TP2: junqtional programming

Detalles del trabajo

Objetivos

- Analizar el problema planteado correctamente.
- Diseñar una solución utilizando correctamente los conceptos de **programación funcional**.
- Seguir las buenas prácticas de programación.

Indicaciones del trabajo

Grupos de trabajo

El trabajo se desarrollará de forma grupal, en grupos de máximo 4 alumnos. El trabajo práctico debe realizarse utilizando Git y Github como herramienta de colaboración.
Todos los alumnos integrantes del grupo **deben** participar activamente del desarrollo del trabajo de forma equitativa. Mientras la participación sea equitativa, queda a criterio de cada grupo la forma de trabajo y organización.

Tecnología

El trabajo debe realizarse en *Scala 3*, utilizando cualquier versión estable de esta versión.

Tiempo de trabajo y entregas parciales

Se contarán con 3 semanas completas para llevar a cabo el desarrollo del TP. El trabajo no cuenta con entregas parciales: transcurrido el tiempo de trabajo, debe realizarse la **entrega**.

Modalidad de entrega

Para realizar la entrega se debe:

- Github:
  - El repositorio tiene que ser privado y debe estar su corrector invitado al mismo.
  - Para realizar la entrega se debe crear un branch llamado `tp-2`, con el código correspondiente a la entrega. No se debe modificar más luego de la fecha de entrega.
  - Se debe contar con un `README.md` que explique cómo correr el programa “desde cero”, cómo instalar las dependencias y otras explicaciones pertinentes. El código debe poder correrse con un comando que compile el código y lo ejecute, **no es válido subir un ejecutable y que el comando simplemente lo corra**.

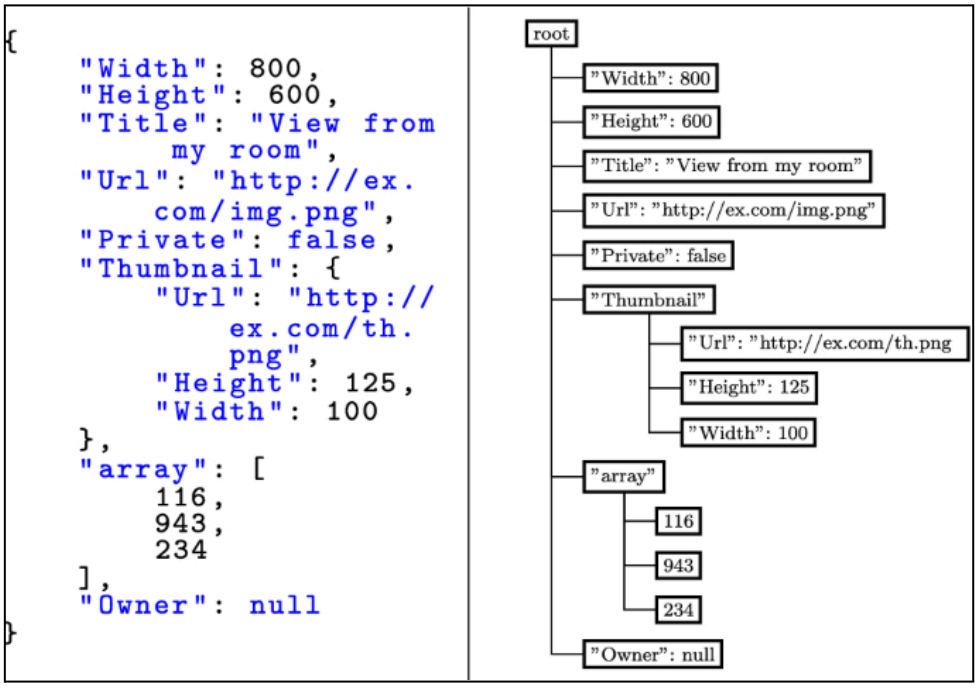
- Mail: Mandar un mail a algoritmos3.fiuba@gmail.com indicando número de grupo y sus integrantes, aclarando cuál es la branch y el repositorio de la entrega. El mail debe llevar adjunto el informe en formato PDF.

Fecha de entrega

La entrega definitiva, que debe alinearse a lo especificado en la Modalidad de entrega, debe realizarse con anterioridad a las 23:59hs del 12/11/2024.

Enunciado: Manipulación de JSON con una tool funcional

Dominio



Ejemplo de JSON y su representación como árbol semántico

El formato de texto **JSON** (JavaScript Object Notation) es un estándar de intercambio de información que ha tomado muchísima popularidad, no solo en la transmisión de datos a través de Internet, sino también como formato de configuración, bases de datos, escritura de logs y más.

El formato JSON se compone, simplificadamente para los fines de este trabajo práctico, de una semántica del estilo:

- `<json-expr> = <literal> | <object> | <array>`
  - `<literal> = string | boolean | number`
  - `<object> = { key1: <json-expr>, key2: <json-expr>, … }`
  - `<array> = [ <json-expr>, <json-expr>, … ]`

Se puede ver con mayor detalle en esta especificación gráfica (o esta otra versión).

Una herramienta muy adoptada en sistemas Unix para el procesamiento de inputs JSON es **jq**, un toolkit muy poderoso para poder leer, filtrar, escribir y más operaciones sobre texto que siguen el formato JSON, tomado a partir de archivos o de entrada estándar. Incluso cuenta con un playground para probar sus comandos.

El objetivo del trabajo es realizar una herramienta que contemple solamente un subset de las mismas o similares funcionalidades de esta herramienta, realizado en Scala y basado en los conceptos de la programación funcional. Para realizar este trabajo, se deberá familiarizar con el formato JSON y con la funcionalidad de *jq*.

Requerimientos

Importante: En caso de tomar una decisión respecto a una definición ambigua del dominio o los requerimientos, volcarlo en la correspondiente sección de hipótesis del informe.

Entrada y argumentos

El programa debe recibir como argumentos al instanciarlo la operación que se desea hacer y los parámetros que correspondan, para manipular el JSON de entrada. El input JSON deberá poder ser recibido por entrada estándar, al instanciar el comando:

```
$ junqtional [opciones]
{ "clave": "valor"}
> <output>
```

O bien redirigiendo un archivo a stdin:

```
$ junqtional [opciones] < archivo.json
> <output>
```

El comando debe procesar la entrada, y determinar si es un JSON válido o no. En caso de que no, debe retornar error.
Si es válido, se aplicará sobre el input recibido, la operación determinada en los argumentos.

(**> <output>** es la salida estándar del programa, no es una redirección de flujo)

Se exige que el output se muestre únicamente por salida estándar. Si se quisiera, se puede redirigir la salida del programa hacia un archivo, con el operador clásico de la shell:

```
$ junqtional [opciones] < input.json > output.json
```

Cada llamada a `junqtional` debe ser una operación atómica sobre el JSON recibido, pero que si se quisiese, se pudiesen realizar operaciones más complejas encadenando a través de *piping* de llamadas al comando:

```
$ cat archivo.json | junqtional [opts] | junqtional [opts] | junqtional [opts]
> <output luego de 3 operaciones>
```

Más sobre redirecciones y pipes

Existe una serie de operadores dentro de la shell de Linux que nos permiten realizar redirecciones de flujo (stdin/stdout) desde o hacia archivos, o redirigir la salida de un programa hacia la entrada de otro:

- El operador < envía la información del fichero (archivo) especificado como stdin
- El operador > redirige el flujo de salida estándar hacia el fichero especificado. Si no existe, lo crea, sino sobreescribe el contenido del fichero.

```
$ echo "Hello World!" > text
$ cat text
Hello World!
```

- El operador | se utiliza para crear un *pipe* entre dos programas. Este canaliza la salida del programa de la izquierda hacia la entrada del programa de la derecha. A fines del desarrollo de este TP, quiere decir que a la salida de la operación aplicada sobre la archivo .json es posible aplicarle otra función en cadena como parte de una única solicitud
- Además, este operador debe permitir canalizar la salida de un programa hacia una invocación de *junqtional.* Esto quiere decir que la salida del programa es tomada por *junqtional* como su entrada estándar (debe funcionar siempre y cuando se provea un .json a través del pipe)
- *cat* es un comando que muestra por salida estándar el contenido de un fichero. Es por esto que se va a realizar lo anterior mencionado a través de este comando. (NO ES NECESARIO IMPLEMENTAR CAT, ES UN COMANDO DE LINUX)

Acceso (paths)

Una de las primeras cosas que querríamos en nuestro procesador JSON, es el de poder leer una estructura JSON, en particular pudiendo acceder a ciertas partes de esa estructura. Esto podría dar por una especie de indexación (por ruta de JSON) o bien por ciertas condiciones.
Tal como indica la documentación de *jq*, el acceso de las entidades JSON se da mediante una sintaxis de filtros, que combinados resultan en lo que denominaremos `path,` donde encontramos los operandos y sintaxis:

- `.` = el operando base, que básicamente toma el input inicial y retorna el mismo input. Sería como referirnos a la raíz de la estructura JSON recibida.
    - Es posible ante cualquier valor de JSON.
    - También servirá para encadenar indexaciones siguientes.
- `.key` o `.["key"]` = esté operando refiere a la indexación por clave ante un objeto JSON. Retorna el JSON que se encuentra como valor para dicha clave en el objeto.

- `.[<index>]` = es el operando de indexación de arrays, donde básicamente se retorna el valor ubicado en la posición "*index*" del arreglo. Se indexa comenzando desde 0.
- `|` = el operando *pipe* otorga una forma de encadenamiento de accesos previamente mencionados. Por ejemplo, si se indicase el path "`.a | .b`" esto indicaría en primer lugar, acceder al valor de la clave "a" del JSON inicial, y sobre el resultado de ese acceso (el valor de "a"), se busca obtener el valor asociado a la clave "b".

Tal como con el operador *pipe*, también se da una forma de azúcar sintáctica para la conformación de paths un poco más legible, utilizando únicamente el encadenamiento vía puntos. Aquí dejamos un tabla de ejemplos con paths, indicando el output a interpretar:

Input JSON	Path(s)	Output esperado
`{ "foo": "v1", "bar": "v2" }`	.	`{ "foo": "v1", "bar": "v2" }`
	.foo	`"v1"`
	.bar	`"v2"`
`{ "a": [1,2,3], "b": { "c": "x" } }`	.a	`[1,2,3]`
	.a[0]	`1`
	.b	`{ "c": "x" }`
	➢ .b.c ➢ .b | .c	`"x"`
`[{ "a": "1" }, {"a": "2"}]`	.[1]	`{ "a": "2" }`
	➢ .[0].a ➢ .[0] | .a	`"1"`
	➢ .[1].a ➢ .[1] | .a	`"2"`

Operaciones

La idea del trabajo es sumar la mayor cantidad de operaciones posibles. Las identificadas como más útiles para una primera versión de la herramienta son las siguientes, clasificadas por categoría.
Cada operación tiene una dificultad estimada por los docentes, que está marcada con una determinada cantidad de estrellas a su lado, que van del 1 al 3 siendo 3 la mayor dificultad. También han sido marcadas con un signo de exclamación rojo algunas operaciones consideradas ***indispensables***, y con un signo de "picante" los que son un desafío y por tanto son opcionales.

Adición y borrado

- ❗(★) `junqtional add-key <path> <key> <value> ⇒ <json-expr>`
  - **Descripción**: Operación que añade un nuevo par clave-valor a un objeto JSON.
  - **Parámetros**:
    - path (string): indica el path a un objeto dentro del input JSON
    - key (string): la nueva clave a agregar al objeto
    - value (json-expr): el valor a setear para la nueva clave
  - **Output** (json-expr): el input JSON original, con la nueva clave-valor agregado sobre el path indicado.
- ❗(★) `junqtional add-item <path> <item> ⇒ <json-expr>`
  - **Descripción**: Operación que añade un nuevo elemento a un array JSON.
  - **Parámetros**:
    - path (string): indica el path a un array dentro del input JSON
    - item (json-expr): el item a setear al final del array
  - **Output** (json-expr): el input JSON original, con el nuevo elemento agregado sobre el path indicado.
- (★☐) `junqtional delete <path> ⇒ <json-expr>`
  - **Descripción**: Operación que borra una parte del input JSON.
  - **Parámetros**:
    - path (string): indica el path a una clave o item de array dentro del input JSON
  - **Output** (json-expr): el input JSON original, sin el elemento borrado.
- (★★☐) `junqtional merge <other> ⇒ <json-expr>`
  - **Descripción**: Operación que mergea un nuevo elemento a un array JSON.
  - **Parámetros**:
    - other (object | array): un objeto o array a mergear con el root del input JSON
  - **Output** (json-expr): el input JSON original, mergeado con los elementos del parámetro 'other'.
    - Ejemplo: si el input es un array, se *mergearan* con los elementos del array 'other'. Si el input es un objeto, se *mergearan* con las claves novedosas (es decir las que no estén presentes en el input) que esten en 'other'.

Transformación

- ❗(★) `junqtional edit <path> <value> ⇒ <json-expr>`
  - **Descripción**: Operación que edita el valor de un elemento a un array JSON.
  - **Parámetros**:
    - path (string): indica el path a un elemento dentro del input JSON
    - value (json-expr): el nueva valor que tendrá asociado ese path
  - **Output** (json-expr): el input JSON original, con el elemento en cuestión editado.
- (★★) `junqtional flatten ⇒ <array>`
  - **Descripción**: Operación que hace flatten de un input JSON de tipo array de arrays (2-dimensional array), resultando en un arreglo que concatena todos los elementos anidados.
  - **Parámetros**: -
  - **Output** (array): un nuevo array con los elementos concatenados de los arrays anidados en el input JSON original.
- (★★★) `junqtional map <transformer> ⇒ <json-expr>`
  - **Descripción**: Operación que aplica una función a cada elemento del root del input JSON (array u objeto).
  - **Parámetros**:

        - transformer (función): una función que toma un parámetro y retorna un valor..
    - **Output** (json-expr): el input JSON original, con los elementos transformados segun lo que indicaba la función transformadora.
- 🌶️(★★★★) `junqtional map_rec <transformer> ⇒ <json-expr>`
    - **Descripción**: Operación que actúa como un map recursivo, aplicando una función a todos los elementos del input JSON, en todos los niveles.
    - **Parámetros**:
        - transformer (función): una función que toma un parámetro y retorna un valor..
    - **Output** (json-expr): un JSON similar al original, pero con todos los elementos -en todos los niveles- transformados según lo que indicaba la función transformadora.

Filtrado

- ❗(★) `junqtional get <path> ⇒ <json-expr>`
    - **Descripción**: Operación que obtiene un determinado valor del input JSON.
    - **Parámetros**:
        - path (string): indica el path a un elemento dentro del input JSON
    - **Output** (json-expr): el valor que se halla en dicho path.
- (★☐) `junqtional depth <level> ⇒ <array>`
    - **Descripción**: Operación que obtiene un todos los elementos que están a una cierta profundidad del root del input JSON (los que se encuentran en el nivel N del árbol semántico).
    - **Parámetros**:
        - level (number): indica el nivel de profundidad al que se quiere acceder
    - **Output** (array): el array cuyos ítems son todos los elementos que se hallan al nivel de profundidad especificado, en el input JSON original.
- 🌶️(★★★) `junqtional select <condición> ⇒ <json-expr>`
    - **Descripción**: Operación que actúa como un “filter”, quedándose con los elementos del root del input JSON (array u objeto) que cumplan con la condición de la función pasada.
    - **Parámetros**:
        - condición (función): una función que recibe un elemento y retorna un booleano.
    - **Output** (json-expr): un JSON con los elementos para los cuales la función pasada retorna “true”.

Otros

- ❗(★) `junqtional exists_key <key> ⇒ <boolean>`
    - **Descripción**: Operación que recibe un input JSON de tipo objeto, y retorna “true” si en a nivel root, existe una clave con el nombre especificado.
    - **Parámetros**:
        - key (string): el nombre de la clave buscada.
    - **Output** (boolean): “true” si existe la clave dentro del input; de lo contrario, “false”.
- (★★) `junqtional exists_key_rec <key> ⇒ <boolean>`
    - **Descripción**: Operación que recibe un input JSON, y retorna “true” si en algún nivel existe una clave con el nombre especificado.
    - **Parámetros**:
        - key (string): el nombre de la clave buscada.
    - **Output** (boolean): “true” si existe la clave dentro del input; de lo contrario, “false”.

- (★★★) `junqtional all <condición> ⇒ <boolean>`
  - **Descripción**: Operación que recibe un input JSON de tipo array, y retorna “true” si todos los elementos cumplen la condición especificada.
  - **Parámetros**:
    - condición (función): una función que toma un parámetro y retorna un booleano.
  - **Output** (boolean): “true” si todos los elementos del input JSON cumplían la condición; de lo contrario, “false”.

Las operaciones que reciben funciones como parámetros, no tienen una sintaxis particular la cual seguir, pueden definir la que mejor les resulte para lograr el objetivo. La misma debe estar bien especificada en el informe.

Informe

Se espera la realización de un informe que contenga:

- Decisiones tomadas respecto al procesamiento, guardado y manipulación de las entidades JSON.
- Detección de los casos y fragmentos del código donde se vieron en la necesidad de salirse de los conceptos de la programación funcional, justificando el por qué debieron hacerlo.
- Una sección de las hipótesis tomadas durante la realización del trabajo.
- Conclusiones.

Criterios de aprobación

Para que un TP se considere aprobado deberá al menos cumplir con los siguientes criterios mínimos:

- Se debe poder interpretar cualquier path de acceso, con al menos una de las notaciones (no es necesario interpretar las diferentes notaciones).
- Deben estar implementadas todas las operaciones indispensables, denotadas con el símbolo “ ❗ ”.
- Se deben implementar suficientes operaciones como para alcanzar un mínimo de 10 estrellas (se contemplan también las otorgadas por las operaciones indispensables).
- No se deberá contemplar el manejo de los casos de error, ya sea por invalidez del input JSON o por imposibilidad de realizar una operación para dicho input y parámetros.

Para la promoción, se deberá alcanzar un mínimo de 13 estrellas acumuladas entre las operaciones implementadas y se deberán validar errores causados por paths no válidos para el input JSON recibido.

Si bien estas son las funcionalidades mínimas requeridas para la aprobación, un TP se considerará completo solo si cumple con todos los requerimientos descritos en las secciones anteriores. Para alcanzar la nota máxima del trabajo, se espera que esté implementada toda la funcionalidad descrita en el enunciado, con su correcto manejo de errores.