# CI3641 - Lenguajes de Programación I

## Examen 1 — *30 puntos*

1. Escoja algún lenguaje de programación de propósito general cuyo nombre empiece con la misma letra que su nombre (por ejemplo, si su nombre es "Pedro", podría escoger "Perl", "PLI", "Python", etc.).

   (a) *De una breve descripción del lenguaje escogido.*

   MATLAB es un lenguaje que no se puede encasillar en uno de los paradigmas de computación, ya que su misión es precisamente ser un lenguaje multiparadigma. Generalmente sigue un flujo de instrucciones secuencial por lo que muchas veces se le considera un lenguaje imperativo, pero cuenta con características de lenguajes funcionales como la aplicación de funciones matemáticas sobre matrices. Esto es particularmente importante porque MATLAB fue diseñado para que estudiantes pudieran acceder a ciertas librerías sin la necesidad de tener conocimiento previo de FORTRAN.

   Actualmente es conocido por ser utilizado para la enseñanza, especialmente en el área de álgebra lineal.

   i. *Diga qué tipo de alcances y asociaciones posee, argumentando las ventajas y desventajas de la decisión tomada por los diseñadores del lenguaje, en el contexto de sus usuarios objetivos.*

   MATLAB tiene un alcance dinámico, lo que es acorde al objetivo del lenguaje. Este fue diseñado para facilitar el uso de ciertas librerías, y en la actualidad para la enseñanza sobre todo de álgebra. Es comúnmente utilizado el interpretador para realizar cómputos rápidos, por lo que las verificaciones semánticas se hacen a tiempo de ejecución. Es conocido por ser un lenguaje de tipado débil, que nuevamente tiene sentido porque era más importante la facilidad de uso que su eficiencia.

   Asimismo, posee una asociación tardía. MATLAB no realiza ningún tipo de verificación antes de ejecutar el código del programa. Cuando intenta ejecutar una instrucción donde un nombre tiene el tipo incorrecto, genera un error en tiempo de ejecución.

   ii. *Diga que tipo de módulos ofrece (de tenerlos) y las diferentes formas de importar y exportar nombres.*

   En MATLAB los módulos son llamados *"Packages"*. Estos *Packages* funcionan como **librerías**, ya que reciben múltiples instancias de un tipo de datos para que sean gestionadas. Para importar los *Packages* se utiliza la palabra reservada `import` seguido del nombre del paquete. A través de la notación

de punto, podems importar clases o funciones de dicho paquete, por ejemplo `import MyPackage.MyClass` o `import MyPackage.MyFunction`.

iii. *Enumere y explique las estructuras de control de flujo que ofrece.*

A. `if-elseif-else-end`: Le permite al usuario ejecutar un fragmento de código cuando una condición se cumple. No necesita llaves pero pueden ser indicadas y son recomendadas porque mejoran la legibilidad del código. Es importante destacar la presencia de la palabra `end` al final del bloque para indicar que la última condición ha finalizado su ejecución.

B. `while loop`: Ejecuta un fragmento de código mientras una condición se cumpla. Cabe acotar que la matriz nula es considerada un valor "falso".

C. `for loop`: Ejecuta un bloque de código tantas veces como se haya definido en la firma del ciclo. En matlab el iterador es un vector que contiene todos los valores que debe tener en la ejecución del ciclo. Si se desea, el iterador puede contener una vector como valor si en el rango se especifica una matriz. Es útil cuando se quieren aplicar distintas funciones a distintas filas de una matriz.
Tanto `while loop` como `for loop` soportan las palabras claves `break` para detener la ejecución del ciclo como `continue` para avanzar hasta la próxima iteración del mismo.

D. `switch`: Ejecuta un fragmento de código dependiendo del valor de una variable. A diferencia de la mayoría de los lenguajes, este operador no necesita una palabra clave para no ejecutar el resto de los casos. Al entrar en uno de ellos y ejecutarlo, finaliza la ejecución de la estrctura completamente sin ejecutar el contenido de ninguno de los otros casos. El *catch-all* case es indicado com la palabra reservada `otherwise`. (Muy *Haskelloso*).

iv. *Diga en qué orden evalúan expresiones y funciones.*
Las expresiones y funciones evalúan de izquierda a derecha, respetando la precedencia definida por sus operadores. Te dejo aquí el link de la precedencia: Precedencia en MATLAB.
La evaluación de funciones también se ejecuta de izquierda a derecha.

(b) *Implemente los siguientes programas en el lenguaje escogido:*

i. *Dado un entero no–negativo n, calcular el factorial de n.*

$$fact(n) = \begin{cases} 1 & si \quad n = 0 \\ n \times fact(n-1) & si \quad n > 0 \end{cases}$$

ii. *Dadas dos matrices $A$ y $B$ (cuyas dimensiones son $N \times M$ y $M \times P$, respectivamente), calcular su producto $A \times B$ (cuya dimensión es $N \times P$).*

$$\forall i \in [1..N], \forall j \in [1..P] : (A \times B)_{i,j} = \sum_{k \in [1..M]} A_{i,k} \times B_{k,j}$$

Puedes encontrar las respuestas a ambas en Repositorio de Github :)

2. *Considere el siguiente programa escrito en pseudo-código:*

```
int x = X + 1, y = Y;

proc ohno(int x) {
    y := 2 * x;
}

proc ohwell(int y, proc waitwhat) {
    if (y < 2 * (X + 1)) {
        proc ohno(int x) {
            x := y * 2;
        }
        ohwell(y + 2 * (X + 1), waitwhat);
    } else if (y < 4 * (X + 1)) {
        ohwell(y + 2 * (X + 1), ohno);
    } else {
        int x = Z;
        waitwhat(x + y);
    }
    print(x, y)
}

ohwell(x, ohno);
print(x, y)
```

*Note que deberá reemplazar los valores para X, Y y Z como fue explicado en los párrafos de introducción del examen.*

Mi programa quedaría de la siguiente manera:

```
int x = 5, y = 6;

proc ohno(int x) {
    y := 2 * x;
}

proc ohwell(int y, proc waitwhat) {
    if (y < 10) {
        proc ohno(int x) {
            x := y * 2;
        }
        ohwell(y + 10, waitwhat);
    } else if (y < 20) {
        ohwell(y + 10, ohno);
    } else {
        int x = 3;
        waitwhat(x + y);
    }
    print(x, y)
}

ohwell(x, ohno);
print(x, y)
```

*Diga qué imprime el programa en cuestión, si el lenguaje tiene:*

(a) *Alcance estático y asociación profunda*

(b) *Alcance dinámico y asociación profunda*

(c) *Alcance estático y asociación superficial*

(d) *Alcance dinámico y asociación superficial*

Te adjunto el PDF con la representanción gráfica de las corridas. También lo puedes encontrar en el repo de github.

3. *Pregunta 3*

   No coloqué el enunciado porque no aportaba nada para la respuesta. Puedes encontrar la implementación aquí.

4. *Considere los siguientes iteradores, escritos en Python:*

(a) *El iterador* `zip`:

```
def zip(a, b):
    if a and b:
        yield (a[0], b[0])
        for p in zip(a[1:], b[1:]):
            yield p
```

*Considere también el siguiente fragmento de código que hace uso del iterador 'zip':*

```
for p in zip([1, 2, 3], ['a', 'b', 'c']):
    print(p)
```

i. *Ejecute, paso a paso, el fragmento de código mostrado (por lo menos al nivel de cada nuevo marco de pila creado) y muestre lo que imprime.*

Para solucionar esto, seguiré el mismo procedimiento que hiciste en clases con los números de línea:

```
1 def zip(a, b):
2     if a and b:
3         yield (a[0], b[0])
4         for p in zip(a[1:], b[1:]):
5             yield p
6
7 for p in zip([1, 2, 3], ['a', 'b', 'c']):
8     print(p)
```

Nota: No encontré una forma bonita de solo marcar las líneas que ejecutan acciones, entonces usaré los números de las líneas que si ejecutan y omitiré los otros. Las líneas que ejecutan instrucciones son: 2, 3, 4, 5, 7 y 8.

A continuación represento los marcos de pila como:
`<NOMBRE DE MARCO>: [<contenido> -> <valor>,]`
y las escrituras en consola como:
`IMPRIME> <valor>`

**Paso 1:**
`GLOBAL: zip -> proc, p -> -, pc -> 7`
**Paso 2:**
`GLOBAL: zip -> proc, p -> -, pc -> 7`
`ZIP (1): a -> [1,2,3], b -> ['a','b','c'], pc -> 2`
**Paso 3:**
`GLOBAL: zip -> proc, p -> -, pc -> 7`

```
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], pc -> 3
```

**Paso 4:**

```
GLOBAL: zip -> proc, p -> (1, 'a'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], pc -> 3
```

**Paso 5:**

```
IMPRIME: (1, 'a')
GLOBAL: zip -> proc, p -> (1, 'a'), pc -> 8
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], pc -> 3
```

**Paso 6:**

```
GLOBAL: zip -> proc, p -> (1, 'a'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], pc -> 3
```

**Paso 7:**

```
GLOBAL: zip -> proc, p -> (1, 'a'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> -, pc -> 4
```

**Paso 8:**

```
GLOBAL: zip -> proc, p -> (1, 'a'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> -, pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], pc -> 2
```

**Paso 9:**

```
GLOBAL: zip -> proc, p -> (1, 'a'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> -, pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], pc -> 3
```

**Paso 10:**

```
GLOBAL: zip -> proc, p -> (1, 'a'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (2, 'b'), pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], pc -> 3
```

**Paso 11:**

```
GLOBAL: zip -> proc, p -> (2, 'b'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (2, 'b'), pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], pc -> 3
```

**Paso 12:**

```
IMPRIME: (2, 'b')
GLOBAL: zip -> proc, p -> (2, 'b'), pc -> 8
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (2, 'b'), pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], pc -> 3
```

**Paso 13:**

```
GLOBAL: zip -> proc, p -> (2, 'b'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (2, 'b'), pc -> 4
```

```
ZIP (2): a -> [2,3], b -> ['b','c'], pc -> 3
```

**Paso 14:**

```
GLOBAL: zip -> proc, p -> (2, 'b'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (2, 'b'), pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], p -> -, pc -> 4
```

**Paso 15:**

```
GLOBAL: zip -> proc, p -> (2, 'b'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (2, 'b'), pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], p -> -, pc -> 4
ZIP (3): a -> [3], b -> ['c'], pc -> 2
```

**Paso 16:**

```
GLOBAL: zip -> proc, p -> (2, 'b'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (2, 'b'), pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], p -> -, pc -> 4
ZIP (3): a -> [3], b -> ['c'], pc -> 3
```

**Paso 17:**

```
GLOBAL: zip -> proc, p -> (2, 'b'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (2, 'b'), pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], p -> (3, 'c'), pc -> 4
ZIP (3): a -> [3], b -> ['c'], pc -> 3
```

**Paso 18:**

```
GLOBAL: zip -> proc, p -> (2, 'b'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (3, 'c'), pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], p -> (3, 'c'), pc -> 5
ZIP (3): a -> [3], b -> ['c'], pc -> 3
```

**Paso 19:**

```
GLOBAL: zip -> proc, p -> (3, 'c'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (3, 'c'), pc -> 5
ZIP (2): a -> [2,3], b -> ['b','c'], p -> (3, 'c'), pc -> 5
ZIP (3): a -> [3], b -> ['c'], pc -> 3
```

**Paso 20:**

```
IMPRIME: (3, 'c')
GLOBAL: zip -> proc, p -> (3, 'c'), pc -> 8
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (3, 'c'), pc -> 5
ZIP (2): a -> [2,3], b -> ['b','c'], p -> (3, 'c'), pc -> 5
ZIP (3): a -> [3], b -> ['c'], pc -> 3
```

**Paso 21:**

```
GLOBAL: zip -> proc, p -> (3, 'c'), pc -> 7
```

```
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (3, 'c'), pc -> 5
ZIP (2): a -> [2,3], b -> ['b','c'], p -> (3, 'c'), pc -> 5
ZIP (3): a -> [3], b -> ['c'], pc -> 3
```

**Paso 22:**

```
GLOBAL: zip -> proc, p -> (3, 'c'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (3, 'c'), pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], p -> (3, 'c'), pc -> 5
ZIP (3): a -> [3], b -> ['c'], pc -> 3
```

**Paso 23:**

```
GLOBAL: zip -> proc, p -> (3, 'c'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (3, 'c'), pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], p -> (3, 'c'), pc -> 4
ZIP (3): a -> [3], b -> ['c'], pc -> 3
```

**Paso 24:**

```
GLOBAL: zip -> proc, p -> (3, 'c'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (3, 'c'), pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], p -> (3, 'c'), pc -> 4
ZIP (3): a -> [3], b -> ['c'], p -> -, pc -> 4
```

**Paso 25:**

```
GLOBAL: zip -> proc, p -> (3, 'c'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (3, 'c'), pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], p -> (3, 'c'), pc -> 4
ZIP (3): a -> [3], b -> ['c'], p -> -, pc -> 4
ZIP (4): a -> [], b -> [], pc -> 2
```

**Paso 26:**

```
GLOBAL: zip -> proc, p -> (3, 'c'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (3, 'c'), pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], p -> (3, 'c'), pc -> 4
ZIP (3): a -> [3], b -> ['c'], p -> -, pc -> 4
ZIP (4): a -> [], b -> [], pc -> 3
```

**Paso 27:**

```
GLOBAL: zip -> proc, p -> (3, 'c'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (3, 'c'), pc -> 4
ZIP (2): a -> [2,3], b -> ['b','c'], p -> (3, 'c'), pc -> 4
ZIP (3): a -> [3], b -> ['c'], p -> -, pc -> 4
```

**Paso 28:**

```
GLOBAL: zip -> proc, p -> (3, 'c'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (3, 'c'), pc -> 4
```

```
ZIP (2): a -> [2,3], b -> ['b','c'], p -> (3, 'c'), pc -> 4
```

**Paso 29:**

```
GLOBAL: zip -> proc, p -> (3, 'c'), pc -> 7
ZIP (1): a -> [1,2,3], b -> ['a','b','c'], p -> (3, 'c'), pc -> 4
```

**Paso 30:**

```
GLOBAL: zip -> proc, p -> (3, 'c'), pc -> 7
```

ii. *Modifique el iterador* `zip` *de tal forma que obtenga un nuevo iterador* `zipWith`*, que recibirá como tercer argumento la función que usará para unir cada par de elementos.*

Puedes encontrar la respuesta en el Repositorio de Github :)

(b) *El iterador* `misterio`*, que hace uso de* `zipWith`*:*

```
def misterio(p):
  yield p
  acum = []
  for p in zipWith([0, *p], [*p, 0], lambda x, y: x + y):
    acum +=[p]
  for m in misterio(acum):
    yield m
```

*Considere también el siguiente fragmento de código que hace uso del iterador misterio:*

```
for m in misterio([1]):
    print p
```

i. *Ejecute, paso a paso, el fragmento de código mostrado (por lo menos al nivel de cada nuevo marco de pila creado) y muestre lo que imprime. Deberá reportar únicamente los primeros 7 elementos que imprime y dar un estimado sobre la cantidad de elementos que imprimiría el fragmento de código si se le permitiese continuar con su ejecución. Pista: Los elementos generados por este iterador serán listas.*

```
1  def zipWith(a, b, func):
2      if a and b:
3          yield func(a[0], b[0])
4          for p in zipWith(a[1:], b[1:], func):
5              yield p
6
7  def misterio(p):
8      yield p
9      acum = []
```

```
10      for p in zipWith([0, *p], [*p, 0], lambda x, y: x + y):
11          acum += [p]
12      for m in misterio(acum):
13          yield m
14
15  for m in misterio([1]):
16      print(m)
```

```
# PASO 1
GLOBAL: misterio -> proc, m -> -, pc -> 15

# PASO 2
GLOBAL: misterio -> proc, m -> -, pc -> 15
MISTERIO (1): p -> [1], pc -> 8

# PASO 3
IMPRIME> [1] # 1ER ELEMENTO
GLOBAL: misterio -> proc, m -> [1], pc -> 16
MISTERIO (1): p -> [1], pc -> 8

# PASO 4
GLOBAL: misterio -> proc, m -> [1], pc -> 15
MISTERIO (1): p -> [1], acum -> [], pc -> 10

# PASO 5
GLOBAL: misterio -> proc, m -> [1], pc -> 15
MISTERIO (1): p -> [1], acum -> [], pc -> 10
ZIPWITH: a -> [0, 1], b -> [1, 0], func -> (lambda x, y: x + y),
         pc -> 2

# PASO 6
GLOBAL: misterio -> proc, m -> [1], pc -> 15
MISTERIO (1): p -> 1, acum -> [1,1], pc -> 12

# PASO 7
GLOBAL: misterio -> proc, m -> [1], pc -> 15
MISTERIO (1): p -> 1, acum -> [1, 1], m -> -, pc -> 12
MISTERIO (2): p -> [1, 1], pc -> 8
```

```
# PASO 8
IMPRIME> [1, 1] # 2DO ELEMENTO
GLOBAL: misterio -> proc, m -> [1, 1], pc -> 16
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 1], pc -> 13
MISTERIO (2): p -> [1, 1], pc -> 8

# PASO 9
GLOBAL: misterio -> proc, m -> [1, 1], pc -> 15
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 1], pc -> 12
MISTERIO (2): p -> [1, 1], acum -> [], pc -> 10
ZIPWITH: a -> [0, 1, 1], b -> [1, 1, 0],
         func -> (lambda x, y: x + y), pc -> 2

# PASO 10
GLOBAL: misterio -> proc, m -> [1, 1], pc -> 15
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 1], pc -> 12
MISTERIO (2): p -> 1, acum -> [1, 2, 1], pc -> 12

# PASO 11
GLOBAL: misterio -> proc, m -> [1, 1], pc -> 15
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 1], pc -> 12
MISTERIO (2): p -> 1, acum -> [1, 2, 1], m -> -, pc -> 12
MISTERIO (3): p -> [1, 2, 1], pc -> 8

# PASO 12
IMPRIME> [1, 2, 1] # 3ER ELEMENTO
GLOBAL: misterio -> proc, m -> [1, 2, 1], pc -> 16
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 2, 1], pc -> 13
MISTERIO (2): p -> 1, acum -> [1, 2, 1], m -> [1, 2, 1], pc -> 13
MISTERIO (3): p -> [1, 2, 1], pc -> 8

# PASO 13
GLOBAL: misterio -> proc, m -> [1, 2, 1], pc -> 15
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 2, 1], pc -> 12
MISTERIO (2): p -> 1, acum -> [1, 2, 1], m -> [1, 2, 1], pc -> 12
MISTERIO (3): p -> [1, 2, 1], acum -> [], pc -> 10
ZIPWITH: a -> [0, 1, 2, 1], b -> [1, 2, 1, 0],
         func -> (lambda x, y: x + y), pc -> 2
```

```
# PASO 14
GLOBAL: misterio -> proc, m -> [1, 2, 1], pc -> 15
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 2, 1], pc -> 12
MISTERIO (2): p -> 1, acum -> [1, 2, 1], m -> [1, 2, 1], pc -> 12
MISTERIO (3): p -> 1, acum -> [1, 3, 3, 1], pc -> 12

# PASO 15
GLOBAL: misterio -> proc, m -> [1, 2, 1], pc -> 15
MISTERIO (1): p -> [1], acum -> [1, 1], m -> [1, 2, 1],
              pc -> 12
MISTERIO (2): p -> 1, acum -> [1, 2, 1], m -> [1, 2, 1],
              pc -> 12
MISTERIO (3): p -> [1, 2, 1], acum -> [1, 3, 3, 1],
              m -> -, pc -> 12
MISTERIO (4): p -> [1, 3, 3, 1], pc -> 8

# PASO 16
IMPRIME> [1, 3, 3, 1] # 4TO ELEMENTO
GLOBAL: misterio -> proc, m -> [1, 3, 3, 1], pc -> 16
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 3, 3, 1],
              pc -> 13
MISTERIO (2): p -> 1, acum -> [1, 2, 1], m -> [1, 3, 3, 1],
              pc -> 13
MISTERIO (3): p -> 1, acum -> [1, 3, 3, 1], m -> [1, 3, 3, 1],
              pc -> 13
MISTERIO (4): p -> [1, 3, 3, 1], pc -> 8

# PASO 17
GLOBAL: misterio -> proc, m -> [1, 3, 3, 1], pc -> 15
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 3, 3, 1],
              pc -> 12
MISTERIO (2): p -> 1, acum -> [1, 2, 1], m -> [1, 3, 3, 1],
              pc -> 12
MISTERIO (3): p -> 1, acum -> [1, 3, 3, 1], m -> [1, 3, 3, 1],
              pc -> 12
MISTERIO (4): p -> [1, 3, 3, 1], acum -> [], pc -> 10
```

```
# PASO 18
GLOBAL: misterio -> proc, m -> [1, 3, 3, 1], pc -> 15
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 3, 3, 1],
              pc -> 12
MISTERIO (2): p -> 1, acum -> [1, 2, 1], m -> [1, 3, 3, 1],
              pc -> 12
MISTERIO (3): p -> 1, acum -> [1, 3, 3, 1], m -> [1, 3, 3, 1],
              pc -> 12
MISTERIO (4): p -> [1, 3, 3, 1], acum -> [], pc -> 10
ZIPWITH: a -> [0, 1, 3, 3, 1], b -> [1, 3, 3, 1, 0],
         func -> (lambda x, y: x + y), pc -> 2

# PASO 19
GLOBAL: misterio -> proc, m -> [1, 3, 3, 1], pc -> 15
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 3, 3, 1],
              pc -> 12
MISTERIO (2): p -> 1, acum -> [1, 2, 1], m -> [1, 3, 3, 1],
              pc -> 12
MISTERIO (3): p -> 1, acum -> [1, 3, 3, 1], m -> [1, 3, 3, 1],
              pc -> 12
MISTERIO (4): p -> 1, acum -> [1, 4, 6, 4, 1], pc -> 12

# PASO 20
GLOBAL: misterio -> proc, m -> [1, 3, 3, 1], pc -> 15
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 3, 3, 1],
              pc -> 12
MISTERIO (2): p -> 1, acum -> [1, 2, 1], m -> [1, 3, 3, 1],
              pc -> 12
MISTERIO (3): p -> 1, acum -> [1, 3, 3, 1], m -> [1, 3, 3, 1],
              pc -> 12
MISTERIO (4): p -> 1, acum -> [1, 4, 6, 4, 1], m -> -, pc -> 12
MISTERIO (5): p -> [1, 4, 6, 4, 1], pc -> 8

# PASO 21
IMPRIME> [1, 4, 6, 4, 1] # 5TO ELEMENTO
GLOBAL: misterio -> proc, m -> [1, 4, 6, 4, 1], pc -> 16
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 4, 6, 4, 1],
              pc -> 13
```

```
MISTERIO (2): p -> 1, acum -> [1, 2, 1], m -> [1, 4, 6, 4, 1],
              pc -> 13
MISTERIO (3): p -> 1, acum -> [1, 3, 3, 1], m -> [1, 4, 6, 4, 1],
              pc -> 13
MISTERIO (4): p -> 1, acum -> [1, 4, 6, 4, 1], m -> [1, 4, 6, 4, 1],
              pc -> 13
MISTERIO (5): p -> [1, 4, 6, 4, 1], pc -> 8

# PASO 22
GLOBAL: misterio -> proc, m -> [1, 4, 6, 4, 1], pc -> 15
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 4, 6, 4, 1],
              pc -> 12
MISTERIO (2): p -> 1, acum -> [1, 2, 1], m -> [1, 4, 6, 4, 1],
              pc -> 12
MISTERIO (3): p -> 1, acum -> [1, 3, 3, 1], m -> [1, 4, 6, 4, 1],
              pc -> 12
MISTERIO (4): p -> 1, acum -> [1, 4, 6, 4, 1], m -> [1, 4, 6, 4, 1],
              pc -> 12
MISTERIO (5): p -> [1, 4, 6, 4, 1], acum -> [], pc -> 10
ZIPWITH: a -> [0, 1, 4, 6, 4, 1], b -> [1, 4, 6, 4, 1, 0],
         func -> (lambda x, y: x + y), pc -> 2

# PASO 23
IMRPIME> [1, 5, 10, 10, 5, 1] # 6TO ELEMENTO
GLOBAL: misterio -> proc, m -> [1, 5, 10, 10, 5, 1], pc -> 16
MISTERIO (1): p -> 1, acum -> [1, 1], m -> [1, 5, 10, 10, 5, 1],
              pc -> 13
MISTERIO (2): p -> 1, acum -> [1, 2, 1], m -> [1, 5, 10, 10, 5, 1],
              pc -> 13
MISTERIO (3): p -> 1, acum -> [1, 3, 3, 1],
              m -> [1, 5, 10, 10, 5, 1], pc -> 13
MISTERIO (4): p -> 1, acum -> [1, 4, 6, 4, 1],
              m -> [1, 5, 10, 10, 5, 1], pc -> 13
MISTERIO (5): p -> 1, acum -> [1, 5, 10, 10, 5, 1],
              m -> [1, 5, 10, 10, 5, 1] pc -> 13
MISTERIO (6): p -> [1, 5, 10, 10, 5, 1], pc -> 8

# PASO 24
```

```
GLOBAL: misterio -> proc, m -> [1, 5, 10, 10, 5, 1], pc -> 15
MISTERIO (1): p -> 1, acum -> [1, 1],
              m -> [1, 5, 10, 10, 5, 1], pc -> 12
MISTERIO (2): p -> 1, acum -> [1, 2, 1],
              m -> [1, 5, 10, 10, 5, 1], pc -> 12
MISTERIO (3): p -> 1, acum -> [1, 3, 3, 1],
              m -> [1, 5, 10, 10, 5, 1], pc -> 12
MISTERIO (4): p -> 1, acum -> [1, 4, 6, 4, 1],
              m -> [1, 5, 10, 10, 5, 1], pc -> 12
MISTERIO (5): p -> 1, acum -> [1, 5, 10, 10, 5, 1],
              m -> [1, 5, 10, 10, 5, 1], pc -> 12
MISTERIO (6): p -> [1, 5, 10, 10, 5, 1], acum -> [],
              pc -> 10
ZIPWITH: a -> [0, 1, 5, 10, 10, 5, 1],
         b -> [1, 5, 10, 10, 5, 1, 0],
         func -> (lambda x, y: x + y), pc -> 2

# PASO 25
GLOBAL: misterio -> proc, m -> [1, 5, 10, 10, 5, 1], pc -> 15
MISTERIO (1): p -> 1, acum -> [1, 1],
              m -> [1, 5, 10, 10, 5, 1], pc -> 12
MISTERIO (2): p -> 1, acum -> [1, 2, 1],
              m -> [1, 5, 10, 10, 5, 1], pc -> 12
MISTERIO (3): p -> 1, acum -> [1, 3, 3, 1],
              m -> [1, 5, 10, 10, 5, 1], pc -> 12
MISTERIO (4): p -> 1, acum -> [1, 4, 6, 4, 1],
              m -> [1, 5, 10, 10, 5, 1], pc -> 12
MISTERIO (5): p -> 1, acum -> [1, 5, 10, 10, 5, 1],
              m -> [1, 5, 10, 10, 5, 1], pc -> 12
MISTERIO (6): p -> 1, acum -> [1, 6, 15, 20, 15, 6, 1],
              m -> -, pc -> 12
MISTERIO (7): p -> [1, 6, 15, 20, 15, 6, 1], pc -> 8

# PASO 26
IMPRIME> [1, 6, 15, 20, 15, 6, 1] # 7MO ELEMENTO
GLOBAL: misterio -> proc, m -> [1, 6, 15, 20, 15, 6, 1], pc -> 16
MISTERIO (1): p -> 1, acum -> [1, 1],
              m -> [1, 6, 15, 20, 15, 6, 1], pc -> 13
```

```
MISTERIO (2): p -> 1, acum -> [1, 2, 1],
              m -> [1, 6, 15, 20, 15, 6, 1], pc -> 13
MISTERIO (3): p -> 1, acum -> [1, 3, 3, 1],
              m -> [1, 6, 15, 20, 15, 6, 1], pc -> 13
MISTERIO (4): p -> 1, acum -> [1, 4, 6, 4, 1],
              m -> [1, 6, 15, 20, 15, 6, 1], pc -> 13
MISTERIO (5): p -> 1, acum -> [1, 5, 10, 10, 5, 1],
              m -> [1, 6, 15, 20, 15, 6, 1], pc -> 13
MISTERIO (6): p -> 1, acum -> [1, 6, 15, 20, 15, 6, 1],
              m -> [1, 6, 15, 20, 15, 6, 1], pc -> 13
MISTERIO (7): p -> [1, 6, 15, 20, 15, 6, 1], pc -> 8
```

El iterador `misterio` calcula los números del triágunlo de Pascal, donde cada elemento que genera el iterador corresponde a un nivel de profundidad del mismo. De tener una máquina con memoria infinita el iterador generaría niveles infinitamente. En la práctica genera números hasta que alcance el nivel máximo de recursión que el *heap* le permite.

ii. *Explique, a grandes rasgos, cómo funciona el iterador misterio y qué colección de elementos conocida está generando. Diga cómo aprovecha el iterador* `zipWith` *para generar la colección deseada.*

Su funcionamiento comienza mientras se le provee un nivel del triángulo, cualquiera que fuese. En el ejemplo se provee con el primero([1]), pero pudiese ser cualquier otro. Realmente se pudiese computar cualquier pirámide que siga el mismo formato de construcción de más niveles a partir de la suma de los elementos del nivel anterior. Para cada nivel que se va generando se va almacenando en la variable `acum` el valor correspondiente a ese nivel, de manera que el siguiente pueda ser computado. Se agregan el valor 0 al final y al inicio de las listas que son pasadas como parámetros a `zipWith` para emular la suma de los extremos de un nivel del triangulo, que en la teoría no se suma con nadie (que es lo mismo que sumar con cero). `zipWith` se hace conveniente con estos parámetros ya que emula sumar cada elemento con el siguiente en la lista, como una especie de *shift* de los elementos del arreglo.

iii. *Modifique el iterador* `misterio` *de tal forma que obtenga un nuevo iterador supenso, que devolverá cada elemento de las listas generadas por* `misterio` *de forma aplanada.*

Puedes encontrar la respuesta en el Repositorio de Github :)

5. *Pregunta 5*

Igual que para la pregunta 3, no coloqué el enunciado porque no aportaba nada para la respuesta. Puedes encontrar la implementación aquí.

No hice las comparativas con gráficos, pero te dejo aquí unas pequeñas conclusiones que saqué solo usando el comando `time`:

```
# PARTE A
$ time ./parte-a
Enter an integer: 200
Result: 138235122./parte-a  0.04s user 0.00s system 3% cpu 1.269 total

$ time ./parte-a
Enter an integer: 250
Result: 2051595540./parte-a  2.53s user 0.00s system 68% cpu 3.679 total

$ time ./parte-a
Enter an integer: 270
Result: 1121408704./parte-a  15.30s user 0.00s system 94% cpu 16.212 total

# PARTE B
$ time ./parte-b
Enter an integer: 200

Result: 138235122
./parte-b  0.00s user 0.00s system 0% cpu 2.472 total

$ time ./parte-b
Enter an integer: 250

Result: 2051595540
./parte-b  0.00s user 0.00s system 0% cpu 0.940 total

$ time ./parte-b
Enter an integer: 300

Result: -1447243264
./parte-b  0.00s user 0.00s system 0% cpu 1.032 total

$ time ./parte-b
```

```
Enter an integer: 900

Result: 1592745412
./parte-b  0.00s user 0.00s system 0% cpu 1.230 total

# PARTE C
$ time ./parte-c
Enter an integer: 200

Result: 138235122
./parte-c  0.00s user 0.00s system 0% cpu 0.958 total

CI3641 - Lenguajes de Programación I/ci3641-examen1/pregunta5  main
$ time ./parte-c
Enter an integer: 250

Result: 2051595540
./parte-c  0.00s user 0.00s system 0% cpu 1.022 total

CI3641 - Lenguajes de Programación I/ci3641-examen1/pregunta5  main
$ time ./parte-c
Enter an integer: 300

Result: -1447243264
./parte-c  0.00s user 0.00s system 0% cpu 1.595 total

CI3641 - Lenguajes de Programación I/ci3641-examen1/pregunta5  main
$ time ./parte-c
Enter an integer: 900

Result: 1592745412
./parte-c  0.00s user 0.00s system 0% cpu 1.981 total
```

Entre los scripts *B* (recursión de cola)y *C* (iterativo) no hay diferencia notable, inclusive si uso unos numerps un poco más grandes que los que te muestro aquí. Para *A* (recursión, NO de cola) si vemos la diferencia, ya para calcular la función evaluada en 270 el uso del CPU aumenta hasta 94%, lo cual tiene todo el sentido del mundo ya que tiene que guardar toda la infomación del marco de cada invocación.

6. *Pregunta 6*

   Igual que para las preguntas 3 y 5, no coloqué el enunciado porque no aportaba nada para la respuesta. Puedes encontrar la implementación aquí.