Exception Handling

The try block contains the code that may raise an exception.

Each except block handles a specific exception type.

You can have multiple except blocks to handle different exception types.

The first matching block will be executed. You can specify the exception type after except, e.g., except ValueError:. If you don't specify a particular exception type, it will catch all exceptions.

You can assign the exception instance to a variable using as. For example, except ValueError as e:.

The else block is executed if no exceptions occur in the try block.

The finally block is always executed, regardless of whether an exception occurred or not. It is typically used to clean up resources.

You can handle multiple exception types in a single except block by enclosing them in parentheses.

The Exception class is the base class for all built-in exceptions. Handling Exception will catch any exception that hasn't been handled by other except blocks.

Handling a specific exception:

In [ ]:
```
try:
    # Code that might raise a ValueError
    ...
except ValueError:
    # Handle the ValueError exception
    ...
```

Handling multiple specific exceptions:

In [ ]:
```
try:
    # Code that might raise a ValueError or TypeError
    ...
except (ValueError, TypeError):
    # Handle the ValueError or TypeError exception
    ...
```

Handling multiple specific exceptions with different handling logic:

In [ ]:
```
try:
    # Code that might raise a ValueError or TypeError
    ...
except ValueError:
    # Handle the ValueError exception
    ...
```

```python
except TypeError:
    # Handle the TypeError exception
    ...
```

Handling any exception:

In [ ]:
```python
try:
    # Code that might raise any exception
    ...
except Exception:
    # Handle any exception
    ...
```

Handling any exception with exception information:

In [ ]:
```python
try:
    # Code that might raise any exception
    ...
except Exception as e:
    # Handle any exception and print the exception information
    print("Exception:", str(e))
```

Handling specific exceptions and any other exception:

In [ ]:
```python
try:
    # Code that might raise a ValueError, TypeError, or any other exception
    ...
except ValueError:
    # Handle the ValueError exception
    ...
except TypeError:
    # Handle the TypeError exception
    ...
except Exception as e:
    # Handle any other exception
    print("Exception:", str(e))
```

Handling exceptions in a loop:

In [ ]:
```python
for item in items:
    try:
        # Code that might raise an exception
        ...
    except Exception:
        # Handle the exception
        ...
```

Handling exceptions and continuing the loop:

In [ ]:
```python
for item in items:
    try:
        # Code that might raise an exception
        ...
    except Exception:
        # Handle the exception
        ...
        continue
    # Code to execute if no exception occurred
    ...
```

Handling exceptions and breaking out of the loop:

In [ ]:
```
for item in items:
    try:
        # Code that might raise an exception
        ...
    except Exception:
        # Handle the exception
        ...
        break
    # Code to execute if no exception occurred
    ...
```

Raising an exception:

In [ ]:
```
try:
    # Code that might raise an exception
    ...
    if condition:
        raise ValueError("Invalid value")
except ValueError as e:
    # Handle the raised ValueError
    print("Exception:", str(e))
```

Reraising an exception:

In [ ]:
```
try:
    # Code that might raise an exception
    ...
except Exception as e:
    # Handle the exception
    ...
    raise
```

Handling exceptions with an else block:

In [ ]:
```
try:
    # Code that might raise an exception
    ...
except Exception:
    # Handle the exception
    ...
else:
    # Code to execute if no exception occurred
    ...
```

Handling exceptions with a finally block:

In [ ]:
```
try:
    # Code that might raise an exception
    ...
except Exception:
    # Handle the exception
    ...
finally:
    # Code to execute regardless of exceptions
    ...
```

Using multiple levels of exception handling:

In [ ]:
```
try:
    # Code that might raise an exception
    ...
except ValueError:
    # Handle the ValueError exception
    ...
    try:
        # Code that might raise another exception
        ...
    except AnotherException:
        # Handle the AnotherException exception
        ...
    finally:
        # Code to execute regardless of exceptions in the inner try-except block
        ...
except Exception:
    # Handle any other exception
    ...
```

Handling exceptions raised by built-in functions:

In [ ]:
```
try:
    result = int("123")
except ValueError:
    # Handle the ValueError exception
    ...
```

Handling exceptions raised by external libraries:

In [ ]:
```
try:
    import external_library
except ImportError:
    # Handle the ImportError exception
    ...
```

Handling exceptions with a custom error message:

In [ ]:
```
try:
    # Code that might raise an exception
    ...
except ValueError as e:
    # Handle the ValueError exception and print a custom error message
    print("ValueError occurred:", str(e))
```

Handling exceptions with multiple except blocks for the same exception type:

In [ ]:
```
try:
    # Code that might raise a ValueError
    ...
except ValueError as e:
    # Handle the ValueError exception with specific handling logic
    ...
except ValueError as e:
    # Handle the ValueError exception with different handling logic
    ...
```

Handling exceptions with a base exception class:

In [ ]:
```
try:
    # Code that might raise any exception
    ...
except Exception:
    # Handle any exception using the base Exception class
    ...
```

Handling exceptions in a function and propagating the exception:

In [ ]:
```
def some_function():
    try:
        # Code that might raise an exception
        ...
    except Exception:
        # Handle the exception
        ...
        raise

try:
    some_function()
except Exception:
    # Handle the propagated exception
    ...
```

Handling exceptions with traceback information:

In [ ]:
```
import traceback

try:
    # Code that might raise an exception
    ...
except Exception as e:
    # Handle the exception and print the traceback information
    traceback.print_exc()
```

Handling exceptions with specific error codes:

In [ ]:
```
try:
    # Code that might raise an exception
    ...
except Exception as e:
    if str(e) == "Error Code 123":
        # Handle the specific error code
        ...
    else:
        # Handle any other exception
        ...
```

Handling exceptions without any handling logic:

In [ ]:
```
try:
    # Code that might raise an exception
    ...
```

```
except:
    # No specific handling logic, just suppress the exception
    pass
```

Handling exceptions and logging the error:

In [ ]:
```
import logging

try:
    # Code that might raise an exception
    ...
except Exception as e:
    # Handle the exception and log the error
    logging.error("Exception occurred: %s", str(e))
```

Handling exceptions and executing cleanup code in the finally block:

In [ ]:
```
try:
    file = open("data.txt", "r")
    try:
        # Code that might raise an exception
        ...
    except Exception:
        # Handle the exception
        ...
    finally:
        # Cleanup code to always close the file
        file.close()
except FileNotFoundError:
    # Handle the FileNotFoundError exception
    ...
```