

Practical class # 3 – Unit Tests

1) Open the project developed on practical class #2.

Open the app/build.gradle file and look at the dependencies. You see some dependencies marked as testImplementation and androidTestImplementation, which correspond to unit and instrumentation tests, respectively. The JUnit library that drives your unit tests, and lets you mark code as a test so that it can be compiled and run in such a way that it can test app code.

2) Open the ExampleUnitTest.kt file

It is located in the test directory and contains this class.

```
class ExampleUnitTest {  
    @Test  
    fun addition_isCorrect() {  
        assertEquals(4, 2 + 2)  
    }  
}
```


3) Run the test

Next to the addition_isCorrect method declaration, click the arrows and then select Run 'ExampleUnitTest.addition_isCorrect'.

Try to run a test which fails to check what happens.

4) Write the Unit Test for the Dice roller.

In the ExampleUnitTest.kt file, delete the generated test method and import statements. Your file should now look like this:

A screenshot of a code editor showing the contents of ExampleUnitTest.kt. The code is as follows:

```
package com.example.diceroller  
  
class ExampleUnitTest {  
}
```

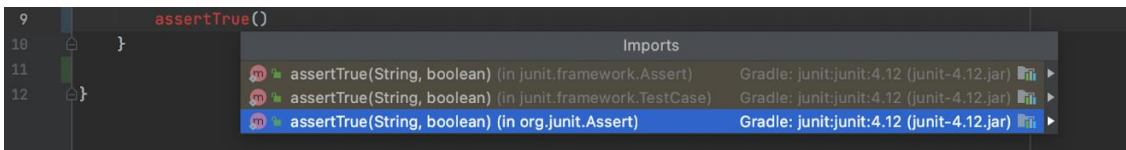
Create a generate_number() function annotated as @Test. Notice that when you try to call @Test, the text is red. This is because it cannot find the declaration of this annotation so you need to import it. You can do this automatically when you press Control+Enter. The code should look like this:

```
1 package com.example.diceroller
2
3 import org.junit.Test
4
5 class ExampleUnitTest {
6
7     @Test
8     fun generates_number() {
9
10    }
11
12 }
```

Create an instance of the Dice class and then call the roll function.

```
@Test
fun generates_number() {
    val dice = Dice(6)
    val rollResult = dice.roll()
}
```

Make an actual assertion. Assert that the method returned a value that is within the number of sides that you passed in. Thus, the value needs to be greater than 0 and less than 7. To accomplish this, use the `assertTrue()` method. Notice that when you try to call the `assertTrue()` method, the text is red at first. This is because it cannot find the declaration of this method so you need to import it, similar to what you encountered with the annotation.



```
9 assertTrue()
10
11 }
12 }
```

If you put your cursor between the parentheses and press `Control+P` you see a tooltip that shows what parameters the method takes.

The `assertTrue()` method takes two parameters: a `String` and a `Boolean`. If the assertion fails, the string is a message that displays in the console. The boolean is a conditional statement.

The final code should look like this:

```
@Test
fun generates_number() {
    val dice = Dice(6)
    val rollResult = dice.roll()
    assertTrue("The value of rollResult was not between 1 and 6", rollResult in 1..6)
}
```

Run the test. Modify the number of sides of the dice without modifying the test to see if it can fail.