Exception handling and File handling

Department of CE-AI/
CE-Big data

Unit no. 4 Exception handling
and File handling

Python Errors are of three types:

- Compile time errors (syntax errors)
- Runtime errors (Exceptions)
- Logical errors

Compile time errors (syntax errors)

Errors caused by not following the proper structure (syntax) of the language are called syntax errors. It is also known as parsing errors.

Example 1

Missing a colon in the statements like if, while, for etc.

```python
if mark >= 50
    print("You passed!")
```

```
File "<ipython-input-2-3a8932c18758>", line 1
if mark >= 50
                  ^

SyntaxError: expected ':'
```

Example 2

```python
if flag:
print("Flag is set!")
```

```
File "<ipython-input-3-3314775bfb77>", line 2
    print("Flag is set!")
    ^
IndentationError: expected an indented block
after 'if' statement on line 1
```

Runtime errors (Exceptions)

Your code may be syntactically correct but it may happen that during run-time python encounters something which it can't handle, then it raises an exception.

Example 1

```python
list=["A","B","c"]
print(list[3])
```

```
IndexError    Traceback (most recent call last)
<ipython-input-6-6722cd0790de> in <cell line: 2>()
      1 list=["A","B","c"]
----> 2 print(list[3])
IndexError: list index out of range
```

Example 2

```python
subtotal = input("Enter total before tax:")
tax = .08 * subTotal
print("tax on", subtotal, "is:", tax)
```

```
NameError Traceback (most recent call last)
<ipython-input-10-fa97cb51da6a> in <cell line: 2>()
      1 subtotal = input("Enter total before tax:")
----> 2 tax = .08 * subTotal
      3 print("tax on", subtotal, "is:", tax)
NameError: name 'subTotal' is not defined
```

Logical errors

Logical errors occur when the program runs without crashing, but produces an incorrect result.

You won't get an error message, because no syntax or runtime error has occurred.

Example

Write a program to print the sum of squares range upto 10.

```
sum_squares = 0
for i in range(10):
    i_sq = i**2
sum_squares += i_sq
print(sum_squares)
```

```
Output:
81
```

What is Exception?

- An exception is an unexpected event that occurs during program execution, that disrupts the normal flow of the program's instructions.
- It is also known as runtime errors.

EXCEPTION	DESCRIPTION
IndexError	When the wrong index of a list is retrieved.
AssertionError	It occurs when the assert statement fails
AttributeError	It occurs when an attribute assignment is failed.
ImportError	It occurs when an imported module is not found.
KeyError	It occurs when the key of the dictionary is not found.
NameError	It occurs when the variable is not defined.
MemoryError	It occurs when a program runs out of memory.
TypeError	It occurs when a function and operation are applied in an incorrect type.

What is Exception Handling?

It is the process of responding to the occurrence, during computation, of exceptional conditions requiring special processing – often changing the normal flow of program execution.

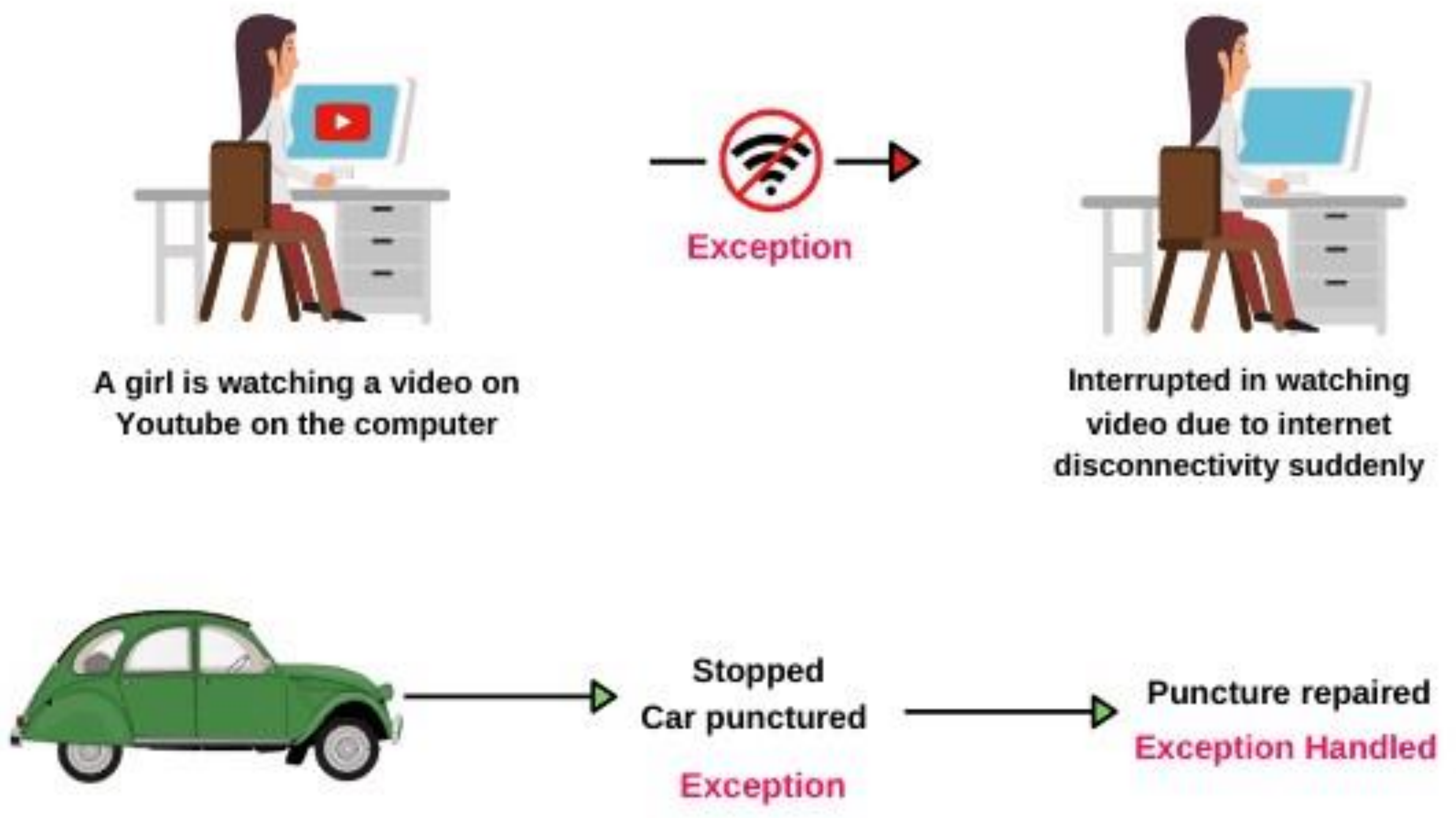
A girl is watching a video on Youtube on the computer
Exception
Interrupted in watching video due to internet disconnectivity suddenly
Stopped
Car punctured
Exception
Puncture repaired
Exception Handled

Fig: Realtime Example of Exception Handling

Try statement

- ✓ First, the try clause (statement(s) between the try and except keywords) is executed.
- ✓ If no exception occurs, the except clause is skipped and execution of the try statement is finished.

Syntax of try statement:

```
try:
    statement
```

Except statement

- ✓ If an exception occurs during execution of the try clause, the rest of the clause is skipped. Then if its types matches the exception named after the except keyword, the except clause is executed, and then execution continues after the try statement.

```
except   :
    statement
```

Example of try – except statement:

```
a = [1, 2, 3]
try:
    print ("Second element = %d" %(a[1]))

    print ("Fourth element = %d" %(a[3]))

except:
    print ("An error occurred")
```

Else statement

You can also use the else clause on the try-except block which must be present after all the except clauses. The code enters the else block only if the try clause does not raise an exception.

```
try:
    Statement
except exception:
    Statement
else:
    statement
```

Example of try-except-else statement

```
def AbyB(a , b):
    try:
        c = ((a+b) / (a-b))
    except ZeroDivisionError:
        print ("a/b result in 0")
    else:
        print (c)

# Driver program to test above function
AbyB(2.0, 3.0)
AbyB(3.0, 3.0)
```

Finally statement:

It is always executed after the try and except blocks. The final block always executes after the normal termination of the try block or after the try block terminates due to some exception.

```
try:
   Statement
except exception:
   Statement
Finally
   Statement
```

Example of Finally statement:

```python
# No exception Exception raised in try block
try:
    k = 5//0  # raises divide by zero exception.
    print(k)

# handles zerodivision exception
except ZeroDivisionError:
    print("Can't divide by zero")

finally:
    # this block is always executed
    # regardless of exception generation.
    print('This is always executed')
```

- ✓ *Finally* block is always executed after leaving the *try* statement. In case if some exception was not handled by except block, it is re-raised after execution of finally block.

- ✓ One can use *finally* just after *try* without using *except* block, but no exception is handled in that case.

```
# Exception is not handled
try:
    k = 5//0 # exception raised
    print(k)

finally:
    print('This is always executed')
```

Argument of an Exception

An exception can have an argument, which is a value that gives additional information about the problem. The contents of the argument vary from exception to exception.

```
try:
  b=float(56+78/0)
except Exception as Argument:
  print('This is the Argument\n', Argument)
```

Raising an Exception

The raise statement in python is used to raise an exception. Try-except blocks can be used to manage exceptions, which are errors that happen while a programme is running. When an exception is triggered, the programme goes to the closest exception handler, interrupting the regular flow of execution.

- ✓ The raise keyword is typically used inside a function or method, and is used to indicate an error condition.
- ✓ We can throw an exception and immediately halt the running of your programme by using the raise keyword.

Example of Raising an Exception

```
def check_age(age):
    if age < 18:
        raise Exception(" Age must be 18
or above. ")
    else:
        print(" Age is valid. ")

try:
    check_age(17)
except Exception as e:
    print(" An error occurred: ", e)
```

- ✓Python looks for the closest exception handler, which is often defined using a try-except block, when an exception is triggered.
- ✓If an exception handler is discovered, its code is performed, and the try-except block's starting point is reached again.
- ✓If an exception handler cannot be located, the software crashes and an error message appears.

File handling

Files

A file is a container in computer storage devices used for storing data.

Hence, in Python, a file operation takes place in the following order:

- ✓ Open a file
- ✓ Read or write (perform operation)
- ✓ Close the file

Types of Files in python

Python provides inbuilt functions for creating, writing, and reading files. There are two types of files:-

1. Normal text files
2. Binary files

- **Text files:** In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is the new line character (‘\n’) in python by default.
- **Binary files:** In this type of file, there is no terminator for a line, and the data is stored after converting it into machine-understandable binary language, 0s, and 1s.

File handling

The process of file handling refers to how we store the available data or information in a file with the help of a program.

That means, we will read from and write into files. To do so, we need to maintain some steps. Those are

1. Open a file
2. Read or write (perform operation)
3. Close the file

How to open a file

The first step to working with files in Python is to learn how to open a file. You can open files using the open() method.

```
#open the file
text_file = open('filename')
```

The open() function in Python accepts two arguments. The first one is the file name along with the complete path and the second one is the file open mode.

```
#open the file
text_file=open('filename','mode')
```

Different Modes to Open a File in Python

Mode	Description
r	Open a file for reading. (default)
w	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
x	Open a file for exclusive creation. If the file already exists, the operation fails.
a	Open a file for appending at the end of the file without truncating it. Creates a new file if it does not exist.
t	Open in text mode. (default)
b	Open in binary mode.
+	Open a file for updating (reading and writing)

Reading Files in Python

After we open a file, we use the read() method to read its contents. For example,

```
# open a file
file1 = open("test.txt", "r")

# read the file
read_content = file1.read()
print(read_content)
```

Closing Files in Python

When we are done with performing operations on the file, we need to properly close the file. It is done using the close() method in Python. For example,

```
# open a file
file1 = open("test.txt", "r")
# read the file
read_content = file1.read()
print(read_content)
# close the file
file1.close()
```

Writing to Files in Python

There are two things we need to remember while writing to a file.

- ✓If we try to open a file that doesn't exist, a new file is created.
- ✓If a file already exists, its content is erased, and new content is added to the file.

In order to write into a file in Python, we need to open it in write mode by passing "w" inside open() as a second argument.

```
# open a file
file1 = open("test.txt", “w")
# write a file
file2.write(‘how are you.')
file2.write(‘I am good.')
# close the file
file1.close()
```

Opening a File in Append Mode

Opening a file in append mode makes it possible for you to add new content to a file without overwriting the existing data.

```
# Open the file in append mode
file = open('myfile.txt', 'a')
# Append new lines to the file
file.write("Line 4\n")
file.write("Line 5\n")
file.write("Line 6\n")
# Close the file
file.close()
```

Python tell() method

tell() method can be used to get the position of File Handle. tell() method returns current position of file object. This method takes no parameters and returns an integer value.

```
# Open the file in read mode
fp = open("myfile.txt", "r")

# Print the position of handle
print(fp.tell())

#Closing file
fp.close()
```

Python seek() Method

Python seek() method is used for changing the current location of the file handle. The file handle is like a cursor, which is used for defining the location of the data which has to be read or written in the file.

Syntax

```
file.seek(offset, from_where)
```

Offset: This is used for defining the number of positions to move forward.

from_where: This is used for defining the point of reference.

Example

```
fi = open("text.txt", "r")
# the second parameter of the seek method
is by default 0
fi.seek(30)
print(fi.read())
fi.close()
```