"Error handling" redirects here. Not to be confused with Error detection and correction.

This article is about computing. For knowledge, see fact checking and problem solving.

In computing and computer programming, **exception handling** is the process of responding to the occurrence of *exceptions* – anomalous or exceptional conditions requiring special processing – during the execution of a program. In general, an exception breaks the normal flow of execution and executes a pre-registered *exception handler*; the details of how this is done depend on whether it is a hardware or software exception and how the software exception is implemented. Exception handling, if provided, is facilitated by specialized programming language constructs, hardware mechanisms like interrupts, or operating system (OS) inter-process communication (IPC) facilities like signals. Some exceptions, especially hardware ones, may be handled so gracefully that execution can resume where it was interrupted.

Python Exception Handling

We have explored basic python till now from Set 1 to 4 (Set 1 | Set 2 | Set 3 | Set 4).

In this article, we will discuss how to handle exceptions in Python using try. except, and finally statement with the help of proper examples.

Error in Python can be of two types i.e. Syntax errors and Exceptions. Errors are the problems in a program due to which the program will stop the execution. On the other hand, exceptions are raised when some internal events occur which changes the normal flow of the program.

Difference between Syntax Error and Exceptions

Syntax Error: As the name suggests this error is caused by the wrong syntax in the code. It leads to the termination of the program.

Exceptions: Exceptions are raised when the program is syntactically correct, but the code resulted in an error. This error does not stop the execution of the program, however, it changes the normal flow of the program.

Example:

Python3

```
# initialize the amount variable
marks = 10000
# perform division with 0
a = marks / 0
print(a)
```

Output:

```
Traceback (most recent call last):
  File "/home/f3ad05420ab851d4bd106ffb04229907.py", line 4, in <module>
    a=marks/0
ZeroDivisionError: division by zero
```

Embedded systems design combines the highly technical disciplines of hardware design and firmware and application software development. Embedded systems engineers face significant challenges throughout the design process, especially when it comes to the integration and debugging of hardware and software systems.

As an embedded systems project matures and grows in complexity, it becomes increasingly hard to track down and isolate bugs in the code. One study conducted in 2013 by researchers at the University of Cambridge identified that software developers spend up to 50% of their time and budget on each project debugging code. This amounts to billions of dollars each year in developer salaries and overhead, funds that could easily be allocated elsewhere if there was a more efficient way to debug your product.

To help bring relief to developers in debug purgatory, we're offering up some of our preferred debugging techniques that actually work. We'll review some of the traditional run-time debugging techniques, describe the benefits of integration testing, and explain how developers can implement real-time trace debugging to discover and rectify software bugs, bringing products to market faster and with fewer errors.

Traditional Debugging Techniques

Our exploration of debugging techniques for embedded systems begins with traditional debugging techniques. These techniques can all be used to identify and isolate coding errors for removal from your firmware or software application, but the efficacy of each method depends on the unique circumstances of your project. Some of these methods include automation while several others are heavily manual processes. More complex embedded systems projects will typically benefit from more sophisticated debugging methodologies.

Debugging Method #1: "Print Method"

Print debugging is probably the simplest and most basic way of debugging an embedded system. The method is carried out by watching live print statements that are written on the screen as the code is executed. To achieve this, the developer must intersperse print statements throughout the code that will be printed on the screen when specific portions of code are executed. In this way, it is possible to visualize when specific parts of the code are executed.

While the print method can provide transparency into how the code is behaving, it is not typically ideal for more complex projects. Printed statements may confirm that a specific piece of code has executed, but they fail to provide a complete view of the system state that would be useful (and sometimes required) for tracking down more elusive errors. While the print method is useful for simple applications and for developers who lack access to a complete debugging environment, a more technical solution is required to meet the needs of more complex embedded systems projects.

Debugging Method #2: "Run-time Methods"

The print debugging method is really just the simplest version of a run-time debugging technique. Run-time debugging methods share a common characteristic: they monitor the live execution of a process or a piece of code while the developer uses either manual techniques or debugging software to debug the process. In addition to the print method, run-time debugging methods include remote debugging and communication-based debugging.

Remote debugging describes a debugging technique where the process or code that is being debugged is executed in an environment that is separate from the debugger itself. Remote debugging is useful for embedded systems that do not have a full operating system implemented on them, making it easier for the developer

Integration Testing Technique

Integration testing, sometimes referred to as System Integration Testing (SIT), describes a fundamentally different approach to testing and debugging embedded systems throughout the development process. This method borrows wisdom from new working methods of software development such as Agile and DevOps that promote periodic testing throughout the development process instead of a single, lengthy debugging process in the late stages of product development.

The wisdom behind **integration testing** can be summarized as follows:

In traditional software development, engineers develop working specifications, program individual modules and combine the modules to complete the program before testing begins. This process may produce a large number of errors. The detection and isolation of errors is complicated by the fact that the code has already been integrated. It may be difficult to identify and isolate bugs once all code modules have already been integrated.

To improve on this method, integration testing begins with **unit testing**. Before code modules are integrated, they must be subjected to unit tests in isolation to ensure they operate bug-free on their own. Unit testing ensures that each software component functions as expected on its own before being integrated with the whole.

Once a module has passed unit testing, it can be integrated with other modules to begin developing a system. Developers can start by combining low-level modules that implement a common functionality and establishing test cases and procedures to verify their correct functioning. Developers must write a battery of test cases that will be used to verify the build is working correctly. When two or more modules are integrated successfully and pass all

the development process.

Debugging with Real-Time Trace

The final debugging technique that we want to mention here, and the most powerful, is known as a real-time trace debugging. With real-time trace, developers implement a hardware device such as a protocol analyzer that records information about the execution of processes in the code. This information can include sequences of program counter values, register reads and writes, and changes in device memory and associated data values.

When a developer notices that a process is behaving unexpectedly, real-time trace debugging provides complete and total insight into the functioning of the process within the code. The developer can produce a log that indicates exactly how the system changed while the process was executed, making it significantly easier to isolate, identify, and correct software bugs.

Real-time trace debugging can be implemented at any stage of development, as a complement to integration testing or when the software and hardware for the embedded system has been fully integrated. Trace information can also be captured non-intrusively, meaning that the system's timing and performance will not be affected by the trace. Real-time trace also enables developers to more easily capture, record and collect test results for future analysis of bugs.

Summary

While debugging can consume a large number of development resources, embedded systems engineers can choose to take advantage of debugging techniques and products that expedite the process and make it

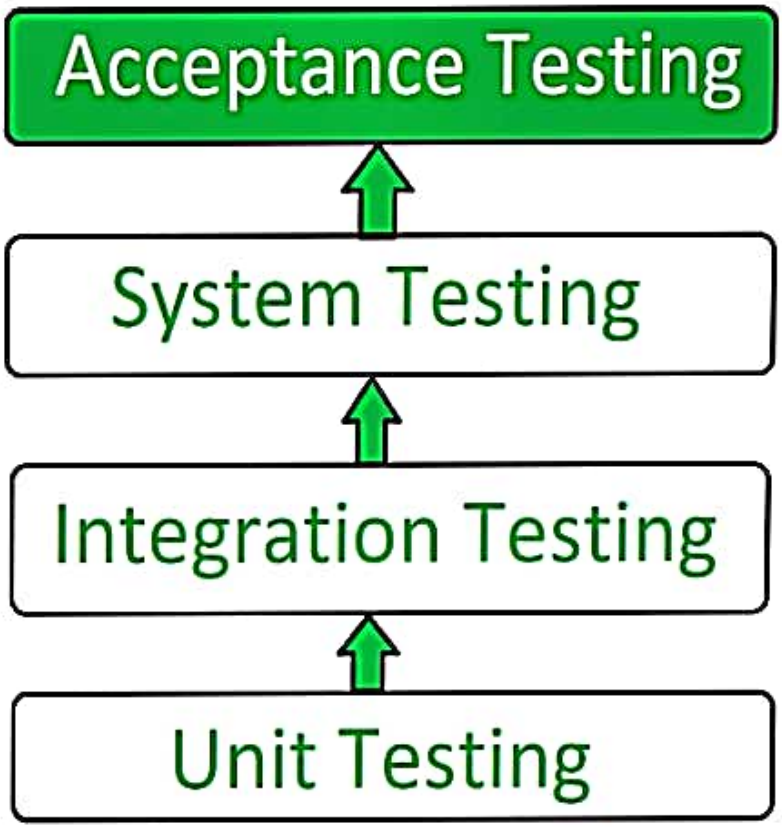
Acceptance Testing | Software Testing

Prerequisite – Software Testing | Basics, Types of Software Testing Acceptance Testing is a method of software testing where a system is tested for acceptability. The major aim of this test is to evaluate the compliance of the system with the business requirements and assess whether it is acceptable for delivery or not. **Standard Definition of Acceptance Testing:**

> It is a formal testing according to user needs, requirements and business processes conducted to determine whether a system satisfies the acceptance criteria or not and to enable the users, customers or other authorized entities to determine whether to accept the system or not.

Acceptance Testing is the last phase of software testing performed after System Testing and before making the system available for actual use.

Acceptance Testing is the last phase of software testing performed after System Testing and before making the system available for actual use.



Types of Acceptance Testing:

1. **User Acceptance Testing (UAT):** User acceptance testing is used to determine whether the product is working for the user correctly. Specific requirements which are quite often used by the customers are primarily picked for the