

# Template Hspec

Generowanie przypadków testowych przy pomocy template-haskell i QuasiQuoters

# Hspec

Hspec (<http://hspec.github.io/>) to framework do testów BDD.

- czytelny dla osób nietechnicznych
- prosty w obsłudze

## Przykład

```
main = hspec $ do
  describe "absolute" $ do
    it "returns the original number when given a positive input" $
      absolute 1 `shouldBe` 1

    it "returns a positive number when given a negative input" $
      absolute (-1) `shouldBe` 1

    it "returns zero when given zero" $
      absolute 0 `shouldBe` 0
```

# Failed tests

```
absolute :: Int -> Int
absolute = undefined
```

Run:

```
$ runhaskell MathSpec.hs
```

absolute

returns the original number when given a positive input FAILED [1]

returns a positive number when given a negative input FAILED [2]

returns zero when given zero FAILED [3]

Failures:

MathSpec.hs:10:

1) absolute returns the original number when given a positive input

uncaught exception: ErrorCall (Prelude.undefined

CallStack (from HasCallStack):

error, called at libraries/base/GHC/Err.hs:79:14 in base:GHC.Err

undefined, called at Math.hs:5:12 in main:Math)

~~~~~

Finished in 0.0005 seconds

3 examples, 3 failures

## Passing tests

```
absolute :: Int -> Int  
absolute n = if n < 0 then negate n else n
```

Run:

```
$ runhaskell MathSpec.hs
```

```
absolute
```

```
  returns the original number when given a positive input
```

```
  returns a positive number when given a negative input
```

```
  returns zero when given zero
```

```
Finished in 0.0005 seconds
```

```
3 examples, 0 failures
```

## Quickcheck

```
describe "read" $ do
  it "is inverse to show" $ property $
    \x -> (read . show) x == (x :: Int)
```

Run:

```
$ runhaskell Spec.hs
```

```
read
  when used with ints
    is inverse to show
```

```
Finished in 0.0005 seconds
1 example, 0 failures
```

# Template Hspec

Chcielibyśmy mieć możliwość przetestowania analogicznych przypadków bez konieczności powtarzania kodu

```
main = hspec $ do
  describe "absolute" $ do
    it "returns the original number when given a positive input" $
      absolute 1 `shouldBe` 1

    it "returns a positive number when given a negative input" $
      absolute (-1) `shouldBe` 1

    it "returns zero when given zero" $
      absolute 0 `shouldBe` 0
```

## Moja propozycja:

```
main = hspec $ do
  describe "absolute" $ do
    $(they "returns #a when given #b" [|
      absolute a `shouldBe` b
    |] [testCases
      | a | b |
      | 1 | 1 |
      | -1 | 1 |
      | 0 | 0 |
    ]])
```



Prawie się udało:

```
main = hspec $ do
  describe "absolute" $ do
    $(they "returns #a when given #b" [|
      absolute a `shouldBe` b
    |] [testCases|
      |   a   |   b   |
      | 1::Int | 1::Int |
      | -1::Int | 1::Int |
      | 0::Int | 0::Int | |])
```

Run:

```
absolute
  returns 1 when given 1
  returns -1 when given 1
  returns 0 when given 0
```

```
Finished in 0.0009 seconds
3 examples, 0 failures
```

## Różne typy argumentów

```
$(they "show #a shows #b" [|  
  show a `shouldBe` b  
|] [testCases | a | b |  
  | 42::Int | "42" |  
  | ('4', '2') | "('4','2')"  
  | ["four", "two"] | "[\"four\", \"two\"]" |])
```

# Użyte narzędzia

## QuasiQuoter `testCases`

- `Data.Text`
- `haskell-src-meta`

`testCases` tworzy krotkę `([String], Int, t)`, której elementy oznaczają odpowiednio

- listę nazw zmiennych
- liczbę przypadków testowych do wygenerowania (`n`)
- krotkę `n`-elementową z wszystkimi potrzebnymi argumentami

## Generator testów `they`

- TemplateHaskell
- Parsec (*parsowanie opisów testów*)

Funkcja `prepareDesc` tworzy parametryzowany opis testu:

```
> runQ (prepareDesc ["a", "b"] "returns #a when given #b") >>= putStrLn . pprint
Data.Foldable.concat ["returns ",
    GHC.Show.show a,
    " when given ",
    GHC.Show.show b]
```

Najważniejszą funkcją jest `singleTest` , generująca pojedynczy test:

```
singleTest :: String -> ExpQ -> [String] -> ExpQ
singleTest desc test vars = lamE [tupP (map (varP . mkName) vars)]
                                   [| it $(prepareDesc vars desc) $(test) |]
```

Tworzy funkcję, która pobiera krotkę argumentów nazwanych tak jak zmienne w tabelce.

Na końcu powstaje funkcja łącząca wszystkie przypadki testowe, która wygląda tak:

```
\(a1, a2, a3, ..., a10) -> do  
  singleTestCase a1  
  singleTestCase a2  
  ...  
  singleTestCase a10  
  return ()
```

**Dziękuję za uwagę**

<https://github.com/piiteer/template-hspec>