N}$ sume los primeros n números impares. Por ejemplo: `sumaImpares 3` $\rightsquigarrow$ `1+3+5` $\rightsquigarrow$ `9`.

Ejercicio 5. Implementar la función `medioFact :: Integer ->Integer` que dado $n \in \mathbb{N}$ calcula $n!! = n\,(n-2)(n-4)\cdots$.

`problema medioFac` (n: $\mathbb{Z}$) : $\mathbb{Z}$ {
`requiere:` { $n \geq 0$ }
`asegura:` { $resultado = \prod_{i=0}^{\lfloor \frac{n-1}{2} \rfloor} (n - 2i)$ }
}

Por ejemplo:
`medioFact` $10 \rightsquigarrow 10 * 8 * 6 * 4 * 2 \rightsquigarrow 3840$.
`medioFact` $9 \rightsquigarrow 9 * 7 * 5 * 3 * 1 \rightsquigarrow 945$.
`medioFact` $0 \rightsquigarrow 1$.

Ejercicio 6. Implementar la función `sumaDigitos :: Integer ->Integer` que calcula la suma de dígitos de un número natural. Para esta función pueden utilizar `div` y `mod`.

Ejercicio 7. Implementar la función `todosDigitosIguales :: Integer ->Bool` que determina si todos los dígitos de un número natural son iguales, es decir:

`problema todosDigitosIguales` (n: $\mathbb{Z}$) : $\mathbb{B}$ {
`requiere:` { $n > 0$ }
`asegura:` { $resultado = True \leftrightarrow (\exists d, k : \mathbb{Z})(n = \sum_{i=0}^{k} d * 10^i)$ }
}

Ejercicio 8. Implementar la función `iesimoDigito :: Integer ->Integer ->Integer` que dado un $n \in \mathbb{N}_{\geq 0}$ y un $i \in \mathbb{N}$ menor o igual a la cantidad de dígitos de n, devuelve el i-ésimo dígito de n.

`problema iesimoDigito` (n: $\mathbb{Z}$, i: $\mathbb{N}$) : $\mathbb{Z}$ {
  `requiere:` { $n \geq 0 \land 1 \leq i \leq cantDigitos(n)$ }
  `asegura:` { $resultado = (n \text{ div } 10^{cantDigitos(n)-i}) \text{ mod } 10$ }
}

`problema cantDigitos` (n: $\mathbb{Z}$) : $\mathbb{N}$ {
  `requiere:` { $n \geq 0$ }
  `asegura:` { $n = 0 \rightarrow resultado = 1$}
  `asegura:` { $n \neq 0 \rightarrow n \text{ div } 10^{resultado-1} > 0 \land n \text{ div } 10^{resultado} = 0))$ }
}

Ejercicio 9. Implementar una función `esCapicua :: Integer ->Bool` que dado $n \in \mathbb{N}_{\geq 0}$ determina si n es un número capicúa.

Ejercicio 10. Implementar y dar el tipo de las siguientes funciones (simil Ejercicio 4 Práctica 2 de Álgebra 1).

a) $f1(n) = \sum_{i=0}^{n} 2^i$, $n \in \mathbb{N}_0$.

b) $f2(n, q) = \sum_{i=1}^{n} q^i$, $n \in \mathbb{N}$ y $q \in \mathbb{R}$

c) $f3(n, q) = \sum_{i=1}^{2n} q^i$, $n \in \mathbb{N}_0$ y $q \in \mathbb{R}$

d) $f4(n, q) = \sum_{i=n}^{2n} q^i$, $n \in \mathbb{N}_0$ y $q \in \mathbb{R}$

Ejercicio 11. a) Implementar una función `eAprox :: Integer ->Float` que aproxime el valor del número e a partir de la siguiente sumatoria:

$$\hat{e}(n) = \sum_{i=0}^{n} \frac{1}{i!}$$

b) Definir la constante `e :: Float` como la aproximación de e a partir de los primeros 10 términos de la serie anterior.
¡Atencion! A veces ciertas funciones esperan un `Float` y nosotros tenemos un `Int`. Para estos casos podemos utilizar la función `fromIntegral :: Int ->Float` definida en el Preludio de Haskell.

Ejercicio 12. Para $n \in \mathbb{N}$ se define la sucesión:

$$a_n = 2 + \cfrac{1}{2 + \cfrac{1}{\ddots \cfrac{}{2 + \cfrac{1}{2 + \cfrac{1}{2}}}}} \qquad \text{(aparece } n \text{ veces el 2).}$$

Lo cual resulta en la siguiente definición recursiva: $a_1 = 2, a_n = 2 + \frac{1}{a_{n-1}}$. Utilizando esta sucesión, implementar una función `raizDe2Aprox :: Integer ->Float` que dado $n \in \mathbb{N}$ devuelva la aproximación de $\sqrt{2}$ definida por $\sqrt{2} \approx a_n - 1$. Por ejemplo:

```
raizDe2Aprox 1 ⇝ 1
raizDe2Aprox 2 ⇝ 1,5
raizDe2Aprox 3 ⇝ 1,4
```

Ejercicio 13. Implementar la siguiente función:

$$f(n, m) = \sum_{i=1}^{n} \sum_{j=1}^{m} i^j$$

Ejercicio 14. Implementar una función `sumaPotencias :: Integer ->Integer ->Integer ->Integer` que dados tres naturales q, n, m sume todas las potencias de la forma q^{a+b} con $1 \leq a \leq n$ y $1 \leq b \leq m$.

Ejercicio 15. Implementar una función `sumaRacionales :: Integer ->Integer ->Float` que dados dos naturales n, m sume todos los números racionales de la forma p/q con $1 \leq p \leq n$ y $1 \leq q \leq m$, es decir:

`problema sumaRacionales` $(n : \mathbb{N}, m : \mathbb{N}) : \mathbb{R}$ {
    `requiere:` { $True$}
    `asegura:` { $resultado = \sum_{p=1}^{n} \sum_{q=1}^{m} \frac{p}{q}$ }
}

Ejercicio 16. Recordemos que un entero $p > 1$ es **primo** si y sólo si no existe un entero k tal que $1 < k < p$ y k divida a p.

a) Implementar `menorDivisor :: Integer ->Integer` que calcule el menor divisor (mayor que 1) de un natural n pasado como parámetro.

b) Implementar la función `esPrimo :: Integer ->Bool` que indica si un número natural pasado como parámetro es primo.

c) Implementar la función `sonCoprimos :: Integer ->Integer ->Bool` que dados dos números naturales indica si no tienen algún divisor en común mayor estricto que 1.

d) Implementar la función `nEsimoPrimo :: Integer ->Integer` que devuelve el n-ésimo primo ($n \geq 1$). Recordar que el primer primo es el 2, el segundo es el 3, el tercero es el 5, etc.

Ejercicio 17. Implementar la función `esFibonacci :: Integer ->Bool` según la siguiente especificación:

`problema esFibonacci` (n: $\mathbb{Z}$) : $\mathbb{B}$ {
    `requiere:` { $n \geq 0$ }
    `asegura:` { $resultado = True \leftrightarrow (\exists i : \mathbb{Z})(i \geq 0 \wedge n = fib(i))$ }
}

Ejercicio 18. Implementar una función `mayorDigitoPar :: Integer ->Integer` según la soguiente especificación:

`problema mayorDigitoPar` (n: $\mathbb{N}$) : $\mathbb{N}$ {
    `requiere:` { $True$ }
    `asegura:` { $resultado$ es el mayor de los dígitos pares de n. Si n no tiene ningún dígito par, entonces $resultado$ es -1.
    }
}

Ejercicio 19. Implementar la funición `esSumaInicialDePrimos :: Int ->Bool` según la siguiente especificación:

`problema esSumaInicialDePrimos` (n: $\mathbb{Z}$) : $\mathbb{B}$ {
    `requiere:` { $n \geq 0$ }
    `asegura:` { $resultado = True \leftrightarrow n$ es igual a la suma de los m primeros números primos, para algún m.}
}

Ejercicio 20. Implementar la función `tomaValorMax :: Int ->Int ->Int` que dado un número entero $n_1 \geq 1$ y un $n_2 \geq n_1$ devuelve algún m entre n_1 y n_2 tal que `sumaDivisores`$(m) = \text{máx}\{$`sumaDivisores`$(i) \mid n_1 \leq i \leq n_2\}$

Ejercicio 21. Implementar una función `pitagoras :: Integer ->Integer ->Integer ->Integer` que dados m, n , $r \in \mathbb{N}_{\geq 0}$, cuente cuántos pares (p, q) con $0 \leq p \leq m$ y $0 \leq q \leq n$ satisfacen que $p^2 + q^2 \leq r^2$. Por ejemplo:
`pitagoras 3 4 5` $\rightsquigarrow$ `20`
`pitagoras 3 4 2` $\rightsquigarrow$ `6`

Introducción a la Programación

Guía Práctica 5
Recursión sobre listas

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

Ejercicio 1. Definir las siguientes funciones sobre listas:

1. `longitud :: [t] -> Integer`, que dada una lista devuelve su cantidad de elementos.
2. `ultimo :: [t] -> t` según la siguiente especificación:

   `problema ultimo` (s: $seq\langle T\rangle$) : $seq\langle T\rangle$ {
   `requiere:` { $|s| > 0$ }
   `asegura:` { $resultado = s[|s| - 1]$ }
   }

3. `principio :: [t] -> [t]` según la siguiente especificación:

   `problema principio` (s: $seq\langle T\rangle$) : $seq\langle T\rangle$ {
   `requiere:` { $|s| > 0$ }
   `asegura:` { $resultado = subseq(s, 0, |s| - 1)$ }
   }

4. `reverso :: [t] -> [t]` según la siguiente especificación:

   `problema reverso` (s: $seq\langle T\rangle$) : $seq\langle T\rangle$ {
   `requiere:` { $True$ }
   `asegura:` { *resultado* tiene los mismos elementos que s pero en orden inverso.}
   }

Ejercicio 2. Definir las siguientes funciones sobre listas:

1. `pertenece :: (Eq t) => t -> [t] -> Bool` según la siguiente especificación:

   `problema pertenece` (e: T, s: $seq\langle T\rangle$) : $\mathbb{B}$ {
   `requiere:` { $True$ }
   `asegura:` { $resultado = true \leftrightarrow e \in s$ }
   }

2. `todosIguales :: (Eq t) => [t] -> Bool`, que dada una lista devuelve verdadero sí y solamente sí todos sus elementos son iguales.
3. `todosDistintos :: (Eq t) => [t] -> Bool` según la siguiente especificación:

   `problema todosDistintos` (s: $seq\langle T\rangle$) : $\mathbb{B}$ {
   `requiere:` { $True$ }
   `asegura:` { $resultado = false \leftrightarrow (\exists i, j : \mathbb{Z})(0 \leq i < |s| \wedge 0 \leq j < |s| \wedge i \neq j \wedge s[i] = s[j]$ }
   }

4. `hayRepetidos :: (Eq t) => [t] -> Bool` según la siguiente especificación:

   `problema hayRepetidos` (s: $seq\langle T\rangle$) : $\mathbb{B}$ {
   `requiere:` { $True$ }
   `asegura:` {$resultado = true \leftrightarrow (\exists i, j : \mathbb{Z})(0 \leq i < |s| \wedge 0 \leq j < |s| \wedge i \neq i \wedge s[i] = s[j])$}
   }

5. `quitar :: (Eq t) => t -> [t] -> [t]`, que dada una lista xs y un elemento x, elimina la primera aparición de x en la lista xs (de haberla).

6. `quitarTodos :: (Eq t ) => t -> [t] -> [t]`, que dada una lista xs y un elemento x, elimina todas las apariciones de x en la lista xs (de haberlas). Es decir:

   `problema quitarTodos` (e: T, s: $seq\langle T\rangle$) : $seq\langle T\rangle$ {
   `requiere:` { $True$ }
   `asegura:` { *resultado* es igual a s pero sin el elemento e. }
   }

7. `eliminarRepetidos :: (Eq t) => [t] -> [t]` que deja en la lista una única aparición de cada elemento, eliminando las repeticiones adicionales.

8. `mismosElementos :: (Eq t) => [t] -> [t] -> Bool`, que dadas dos listas devuelve verdadero sí y solamente sí ambas listas contienen los mismos elementos, sin tener en cuenta repeticiones, es decir:

   `problema mismosElementos` (s: $seq\langle T\rangle$, r: $seq\langle T\rangle$) : $\mathbb{B}$ {
   `requiere:` { $True$ }
   `asegura:` { $resultado = true \leftrightarrow (\forall e : T)(e \in s \leftrightarrow e \in r)$ }
   }

9. `capicua :: (Eq t) => [t] -> Bool` según la siguiente especificación:

   `problema capicua` (s: $seq\langle T\rangle$) : $\mathbb{B}$ {
   `requiere:` { $True$ }
   `asegura:` { $resultado = true \leftrightarrow (\forall i : \mathbb{Z})(0 \leq i < \lfloor \frac{|s|}{2} \rfloor \rightarrow s[i] = s[|s| - 1 - i])$ }
   }

   Por ejemplo `capicua [á','c', 'b', 'b', 'c', á']` es `true`, `capicua [á', 'c', 'b', 'd', á']` es `false`.

Ejercicio 3. Definir las siguientes funciones sobre listas de enteros:

1. `sumatoria :: [Integer] -> Integer` según la siguiente especificación:

   `problema sumatoria` (s: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {
   `requiere:` { $True$ }
   `asegura:` { $resultado = \sum_{i=0}^{|s|-1} s[i]$ }
   }

2. `productoria :: [Integer] -> Integer` según la siguiente especificación:

   `problema productoria` (s: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {
   `requiere:` { $True$ }
   `asegura:` { $resultado = \prod_{i=0}^{|s|-1} s[i]$ }
   }

3. `maximo :: [Integer] -> Integer` según la siguiente especificación:

   `problema maximo` (s: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {
   `requiere:` { $|s| > 0$ }
   `asegura:` { $resultado \in s \wedge (\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow resultado \geq s[i]$ }
   }

4. `sumarN :: Integer -> [Integer] -> [Integer]` según la siguiente especificación:

   `problema sumarN` (n: $\mathbb{Z}$, s: $seq\langle\mathbb{Z}\rangle$) : $seq\langle\mathbb{Z}\rangle$ {
   `requiere:` { $True$ }
   `asegura:` {$|resultado| = |s| \wedge (\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow resultado[i] = s[i] + n$ }
   }

5. `sumarElPrimero :: [Integer] -> [Integer]` según la siguiente especificación:

   `problema sumarElPrimero` (s: $seq\langle\mathbb{Z}\rangle$) : $seq\langle\mathbb{Z}\rangle$ {
   `requiere:` { $|s| > 0$ }
   `asegura:` {$|resultado| = |s| \wedge (\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow resultado[i] = s[i] + s[0]$ }
   }

   Por ejemplo `sumarElPrimero [1,2,3]` da `[2,3,4]`

6. `sumarElUltimo :: [Integer] -> [Integer]` según la siguiente especificación:

   `problema sumarElUltimo` (s: $seq\langle\mathbb{Z}\rangle$) : $seq\langle\mathbb{Z}\rangle$ {
   `requiere:` { $|s| > 0$ }
   `asegura:` {$|resultado| = |s| \wedge (\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow resultado[i] = s[i] + s[|s| - 1]$ }
   }

   Por ejemplo `sumarElUltimo [1,2,3]` da `[4,5,6]`

7. `pares :: [Integer] -> [Integer]` según la siguiente especificación:

   `problema pares` (s: $seq\langle\mathbb{Z}\rangle$) : $seq\langle\mathbb{Z}\rangle$ {
   `requiere:` { $True$ }
   `asegura:` {*resultado* sólo tiene los elementos pares de s en el orden dado, respetando las repeticiones.}
   }

   Por ejemplo `pares [1,2,3,5,8,2]` da `[2,8,2]`

8. `multiplosDeN :: Integer -> [Integer] -> [Integer]` que dado un número n y una lista xs, devuelve una lista con los elementos de xs múltiplos de n.

9. `ordenar :: [Integer] -> [Integer]` que ordena los elementos de la lista en forma creciente.

Ejercicio 4. Definir las siguientes funciones sobre listas de caracteres, interpretando una palabra como una secuencia de caracteres sin blancos:

1. `sacarBlancosRepetidos :: [Char] -> [Char]`, que reemplaza cada subsecuencia de blancos contiguos de la primera lista por un solo blanco en la segunda lista.
2. `contarPalabras :: [Char] -> Integer`, que dada una lista de caracteres devuelve la cantidad de palabras que tiene.
3. `palabraMasLarga :: [Char] -> [Char]`, que dada una lista de caracteres devuelve su palabra más larga.
4. `palabras :: [Char] -> [[Char]]`, que dada una lista arma una nueva lista con las palabras de la lista original.
5. `aplanar :: [[Char]] -> [Char]`, que a partir de una lista de palabras arma una lista de caracteres concatenándolas.
6. `aplanarConBlancos :: [[Char]] -> [Char]`, que a partir de una lista de palabras, arma una lista de caracteres concatenándolas e insertando un blanco entre cada palabra.
7. `aplanarConNBlancos :: [[Char]] -> Integer -> [Char]`, que a partir de una lista de palabras y un entero n, arma una lista de caracteres concatenándolas e insertando n blancos entre cada palabra (n debe ser no negativo).

Ejercicio 5. Definir las siguientes funciones sobre listas:

1. `nat2bin :: Integer -> [Integer]`, que recibe un número no negativo y lo transforma en una lista de bits correspondiente a su representación binaria. Por ejemplo `nat2bin 8` devuelve `[1, 0, 0, 0]`.
2. `bin2nat :: [Integer] -> Integer` según la siguiente especificación:

   `problema bin2nat` (s: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {
   `requiere:` { s es una lista de 0's y 1's}
   `asegura:` { *resultado* es el número entero no negativo cuya representación binaria es s.}
   }

   Por ejemplo `bin2nat [1, 0, 0, 0, 1]` devuelve `17`.

3. `nat2hex :: Integer -> [Char]`, que recibe un número no negativo y lo transforma en una lista de caracteres correspondiente a su representación hexadecimal. Por ejemplo `nat2hex 45` devuelve ['2', 'D'].

4. `sumaAcumulada :: (Num t) => [t] -> [t]` según la siguiente especificación:

   `problema sumaAcumulada` (s: $seq\langle T\rangle$) : $seq\langle T\rangle$ {

   `requiere:` $\{T \in [\mathbb{N}, \mathbb{Z}, \mathbb{R}]\}$

   `asegura:` $\{(\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow resultado[i] = \sum_{k=0}^{i} s[k]\}$

   }

   Por ejemplo `sumaAcumulada [1, 2, 3, 4, 5]` es `[1, 3, 6, 10, 15]`.

5. `descomponerEnPrimos :: [Integer] -> [[Integer]]` según la siguiente especificación:

   `problema descomponerEnPrimos` (s: $seq\langle\mathbb{Z}\rangle$) : $seq\langle seq\langle\mathbb{Z}\rangle\rangle$ {

   `requiere:` { $True$ }

   `asegura:` { *resultado* es lista de listas de enteros, que resulta de descomponer en números primos cada uno de los números de s, manteniendo el orden.}

   }

   Por ejemplo `descomponerEnPrimos [2, 10, 6]` es `[[2], [2, 5], [2, 3]]`.

Ejercicio 6. Definir las siguientes funciones sobre conjuntos:

1. `agregarATodos :: Integer -> Set (Set Integer) -> Set (Set Integer)` que dado un número n y un conjunto de conjuntos de enteros cls agrega a n en cada conjunto de cls.

2. `partes :: Integer -> Set (Set Integer)` que genere todos los subconjuntos del conjunto $\{1, 2, \ldots, n\}$. Por ejemplo `partes 2` es `[[], [1], [2], [1, 2]]`.

3. `productoCartesiano :: Set Integer -> Set Integer -> Set (Integer, Integer)` según la siguiente especificación:

   `problema productoCartesiano` (s: $seq\langle\mathbb{Z}\rangle$,r: $seq\langle\mathbb{Z}\rangle$) : $seq\langle < \mathbb{Z}, \mathbb{Z} > \rangle$ {

   `requiere:` { sinRepetidos(s) $\wedge$ sinRepetidos(r) }

   `asegura:` { *resultado* es el conjunto de todas las duplas posibles (como pares de dos elementos) tomando el primer elemento de s y el segundo elemento de r.}

   }

   pred sinRepetidos($s : seq\langle\mathbb{Z}\rangle$) {

   $(\forall i, j : \mathbb{Z})(0 \leq i < |s| \wedge 0 \leq j < |s| \wedge i \neq i \rightarrow s[i] \neq s[j])$

   }

   Por ejemplo `productoCartesiano [1, 2, 3] [3, 4]` es `[(1, 3), (2, 3), (3, 3), (1, 4), (2, 4), (3, 4)]`.

Introducción a la Programación

Guía Práctica 6
Testing de caja Negra

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

Ejercicio 1. Pensar y responder las siguientes preguntas:

1. ¿Cuál es el objetivo de realizar testing de un software?
2. ¿Realizar testing sobre un software nos demuestra que el software funciona correctamente? Justificar.
3. ¿En qué consiste el testing de caja negra?. ¿Cuál es su principal característica?
4. ¿Se puede realizar testing de caja negra sin contar con una especificación del programa a testear? Justificar.

Ejercicio 2. Pensando en el test de caja negra utilizando el método de partición por categorías de un programa que, dados tres enteros que se interpretan como la longitud de cada uno de los lados de un triángulo, dice si el triángulo resultante es isósceles, escaleno o equilátero. Tener en cuenta que para que un triangulo sea factible, la suma de dos de sus lados debe ser mayor a la longitud del tercer lado.

1. Indicar cuáles son los Factores del programa.
2. ¿Existen Factores que son relaciones entre otros Factores? ¿Cuáles?
3. Pensar características relevantes (Categorías) para cada uno de los factores encontrados.
4. Determinar Elecciones (Choices) para cada Categoría de cada Factor.
5. Clasificar las Elecciones: errores, únicos, restricciones, etc
6. Armar los casos de prueba, combinando las distintas Elecciones determinadas para cada Categoría detallando el resultado esperado en cada caso.

Ejercicio 3. Dada la siguiente especificación del problema parteEntera (Ej 2 de la Guía 4):

`problema parteEntera` (x: $\mathbb{R}$) : $\mathbb{Z}$ {
  `requiere:` { $True$ }
  `asegura:` { $resultado \leq x < resultado + 1$ }
}

1. Diseñar los casos de test de caja negra utilizando el método de partición por categorías
2. Proveer datos de test para cada uno de los casos presentados.

Ejercicio 4. Dada la siguiente especificación del problema quitarTodos (Ej 2.6 de la Guía 5):

`problema quitarTodos` (e: T, s: $seq\langle T \rangle$) : $seq\langle T \rangle$ {
  `requiere:` { $True$ }
  `asegura:` { *resultado* es igual a s pero sin el elemento e. }
}

1. Diseñar los casos de test de caja negra utilizando el método de partición por categorías
2. Proveer datos de test para cada uno de los casos presentados.

Ejercicio 5. Dada la siguiente especificación del problema sumarN (Ej 3.4 de la Guía 5):

`problema sumarN` (n: $\mathbb{Z}$, s: $seq\langle \mathbb{Z} \rangle$) : $seq\langle \mathbb{Z} \rangle$ {
  `requiere:` { $True$ }
  `asegura:` {$|resultado| = |s| \land (\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow resultado[i] = s[i] + n$ }
}

1. Diseñar los casos de test de caja negra utilizando el método de partición por categorías

2. Proveer datos de test para cada uno de los casos presentados.

Ejercicio 6. Diseñar los casos de test de caja negra utilizando el método de partición por categorías para los siguientes problemas:

1. `multiplosDeN :: Integer -> [Integer] -> [Integer]` que dado un número n y una lista xs, devuelve una lista con los elementos de xs múltiplos de n. (Ej 3.8 de la Guía 5)

2. `ordenar :: [Integer] -> [Integer]` que ordena los elementos de la lista en forma creciente. (Ej 3.9 de la Guía 5)

3. `aplanarConNBlancos :: [[Char]] -> Integer -> [Char]`, que a partir de una lista de palabras y un entero n, arma una lista de caracteres concatenándolas e insertando n blancos entre cada palabra (n debe ser no negativo). (Ej 4.7 de la Guía 5)

Introducción a la Programación

Guía Práctica 7
Introducción a Lenguaje Imperativo

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

Recordar usar las anotaciones de tipado en todas las variables. Por ejemplo: `def funcion(numero: int) ->bool:`.

En los ejercicios se pueden usar funciones matemáticas como por ejemplo: `sqrt, round, floor, ceil, %`. Ver especificaciones de dichas funciones en la documentación de Python: `https://docs.python.org/es/3.10/library/math.html` y `https://docs.python.org/es/3/library/functions.html`

Ejercicio 1. Definir las siguientes funciones y procedimientos:

1. `raizDe2()`: que devuelva la raíz cuadrada de 2 con 4 decimales. Ver función `round`
2. `problema imprimir_hola` () {
   `requiere:` { True }
   `asegura:` { imprime 'hola'por consola}
   }
3. `imprimir_un_verso()`: que imprima un verso de una canción que vos elijas, respetando los saltos de línea.
4. `factorial_de_dos()`

   `problema factorial_2` () : $\mathbb{Z}$ {
   `requiere:` { True }
   `asegura:` {res = 2!}
   }
5. `problema factorial_3` () : $\mathbb{Z}$ {
   `requiere:` { True }
   `asegura:` {res = 3!}
   }
6. `problema factorial_4` () : $\mathbb{Z}$ {
   `requiere:` { True }
   `asegura:` {res = 4!}
   }
7. `problema factorial_5` () : $\mathbb{Z}$ {
   `requiere:` { True }
   `asegura:` {res = 5!}
   }

Ejercicio 2. Definir las siguientes funciones y procedimientos con parámetros:

1. `problema imprimir_saludo` (in nombre: String) {
   `requiere:` { True }
   `asegura:` {imprime "Hola < *nombre* >"por pantalla}
   }
2. `raiz_cuadrada_de(numero)`: que devuelva la raíz cuadrada del número.

3. `imprimir_dos_veces(estribillo)`: que imprima dos veces el estribillo de una canción. Nota: Analizar el comportamiento del operador (*) con strings.

4. `problema es_multiplo_de` (in n: $\mathbb{Z}$, in m:$\mathbb{Z}$) {
   `requiere:` { m $\neq 0$ }
   `asegura:` $\{res = True \leftrightarrow (\exists k : \mathbb{Z})(n = m * k)\}$
   }

5. `es_par(numero)`: que indique si *numero* es par (usar la función `es_multiplo_de()`).

6. `cantidad_de_pizzas(comensales, min_cant_de_porciones)` que devuelva la cantidad de pizzas que necesitamos para que cada comensal coma como mínimo $min_cant_de_porciones$ porciones de pizza. Considere que cada pizza tiene 8 porciones y que se prefiere que sobren porciones.

Ejercicio 3. Resuelva los siguientes ejercicios utilizando los operadores lógicos `and, or, not`. Resolverlos sin utilizar alternativa condicional (if).

1. `alguno_es_0(numero1, numero2)`: dados dos números racionales, decide si alguno de los dos es igual a 0.

2. `ambos_son_0(numero1, numero2)`: dados dos números racionales, decide si ambos son iguales a 0.

3. `problema es_nombre_largo` (in nombre: String) : Bool {
   `requiere:` { True }
   `asegura:` $\{res = True \leftrightarrow 3 \leq |nombre| \leq 8\}$
   }

4. `es_bisiesto(año)`: que indica si un año tiene 366 días. Recordar que un año es bisiesto si es múltiplo de 400, o bien es múltiplo de 4 pero no de 100.

Ejercicio 4. Usando las funciones de python `min` y `max` resolver:
En una plantación de pinos, de cada árbol se conoce la altura expresada en metros. El peso de un pino se puede estimar a partir de la altura de la siguiente manera:

- 3 kg por cada centímetro hasta 3 metros,
- 2 kg por cada centímetro arriba de los 3 metros.

Por ejemplo:

- 2 metros pesan 600 kg, porque 200 * 3 = 600
- 5 metros pesan 1300 kg, porque los primeros 3 metros pesan 900 kg y los siguientes 2 pesan los 400 restantes.

Los pinos se usan para llevarlos a una fábrica de muebles, a la que le sirven árboles de entre 400 y 1000 kilos, un pino fuera de este rango no le sirve a la fábrica.
Definir las siguientes funciones, deducir qué parámetros tendrán a partir del enunciado. Se pueden usar funciones auxiliares si fuese necesario para aumentar la legibilidad.

1. Definir la función `peso_pino`

2. Definir la función `es_peso_util`, recibe un peso en kg y responde si un pino de ese peso le sirve a la fábrica.

3. Definir la función `sirve_pino`, recibe la altura de un pino y responde si un pino de ese peso le sirve a la fábrica.

4. Definir `sirve_pino` usando composición de funciones.

Ejercicio 5. Implementar los siguientes problemas de alternativa condicional (if). Intentá especificarlos alguno de ellos (todos los que te salgan) en lenguaje semiformal y formal sin utilizar IfThenElseFi.

1. `devolver_el_doble_si_es_par(un_numero)`. Debe devolver el mismo número en caso de no ser par.

2. `devolver_valor_si_es_par_sino_el_que_sigue(un_numero)`. Analizar distintas formas de implementación (usando un if-then-else, y 2 if), ¿todas funcionan?

3. `devolver_el_doble_si_es_multiplo3_el_triple_si_es_multiplo9(un_numero)`. En otro caso devolver el número original. Analizar distintas formas de implementación (usando un if-then-else, y 2 if, usando alguna opción de operación lógica), ¿todas funcionan?.
4. Dado un nombre, si la longitud es igual o mayor a 5 devolver una frase que diga "Tu nombre tiene muchas letras!" y sino, "Tu nombre tiene menos de 5 caracteres".
5. En Argentina una persona del sexo femenino se jubila a los 60 años, mientras que aquellas del sexo masculino se jubilan a los 65 años. Quienes son menores de 18 años se deben ir de vacaciones junto al grupo que se jubila. Al resto de las personas se les ordena ir a trabajar. Implemente una función que, dados los parámetros de sexo (F o M) y edad, imprima la frase que corresponda según el caso: "Andá de vacaciones" o "Te toca trabajar".

Ejercicio 6. Implementar las siguientes funciones usando repetición condicional `while`:

1. Escribir una función que imprima los números del 1 al 10.
2. Escribir una función que imprima los números pares entre el 10 y el 40.
3. Escribir una función que imprima la palabra "eco" 10 veces.
4. Escribir una función de cuenta regresiva para lanzar un cohete. Dicha función irá imprimiendo desde el número que me pasan por parámetro (que será positivo) hasta el 1, y por último "Despegue".
5. Hacer una función que monitoree un viaje en el tiempo. Dicha función recibe dos parámetros, "el año de partida" y "algún año de llegada", siendo este último parámetro siempre más chico que el primero. El viaje se realizará de a saltos de un año y la función debe mostrar el texto: "Viajó un año al pasado, estamos en el año: <año>" cada vez que se realice un salto de año.
6. Implementar de nuevo la función de monitoreo de viaje en el tiempo, pero desde el año de partida hasta *lo más cercano* al 384 a.C., donde conoceremos a Aristóteles. Y para que sea más rápido el viaje, ¡vamos a viajar de a 20 años en cada salto!

Ejercicio 7. Implementar las funciones del ejercicio 6 utilizando `for num in range(i,f,p):`. Recordar que la función `range` para generar una secuencia de números en un rango dado, con un valor inicial i, un valor final f y un paso p. Ver documentación: `https://docs.python.org/es/3/library/stdtypes.html#typesseq-range`

Ejercicio 8. Realizar la ejecución simbólica de los siguientes códigos:

1. `x=5 ; y=7`
2. `x=5 ; y=7 ; z=x+y`
3. `x=5 ; x=''hora''`
4. `x=True ; y=False ; res=x and y`
5. `x=False ; res=not(x)`

Ejercicio 9. Sea el siguiente código:

```
def rt(x: int, g: int) -> int:
  g = g + 1
  return x + g

g: int = 0
def ro(x: int) -> int:
  global g
  g = g + 1
  return x + g
```

1. ¿Cuál es el resultado de evaluar tres veces seguidas `ro(1)`?
2. ¿Cuál es el resultado de evaluar tres veces seguidas `rt(1, 0)`?
3. En cada función, realizar la ejecución simbólica.
4. Dar la especificación en lenguaje natural para cada función, rt y ro.

Introducción a la Programación

Guía Práctica 8
Funciones sobre listas (tipos complejos)

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

Recordar usar las anotaciones de tipado en todas las variables. Por ejemplo: `def funcion(numero: int) ->bool:`.

En los ejercicios se pueden usar funciones matemáticas como por ejemplo: `sqrt, round, floor, ceil, %`. Ver especificaciones de dichas funciones en la documentación de Python: `https://docs.python.org/es/3.10/library/math.html` y `https://docs.python.org/es/3/library/functions.html`

Revisar la especificación de las operaciones comunes sobre secuencias: `https://docs.python.org/es/3/library/stdtypes.html?highlight=list#typesseq`

1. Primera Parte

Ejercicio 1. Codificar en Python las siguientes funciones sobre secuencias:

Nota: Cada problema puede tener más de una implementación. Probar utilizando distintas formas de recorrido sobre secuencias, y distintas funciones de Python. No te conformes con una solución, recordar que siempre conviene consultar con tus docentes.

1. `problema pertenece` (in s:seq<$\mathbb{Z}$>, in e: $\mathbb{Z}$) : Bool {
   `requiere:` { True }
   `asegura:` { $res = true \leftrightarrow (\exists i : \mathbb{Z})(0 \leq i < |s| \rightarrow s[i] = e)$}
   }

   Implementar al menos de 3 formas distintas éste problema.

   ¿Si la especificaramos e implementaramos con tipos genéricos, se podría usar esta misma función para buscar un caracter dentro de un string?

2. `problema divideATodos` (in s:seq<$\mathbb{Z}$>, in e: $\mathbb{Z}$) : Bool {
   `requiere:` {$e \neq 0$ }
   `asegura:` { $res = true \leftrightarrow (\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow s[i] \ mod \ e = 0)$}
   }

3. `problema sumaTotal` (in s:seq<$\mathbb{Z}$>) : $\mathbb{Z}$ {
   `requiere:` { True }
   `asegura:` { res es la suma de todos los elementos de s}
   }

   Nota: no utilizar la función `sum()` nativa

4. `problema ordenados` (in s:seq<$\mathbb{Z}$>) : Bool {
   `requiere:` { True }
   `asegura:` { $res = true \leftrightarrow (\forall i : \mathbb{Z})(0 \leq i < (|s| - 1) \rightarrow s[i] < s[i + 1]$}
   }

5. Dada una lista de palabras, devolver verdadero si alguna palabra tiene longitud mayor a 7.

6. Dada una cadena de texto (string), devolver verdadero si ésta es palíndroma (se lee igual en ambos sentidos), falso en caso contrario.

7. Analizar la fortaleza de una contraseña. El parámetro de entrada será un string con la contraseña a analizar, y la salida otro string con tres posibles valores: VERDE, AMARILLA y ROJA. **Nota:** en python la "ñ/Ñ" es considerado un caracter especial y no se comporta como cualquier otra letra.

   - La contraseña será VERDE si:
     a) la longitud es mayor a 8 caracteres
     b) Tiene al menos 1 letra minúscula (probar qué hace `''a''<=''A''<=''z''`)
     c) Tiene al menos 1 letra mayúscula Tiene al menos 1 letra minúscula (probar qué hace `''A''<=''A''<=''Z''`)
     d) Tiene al menos 1 dígito numérico (0..9)
   - La contraseña será ROJA si:
     a) la longitud es menor a 5 caracteres.
   - En caso contrario será AMARILLA.

8. Dada una lista de tuplas, que representa un historial de movimientos en una cuenta bancaria, devolver el saldo actual. Asumir que el saldo inicial es 0. Las tuplas tienen una letra que nos indica el tipo de movimiento "I" para ingreso de dinero y "R" para retiro de dinero, y además el monto de cada operación. Por ejemplo `[(''I'', 2000), (''R'', 20),(''R'', 1000),(''I'', 300)]` → `saldo = 1280`.

9. Recorrer una palabra y devolver True si ésta tiene al menos 3 vocales distintas. En caso contrario devolver False.

2. Segunda Parte

Ejercicio 2. Implementar las siguientes funciones sobre secuencias pasadas por parámetro:

1. Implementar una función que dada una lista de números, en las posiciones pares borra el valor original y coloca un cero. Esta función modifica el parámetro ingresado. **Nota:** La lista será un tipo inout.

2. Implementar la función del punto anterior pero esta vez sin modificar la lista original, devolviendo una nueva lista, igual a la anterior pero con las posiciones pares en cero. **Nota:** La lista será de tipo in.

3. Implementar una función que dada una cadena de texto de entrada (in) devuelva una cadena igual a la anterior, pero sin las vocales. **Nota:** No agrega espacios, sino que borra la vocal y concatena a continuación.

4. `problema reemplazaVocales` (in s:seq<Char>) : seq<Char> {
   `requiere:` { True }
   `asegura:` { $(\forall i : \mathbb{Z})(0 \leq i < |res| \rightarrow$ (pertenece(<'a','e','i','o','u'>, s[i]) $\wedge$ res[i] = '_') $\vee$
   ($\neg$ pertenece(<'a','e','i','o','u'>, s[i]) $\wedge$ res[i] = s[i])) }
   }

5. `problema daVueltaStr` (in s:seq<Char>) : seq<Char> {
   `requiere:` { True }
   `asegura:` { $(\forall i : \mathbb{Z})(0 \leq i < |res| \rightarrow$ (res[i]=s[|s|-i-1]) }
   }

Ejercicio 3. Vamos a elaborar programas interactivos (usando la función `input()`[1]) que nos permita solicitar al usuario información cuando usamos las funciones.

1. Implementar una función para construir una lista con los nombres de mis estudiantes. La función solicitará al usuario los nombres hasta que ingrese la palabra "listo". Devuelve la lista con todos los nombres ingresados.

2. Implementar una función que devuelve una lista con el historial de un monedero electrónico (por ejemplo la SUBE). El usuario debe seleccionar en cada paso si quiere:

   - "C" = Cargar créditos,
   - "D" = Descontar créditos,
   - "X" = Finalizar la simulación (terminar el programa).

[1] `https://docs.python.org/es/3/library/functions.html?highlight=input#input`

En los casos de cargar y descontar créditos, el programa debe además solicitar el monto para la operación. Vamos a asumir que el monedero comienza en cero. Para guardar la información grabaremos en el historial tuplas que representen los casos de cargar ("C", monto a cargar) y descontar crédito ("D", monto a descontar).

3. Vamos a escribir un programa para simular el juego conocido como 7 y medio. El mismo deberá generar un número aleatorio entre 0 y 12 (excluyendo el 8 y 9) y deberá luego preguntarle al usuario si desea seguir sacando otra "carta" o plantarse. En este último caso el programa debe terminar. Los números aleatorios obtenidos deberán sumarse según el número obtenido salvo por las "figuras" (10,11 y 12) que sumarán medio punto cada una. El programa deberá ir acumulando los valores y si se pasa de 7.5 deberá informar que el usuario ha perdido. Al finalizar la función devuelve el historial de "cartas" que hizo que el usuario gane o pierda. Nota: Para esta función utilizaremos la función `random.randint(1,12)` para generar números aleatorios entre 1 y 12. **Nota:** La función random.choice() puede ser de gran ayuda a la hora de repartir cartas.

Ejercicio 4. Implementar las siguientes funciones sobre listas de listas:

1. `problema perteneceACadaUno` (in s:seq<seq<$\mathbb{Z}$>>, in e:$\mathbb{Z}$, out res: seq<Bool>) {
   `requiere:` { True }
   `asegura:` { $(\forall i : \mathbb{Z})(0 \leq i < |res| \rightarrow$ (res[i] = true $\leftrightarrow$ pertenece(s[i],e))) }
   }

   Nota: Reutilizar la función `pertenece()` implementada previamente para listas

2. `problema esMatriz` (in s:seq<seq<$\mathbb{Z}$>>) : Bool {
   `requiere:` { True }
   `asegura:` { res = true $\leftrightarrow$ $(|s| > 0) \wedge (|s[0]| > 0) \wedge (\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow |s[i]| = |s[0]|)$ }
   }

3. `problema filasOrdenadas` (in m:seq<seq<$\mathbb{Z}$>>, out res: seq<Bool>) {
   `requiere:` { esMatriz(m)}
   `asegura:` { $(\forall i : \mathbb{Z})(0 \leq i < |res| \rightarrow$ (res[i] = true $\leftrightarrow$ ordenados(s[i]))) }
   }

   Nota: Reutilizar la función `ordenados()` implementada previamente para listas

4. Implementar una función que tome un entero d y un float p y eleve una matriz cuadrada de tamaño d (con valores generados al azar) a la potencia p. Es decir, multiplique a la matriz por sí misma p veces. Luedo de implementarla probar con diferentes valores de d. ¿Qué pasa con valores muy grandes?

   Nota 1: recordá que en la multiplicación de una matriz cuadrada de dimensión d por si misma cada posición se calcula como res[i][j] = $\sum_{n=0}^{d-1} m[i][n] * m[n][j]$

   Nota 2: para generar una matriz cuadrada de dimensión d con valores aleatorios hay muchas opciones de implementación, analizar las siguientes usando el módulo numpy (ver recuadro):

   Opción 1:

   ```
   import numpy as np
   m = np.random.random((d, d))
   ```
   [2]

   Opción 2:

   ```
   import numpy as np
   m = np.random.randint(i,f, (d, d))
   ```
   [3]

   Para poder importar la librería `numpy` es necesario instalarla. Para ello es necesario tener instalado un gestor de paquetes, por ejemplo `pip` (**Ubuntu:** `sudo apt install pip3`. **Windows:** se instala junto con Python). Una vez instalado `pip` se ejecuta `pip install numpy`.

[2] https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.random.html#numpy.random.Generator.random
[3] https://numpy.org/doc/stable/reference/random/generated/numpy.random.randint.html

Introducción a la Programación

Guía Práctica 9
Testing de caja blanca

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

Ejercicio 1. ★ Sea el siguiente programa:

```
def max (x: int, y: int) -> int:
L1:     result: int = 0
L2:     if x < y:
L3:         result = y
        else:
L4:         result = x
L5:     return result
```

Y los siguientes casos de test:

- test1:
  - Entrada x = 0, y=0
  - Resultado esperado result=0
- test2:
  - Entrada x = 0, y=1
  - Resultado esperado result=1

1. Describir el diagrama de control de flujo (control-flow graph) del programa `max`.
2. Detallar qué líneas del programa cubre cada test

Test	L1	L2	L3	L4	L5
test1					
test2					

3. Detallar qué decisiones (branches) del programa cubre cada test

Test	L2-True	L2-False
test1		
test2		

4. Decidir si la siguiente afirmación es verdadera o falsa: *"El test suite compuesto por test1 y test2 cubre el 100 % de las líneas del programa y el 100 % de las decisiones (branches) del programa"*. Justificar.

Ejercicio 2. ★ Sea la siguiente especificación del problema de retornar el mínimo elemento entre dos números enteros:

`problema min` (in x: $\mathbb{Z}$, in y: $\mathbb{Z}$) : $\mathbb{Z}$ {
`requiere:` $\{True\}$
`asegura:` $\{(x < y \rightarrow result = x) \wedge (x \geq y \rightarrow result = y)\}$
}

Un programador ha escrito el siguiente programa para implementar la especificación descripta:

```
def min (x: int, y: int) -> int:
L1:     result: int = 0
L2:     if x < y:
L3:         result = x
        else:
L4:         result = x
L5:     return result
```

Y el siguiente conjunto de casos de test (test suite):

- minA:
  - Entrada x=0,y=1
  - Salida esperada 0
- minB:
  - Entrada x=1,y=1
  - Salida esperada 1

1. Describir el diagrama de control de flujo (control-flow graph) del programa `min`.
2. ¿La ejecución del test suite resulta en la ejecución de todas las líneas del programa `min`?
3. ¿La ejecución del test suite resulta en la ejecución de todas las decisiones (branches) del programa?
4. ¿Es el test suite capaz de detectar el defecto de la implementación del problema de encontrar el mínimo?
5. Agregar nuevos casos de tests y/o modificar casos de tests existentes para que el test suite detecte el defecto.

Ejercicio 3. ★ Sea la siguiente especificación del problema de sumar y una posible implementación en lenguaje imperativo.

```
problema sumar (in x: ℤ, in y: ℤ) : ℤ {
      requiere: {True}
      asegura: {result = x + y}
}
```

```
def sumar (x: int, y: int) -> int:
L1:    result: int = 0
L2:    result = result + x
L3:    result = result + y
L4:    return result
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `sumar`.
2. Escribir un conjunto de casos de test (o "test suite") que ejecute todas las líneas del programa `sumar`.

Ejercicio 4. Sea la siguiente especificación del problema de restar y una posible implementación en lenguaje imperativo:

```
problema restar (in x: ℤ, in y: ℤ) : ℤ {
      requiere: {True}
      asegura: {result = x - y}
}
```

```
def restar (x: int, y: int) -> int:
L1:    result: int = 0
L2:    result = result + x
L3:    result = result + y
L4:    return result
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `restar`.
2. Escribir un conjunto de casos de test (o "test suite") que ejecute todas las líneas del programa `restar`.
3. La línea L3 del programa `restar` tiene un defecto, ¿es el test suite descripto en el punto anterior capaz de detectarlo? En caso contrario, modificar o agregar nuevos casos de test hasta lograr detectarlo.

Ejercicio 5. Sea la siguiente especificación del problema de `signo` y una posible implementación en lenguaje imperativo:

```
problema signo (in x: ℝ) : ℤ {
      requiere: {True}
      asegura: {(result = 0 ∧ x = 0) ∨ (result = −1 ∧ x < 0) ∨ (result = 1 ∧ x > 0)}
}
```

```
def signo(x: float) -> int:
L1:    result: int = 0
L2:    if x<0:
L3:       result = −1
L4:    elif x>0:
L5:       result = 1
L6:    return result
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `signo`.
2. Escribir un test suite que ejecute todas las líneas del programa `signo`.
3. ¿El test suite del punto anterior ejecuta todas las posibles decisiones ("branches") del programa?

Ejercicio 6. Sea la siguiente especificación del problema de `signo` y una posible implementación en lenguaje imperativo:

```
problema fabs (in x: ℝ) : ℝ {
      requiere: {True}
      asegura: {result = |x|}
}
```

```
  def fabs(x: float) -> float:
  L1:    result: float = 0
  L2:    if x<0:
  L3:       result = −x
  L4:    return result
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `signo`.
2. Escribir un test suite que ejecute todas las líneas del programa `signo`.
3. Escribir un test suite que ejecute todas las posibles decisiones ("branches") del programa.
4. Escribir un test suite que ejecute todas las líneas del programa pero no ejecute todos las decisiones del programa.
5. ¿Los test suites de los puntos anteriores detectan el defecto en la implementación? De no ser así, modificarlos para que lo hagan.

Ejercicio 7. ★ Sea la siguiente especificación:

```
problema fabs (in x: ℤ) : ℤ {
      requiere: {True}
      asegura: {result = |x|}
}
```

Y la siguiente implementación:

```
def fabs(x: int) -> int:
L1:    if x < 0:
L2:       return −x
       else:
L3:       return +x
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `fabs`.

2. Escribir un test suite que ejecute todas las líneas y todos los branches del programa.

Ejercicio 8. ★ Sea la siguiente especificación del problema de `mult10` y una posible implementación en lenguaje imperativo:

```
problema mult10 (in x: ℤ) : ℤ {
      requiere: {True}
      asegura: {result = x * 10}
}
```

```
def mult10(x: int) -> int:
L1:   result: int = 0
L2:   count: int = 0
L3:   while(count < 10):
L4:       result = result + x
L5:       count = count + 1
L6:   return result
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `mult10`.
2. Escribir un test suite que ejecute todas las líneas del programa `mult10`.
3. ¿El test suite anterior ejecuta todas las posibles decisiones ("branches") del programa?

Ejercicio 9. Sea la siguiente especificación del problema de `sumar` y una posible implementación en lenguaje imperativo:

```
problema sumar (in x: ℤ, in y: ℤ) : ℤ {
      requiere: {True}
      asegura: {result = x + y}
}
```

```
def sumar(x: int , y: int) -> int:
L1:    sumando: int = 0
L2:    abs_y: int = 0
L3:    if y < 0:
L4:      sumando = −1
L5:       abs_y = −y
       else:
L7:      sumando = 1
L8:      abs_y = y
L9:    result: int = x
L10:   count: int = 0
L11:   while(count < abs_y):
L12:       result = result + sumando
L13:       count = count + 1
L14:   return result
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `sumar`.
2. Escribir un test suite que ejecute todas las líneas del programa `sumar`.
3. Escribir un test suite que ejecute todas las posibles decisiones ("branches") del programa.

Ejercicio 10. Sea el siguiente programa que computa el máximo común divisor entre dos enteros.

```
def mcd(x: int , y: int) -> int:
L1:   assert (x >= 0 & y >= 0) # requiere: x e y tienen que ser no negativos
L2:   tmp: int = 0
L3:   while(y != 0):
L4:     tmp = x % y
L5:     x = y
L6:     y = tmp
L7:   return x
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `mcd`.
2. Escribir un test suite que ejecute todas las líneas y todos los branches del programa.

Ejercicio 11. ★ Sea el siguiente programa que retorna diferentes valores dependiendo si a, b y c, definen lados de un triángulo inválido, equilátero, isósceles o escaleno.

```
def triangle(a: int , b: int, c: int) -> int:
L1:     if(a <= 0 | b <= 0 | c <= 0):
L2:         return 4 # invalido
L3:     if(not ((a + b > c) & (a + c > b) & (b + c > a))):
L4:         return 4 # invalido
L5:     if(a == b & b == c):
L6:         return 1 # equilatero
L7:     if(a == b | b == c | a == c):
L8:         return 2 # isosceles
L9:     return 3  # escaleno
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `triangle`.
2. Escribir un test suite que ejecute todas las líneas y todos los branches del programa.

Ejercicio 12. ★ Sea la siguiente especificación del problema de `multByAbs` y una posible implementación en lenguaje imperativo:

`problema multByAbs` (in x: $\mathbb{Z}$, in y:$\mathbb{Z}$, out result: $\mathbb{Z}$) {
`requiere:` $\{True\}$
`asegura:` $\{result = x * |y|\}$
}

```
def multByAbs(x: int , y: int) -> int:
L1:     abs_y: int = fabs(y); # ejercicio anterior
L2:     if abs_y < 0:
L3:         return -1
L4:     else:
L5:         result: int = 0;
L6:         i: int = 0;
L7:         while i < abs_y:
L8:             result = result + x;
L9:             i += 1
L10:     return result
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `multByAbs`.
2. Detallar qué líneas y branches del programa no pueden ser cubiertos por ningún caso de test. ¿A qué se debe?
3. Escribir el test suite que cubra todas las líneas y branches que puedan ser cubiertos.

Ejercicio 13. Sea la siguiente especificación del problema de `vaciarSecuencia` y una posible implementación en lenguaje imperativo:

`problema vaciarSecuencia` (inout s: $seq\langle\mathbb{Z}\rangle$) {
`modifica:` {s}
`asegura:` $\{|s| = |x@pre| \land (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] = 0)\}$
}

```
def vaciarSecuencia(s: [int]):
L1:     for i in range(len(s)):
L2:         s[i] = 0
```

1. Escribir el diagrama de control de flujo (control-flow graph) del programa `vaciarSecuencia`.
2. Escribir un test suite que cubra todos las líneas de programa.
3. En caso que el test suite del punto anterior no cubriera todo los branches del programa, extenderlo de modo que logre cubrirlos.

Ejercicio 14. Sea la siguiente especificación del problema de `existeElemento` y una posible implementación en lenguaje imperativo:

```
problema existeElemento (s: seq⟨ℤ⟩, e: ℤ) : Bool {
    requiere: {True}
    asegura: {result = True ↔ (∃j : ℤ)(0 ≤ j < |s| ∧ s[j] = e)}
}
```

```
def existeElemento(s: [int], e: int) -> bool:
L1:     result: bool = False
L2:     for i in range(len(s)):
L3:         if s[i] == e:
L4:             result = True
L5:             break
L6:     return result
```

1. Escribir el diagrama de control de flujo (control-flow graph) del programa `existeElemento`.
2. Escribir un test suite que cubra todos las líneas de programa (observar que un `for` contiene 3 líneas distintas)
3. En caso que el test suite del punto anterior no cubriera todo los branches del programa, extenderlo de modo que logre cubrirlos.

Ejercicio 15. Sea la siguiente especificación del problema de `cantidadDePrimos` y una posible implementación en lenguaje imperativo:

```
problema cantidadDePrimos (in n: ℤ) : ℤ {
    requiere: {n ≥ 0}
    asegura: {result = Σ_{i=2}^{n} (if esPrimo(i) then 1 else 0 fi)}
}
```

```
        def cantidadDePrimos(n: int) -> int:
L1:         result: int = 0
L2,L3,L4:   for i in range(2,n+1,1):
L5:             inc: bool = esPrimo(i)
L6:             if inc==True:
L7:                 result += 1
L8:         return result

        #Funcion auxiliar
        def esPrimo(x: int) -> bool:
L9:         result: bool = True
L10,L11,L12:    for i in range(2, x, 1):
L13:            if x % i == 0:
L14:                result = False
L15:        return result
```

1. Escribir los diagramas de control de flujo (control-flow graph) para `cantidadDePrimos` y la función auxiliar `esPrimo`.
2. Escribir un test suite que cubra todos las líneas de programa del programa `cantidadDePrimos` y `esPrimo`.
3. En caso que el test suite del punto anterior no cubriera todo los branches del programa, extenderlo de modo que logre cubrirlos.

Ejercicio 16. Sea la siguiente especificación del problema de `esSubsecuencia` y una posible implementación en lenguaje imperativo:

in s: $seq\langle \text{Char} \rangle \mathbb{Z}$

`problema esSubsecuencia` (s: $seq\langle \mathbb{Z} \rangle$, r: $seq\langle \mathbb{Z} \rangle$) : Bool {
`requiere:` $\{True\}$
`asegura:` $\{result = True \leftrightarrow |r| \leq |s|$
$\wedge_L (\exists i : \mathbb{Z})((0 \leq i < |s| \wedge i + |r| < |s|) \wedge_L (\forall j : \mathbb{Z})(0 \leq j < |r| \rightarrow_L s[i + j] = r[j])))\}$
}

```
1  def esSubsecuencia(s: [int], r: [int]) -> bool:
2      result: bool = False
3      ultimoIndice: int = len(s) - len(r)
4      for i in range(ultimoIndice + 1):
5          # obtener una subsecuencia de s
6          subseq: [int] = subsecuencia(s, i, len(r))
7          # chequear si la subsecuencia es igual a r
8          sonIguales: [int] = iguales(subseq, r)
9          if sonIguales:
10             result = True
11             break
12     return result
13
14 # procedimiento auxiliar subsecuencia
15 def subsecuencia(s: [int], desde: int, longitud: int) -> [int]:
16     rv: [int] = []
17     hasta: int = desde + longitud
18     for i in range(desde, hasta):
19         elem = s[i]
20         rv.append(elem)
21     return rv
22
23 # procedimiento auxiliar iguales
24 def iguales(a: [int], b: [int]) -> bool:
25     result: bool = True
26     if len(a) == len(b):
27         for i in range(len(a)):
28             if a[i] != b[i]:
29                 result = False
30                 break
31     else:
32         result = False
33     return result
```

1. Escribir los diagramas de control de flujo (control-flow graph) para `esSubsecuencia` y las funcones auxiliares `subsecuencia` e `iguales`.
2. Escribir un test suite que cubra todos las líneas de programa *ejecutables* de todos los procedimientos. Observar que un `for` contiene 3 líneas distintas.
3. En caso que el test suite del punto anterior no cubriera todo los branches del programa, extenderlo de modo que logre cubrirlos.

Ejercicio 17. El programa que se transcribe más abajo pretende determinar la longitud del *fragmento* más largo en un texto. Los fragmentos son porciones del texto que no contienen punto y coma. Por ejemplo, en el siguiente texto el fragmento más largo es `"Mercurio"`, de ocho letras:

`"Mercurio;Venus;Tierra;Marte;Júpiter"`

La especificación formal es la siguiente:

`problema calcularFragmentoMásLargo` (s: $seq\langle \text{Char} \rangle$) : $\mathbb{Z}$ {

requiere: $\{True\}$
asegura: $\{(\exists i, j : \mathbb{Z})(\texttt{esFragmento}(s, i, j) \land n \leq j - i) \land (\forall i, j : \mathbb{Z})(\texttt{esFragmento}(s, i, j) \longrightarrow n \geq j - i)\}$
}

pred esFragmento (s: $seq\langle \mathsf{Char} \rangle$, $i, j : \mathbb{Z}$) {
$0 \leq i \leq j \leq |s| \land_L (\forall k : \mathbb{Z})(i \leq k < j \longrightarrow_L \neg \mathsf{esPuntoYComa}(s[k]))$
}
pred esPuntoYComa (c: Char) {
$c = \text{';'}$
}

El programa es el siguiente:

```
def calcular_fragmento_mas_largo(s: str) -> int:
        n: int = 0
        i: int = 0
        f: int = 0 # Longitud del fragmento actual
        while i < len(s):
                if f > n:
                        n = f
                if s[i] == ';':
                        f = 0
                else:
                        f+=1
                i+=1
        return n
```

1. Escribir el CFG (control-flow graph) de la función.
2. Escribir *un test* que encuentre el defecto presente en el código. Es decir, escribir una entrada que cumple con el *requiere* pero que el resultado de ejecutar el código no cumple el *asegura* (Justificar la respuesta).
3. Agregar casos de test para cubrir todos los branches del programa.

Introducción a la Programación

Guía Práctica 10
Archivos, Pilas, Colas y Diccionarios

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

1. Archivos

Ejercicio 1. Implementar en python:

1. una función `contarlineas(in nombre_archivo : str) → int` que cuenta la cantidad de líneas de texto del archivo dado
2. una función `existePalabra(in palabra : str, in nombre_archivo : str) → bool` que dice si una palabra existe en un archivo de texto o no
3. una función `cantidadApariciones(in nombre_archivo : str, in palabra : str) → int` que devuelve la cantidad de apariciones de una palabra en un archivo de texto

Ejercicio 2. Dado un archivo de texto con comentarios, implementar una función `clonarSinComentarios(innombre_archivo : str)` que toma un archivo de entrada y genera un nuevo archivo que tiene el contenido original sin las líneas comentadas. Para este ejercicio vamos a considerar comentarios como aquellas líneas que tienen un caracter # como primer caracter de la línea, o si no es el primer caracter, se cumple que todos los anteriores son espacios.
Ejemplo:

```
# esto es un comentario
    # esto tambien
esto no es un comentario # esto tampoco
```

Ejercicio 3. Dado un archivo de texto, implementar una función que escribe un archivo nuevo (‘reverso.txt‘) que tiene las mismas líneas que el original, pero en el orden inverso.
Ejemplo: si el archivo original es

```
Esta es la primer linea.
Y esta es la segunda.
```

debe generar:

```
Y esta es la segunda.
Esta es la primer linea.
```

Ejercicio 4. Dado un archivo de texto y una frase (es decir, texto que puede estar separado por ’\n’), implementar una función que la agregue al final del archivo original (sin hacer una copia).

Ejercicio 5. idem, pero agregando la frase al comienzo del archivo original (de nuevo, sin hacer una copia del archivo).

Ejercicio 6. Implementar una función que lea un archivo en modo *binario* y devuelva la lista de ’palabras legibles’. Vamos a definir una palabra legible como

- secuencias de texto formadas por numeros, letras mayusculas/minusculas y los caracteres ’ ’(espacio) y ’_’(guion bajo)
- que tienen longitud $>= 5$

Una vez implementada la función, probarla con diferentes archivos binarios (.exe, .zip, .wav, .mp3, etc).
Referencia: https://docs.python.org/es/3/library/functions.html#open

Ejercicio 7. Implementar una función que lea un archivo de texto separado por comas (comma-separated values, o .csv) que contiene las notas de toda la carrera de un grupo de alumnos y calcule el promedio final de un alumno dado. La función `promedioEstudiante(in lu : str) → float`. El archivo tiene el siguiente formato:

```
nro de LU (str), materia (str), fecha (str), nota (float)
```

2. Pilas

Ejercicio 8. Implementar una función `generarNrosAlAzar(in n : int, in desde : int, in hasta : int) → list[int]` que genere una lista de n numeros enteros al azar en el rango $[desde, hasta]$. Pueden user la función `random.sample()`

Ejercicio 9. Usando la función del punto anterior, implementar otra función que arme una pila con los numeros generados al azar. Pueden usar la clase `LifoQueue()` que es un ejemplo de una implementación básica:

```
from queue import LifoQueue as Pila

p = Pila()
p.put(1) # apilar
elemento = p.get() # desapilar
p.empty() # vacia?
```

Ejercicio 10. Implementar una función `cantidadElementos(in p : pila) → int` que, dada una pila, cuente la cantidad de elementos que contiene.

Ejercicio 11. Dada una pila de enteros, implementar una función `buscarElMaximo(in p : pila) → int` que devuelva el máximo elemento.

Ejercicio 12. Implementar una función `estaBienBalanceada(in s : str) → bool` que dado un string con una formula aritmética sobre los enteros, diga si los paréntesis estan bien balanceados. Las fórmulas pueden formarse con:

- los numeros enteros
- las operaciones basicas $+$, $-$, x y $/$
- parentesis
- espacios

Entonces las siguientes son formulas aritméticas con sus paréntesis bien balanceados:

$1 + (2 \text{ x } 3 - (20 / 5))$
$10 * (1 + (2 * (-1)))$

Y la siguiente es una formula que no tiene los paréntesis bien balanceados:

$1 +) \ 2 \text{ x } 3 \ (\ ()$

3. Colas

Ejercicio 13. Usando la función `generarNrosAlAzar()` definida en la sección anterior, implementar una función que arme una cola de enteros con los numeros generados al azar. Pueden usar la clase `Queue()` que es un ejemplo de una implementación básica:

```
from queue import Queue as Cola

c = Cola()
c.put(1) # encolar
elemento = c.get() # desencolar()
c.empty() # vacia?
```

Ejercicio 14. Implementar una función `cantidadElementos(in c : cola) → int` que, dada una cola, cuente la cantidad de elementos que contiene. Comparar con la versión usando pila.

Ejercicio 15. Dada una cola de enteros, implementar una función `buscarElMaximo(in c : cola) → int` que devuelva el máximo elemento. Comparar con la versiń usando pila.

Ejercicio 16. Bingo: un cartón de bingo contiene 12 números al azar en el rango $[0, 99]$.

1. implementar una función `armarSecuenciaDeBingo() → Cola[int]` que genere una cola con los números del 0 al 99 ordenados al azar.
2. implementar una función `jugarCartonDeBingo(in carton : list[int], in bolillero : cola[int]) → int` que toma un cartón de Bingo y una cola de enteros (que corresponden a las bolillas numeradas) y determina cual es la cantidad de jugadas de ese bolillero que se necesitan para ganar.

Ejercicio 17. Vamos a modelar una guardia de un hospital usando una cola donde se van almacenando los pedidos de atención para los pacientes que van llegando. A cada paciente se le asigna una prioridad del 1 al 10 (donde la prioridad 1 es la mas urgente y requiere atención inmediata) junto con su nombre y la especialidad medica que le corresponde.

Implementar la función `nPacientesUrgentes(in c : Cola[(int, str, str)]) → int` que devuelve la cantidad de pacientes de la cola que tienen prioridad en el rango $[1, 3]$.

4. Diccionarios

Para esta sección vamos a usar el tipo `dict` que nos provee python:

Ejercicio 18. Leer un archivo de texto y agrupar la cantidad de palabras de acuerdo a su longitud. Implementar la función `agruparPorLongitud(in nombre_archivo : str) → dict` que devuelve un diccionario {`longitud_en_letras : cantidad_de_palabras`}.

Ej el diccionario

```
{
       1: 2,
       2: 10,
       5: 4
}
```

indica que se encontraron 2 palabras de longitud 1, 10 palabras de longitud 2 y 5 palabras de longitud 4. Para este ejercicio vamos a considerar palabras a todas aquellas secuencias de caracteres que no tengan espacios en blanco.

Ejercicio 19. Volver a implementar la función que calcula el promedio de las notas de los alumnos, pero ahora devolver un diccionario {`libreta_universitaria : promedio`} con los promedios de todos los alumnos.

Ejercicio 20. Implementar la función `laPalabraMasFrecuente(in nombre_archivo : str) → str` que devuelve la palabra que más veces aparece en un archivo de texto.