# Behavior Driven Development (BDD) Lab - Behave

# Introduction

In software development, Behavior Driven Development is a methodology that promotes collaboration and testing automation among developers and testers. Behavior in the software can be defined as how features operate. BDD is an approach that emphasizes the behaviors of features in the software. There are many BDD frameworks for Python, but this tutorial will focus on using Behave.

# Getting Started

1. Behave can be installed by executing the following command:  
  
a. pip install behave

This command can be executed on your local command prompt or through the terminal in Python if that’s the specified programming language being used. It’s optional to install other packages such as Selenium, for web browser interactions. Or the package for requests, for API calls. These packages are not required but can be useful for testing purposes.

2. Create a directory for the project called “features.” This is where you will store all your feature files. In this same directory, create a file called “fruitstand.feature.” This file will contain the following:

Feature: Running the Fruit Stand Application  
As a user  
I want to be able to run the Fruit Stand application  
So that I can interact with its features and navigate the application   
Scenario: Launching the Application  
Given I have initiated the application by running the app module in my project  
When I launch the application with the given link provided by the output   
Then I should see the main screen for the Fruit Stand application

3. Create another directory for the project called “features/steps.” In this directory, create a file called “fruitstand.py.” The file will contain the following:

from behave import \*  
@given(‘I have initiated the application by running the app module in my project’)  
def run\_the\_application()  
# Code to start the application  
pass   
@when(‘I launch the application with the given link provided by the output’)

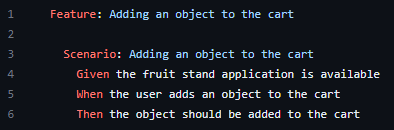
# Code to open the application  
def open\_the\_application():  
pass  
@then(‘I should see the main screen for the Fruit Stand application’)  
# Code that verifies that the main screen of the application is displayed   
def main\_screen\_of\_application():  
pass

4. Finally, it’s time to run the tests using Behave. Use the behave command to run all the tests for the project. If you would like to run the scenarios in the feature file, use the command behave features/fruitstand.feature.

# Adding an Item

Now, we can create more features to test our application. The expected outcome of a user selecting and adding an item to their cart, that the cart will be updated with the change. Let’s test that.

1. In your features directory, create a new file named, “additem.feature”. This file will contain the following:



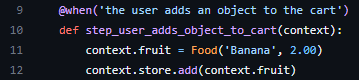
1. Now, in the features/steps directory, create a file called add\_shopping\_cart\_steps.py. All following steps should be in that file.
2. Import all dependencies



1. Let’s write the given step:



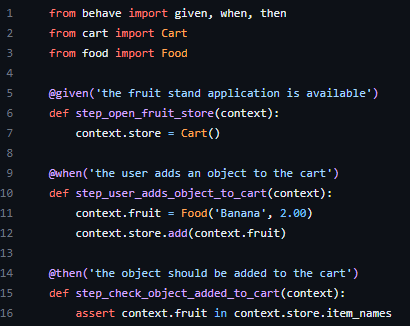
1. Let’s write the “when” step:



1. Let’s write the “then” step:



1. Put it all together:

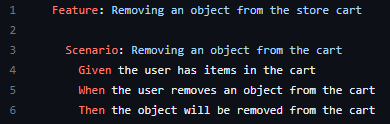


1. And finally, run in command prompt using the behave command!

# Removing an Item

Let’s also make sure that removing an item from your cart works as well. This will be extremely similar to the adding an item test.

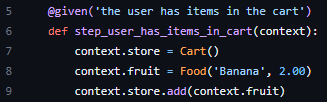
1. In your features directory, create a new file named, “removeitem.feature”. This file will contain the following:



1. Now, in the features/steps directory, create a file called remove\_shopping\_cart\_steps.py. All following steps should be in that file.
2. Import all dependencies:



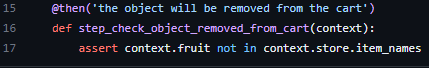
1. Let’s write the given step:



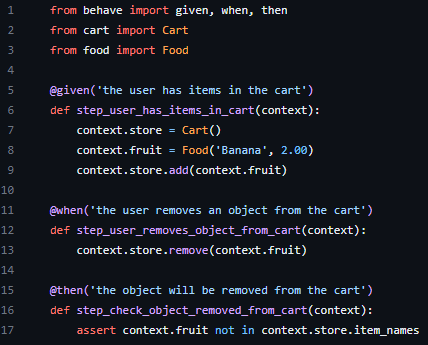
1. Let’s write the “when” step:



1. Let’s write the “then” step:



1. Put it all together:



1. And finally, run in command prompt using the behave command!

# Applying a Discount Code to Cart Total

Let’s make sure that the discount code functions as expected.

1. In your features directory, create a new file named, “discountcode.feature”. This file will contain the following:
   1. Feature: Shopping Cart
      1. Scenario: Applying a discount code to the cart total
         1. Given the fruit stand application is open and the user has items in their cart
         2. When the user applies discount code “DISCOUNT20”
         3. Then the cart total should be updated with a 20% discount
2. Now, in the features/steps directory, create a file called ‘discount\_code\_steps.py’. All the following steps should be in that file.
3. Import Behave
   1. From behave import given, when, then
4. Let’s write the given step:
   1. ‘@given(‘the user has fruit items in the cart’)
   2. Def step\_user\_has\_items\_in\_cart(context):
   3. ADD CODE HERE
5. Let’s write the when step:
   1. ‘@when(‘the user attempts to apply a discount code {DISCOUNT20}’)’
   2. Def step\_apply\_discount\_code(context, discount\_code):
   3. ADD CODE HERE
6. Let’s write the then step:
   1. ‘@then(‘the cart total should be updated with a 20% discount’)
   2. Def step\_check\_discount\_applies(context):
   3. ADD CODE HERE
7. Put it all together
   1. ADD CODE HERE
8. Finally, run in command prompt using the behave command!

# Removing an Item not in the Cart

Next, let’s check to see how the application behaves when a user tries to remove an item that is not in the cart.

1. In your features directory, create a new file named, “notincart.feature”. This file will contain the following:
   1. Feature: Shopping Cart
      1. Scenario: Removing an item not in the cart
         1. Given the fruit stan application is open and the user has items in the cart
         2. When the user attempts to remove an incorrect item from the cart
         3. Then the cart contents should remain the same
2. Now, in the features/steps directory, create a file called ‘not\_in\_cart.py’. All the following steps should be in that file:
3. Import Behave
   1. From behave import given, when, then
4. Let’s write the given step:
   1. ‘@given(‘the user has items in their cart’)
   2. Def step\_user\_has\_items\_in\_cart(context):
   3. ADD CODE HERE
5. Let’s write the when step:
   1. ‘@when(‘the user attemps to remove fruit from the cart’)
   2. Def step\_attempt\_remove\_incorrect\_item(context, fruit):
   3. ADD CODE HERE
6. Let’s write the then step:
   1. ‘@then(‘the cart’s contents should remain unchanged’)
   2. Def step\_check\_cart\_unchanged(context):
   3. ADD CODE HERE
7. Put it all together
   1. ADD CODE HERE
8. Finally, run in command prompt using the behave command!

# Cleared Cart After Purchase

Finally, let us check to see if the cart is successfully cleared after a user makes a purchase.

1. In your features directory, create a new file named, “afterpurchasefeature”. This file will contain the following:
   1. Feature: Shopping Cart
      1. Scenario: Removing an item not in the cart
         1. Given the fruit stan application is open and the user has items in the cart
         2. When the user clicks the ‘Purchase’ button
         3. Then the cart should be empty
2. Now, in the features/steps directory, create a file called ‘after\_purchase.py’. All the following steps should be in that file:
3. Import Behave
   1. From behave import given, when, then
4. Let’s write the given step:
   1. ‘@given(‘the user has items in their cart’)
   2. Def step\_user\_has\_items\_in\_cart(context):
   3. ADD CODE HERE
5. Let’s write the when step:
   1. @when(‘the user clicks the “Purchase” button’)
   2. def step\_click\_purchase\_button(context):
   3. ADD CODE HERE
6. Let’s write the then step:
   1. ‘@then(‘the cart’s contents should remain unchanged’)
   2. def step\_check\_cart\_empty(context):
   3. ADD CODE HERE
7. Put it all together
   1. ADD CODE HERE
8. Finally, run in command prompt using the behave command!