Preparatory Course Informatics for Life Scientists

An Introduction to Python 6:
Exception Handling

Philipp Thiel

September 13, 2022

Modules

- Python programs terminate when encountering a **syntax error** or **exception**
- We already met syntax errors and they are detected by the parser
- At this stage the program is not yet executed
- Python reads and interprets what to execute
- In contrast, exceptions happen during program execution
- Exceptions happen when syntactically correct code leads to an error
- Exceptions need to be handled to prevent program crash
- Unhandled exceptions can lead to severe problems and security risks

Exception Handling

- Some syntax errors

myscript.py

```
1  a = 'literal"
```

Exception Handling

- Some syntax errors

myscript.py

```
1  a = 'literal"
```

```
1  $ python myscript.py
2    File "myscript.py", line 1
3      a = 'literal"
4                  ^
5  SyntaxError: EOL while scanning string literal
6  $
```

Exception Handling

- Some syntax errors

myscript.py

```
1  a = 'literal'
2   b = 'literal'
```

Exception Handling

- Some syntax errors

myscript.py

```
1  a = 'literal'
2   b = 'literal'
```

```
1  $ python myscript.py
2    File "myscript.py", line 2
3      b = 'literal'
4      ^
5  IndentationError: unexpected indent
6  $
```

Exception Handling

- Some syntax errors

myscript.py

```
1  print( 1/0 ))
```

Exception Handling

- Some syntax errors

myscript.py

```
1  print( 1/0 ))
```

```
1  $ python myscript.py
2    File "myscript.py", line 1
3      print( 1/0 ))
4                   ^
5  SyntaxError: unmatched ')'
6  $
```

Exception Handling

- An exception

myscript.py

```
1  print( 1 / 0 )
```

Exception Handling

- An exception

myscript.py

```
1  print( 1 / 0 )
```

```
1  $ python myscript.py
2  Traceback (most recent call last):
3    File "myscript.py", line 1, in <module>
4      print( 1 / 0 )
5  ZeroDivisionError: division by zero
6  $
```

Exception Handling

- An exception

```
$ python myscript.py
Traceback (most recent call last):
  File "myscript.py", line 1, in <module>
    print( 1 / 0 )
ZeroDivisionError: division by zero
$
```

- Python could have foreseen … but look here

Exception Handling

- Now, the operands are entered by the user
- No chance to detect such a situation beforehand

myscript.py

```
1  a = int(input())
2  print( 1 / a )
```

```
1  $ python myscript.py
2  0
3  Traceback (most recent call last):
4    File "myscript.py", line 2, in <module>
5      print( 1 / a )
6  ZeroDivisionError: division by zero
7  $
```

Exception Handling

- A mechanism to isolate critical code is to use **try** and **except**
- Exceptions in the **try** body are catched
- Upon an exception the **except** body is executed before termination

myscript.py

```
a = int(input())

try:
    # Critical code goes here
    print( 1 / a )
except:
    # Chance for the programmer to implement cleanup code
    # E.g. closing open files
    print('Division by zero. Cleaning up and abort.')
```

```
$ python myscript.py
0
Division by zero. Cleaning up and abort.
$
```

Exception Handling

- It is also possible to catch the error that has been raised
- Then you have access to the final error message
- This can give helpful information to understand what went wrong
- You need to know the type of error, however

myscript.py

```
try:
    # Critical code goes here
    f = open('P0108.fasta', 'r')
except FileNotFoundError as error:
    print(error)
```

```
$ python myscript.py
[Errno 2] No such file or directory: 'P0108.fasta'
$
```

Diving Deeper ...

A good overview about exception handling

→ https://realpython.com/python-exceptions/

License and Contributors

Main Author
Philipp Thiel

Additional Contributors
n.a.