An exception is an event that occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

### STRUCTURE OF A PYTHON EXCEPTION SUITE

```
try:
        code to execute
except ExceptionClassName:
        action
else:  alternative action
finally:
        alternative action
```

1. The “try” clause is mandatory. It contains the code that is to be run assuming that no exception is raised.
2. At least one “except” clause is required. You can include as many “except” clauses as necessary to handle anticipated exceptions.
3. The “else” statement is optional but recommended. It will execute if none of the except conditions are applicable to the actual error raised.
4. The “finally”: clause is optional. When it is included, it always executes. If an exception occurs and is not processed by the time the finally suite executes, stack unwinding occurs; the function’s stack frame is removed from the function-call stack.. Any finally clause that includes code should embed it in a try statement of its own!

### USING A TRY SUITE WHEN OPENING A RESOURCE (FILE)

Resources that require explicit release, such as files, network connections, and database conndctions.

Robust code that use these resources embed the with statement in a try suite.

You will not need a finally: clause because the with statement handles resource deallocation.

### EXPLICITLY RAISING AN EXCEPTION

You may need to write functions that raise exceptions to inform callers of errors that occur.

```
raise ExceptionClassName
```

You can add an argument (inside parentheses) to initiatialize the exception object – typically an error message. You should release any resources acquired before the exception occurred

### TRY SUITE EFFICIENCY

To minimize try code, place a try suite in a significant logical section of a program in which several statements can raise exceptions, but be careful your granularity is appropriate.

An exception object stores information about the precise series of function calls that led to the exception. Useful for debugging !

### TIPS FOR READING TRACEBACKS

Sometimes the libraries you use will raise exceptions. Tp fomd the errors that actually arose from your own code, start reading from the end of the traceback and read the error message firt. Then, read upward through the traceback, looking for the first line that indicates code you wrote. This is typically what caused the exception!

| | |
|---|---|
| 1 | **Exception**<br>Base class for all exceptions |
| 2 | **StopIteration**<br>Raised when the next() method of an iterator does not point to any object. |
| 3 | **SystemExit**<br>Raised by the sys.exit() function. |
| 4 | **StandardError**<br>Base class for all built-in exceptions except StopIteration and SystemExit. |
| 5 | **ArithmeticError**<br>Base class for all errors that occur for numeric calculation. |
| 6 | **OverflowError**<br>Raised when a calculation exceeds maximum limit for a numeric type. |
| 7 | **FloatingPointError**<br>Raised when a floating point calculation fails. |
| 8 | **ZeroDivisionError**<br>Raised when division or modulo by zero takes place for all numeric types. |
| 9 | **AssertionError**<br>Raised in case of failure of the Assert statement. |
| 10 | **AttributeError**<br>Raised in case of failure of attribute reference or assignment. |
| 11 | **EOFError**<br>Raised when there is no input from either the raw_input() or input() function and the end of file is reached. |
| 12 | **ImportError**<br>Raised when an import statement fails. |
| 13 | **KeyboardInterrupt**<br>Raised when the user interrupts program execution, usually by pressing Ctrl+c. |
| 14 | **LookupError**<br>Base class for all lookup errors. |
| 15 | **IndexError**<br>Raised when an index is not found in a sequence. |
| 16 | **KeyError**<br>Raised when the specified key is not found in the dictionary. |
| 17 | **NameError**<br>Raised when an identifier is not found in the local or global namespace. |
| 18 | **UnboundLocalError**<br>Raised when trying to access a local variable in a function or method but no value has been assigned to it. |
| 19 | **EnvironmentError**<br>Base class for all exceptions that occur outside the Python environment. |
| 20 | **IOError**<br>Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist. |
| 21 | **IOError**<br>Raised for operating system-related errors. |
| 22 | **SyntaxError** |
| | Raised when there is an error in Python syntax. |
| 23 | **IndentationError**<br>Raised when indentation is not specified properly. |
| 24 | **SystemError**<br>Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit. |
| 25 | **SystemExit**<br>Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit. |
| 26 | **TypeError**<br>Raised when an operation or function is attempted that is invalid for the specified data type. |
| 27 | **ValueError**<br>Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified. |
| 28 | **RuntimeError**<br>Raised when a generated error does not fall into any category. |
| 29 | **NotImplementedError**<br>Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented. |

"The built-in exception classes can be subclassed to define new exceptions; programmers are encouraged to derive new exceptions from the Exception class or one of its subclasses, and not from BaseException. More information on defining exceptions is available in the Python Tutorial under User-defined Exceptions."
https://docs.python.org/3/tutorial/errors.html#tut-userexceptions