TDD

Test Driven Development

What is TDD?

TDD process:

1. Determine the behaviour that needs to be implemented
2. Write a single FAILING test that verifies part of the behaviour
3. Write the simplest possible code that makes the test pass (then commit code)
4. Refactor your code for clarity and readability while keeping tests passing (then commit your code again)
5. Write another failing test to verify another part of the behaviour (iterate process from 2)

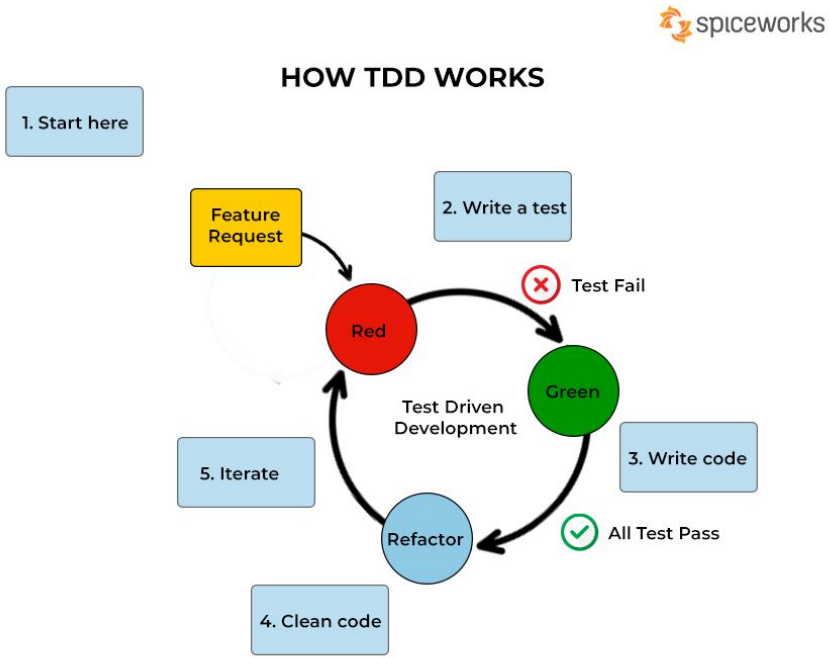


Diagram is courtesy of Spiceworks: https://www.spiceworks.com/tech/devops/articles/what-is-tdd/

Why?

- TDD leads to solutions of testable or loosely coupled modules. This means code that is more:
  - Flexible
  - Maintainable
  - Clean

- Makes intended behaviour documented through the tests

- Easy debugging. Focused around a narrow behaviour or scope

- Builds confidence in your code and that later changes does not break existing behaviour

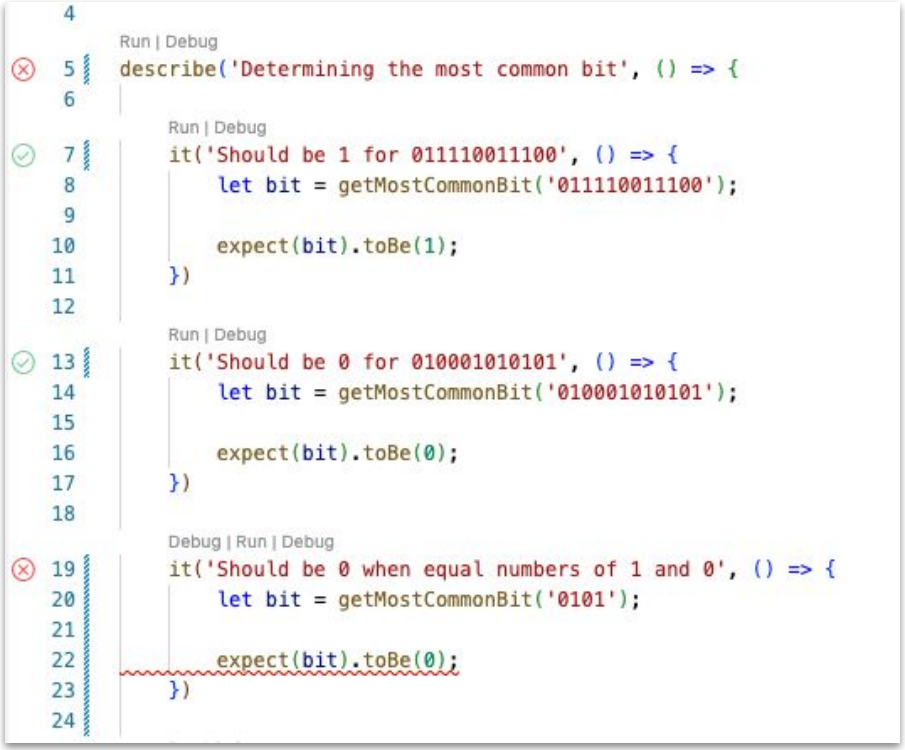
The three laws of TDD

1. No production code unless it is to make a test green
2. Limit the next unit test to what makes it fail
3. Do not write more production code than needed to make failing test pass

Refactoring - use the Rule of Three:

- Extract duplication only when you see it for the third time

Writing the failing test



```
4
   Run | Debug
5  describe('Determining the most common bit', () => {
6
       Run | Debug
7      it('Should be 1 for 011110011100', () => {
8          let bit = getMostCommonBit('011110011100');
9
10         expect(bit).toBe(1);
11     })
12
       Run | Debug
13     it('Should be 0 for 010001010101', () => {
14         let bit = getMostCommonBit('010001010101');
15
16         expect(bit).toBe(0);
17     })
18
       Debug | Run | Debug
19     it('Should be 0 when equal numbers of 1 and 0', () => {
20         let bit = getMostCommonBit('0101');
21
22         expect(bit).toBe(0);
23     })
24
```

Principles when writing tests

- Always write test before implementing production code
- Make sure the test is failing
- Focus on a specific functionality detail
- Write one test at a time
- Test should verify some behaviour existing tests does not
- Test one degree of freedom at a time
- Test behaviour - not structure
- Commit production code and test when tests are passing
- When you cannot think of another test - you are done

Write code

Three main steps

1. **Fake implementation**
   E.g. return the answer that satisfies the test
2. **Obvious (simple) implementation**
   Following the Transformation Priority Premise, TPP
3. **Triangulation with the next test**
   Starting with fake implementation and add more tests -> will force the code more generic

```
25
26  export function getMostCommonBit(line: string): number {
27
28      let zeros = 0;
29      let ones = 0;
30
31      for (let i = 0; i < line.length; i++) {
32
33          if (line.charAt(i) === '0') {
34              zeros += 1;
35          }
36          if (line.charAt(i) === '1') {
37              ones += 1;
38          }
39      }
40
41      if (zeros > ones) {
42          return 0;
43      }
44      return 1;
45  }
46
```

Transformation Priority Premise

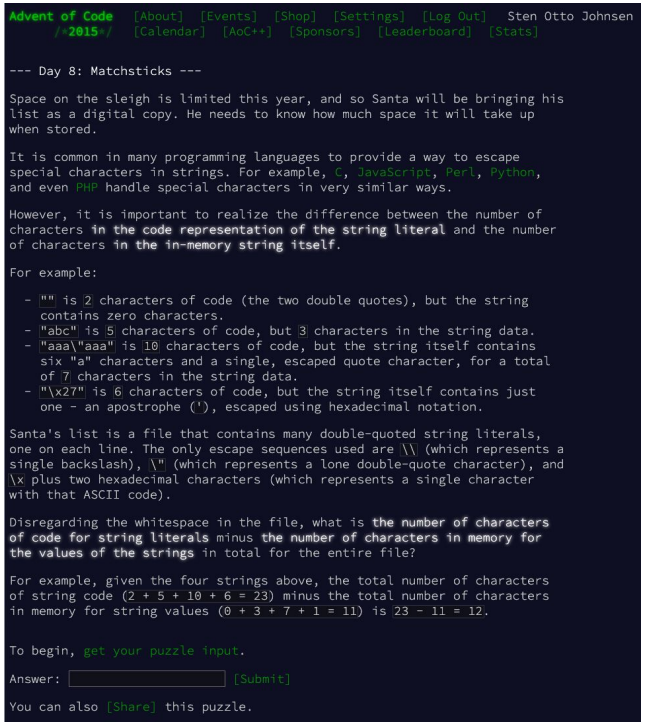
#	Transformation	Start code	End code
1	{} -> nil	{}	[return] nil
2	Nil -> constant	[return] nil	[return] “1”
3	Constant -> constant+	[return] “1”	[return] “1” + “2”
4	Constant -> scalar	[return] “1” + “2”	[return] argument
5	Statement -> statements	[return] argument	[return] min(max(0, argument), 10)
6	Unconditional -> conditional	[return] argument	if(condition) [return] 1 else [return] 0
7	Scalar -> array	dog	[dog, cat]
8	Array -> container	[dog, cat]	{dog=”DOG”, cat=”CAT”}
9	Statement -> tail recursion	a + b	a + recursion
10	If -> loop	if(condition)	loop(condition)
11	Statement -> recursion	a + recursion	recursion
12	Expression -> function	today – birth	CalculateBirthDate()
13	Variable -> mutation	day	var Day = 10; Day = 11;

Transformations on the top of the list preferred

When making test pass, do so with transformations that are simpler rather than more complex

> “As the tests get more specific, the code gets more generic.”
>
> - Robert C. Martin

Matching strings



Advent of Code [About] [Events] [Shop] [Settings] [Log Out] Sten Otto Johnsen
/*2015*/ [Calendar] [AoC++] [Sponsors] [Leaderboard] [Stats]

--- Day 8: Matchsticks ---

Space on the sleigh is limited this year, and so Santa will be bringing his list as a digital copy. He needs to know how much space it will take up when stored.

It is common in many programming languages to provide a way to escape special characters in strings. For example, C, JavaScript, Perl, Python, and even PHP handle special characters in very similar ways.

However, it is important to realize the difference between the number of characters **in the code representation of the string literal** and the number of characters **in the in-memory string itself**.

For example:

- "" is 2 characters of code (the two double quotes), but the string contains zero characters.
- "abc" is 5 characters of code, but 3 characters in the string data.
- "aaa\"aaa" is 10 characters of code, but the string itself contains six "a" characters and a single, escaped quote character, for a total of 7 characters in the string data.
- "\x27" is 6 characters of code, but the string itself contains just one - an apostrophe ('), escaped using hexadecimal notation.

Santa's list is a file that contains many double-quoted string literals, one on each line. The only escape sequences used are \\ (which represents a single backslash), \" (which represents a lone double-quote character), and \x plus two hexadecimal characters (which represents a single character with that ASCII code).

Disregarding the whitespace in the file, what is **the number of characters of code for string literals** minus **the number of characters in memory for the values of the strings** in total for the entire file?

For example, given the four strings above, the total number of characters of string code (2 + 5 + 10 + 6 = 23) minus the total number of characters in memory for string values (0 + 3 + 7 + 1 = 11) is 23 - 11 = 12.

To begin, get your puzzle input.

Answer: [Submit]

You can also [Share] this puzzle.

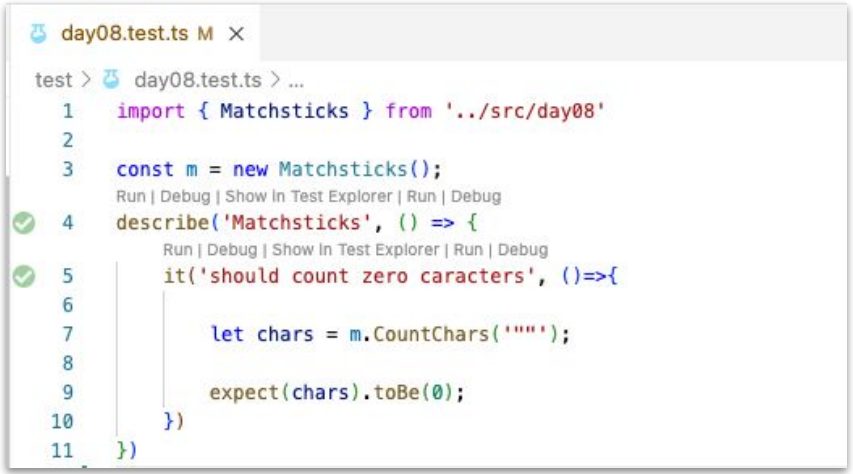
From **Advent of Code** 2015 day 8:

- Parse strings separating escape characters and code characters from the character in string and counting them
- We find that:
  - “” has zero characters but 2 characters of code
  - “abc” has 3 characters and 5 characters of code
  - “aaa\”aaa” has 7 characters and 10 characters of code
  - “\x27” has 1 character (‘) as hex ascii code and 6 characters of code
- Task is to separate the number of characters from the total number of characters of code and subtract the first from the second

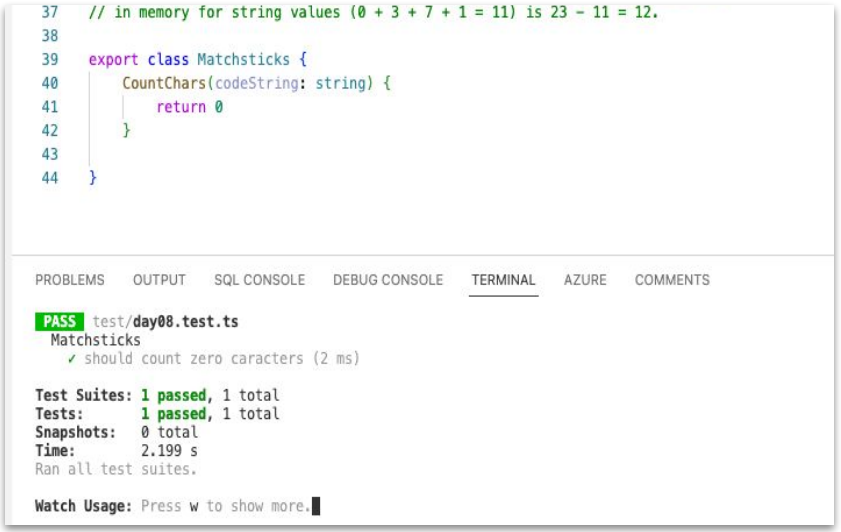
(Solution implemented in TypeScript)

TDD First steps

- Write simple test that verifies simplest functionality (an empty string)



- Implement fake first code to make test pass



Add test and see it fail

day08.test.ts M

test > day08.test.ts > ...

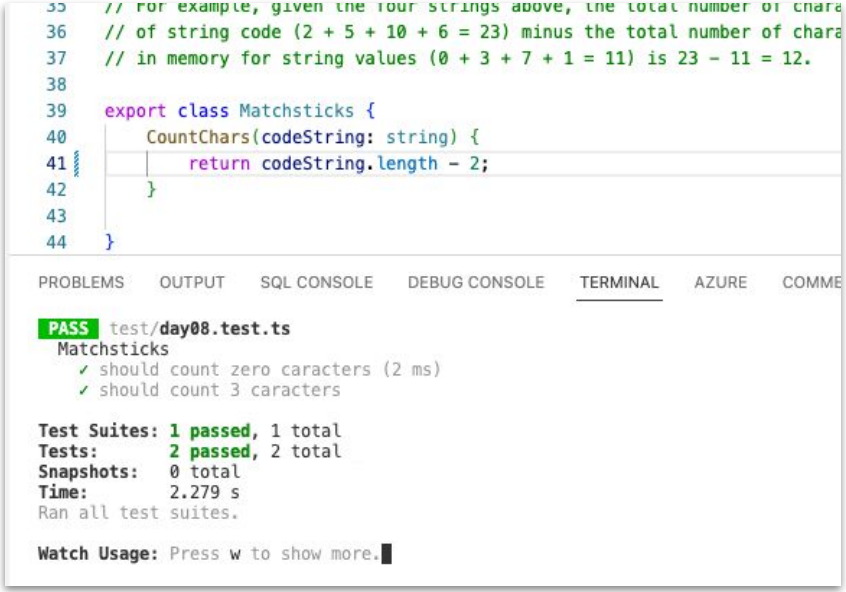
```
1   import { Matchsticks } from '../src/day08'
2
3   const m = new Matchsticks();
    Run | Debug | Show in Test Explorer | Run | Debug
4   describe('Matchsticks', () => {
        Run | Debug | Show in Test Explorer | Run | Debug
5       it('should count zero caracters', ()=>{
6
7           let chars = m.CountChars('""');
8
9           expect(chars).toBe(0);
10      })
11
        Run | Debug | Show Log | Show in Test Explorer | Run | Debug
12      it('should count 3 caracters', ()=>{
13
14          let chars = m.CountChars('"abc"');
15
16          expect(chars).toBe(3); // Expected: 3, Received: 0
17      })
18  })
19
```

```
38
39  export class Matchsticks {
40      CountChars(codeString: string) {
41          return 0
42      }
43
44  }
```

PROBLEMS OUTPUT SQL CONSOLE DEBUG CONSOLE TERMINAL AZURE COMMENTS

```
FAIL test/day08.test.ts
  Matchsticks
    ✓ should count zero caracters (2 ms)
    × should count 3 caracters (2 ms)

  ● Matchsticks › should count 3 caracters

    expect(received).toBe(expected) // Object.is equality

    Expected: 3
    Received: 0

      14 |         let chars = m.CountChars('"abc"');
      15 |
    > 16 |         expect(chars).toBe(3);
         |                       ^
      17 |     })
      18 | })
      19 |

      at Object.<anonymous> (test/day08.test.ts:16:23)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 passed, 2 total
Snapshots:   0 total
Time:        2.406 s
Ran all test suites.

Watch Usage: Press w to show more.
```

Write simple implementation to make test pass



When deciding for the implementation, we are considering the next, but highest possible level of the Transformation Priority Premise:

- Moving from returning a constant to returning advanced constant

Obviously not a sufficiently generalized solution for the end but satisfying the tests we have now

Moving on to Triangulation:

Time to commit and consider next test

Adding triangulation test

Next partial functionality will force us to generalise the code even further.

We can no longer avoid actually parsing the string, looking for specific escape characters

Make test green and refactor

```
38
39  export class Matchsticks {
40      CountChars(codeString: string): number {
41
42          let escaped = false;
43          let count = 0;
44          codeString.split('').forEach(code => {
45              if (code === '\\') {
46                  escaped = true;
47                  return
48              }
49              if (code == '"' && !escaped) {
50                  return
51              }
52
53              count++;
54              escaped = false;
55          })
56          return count;
57      }
58  }
59
```

PROBLEMS OUTPUT SQL CONSOLE DEBUG CONSOLE TERMINAL AZURE COMMENTS

```
PASS test/day08.test.ts
  Matchsticks
    ✓ should count zero caracters (1 ms)
    ✓ should count 3 caracters (1 ms)
    ✓ should count 7 caracters from escaped char in string

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        2.413 s
Ran all test suites.

Watch Usage: Press w to show more.
```

Add another test to expand functionality

```
3   const m = new Matchsticks();
    Run | Debug | Show in Test Explorer | Run | Debug
4   describe('Matchsticks', () => {
        Run | Debug | Show in Test Explorer | Run | Debug
5       it('should count zero caracters', ()=>{
6
7           let chars = m.CountChars('""');
8
9           expect(chars).toBe(0);
10      })
11
        Run | Debug | Show in Test Explorer | Run | Debug
12      it('should count 3 caracters', ()=>{
13
14          let chars = m.CountChars('"abc"');
15
16          expect(chars).toBe(3);
17      })
18
        Run | Debug | Show in Test Explorer | Run | Debug
19      it('should count 7 caracters from escaped char in string', ()=>{
20
21          let chars = m.CountChars('"aaa\\"aaa"');
22
23          expect(chars).toBe(7);
24      })
25
        Run | Debug | Show in Test Explorer | Run | Debug
26      it('should count 1 caracter from ascii char in string', ()=>{
27
28          let chars = m.CountChars('"\\x27"');
29
30          expect(chars).toBe(1);
31      })
32  })
33
```

```
41  export class Matchsticks {
42      CountChars(codeString: string): number {
43
44          let escaped = false;
45          let count = 0;
46          let ascii = "";
47          codeString.split('').forEach(code => {
48              if (code === '\\' && !escaped) {
49                  escaped = true;
50                  return
51              }
52              if (code == '"' && !escaped) {
53                  return
54              }
55
56              if (code === 'x' && escaped) {
57                  ascii += code;
58                  return
59              }
60
61              if (escaped && ascii !== "") {
62                  ascii += code;
63                  if (ascii.length < 3) {
64                      return
65                  }
66              }
67
68              ascii = "";
69              count++;
70              escaped = false;
71          })
72          return count;
73      }
74
```

Questions?

Thank you

How to contact me:
sten.johnsen@bouvet.no
@stenjo on Twitter