Practica 7

Ejercicio 1: Sistemas de tipos:

1. ¿Qué es un sistema de tipos y cuál es su principal función?
2. Definir y contrastar las definiciones de un sistema de tipos fuerte y débil (probablemente
   en la bibliografía se encuentren dos definiciones posibles. Volcar ambas en la
   respuesta). Ejemplificar con al menos 2 lenguajes para cada uno de ellos y justificar.
3. Además de la clasificación anterior, también es posible caracterizar el tipado como
   estático o dinámico. ¿Qué significa esto? Ejemplificar con al menos 2 lenguajes para
   cada uno de ellos y justificar.

1. Un sistema de tipos son un conjunto de reglas usadas por un lenguaje para estructurar y organizar sus tipos. Su principal función es garantizar seguridad y corrección al programa, evitando errores al detectar usos incorrectos de los datos.
2. Tipado fuerte → Un lenguaje es fuertemente tipado si **no permite operaciones entre tipos incompatibles sin conversión explícita**. Ejemplos: python y java.

   Tipado débil → Un lenguaje es débilmente tipado si **permite operaciones entre distintos tipos mediante conversiones implícitas**, muchas veces sin advertencias. Ejemplos: javascript y C.
3. Un lenguaje tiene **tipado estático** cuando **los tipos de las variables se determinan y verifican en tiempo de compilación**.
   - Los errores de tipo se detectan **antes de ejecutar** el programa.
   - El tipo de una variable **no puede cambiar** una vez declarado.

Un lenguaje tiene **tipado dinámico** cuando **los tipos se determinan y verifican en tiempo de ejecución**.

- Una variable puede cambiar de tipo durante la ejecución.
- Más flexible, pero los errores de tipo **pueden aparecer al correr el programa**.

Ejercicio 2: Tipos de datos:

1. Dar una definición de tipo de dato.
2. ¿Qué es un tipo predefinido elemental? Dar ejemplos.
3. ¿Qué es un tipo definido por el usuario? Dar ejemplos

1. Es una forma de clasificar los datos para que el programa sepa tratarlos correctamente determinando un conjunto de valores y un conjunto de operaciones que se pueden utilizar para manipular esos datos.
2. Un tipo predefinido es aquel que refleja el comportamiento del hardware subyacente. Ejemplos: integer, string, boolean.
3. Un tipo definido por el usuario permiten especificar agrupaciones de objetos de datos elementales que son definidos a partir de constructores. Ejemplos: Arrays, registros, listas.

Ejercicio 3: Tipos compuestos:

1. Dar una breve definición de: producto cartesiano (en la bibliografía puede aparecer
   también como product type), correspondencia finita, uniones (en la bibliografía puede
   aparecer también como sum type) y tipos recursivos.
2. Identificar a qué clase de tipo de datos pertenecen los siguientes extractos de código.
   En algunos casos puede corresponder más de una:

Java

```
class Persona {
  String nombre;
  String apellido;
  int edad;
}
```

Producto cartesiano

C

```
typedef struct _nodoLista {
  void *dato;
  struct _nodoLista *siguiente
} nodoLista;

typedef struct _lista {
  int cantidad;
  nodoLista *primero
} Lista;
```

Recursión

C

```
union codigo {
  int numero;
  char id;
};
```

Union y union descriminada

Ruby

```
hash = {
  uno: 1,
  dos: 2,
  tres: 3,
  cuatro: 4
}
```

Producto cartesiano

PHP

```
function doble($x) {
  return 2 * $x;
}
```

Correspondencia finita

Python

```
tuple = ('physics',
'chemistry', 1997, 2000)
```

Producto cartesiano

Haskell

```
data ArbolBinarioInt =
  Nil |
  Nodo int
    (ArbolBinarioInt dato)
    (ArbolBinarioInt dato)
```

Ayuda para interpretar:
'ArbolBinarioInt' es un tipo de dato que puede ser Nil ("vacío") o un Nodo con un dato número entero (int) junto a un árbol como hijo izquierdo y otro árbol como hijo derecho

Recursión

Haskell

```
data Color =
  Rojo |
  Verde |
  Azul
```

Ayuda para interpretar:
'Color' es un tipo de dato que puede ser Rojo, Verde o Azul.

Correspondencia finita

1. Producto cartesiano → es un producto de N conjuntos cuyos elementos están ordenados por tuplas. Ejemplo: registro.

   Correspondencia finita → es una función de un conjunto finito de valores de un tipo del dominio DT (tipo de dominio) en valores de un tipo de dominio RT (tipo de resultado). Ejemplo: Array.

   Unión y unión discriminada → define un tipo como la disyunción de los tipos dados. Permite manipular distintos tipos en distintos momentos de la ejecución y hacer un chequeo dinámico del valor y de su tipo. Además, para la unión discriminada, se agrega un discriminante para indicar la opción elegida.

   Recursión → se define como una estructura que puede contener componentes de su mismo tipo y se implementan a través de punteros.

Ejercicio 4: Mutabilidad/Inmutabilidad:

1. Definir mutabilidad e inmutabilidad respecto a un dato. Dar ejemplos en al menos 2
   lenguajes. TIP: indagar sobre los tipos de datos que ofrece Python y sobre

la operación
#freeze en los objetos de Ruby.

2. Dado el siguiente código:
   a = Dato.new(1)
   a = Dato.new(2)
   ¿Se puede afirmar entonces que el objeto “Dato.new(1)” es mutable? Justificar la
   respuesta sea por afirmativa o por la negativa.

1. Mutabilidad → un dato puede cambiar su contenido después de haber sido creado.

Inmutabilidad → un datoinmutable no puede modificarse una vez creado.

Concepto	Mutable	Inmutable
Python	list, dict, set	int, str, tuple
Ruby	String, Array (por defecto)	Cualquier objeto con `.freeze` aplicado

2. Con el código proporcionado no se puede afirmar la sentencia. Al realizar a = Dato.new(2), cambia el valor la variable a y no la direccion donde estaba apuntando.

Ejercicio 5: Manejo de punteros:

1. ¿Permite C tomar el l-valor de las variables? Ejemplificar.
2. ¿Qué problemas existen en el manejo de punteros? Ejemplificar.

1. Si, permite tomar el l-valor a través de los punteros.Por ejemplo:

```
int x = 5;
int *p = &x;  // aquí se toma el l-valor de x con el operador &

*p = 10;      // se modifica el valor de x a través del puntero
```

2. Los punteros pueden contener inseguridades en su uso:

- Referencias sueltas → es un puntero que contiene una dirección de una variables que fue desalocada.
- Punteros no inicializados → peligro en el acceso descontrolado a posiciones de memoria.
- Alias.
- Punteros y unión discriminada.
- Liberación de memoria.
- Validación de tipos.

Ejercicio 6: TAD :

1. ¿Qué características debe cumplir una unidad para que sea un TAD?
2. Dar algunos ejemplos de TAD en lenguajes tales como ADA, Java, Python, entre otros.

1. Un TAD es una abstracción que define un tipo de dato junto con las operaciones que se pueden realizar con el. Para ser considerado un TAD, una unidad debe cumplir con las siguientes características:
   - Encapsulamiento → los detalles de la implementación deben estar ocultos y solo se expone una interfaz con operaciones permitidas.
   - Ocultamiento de información → la representación de los objetos y la implementación del tipo están ocultas.
2. Java → pilas.

   Python → colas.