Beyond the Textbook (Zelle 4e - Chapter 6)

Decision Structures

Topics

- Manual Unit Testing
- Review Simple `if`
- Chaining Comparison Operators
- Truth Value Testing
- Review Two-Way `if`
- Review Multi-Way `if`
- Automated Unit Testing Using Pytest
- Using Multi-Way `if` For Lookups
- Using Nested `ifs` to Implement Complex Choices

Topics (Continued)

- Using `try` / `except` Blocks
- Using `raise` to Signal an Error Condition
- Using `ifs` For Multi-Faceted Validation
- Structural Pattern Matching
- Easier Formatting With Ternary `if`

Manual Unit Testing

- To this point in the course, we have been using manual unit testing.
- In manual unit testing, we develop a test plan. Then we execute each of the test cases in the plan manually and inspect the outputs.
- Manual unit testing can be quick and effective.
- Most people who do manual unit testing do not write down their test plan or keep documentation on their test results.
- Manual unit tests are difficult to re-run at a later time as the developer forgets details of the test cases and the expected results.
- Automated unit testing can have many advantages over this manual approach. We will discuss automated unit testing later in this lecture.

Review Simple `if`

- See: *_01_simple.py*

Chaining Comparison Operators

- In the proper circumstances, chaining multiple comparison operators can lead to more readable code.
- See: *_02_chaining_comparison_operators.py*

Truth Value Testing

- All Python variables may be tested for **truthiness** regardless of their type.
- Empty and zero values evaluate to `False`.
- Non-empty and non-zero values evaluate to `True`.
- Some Python programmers believe that this leads to more readable code.
- See Tutorial Article.
- See: *_03_truth_value_testing.py*

Review Two-Way `if`

- See: *_05_two_way.py*

Review Multi-Way `if`

- When constructing a multi-way `if` that uses **inequalities**, you must test conditions **in order**.
- Ascending order is preferred.
- See: *_10_multi_way.py*

Automated Unit Testing Using Pytest

- Automated unit testing is an important tool in modern software development.
- Using a unit testing framework, developers create piece of code to implement each test case.
- These test cases become an important asset that allow code to be tested and retested over it useful life.
- We will be using Pytest, one of several popular Python automated unit test frameworks.
- While Pytest has many sophisticated features, we will be using only the basic features.
- A modest investment in learning automated unit testing can have an enormous payback over time.

Using Multi-Way `if` For Lookups

- Inline lookups can be coded with a multi-way `if`.
- Refactoring the lookup into a function often leads to more readable code.
- Later in the course, we will learn how to do lookups using a Python `dictionary`.
- At this point in the course, we are learning how to do this without the `dictionary`.
- See:
  - *_25_lookup_in_function.py*
  - *test__25_lookup_in_function.py*

Using Nested `ifs` to Implement Complex Choices

- **Nested** `ifs` can be used to implement complex choices.
- Any code block can contain a simple, two-way, or multi-way `if`.
- Nesting `ifs` two levels deep is most common.
- Nesting `ifs` three levels deep is recommended only if it results in readable code.
- Nesting `ifs` more than three-levels deep is generally considered a bad practice.
- Refactoring a nested `if` into a function often leads to more readable and testable code.
- See:
  - *_35_nested_in_function.py*
  - *test__35_nested_ifs_in_function.py*

Using `try`/`except` Blocks

- `try`/`except` blocks allow recovery from anticipated program exceptions.
- Otherwise, exceptions cause a **stack trace** to print on the console and execution stops.
- The `try` block contains the code that might raise an exception.
- `except` blocks contain code that handles exceptions.
- The `finally` block allows for some code to run regardless of whether an exception was raised.
- See:
    - *_40_try.py*
    - *_43_try_with_called_code.py*

Using `raise` to Signal an Error Condition

- We can `raise` exceptions in our own code as a way of signalling error conditions.
- This architecture allows called code to detect errors and calling code to handle them.
- Exceptions are implemented with Python classes.
- When raising exceptions, we often re-use the builtin Python exception classes. See Python Documentation.
- It is possible to create our own exception class by creating a custom Python class. See tutorial article.
- See:
    - *_45_raise.py*
    - *test__45_raise.py*

Using `ifs` For Multi-Faceted Validation

- **Multi-faceted validation** can be implemented using a series of `if` statements.
- In this design pattern, we usually begin by assuming the the input is **valid**.
- Then, each facet is tested in turn.
- A failure of any one test, makes the input **invalid**.
- See:
    - *_75_validation_using_function.py*
    - *test__75_validation_using_function.py*
    - *_80_validation_using_function_and_messages.py*
    - *test__80_validation_using_function_and_messages.py*

Extra Python Features (Syntactic Sugar)

See https://en.wikipedia.org/wiki/Syntactic_sugar

Structural Pattern Matching

- This is a `switch` statement for Python. See Wikipedia article
- Feature added to Python in version 3.10.
- We are covering it here in its simplest form: a substitute for the multi-way `if`.
- It also introduces a **pattern matching** mechanism that is potentially much more powerful than the multi-way `if`. See tutorial in Python documentation.
- See:
    - *_92_lookup_in_function_using_structural_pattern_matching.py*
    - *test__92_lookup_in_function_using_structural_pattern_matching.py*

Easier Formatting With Ternary `if`

- Sometimes we want to format an output message that is slightly different depending upon data values.
- A classic example is when we want the message to include plural or singluar terms based upon data values.
- This is possible using the two-way `if`.
- But, it may be easier to code using the **ternary** `if`.
- See:
  - *_94_formatting_without_ternary_if.py*
  - *_96_formatting_with_ternary_if.py*

Last Revised 2025-08-04