What is an Exception?

- An exception is an error that happens during execution of a program.
- When that error occurs, Python generate an exception that can be handled, which avoids your program to crash.
- Whenever an exception occurs the program halts the execution and thus further code is not executed.
- Thus exception is that error which python script is unable to tackle with.

Why use Exceptions?

- Exception in a code can also be handled.
- In case it is not handled, then the code is not executed further and hence execution stops when exception occurs.

Hierarchy Of Exception

- **ZeroDivisionError:** Occurs when a number is divided by zero.
- **NameError:** It occurs when a name is not found. It may be local or global.
- **IndentationError:** If incorrect indentation is given.
- **IOError:** It occurs when Input Output operation fails.
- **EOFError:** It occurs when end of file is reached and yet operations are being performed

Exception Handling

- Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them-

1. **Exception Handling.**
2. **Assertions.**

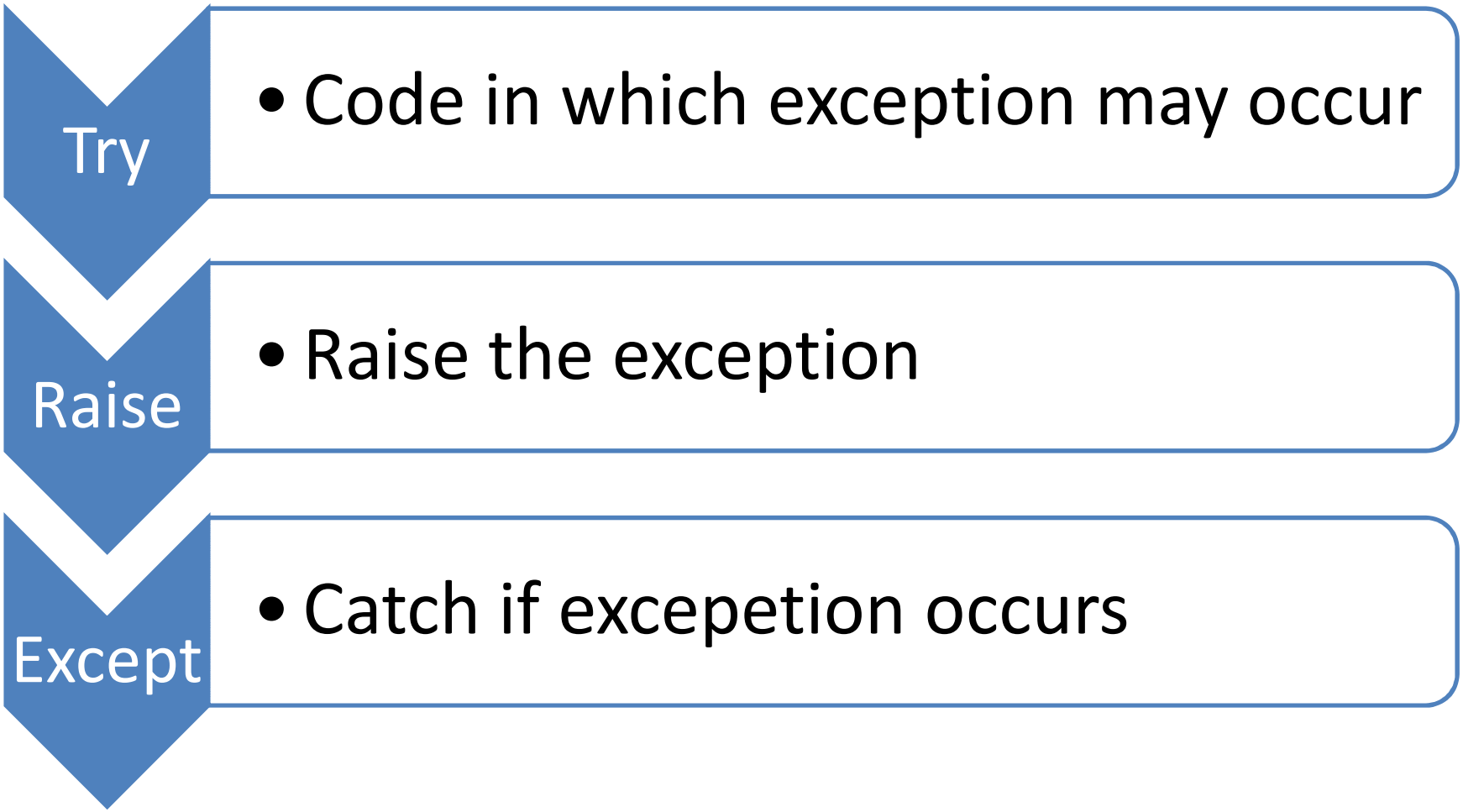
Standard Exceptions

EXCEPTION NAME	DESCRIPTION
Exception	Base class for all exceptions
ArithmeticError	Base class for all errors that occur for numeric calculation.
ZeroDivisonError	Raised when division or modulo by zero takes place for all numeric types.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
ImportError	Raised when an import statement fails.
KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.
NameError	Raised when an identifier is not found in the local or global namespace.
SyntaxError	Raised when there is an error in Python syntax.

EXCEPTION NAME	DESCRIPTION
IndentationError	Raised when indentation is not specified properly.
TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.

Exception Handling

- The suspicious code can be handled by using the try block.
- Enclose the code which raises an exception inside the try block.
- The try block is followed by except clause.
- It is then further followed by statements which are executed during exception and in case if exception does not occur.


Try
• Code in which exception may occur
Raise
• Raise the exception
Except
• Catch if excepetion occurs

Declaring Multiple Exception

- Multiple Exceptions can be declared using the same except statement:

try:

code

except (Exception1,Exception2,Exception3,..,ExceptionN):

*execute this code **in** case any Exception of these occur.*

else:

*execute code **in** case no exception occurred.*

try:
    malicious code
except (Exception1):
    execute code
except (Exception2):
    execute code
....
....
except (ExceptionN):
    execute code
else:
    In case of no exception, execute the **else** block code.

```
try:
    a=10/0
    print (a)
except (ArithmeticError):
        print( "This statement is raising an exception"  )
else:
    print ("Welcome“)
```

```
def fun(a,b):
    try:
        c=(a+b)/(a-b)
    except (ArithmeticError):
        print("EXCEPTION OCCUR")
    else:
        print(c)
    finally:
        print("End of the world")

# Driver program to test above function
fun(2.0, 3.0)      # -5.0      End of the world
fun(3.0, 3.0)      # EXCEPTION OCCUR    End of the world
```

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 by exception.
- You capture an exception's argument by supplying a variable in the except clause as follows-

try:

    You do your operations here

    ....

except ExceptionType **as** Argument:

    You can print value of Argument here...

Example 1

```
def fun(a):
    try:
        return int(a)
    except ValueError as VE:
        print("argument is not a number",VE)


fun(11)      # 11

fun("string")
#argument is not a number invalid literal for int() with base 10: 'string’
```

Example 2

```
def fun(a,b):
        c= a/b
        return c
```

```
try:
    fun("a","b")
except TypeError as VE:
    print("Exception: ",VE)
```

Output:

Exception: unsupported operand type(s) for /: 'str' and 'str'

Raising Exception

- The raise statement allows the programmer to force a specific exception to occur.
- The sole argument in raise indicates the exception to be raised.
- This must be either an exception instance or an exception class (a class that derives from Exception).

```
# Program to depict Raising Exception
try:
    raise NameError("Hey! are you sleeping!")  # Raise Error
except NameError as NE:
    print (NE)
```

```
try:
        raise Exception("How are you")
except Exception as E:
        print(E)
```

```
How are you
```

Explanation:

i) To raise an exception, raise statement is used.

It is followed by exception class name.

ii) Exception can be provided with a value(*optional*) that can be given in the parenthesis. (here, *Hey! are you sleeping!)*

iii) To access the value "as" keyword is used.

“NE" is used as a reference variable which stores the value of the exception.

User-defined Exceptions in Python

Creating User-defined Exception

- Programmers may name their own exceptions by creating a new exception class.
- Exceptions need to be derived from the ***Exception*** class, either directly or indirectly.
- Although not mandatory, most of the exceptions are named as names that end in **“Error”**similar to naming of the standard exceptions in python.

```
# A python program to create user-defined exception
# class MyError is derived from super class Exception
class MyError(Exception):
    # Constructor or Initializer
    def __init__(self, value):
        self.value = value

# __str__ is to print() the value
    def __str__(self):
        return(repr(self.value))
```

```
try:
    raise(MyError(3*2))

                                # Value of Exception is stored in error
except MyError as error:
    print('A New Exception occured: ',error.value)
```

```
class point(Exception):
    def __init__(self,x=0,y=0):
        self.x=x
        self.y=y
    def __str__(self):
        return "({0},{1})".format(self.x, self.y)
```

```
p=point()
 try:
        if(p.x==0 and p.y ==0):
                    raise point()
 except point as error:
        print("point is ORIGIN",error)
 else:
           print(p)
```

OUTPUT:

point is ORIGIN (0,0)