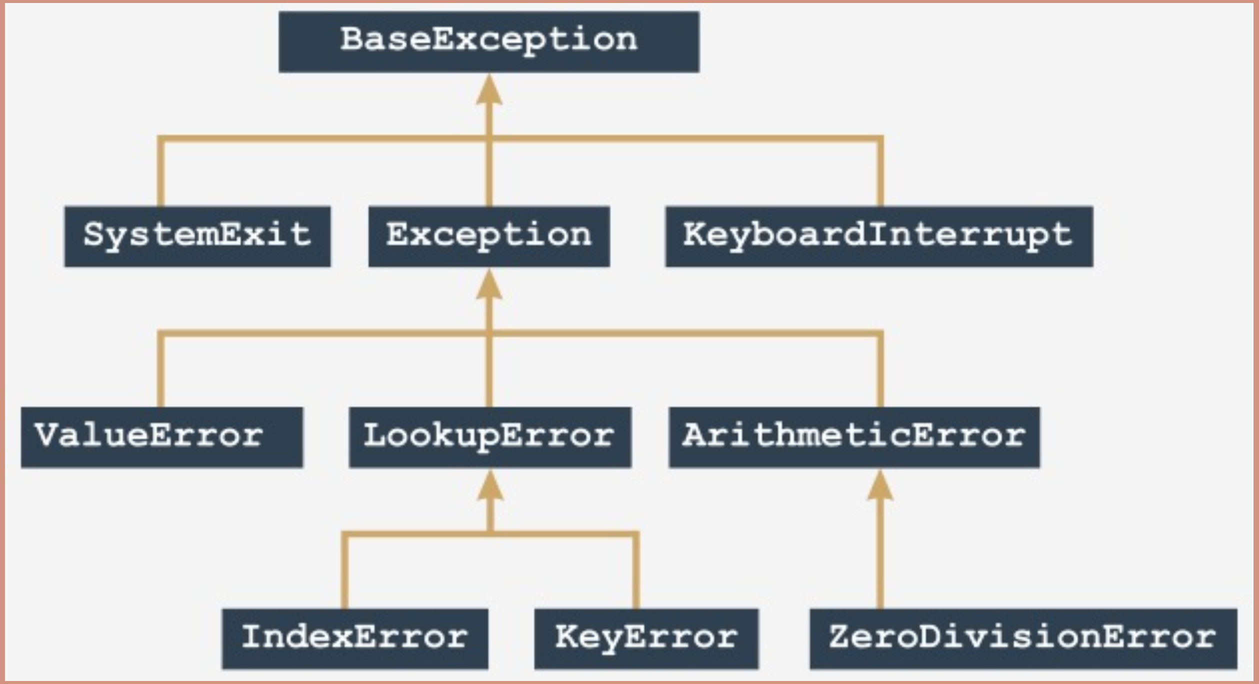
CERTIFIED ASSOCIATE IN PYTHON PROGRAMMING

BY: IMRAN

Exceptio ns

- Python 3 defines **63 built-in exceptions**, and all of them form a **tree-shaped hierarchy**, although the tree is a bit weird as its root is located on top.
- Some of the built-in exceptions are more general (they include other exceptions) while others are completely concrete (they represent themselves only). We can say that **the closer to the root an exception is located, the more general (abstract) it is**. In turn, the exceptions located at the branches' ends (we can call them **leaves**) are concrete.

Exceptions



BaseException

SystemExit

Exception

KeyboardInterrupt

ValueError

LookupError

ArithmeticError

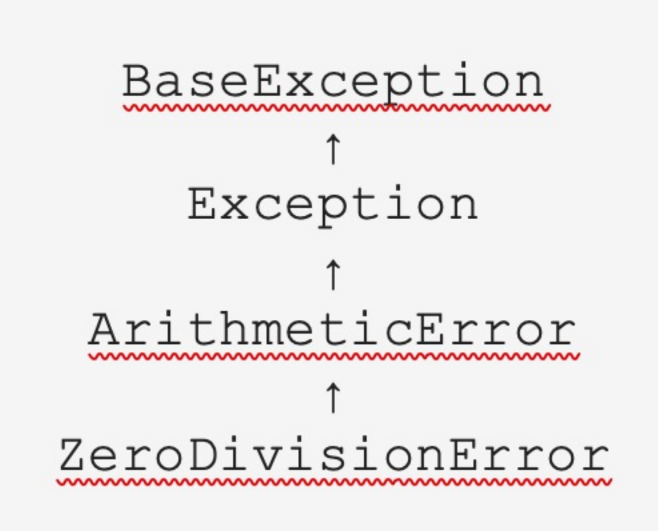
IndexError

KeyError

ZeroDivisionError

Exceptions

- Note:
- `ZeroDivisionError` is a special case of more a general exception class named `ArithmeticError`;
- `ArithmeticError` is a special case of a more general exception class named just `Exception`;
- `Exception` is a special case of a more general class named `BaseException`;
- We can describe it in the following way (note the direction of the arrows - they always point to the more general entity):



```
   BaseException
         ↑
     Exception
         ↑
  ArithmeticError
         ↑
ZeroDivisionError
```

Exceptions

- Look at the code in the editor. It is a simple example to start with. Run it.
-

```
try:
```

- `    y = 1 / 0`
- `except ZeroDivisionError:`
- `    print("Oooppsss...")`
- `print("THE END.")`

Exceptions

- Something has changed in it - we've replaced ZeroDivisionError with ArithmeticError.
- You already know that ArithmeticError is a general class including (among others)
  the ZeroDivisionError exception.
- Thus, the code's output remains unchanged.

```
try:
    y = 1 / 0
except ArithmeticError:
    print("Oooppsss...")

print("THE END.")
```

Exceptions

- This also means that replacing the exception's name with either Exception or BaseException won't change the program's behavior.
- Let's summarize:
  - each exception raised **falls into the first matching branch**;
  - the matching branch doesn't have to specify the same exception exactly - it's enough that the exception is **more general** (more abstract) than the raised one.
-

Exceptions

```
try:
    y = 1 / 0
except ZeroDivisionError:
    print("Zero Division!")
except ArithmeticError:
    print("Arithmetic problem!")
print("THE END.")
```

```
try:
y = 1 / 0
except ArithmeticError:
print("Arithmetic problem!")
except ZeroDivisionError:
print("Zero Division!")
print("THE END.")
```

Exceptions

- The exception is the same, but the more general exception is now listed first - it will catch all zero divisions too. It also means that there's no chance that any exception hits the `ZeroDivisionError` branch. This branch is now completely unreachable.
- Remember:
  - the order of the branches matters!
  - don't put more general exceptions before more concrete ones;
  - this will make the latter one unreachable and useless;
  - moreover, it will make your code messy and inconsistent;
  - Python won't generate any error messages regarding this issue.

Exceptions

```
def bad_fun(n):
    try:
        return 1 / n
    except ArithmeticError:
        print("Arithmetic Problem!")
    return None
bad_fun(0)
print("THE END.")
```

- Note: the **exception raised can cross function and module boundaries**, and travel through the invocation chain looking for a matching except clause able to handle it.
- If there is no such clause, the exception remains unhandled, and Python solves the problem in its standard way - **by terminating your code and emitting a diagnostic message**.

Exceptions

- The raise instruction raises the specified exception named exc as if it was raised in a normal (natural) way:
- `raise exc`
-

  Note: raise is a keyword.
  - The instruction enables you to:
  - **simulate raising actual exceptions** (e.g., to test your handling strategy)
  - partially **handle an exception** and make another part of the code responsible for completing the handling (separation of concerns).

Exceptions

```
def bad_fun(n):
    raise ZeroDivisionError
try:
    bad_fun(0)
except ArithmeticError:
    print("What happened? An error?")


print("THE END.")
```

Exceptions

```
• def bad_fun(n):
•     try:
•         return n / 0
•     except:
•         print("I did it again!")
•         raise
• try:
•     bad_fun(0)
• except ArithmeticError:
•     print("I see!")
• print("THE END.")
```

Exceptions

- The `ZeroDivisionError` is raised twice:
- first, inside the try part of the code (this is caused by actual zero division)
- second, inside the except part by the raise instruction.

Exceptions

- Now is a good moment to show you another Python instruction, named assert. This is a keyword.
- assert experssion
-

  How does it work?
  - It evaluates the expression;
  - if the expression evaluates to 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 named AssertionError (in this case, we say that the assertion has failed)

Exceptions

- import math
- x = float(input("Enter a number: "))
- assert x >= 0.0
- x = math.sqrt(x)
- print(x)

Exceptions

- How it can be used?
  - you may want to put it into your code where you want to be **absolutely safe from evidently wrong data**, and where you aren't absolutely sure that the data has been carefully examined before (e.g., inside a function used by someone else)
  - raising an `AssertionError` exception secures your code from producing invalid results, and clearly shows the nature of the failure;
  - **assertions don't supersede exceptions or validate the data** - they are their supplements.

Module 2: Strings, Lists, and Exceptions
Part 7: Exceptions

Module 2: Strings, Lists, and Exceptions
Part 7: Exceptions

Module 2: Strings, Lists, and Exceptions
Part 7: Exceptions

Module 2: Strings, Lists, and Exceptions
Part 7: Exceptions

Module 2: Strings, Lists, and Exceptions
Part 7: Exceptions

Module 2: Strings, Lists, and Exceptions
Part 7: Exceptions

Module 2: Strings, Lists, and Exceptions
Part 7: Exceptions

ImranNust

imran_muet

muhammad-imran-b7865495

Module 2: Strings, Lists, and Exceptions
Part 7: Exceptions

ImranNust

imran_muet

muhammad-imran-b7865495

Module 2: Strings, Lists, and Exceptions
Part 7: Exceptions

Module 2: Strings, Lists, and Exceptions
Part 7: Exceptions

Module 2: Strings, Lists, and Exceptions
Part 7: Exceptions

ImranNust

imran_muet

muhammad-imran-b7865495