1. The output from Behave does something like Red/Green/Refactor of TDD. What does it indicate when the steps are yellow in color? 1 / 1 point

- [✓ The steps are undefined.]
- The steps have passed.
- The steps have failed.
- The steps are skipped.

Correct
Correct! When the steps are undefined, they show as yellow.

2. How does Behave differentiate the type of step when it gives you snippets of code for the undefined steps? 1 / 1 point

- By showing None
- [✓ By giving a proper decorator keyword indicating the type of step: Given, When, or Then]
- By coloring the step red to show a failed step
- By coloring the step green to indicate passing of the step

Correct
Correct! Each code set has a proper decorator keyword indicating the type of step: Given, When, or Then. The remainder of the sentence is passed in as the string to match.

3. Which is the first line of code for creating a generic web steps file that works across web applications? 1 / 1 point

- @then(u'I should see the message "Success"')
- [✓ from behave import given, when, then]
- @given(u'I am on the "Home Page"')
- from behave import

Correct
Correct! This is the first step for importing the decorators.

4. Why does Behave skip certain steps and show them as blue? 1 / 1 point

- The steps belong to another scenario
- The steps are the default steps
- Behave cannot find them in the feature file
- [✓ To show they are skipped when the current scenario has failed]

Correct
Correct! Behave only skips the remaining steps in the current failing scenario.

5. In the Behave workflow, which step should you implement after the first step passes? 1 / 1 point

- Repeat the process until all steps pass
- Implement a step
- After the first step passes, all steps turn green
- [✓ Implement the next step that fails]

Correct
Correct! You implement the next step that's red, and then repeat until the steps are all green.

6. Which steps show the NotImplementedError exception? 1 point

- [✓ Missing steps]
- Failed steps
- Passed steps
- Default steps

Incorrect
Incorrect. Review the Implementing Your First Steps video.

7. In the following code, which element has the message string? 1 / 1 point

assert message in str(context.response.data)

- [✓ context.response.data]
- context.response
- context
- str

Correct
Correct! The step references the data attribute of context.response and asserts that the message string passed into the function can be found somewhere in the data.

8. How does context act as a container in your feature file? 1 / 1 point

- It contains description of the steps.
- It stores data of other feature files.
- [✓ It stores whatever data you need to make available to all of your steps.]
- It contains different data for each step.

Correct
Correct! It stores whatever data you need that is available to all of your steps for the duration of the feature file.

9. How can you pass information from one step to another? 1 / 1 point

- [✓ Store information in the context variable of one step and call the variable in another step]
- Store the information in import and call it from another step
- Store the information in the get method and call it from another step
- Store the information in data and call the string function

Correct
Correct! To pass information between steps, store it in the context variable of one step and call on that variable in another step.

10. What are rules to substitute variables in your steps? Select two. 1 / 1 point

- [✓] Replace data in the decorator string with variables enclosed by curly braces

Correct
Correct! You need to replace data in the decorator string with variables enclosed by curly braces.

- [ ] Replace data in the decorator with variables preceded by an underscore
- [ ] Add step implementation parameters with different names from the variables
- [✓] Substitute the variable names in place of the strings passed in from the feature file

Correct
Correct! You need to substitute the variable names in place of the strings passed in from the feature file.