

## 3 – TDD, test-driven development

Till denna övning ska vi fortsätta med att skriva unit-tester med jest för vårt program från felsökningsövningen. Testen vi skriver ska hjälpa oss att implementera ny funktionalitet i vårt program! Vi vill sträva efter att arbeta i röd-grön-refaktorering cykler, ibland så kastar vi om ordning på stegen när det behövs.

Använd gärna describe-funktioner för att kapsla in testning av samma funktion och skriv test-texterna som meningar så att det lättare går att läsa i jest-loggen vad som har testats.

Vi ska fortsätta att arbeta vidare på programmet från felsökningsövningen.

Vi ska implementera 4 funktioner med de verktyg som vi kan: ***expensiveCheapProduct()***, ***removeProduct()***, ***averageProductPrice()*** och ***isStringAValidPrice()***. Arbeta gärna i smågrupper där varje deltagare fokuserar på att implementera en av funktionerna som man sedan delar med sig av! Prova gärna att arbeta med git och github där varje funktion implementeras på en egen sk feature-branch som ni sedan, när funktionen är klar, merge:ar in med master-branchen! I underrubrik **3 – Krav på funktionerna** finns kraven för funktionerna i punktlista.

### 1 – Genomgång av TDD med implementering av funktionen *removeProduct()*

För att komma igång med test-implementera-refaktorings cykeln så kan ni börja så här! Jag kommer använda funktionen *removeProduct()* som exempel.

Börja med ett test som helt enkelt testar om funktionen är skapad:

```
describe("Function removeProduct", () => {
  test("is implemented", () => {
    expect(functions.removeProduct).toBeDefined();
  });
});
```

I mitt fall har jag inte skapat funktionen ännu, så jag får följande resultat:

```
FAIL ./functions.test.js
Function removeProduct
  ✕ is implemented (7ms)

• Function removeProduct > is implemented

expect(received).toBeDefined()

Received: undefined

   183 | describe("Function removeProduct", () => {
   184 |   test("is implemented", () => {
>  185 |     expect(functions.removeProduct).toBeDefined();
       |                                     ^
   186 |   });
   187 | });
   188 |

at Object.toBeDefined (functions.test.js:185:37)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:  0 total
Time:        3.415s, estimated 9s
Ran all test suites.
npm ERR! Test failed.  See above for more details.
```

Jag har skapat ett test som blir fel, ett **rött test**. Jag går tillbaka till min fil med alla funktioner och skapar funktionen med en helt tom kropp, samt exporterar den:

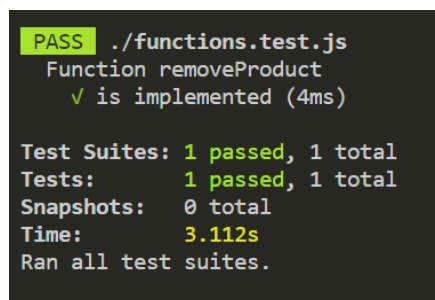
```
// functions.js
function removeProduct(){

}

const functions = {
  addProduct,
  totalProductCost,
  emptyList,
  removeProduct
}

if (typeof module === "object"){
  module.exports = functions;
}
```

Om jag kör om jest så klarar jag testet, jag har skrivit tillräckligt mycket kod för att skapa ett **grönt test** med inga röda test!



```
PASS ./functions.test.js
Function removeProduct
  ✓ is implemented (4ms)

Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 3.112s
Ran all test suites.
```

Om vi skulle ha några röda test skulle vi försöka åtgärda dem innan vi kan skriva nästa test. Eftersom vi inte har några sådana kan vi skriva ett nytt test som vi kan förvänta oss ska bli rött.

```
describe("Function removeProduct", () => {
  test("is implemented", () => {
    expect(functions.removeProduct).toBeDefined();
  });
  test("returns empty list when removing first product in list with 1 item", () => {
    const products = [{name: "Boll", price: 50}];
    const index = 0;
    expect(functions.removeProduct(products, index)).toEqual([]);
  });
});
```

Om vi kör detta får vi ett rött test.

Detta test vi nyss skrev guidar oss i **hur** vi vill att funktionen ska bete sig, nu kan vi fokusera på **implementeringen** av det nya beteendet! Låt oss använda den inbyggda **array metoden splice** för att lösa detta:

```
function removeProduct(products, index){
  products.splice(index, 1);
  return products;
}
```

Med denna implementering får vi bara gröna tester! Vi skriver ett till test:

```
test("contains 2 items when removing first product in list with 3 items",()=>{
  const products = [
    { name: "Boll", price: 50 },
    { name: "Spade", price: 100 },
    { name: "Sten", price: 1 }
  ];
  const index = 0;
  expect(functions.removeProduct(products, index)).toHaveLength(2);
});
```

Detta resulterar i att vi får ett grönt test! Vi har nu bevis på att vår implementering kan vara korrekt under dessa kriterier!

Låt oss skriva ett nytt test som utför två anrop till funktionen som använder samma produkt-lista som innan, men som vi flyttar ut som objekt för att lättare läsa koden:

```
test("removes 'Boll', contain 'Sten' and 'Spade'", () => {
  const boll = { name: "Boll", price: 50 };
  const spade = { name: "Spade", price: 100 };
  const sten = { name: "Sten", price: 1 };
  const products = [ boll, spade, sten ];
  const index = 0;
  expect(functions.removeProduct(products, index)).toContainEqual(sten);
  expect(functions.removeProduct(products, index)).toContainEqual(spade);
});
```

Detta test blir rött!

```
FAIL ./functions.test.js
Function removeProduct
  ✓ is implemented (3ms)
  ✓ returns empty list when removing first product in list with 1 item (1ms)
  ✓ contains 2 items when removing first product in list with 3 items (1ms)
  ✗ removes 'Boll', contain 'Sten' and 'Spade' (4ms)

● Function removeProduct › removes 'Boll', contain 'Sten' and 'Spade'

expect(array).toContainEqual(value) // deep equality

Expected value: {"name": "Spade", "price": 100}
Received array: [{"name": "Sten", "price": 1}]

   206 |     const index = 0;
   207 |     expect(functions.removeProduct(products, index)).toContainEqual(sten);
>  208 |     expect(functions.removeProduct(products, index)).toContainEqual(spade);
       |                                                         ^
   209 |   });
   210 | });
   211 |

at Object.toContainEqual (functions.test.js:208:54)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 3 passed, 4 total
Snapshots:  0 total
Time:        2.971s, estimated 3s
Ran all test suites.
npm ERR! Test failed.  See above for more details.
```

Det visar sig att vi har hittills gjort ett felaktigt antagande angående vår implementation av `removeProduct`. Om vi läser på om metoden `splice` för arrays så ser vi att den muterar array:en den anropas på! Dvs, anledningen till varför testet misslyckades var för att vår implementering av `removeProduct` modifierade listan som skickades in! Om vi skapar en kopia av listan och arbetar med den så får vi inte detta problem!

```
function removeProduct(products, index){
  const updateProducts = [...products];
  updateProducts.splice(index, 1);
  return updateProducts;
}
```

Kör vi testen igen så blir alla gröna. **Genom att arbeta iterativt på detta sätt så blir kodkvalitén bättre och bättre!**

Nu har vi skrivit en del kod, då kan det vara bra att ta ett steg tillbaka och se om vi kan **refaktorera** vår kod, dvs skriva om koden så att den innehåller samma logik men blir lättare att arbeta med. Vår funktion `removeProduct` är ganska liten med hyfsat bra variabelnamn så vi gör inget med den. Låt oss istället titta på all testkod vi hittills har skrivit:

```
describe("Function removeProduct", () => {
  test("is implemented", () => {
    expect(functions.removeProduct).toBeDefined();
  });
  test("returns empty list when removing first product in list with 1 item", () => {
    const products = [{ name: "Boll", price: 50 }];
    const index = 0;
    expect(functions.removeProduct(products, index)).toEqual([]);
  });
  test("contains 2 items when removing first product in list with 3 items", () => {
    const products = [
      { name: "Boll", price: 50 },
      { name: "Spade", price: 100 },
      { name: "Sten", price: 1 }
    ];
    const index = 0;
    expect(functions.removeProduct(products, index)).toHaveLength(2);
  });
  test("removes 'Boll', contain 'Sten' and 'Spade'", () => {
    const boll = { name: "Boll", price: 50 };
    const spade = { name: "Spade", price: 100 };
    const sten = { name: "Sten", price: 1 };
    const products = [ boll, spade, sten ];
    const index = 0;
    expect(functions.removeProduct(products, index)).toContainEqual(sten);
    expect(functions.removeProduct(products, index)).toContainEqual(spade);
  });
});
```

Vi ser att samma produkter används i flera av testen, samt att det är samma index-position vi använder. I testramverk är det vanligt med globala funktioner som kan köras före och efter testfunktioner har körts. **Här finns en lista på alla som man kan använda i jest.**

Låt oss använda den globala metoden ***beforeEach*** som kommer att köras före varje test, i den kan vi nollställa alla variabler inför testerna samt ge variablerna bättre namn!

```
describe("Function removeProduct", () => {

  let boll, spade, sten;
  let threeProducts, oneProduct;
  let firstElementIndex;
  beforeEach(()=>{
    boll = { name: "Boll", price: 50 };
    spade = { name: "Spade", price: 100 };
    sten = { name: "Sten", price: 1 };
    threeProducts = [ boll, spade, sten ];
    oneProduct = [ boll ];
    firstElementIndex = 0;
  });

  test("is implemented", () => {
    expect(functions.removeProduct).toBeDefined();
  });
  test("returns empty list when removing first product in list with 1 item", () => {
    expect(functions.removeProduct(oneProduct, firstElementIndex)).toEqual([]);
  });
  test("contains 2 items when removing first product in list with 3 items", () => {
    expect(functions.removeProduct(threeProducts, firstElementIndex)).toHaveLength(2);
  });
  test("removes 'Boll', contain 'Sten' and 'Spade'", () => {
    expect(functions.removeProduct(threeProducts, firstElementIndex)).toContainEqual(sten);
    expect(functions.removeProduct(threeProducts, firstElementIndex)).toContainEqual(spade);
  });
});
```

Om vi testkör testerna igen så ser vi att alla tester är gröna. Hoppas att denna introduktion hjälper er att komma igång med att implementera de resterande 3 funktionerna!

## 2 – Kör test kontinuerligt i bakgrunden

Inbyggt i jest finns möjligheten att låta jest köra testen i bakgrunden så fort du ändrar i koden, en sk "watcher". För att göra detta editerar vi i **package.json** med att lägga till ett nytt kommando under "scripts". I mitt fall döper jag kommandot till **testwatcher**, och det som ska köras är kommandot **jest** med flaggan **--watchAll**.

```
"scripts": {
  "test": "jest",
  "testwatch": "jest --watchAll"
},
```

För att starta igång den skriver du i terminalen, om du använt samma kommandonamn som jag, **npm run testwatch** vilket startar igång jest med följande hjälpkommandon:

```
Watch Usage
> Press f to run only failed tests.
> Press o to only run tests related to changed files.
> Press p to filter by a filename regex pattern.
> Press t to filter by a test name regex pattern.
> Press q to quit watch mode.
> Press Enter to trigger a test run.
```

Om menyn inte dyker upp på skärmen kan du trycka **w** för att få upp menyn:

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

Om du använder den inbyggda terminalen i Visual Studio Code kan du använda kortkommandot **Ctrl + Ö** för att togglar mellan att ta upp och dölja terminalen. Jest-watch:en kommer att köras i bakgrunden även om terminalen är dold.

### 3 – Krav på funktionerna

#### Funktionen `expensiveCheapProduct()`

- Indata: en lista på produkter  
Utdata: en lista med produkten som är billigast och produkten som är dyrast
- Om indata är en tom lista är utdata en tom lista
- Om indata innehåller en produkt så är utdata en lista med samma produkt

Valfri implementation av övriga krav ni kommer på för funktionen.

#### Funktionen `averageProductPrice()`

- Indata: en lista på produkter  
Utdata: medelvärde av en produkt i listan, avrundat till 2 decimaler
- Om indata är en tom lista är medelvärde 0

#### Funktionen `isStringAValidPrice()`

- Indata: en sträng  
Utdata: en boolean - true om strängen representerar ett giltigt pris, annars false
- Indata måste vara en sträng, annars returnera false
- Indata får inte vara en tom sträng för att vara ett giltigt pris
- Indata får inte innehålla tecken som ej klassificeras som Number enligt javascript
- Negativa numeriska värden är inte giltiga belopp på ett pris