



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

ACCA5036

Scripting and Programming Concepts

Week 8: Error Handling and Debugging

-Suresh

Trawsnewid Addysg; Trawsnewid Bywydau
Transforming Education; Transforming Lives

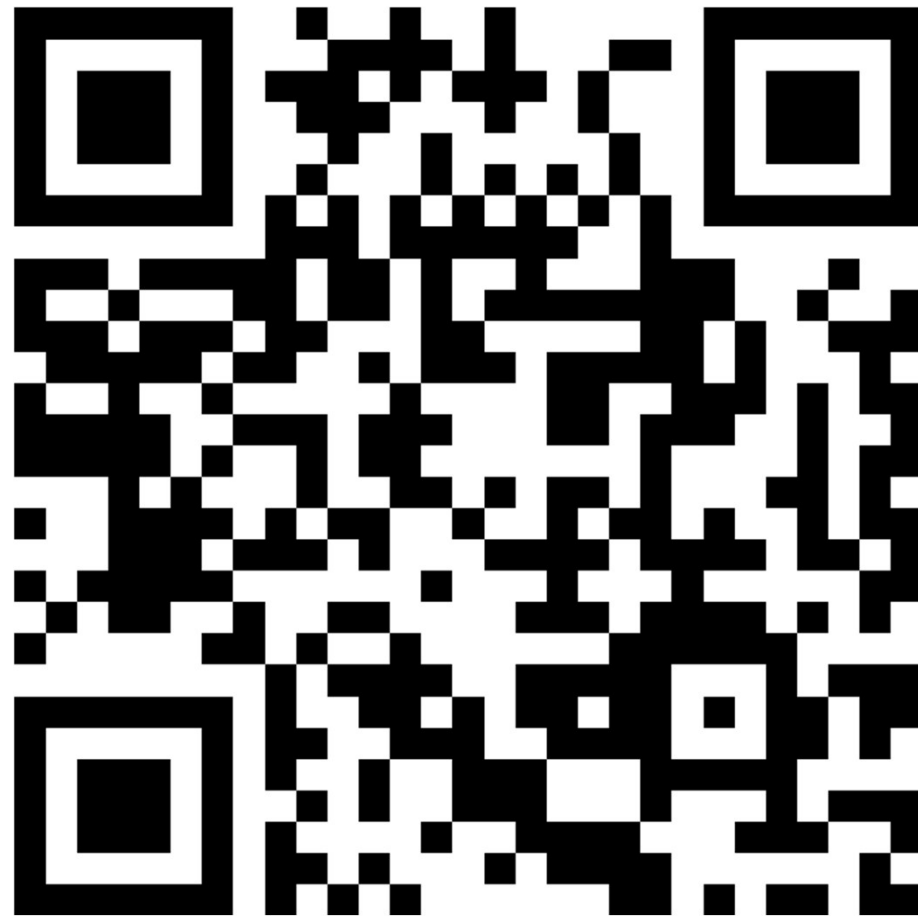


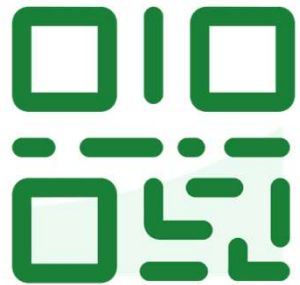
www.uwtsd.ac.uk

Lesson Objectives

- Functions in Python Recap
- Error Handling
- Common Python Errors
- Basic Error Handling
- Multiple Exception Handling
- Raising Exceptions
- Debugging Technique/Assertions

<https://app.sli.do/event/ntFmRpU5wWKPJwQ1DBp9cD>





**Join at slido.com
#4185467**

① The Slido app must be installed on every computer
you're presenting from

slido



What keyword is used to define a function in Python?

- ① The Slido app must be installed on every computer you're presenting from



What is the default return value of a function that does not explicitly return a value?

① The Slido app must be installed on every computer you're presenting from

slido



**Which of the following is
a valid way to call a
function named
'my_function'?**

- ① The Slido app must be installed on every computer you're presenting from



What is the correct syntax to define a function with no parameters?

① The Slido app must be installed on every computer you're presenting from



Which of the following is used to pass a variable number of arguments to a function?

- ① The Slido app must be installed on every computer you're presenting from



**How do you call a
function named
'my_function'?**

- ① The Slido app must be installed on every computer you're presenting from



What is the purpose of the 'return' statement in a function?

- ① The Slido app must be installed on every computer you're presenting from



What keyword is used to define a function in Python?

- ① The Slido app must be installed on every computer you're presenting from



How do you define a function with one parameter 'x'?

- ① The Slido app must be installed on every computer you're presenting from



What will be the output of the function call 'my_function(2)' if the function is defined as 'def my_function(x): return x * 2'?



**What does the following
Python function return:
def add(a, b): return a +
b?**



What will the function call `greet('Alice')` output if the function is defined as: `def greet(name): return 'Hello ' + name`?



**What does the following
function return: def
square(x): return x * x?**



What will the function call `is_even(4)` return if the function is defined as: `def is_even(n): return n % 2 == 0`?



**What does the following function
return: `def factorial(n): return 1 if n
== 0 else n * factorial(n-1)`?**



What is a function parameter?

- ① The Slido app must be installed on every computer you're presenting from

What will be the output of the following code?



```
def multiply(x, y=2):  
    return x * y
```

```
print(multiply(3))
```

What is wrong with the following function definition?



```
def my_func  
    print("Hello")
```

- ① The Slido app must be installed on every computer you're presenting from



Which is a valid lambda function?

- ① The Slido app must be installed on every computer you're presenting from



Which code uses a default parameter correctly?

- ① The Slido app must be installed on every computer you're presenting from

What is the output of the code?



```
x = 10  
def update():  
    x = 5
```

```
update()  
print(x)
```



What is a recursive function?

① The Slido app must be installed on every computer you're presenting from

What will happen if you call a function before defining it?



```
greet()  
def greet():  
    print("Hi!")
```

What is the error in this function call?



```
def add(a, b):  
    return a + b
```

```
print(add(5))
```

What is the error in this function?



```
def greet(name):  
    print("Hello", name)
```

What will happen if you define a function with the same name twice?



```
def hello():  
    print("Hello")
```

```
def hello():  
    print("Hi!")
```

```
hello()
```

What is the problem with this function definition?



```
def square(x):  
    print(x ** 2)
```

```
result = square(4)  
print(result)
```

Identify the issue in this function.



```
def func(a=5, b):  
    return (a + b)
```

```
result = func(3)  
print(result)
```

Error Handling

- Prevents Crashes
- Improves User Experience
- Allows Recovery



Error Handling

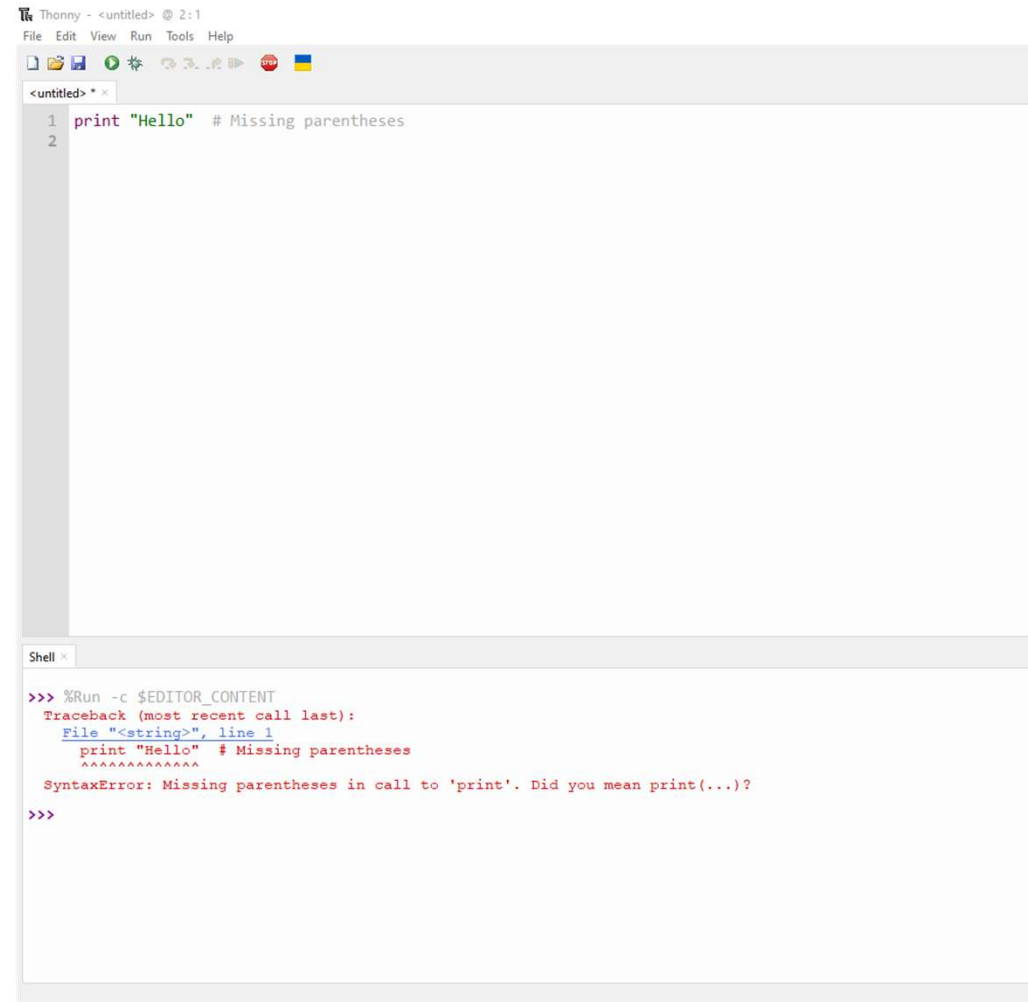
- The process of **anticipating**, **detecting**, and **resolving** programming, application, or communication errors.
- Prevents the program from crashing by gracefully handling **runtime errors**.
- Types of Errors:
 1. Syntax errors,
 2. Exceptions (runtime errors).
- Python uses exceptions to handle errors.



<https://images.app.goo.gl/WmsoWQVccRuBpFeZ9>

Types of Errors in Python

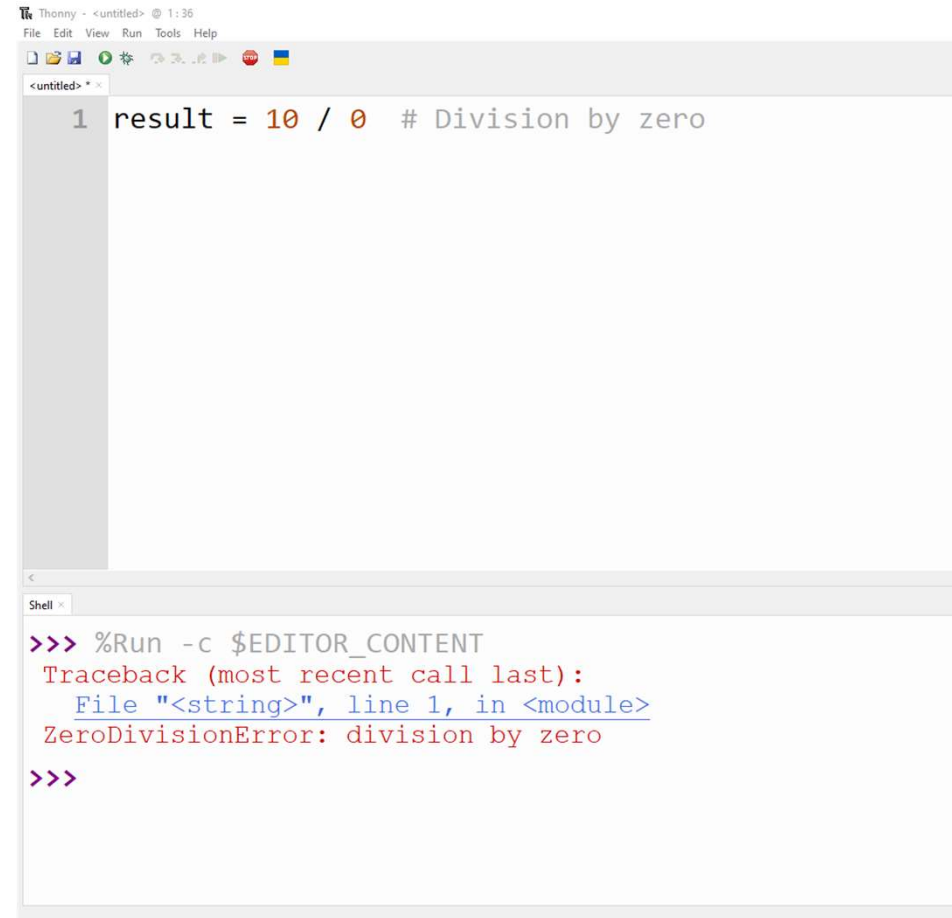
- **Syntax Errors:**
Problems in the code structure



The screenshot shows the Thonny IDE interface. The editor window displays two lines of Python code: `1 print "Hello" # Missing parentheses` and `2`. The shell window at the bottom shows the execution of the code, which results in a `SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?`. The error message is displayed in red text, and the shell prompt `>>>` is visible at the bottom.

Types of Errors in Python

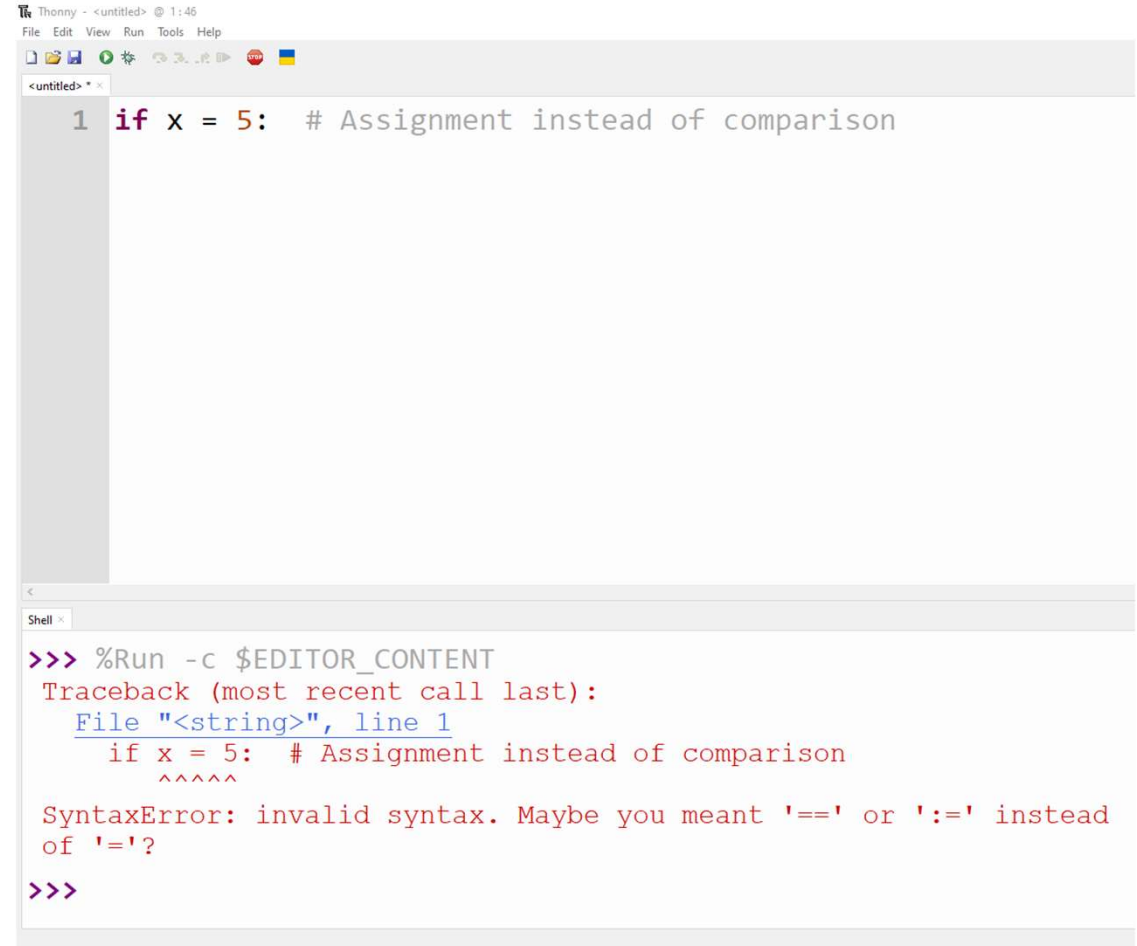
- **Runtime Errors:**
Errors that occur during execution.



The screenshot shows the Thonny Python IDE interface. The main editor window contains a single line of Python code: `1 result = 10 / 0 # Division by zero`. The number 10 is highlighted in orange, and the 0 is highlighted in red. Below the editor, the Shell window displays the output of running the code. It shows a prompt `>>>` followed by `%Run -c $EDITOR_CONTENT`, then a `Traceback (most recent call last):` message, followed by `File "<string>", line 1, in <module>`, and finally `ZeroDivisionError: division by zero`. The prompt `>>>` is shown again at the bottom.

Types of Errors in Python

- **Logical Errors:** The program runs but produces incorrect results due to logic mistakes.



The screenshot shows the Thonny IDE interface. The main editor window displays a single line of Python code: `1 if x = 5: # Assignment instead of comparison`. The code is highlighted in a light blue background. Below the editor, the Shell window shows the output of running the code. It starts with `>>> %Run -c $EDITOR_CONTENT`, followed by a `Traceback (most recent call last):` message. The traceback points to `File "<string>", line 1` and shows the code `if x = 5: # Assignment instead of comparison` with four red carets (`^^^^`) under the assignment operator. The final message is `SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?`. The Shell prompt `>>>` is visible at the bottom.

```
Thonny - <untitled> @ 1:46
File Edit View Run Tools Help

<untitled> *
1 if x = 5: # Assignment instead of comparison

Shell x
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<string>", line 1
    if x = 5: # Assignment instead of comparison
        ^^^^^
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead
of '='?
>>>
```

Common Python Errors

- **Syntax Errors:** Errors where the code is not valid Python (e.g., missing colon, unbalanced parentheses).
- **Exceptions:** Errors detected during execution (e.g., `IndexError`, `TypeError`, `ValueError`).

Syntax Error

```
for i in range(10)  
    print(i)
```

Syntax error
Missing Colon (:)

Correct code

```
for i in range(10):  
    print(i)
```

Name Error

```
print(age)
```

The variable **age** is not defined

Correct code

```
age = 25  
print(age)
```



Type Error

```
'a' + 2
```

Attempting to concatenate a string ('a') with an integer (2), which are incompatible types for this operation

Correct code

```
'a' + str(2) # Outputs: 'a2'
```

Index Error

```
numbers = [1, 2, 3]  
print(numbers[3])
```

Attempting to access an index that is out of range (index 3 where the last index is 2 in a list of three elements)

Correct code

```
print(numbers[2]) # Outputs: 3
```

Zero Division Error

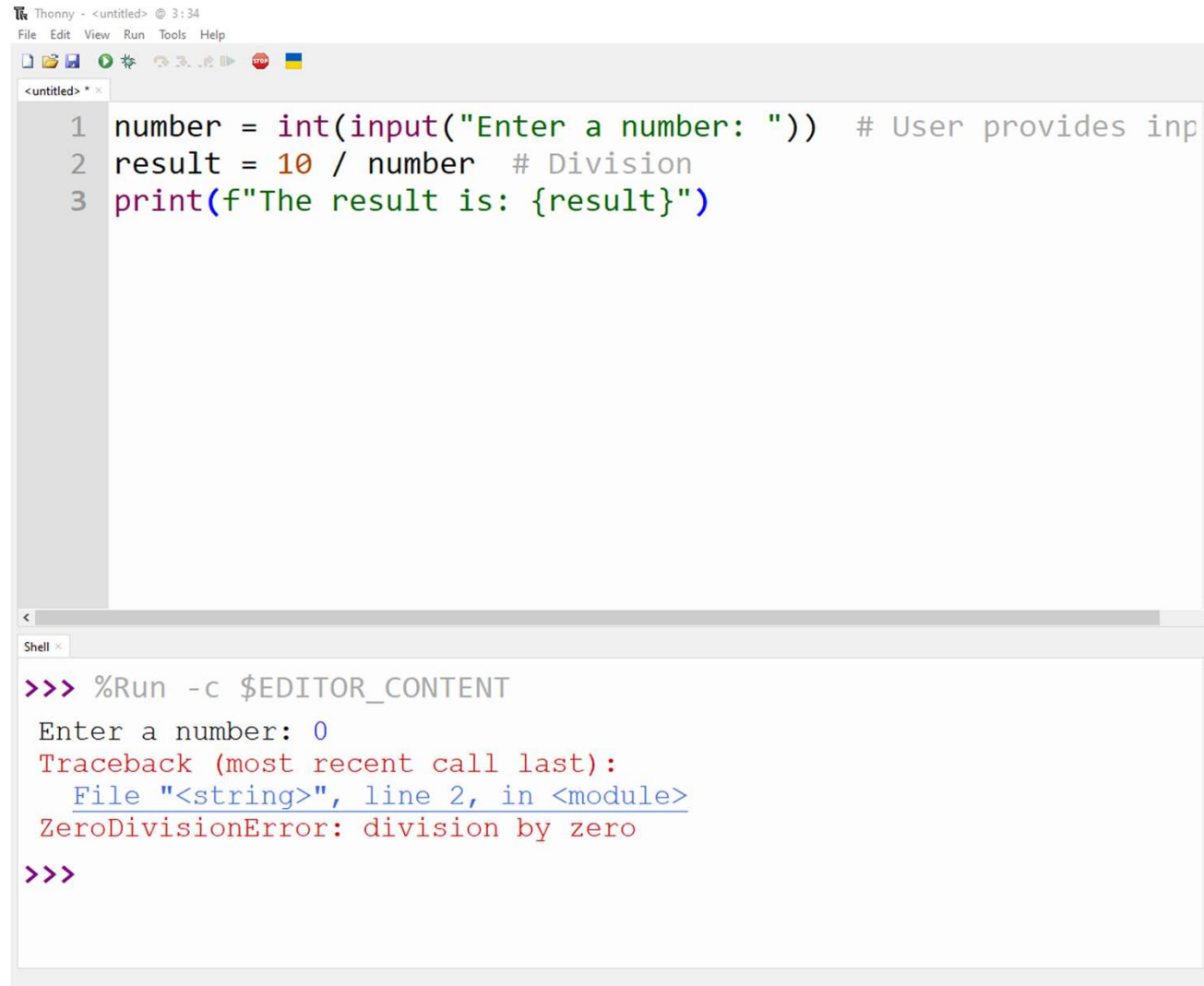
```
print(10 / 0)
```

Dividing a number by zero, which is mathematically undefined and not allowed in programming.

Correct code

```
denominator = 0
if denominator != 0:
    print(10 / denominator)
else:
    print("Cannot divide by zero.")
```

Code snippet without Error handling



The screenshot shows the Thonny IDE interface. The top pane contains a Python script in a file named '<untitled>'. The script consists of three lines: a comment, an input statement, a division operation, and a print statement. The bottom pane shows the execution output in a shell window, where the user has entered '0', leading to a ZeroDivisionError.

```
Thonny - <untitled> @ 3:34
File Edit View Run Tools Help

<untitled> * x
1 number = int(input("Enter a number: ")) # User provides inp
2 result = 10 / number # Division
3 print(f"The result is: {result}")

Shell x
>>> %Run -c $EDITOR_CONTENT
Enter a number: 0
Traceback (most recent call last):
  File "<string>", line 2, in <module>
ZeroDivisionError: division by zero
>>>
```

Basic Error Handling: **try** and **except**

- Syntax: **try** block contains code that might throw an exception, followed by one or more **except** blocks to handle specific exceptions.

```
try:
```

```
    # Code that might cause an error  
    codewithError()
```

```
except ExceptionType:
```

```
    # Code to handle the error  
    handle_error()
```

Basic Error Handling: **try** and **except**

ExceptionType(s)

ZeroDivisionError

FloatingPointError

try:

Code that might cause an error
codewithError()

except **ExceptionType:**

Code to handle the error
handle_error()

OverflowError

Basic Error Handling: **try** and **except**

ExceptionType(s)

```
try:  
    # Code that might cause an error  
    codewithError()  
except ExceptionType:  
    # Code to handle the error  
    handle_error()
```

IndexError

KeyError

Basic Error Handling: **try** and **except**

ExceptionType(s)

```
try:  
    # Code that might cause an error  
    codewithError()  
except ExceptionType:  
    # Code to handle the error  
    handle_error()
```

ValueError

TypeError

Basic Error Handling: **try** and **except**

ExceptionType(s)

```
try:  
    # Code that might cause an error  
    codewithError()  
except ExceptionType:  
    # Code to handle the error  
    handle_error()
```

AttributeError

NameError

Basic Error Handling: **try** and **except**

ExceptionType(s)

SyntaxError: Invalid Python syntax.

IndentationError: Incorrect indentation.

TabError: Mixing tabs and spaces in indentation.

RuntimeError: Generic error not covered by other exceptions.

NotImplementedError: A method or operation is not implemented.

RecursionError: Maximum recursion depth is exceeded.

```
try:  
    # Code that might cause an error  
    codewithError()  
except ExceptionType:  
    # Code to handle the error  
    handle_error()
```

Basic Error Handling: **try** and **except**

- Syntax: **try** block contains code that might throw an exception, followed by one or more **except** blocks to handle specific exceptions.

Example

```
try:
    number = int(input("Enter a number: "))
except ValueError:
    print("That was not a valid number.")
```

Code with no Error handling

```
number = int(input("Enter a number: "))  
result = 10 / number  
print(f"Result: {result}")
```

Code with Error handling

try:

```
number = int(input("Enter a number: "))
```

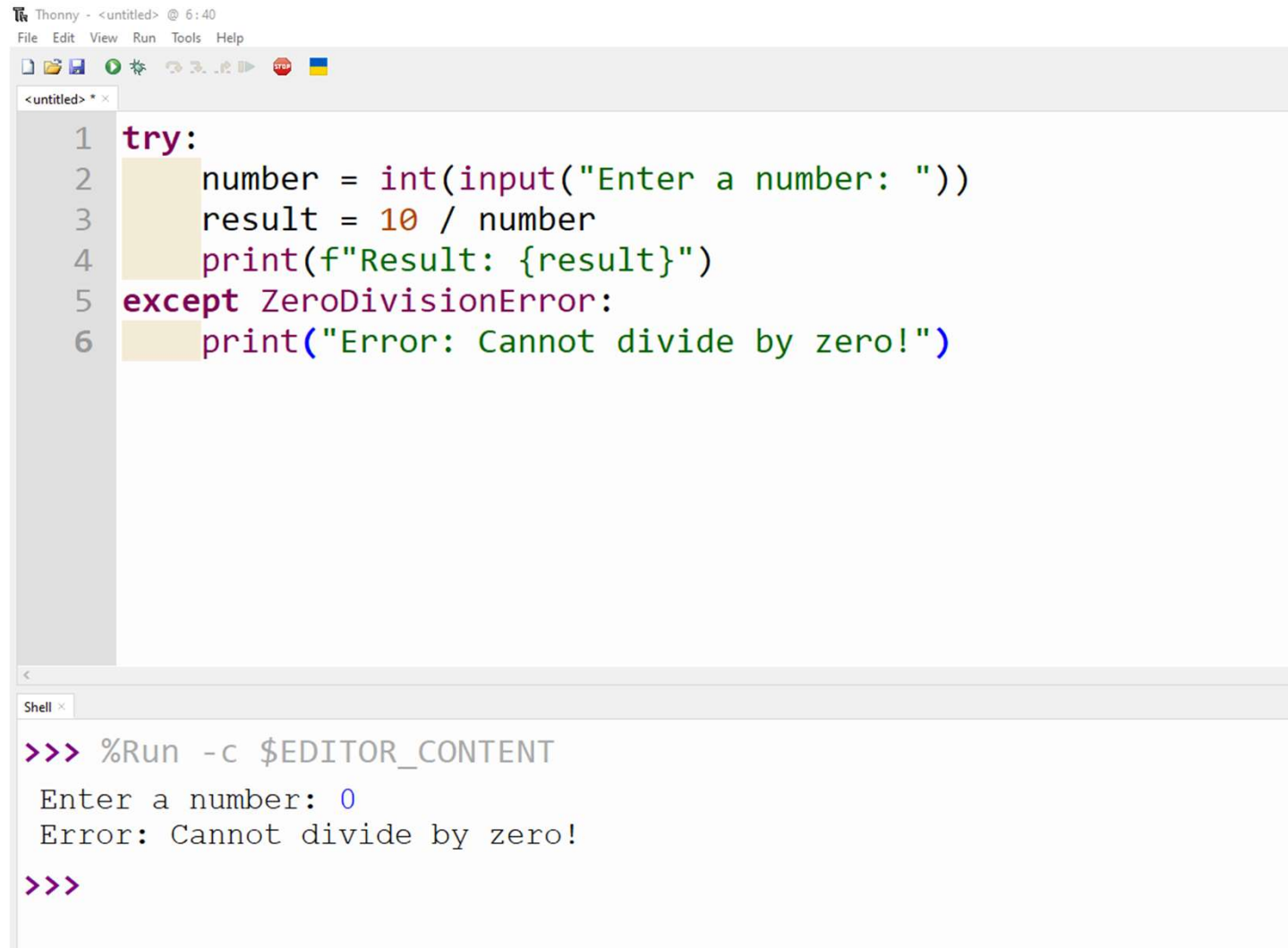
```
result = 10 / number
```

```
print(f"Result: {result}")
```

except ZeroDivisionError:

```
print("Error: Cannot divide by zero!")
```

Code with Error handling



The screenshot shows the Thonny Python IDE interface. The top menu bar includes 'File', 'Edit', 'View', 'Run', 'Tools', and 'Help'. Below the menu is a toolbar with icons for file operations and execution. The main editor window, titled '<untitled>', contains the following Python code:

```
1 try:
2     number = int(input("Enter a number: "))
3     result = 10 / number
4     print(f"Result: {result}")
5 except ZeroDivisionError:
6     print("Error: Cannot divide by zero!")
```

Below the editor is a 'Shell' window. It displays the output of running the code:

```
>>> %Run -c $EDITOR_CONTENT
Enter a number: 0
Error: Cannot divide by zero!
>>>
```

Multiple Exception Handling

- Handle different exceptions differently.
- **Syntax:** Multiple except clauses.

Example

```
try:
    # some code that can throw multiple exceptions
    result = 10 / int(input("Enter a divisor: "))
except ValueError:
    print("Please enter a valid integer.")
except ZeroDivisionError:
    print("A number cannot be divided by zero.")
```

Multiple Exception Handling

- Handle different exceptions differently.
- **Syntax:** Multiple except clauses.

try:

```
    number = int(input("Enter a number: "))  
    result = 10 / number  
    print(f"Result: {result}")
```

except ValueError:

```
    print("Error: Invalid input. Must be a number.")
```

except ZeroDivisionError:

```
    print("Error: Cannot divide by zero!")
```

Multiple Exception Handling

try:

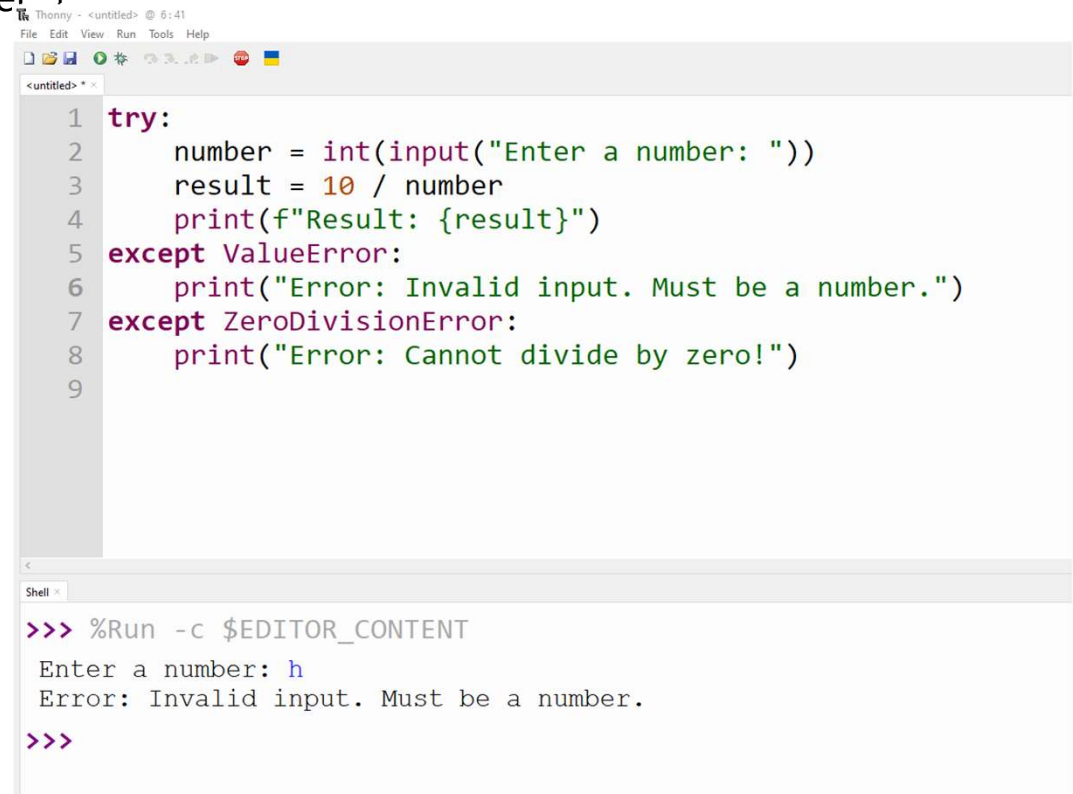
```
number = int(input("Enter a number: "))  
result = 10 / number  
print(f"Result: {result}")
```

except ValueError:

```
print("Error: Invalid input. Must be a number")
```

except ZeroDivisionError:

```
print("Error: Cannot divide by zero!")
```



The screenshot shows a Thonny IDE window with a Python script and its execution output in a shell. The script defines a try-except block to handle ValueError and ZeroDivisionError. The shell output shows the program running, prompting for input, and displaying an error message when the input 'h' is provided.

```
Thonny - <untitled> @ 6:41  
File Edit View Run Tools Help  
<untitled> *  
1 try:  
2     number = int(input("Enter a number: "))  
3     result = 10 / number  
4     print(f"Result: {result}")  
5 except ValueError:  
6     print("Error: Invalid input. Must be a number.")  
7 except ZeroDivisionError:  
8     print("Error: Cannot divide by zero!")  
9  
Shell  
>>> %Run -c $EDITOR_CONTENT  
Enter a number: h  
Error: Invalid input. Must be a number.  
>>>
```

else and finally

- **else Block:** Runs if the try block does not raise an exception.
- **finally Block:** Executes after all other blocks, always runs, used for clean-up activities.

Example

```
try:
    value = int(input("Enter a number: "))
except ValueError:
    print("You must enter an integer.")
else:
    print("The input was an integer.")
finally:
    print("This will execute no matter what.")
```

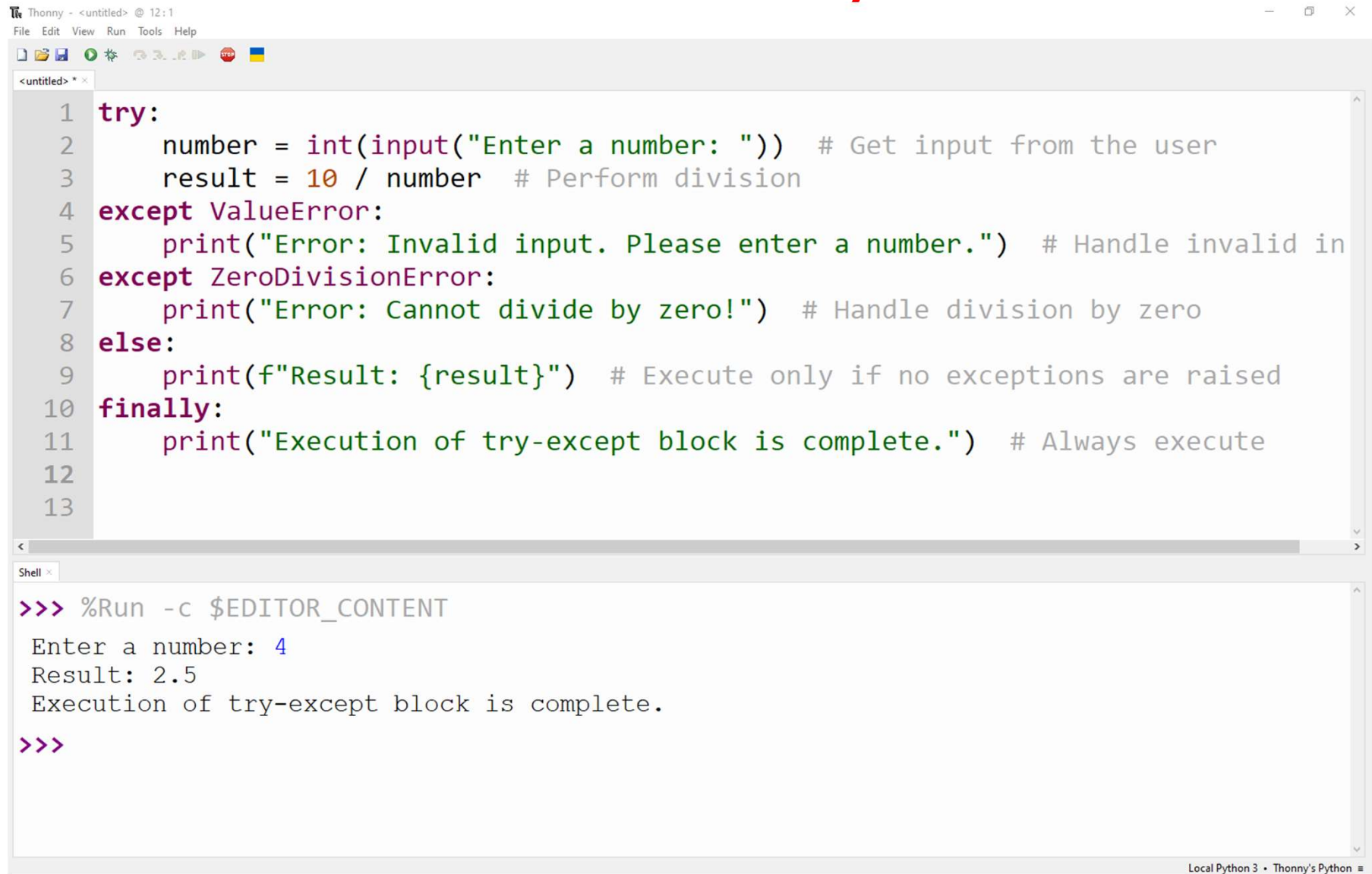
else and finally

- **else Block:** Runs if the try block does not raise an exception.
- **finally Block:** Executes after all other blocks, always runs, used for clean-up activities.

Example

```
try:
    value = int(input("Enter a number: "))
except ValueError:
    print("You must enter an integer.")
else:
    print("The input was an integer.")
finally:
    print("This will execute no matter what.")
```

else and finally



```
Thonny - <untitled> @ 12:1
File Edit View Run Tools Help

<untitled> * x
1 try:
2     number = int(input("Enter a number: ")) # Get input from the user
3     result = 10 / number # Perform division
4 except ValueError:
5     print("Error: Invalid input. Please enter a number.") # Handle invalid in
6 except ZeroDivisionError:
7     print("Error: Cannot divide by zero!") # Handle division by zero
8 else:
9     print(f"Result: {result}") # Execute only if no exceptions are raised
10 finally:
11     print("Execution of try-except block is complete.") # Always execute
12
13

Shell x
>>> %Run -c $EDITOR_CONTENT
Enter a number: 4
Result: 2.5
Execution of try-except block is complete.
>>>

Local Python 3 • Thonny's Python
```

Raising Exceptions

- Deliberately trigger an exception when a certain condition is met.
- **Syntax:** `raise ExceptionType("Message")`

Example

```
age = int(input("Enter your age: "))  
if age < 0:  
    raise ValueError("Age cannot be negative.")
```

Raising Exceptions

```
Thonny - <untitled> @ 14:74
File Edit View Run Tools Help

<untitled> * x
1 try:
2     number = int(input("Enter a number: ")) # Get input from the user
3     if number == 0:
4         # Manually raise a ValueError if the number is negative
5         raise ValueError("Error: The number cannot be negative.")
6     result = 10 / number # Perform division
7 except ValueError as ve:
8     print(ve) # Handle invalid input or negative number error
9 except ZeroDivisionError:
10    print("Error: Cannot divide by zero!") # Handle division by zero
11 else:
12    print(f"Result: {result}") # Execute only if no exceptions are raised
13 finally:
14    print("Execution of try-except block is complete.") # Always execute

Shell x
>>> %Run -c $EDITOR_CONTENT
Enter a number: 0
Error: The number cannot be negative.
Execution of try-except block is complete.
>>>

Local Python 3 • Thonny's Python
```

Debugging Techniques

- **Definition:** Process of identifying and removing errors from software applications.
- **Tools and Technique:** Use of print statements, debugging with IDEs (breakpoints, watch variables), and logging.
- **Tips for Effective Debugging:** Simplifying expressions, checking assumptions, and using external tools like PyLint or PyCharm.

Debugging Techniques in Thonny

Step Over (F6) → Executes the current line and moves to the next.

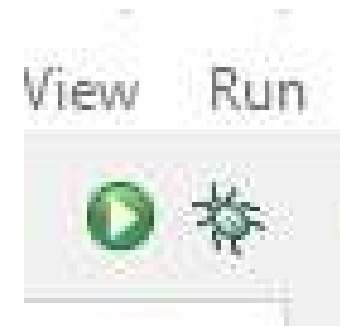
Step Into (F7) → Goes inside functions to debug them line by line.

Step Out (Shift+F7) → Exits the function and returns to the main script.

Debug Current script

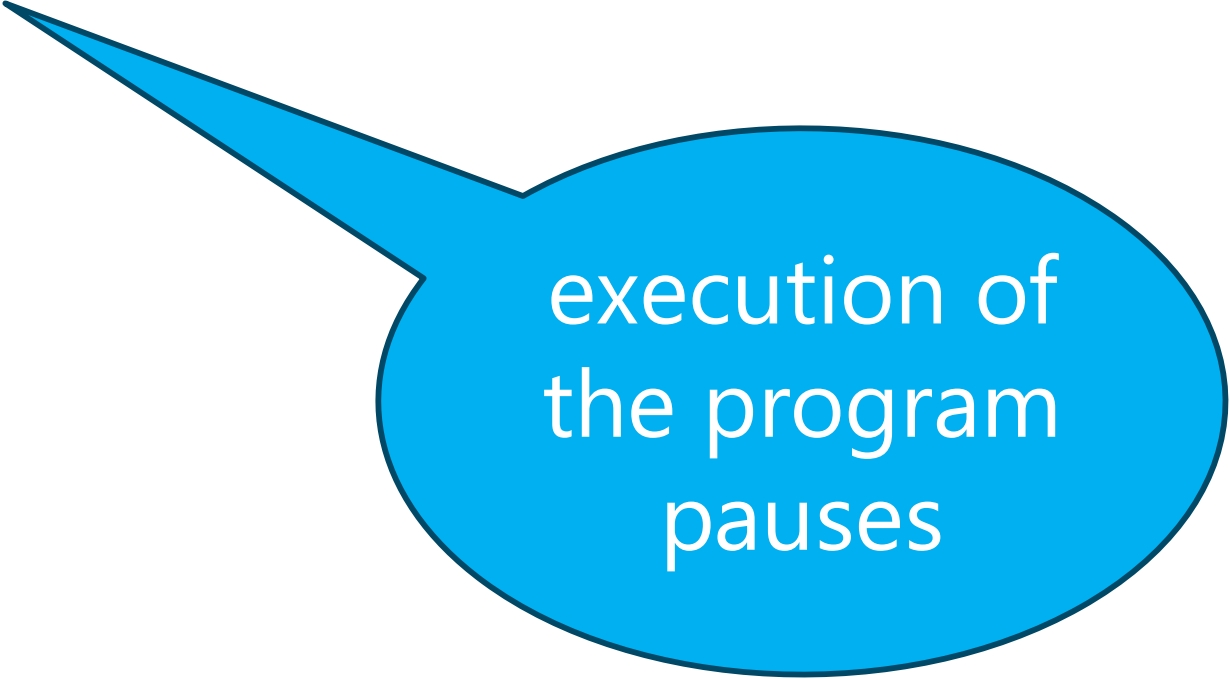
```
def add_numbers(a, b):  
    result = a + b #  
    print("Result:", result)  
    return result
```

```
x = 5  
y = 10  
sum_value = add_numbers(x, y)  
print("Sum:", sum_value)
```



Debugging Techniques in Thonny

- Setting Breakpoints



execution of
the program
pauses

Setting Breakpoints - Example

```
def add_numbers(a, b):  
    result = a + b #  
    print("Addition Result:", result)  
    return result
```

```
def subtract_numbers(a, b):  
    result = a - b #  Set a breakpoint here  
    print("Subtraction Result:", result)  
    return result
```

```
x = 10
```

```
y = 5
```

```
sum_value = add_numbers(x, y) # Call add_numbers  
diff_value = subtract_numbers(x, y) # Call subtract_numbers
```

```
print("Final Results:", sum_value, diff_value)
```

Setting Breakpoints - Example

Steps to Debug:

Click on the left margin next to `result = a - b` to set the breakpoint.

Click "Debug current script" (Ctrl+F5).

The script should pause at the breakpoint.

Use Step Over (F6) or Step Into (F7) to continue debugging.

Assertion

- What is an Assertion?
 - An assert statement checks if a condition is True.
 - If the condition is False, it raises an AssertionError with an optional error message.
 - Syntax: assert condition, "Error message"

Assertion: Example 1: Validating Inputs

Scenario: A function to calculate the area of a rectangle.

Assertion Use: Ensure width and height are positive.

```
def calculate_area(width, height):  
    assert width > 0 and height > 0, "Width and height must be  
    positive!"  
    return width * height
```

Test cases

```
print(calculate_area(5, 3)) # Valid (returns 15)
```

```
print(calculate_area(-2, 4)) # Triggers AssertionError
```

Assertion: Example 2: Checking Output Validity

Scenario: A function to compute the average of a list.

Assertion Use: Ensure the result is within a valid range.

```
def compute_average(numbers):  
    assert len(numbers) > 0, "List must not be empty"  
    average = sum(numbers) / len(numbers)  
    assert 0 <= average <= 100, "Average must be between 0 and 100"  
    return average
```

Test cases

```
print(compute_average([80, 90, 70])) # Valid (returns 80.0)  
print(compute_average([])) # Triggers AssertionError (empty list)
```

Example 3: Data Validation

Scenario: A function to process user data.

Assertion Use: Validate username format.

```
def register_user(username):  
    assert isinstance(username, str), "Username must be a string"  
    assert len(username) >= 3, "Username must be at least 3 characters long"  
    assert username.isalnum(), "Username must be alphanumeric"  
    return f"User {username} registered!"  
  
# Test cases  
print(register_user("Alice123")) # Valid  
print(register_user("A"))        # Triggers AssertionError (too short)
```

Common Use Cases for Assertions

- 1. Input Validation:** Check if inputs meet expected criteria (e.g., positive numbers).
- 2. Output Validation:** Verify results are within logical bounds.
- 3. Invariants:** Ensure critical assumptions hold true during execution (e.g., a list is never empty).
- 4. Data Integrity:** Validate data formats (e.g., strings, numeric ranges).

Activity: List Element Retrieval and Manipulation

Example outputs

Enter an index to retrieve from the list (0-4): 2
Element at index 2 is 30.
Doubling the value: 60
Operation complete. Thank you for using the list retrieval program.

Enter an index to retrieve from the list (0-4): 7
Error: Index out of range. Please choose between 0 and 4.
Operation complete. Thank you for using the list retrieval program.

Enter an index to retrieve from the list (0-4): abc
Error: You must enter an integer.
Operation complete. Thank you for using the list retrieval program.

Activity: List Element Retrieval and Manipulation

Code

```
my_list = [10, 20, 30, 40, 50]

try:

    index = int(input("Enter an index to retrieve from the list (0-4):
    "))
    value = my_list[index]
except ValueError:
    print("Error: You must enter an integer.")
except IndexError:
    print("Error: Index out of range. Please choose between 0 and 4.")
else:
    print(f"Element at index {index} is {value}.")
    print(f"Doubling the value: {value * 2}")
finally:
    print("Operation complete. Thank you for using the list retrieval
    program.")
```

Activity: List Element Retrieval and Manipulation

Objective:

Create a program that:

1. Prompts the user to input an index to retrieve a value from a list.
2. Handles potential errors gracefully (e.g., invalid index, non-integer input).
3. Executes a special operation if everything works correctly.
4. Ensures cleanup or a final message regardless of errors.

Exercise



<https://forms.office.com/e/dg2d0J2Tn0>

Guidelines for Assessment : Simple Library Management System

- **Overview of the Task**
- **Scenario:** Develop a prototype for a **Library Management System** using Python.
- **Objective:** Replace the current manual system with a digital solution for managing book records, searches, and other functionalities.

Key Requirements

- **Programming Language:** Python.
- **Data Structures:** Use lists, arrays, etc. (no database implementation required).
- **Functionality:**
 - Add book records.
 - Search for books.
 - Perform other meaningful library operations.

Development Steps

Analysis:

- Use abstraction and problem decomposition.
- Identify required features (e.g., add/search books).

Design:

- Develop algorithms using flowcharts and pseudocode.
- Select appropriate variables and data structures.

Development Steps

Implementation:

- Code a user-friendly solution with validation for user inputs.
- Use user-defined functions.

Testing and Debugging:

- Create and apply test cases.
- Provide screenshots and commentary for validation under various conditions (e.g., correct/incorrect inputs).

Documentation:

- Detail the design and implementation process.

Report Structure

- **Title Page:** Include name, student number, course, module, and submission date.
- **Code Repository Link:** Provide a functional GitHub or other version control link.
- **Table of Contents:** List all sections with page numbers.
- **Sections to Include:**
 1. **Analysis (500 words):** Define the problem, analyze requirements.
 2. **Software Design & Development:**
 - Flowcharts and pseudocode for core functionalities.
 - Data structure and variable choices with justifications.
 3. **Testing & Debugging Report:**
 - Explain testing importance.
 - Document test cases with results (screenshots and comments).
 4. **Evaluation:** Reflect on challenges, propose improvements, and self-reflection.
 5. **References and Appendices:** Use IEEE format.

Submission Details

- **Deadline** : **Friday, 25th April 2025 by 2 PM.**
- **Submission Platform** : Turnitin.
- **Late Penalty:**
 - **Up to 1 week late** : Maximum mark of 40%.
 - **More than 1 week late** : 0 marks
(unless approved by Extenuating Circumstances Panel).



Additional Notes

- **Academic Honesty:** Cite all sources and provide a reference list.
- **Code Requirements:**
 - Well-organized, readable, and properly commented.
 - Copy code into the report (not screenshots).