School of Tech
part of accenture

TDD bowling game kata (Python)

Overview

- Do some real TDD on a basic app
- Write tests before app code
- Feel good (or, at least, better) about TDD

Objectives

- Use PyTest in Python
- Write tests before app code
- Run tests and see them fail
- Fill in the *minimum* code to make the test pass
- Repeat - with refactors!
- Don't break previous tests

Bowling Game Kata

This is a "kata" or fixed set of moves to practice in a given order. To be perfect, a kata should be performed many times.

> A *kata* is a sequence of moves practiced many times until it becomes instinct

School of Tech
part of accenture

A great intro

Let's all read the start of the description here:

> The kata: https://codingdojo.org/kata/Bowling/

A good resource

This video is what our session is based on - well worth a look offline:

> Great example: https://www.youtube.com/watch?v=BoTTSZI6wQg

When (not if!) you repeat this kata in the future, you can do worse than code along to the above!

School of Tech
part of accenture

Scoring Bowling

A super-short summary of 10 pin bowling:

- Each game consists of 10 *frames* of 2 bowls/rolls each
- In each *frame*, the bowler tries to knock down all 10 pins
- Knocking down all 10 pins gives you bonus scores
- A *spare* is when the bowler knocks down all the pins in 2 rolls
  - The score for that frame is 10 plus the number of pins knocked down on the next roll
- A *strike* is when the bowler knocks down all 10 pins on the first roll of the frame
  - The score for that frame is 10 plus the number of pins knocked down in the next two rolls
- If there is a spare or strike on the last frame, the bowler gets one or two extra rolls, respectively

School of Tech
part of accenture

Implementation

- We are creating a new feature for a bowling game that will calculate the score for a player
- We are going to use a test-driven-development (TDD) approach to develop this feature

School of Tech
part of accenture

Our coding Kata

> A *kata* is a sequence of moves practiced many times until it becomes instinct

- We'll write a unit test for each scoring scenario
- We'll implement **only** enough logic to get the current test to past
- There are 5 scenarios - so at least 5 happy-case tests to write, if not more!

Baby Steps

What is the simplest test we can write to make a start?

- "Gutter" game - all 20 rolls miss the pins
- all ones - all 20 rolls knock down a single pin
- spares and bonus
- strikes and bonus
- perfect game

School of Tech
part of accenture

Test Scenarios

- (1) Bowler has a *Gutter* game of all misses
  - `-/- -/- -/- -/- -/- -/- -/- -/- -/- -/- = 0`
- (2) Bowler throws *All ones*
  - `1/1 1/1 1/1 1/1 1/1 1/1 1/1 1/1 1/1 1/1 = 20`
- (3) Bowler gets a *Spare*
  - `5/5 3/- -/- -/- -/- -/- -/- -/- -/- -/- = 16`
- (4) Bowler gets a *Strike*
  - `10/ 3/4 -/- -/- -/- -/- -/- -/- -/- -/- = 24`
- (5) Bowler gets a *Perfect Game*
  - `10/ 10/ 10/ 10/ 10/ 10/ 10/ 10/ 10/ 10/ = 300`

Notes on (5) on next slide...

Test Scenarios - Errata

Consider this scenario:

- (5) Bowler gets a *Perfect Game*
    - `10/ 10/ 10/ 10/ 10/ 10/ 10/ 10/ 10/ 10/ = 300`

So: `10/` is a strike so the second ball is not thrown in a real bowling alley (as there are no pins left up!).

You can also represent this as `10/-` in the code if it helps. Technically the last frame can be shown as `10/10/10` or extra frames.

Task prep

- Open a terminal in the unit-testing-bowling-game-tdd/exercises folder
- Run `python3 -m pip install -r requirements.txt`
  - On windows use `py`
- This will complain there are no tests!

School of Tech
part of accenture

Task description (part 1/3)

- Write a unit test that requires the first scenario (1) to work in file `test_bowling_game.py`
- Run the unit tests with `python3 -m pytest`
  - This should fail (not even compile yet)
- Write the code to make it pass in file `bowling_game.py`
- Run the unit tests with `python3 -m pytest`
  - This should pass now

Task description (part 2/3)

> Hint:
>
> If you wish, you can make an *empty* function in `bowling_game.py`, and import it into `test_bowling_game.py` while making the initial test, so that the test compiles.
>
> But you can **not** put any code in the function till *after* the test is complete :-)

Task description (part 3/3)

- Write an *additional* unit test that requires the second scenario (2) to work, also in file `test_bowling_game.py`
- Update your code in `test_bowling_game.py` to make *both* tests pass with a single function

Then

- Repeat the above for the other scenarios (3,4,5), one at a time, adding a test for each and then making each pass

Task - Bowling Game Kata - 60 mins

> We will now go into breakout rooms, with an Instructor per room, and live code the whole kata of 5 exercises in each group

Each group starts with an empty app and test file from the `exercises.zip` and works through the scenarios with (at least) one happy-case test per scenario.

Task - Discussion - 10 mins

> How did we all get on?
>
> What learnings did we take away from the coding?

Some suggestions on next slide...

Some possible learnings

- Red, Green, Refactor (What does this mean?)
- Writing only the code you need *right now* is hard
- Don't predict the future
- Previously working tests should not be broken to make new ones work
- Ideally, previously working tests should not need to be be updated to make new ones work
- KISS, YAGNI, WYSIWYG and other relevant terms apply

Overview - recap

- Do some real TDD on a basic app
- Write tests before app code
- Feel good (or, at least, better) about TDD

Objectives - recap

- Use PyTest in Python
- Write tests before app code
- Run tests and see them fail
- Fill in the *minimum* code to make the test pass
- Repeat - with refactors!
- Don't break previous tests

Emoji Check:

On a high level, do you think you understand the main concepts of this session? Say so if not!

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively