Exceptions in Python

Exception causes the code to stop if there is an error, for example if you input a string for what is supposed to be an integer, it will show “ValueError” as followed.

```
a = int(input())
print(a)

# HelloWorld
# Traceback (most recent call last):
#   File "d:\LSC\Teams\Computer Team\testing.py", line 1, in <module>
#     a = int(input())
# ValueError: invalid literal for int() with base 10: 'HelloWorld'
```

Prevention:

Try and except branch can help

```
try:
    a = int(input())
    print(a)
except:
    print("this is not an integer")
```

Once an error occurred in the try branch, it will go to the except branch, the terminal doesn’t show any error.

Either of those is executed successfully, the program continues.

We can also specify the error

```
try:
    a = float(input())
    print(1 / a)
except ValueError:
    print("Please input a number")
except ZeroDivisionError:
    print("Please input a non-zero number")
```

Still like if-elif branch you can add except at the end

```
try:
    a = float(input())
    print(1 / a)
except ValueError:
    print("Please input a number")
except ZeroDivisionError:
    print("Please input a non-zero number")
except:
    print("Something unexpected happened")
```

Normal errors

```
short_list = [1]
short_list = [1]
short_list.append(2)
short_list.depend(3) # AttributeError

short_list = [1]
one_value = short_list[0.5] # TypeError

print(1 / 0) # ZeroDivisionError

s = "abc"
a = int(s) # ValueError

# SyntaxError
```

You can also use the $raise$ keyword to raise specified exception.

```
def func ():
    raise ZeroDivisionError

try:
    func()
except ArithmeticError:
    print("Hello there") # executed
```

Another keyword worth mentioning is $assert$. It evaluates the expression, if it's $True$, or a non-zero numerical value, or a non-empty string, or any other value different than $None$, it won't do anything else. Otherwise, it automatically and immediately raises an exception name $AssertionError$. (We say that the assertion has failed)

```
assert expression
```

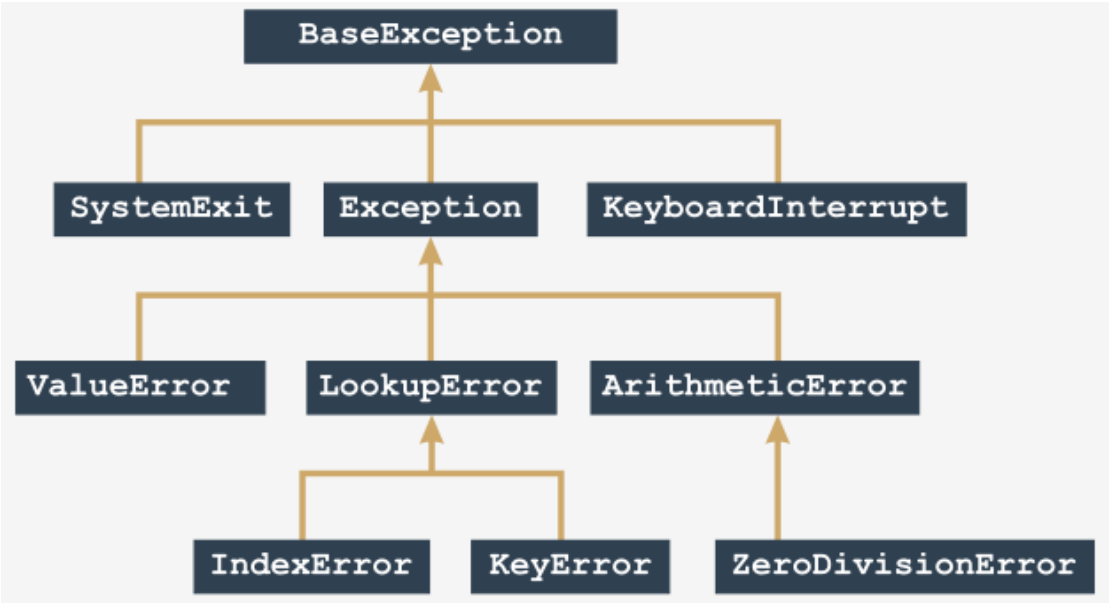
This can make your code absolutely safe from evidently wrong data, and clearly shows the nature of the failure. They also don't supersede exceptions or validate the data.

Exception $KeyboardInterrupt$.

Location: $BaseException$ <- $KeyboardInterrupt$.

A exception raised when the user used a keyboard shortcut designed to terminate a program's execution (Control-C in most OSs).

Python 3 defines 63 built-in exceptions, and all of them form a tree-shaped hierarchy.



Source: Cisco/Python Institute

List of exceptions worth noting:

- $ArithmeticError$ (Abstract built-in Python exception)
- $BaseException$ (Abstract built-in Python exception)
- $LookupError$ (Abstract built-in Python exception)
- $AssertionError$ (Concrete built-in Python exception)
- $ImportError$ (Concrete built-in Python exception)
- $IndexError$ (Concrete built-in Python exception)
- $KeyboardInterrupt$ (Concrete built-in Python exception)
- $KeyError$ (Concrete built-in Python exception)
- $MemoryError$ (Concrete built-in Python exception)
- $OverflowError$ (Concrete built-in Python exception)