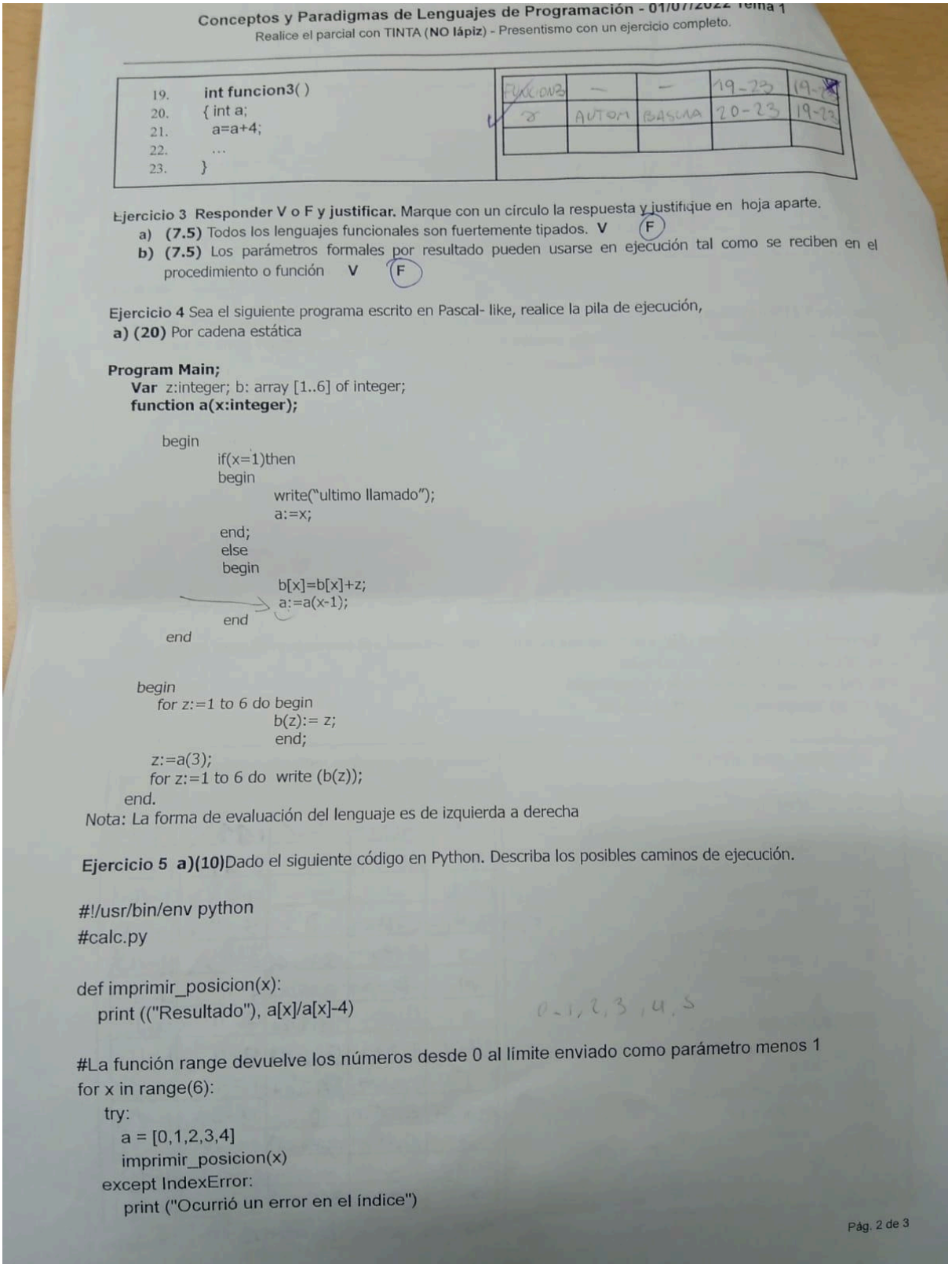
CPLP - 1er parcial - Parciales viejos

01/07/2022 - Tema 1



Conceptos y Paradigmas de Lenguajes de Programación - 01/07/2022 Tema 1

Realice el parcial con TINTA (**NO lápiz**) - Presentismo con un ejercicio completo.

```
19.  int funcion3( )
20.  { int a;
21.    a=a+4;
22.    …
23.  }
```

FUNCION3	—	—	19-23	19-23
3	AUTOM	BASICA	20-23	19-23

Ejercicio 3 Responder V o F y justificar. Marque con un círculo la respuesta y justifique en hoja aparte.

a) **(7.5)** Todos los lenguajes funcionales son fuertemente tipados. **V** (F)

b) **(7.5)** Los parámetros formales por resultado pueden usarse en ejecución tal como se reciben en el procedimiento o función **V** (F)

Ejercicio 4 Sea el siguiente programa escrito en Pascal- like, realice la pila de ejecución,

a) (20) Por cadena estática

```
Program Main;
  Var z:integer; b: array [1..6] of integer;
  function a(x:integer);

    begin
        if(x=1)then
        begin
            write("ultimo llamado");
            a:=x;
        end;
        else
        begin
            b[x]=b[x]+z;
            a:=a(x-1);
        end
    end


  begin
    for z:=1 to 6 do begin
                b(z):= z;
                end;
    z:=a(3);
    for z:=1 to 6 do  write (b(z));
  end.
```

Nota: La forma de evaluación del lenguaje es de izquierda a derecha

Ejercicio 5 a)(10) Dado el siguiente código en Python. Describa los posibles caminos de ejecución.

```
#!/usr/bin/env python
#calc.py

def imprimir_posicion(x):
   print (("Resultado"), a[x]/a[x]-4)

#La función range devuelve los números desde 0 al límite enviado como parámetro menos 1
for x in range(6):
    try:
      a = [0,1,2,3,4]
      imprimir_posicion(x)
    except IndexError:
      print ("Ocurrió un error en el índice")
```

0, 1, 2, 3, 4, 5

```
except ZeroDivisionError:
  print ("Ocurrió una división por cero")
else:
  print ("Se pudo acceder correctamente")
finally:
  print ("Vuelve a probar")
```

b)(10p). Indique cuáles son los tipos de datos identificados en el siguiente código C. Justifique en cada caso

```
#include <stdio.h>
#include <string.h>

struct Punto{
  int x;
  int y;
};

union estudiante
{
        char nombre[20];
        char apellido[20];
        float promedio;
};

int sumaPuntos(Punto p)
{
    int result;
    result = p.x + p.y;
    return result;
 }
```

Ejercicio 3

a) Falso, en general lo son, pero podrían no serlo, ya que no es una característica excluyente de los lenguajes funcionales.
b) Falso, no pueden usarse en ejecución tal como se reciben, ya que los mismos no están inicializados, y usarlos sin inicializarlos (tal como se reciben), implicaría provocar un error de semántica, al hacer uso de un parámetro no inicializado.

Ejercicio 4

CPLP - 2do parcial - Parciales viejos

Ejercicio 5

a)

El programa se ejecutará de la siguiente manera:

- Para el número 0 (de la variable y), al intentar dividir a[y]/a[y]-4, es decir, 0/0-4 (la división tiene prioridad sobre la suma), ocurre una excepción del tipo ZeroDivisionError, por lo que se ejecuta el manejador definido para dicho tipo de excepción, que imprime “Ocurrió una división por cero” y luego “Vuelve a probar” (cláusula finally)
- Para los números (de la variable y) del 1 al 4 no hay problema y se imprime:
    - “Resultado -3.0”
    - “Resultado -3.0”
    - “Resultado -3.0”
    - “Resultado -3.0”
- Y luego de cada una de las impresiones se imprime “Se pudo acceder correctamente” (cláusula else) y “Vuelve a probar” (cláusula finally)
- Para el número 5 (de la variable y), dado que la lista a no tiene una posición 5, ocurre una excepción de tipo IndexError, por lo que se ejecuta el manejador definido para dicho tipo de excepción, que imprime “Ocurrió un error en el índice” y luego “Vuelve a probar” (cláusula finally)

b)

```
struct Punto {
        int x;
        int y;
};
```

Es un tipo de datos compuesto definido por el usuario, cuyo constructor es el producto cartesiano, ya que relaciona diferentes elementos de varios conjuntos (en este caso son 2 conjuntos, ambos enteros(?).

```
union estudiante {
        char nombre[20];
        char apellido[20];
        long dni;
}
```

Es un tipo de datos compuesto definido por el usuario, cuyo constructor es el de unión. Esto significa que tan solo uno de los campos puede tener valor en un mismo momento (los campos son mutuamente excluyentes).

```
int sumaPuntos(Punto p){
        int result;
        result = p.x + p.y;
        return result;
}
```

Es una función, la cual es un tipo de datos compuesto definido por el usuario, cuyo constructor es la correspondencia finita, puesto que, para cada valor de un conjunto finito (para cada valor posible del struct Coordenadas), se devolverá un y sólo un valor que se corresponda con el parámetro ingresado.

01/07/2022 - Tema 2

Conceptos y Paradigmas de Lenguajes de Programación - 01/07/2022 Tema 2

Realice el parcial con TINTA (**NO lápiz**) - Presentismo con un ejercicio completo.

```
17.    ...
18.  }
19.  int funcion3( )
20.  { int b;
21.    b=b+4;
22.    ...
23.  }
```

FALTA → b(L.20)

FUNCION3	—	—	20 -23	12 -23
				20 -23

Ejercicio 3 Responder V o F y justificar. Marque con un círculo la respuesta y justifique en hoja aparte.

a) (7.5) El polimorfismo radica en definir el mismo método con distintos parámetros **V** **F**

b) (7.5) La unión discriminada es menos segura que la unión **V** **F**

Ejercicio 4 Sea el siguiente programa escrito en Pascal- like, realice la pila de ejecución,

a) (20) Por cadena dinámica

```
Program Main;
  Var z:integer; b: array [1..6] of integer;
  function a(y:integer);
     begin
          if(y=1)then
          begin
               write("caso base");
               a:=y;
          end;
          else
          begin
               b[y]=b[y]*z;
               a:=a(y-1);
          end
     end


   begin
     for z:=1 to 6 do begin
                    b(z):= z;
                    end;
     z:=a(3);
     for z:=1 to 6 do  write (b(z));
   end.
```

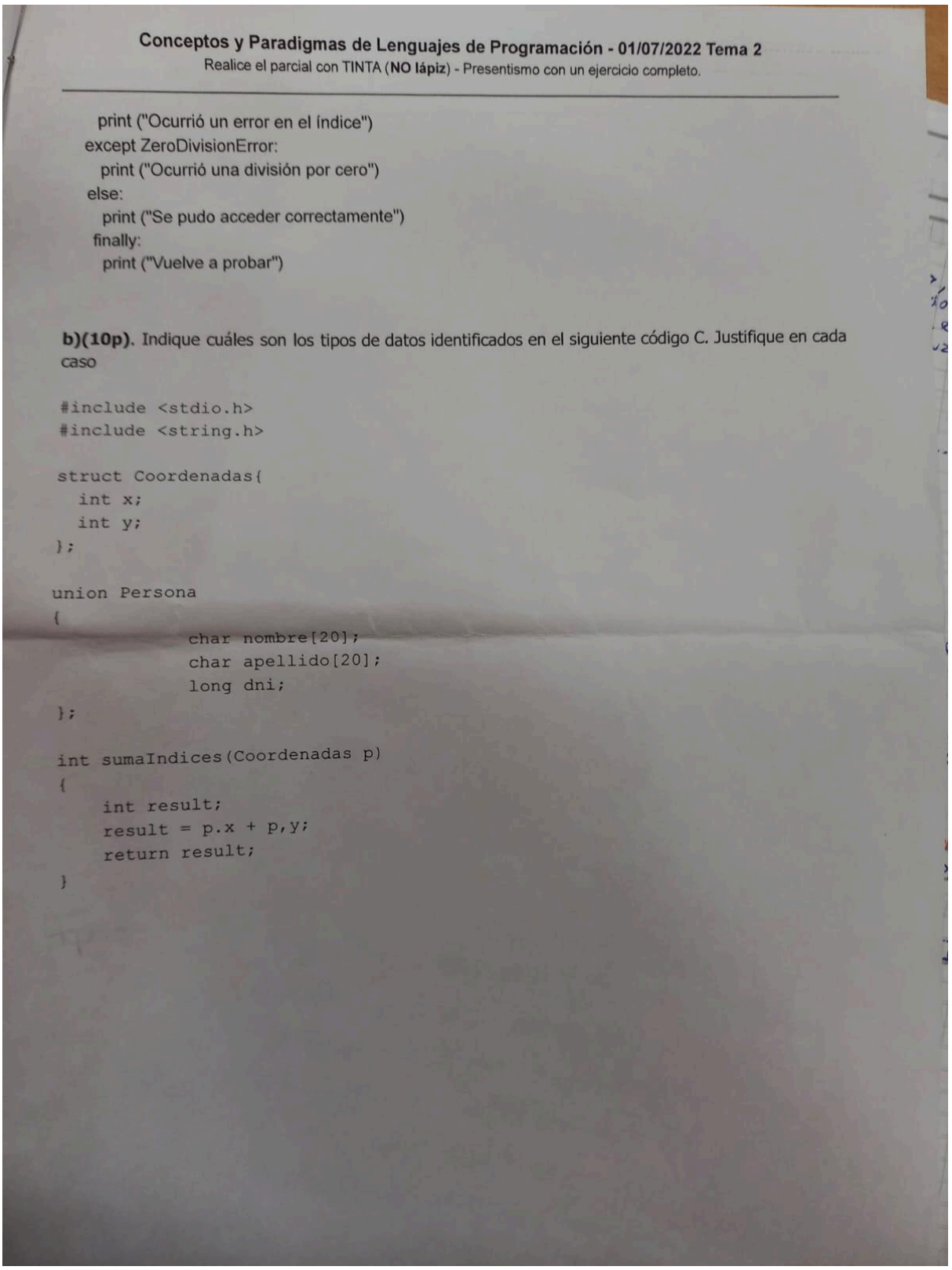
Nota: La forma de evaluación del lenguaje es de izquierda a derecha

Ejercicio 5 a)(10) Dado el siguiente código en Python. Describa los posibles caminos de ejecución.

```
#!/usr/bin/env python
#calc.py

def imprime_pos(y):
  print (("Resultado"), a[y]/a[y]-4)

#La función range devuelve los números desde 0 al límite enviado como parámetro menos 1
for y in range(6):
  try:
    a = [0,1,2,3,4]
    imprime_pos(y)
  except IndexError:
```



Conceptos y Paradigmas de Lenguajes de Programación - 01/07/2022 Tema 2

Realice el parcial con TINTA (**NO lápiz**) - Presentismo con un ejercicio completo.

```
    print ("Ocurrió un error en el índice")
  except ZeroDivisionError:
    print ("Ocurrió una división por cero")
  else:
    print ("Se pudo acceder correctamente")
  finally:
    print ("Vuelve a probar")
```

b)(10p). Indique cuáles son los tipos de datos identificados en el siguiente código C. Justifique en cada caso

```
#include <stdio.h>
#include <string.h>

struct Coordenadas{
  int x;
  int y;
};

union Persona
{
            char nombre[20];
            char apellido[20];
            long dni;
};

int sumaIndices(Coordenadas p)
{
    int result;
    result = p.x + p,y;
    return result;
}
```

Ejercicio 3

a) Falso. Lo que se menciona es sobrecarga, que si bien es una forma de polimorfismo ad hoc, no engloba todo el polimorfismo, ya que este término se usa para describir

cuando una expresión puede estar ligadas a varios tipos distintos. Los tipos de polimorfismo son:

- ad hoc: al aplicarse una función sobre distintos tipos se realizan distintas operaciones:
  - sobrecarga
  - coerción
- Universal: cuando una única operación se aplica sobre distintos tipos relacionados de la misma manera:
  - Tipos parametrizados (List<T>)
  - Por inclusión (herencia)

b) Falso, la unión discriminada es más segura que la unión, ya que la primera posee un discriminante que puede ser chequeado por el programador para saber cuál de las distintas posibles opciones dentro de la unión, está siendo actualmente usada, por lo que permite realizar operaciones con mayor seguridad (aunque en algunos lenguajes puede no especificarse por lo que no aportaría seguridad extra).

Ejercicio 4

CPLP - 2do parcial - Parciales viejos

Ejercicio 5

a)

El programa se ejecutará de la siguiente manera:

- Para el número 0 (de la variable y), al intentar dividir a[y]/a[y]-4, es decir, 0/0-4 (la división tiene prioridad sobre la suma), ocurre una excepción del tipo ZeroDivisionError, por lo que se ejecuta el manejador definido para dicho tipo de excepción, que imprime “Ocurrió una división por cero” y luego “Vuelve a probar” (cláusula finally)
- Para los números (de la variable y) del 1 al 4 no hay problema y se imprime:
  - “Resultado -3.0”
  - “Resultado -3.0”
  - “Resultado -3.0”

    - “Resultado -3.0”
- Y luego de cada una de las impresiones se imprime “Se pudo acceder correctamente” (cláusula else) y “Vuelve a probar” (cláusula finally)
- Para el número 5 (de la variable y), dado que la lista a no tiene una posición 5, ocurre una excepción de tipo IndexError, por lo que se ejecuta el manejador definido para dicho tipo de excepción, que imprime “Ocurrió un error en el índice” y luego “Vuelve a probar” (cláusula finally)

b)

```
struct Coordenadas {
        int x;
        int y;
};
```

Es un tipo de datos compuesto definido por el usuario, cuyo constructor es el producto cartesiano, ya que relaciona diferentes elementos de varios conjuntos (en este caso son 2 conjuntos, ambos enteros(?).

```
union Persona {
        char nombre[20];
        char apellido[20];
        long dni;
}
```

Es un tipo de datos compuesto definido por el usuario, cuyo constructor es el de unión. Esto significa que tan solo uno de los campos puede tener valor en un mismo momento (los campos son mutuamente excluyentes).

```
int sumaIndices(Coordenadas p){
        int result;
        result = p.x + p.y;
        return result;
}
```

Es una función, la cual es un tipo de datos compuesto definido por el usuario, cuyo constructor es la correspondencia finita, puesto que, para cada valor de un conjunto finito (para cada valor posible del struct Coordenadas), se devolverá un y sólo un valor que se corresponda con el parámetro ingresado.

1era fecha 2023

Ejercicio 1

CPLP - 2do parcial - Parciales viejos

Ejercicio 2

a)

i)

```
class Alumno {
	String nombre;
	String apellido;
	int edad;
	float promedio;
	String domicilio;

	public float getPromedio(){
		return this.promedio;
	}
}
```

Es un tipo compuesto definido por el usuario cuyo constructor es el producto cartesiano, ya que permite realizar la combinación de varios conjuntos (String, String, int, float y String).

ii)

```
typedef struct _nodoArbol {
	void *info;
	struct _nodoArbol *hijoIzq;
	struct_nodoArbol *hijoDer;
} nodoArbol;

typedef struct _arbolBinario {
	int valor_guardado;
```

```
    nodoArbol *raiz;
} arbolBinario;
```

Es un tipo compuesto definido por el usuario cuyo constructor es el producto cartesiano, ya que permite realizar la combinación de varios conjuntos. A su vez, también es recursivo puesto que el tipo _nodoArbol tiene dentro de sí, dos punteros a _nodoArbol, dando lugar a una estructura recursiva.

b)

i) Falso. Python.
ii) Falso. Es correspondencia finita donde cada valor de un conjunto finito se corresponde a un solo valor.
iii) Verdadero. Las uniones no permiten saber el tipo de dato que se está usando, lo que puede traer problemas, y las uniones discriminadas se pueden hackear (ahre) con el uso de punteros, referenciando a un valor que no es el que indica el discriminante (ver teoría 7).

Ejercicio 3

a)
i) Es válido:

- Comienza el for con i=1
- Entra al if(i==1), donde imprime 1 y llama a rethrow(“Primera”)
- Entra al if(s.equals(“Primera”)) donde lanza una FirstException
- Se catchea la FirstException en su manejador, donde se lanza una ThirdException
- Se catchea la ThirdException en el método main, donde se imprime su mensaje (“Tercera excepción”)

b)
No genera el mismo resultado ya que, luego de lanzarse la excepción ThirdException, ésta se catchearia en el método main pero, al estar el try/catch dentro del for, una vez finalice el catch (ThirdException e), se continuaría con el valor 2 para i, y al lanzarse más excepciones se seguiría incrementando el valor de i hasta llegar a 3.

Ejercicio 4

a. Falso, ADA es más seguro porque en Pascal se puede modificar el índice de un for siempre y cuando el índice sea global y esa modificación se haga dentro de una rutina llamada desde dentro del for.
b. Verdadero. Vamos a recurrir a la genealogía mamífera de OO1 y OO2 😎. Ejemplo:
   bool esAbuelaMaterna = madre != null && madre.getMadre().equals(unMamifero);
   De esta manera, si madre es null, esAbuelaMaterna es falso y no se evalúa madre.getMadre() que, de evaluarse al ser madre == null daría error (NullPointerException).
c. Falso. Yield permite generar una secuencia de valores en lugar de retornar un solo valor como lo hace return. A medida que se le solicita un valor, yield lo genera y retorna.
d. Falso. Se continúa la ejecución del programa en la siguiente instrucción a donde se generó la excepción.
e. Falso. La sentencia else de Python se ejecuta si no ocurrió ninguna excepción.