**Sharif University of Technology**
**Department of Computer Engineering**

# Fundamentals of Programming

Python Language

**Arman Malekzadeh**
**PhD Candidate in Artificial Intelligence**

## Table of contents

## Exception Handling

## Exceptions in Python

- An exception is an error that happens during execution of a program.
- When that error occurs, Python generate an exception that can be handled, which avoids your program to crash.
- Python has many built-in exceptions that are raised when your program encounters an error (something in the program goes wrong).
- When these exceptions occur, it causes the current process to stop and passes it to the calling process until it is handled.

## Exceptions in Python

- If not handled, our program will crash.
- Handling exceptions that occur at runtime is very important, because we don’t want our program to crash.
- When an exception occurs, the program immediately stops.
- The program execution will not continue until the exception is handled.

## Exceptions in Python

- The try block lets you test a block of code for errors.
- The except block lets you handle the error.
- The finally block lets you execute code, regardless of the result of the try- and except blocks.

## Expection Handling in Python: Example

- The try block will generate an exception, because x is not defined:

```python
try:
    print(x)
except:
    print("An exception occurred")
```

## Expection Handling in Python: Advanced Example

- The try block will generate an exception, because x is not defined:

```python
try:
    print(x)
except NameError:
    print("Variable x is not defined")
except:
    print("Something else went wrong")
```

## Types of Exceptions in Python

- Python has many built-in exceptions that are raised when your program encounters an error (something in the program goes wrong).
- When these exceptions occur, it causes the current process to stop and passes it to the calling process until it is handled.
- Some of the common exceptions are:
  - ImportError: an import fails;
  - IndexError: a list is indexed with an out-of-range number;
  - NameError: an unknown variable is used;
  - SyntaxError: the code can't be parsed properly;
  - TypeError: a function is called on a value of an inappropriate type;
  - ValueError: a function is called on a value of the correct type, but with an inappropriate value.

## Raise Statement in Python

- The raise statement allows the programmer to force a specified exception to occur.
- The sole argument in raise indicates the exception to be raised. This must be either an exception instance or an exception class (a class that derives from Exception).
- If an exception class is passed, it will be implicitly instantiated by calling its constructor with no arguments.

## Raise Statement in Python: Example

- Raise a TypeError exception:

```
    if not type(x) is int:
        raise TypeError("This is a TypeError")
```

## The Finally Clause in Python

- The finally clause is optional. However, use of finally is recommended to prevent resource leaks.
- The finally clause always executes when the try block exits. This ensures that the finally clause is executed even if an unexpected exception occurs.
- But finally is useful for more than just exception handling — it allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break.

## The Finally Clause in Python: Example

```python
try:
    print("Hello")
except:
    print("Something went wrong")
finally:
    print("The ’try except’ is finished")
```

## The Finally Clause in Python: Advanced Example

```
try:
    f = open("demofile.txt")
    f.write("Lorum Ipsum")
except:
    print("Something went wrong when writing to the file")
finally:
    f.close()
```

## Silent Failure in Python

- The silent failure is a common mistake in exception handling.
- The silent failure occurs when we don’t handle the exception and we don’t raise it.

```python
try:
    print(x)
except:
    pass
```

## Custom Exceptions in Python

- Python has many built-in exceptions that are raised when your program encounters an error (something in the program goes wrong).
- When these exceptions occur, it causes the current process to stop and passes it to the calling process until it is handled.
- But sometimes we need to create our own exceptions.

## Custom Exceptions in Python: Example

- Create a custom exception:

```
class MyError(Exception):
    def __init__(self, message):
        self.message = message
```

## Exceptions in Python: Else Clause

- In some situations, you might want to run a certain block of code if the code block inside try ran without any errors.
- For these cases, you can use the optional else keyword with the try statement.
- The else block must follow all except blocks that follow the try.
- Any exceptions raised in the else block will be caught by the preceding except blocks.

## Else Clause in Python Exceptions: Example

```
try:
    print("Hello")
except:
    print("Something went wrong")
else:
    print("Nothing went wrong")
```

## Python Exceptions: Error Logging

- Logging is a means of tracking events that happen when some software runs.
- The software's developer adds logging calls to their code to indicate that certain events have occurred.
- An event is described by a descriptive message which can optionally contain variable data (i.e. data that is potentially different for each occurrence of the event).
- Events also have an importance which the developer ascribes to the event; the importance can also be called the level or severity.

## Python Exceptions: Error Logging Example

```
import logging

logging.basicConfig(filename='example.log', level=logging.DEBUG)
logging.debug('This message should go to the log file')
logging.info('So should this')
logging.warning('And this, too')
```

## Python Logging Types

- There are five standard levels indicating the severity of events. Each has a corresponding method that can be used to log events at that level of severity. The defined levels, in order of increasing severity, are the following:
  - DEBUG: Detailed information, typically of interest only when diagnosing problems.
  - INFO: Confirmation that things are working as expected.
  - WARNING: An indication that something unexpected happened, or indicative of some problem in the near future (e.g. ‘disk space low’). The software is still working as expected.
  - ERROR: Due to a more serious problem, the software has not been able to perform some function.
  - CRITICAL: A serious error, indicating that the program itself may be unable to continue running.

## References

## References I

[1] B Downey, A. (2015). Think Python: How to Think Like a Computer Scientist-2nd Edition.

[2] Deitel, H. M., & Deitel, P. J. (2004). C: How to program. Pearson Educacion.

Sharif University of Technology
Department of Computer Engineering

Arman Malekzadeh

Fundamentals of Programming
Python Language