|  |  |  |
| --- | --- | --- |
| Document name | : | HR.Skills.A2W7ST |
| Document description | : | This document contains the supporting topic: “Unit Testing”. |
| Document Version | : | V 1.0 |
| Written by | : | Danny de Snoo |
| Date(s) written | : | 06/06/2024 |

Supporting topic description:

**Introduction**

Unit testing is like checking individual ingredients before you bake a cake. Imagine you have flour, sugar, eggs, and butter. You want to make sure each ingredient is good on its own before you mix them together. In programming, unit testing means testing small parts, or units, of a software application in isolation to ensure they work as expected. It helps developers catch bugs early, making it easier to fix them before they cause bigger issues in the overall program. Just like you want your cake ingredients to be perfect, unit testing ensures each piece of your code is reliable before it all comes together to create the final product. In unit testing, we use programming to test units automatically. In programming, developers write code to automatically test individual parts of their software.

**Activity**

* Review the provided code snippet below.
* The function test() has been designed to evaluate the functionality of other functions (i.e., units). Examine the code to understand how each function is tested, and consider adding additional test cases to complete the remaining todos.
* This code currently employs if statements and print statements for implementing tests. Python offers a more efficient solution through the use of assert statements. Conduct a brief research on assert statements and study some examples.
* Within the test() function, replace the existing if statements with appropriate assert statements.

***Note: The approach demonstrated here represents a fundamental introduction to unit testing. Later, you will experience more in-depth techniques using Python's built-in frameworks and libraries, eliminating the need for developers to integrate testing logic within their programs.***

Code given for this supporting topic:

|  |
| --- |
| contacts = []  def add\_contact(name, phone\_number, email):  contact = {  'name': name,  'phone\_number': phone\_number,  'email': email  }  contacts.append(contact)  def search\_by\_name(name):  return list(filter(lambda c: name.lower() in c['name'].lower(), contacts))  def delete\_contact(name):  for contact in contacts:  if contact['name'].lower() == name.lower():  contacts.remove(contact)  def test():  # Test adding a contact  add\_contact('John Doe', '06876543210', 'john@hotemail.com')  # Let's check if the function works correctly  if len(contacts) != 1 or contacts[0]['name']!='John Doe':  print('Test: ERROR in add\_contact()')  # Test searching contacts  search\_results = search\_by\_name("John")  # Let's check if the function works correctly  if len(search\_results) < 1:  print('Test: ERROR in search\_by\_name()')  # Test deleting a contact  delete\_contact("John Doe")  # todo: Implement a test here.  print("All tests are executed.")  test() |

* The function test() has been designed to evaluate the functionality of other functions (i.e., units). Examine the code to understand how each function is tested, and consider adding additional test cases to complete the remaining todos.

|  |
| --- |
| contacts = []  def add\_contact(name, phone\_number, email):  contact = {  'name': name,  'phone\_number': phone\_number,  'email': email  }  contacts.append(contact)  def search\_by\_name(name):  return list(filter(lambda c: name.lower() in c['name'].lower(), contacts))  def delete\_contact(name):  for contact in contacts:  if contact['name'].lower() == name.lower():  contacts.remove(contact)  def test():  # Test adding a contact  add\_contact('John Doe', '06876543210', 'john@hotemail.com')  # Let's check if the function works correctly  if len(contacts) != 1 or contacts[0]['name']!='John Doe':  print('Test: ERROR in add\_contact()')  # Test searching contacts  search\_results = search\_by\_name("John")  # Let's check if the function works correctly  if len(search\_results) < 1:  print('Test: ERROR in search\_by\_name()')  # Test deleting a contact  delete\_contact("John Doe")  # CHECK HAS BEEN ADDED  if len(contacts) != 0:  print('Test: ERROR in delete\_contact()')  print("All tests are executed.")  test() |

* This code currently employs if statements and print statements for implementing tests. Python offers a more efficient solution through the use of assert statements. Conduct a brief research on assert statements and study some examples.

|  |  |  |
| --- | --- | --- |
| Example code (of a test that will fail):   |  | | --- | | def func(x)  return 4  def test\_example():  assert func(3) == 3, “We get the result we want” |   Example code (of a test that will succeed)   |  | | --- | | def func(x)  return x  def test\_example():  assert func(3) == 3, “We get the result we want” |   Assert is a construct the allows you to test your code and your assumptions about its results. It is a check function to make sure that conditions are met during its execution.  The purpose of the Assert statement is to check for logic errors and assumptions in your code.  Assert can be used in debugging to check for example that a certain function returns the value that you assume it should. Example  We made a function that would should return a certain value based on a certain input. With assert we can test this assumption and see if our assumption is correct and if there are any logic errors. |

|  |  |  |
| --- | --- | --- |
| **Sources** |  | **Website url:** |
| pytest | : | https://docs.pytest.org/en/8.2.x/ |
| browserstack | : | https://www.browserstack.com/guide/assert-in-python#:~:text=In%20Python%2C%20the%20assert%20statement,the%20execution%20of%20a%20program. |

* Within the test() function, replace the existing if statements with appropriate assert statements.

|  |
| --- |
| contacts = []  def add\_contact(name, phone\_number, email):  contact = {  'name': name,  'phone\_number': phone\_number,  'email': email  }  contacts.append(contact)  def search\_by\_name(name):  return list(filter(lambda c: name.lower() in c['name'].lower(), contacts))  def delete\_contact(name):  for contact in contacts:  if contact['name'].lower() == name.lower():  contacts.remove(contact)  def test\_addcontact():  add\_contact('John Doe', '06876543210', 'john@hotemail.com')  assert len(contacts) == 1 or contacts[0]['name'] == 'John Doe'  def test\_searchcontact():  add\_contact('John Doe', '06876543210', 'john@hotemail.com')  search\_results = search\_by\_name("John")  assert len(search\_results) >= 1  def test\_deletecontact():  add\_contact('John Doe', '06876543210', 'john@hotemail.com')  delete\_contact("John Doe")  assert len(contacts) == 0 |