LECTURER: TAI LE QUY

Introduction to Programming with Python

Modules and Packages

UNIT 5

Errors and Exceptions

STUDY GOALS

- Know to how interpret error messages and trace the errors to their root cause
- Learn about exception handling and how to implement it
- Use logs to trace program workflow

EXPLAIN SIMPLY

1. What are the three types of errors?
2. What is the basic construct of exception handling in Python?
3. What are the five different levels of logs?

Types of error

An error is a location in the code at which the code does not perform as expected. There are three types of errors: syntax error, exception and logical error.

Error type	Description
Syntax error	Error due to code not conforming with syntax rules. Syntax errors can prevent an application from running.
Exception	Error due to incorrect arithmetic operations, type mismatch or unexpected operations. Exceptions, if not caught, will prevent the program from proceeding.
Logical error	Error due to incorrect programming logic. Logical errors do not stop the program execution and are therefore hard detect.

Most common syntax errors

Error	Incorrect	Corrected
Invalid name	`1st_name = 'Jane'`	`name_1st = 'Jane'`
Incorrect indentation	`if x > 3:` `print(x)`	`if x > 3:` `    print(x)`
Missing colon :	`if x > 3`	`if x > 3:`
Missing keyword	`add(x, y):` `    return x + y`	`def add(x, y):` `    return x + y`
Misspelt keyword	`fi x > 3:`	`if x > 3:`
Invalid expression	`if x = 0:`	`if x == 0:`

Wrong arguments	`def add(x, y):`
	`    return x + y`
	`add(3)`
		`def add(x, y):`
		`    return x + y`
		`add(3, 4)`

Most common exceptions

Error	Description
ZeroDivisionError	A number is divided by zero.
TypeError	Operation is inappropriate for the data type.
NameError	Variable name is not defined.
IndexError	Array or list index is not in the sequence.
KeyError	The key is not found in a dictionary.
RuntimeError	An error occurs during execution.
OverflowError	An arithmetic operation exceeds the predefined limits e.g., integer value that is too big.
FileNotFoundError	The file does not exist.
ImportError	The imported module is not found.

Basic construction

In Python, exceptions are handled by using the `try except` code blocks. The construction is as follows:

```
try:
    <codes to check for exception>
except <error type>:
    <codes when exception is encountered>
```

When an error is encountered the remaining codes within the `try` block are skipped. The `except` block will be executed.

The `except` block will be ignored if there is no error.

Example:

```
try:
    value = input('Type an integer: ')
    result = str(5 / int(value))
    print('5 divided by ' + value)
except ZeroDivisionError:
    print('You tried to divide by zero!')
```

When `value` is zero the next line is skipped. The `except` block will be executed.

Source of the text: Lutz, 2013.

Basic construct (cont.)

There can be more than one `except` clauses and the error type need not be specified (in which case it captures all the errors that have not been explicitly specified):

Example:

```
try:
    value = input('Type an integer: ')
    result = str(5 / int(value))
    print('5 divided by ' + value)
except ZeroDivisionError:
    print('You tried to divide by zero!')
 except:
    print('An error has occurred!')
```

When `value` is not a number
This `except` block will be executed.

`else` clause

The `else` clause will be executed when no exception is encountered. If exception is detected, the `else` block will be ignored.

Example:

```
try:
    value = input('Type an integer: ')
    result = str(5 / int(value))
    print('5 divided by ' + value)
except ZeroDivisionError:
    print('You tried to divide by zero!’)
else:
    print('The result is ' + result)
```

If no exception is encountered. This `else` block will be executed. Otherwise, it will be ignored.

`finally` clause

The `finally` clause will be executed regardless of whether there is an exception or not. This is useful for cleaning up operations.

Example:

```
try:
    value = input('Type an integer: ')
    result = str(5 / int(value))
    print('5 divided by ' + value)
except ZeroDivisionError:
    print('You tried to divide by zero!’)
else:
    print('The result is ' + result)
finally:
    result = ''
    print('The try except is completed.')
```

`finally` does a reset before exiting the `try except` block.

Debugging using logs

Logical errors are errors due to incorrect programming logic. The process of identifying and fixing the cause of a logical error is called debugging.

To debug, one uses logs as they enable the operations of a program to be displayed at runtime. A discrepancy between the program operations and their expected behaviour indicates a logical error.

To use log, one imports the Python library called `logging` and configures it with the logging parameters (see later):

```
import logging
logging.basicConfig(<configuration parameters>)
```

Types of log

One can set five levels of logs to indicate the severity level to be tracked.

Logging type	Description
logging.debug	Lowest level of severity. Useful for diagnosing problems.
logging.info	Second level of severity. Useful for confirming the expected behaviour.
logging.warning	Third level of severity. Useful for issuing a warning regarding a particular runtime event – something unexpected happens.
logging.error	Fourth level of severity. Useful for catching a specific error.
logging.critical	Highest level of severity. Useful for indicating serious errors which can prevent the program from running.

Logging configurations

```
logging.basicConfig(<configuration parameters>)
```

```
<configuration parameters>
```

```
level=logging.DEBUG
```

any logging equal to or more severe than debug will be shown.

```
filename=mylog.log', filemode='w', level=logging.DEBUG
```

saving the logs to a file called `mylog.log`.

```
format='%(asctime)s: %(message)s'
```

add date time to the logs.

REVIEW STUDY GOALS

- Know to how interpret error messages and trace the errors to their root cause
- Learn about exception handling and how to implement it
- Use logs to trace program workflow

SESSION 5

TRANSFER TASK

1. Inspect the following code and identify the errors and their types?

```
length = input('Length: ')
width = input('Width: ')
def area_rect(length)
      $area = length * width

$area = area_rect(length, width)
print('Area of rectangle is ', $area
```

2. Use `try except` to do the following: open a file called personnels.csv. If the file does not exist, create the file with file name personnels.csv and insert the header ‘Name, Age, Position’. If the file exists, append the file with ‘Jane, 20, Senior manager’, 'John, 20, Developer’ and ‘Frank, 32, CFO’. The output should look like this when opened in Excel:

Name	Age	Position
Jane	25	Senior manager
John	20	Developer
Frank	32	CFO

TRANSFER TASK
PRESENTATION OF THE RESULTS

Please present your results.

The results will be discussed in plenary.

1. What is the difference between a syntax error and an exception?

2. Some code is being written where an exception may occur. The exception should be handled properly so the application will not crash. The specific exception that may occur is the ZeroDivisionError exception. Regardless of whether or not the exception occurs, there’s some cleanup code that needs to run. What should be done?

3. Which of the following will set my logging level to CRITICAL?
    a. logging.basicConfig = logging.CRITICAL
    b. logging.basicConfig(level = logging.CRITICAL)
    c. logging.basicConfig.level = logging.CRITICAL
    d. logging. level = logging.CRITICAL

LIST OF SOURCES

Lutz, M. (2013). Introducing Python Object Types. *Learning Python* (5th ed.). O'Reilly.