Armen Stepanians
March 30, 2018

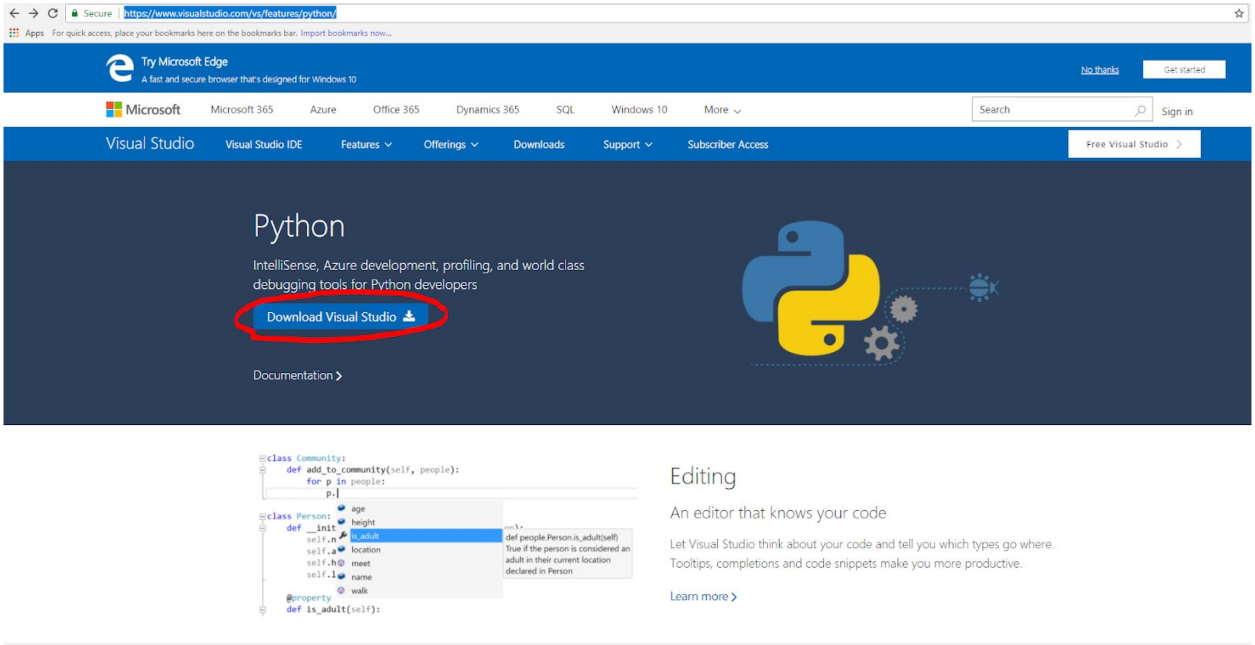
Microsoft Visual Studio Setup for Python

Setup

There are a few steps that need to be completed before starting to work on a Python project on Microsoft Visual Studio. These including downloading and installing the IDE, as well as setting up the proper Python environment and libraries which will be used in this project. These project instructions are done in a Windows 10 OS environment.

Downloading the IDE

Head over to https://www.visualstudio.com/vs/features/python/ and download the IDE from the provided link highlighted below. Once hovering over the button, a drop down will appear. Choose the community edition.



IntelliSense, Azure development, profiling, and world class debugging tools for Python developers

```
class Community:
    def add_to_community(self, people):
        for p in people:
            p.|

class Person:
    def __init
        self.n
        self.a
        self.h
        self.l

    @property
    def is_adult(self):
```

def people.Person.is_adult(self)
True if the person is considered an adult in their current location declared in Person

An editor that knows your code

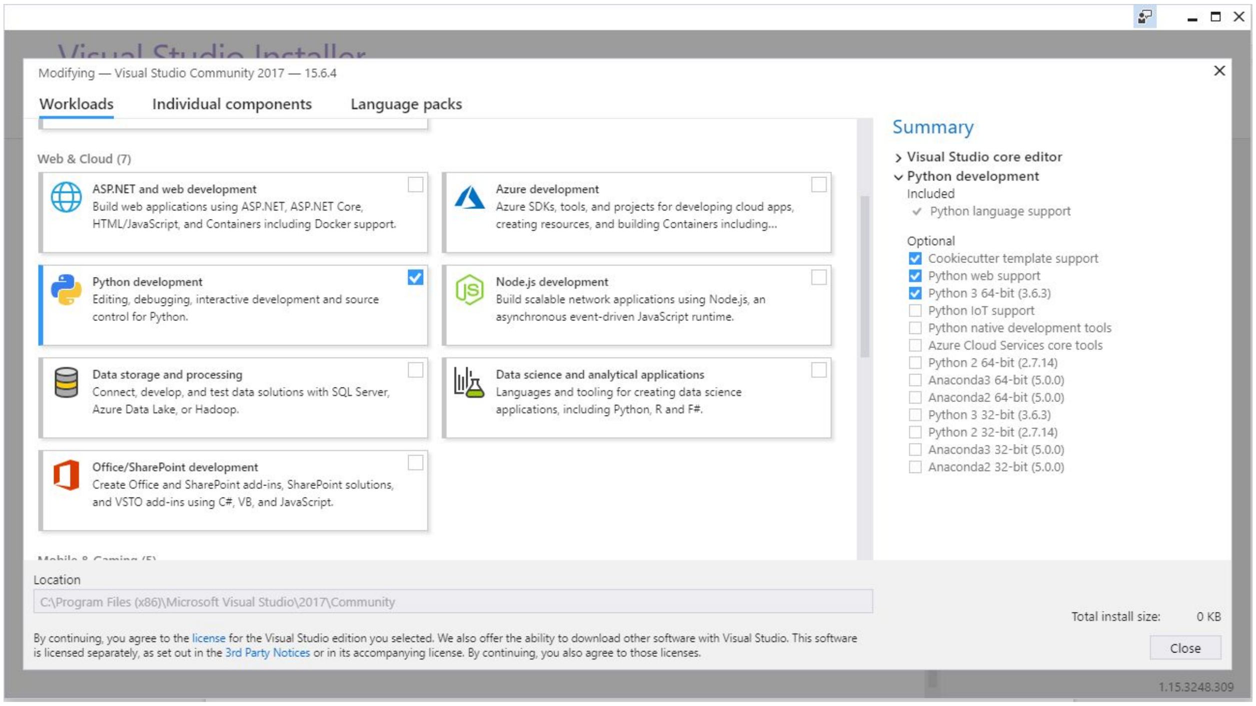
Let Visual Studio think about your code and tell you which types go where. Tooltips, completions and code snippets make you more productive.

Figure 1 - Downloading the IDE

There are different versions of the IDE available on the provided website. Download the latest stable available version (2017).

Installing the IDE

After downloading and installing the downloader, a window will pop open, asking you to choose the modules to install. Select the Python Development modules, as seen below. Then click next, and trigger the installation. This will first download all the necessary modules from the Microsoft servers (~2GB), and start installing.



Modifying — Visual Studio Community 2017 — 15.6.4

Web & Cloud (7)

□ ASP.NET and web development
Build web applications using ASP.NET, ASP.NET Core, HTML/JavaScript, and Containers including Docker support.

□ Azure development
Azure SDKs, tools, and projects for developing cloud apps, creating resources, and building Containers including...

☑ Python development
Editing, debugging, interactive development and source control for Python.

□ Node.js development
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

□ Data storage and processing
Connect, develop, and test data solutions with SQL Server, Azure Data Lake, or Hadoop.

□ Data science and analytical applications
Languages and tooling for creating data science applications, including Python, R and F#.

□ Office/SharePoint development
Create Office and SharePoint add-ins, SharePoint solutions, and VSTO add-ins using C#, VB, and JavaScript.

Summary

- Visual Studio core editor
- Python development
  - Included
    - ✓ Python language support
  - Optional
    - ☑ Cookiecutter template support
    - ☑ Python web support
    - ☑ Python 3 64-bit (3.6.3)
    - □ Python IoT support
    - □ Python native development tools
    - □ Azure Cloud Services core tools
    - □ Python 2 64-bit (2.7.14)
    - □ Anaconda3 64-bit (5.0.0)
    - □ Anaconda2 64-bit (5.0.0)
    - □ Python 3 32-bit (3.6.3)
    - □ Python 2 32-bit (2.7.14)
    - □ Anaconda3 32-bit (5.0.0)
    - □ Anaconda2 32-bit (5.0.0)

Location
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total install size: 0 KB

1.15.3248.309

Figure 2 - Installation

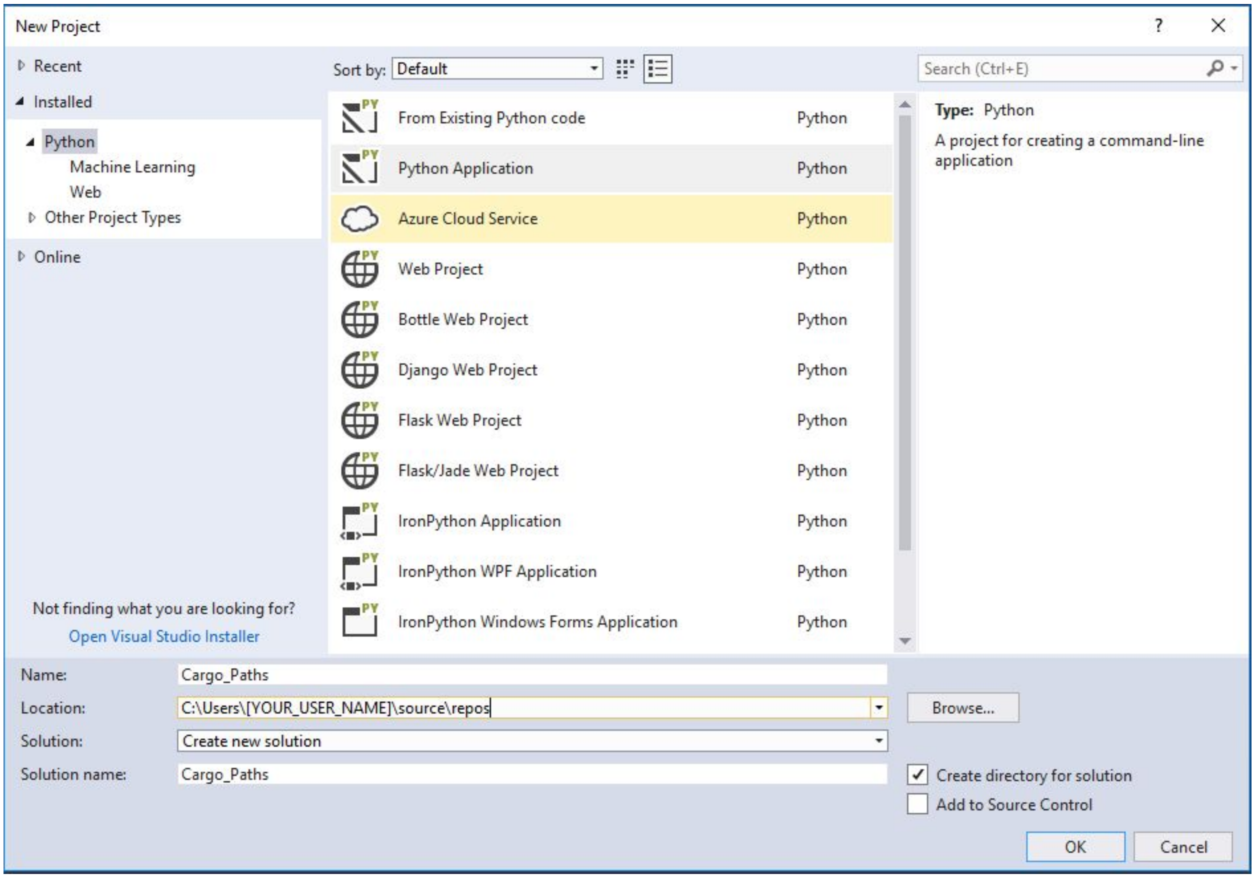
Installing Mockito

In order to run some of the tests, the python Mockito module will be used. This is an external Python library, and hence needs to be separately installed. In order to do so, open the command line from the start menu by typing *cmd* in the search bar, and selecting *Command Prompt* from the list. Once opened, type in *pip install mockito*. This will install the mockito module, so the interpreter can recognize the library once we use it. Make sure a standard supported version of Python is installed on the machine beforehand. This project is completed using *Python 3.4* (available in this link), therefore the reader is recommended to proceed with the same version as well.

Using Microsoft Visual Studio

Creating a Python Project

In order to create a new Python project, open Microsoft Visual Studio, and go to **File -> Name -> Project**. A window will appear, asking you to choose a project type. Choose *Python Application*. You can name the project in the text bar labeled *Name*. The default name is *PythonApplication1*. Configure the location in which you want the project to be saved in. Figure 3 shows these steps. The recommended path is the one seen in the figure. Replace the *[YOUR_USER_NAME]* with the username configured on the machine.



New Project
?
Recent
Installed
Python
Machine Learning
Web
Other Project Types
Online
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Search (Ctrl+E)
From Existing Python code Python
Python Application Python
Azure Cloud Service Python
Web Project Python
Bottle Web Project Python
Django Web Project Python
Flask Web Project Python
Flask/Jade Web Project Python
IronPython Application Python
IronPython WPF Application Python
IronPython Windows Forms Application Python
Browse...
OK
Cancel

Type: Python

A project for creating a command-line application

Name: Cargo_Paths

Location: C:\Users\[YOUR_USER_NAME]\source\repos

Solution: Create new solution

Solution name: Cargo_Paths

☑ Create directory for solution

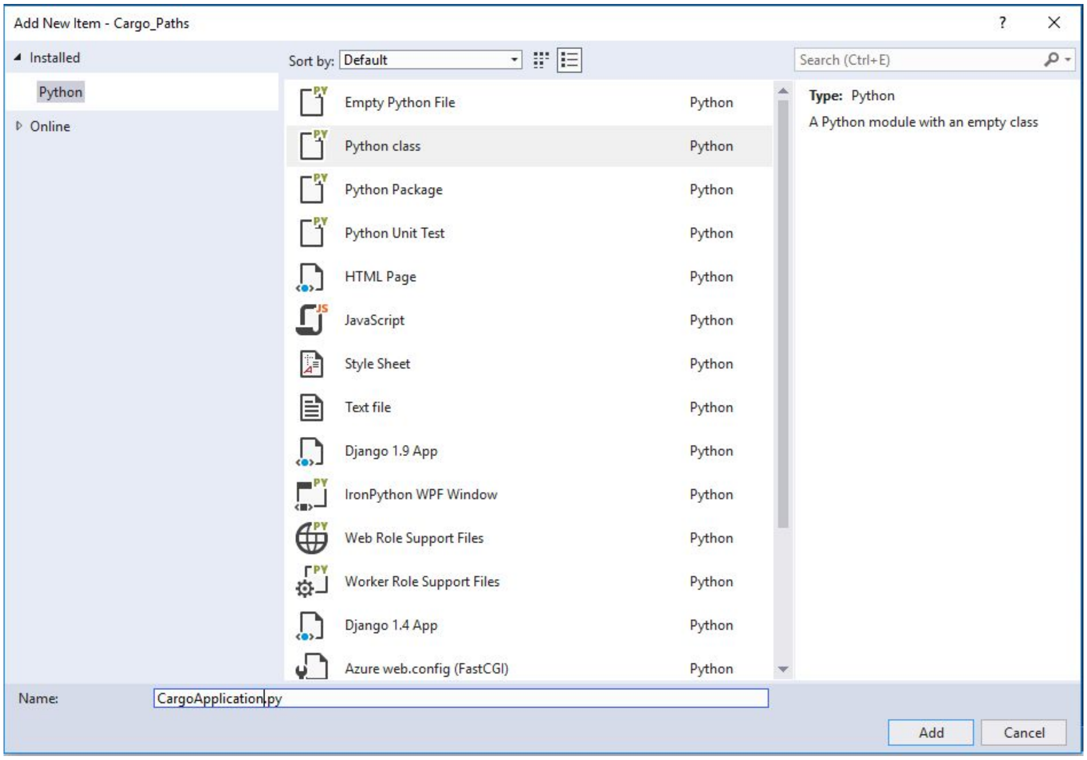
□ Add to Source Control

Figure 3 - Creating a new project

Note that the *Name* and *Solution name* fields are configured to be the same by default at the time of the project creation. However, they can be manually set to be different names.

Creating a Python Class

To create a new python class within the project, simply right click on the solution name in the *Solution Explorer* (located on the top right corner). For example, from our case illustrated above, the solution name would be *Cargo_Paths*. From the appearing menu, go to **Add -> New Item**. A window will appear asking you to choose the type of item to be added. Select *Python class* and name the class. These steps are illustrated in the figure below. The same steps follow for test class creation.



Add New Item - Cargo_Paths

Sort by: Default

Empty Python File	Python
Python class	Python
Python Package	Python
Python Unit Test	Python
HTML Page	Python
JavaScript	Python
Style Sheet	Python
Text file	Python
Django 1.9 App	Python
IronPython WPF Window	Python
Web Role Support Files	Python
Worker Role Support Files	Python
Django 1.4 App	Python
Azure web.config (FastCGI)	Python

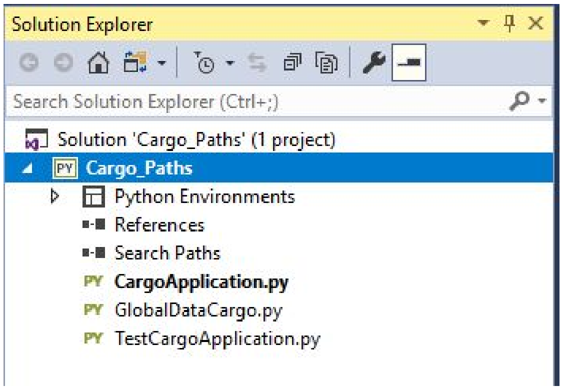
Type: Python

A Python module with an empty class

Name: CargoApplication.py

Figure 4 - Python class creating window

After the creation and implementation of all the classes and test cases, the solution explorer should look something similar like this:



Solution Explorer

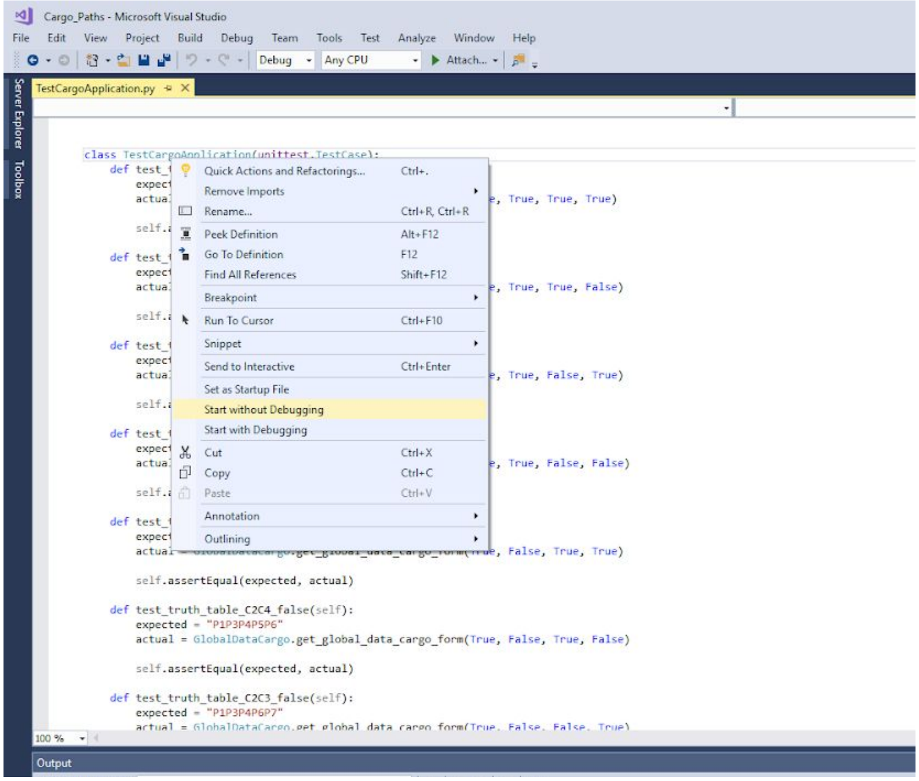
Search Solution Explorer (Ctrl+;)

```
Solution 'Cargo_Paths' (1 project)
  Cargo_Paths
    Python Environments
    References
    Search Paths
    CargoApplication.py
    GlobalDataCargo.py
    TestCargoApplication.py
```

Figure 5 - Sample solution explorer after implementation

Running Unit Tests

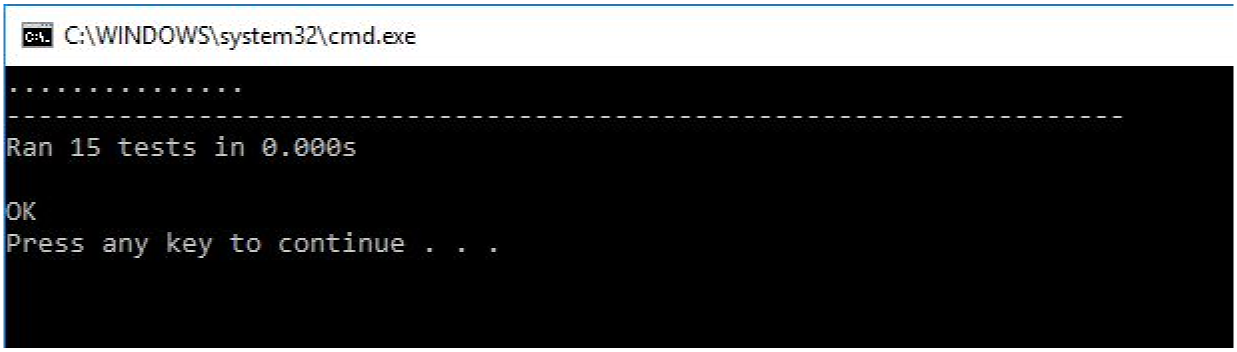
To run the unit tests, open the desired test file. The *TestCargoApplication.py* file is chosen to demonstrate as an example. Right click on any part of the file. This will show a list of option, one which is *Start without debugging,* shown in Figure 6. This is the MS Visual Studio equivalent of running the code. This option runs all the available tests in the file.



```
class TestCargoApplication(unittest.TestCase):
    def test_
        expect
        actua                                       e, True, True, True)

        self.a

    def test_
        expect
        actua                                       e, True, True, False)

        self.a

    def test_
        expect
        actua                                       e, True, False, True)

        self.a

    def test_
        expect
        actua                                       e, True, False, False)

        self.a

    def test_
        expect
        actual                                      e, False, True, True)

        self.assertEqual(expected, actual)

    def test_truth_table_C2C4_false(self):
        expected = "P1P3P4P5P6"
        actual = GlobalDataCargo.get_global_data_cargo_form(True, False, True, False)

        self.assertEqual(expected, actual)

    def test_truth_table_C2C3_false(self):
        expected = "P1P3P4P6P7"
        actual = GlobalDataCargo.get_global_data_cargo_form(True, False, False, True)
```

Figure 6 - Running the python unit tests in MS Visual Studio

When running the tests, a command prompt window will appear, showing the results of the tests. It will show the result of each test case ran. If the test case is passed, the result will appear as a "**.**" (dot), indicating a successful run. In our case, there were 15 test cases, all of which have passed, hence 15 dots.

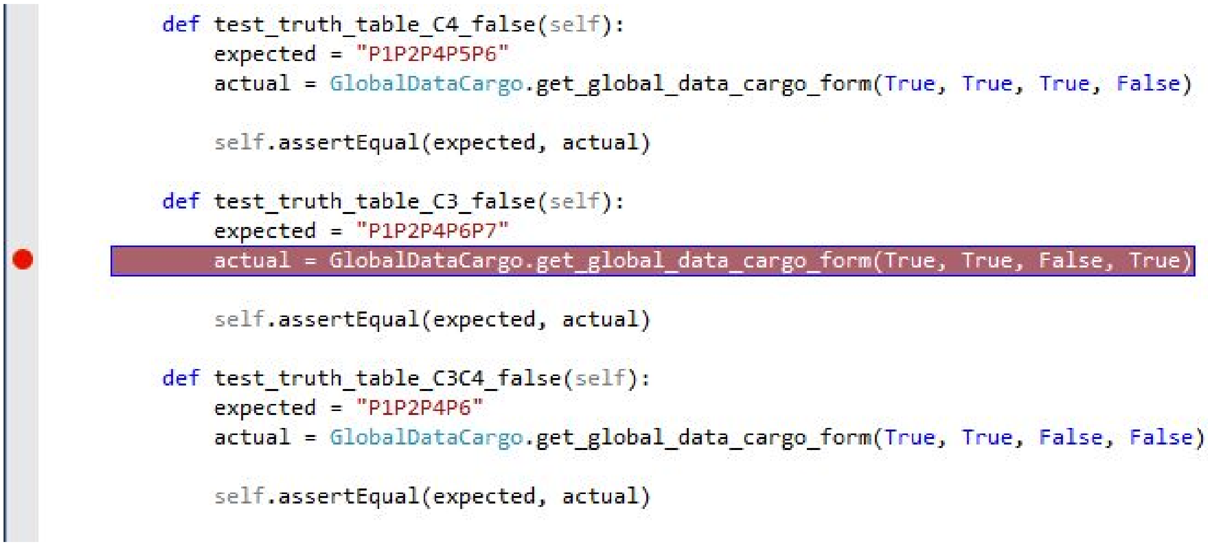


```
...............
----------------------------------------------------------------------
Ran 15 tests in 0.000s

OK
Press any key to continue . . .
```

Figure 7 - Unit test results

Debugging Unit Tests

To debug a test case, first we have to set a breakpoint on the desired line. This can be achieved by clicking the left side of the desired line, on the vertical bar, shown in Figure 8. Then, right click anywhere in the file, and from the menu shown in Figure 6, choose *Start with debugging* this time. The rest of the steps are similar to the previous section, but now we have the debugging functionality enabled, and hence we can debug our program.



```
    def test_truth_table_C4_false(self):
        expected = "P1P2P4P5P6"
        actual = GlobalDataCargo.get_global_data_cargo_form(True, True, True, False)

        self.assertEqual(expected, actual)

    def test_truth_table_C3_false(self):
        expected = "P1P2P4P6P7"
        actual = GlobalDataCargo.get_global_data_cargo_form(True, True, False, True)

        self.assertEqual(expected, actual)

    def test_truth_table_C3C4_false(self):
        expected = "P1P2P4P6"
        actual = GlobalDataCargo.get_global_data_cargo_form(True, True, False, False)

        self.assertEqual(expected, actual)
```

Figure 8 - Debugging a test case