# INSTITUTO TECNOLÓGICO SUPERIOR DE CHICONTEPEC

## INGENIERÍA EN SISTEMAS COMPUTACIONALES

**Producto:** Funciones en Haskell.

**Asignatura:** Programación Lógica y Funcional.

**Docente:** Ing. Efrén Flores Cruz.

**Estudiante:** Manuel Zúñiga Hernández.

**Semestre:** Octavo

Chicontepec, Veracruz, a 04 de Abril 2020.

## Contenido

# Introducción

En el siguiente reporte de la asignatura de Programación Lógica, en el apartado de desarrollo se encuentran algunos de los ejercicios que se estuvieron elaborando durante las horas de clase, y otras que se encargaron como prácticas para elaborar la carpeta de evidencias de dicha asignatura.

Los siguientes ejercicios que se realizaron fueron creados y elaborados para ejecutarlos en el programa Haskell, este programa nos permite realizar diferentes ejercicios tal y como se muestra en el presente reporte.

## Desarrollo

1. Las siguientes funciones son las siguientes: succ, min y max, se pueden combinar entre ellas para saber diferentes valores.

```
Prelude> succ 9
10
Prelude>
```

```
Prelude> succ 'a'
'b'
Prelude>
```

```
Prelude> succ 'h'
'i'
Prelude>
```

2. Las siguientes funciones devuelven los valores que se comparan ya sean menores o mayores.

```
Prelude> min 3 55
3
Prelude>
```

```
Prelude> min 8 44
8
Prelude>
```

```
Prelude> max 10 55.3
55.3
Prelude>
```

```
Prelude> max 10 22
22
Prelude>
```

3. Las siguientes realizan comparaciones entre las diferentes funciones.

```
Prelude> max 4 (succ 10)
11
Prelude>
```

```
Prelude> succ (max 10 15)
16
Prelude>

Prelude> succ (max 10 (min 11 22.3))
12.0
Prelude>
```

4. Las siguientes funciones pertenecen a las listas. En las siguientes imágenes que se muestran se puede observar las diferentes listas que se pueden crear.

```
GHCi, version 8.6.5: http://www.haskell.org/ghc/  :? for help
Prelude> listas = [1,2,3,4,5]
Prelude> listas
[1,2,3,4,5]
Prelude>

Prelude> lista = ['a','b','c']
Prelude> lista
"abc"
Prelude>
```

5. En esta parte se concatenan las siguientes listas.

```
Prelude> lista = [6,7,8]++[2,5]
Prelude> lista
[6,7,8,2,5]
Prelude>

Prelude> lista = ['h','o']++['l','a']
Prelude> lista
"hola"
Prelude>

Prelude> lista = ['h','o','l','a']
Prelude> lista
"hola"
```

6. Otra forma de concatenar las siguientes funciones tendremos que escribir la palabra concatenar y después ingresar lo que queramos concatenar.

```
Prelude> concatenar = 45:[35,25,15]
Prelude> concatenar
[45,35,25,15]
Prelude>

Prelude> concatenar = 'H':"ola mundo"
Prelude> concatenar
"Hola mundo"
Prelude>

Prelude> lista = [[1,2], [3,4]]
Prelude> lista !!0 !!1
2
Prelude> lista !!0
[1,2]
Prelude>
```

7. Con **let** nos permite crear una lista, asociarlo y poder usar esa lista.

```
Prelude> let lista = [23,24,25]
Prelude> lista !!0
23
Prelude>
```

8. Podremos realizar las funciones de listas que muestren números pares o de un rango a cierto rango de la siguiente forma tal como se muestra en las siguientes imágenes.

```
Prelude> let lista = [impares|impares<- [1..20], impares `mod` 2==1]
Prelude> lista
[1,3,5,7,9,11,13,15,17,19]
Prelude>
```

```
Prelude> let lista = [pares*10|pares<- [1..20], pares `mod` 2==0]
Prelude> lista
[20,40,60,80,100,120,140,160,180,200]
Prelude>
```

```
Prelude> let lista = [simple+doble | simple<-[1..20], doble<-[1..100], simple<10, doble `mod` 10==0]
Prelude> lista
[11,21,31,41,51,61,71,81,91,101,12,22,32,42,52,62,72,82,92,102,13,23,33,43,53,63,73,83,93,103,14,24,34,44,54,64,74,84,94,104,15,25,35,45,55,65,75,85,95,105,16,26,36,46
,56,66,76,86,96,106,17,27,37,47,57,67,77,87,97,107,18,28,38,48,58,68,78,88,98,108,19,29,39,49,59,69,79,89,99,109]
Prelude>
```

9. Las siguientes imágenes pertenecen a funciones infinitas.

repeat [1,2,3]

cycle [‘h’,’o’]

```
Prelude> replicate 2 "ho"
["ho","ho"]
Prelude>
```

10. Las siguientes son duplas y esto permite la combinación de diferentes tipos de datos.

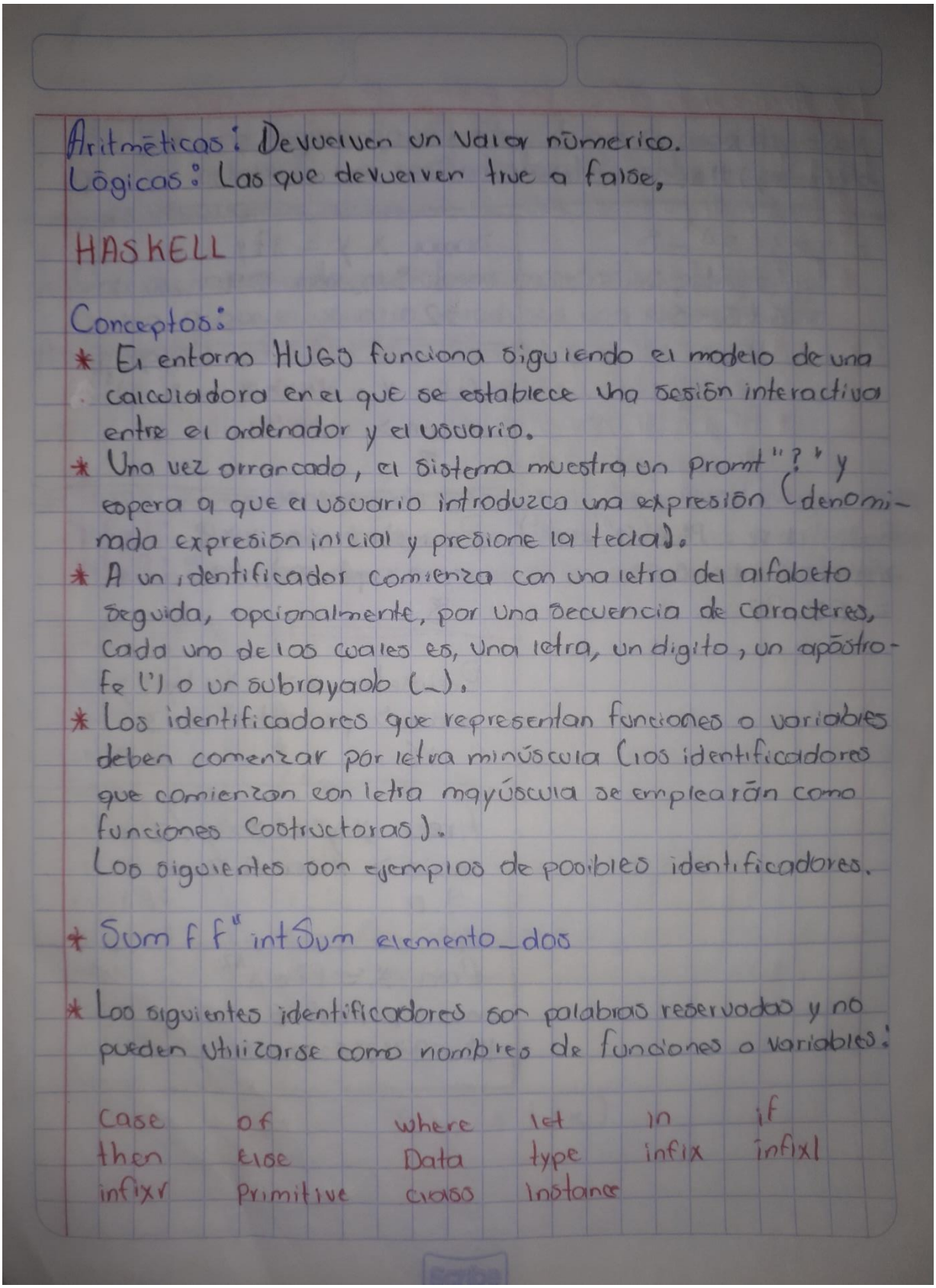
```
Prelude> let dupla = (1,"dos")
Prelude> dupla
(1,"dos")
Prelude>
```

```
Prelude> let tripla = (1, "Pedro", 7461134094)
Prelude> tripla
(1,"Pedro",7461134094)
Prelude>
```

```
Prelude> let lista = [(1, "dos"), (2, "uno")]
Prelude> lista
[(1,"dos"),(2,"uno")]
Prelude>
```

11. En la siguiente imagen se utilizó el comando: t se utiliza para ver el tipo de dato o función.

```
Prelude> :t head
head :: [a] -> a
Prelude> :t fst
fst :: (a, b) -> a
Prelude>
```



Aritméticas: Devuelven un valor numérico.
Lógicas: Las que devuelven true o false.

HASKELL

Conceptos:

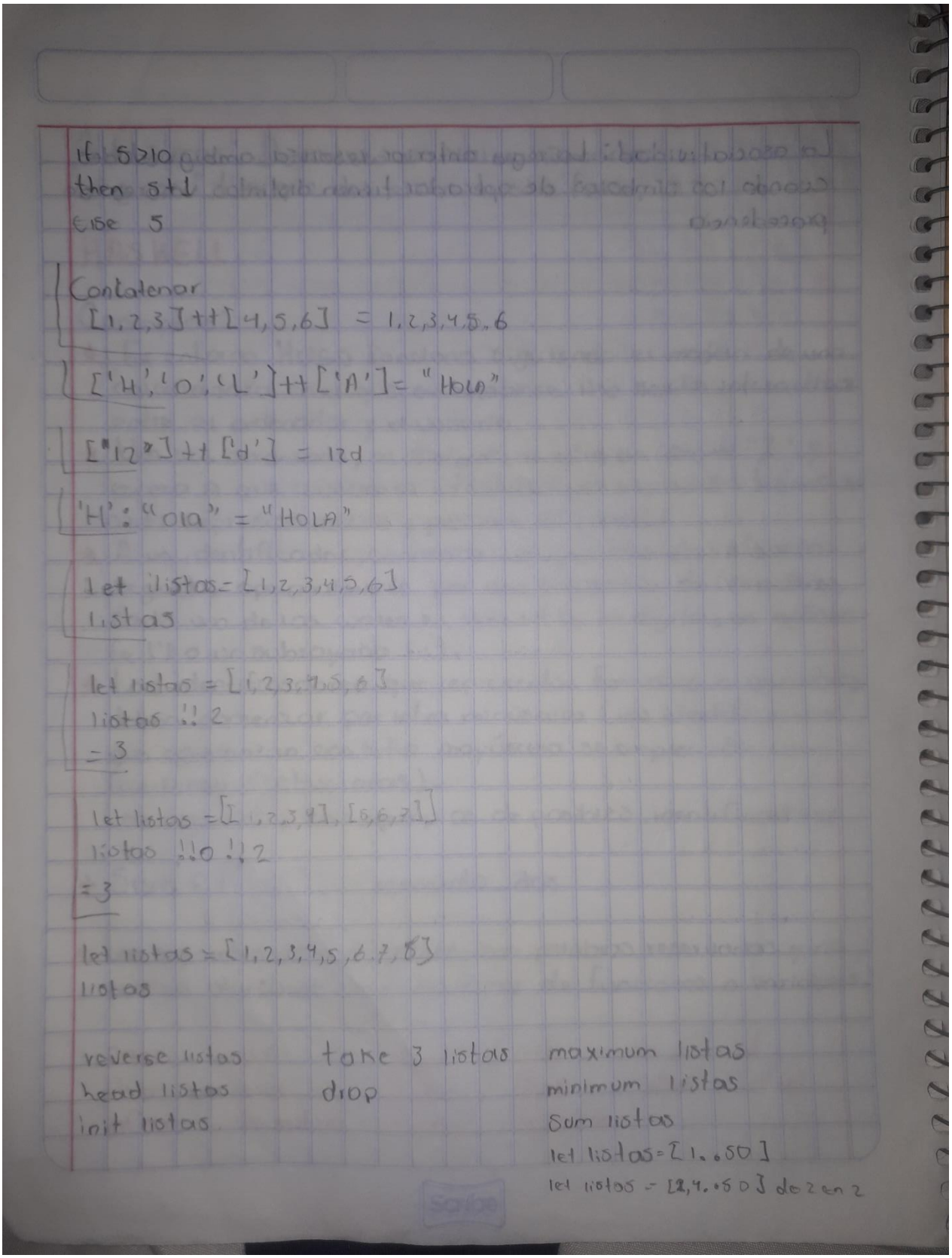
* El entorno HUGS funciona siguiendo el modelo de una calculadora en el que se establece una sesión interactiva entre el ordenador y el usuario.
* Una vez arrancado, el sistema muestra un promt "?" y espera a que el usuario introduzca una expresión (denominada expresión inicial y presione la tecla).
* A un identificador comienza con una letra del alfabeto seguida, opcionalmente, por una secuencia de caracteres, cada uno de los cuales es, una letra, un digito, un apóstrofe (') o un subrayado (_).
* Los identificadores que representan funciones o variables deben comenzar por letra minúscula (los identificadores que comienzan con letra mayúscula se emplearán como funciones Coostructoras).

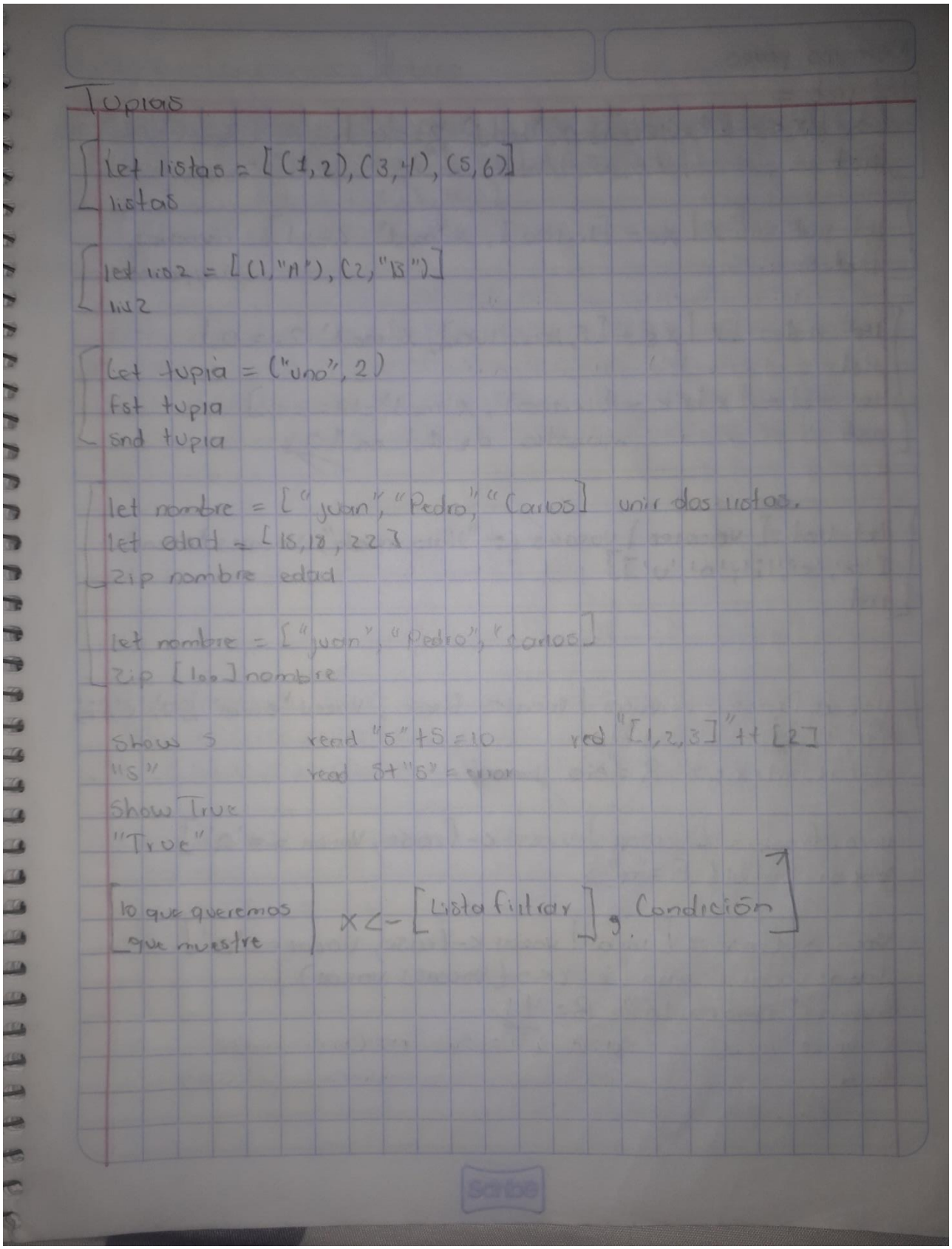
Los siguientes son ejemplos de posibles identificadores.

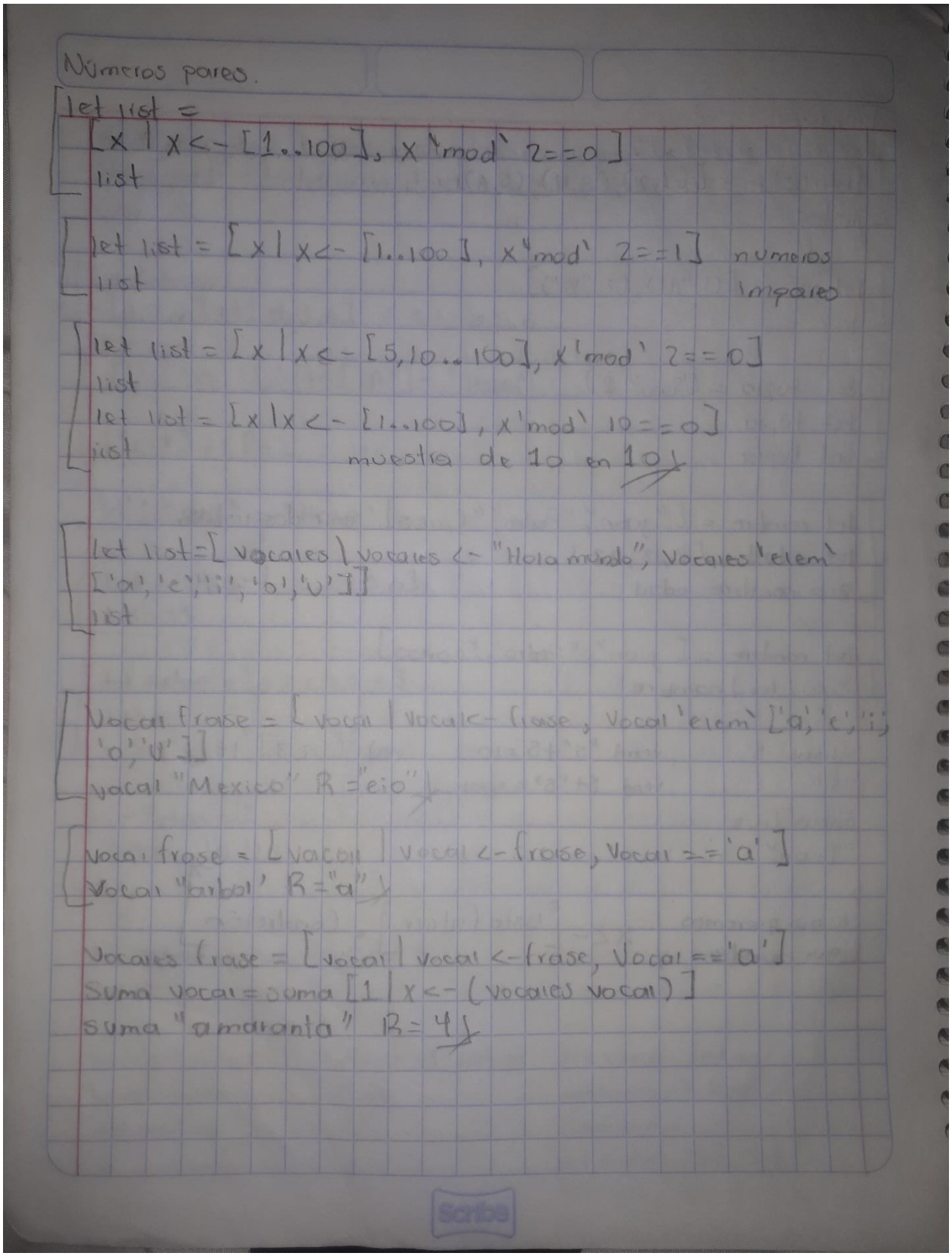
* Sum f f' intSum elemento_doo

* Los siguientes identificadores son palabras reservadas y no pueden utilizarse como nombres de funciones o variables:

| | | | | | |
|---|---|---|---|---|---|
| Case | of | where | let | in | if |
| then | else | Data | type | infix | infixl |
| infixr | Primitive | class | Instance | | |


if 5>10
then 5+1
else 5
Contatenar
[1,2,3]++[4,5,6] = 1,2,3,4,5,6
['H','O','L']++['A']= "HOLA"
["12"]++['d'] = 12d
'H':"ola" = "HOLA"
let listas=[1,2,3,4,5,6]
listas
let listas = [1,2,3,4,5,6]
listas !! 2
=3
let listas =[[1,2,3,4],[5,6,7]]
listas !!0 !!2
=3
let listas = [1,2,3,4,5,6,7,8]
listas
reverse listas
head listas
init listas
take 3 listas
drop
maximum listas
minimum listas
sum listas
let listas=[1..50]
let listas = [2,4..50] de 2 en 2


Tuplas
let listas = [(1,2),(3,4),(5,6)]
listas
let lis2 = [(1,"A"),(2,"B")]
lis2
let tupla = ("uno", 2)
fst tupla
snd tupla
let nombre = ["juan","Pedro","Carlos] unir dos listas.
let edad = [18,19,22]
zip nombre edad
let nombre = ["juan","Pedro","Carlos]
zip [1..] nombre
show 5
"5"
read "6"+5 =10
read "[1,2,3]"++[2]
read 5++"6" = error
Show True
"True"
lo que queremos que muestre
x <- [lista filtrar] , Condición


Números pares.
let list =
[x | x <- [1..100], x 'mod' 2 == 0]
list
let list = [x | x <- [1..100], x 'mod' 2 == 1] numeros impares
list
let list = [x | x <- [5,10..100], x 'mod' 2 == 0]
list
let list = [x | x <- [1..100], x 'mod' 10 == 0]
list
muestra de 10 en 10
let list=[ vocales | vocales <- "Hola mundo", vocales 'elem' ['a','e','i','o','u']]
list
Vocal frase = [vocal | vocal <- frase, vocal 'elem' ['a','e','i','o','u']]
vocal "Mexico" R = "eio"
Vocal frase = [vocal | vocal <- frase, vocal == 'a']
Vocal "arbol" R = "a"
Vocales frase = [vocal | vocal <- frase, vocal == 'a']
Suma vocal = suma [1 | x <- (vocales vocal)]
suma "amaranta" R = 4

# Conclusión

En el presente reporte que se realizó en la asignatura de Programación Lógica se pudieron observar los diferentes ejercicios y funciones que se pueden crear en el programa de Haskell, de las diferentes funciones pudimos observar las listas, duplas, números menores y máximos, comparaciones y combinación de listas.

Estas funciones se realizaron en las horas de clase, algunas se encargaron como tarea para seguir practicando y también para crear nuestra carpeta de evidencias de la unidad 2 de Programación Lógica.