



Debugging and Testing


Python Fundamentals

Name of presenter

Date

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Welcome to Debugging and Testing.



Debugging

What you will learn

At the core of the lesson

You will learn how to:

- Explain the purpose of debugging and testing code
- Explain how debuggers enable developers to find bugs in their code
- Recognize how to perform a static analysis on Python code
- Recognize how to evaluate Python code for ways to implement dynamic analysis



In this lesson, you will learn how to:

- Explain the purpose of debugging and testing code
- Explain how debuggers enable developers to find bugs in their code
- Recognize how to perform a static analysis on Python code
- Recognize how to evaluate Python code for ways to implement dynamic analysis

What is debugging?

Logging can be a useful tool for finding logical problems in code.

Debugging is done to identify defects in the code itself. It is simple in concept and it has two parts:

Finding errors in your code

Fixing those errors

Debugging can be complicated in practice. Every language has tools for debugging to mitigate this challenge.

PDB: The Python debugger

The Python debugger is activated by entering:

```
Python -m pdb <filename>
```

The first line of the code runs, and then a prompt (pdb) appears.

You can now use several commands.

All of these tools are used to find errors and help developers identify the reason behind them.

Types of debugging

Static analysis

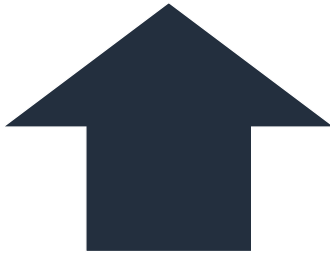
Dynamic analysis

Static analysis

Static analysis has these characteristics:

- It can be done continuously in the development process.
- The Python interpreter includes a level of static analysis because it reports syntax and semantic errors. Integrated development environments (IDEs) can help identify issues while you write the code.
- It might consider factors such as proper nesting, function calls, and code complexity

Static analysis: Advantages and disadvantages



Advantages of static analysis include:

- You can identify the exact location of code issues.
- It has a faster turnaround time for fixes.
- Later tests have fewer issues.
- Earlier detection of bugs reduces costs.



Disadvantages of static analysis include:

- Manual analysis is time consuming.
- Automation tools can produce false positives and false negatives.
- Automation might result in taking security for granted.
- Automation is only as good as the parameters that are used to set up the tool.

Demonstration: Static Debugging



9

Instructions

1. Create a file that is called: **debug.py**
2. In that file, write a basic Python program based on a topic that you have learned. Make sure that you include a programming error. It could be a syntax error, a spelling error, or a different kind of error.
3. Run the program and show how the error is displayed.



Code example: The following code has two errors.

```
# Python program with 2 errors
var = "Double Value"
sumvalue = var + 4

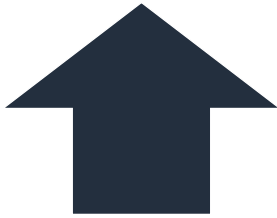
def dosomething(valuetocheck):
    if valuetocheck > 4:
        print("Bad indent")
```

Dynamic analysis

Dynamic analysis has these characteristics:

- It is done by analyzing running applications
- Most IDEs for Python include a "debugging" mode. In this mode the application runs and a developer can execute code "line by line", until a particular line of code is called, or until a variable has a value (equal, less than, greater than).
- A invaluable technique during dynamic analysis is to write out values and conditions happening in a running application of a log file.

Dynamic analysis: Advantages and disadvantages



Advantages of dynamic analysis include:

- It identifies issues in a runtime environment (it could be a test server or a live version of the software).
- You can analyze applications even when you cannot observe the actual code.
- It can prove or confirm the false negatives that you identified from static analysis.
- It can be used with every application.



Disadvantages of dynamic analysis include:

- Automation might result in taking security for granted.
- You cannot guarantee full coverage.
- It can be more difficult to isolate code issues.

Assertions


- Assertions are conditions, such as *if* statements, that check the values in the application
- Dynamic analysis uses assertion statements during runtime to raise errors when conditions certain conditions occur.
- As an example consider the following function. The developer wants to ensure that the age value is always a positive number greater than zero. The following assertion checks this:

```
def loguserage(age):  
    assert age >= 0, "Invalid age was supplied"
```

Log monitoring

Developers, in code, writes to a text file often referred to as a log file. As certain conditions happen in the running application, the logging code writes information to the log file

What does log monitoring do?



Keep track of errors in a running program

Keep records of the last time the program was run for later review, maybe to see what went wrong in a specific area

With log monitoring you get a full view of the application as it is running. The application can be exercised by a user and the developer can see inspect the log file for a “real world” use of the application.

What to log?

Consider every significant event in your application:

- Where did it occur?
- What time did it happen?
- What were the arguments?
- What is the state of important resources?

Capture all information when an error occurs:

- All arguments
- Exception, plus inner exceptions
- Traceback object: Stack traces

Log monitoring tools

In Python, the default, native log monitoring tool is the library *logging*.

To have access to the library

Enter `import logging` at the top of a Python file.

- That command enables access to customizable error messages that can be assigned different priority levels and then saved into a file for later review.

To save the output into a file

Add a line of code before you use the *logging* library:
`logging.basicConfig(filename="app.log", level=logging.DEBUG)`

Demonstration: Dynamic Debugging



16

Instructions

1. Open the **debug.py** file.
2. Modify this file to take user input.
3. How can you dynamically test whether the value that the user supplies is greater than 0?



Code example:

```
# Ask the user for a value and confirm the supplied value is greater than 0
```

```
def checkvalue(valuetocheck):  
    assert (type(valuetocheck) is int), "You must enter a number."  
    assert (valuetocheck > 0), "Value entered must be greater than 0"  
    if valuetocheck > 4:  
        print("Value is greater than 4")
```

```
var = int(input("Enter a number greater than 0: "))  
checkvalue(var)
```




Software testing

What you will learn

At the core of the lesson

You will learn how to:

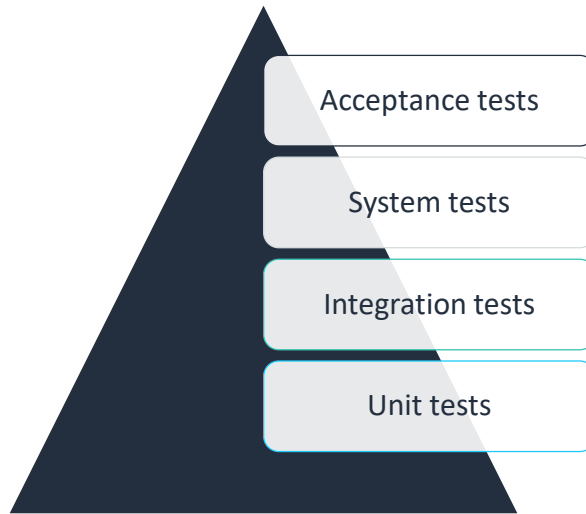
- Explain why software testing is necessary



In this lesson, you will learn how to:

- Explain why software testing is necessary

Testing



Unit tests

A *unit* is the smallest testable part of any software. It is the most basic level of testing. A unit usually has only one or a few inputs with a single output.

An example of a unit test is verifying each individual function of a program. Developers are responsible for their own unit testing.

Integration tests

Individual units are combined and tested as a group. You test the interaction between the different parts of the software so that you can identify issues.

Analogy: When a pen is manufactured, all the pieces of the pen (units) are produced separately—the cap, the body, the ink cartridge, and so on. All components are tested individually (unit testing). When more than one unit is ready, you can test them to see how they interact with each other.

A developer can perform integration tests, but dedicated testers are frequently used to do these tests.

System tests

A complete and integrated application is tested. This level determines whether the software meets specific requirements.

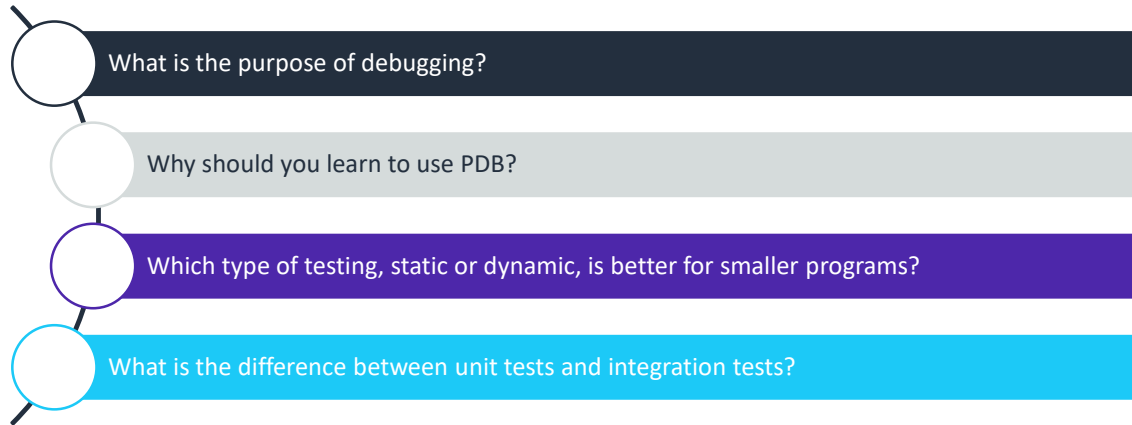
Analogy, continued: When the pen is assembled, testing is performed to see if the pen works. Does it write in the correct color? Is it the specified size?



Acceptance testing

Acceptance testing is formalized testing that considers user needs, business needs, and whether the software is acceptable for delivery to the final user.

Checkpoint questions



Answers:

1. Debugging enables developers to verify an application's logic.
2. PDB, which is the Python debugger, enables developers to run various tools on their code for possible errors.
3. For smaller programs, static analysis is usually better because logic errors can be quickly identified in small code bases.
4. Unit tests cover code for functionality, and integration tests check the overall functionality of all parts of an application.

Key takeaways



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

25

- The continuous integration of code can include style checks and many forms of testing.
- Use logs to capture when errors occur in a program at runtime.
- Debugging is used to find errors in a program, and it can be done statically or dynamically.
- Many types of testing are done to ensure that applications work as expected, such as:
 - Unit testing
 - Integration testing
 - System testing
 - Acceptance testing

aws re/start

Some key takeaways from this lesson include:

- The continuous integration of code can include style checks and many forms of testing.
- Use logs to capture when errors occur in a program at runtime.
- Debugging is used to find errors in a program, and it can be done statically or dynamically.
- Many types of testing are done to ensure that applications work as expected, such as:
 - Unit testing
 - Integration testing
 - System testing
 - Acceptance testing