Computational Thinking with Programming

Lecture - 16

Exception Handling

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

@cse_bennett @csebennett

Today…

Last Session:

- Sequences and Strings

Today's Session:

- Error and Exceptions:
  - Fundamentals
  - Exception Handling

Hands on Session with Jupyter Notebook:

- We will practice on the exception handling in Jupyter Notebook.

Syntax Errors

- Syntax errors, also known as parsing errors, are perhaps the most common kind of error you encounter while you are still learning Python.
- The parser repeats the offending line and displays a little ‘arrow’ pointing at the earliest point in the line where the error was detected. The error is detected at the token *preceding* the arrow. File name and line number are printed so you know where to look in case the input came from a script.

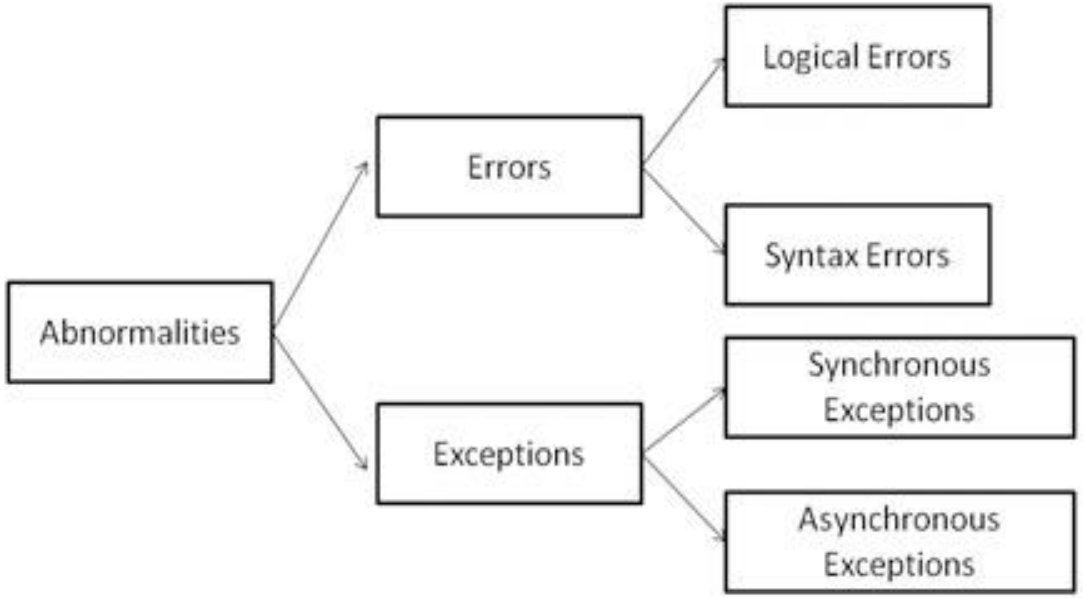
```
In [13]: run debug_example.py
------------------------------------------------------------
   File "debug_example.py", line 1
     def hello()
                ^
SyntaxError: invalid syntax

WARNING: Failure executing file: <debug_example.py>

In [14]:
```

Exceptions

- Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution are called *exceptions* and are not unconditionally fatal.
- Most exceptions are not handled by programs, however, and result in error messages like “cannot divide by zero” or “cannot concatenate ‘str’ and ‘int’ objects”.



Exceptions (cont'd.)

Exception: error that occurs while a program is running

- Usually causes program to abruptly halt

Traceback: error message that gives information regarding line numbers that caused the exception

- Indicates the type of exception and brief description of the error that caused exception to be raised

Exceptions (cont'd.)

Many exceptions can be prevented by careful coding

- Example: input validation
- Usually involve a simple decision construct

Some exceptions cannot be avoided by careful coding

- Examples
  - Trying to convert non-numeric string to an integer
  - Trying to open for reading a file that doesn't exist

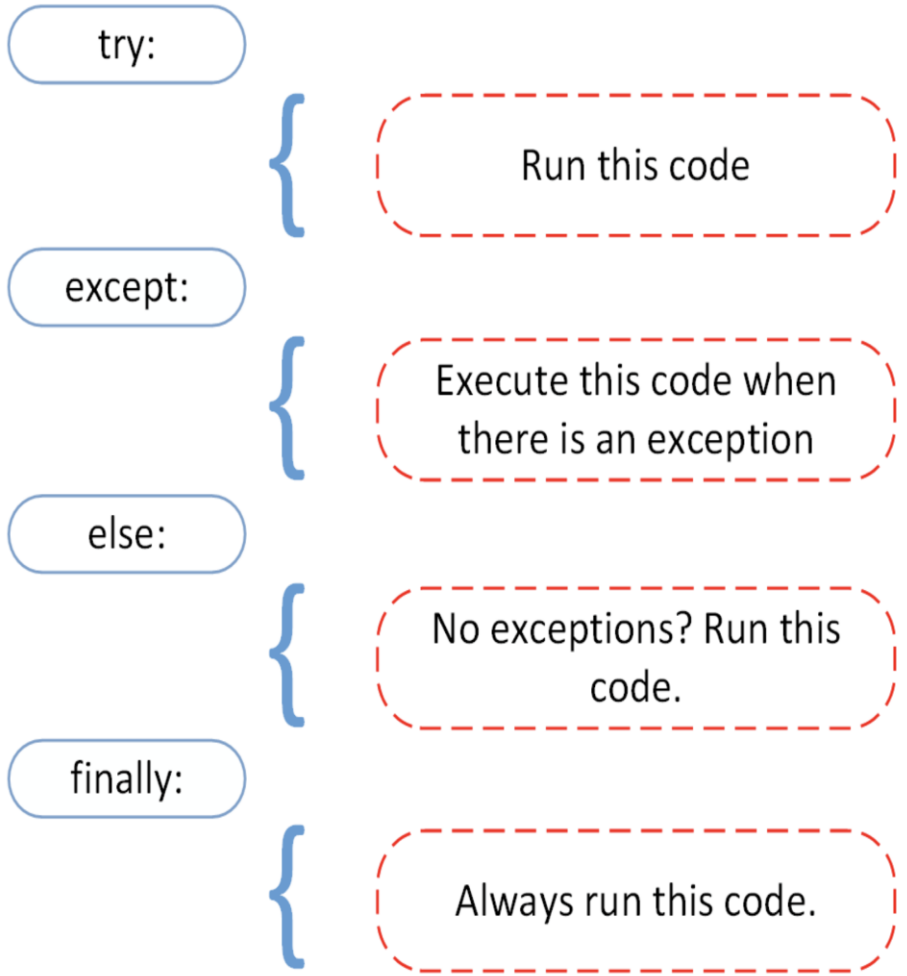
Exceptions (cont'd.)

- Exception handler: code that responds when exceptions are raised and prevents program from crashing
  - In Python, written as `try/except` statement
    - General format:

```
try:
    statements
except exceptionName:
    statements
```

    - Try suite: statements that can potentially raise an exception
    - Handler: statements contained in `except` block

Exceptions (cont'd.)



Example

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

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

Print one message if the try block raises a `NameError` and another for other errors:

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

Exceptions (cont'd.)

If statement in try suite raises exception:

- Exception specified in except clause:
  - Handler immediately following except clause executes
  - Continue program after try/except statement
- Other exceptions:
  - Program halts with traceback error message

If no exception is raised, handlers are skipped

Handling Multiple Exceptions

- Often code in try suite can throw more than one type of exception
  - Need to write `except` clause for each type of exception that needs to be handled
- An `except` clause that does not list a specific exception will handle any exception that is raised in the try suite
  - Should always be last in a series of `except` clauses

Displaying an Exception's Default Error Message

- Exception object: object created in memory when an exception is thrown
  - Usually contains default error message pertaining to the exception
  - Can assign the exception object to a variable in an `except` clause
    - Example: `except ValueError as err:`
  - Can pass exception object variable to `print` function to display the default error message

The `else` Clause

- `try/except` statement may include an optional `else` clause, which appears after all the `except` clauses
  - Aligned with `try` and `except` clauses
  - Syntax similar to `else` clause in decision structure
  - Else suite: block of statements executed after statements in try suite, only if no exceptions were raised
    - If exception was raised, the else suite is skipped

The `finally` Clause

- `try/except` statement may include an optional `finally` clause, which appears after all the `except` clauses
  - Aligned with `try` and `except` clauses
  - General format: `finally:`

    ```
        statements
    ```

  - Finally suite: block of statements after the `finally` clause
    - Execute whether an exception occurs or not
    - Purpose is to perform cleanup before exiting

Example

Use of else:

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

Use of finally:

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

Try to open and write to a file that is not writable:

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

Raise an Exception: Example

- As a Python developer you can choose to throw an exception if a condition occurs.
- To throw (or raise) an exception, use the raise keyword.

Example: Raise an error and stop the program if x is lower than 0:

```
x = -1
if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

- You can define what kind of error to raise, and the text to print to the user.

Example: Raise a TypeError if x is not an integer:

```
x = "hello"
if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

Raise an Exception: Example2

Example: Raise an error and stop the program if x is lower than 0:

```
try:
        raise NameError('HiThere’)
except NameError: ('An exception flew by!’)

except:
        print(“All Exception handled”)
```

What If an Exception Is Not Handled?

Two ways for exception to go unhandled:

- No except clause specifying exception of the right type
- Exception raised outside a try suite

In both cases, exception will cause the program to halt

- Python documentation provides information about exceptions that can be raised by different functions

Thank You

?