Lecture 10: Exceptions

Swakkhar Shatabda

B.Sc. in Data Science
Department of Computer Science and Engineering
United International University

March 19, 2024

Exceptions

- Python uses special objects called exceptions to manage errors that arise during a program's execution.
- Whenever an error occurs that makes Python unsure what to do next, it creates an exception object.
- If you write code that handles the exception, the program will continue running.
- If you don't handle the exception, the program will halt and show a traceback, which includes a report of the exception that was raised.

```
print(5/0)
```

```
Traceback (most recent call last):
  File "/Users/swakkhar/Desktop/search.py", line 1, in <module
    print(5/0)
ZeroDivisionError: division by zero
```

Exception Handling

- Exceptions are handled with try-except blocks.
- A try-except block asks Python to do something, but it also tells Python what to do if an exception is raised.
- When you use try-except blocks, your programs will continue running even if things start to go wrong.
- Instead of tracebacks, which can be confusing for users to read, users will see friendly error messages that you write.

```
try:
    print(5/0)
except ZeroDivisionError:
    print("You can't divide by zero!")
```

```
You can't divide by zero!
```

The else block

- Any code that depends on the try block executing successfully goes in the else block.

```
print("Give me two numbers, and I'll divide them.")
print("Enter 'q' to quit.")
while True:
    first_number = input("\nFirst number: ")
    if first_number == 'q':
        break
    second_number = input("Second number: ")
    try:
        answer = int(first_number) / int(second_number)
    except ZeroDivisionError:
            print("You can't divide by 0!")
    else: print(answer)
```

Handling the FileNotFoundError Exception

- One common issue when working with files is handling missing files.
- The file you're looking for might be in a different location, the filename may be misspelled, or the file may not exist at all.

```
filename = 'search.py'
with open(filename) as f_obj:
    contents = f_obj.read()
```

```
Traceback (most recent call last):
  File "/Users/swakkhar/Desktop/search.py", line 2, in <module
    with open(filename) as f_obj:
FileNotFoundError: [Errno 2] No such file or directory: 'searc
```

Analyzing text

```
filename = 'search.py'
try:
    with open(filename) as f_obj:
        contents = f_obj.read()
except FileNotFoundError:
    msg = "Sorry, the file " + filename + " does not exist."
    print(msg)
else:
    words = contents.split()
    num_words = len(words)
    print("The file " + filename + " has about " + str(num_words) + " word
```

Multiple Exceptions

- The point in the program at which an exception occurs is often referred to as the raise point.
- If there are several except handlers, the interpreter searches for the first one that matches the type of the raised exception.

```
while True:
    try:
        number1 = int(input('Enter numerator: '))
        number2 = int(input('Enter denominator: '))
        result = number1 / number2
    except ValueError:
        print('You must enter two integers\n')
    except ZeroDivisionError:
        print('Attempted to divide by zero\n')
    else:
        print(f'{number1:.3f} / {number2:.3f} = {result:.3f}')
        break
```

Finally

- A try statement may have a finally clause as its last clause after any except clauses or else clause.
- The finally clause is guaranteed to execute, regardless of whether its try suite executes successfully or an exception occurs.

```
try:
    print('try suite with no exceptions raised')
except:
    print('this will not execute')
else:
    print('else executes because no exceptions in the try suite')
finally:
    print('finally always executes')
```

Explicitly Raising an Exception

```
def add(a,b):
    if not isinstance(a,type(1)) or not isinstance(b,type(1)):
        raise ValueError("Operand not integer type")
    else:
        return a+b

add(4,5)
add(4,"dd")
```

```
Traceback (most recent call last):
  File "/Users/swakkhar/Desktop/search.py", line 8, in <module
    add(4,"dd")
  File "/Users/swakkhar/Desktop/search.py", line 3, in add
    raise ValueError("Operand not integer type")
ValueError: Operand not integer type
```

Except raised errors

```
def add(a,b):
    if not isinstance(a,type(1)) or not isinstance(b,type(1)):
        raise ValueError("Operand not integer type")
    else:
        return a+b

try:
    add(4,"dd")
except ValueError as v:
    print(v)
```

```
Operand not integer type
```