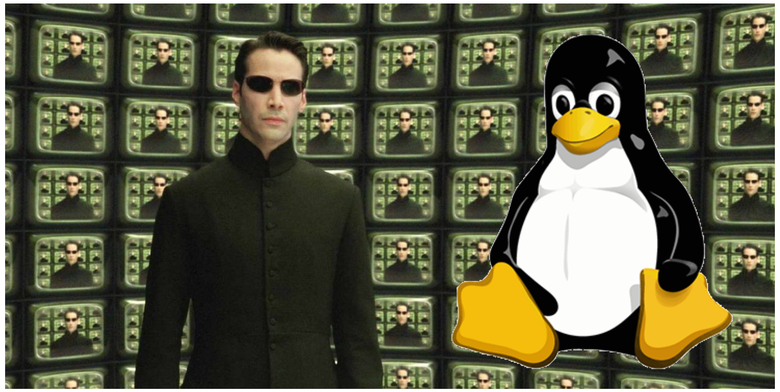
02.11.21 15:12

CSCI 120
Introduction to CompSci and Programming I
Lecture 6: Error Handling



Jetic Gū

Overview

- Focus: Python Programming
- Architecture: von Neumann
- Core Ideas:
    1. Error Messages
    2. Handling Errors with Exceptions

Python Error Messages

Summary

Syntax Error

- Also called parser error: the interpreter doesn’t understand your code
- Most common mistakes
  - Indentation errors
  - Incorrect bracket usage

Concept

Runtime Error

- Errors that occur in run time
  - The code is syntactically correct
  - The code caused an error when the interpreter attempts to execute it
  - You need to learn to interpret the error and resolve them

```
a = [1, 2, 3, 4, 5]
for i in range(a):
    print(a)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'list' object cannot be interpreted as an integer
>>>
```

Concept

Runtime Error

- Errors that occur in run time
  - The code is syntactically correct
  - The code caused an error when the interpreter attempts to execute it
  - You need to learn to interpret the error and resolve them

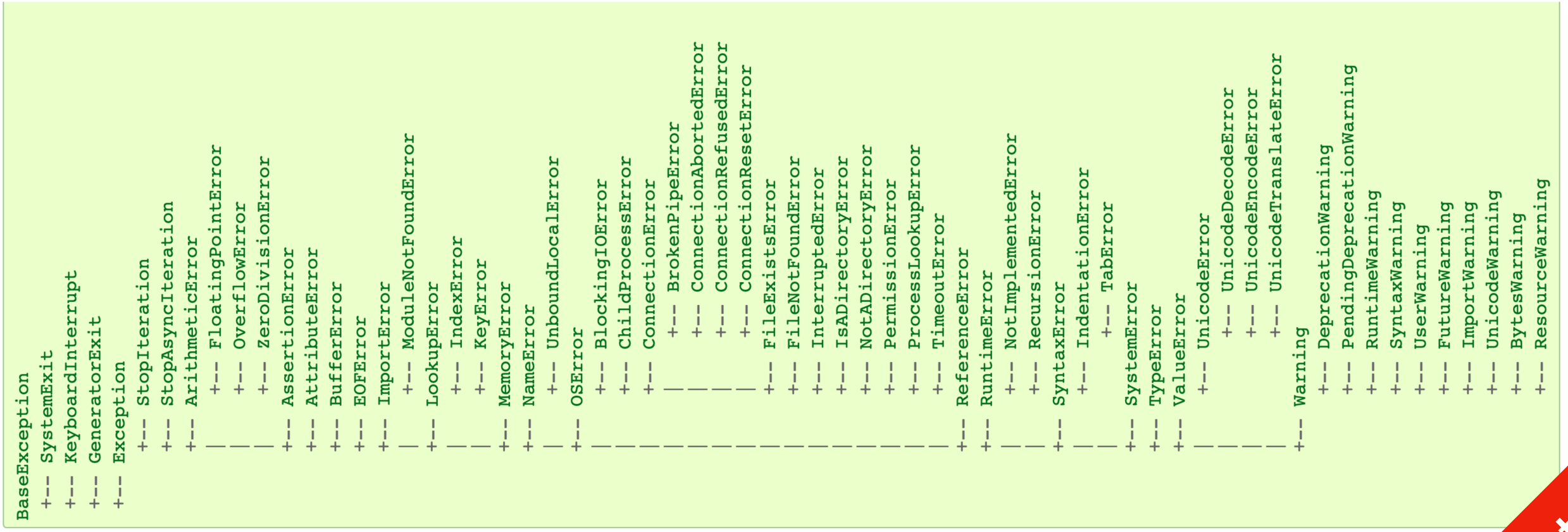
```
a = [1, 2, 3, 4, 5]
for i in range(a):
    print(a)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'list' object cannot be interpreted as an integer
>>>
```

Concept

P1
Error Messages

Common Errors



Concept

1. https://docs.python.org/3/library/exceptions.html

Common Errors

- `ImportError`
- `ZeroDivisionError`
- `IndexError`
- `KeyError`
- `TypeError`
- `ValueError`

- These are only some selected errors, there are more
- They are all called `Exceptions` in Python
- `Exceptions` are essentially `Class`es as well, and there are inheritance relations
  We will talk about it later

Concept

1. https://docs.python.org/3/library/exceptions.html

P1
Error Messages

Common Errors

- `ImportError`
- `ZeroDivisionError`
- `IndexError`
- `KeyError`
- `TypeError`
- `ValueError`

- Raised when the `import` statement fails
  - attempting to import a `module` that has not been installed
  - attempting to import with incorrect name, etc...
  - e.g.
    `import defautdict`
  -
```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'defauldict'
```
  - problem: wrong spelling of the module
  - fix: `import defaultdict`

Technical

Common Errors

- `ImportError`
- `ZeroDivisionError`
- `IndexError`
- `KeyError`
- `TypeError`
- `ValueError`

- Raised when the second argument of a division or modulo operation is zero
  - subclass of `ArithmeticError`
  - e.g.
    `print(5 / (5-5))`
  -
```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```
  - problem: trying to divide 5 by 0
  - fix: fix your math

P1
Error Messages

Common Errors

- `ImportError`
- `ZeroDivisionError`
- `IndexError`
- `KeyError`
- `TypeError`
- `ValueError`

- Raised when index used on a mapping or sequence is invalid (common in Arrays)
  - subclass of `LookupError`
  - e.g.

```
a = [1, 2, 3, 4, 5]
print(a[5])
```

  -

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

  - problem: array index exceeded array size
  - fix: use if statement to check boundary

Technical

P1
Error Messages

Common Errors

- `ImportError`
- `ZeroDivisionError`
- `IndexError`
- `KeyError`
- `TypeError`
- `ValueError`

- Raised when key used on a mapping (dictionary) is not found in the set of existing keys.
  - subclass of `LookupError`
  - e.g.
    ```
    a = {'a': 1, 'b': 2}
    print(a['z'])
    ```
  -
    ```
    Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
    KeyError: 'z'
    ```
  - problem: key `'z'` is not found in this `dict` object
  - fix: use if statement to check key's existence
    ```
    if key in a:
    ```

Technical

P1
Error Messages

Common Errors

- `ImportError`
- `ZeroDivisionError`
- `IndexError`
- `KeyError`
- `TypeError`
- `ValueError`

- Raised when an object does not support attribute references or attribute assignments at all
  - Operation or function is applied to an object of inappropriate type
  - e.g.

```
a = [1, 2, 3, 4, 5]
for i in range(a):
    print(i)
```

  -

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'list' object cannot be interpreted as an integer
```

  - problem: `range()` is expecting an `integer` as parameter, instead it got a `list`
  - Fix: figure out exactly what you want here then apply fix

Technical

P1
Error Messages

Common Errors

- `ImportError`
- `ZeroDivisionError`
- `IndexError`
- `KeyError`
- `TypeError`
- `ValueError`

- Raised when an operation or function receives an argument that has the right type but an inappropriate value
  - Operation or function is applied to an object of inappropriate type
  - e.g.
    ```
    import math
    math.sqrt(-10)
    ```
  - ```
    Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
    ValueError: math domain error
    ```
  - problem: `math.sqrt(n)` performs square root, for which `n` must be a natural number
  - Fix: fix your math

Technical
    ```

P1
Error Messages

OJ Errors

- **AC** - Accepted
- **WA** - Wrong Answer
- **IR** - Invalid Return
  - Contains python exception types
- **TLE** - Time Limit Exceeded
- **MLE** - Memory Limit Exceeded
- **OLE** - Output Limit Exceeded

Execution Results

Test case #1: IR (ValueError) [0,052s, 9.06 MB] (0/1)
Test case #2: IR (ValueError) [0,033s, 9.06 MB] (0/1)
Test case #3: IR (ValueError) [0,050s, 9.06 MB] (0/1)
Test case #4: IR (ValueError) [0,051s, 9.06 MB] (0/1)
Test case #5: IR (ValueError) [0,033s, 9.06 MB] (0/1)
Test case #6: IR (ValueError) [0,050s, 9.06 MB] (0/1)
Test case #7: IR (ValueError) [0,051s, 9.06 MB] (0/1)
Test case #8: IR (ValueError) [0,034s, 9.06 MB] (0/1)
Test case #9: IR (ValueError) [0,050s, 9.06 MB] (0/1)
Test case #10: IR (ValueError) [0,051s, 9.06 MB] (0/1)
Test case #11: IR (ValueError) [0,051s, 9.06 MB] (0/1)
Test case #12: IR (ValueError) [0,051s, 9.06 MB] (0/1)
Test case #13: IR (ValueError) [0,049s, 9.06 MB] (0/1)
Test case #14: IR (ValueError) [0,050s, 9.06 MB] (0/1)
Test case #15: IR (ValueError) [0,050s, 9.06 MB] (0/1)
Test case #16: IR (ValueError) [0,049s, 9.06 MB] (0/1)
Test case #17: IR (ValueError) [0,052s, 9.06 MB] (0/1)
Test case #18: IR (ValueError) [0,050s, 9.06 MB] (0/1)
Test case #19: IR (ValueError) [0,054s, 9.06 MB] (0/1)
Test case #20: IR (ValueError) [0,050s, 9.06 MB] (0/1)

Concept

Python Error Handling

Summary

Handling exceptions

- By default, the interpreter handles exceptions by stopping the program and printing an error message

- However, we can override this behaviour by ***catching*** the exception

Concept

Example: nocatch.py

```
fin = open('bad_file')

for line in fin:

    print line

fin.close()
```

Example: catch.py

```
try:
    fin = open('bad_file')
    for line in fin:
        print line
    fin.close()
except:
    print('Something went wrong.')
    raise
```

Concept

Catching specific problems

- There are a number of kinds of exceptions, including:
  - `IndexError`
  - `EOFError`
  - `KeyError`
  - `SyntaxError`

Concept

Example

```
try:
    fin = open('bad_file')
    for line in fin:
        print line
    fin.close()
except IOError:
    print('Something went wrong.')
```

Example

Error handling

- Once you catch the error, you need to handle it

  1. Perform an alternate action

  2. Generate a customised error message and gracefully end the program

Concept