Hands-On Lab: Setting Up an Environment in Behave

Estimated time needed: 15 minutes

Welcome to the **Setting Up an Environment in Behave** lab. By setting up the environment in Behave, you can control what happens before and after each feature, scenario, step, or tag. You can also control what happens before all and after all features, which is the focus of this lab.

In this lab, you will set up the initial environment for BDD testing. To do so, you will import configuration values from environment variables that you will use for testing.

Learning Objectives

After completing this lab, you will be able to:

- Integrate environment variables into your BDD environment
- Group configuration information in the BDD context

About Theia

Theia is an open-source IDE (Integrated Development Environment) that can be run on desktop or on cloud. You will be using the Theia IDE to complete this lab. When you log into the Theia environment, you are presented with a 'dedicated computer on the cloud' exclusively for you. This is available to you as long as you work on the labs. Once you log off, this 'dedicated computer on the cloud' is deleted along with any files you may have created. So, it is a good idea to finish your labs in a single session. If you finish part of the lab and return to the Theia lab later, you may have to start from the beginning. Plan to work out all your Theia labs when you have the time to finish the complete lab in a single session.

Set Up the Lab Environment

You have a little preparation to do before you can start the lab.

Open a Terminal

Open a terminal window by using the menu in the editor: Terminal > New Terminal.

In the terminal, if you are not already in the `/home/projects` folder, change to your project folder now.

```
cd /home/project
```

Clone the Code Repo

Now, get the code that you need to test. To do this, use the `git clone` command to clone the git repository:

```
git clone https://github.com/ibm-developer-skills-network/duwjx-tdd_bdd_PracticeCode.git
```

Change into the Lab Folder

Once you have cloned the repository, change to the lab directory:

```
cd duwjx-tdd_bdd_PracticeCode/labs/08_environment_setup
```

Install Python Dependencies

The final preparation step is to use `pip` to install the Python packages needed for the lab:

```
python3.8 -m pip install -r requirements.txt
```

You are now ready to start the lab.

Optional

If working in the terminal becomes difficult because the command prompt is very long, you can shorten the prompt using the following command:

```
export PS1="[\[\033[01;32m\]\u\[\033[00m\]: \[\033[01;34m\]\W\[\033[00m\]]\$ "
```

Navigate to the Code

In the IDE on the right of your screen, navigate to the `duwjx-tdd_bdd_PracticeCode/labs/08_environment_setup` folder. This folder contains all of the source code that you will use for this lab.

You will edit the file called `environment.py` throughout this lab.

Working with Environment Variables

You can read an environment variable using the `getenv()` function from the `os` package. For your first parameter, you pass in the name of the environment variable you want. For an optional second parameter, you can pass in the default that you would like returned if the environment variable doesn't exist. Otherwise, the function returns `None`.

For example, if you want to get an environment variable called `MESSAGE` with a default value of `Hello`, you write the following code:

```
from os import getenv
MESSAGE = getenv('MESSAGE', 'Hello')
```

This code returns the value of the environment variable `MESSAGE` or, if that doesn't exist, the string `Hello`.

Add the BASE_URL Environment Parameter

You will start by importing an environment variable called `BASE_URL` and saving it in the `context` as `base_url`. Doing so allows the Python steps to know which website to test.

Your Task

In `environment.py`, implement the following steps:

1. Use the `getenv()` function to get an environment variable called `BASE_URL`.
2. Set this function妡ult to `http://localhost:8080` in case `BASE_URL` doesn’t exist.
3. Add the `BASE_URL` environmental variable to the `context` as `base_url` before any tests are run.

Open **environment.py** in IDE

▼ Click here for a hint.

You set context variables in the before_all() function.

Solution

▼ Click here for the solution.

This is the solution for adding BASE_URL to the context.

```
BASE_URL = getenv('BASE_URL', 'http://localhost:8080')
def before_all(context):
    """ Executed once before all tests """
    context.base_url = BASE_URL
```

Run the Tests

Run `behave` to ensure that it runs successfully. There are no features yet. You are just testing to ensure that `environment.py` has no errors.

```
behave
```

Your output should look like this:

```
$ behave
0 features passed, 0 failed, 0 skipped
0 scenarios passed, 0 failed, 0 skipped
0 steps passed, 0 failed, 0 skipped, 0 undefined
```

```
Took 0m0.000s
```

Add the WAIT_SECONDS Environment Parameter

Next, you will import an environment variable called `WAIT_SECONDS` and save it in the `context` as `wait_seconds`. Doing so lets the Python steps know how long to wait for the application to respond.

Your Task

In `environment.py`, implement the following steps:

1. Use the `getenv()` function to get an environment variable called `WAIT_SECONDS`.
2. Set the default to `60` in case `WAIT_SECONDS` doesn't exist.
3. Convert the data to an integer before assigning it.
4. Add the `WAIT_SECONDS` environment variable to the `context` as `wait_seconds` before any tests are run.

▼ Click here for a hint.

You set context variables in the before_all() function.

Solution

▼ Click here for the solution.

This is the solution for adding WAIT_SECONDS to the context.

```
BASE_URL = getenv('BASE_URL', 'http://localhost:8080')
WAIT_SECONDS = int(getenv('WAIT_SECONDS', '60'))
def before_all(context):
    """ Executed once before all tests """
    context.base_url = BASE_URL
    context.wait_seconds = WAIT_SECONDS
```

Note: You can cast `WAIT_SECONDS` to an `int` when defining it, as shown in this example, or cast it to an `int` before assigning it to `context.wait_seconds`. Either solution works.

Run the Tests

Run `behave` to ensure that it runs successfully. There are no features yet. You are just testing to ensure that `environment.py` has no errors.

```
behave
```

Your output should look like this:

```
$ behave
0 features passed, 0 failed, 0 skipped
0 scenarios passed, 0 failed, 0 skipped
0 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.000s
```

Conclusion

Congratulations! You just completed the **Setting Up an Environment** lab.

Hopefully, you now understand how to ensure your testing environment sets up appropriately before and after you run your BDD tests. Anything you need to initialize should be in the `before_all()` function, and anything you need to shut down should be in the `after_all()` function. You should use the `context` to pass configuration information to the Python steps.

Author(s)

John J. Rofrano