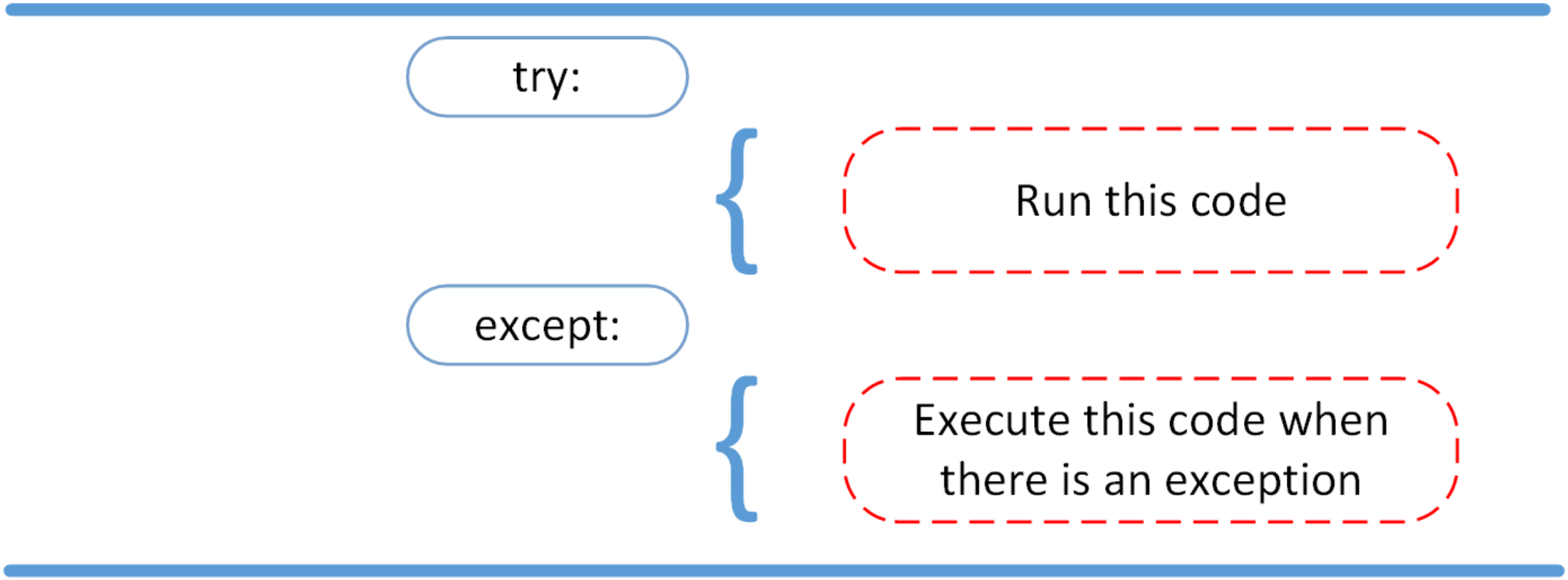
Handling run-time errors

Week 9

Exception handling

- Python provides a mechanism for handling and catching errors, known as **exception handling**.
- This allows you to manage unexpected errors and prevent your program from crashing.
- The core of exception handling in Python is the `try-except` block:
  - **try block**: This block contains the code that might raise an exception.
  - **except block**: This block is executed if an exception (=error) occurs within the try block.

Handling Exceptions: `try + except:`



File: `Example1.py`

`except Exception as e`

- When an error occurs, Python creates an **exception object** that contains information about the error, such as its type and message.
- The **type** of the exception object indicates the kind of error that occurred. For example, `ZeroDivisionError`, `ValueError`, and `TypeError`.
- The exception object usually contains a **message** that provides more details about the error. This message can be accessed and printed to help understand what went wrong.

`except Exception as e`

- In Python, `except Exception as e` is used to catch exceptions and assign the caught exception to a variable (`e` in this case).
- This allows you to access the **exception object** and its attributes within the except

File: `Example2.py`

Catching different types of errors

- To catch **different types** of errors in Python, you can use multiple except blocks, each handling a specific type of exception.
- This allows you to provide different handling logic for different errors.

File: `Example3.py`

- `ValueError` is raised when the input cannot be converted to an integer.
- `ZeroDivisionError` is raised when you try to divide a number by zero.
- `KeyboardInterrupt` does not display a full traceback like other exceptions. This is raised when user presses `ctrl+C`.