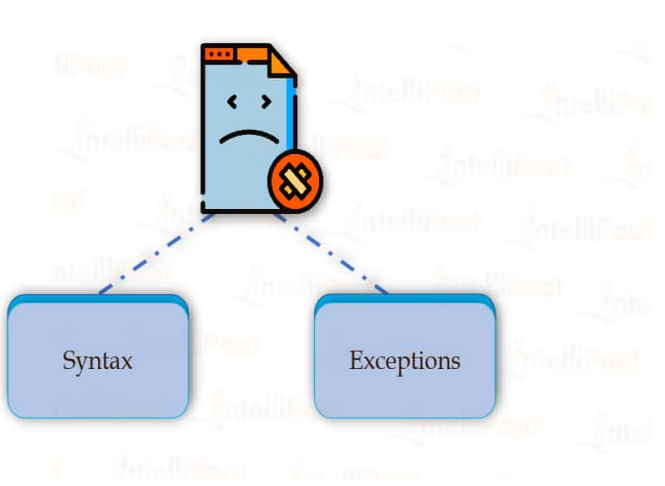
Errors and Exceptions in Python

Here, we will discuss :

- **What an exception is and how it differs from a syntax error**
- **Anatomy of Error**
- **Some built in Exceptions**

What Is an Error in Python ?

- **An error is an action that is incorrect or inaccurate. Due to which the program fails to execute.**
- **The errors can be broadly classified into two types :**
  - **Syntax errors**
  - **Logical errors (Exception)**



Syntax Error :

- **As the name suggests this error is caused by the wrong structure or syntax in the code.**
- **A syntax error is also known as a parsing error.**
- **When the parser found a syntax error it exits with an error message without running anything.**

Common Python Syntax errors :

- **Incorrect indentation.**
- **Missing colon, comma, or brackets.**
- **Putting keywords in the wrong place.**

In [4]:
```
#Syntax Error
print("syntax error")
for i in range(10)
    print(i)
```

```
  File "<ipython-input-4-0368748bca5f>", line 3
    for i in range(10)
                      ^
SyntaxError: invalid syntax
```

The arrow indicates where the parser ran into the syntax error

Logical errors (Exception) :

- **Even if code is syntactically correct, the error that occurs at the runtime is known as a Logical error or Exception.**
- **An event that occurs during the execution of programs that disrupt the flow of execution are called EXCEPTIONS.**
Runtime exceptions, generally a result of programming errors, such as:

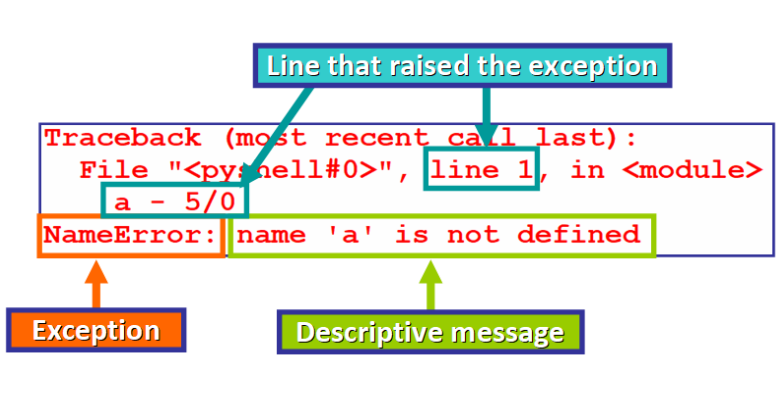
- **Indenting a block to the wrong level.**
- **Using the wrong variable name.**
- **Making a mistake in a boolean expression.**
- **Reading a file that is not present.**
- **Trying to read data outside the available index of a list.**
- **Dividing an integer value by zero.**

In [3]:
```
#Exception
a = 10
b = 20
print("Addition:", a + c)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-c09b5b0468c8> in <module>
      2 a = 10
      3 b = 20
----> 4 print("Addition:", a + c)

NameError: name 'c' is not defined
```

Anatomy of an Exception :

- **An exception is an object derives from the BaseException class that contains information about an error.**
- **Exception object contains :**
  - **Error type (exception name)**
  - **The state of the program when the error occurred**
  - **An error message describes the error event.**



- **If the line that raised the exception belongs to a function, module is replaced by the name of the function.**
- **A traceback is basically a list detailing the function calls that were made before the exception was raised.**

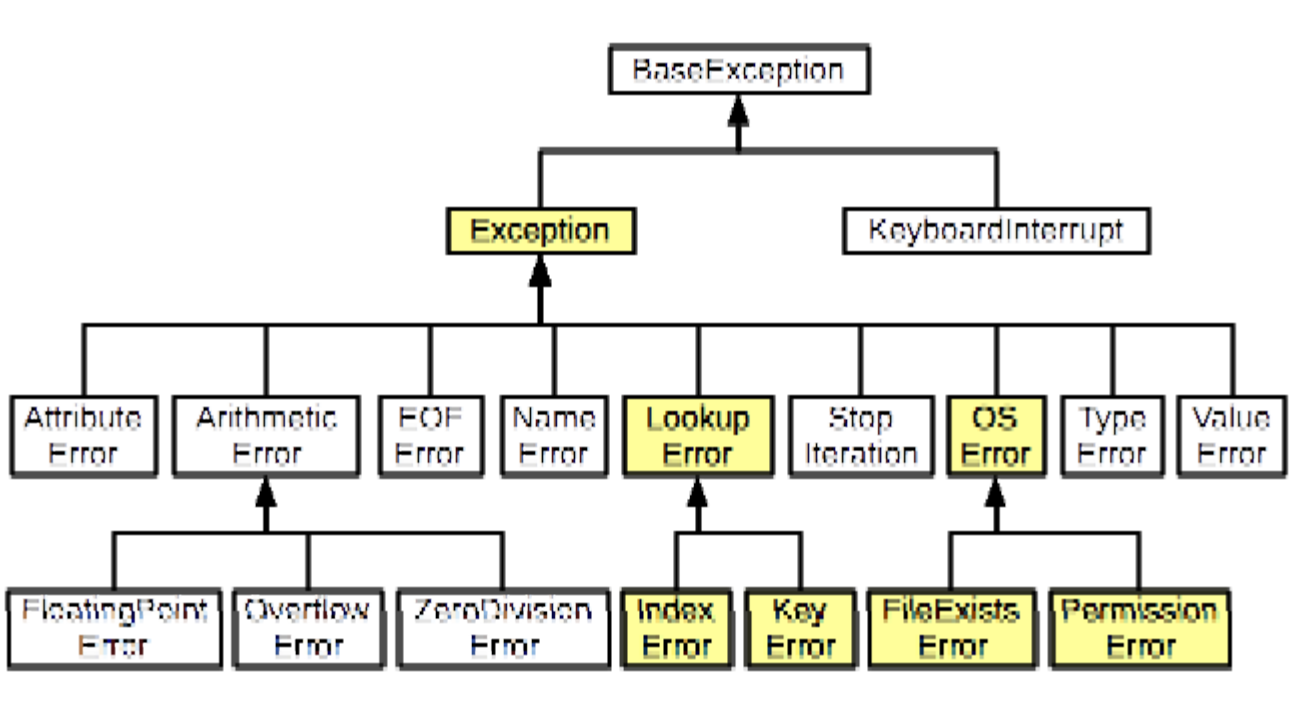
Default Exception Handing in Python :

- **There are many different types of exceptions, and they are all raised in particular situations**
- **For every exception , a corresponding class is available and every exception is object to its corresponding class**
- **Whenever an exception occurs, PVM will create the corresponding exception object and check for handling code.**
- **If handling code is not available, then interpreter terminates the program and prints exception info to the console**

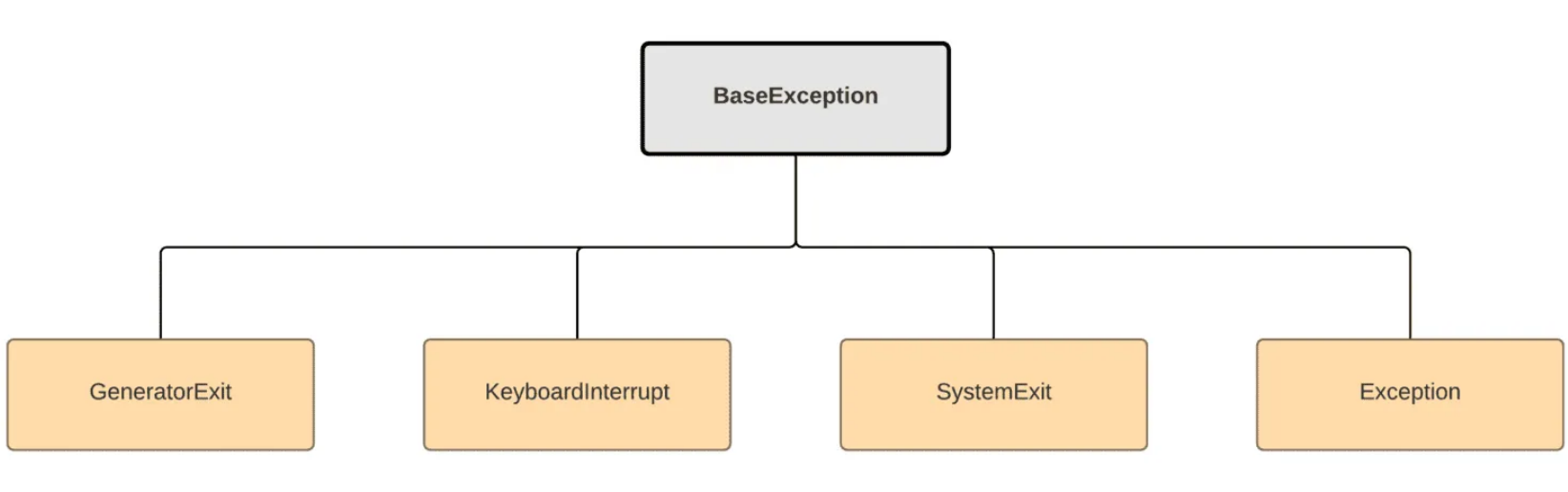
Exception Hierarchy :

- **The BaseException class is the root class for all exception classes in python exception hierarchy**

The below image shows different built-in exceptions :



Base exception classes in Python :



- **BaseException - is the base class for all the built-in exceptions.**

We should not directly inherit this class to define any custom exception classes

- **GeneratorExit - Python throws this exception when a coroutine or generator is closed.**
- **KeyboardInterrupt - When a user hits the interrupt key such as Delete or Control+C, this error exception is raised.**
- **SystemExit - When the sys.exit( ) function is called, this exception is raised.**
- **Exception - is a base class for all built-in non-system-exiting exceptions and user-defined exceptions**

In [5]:
```
#keyboard Interrupt
import time
while True:
    print("Stop the kernel")
    time.sleep(5)
```

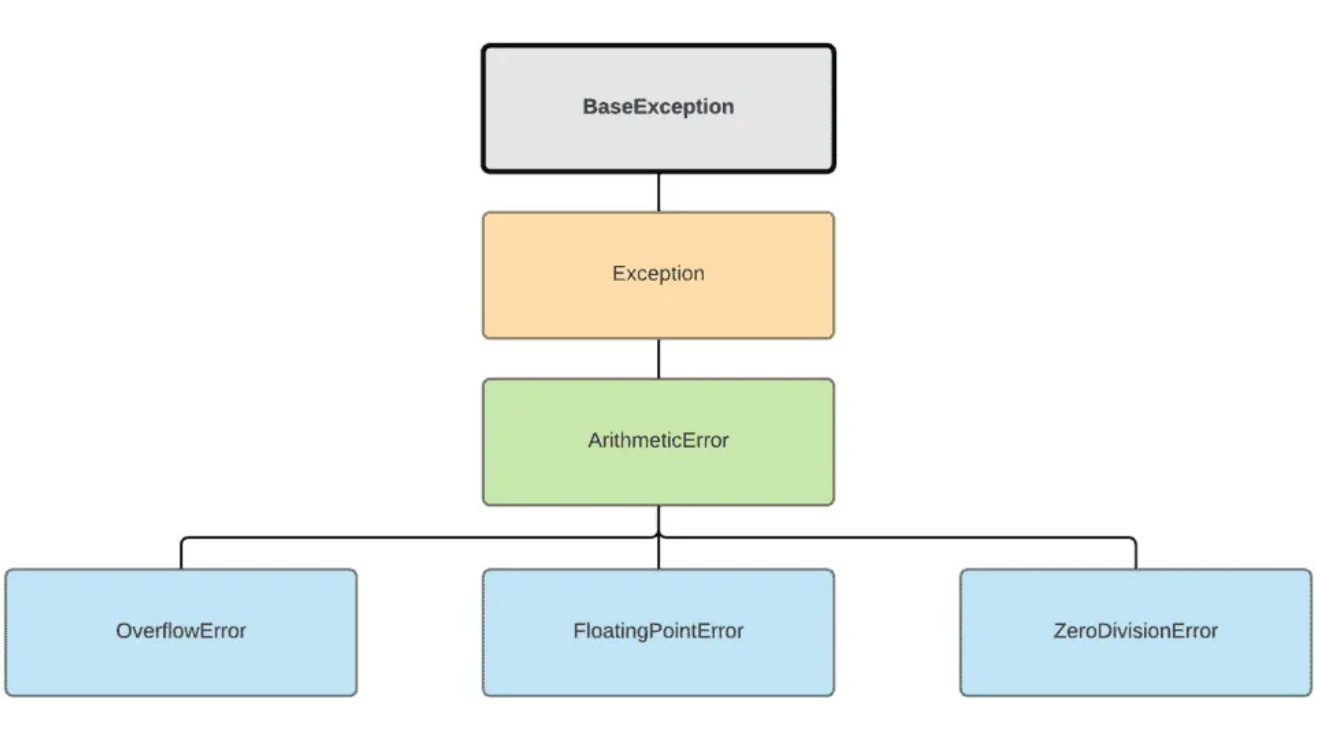
```
Stop the kernel
Stop the kernel
Stop the kernel
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-5-f1e4cc0cec53> in <module>
      3 while True:
      4     print("Stop the kernel")
----> 5     time.sleep(5)

KeyboardInterrupt:
```

In [11]:
```
#GeneratorExit
def sample():
    arr = [1, 2, 3, 4, 5]
    for i in arr:
        yield i
        print("From Function: ", i)
        print("Reached the end!!!")

for num in sample():
    print("Output: ", num)
    if(num >= 3):
        break
print("Generatorexit is called")
#so the print statements after last yield is not excuted - it is not an error.
```

```
Output:  1
From Function:  1
Reached the end!!!
Output:  2
From Function:  2
Reached the end!!!
Output:  3
Generatorexit is called
```

TO DO : Excute a function with sys.exit( )

Arithmetic Error classes in Python :



- **ArithmeticError - is the base exception class for arithmetic errors such as**

FloatingPointError, ZeroDivisionError, and OverflowError.

- **FloatingPointError - you can catch this exception when the operation of a floating-point value fails.**

However, this exception is always defined and handled internally

- **ZeroDivisionError -happens when second argument of the division or modulo operation equals zero.**
- **OverflowError - this exception happens when the results from an arithmetic operation are out of range**

and also if integers are out of a required range.

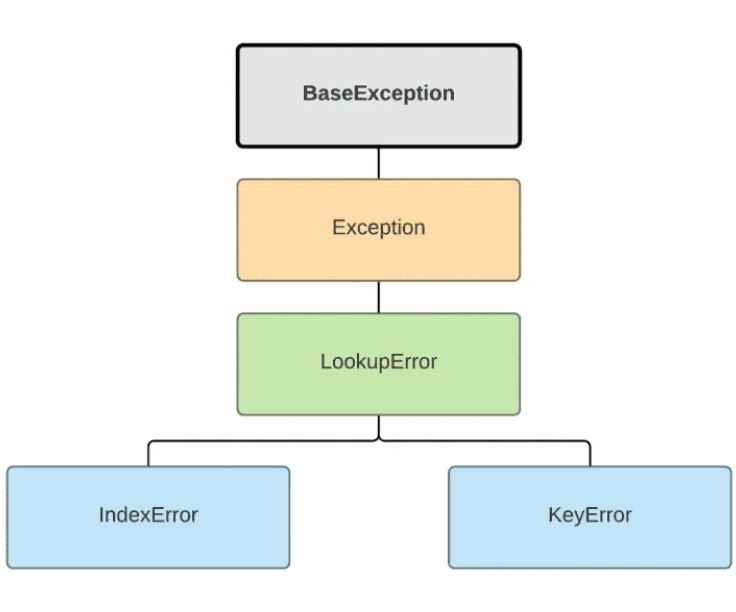
In [14]:
```
#Zero Division Error
print(10/0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-14-f76ff30294c8> in <module>
      1 #Zero Division Error
----> 2 print(10/0)

ZeroDivisionError: division by zero
```

In [13]:
```
#Simple program for showing overflow error
import math
print(math.exp(1000))
```

```
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
<ipython-input-13-872e1daed7c7> in <module>
      1 #Simple program for showing overflow error
      2 import math
----> 3 print(math.exp(1000))

OverflowError: math range error
```

Lookup Error classes in Python :



- **LookupError - Base class for IndexError and KeyError exceptionswhich are raised**

when we try to manipulate non-existing or invalid index or key values at sequence or mapping.

- **IndexError - exception happens when a referenced sequence is out of range**
- **KeyError - We can catch this exception if a mapping key not found in the set of existing keys.**

In [15]:
```
#IndexError
num = [1, 2, 6, 5]
num[5]
```

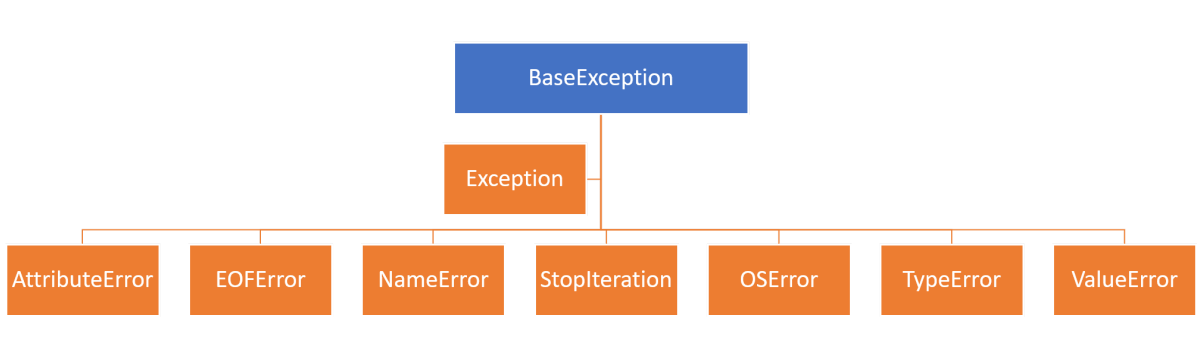
```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-15-bdaf1a1038b2> in <module>
      1 #IndexError
      2 num = [1, 2, 6, 5]
----> 3 num[5]

IndexError: list index out of range
```

In [17]:
```
#KeyError
students = {"Sirisha": 15, "Vinay": 30}
students["Nitheesh"]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-17-5c095098efd9> in <module>
      1 #KeyError
      2 students = {"Sirisha": 15, "Vinay": 30}
----> 3 students["Nitheesh"]

KeyError: 'Nitheesh'
```

Concrete exceptions in Python:

- **Concrete exceptions in Python are build-in exceptions that inherited directly from the Exception class.**

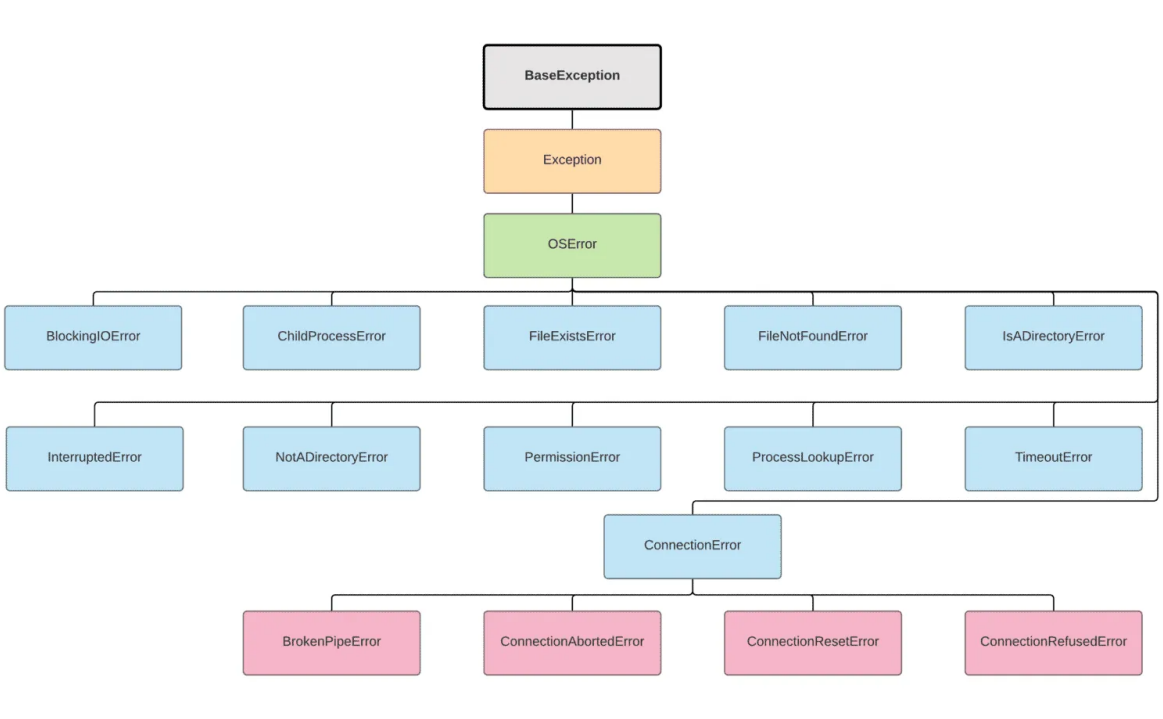


- **AttributeError - This can happen because the referred attribute does not exist.**
- **EOFError - raised if the built-in functions such as input( ) encounters an end-of-file (EOF) condition**

without reading any data.

- **NameError - The exception is raised when a global or local name is not found.**
- **StopIteration - to indicate that all items are produced by the iterator.**
- **OSError([arg]) - Whenever a system function returns a system-related error such as I/O failures**

including “disk full” or “file not found” errors.



- **TypeError - Raised when we apply an inappropriate type object to a function or an operation.**
- **ValueError - when a built-in function or operation receives the correct type of argument but with an invalid value**
- **RecursionError - Derived from RuntimeError and raised when maximum recursion depth has been exceeded.**
- **ImportError - ModuleNotFoundError - Raised when the import statement fails to load a module**

In [21]:
```
#NameError
a = b
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-21-b0c5f1e22258> in <module>
      1 #NameError
----> 2 a = b

NameError: name 'b' is not defined
```

In [18]:
```
#Type Error
print("Hello" + 10)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-18-5741b3f27373> in <module>
      1 #Type Error
----> 2 print("Hello" + 10)

TypeError: can only concatenate str (not "int") to str
```

In [19]:
```
#Value Error
import math
math.sqrt(-10)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-19-0454fc4d1271> in <module>
      1 #Value Error
      2 import math
----> 3 math.sqrt(-10)

ValueError: math domain error
```

In [26]:
```
#EOF Error
student = {
  "name": "John",
  "level": "400",
  "faculty": "Engineering and Technology"
```

```
  File "<ipython-input-26-3d48a6e86ec0>", line 5
    "faculty": "Engineering and Technology"
                                           ^
SyntaxError: unexpected EOF while parsing
```

In [28]:
```
#FileNotFound Error
fp = open("test.txt", "r")
if fp:
    print("file is opened successfully")
```

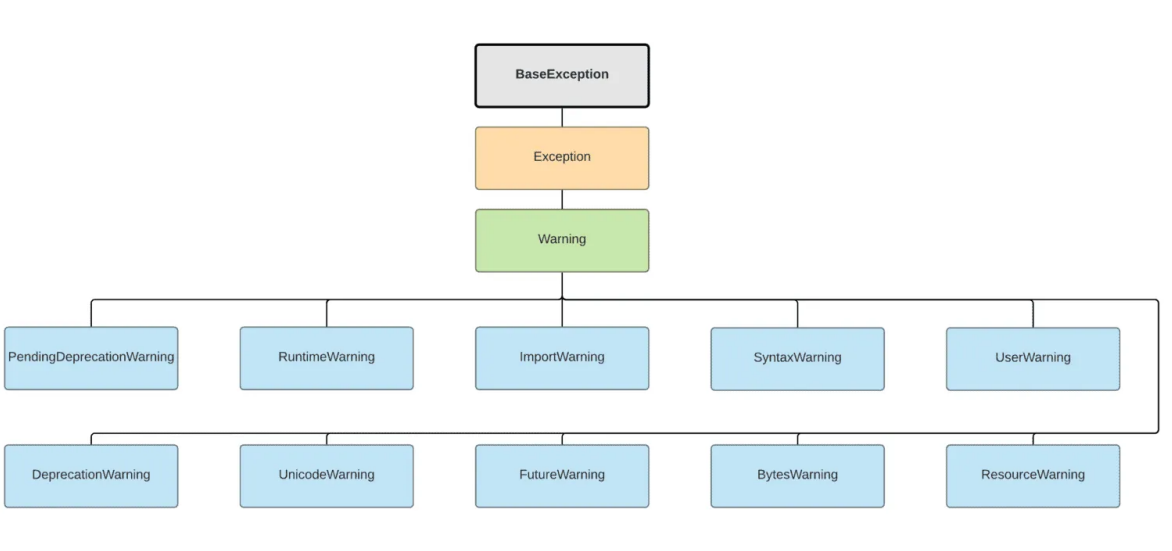
```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-28-ff644ed9477a> in <module>
      1 #FileNotFound Error
----> 2 fp = open("test.txt", "r")
      3 if fp:
      4     print("file is opened successfully")

FileNotFoundError: [Errno 2] No such file or directory: 'test.txt'
```

In [20]:
```
#import error
import yell
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-20-ff98ef686846> in <module>
      1 #import error
----> 2 import yell

ModuleNotFoundError: No module named 'yell'
```

Warnings :

- **The warning doesn’t stop the execution of a program it indicates the possible improvement.**
- **Below is the list of warning exception :**



Handling Errors :

Python provides two very important features to handle any unexpected error in your Python programs and

to add debugging capabilities in them

- **Exception Handling**
- **Assertions**