Miquel de Domingo
Sesión 2

Ejercicio 2

El enlace al repositorio es este.

Desarrollo del ejercicio

El ejercicio se ha desarrollado siguiendo la metodología expuesta en el enunciado, creando issues para los diferentes features/fix a añadir. Se ha intentado mantener al máximo la atomicidad de las issues, haciéndolas pequeñas pero concisas.

Issue #1

Ensure git ignores MacOs specific files

La issue arregla el problema con los ficheros .DS_Store generados por MacOs. Al añadirlos al fichero .gitignore, evitamos subirlos por error al repositorio (ya que no deberían estar).

Issue #2

Improve CI/CD and other simple issues with CI

El objetivo principal de esta issue era arreglar los problemas con el código que hacían que el pipeline no pasaran. Los cambios hechos, y la razón de ellos, son:

- Fijar las versiones de npm y node. Considero que son buenas prácticas cuando se trabaja en equipo, de esta forma nos aseguramos que los desarrolladores utilizan las mismas versiones. También, evitamos problemas repentinos que pueden llegar a ser causados por cambios de versión no migrados correctamente.
- Añadir prettier. Prettier permite formatear el código de forma que todos los desarrolladores tendrán el mismo formato, junto con ESLint.
- Código fuente dentro de la carpeta source. Los proyectos de JS suelen seguir está estructura base.
- Añadir hooks pre-commit. Husky es una librería que permite ejecutar comandos antes del commit, de esta forma se puede validar el código antes de hacer el commit. Con esto, añadí también lint-staged, que es una librería que permite ejecutar los pre-commit solo a los ficheros modificando. En un proyecto tan pequeño quizás no tenga mucho sentido, pero en proyectos grandes sí.

Los otros cambios corresponden a fixs pequeños a los ficheros de CI y otros.

Issue #3

Subtract operation

El objetivo de esta issue es añadir la operación de resta (-) a la calculadora. Le corresponde la merge request !4. En la misma, se ha añadido la operación como los tests que aseguran su funcionamiento.
Los cambios añadidos son los siguientes:

```
+   subtract(...values) {
+     return values.reduce((prev, curr) => prev - curr, 0);
+   },
```

Los tests añadidos para la operación:

```
+   describe('subtraction', () => {
+     it.each([
+       { n: [4, 5], r: -9 },
+       { n: [4, 5, 4, 5], r: -18 },
+       { n: [0, 1, 2, 3, 4], r: -10 },
+       { n: [0, 0, 0, 0, 0], r: 0 },
+       { n: [0, 1, -2, 3, -4], r: 2 },
+     ])('should add and result $r', ({ n, r }) => {
+       expect(C.subtract(...n)).toBe(r);
+     });
+   });
```

Como pequeño detalle, esta issue se ha arreglado después de la #6. También, la implementación contiene un error que se arregla en la #8.

Issue #4

Divide operation

El objetivo de esta issue es añadir la operación de división (/) a la calculadora. Le corresponde la merge request !5. En la misma, se ha añadido la operación como los tests que aseguran su funcionamiento.
Los cambios añadidos son los siguientes:

```
+   divide(...values) {
+     return values.reduce((prev, curr, i) => (i === 0 ? curr : prev / curr));
+   },
```

Los tests añadidos para la operación:

```
+   describe('divide', () => {
+     it.each([
+       { n: [4, 2, 2], r: 1 },
+       { n: [1, 1], r: 1 },
+       { n: [10], r: 10 },
+       { n: [10, 5, 0, 2], r: Infinity },
+       { n: [0, 5], r: 0 },
+     ])('should divide and result $r', ({ n, r }) => {
+       expect(C.divide(...n)).toBe(r);
+     });
+   });
```

Como pequeño detalle, esta issue se ha arreglado después de la #6.

Issue #5

Multiply operation

El objetivo de esta issue es añadir la operación de multiplicación (*) a la calculadora. Le corresponde la merge request !6. En la misma, se ha añadido la operación como los tests que aseguran su funcionamiento.
Los cambios añadidos son los siguientes:

```
+   multiply(...values) {
+     return values.reduce((prev, curr, i) => (i === 0 ? curr : prev * curr));
+   },
```

Los tests añadidos para la operación:

```
+   describe('multiply', () => {
+     it.each([
+       { n: [1, 1, 1, 1], r: 1 },
+       { n: [1, 2, 3, 4, 5], r: 120 },
+       { n: [1, 2, 0, 3, 4], r: 0 },
+       { n: [1], r: 1 },
+     ])('should multiply and result $r', ({ n, r }) => {
+       expect(C.multiply(...n)).toBe(r);
+     });
+   });
+
```

Como pequeño detalle, esta issue se ha arreglado después de la #6.

Issue #6

Static calculator

La intención de esta issue es convertir la calculadora en un objeto estático para poder utilizar los métodos de la clase sin depender de ningún estado. La implementación inicial solo permite el cálculo entre dos números (el estado de la calculadora). En cambio, con la implementación estática podemos ejecutar

operaciones con tantos números queramos, siguiendo un poco la API de Math de JS. Al no tener ningún otro método implementado, a parte de la suma, el refactor es bastante directo (así como los cambios en los tests).

```
+ const Calculator = {
+   add(...values) {
+     return values.reduce((prev, curr) => prev + curr, 0);
+   },
+ };
```

Issue #7

Mod operation

El objetivo de esta issue es añadir la operación de módulo (%) a la calculadora. Le corresponde la merge request !7. En la misma, se ha añadido la operación como los tests que aseguran su funcionamiento.
Los cambios añadidos son los siguientes:

```
+   mod(...values) {
+     return values.reduce((prev, curr) => prev % curr);
+   },
```

Los tests añadidos para la operación:

```
+
+   describe('mod', () => {
+     it.each([
+       { n: [10, 0], r: NaN },
+       { n: [10, 5], r: 0 },
+       { n: [98765, 1000, 100, 10], r: 5 },
+       { n: [98765, 1000], r: 765 },
+       { n: [0, 10], r: 0 },
+       { n: [100, 7, 4, 3], r: 2 },
+     ])('should mod and result $r', ({ n, r }) => {
+       expect(C.mod(...n)).toBe(r);
+     });
+   });
```

Issue #8

Fix subtract method

Como se ha comentado en una issue anterior, el método de subtract contenía un error. Este consistía en que el método reduce contenía un valor inicial de 0, y esto no es correcto. Si lo ejemplificamos con la diferencia de los valores 1, 2, 3, 4; tendríamos:

- Con el valor inicial: 0 - 1 - 2 - 3 - 4 = -10
- Sin el valor inicial: 1 - 2 - 3 - 4 = -9

El comportamiento que se quiere es el del no tener el valor inicial, pues queremos aplicar la diferencia valor por valor, sin añadir ningún más. La merge request contiene la eliminación del reduce, como arreglar los tests.

Issue #9

Power operator

El objetivo de esta issue es añadir la operación de potencia (**) a la calculadora. Le corresponde la merge request !9. En la misma, se ha añadido la operación como los tests que aseguran su funcionamiento.
Los cambios añadidos son los siguientes:

```
+   power(...values) {
+     return values.reduce((prev, curr) => prev ** curr);
+   },
```

Los tests añadidos son los siguientes:

```
+   describe('power', () => {
+     it.each([
+       { n: [0, 0], r: 1 },
+       { n: [2, 2, 3, 4], r: 16777216 },
+       { n: [10, 10, 10], r: 1e100 },
+       { n: [10, -10], r: 1e-10 },
+     ])('should power and result $r', ({ n, r }) => {
+       expect(C.power(...n)).toBe(r);
+     });
+   });
```

Issue #10

Bioperize operator

La bi-operación es una operación que me he inventado para darle un poco más de complejidad al código y a los tests. La operación tiene la siguiente signatura y funcionamiento:

- Recibe un argumento que es una tupla.
- El primer valor de la tupla es una tabla de referencias a las funciones de la calculadora.
- El segundo valor es un array de números.
- Se itera la tabla de números, de forma que: dado el índice actual de la iteración i, dos números n_r y n_{i+1}, y la operación op_i, se aplica el $op_i(n_r, n_{i+1})$. n_r es el resultado de la aplicación de $op_{i-1}(n_{r-1}, n_i)$. El proceso se repite hasta terminar los números.
- Devuelve el resultado de la iteración

Existen cuatro casos especiales:

- Si no se entran operaciones (no hay tabla o está vacía), debe lanzar un error.
- Si no se entran números (no hay tabla o está vacía), debe lanzar un error también.
- Si hay más operaciones que números, se terminará de aplicar operaciones cuando se terminen los números.
- Si hay más números que operaciones, se aplicará la última operación hasta terminar los números.

La implementación del método, es la siguiente (quizás es más claro en código que en la explicación):

```
+   bioperize([operations, values]) {
+     if (!operations?.length) {
+       throw Error('no operations received');
+     }
+     if (!values?.length) {
+       throw Error('no values received');
+     }
+
+     return values.reduce((prev, curr, i) => {
+       const op = operations[i - 1] ?? operations[operations.length - 1];
+       return op(prev, curr);
+     });
+   },
```

Los tests para los diferentes casos, son:

```
+  describe('bioperize', () => {
+    it('should solve basic operations', () => {
+      expect(
+        C.bioperize([
+          [C.add, C.subtract, C.multiply],
+          [1, 2, 1, 4],
+        ]),
+      ).toBe(8);
+    });
+
+    it('should apply the last operation if there are at least 2 more values than operators', () => {
+      expect(C.bioperize([[C.add], [1, 2, 3, 4, 5]])).toBe(15);
+    });
+
+    it('should not apply further operations if there are more operations than values', () => {
+      expect(
+        C.bioperize([
+          [C.add, C.subtract, C.add, C.divide],
+          [1, 2],
+        ]),
+      ).toBe(3);
+    });
+
+    it('should throw an error if there are no operations', () => {
+      try {
+        C.bioperize([[], [1, 2, 3]]);
+      } catch ({ message }) {
+        expect(message).toBe('no operations received');
+      }
+    });
+
+    it('should throw an error if there are no values', () => {
+      try {
+        C.bioperize([[C.add]]);
+      } catch ({ message }) {
+        expect(message).toBe('no values received');
+      }
+    });
+  });
```