

# Module 1

## Introduction to everything

### Section

Python programming: exception handling & file handling

# Exception handling in python

## Python Syntax errors v/s exceptions

**Syntax Error:** As the name suggests this error is caused by the wrong syntax in the code. It leads to the termination of the program.

**Exceptions:** Exceptions are raised when the program is syntactically correct, but the code resulted in an error. This error does not stop the execution of the program; however, it changes the normal flow of the program.

The syntax error exception occurs when the code does not conform to Python keywords, naming style, or programming structure. The interpreter sees the invalid syntax during its parsing phase and raises a Syntax Error exception. The program stops and fails at the point where the syntax error happened. That's why syntax errors are exceptions that can't be handled.

On the other hand, an exception happens when the code has no syntax error but encounters other error situations. These conditions can be addressed within the code—either in the current function or in the calling stack. In this sense, exceptions are not fatal. A Python program can continue to run if it gracefully handles the exception.

Syntax error  
Not using colon (:)  
after the condition is  
invalid in python

```
1 # initialize the age variable
2 age = 18
3
4 # checking if you are eligible to cast your vote or not
5 if (age < 18)
6 print("You are NOT eligible to cast your vote.")

File "<ipython-input-42-85eadb0eae2a>", line 5
    if (age < 18)
        ^
SyntaxError: invalid syntax
```

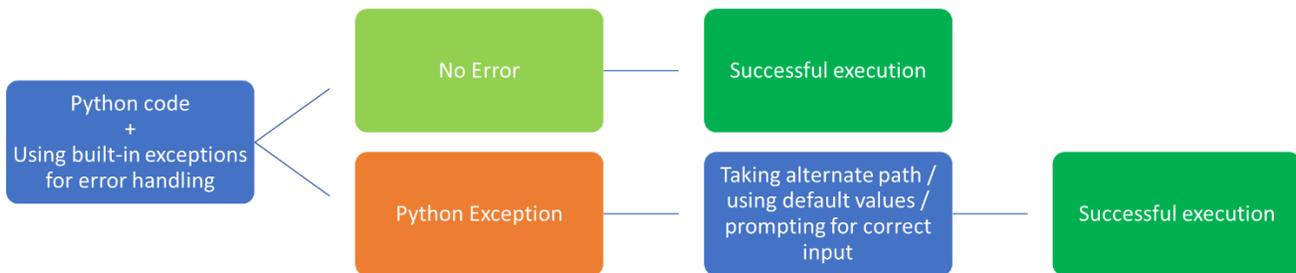
```
1 # initialize the age variable
2 age = 18
3
4 # dividng age by 0
5 out = age/0

-----
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-43-035f83291406> in <module>
      3
      4 # dividng age by 0
----> 5 out = age/0

ZeroDivisionError: division by zero
```

Dividing by zero  
is a python  
exception

**Exceptions** are raised when the program is syntactically correct, but the code resulted in an error. This error does not stop the execution of the program; however, it changes the normal flow of the program. The exceptions are handled using try, except and finally.



## Notes –

- Exceptions in Python applications can happen for many reasons; and if they aren't handled well, these exceptions can cause the program to crash, causing data loss, or worse, corrupted data.
- While coding with python, it is important to plan about possible exception situations and include error handling in your code.
- Python has many built-in exceptions that are raised when program encounters an error (something in the program goes wrong).
- With the help of built-in exceptions, programs can determine the error type at run time and act accordingly to take alternative path or using default values or prompting for correct input.

## Handling Exceptions

- The **try** block lets you test a block of code for errors.
- The **except** block lets you handle the error.
- The **finally** block lets you execute code, regardless of the result of the try- and except blocks.

## Example Code

```

try:
    print(x)
except:

```

```
print("An exception occurred")
```

```
finally:
```

```
print("Hello")
```

Output

```
An exception occurred
```

```
Hello
```

## File handling in python

Variables can store data when the program is running, but if the data has to be persistent, even after the program has finished, the data has to be stored in files.

- Files can be used to save an enormous amount of data and they can be reused any number of times as required.
- Files have two important properties, file name and file path, i.e., the location of the file in the computer.
- Using files will make the programs easy to use. When a file name is passed, Python will look for the directory in which the file is saved and using Python, we can create, read, write and save files.

Access mode	Operation	Position of file pointer in file
r	Read only	Beginning
rb	Read only in binary format	Beginning
r+	Read and write both	Beginning
rb+	Read and write both in binary format	Beginning
w	Write only, overwrite existing file, creates new if no file exist	Beginning
wb	Write only in binary format, overwrite existing file, creates new if no file exist	Beginning
w+	Write and read modes, overwrites existing file, creates new file if not no file exist	Beginning
wb+	Read and write in binary format	Beginning
a	Append mode, creates new file if no file exist	End
ab	Append mode in binary format, creates new file if no file exist	End
a+	Append and read, creates new file if no file exist	End
ab+	Append and read in binary format	End

## Reading from files

- A Python program can read the content in a file and also can modify its content or can even delete it.
- Content of a file can be read entirely or line by line.
- Open () function is used to open a file, which will return an object that represents the file to be accessed.
- The read () method tells Python to read the contents of the opened file.
- We can specify the mode while opening a file. In mode, we specify whether we want to read r, write w or append a to the file. We can also specify if we want to open the file in text mode or binary mode.

## Example Code

 jupyter abcd.txt ✓ a few seconds ago

File Edit View Language

```
1 This is line 1.
2 This is line 2.
3 This is line 3.
4 This is line 4.
5 This is line 5.
```

```
file = open('abcd.txt', 'r')
```

```
# This will print every line in file
```

```
for line in file:
```

```
    print (line)
```

## Output

This is line 1.

This is line 2.

This is line 3.

This is line 4.

This is line 5.

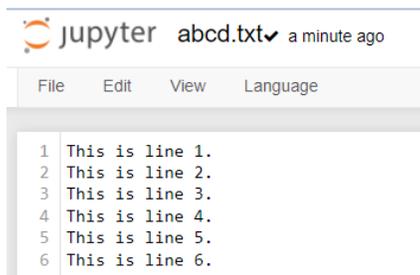
## Writing to files

- To write text to a file, you need to call `open ()` with a second argument telling Python that you want to write to the file.
- Python can only write strings to a text file. If you want to store numerical data in a text file, you'll have to convert the data to string format first using the `str ()` function.
- If you want to add content to a file instead of writing over existing content, the file can be opened in append mode. This will not erase the existing content, but adds the new content over it.
- **With** statement can be used with `open ()` in Python. It aids in making the code cleaner and much more readable. It simplifies the management of common resources like file streams. Using this method any files opened will be closed automatically after one is done, so auto-cleanup.

## Example Code

```
file = open('abcd.txt', 'a')
file.write("\nThis is line 6.")
file.close()
```

## Output

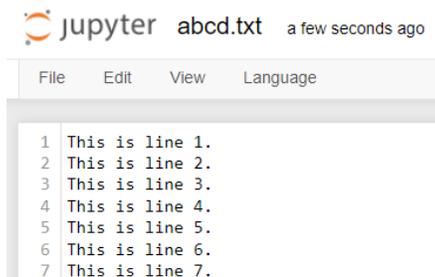


```
jupyter abcd.txt a minute ago
File Edit View Language
1 This is line 1.
2 This is line 2.
3 This is line 3.
4 This is line 4.
5 This is line 5.
6 This is line 6.
```

## Example Code

```
with open ('abcd.txt','a') as file:
    file.write("\nThis is line 7.")
```

## Output



```
jupyter abcd.txt a few seconds ago
File Edit View Language
1 This is line 1.
2 This is line 2.
3 This is line 3.
4 This is line 4.
5 This is line 5.
6 This is line 6.
7 This is line 7.
```

## Summary

- Python has many built-in code exceptions which can be useful in programming alternate actions when certain exceptions occur.
- Python function `open ()` can be used to handle files, it helps in reading, writing, appending data to files.