List of Errors and Exceptions: (More common ones in bold)

1. **AssertionError**
2. **AttributeError**
3. **EOFError**
4. **FloatingPointError**
5. GeneratorExit
6. **ImportError**
7. **IndexError**
8. **KeyError**
9. KeyboardInterrupt
10. MemoryError
11. **NameError**
12. **NotImplementedError**
13. OSError
14. OverflowError
15. ReferenceError
16. RuntimeError
17. **StopIteration**
18. **SyntaxError**
19. **IndentationError**
20. TabError
21. SystemError
22. SystemExit
23. TypeError
24. **UnboundLocalError**
25. **UnicodeError**

1. AssertionError

1. It appears when an assert statement fails.
2. assert is a keyword used during debugging.
3. It is used extensively for unit testing functions and ensuring they do what you expect them to do.
4. It will return the message after ',' if the condition returns False.
5. Always use it during DSA interviews.

In [6]:
```
assert 2 == 3, "2 is not equal to 3"
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-6-28bbbf4ef9b7> in <module>
----> 1 assert 2 == 3, "2 is not equal to 3"

AssertionError: 2 is not equal to 3
```

In [7]:
```
def add_integers(a, b):
    return a + b
assert add_integers(2,3) == 5, "Incorrect answer"
```

2. AttributeError

1. It appears when attribute assignment or reference fails.

In [13]:
```
# Attribute Assignment Failure
class AttributeErrorDemo:
    def __init__(self):
        self.attribute_a = 0
```

In [16]:
```
attribute_error_demo_instance = AttributeErrorDemo()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-16-b7c6863a4b3a> in <module>
      5
      6 # attribute_b is absent. It will cause an attribute error.
----> 7 attribute_error_demo_instance.attribute_b

AttributeError: 'AttributeErrorDemo' object has no attribute 'attribute_b'
```

attribute_a is present

In [17]:
```
attribute_error_demo_instance.attribute_a
```

Out[17]:
```
0
```

attribute_b is absent.

1. It will cause the attribute error since we are trying to access something which is not present.

In [18]:
```
attribute_error_demo_instance.attribute_b
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-18-59c3afbb33fe> in <module>
----> 1 attribute_error_demo_instance.attribute_b

AttributeError: 'AttributeErrorDemo' object has no attribute 'attribute_b'
```

Using `python __dict__` to see available attributes

1. Available attributes are returned as dictionary of key:value pairs.

In [19]:
```
attribute_error_demo_instance.__dict__
```

Out[19]:
```
{'attribute_a': 0}
```

Using `python dir()` to see all available attributes and methods

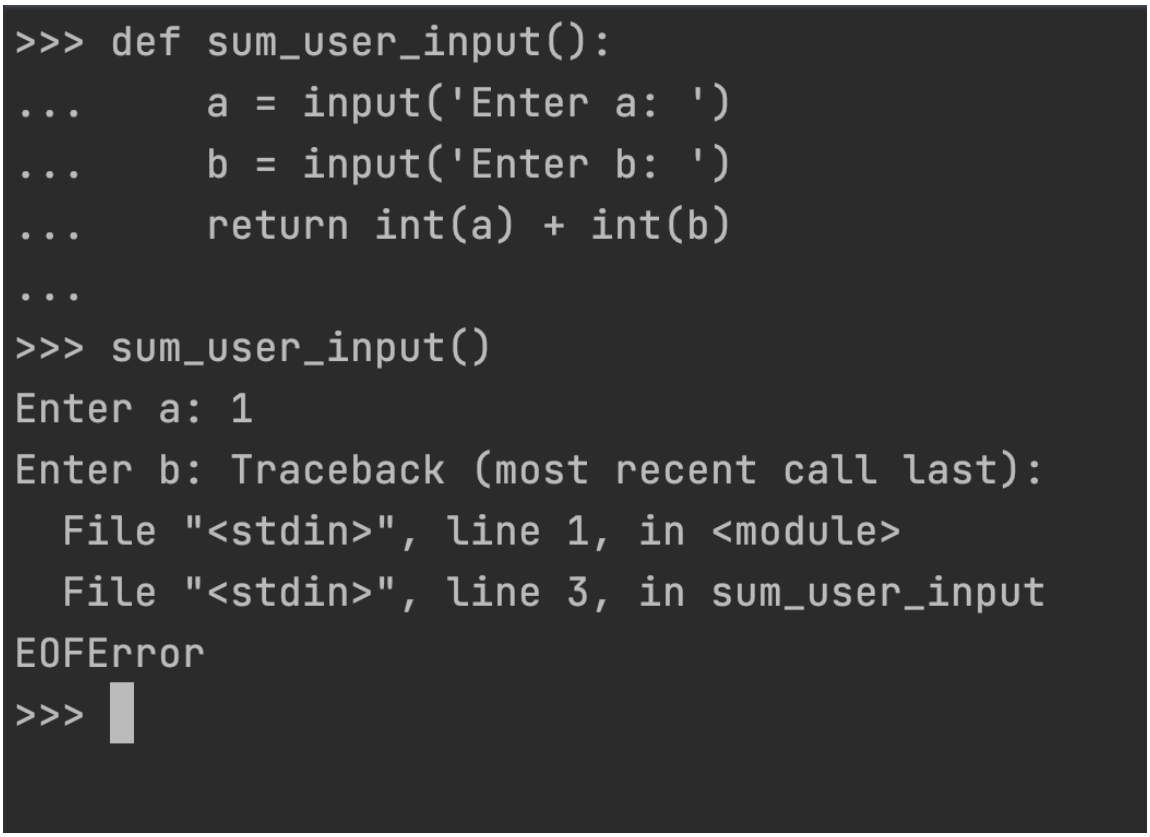
In [20]:
```
dir(attribute_error_demo_instance)
```

Out[20]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
```

```
'__format__',
'__ge__',
'__getattribute__',
'__gt__',
'__hash__',
'__init__',
'__init_subclass__',
'__le__',
'__lt__',
'__module__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'__weakref__',
'attribute_a']
```

3. EOFError

1. It appears Rwhen the input() function hits end-of-file condition.
2. It occurs when an invalid input is provided to the input() function.
3. Refer to this link for more information: https://codefather.tech/blog/python-unexpected-eof-while-parsing/
4. I was unable to product this error in Jupyter Notebook.
5. Attached is screenshot of error in terminal



In [ ]:

In [30]:
```
# Run this code in terminal and hit Ctrl D to exit when prompted for second i
def sum_user_input():
    a = input('Enter a')
```

```
    b = input('Enter b')
    return int(a) + int(b)

# sum_user_input()
```

4. FloatingPointError

1. It occurs when a floating point operation fails.
2. It is a whole world in itself.
3. Refer to https://floating-point-gui.de/basic/ for a more comprehensive understanding.
4. Refer to https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html if you really like pain!
5. This stack overflow link also has great discussions - https://stackoverflow.com/questions/588004/is-floating-point-math-broken
6. Let me show you something unbelievable! How can 0.1 + 0.2 not be equal to 0.3? It is because how these numbers are internally represented.

In [34]:
```
0.1 + 0.2 == 0.3
```

Out[34]:
```
False
```

5. GeneratorExit

1. It calls generator.close().
2. For more details, refer to https://docs.python.org/3/reference/expressions.html#generator.close

In [68]:
```
count = 10
def print_integer(upper_limit):
    try:
        for i in range(0, upper_limit):
            yield i
    finally:
        print("Close is called")
```

In [69]:
```
a = print_integer(3)
```

In [70]:
```
next(a)
```

Out[70]:
```
0
```

In [71]:
```
# generator.exit()
a.close()
```

```
Close is called
```

6. ImportError

1. It appears when the imported module is not found.
2. It is one of the most common erros faced by a beginner.
3. Use pip install to install the module.

4. You can use !pip install to install from inside Jupyter Notebook.

In [1]:
```
import scipy
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-4363d2be0702> in <module>
----> 1 import scipy

ModuleNotFoundError: No module named 'scipy'
```

In [ ]:
```
!pip install scipy
```

7. IndexError

1. It occurs when the index of a sequence is out of range.
2. It is one of the most common erros you will encounter when solving a coding problem due to setting the loop counter incorrectly.
3. It also happens due to incorrect indexing a sequence (string or list).

In [2]:
```
# String
sample_string = "I am a Python coder."
len(sample_string)    # Length is 20. Hence, the indexing is available till l
```

Out[2]:
```
20
```

In [5]:
```
sample_string[19]
```

Out[5]:
```
'.'
```

In [8]:
```
sample_string[20]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-8-09e0e7cee0ea> in <module>
----> 1 sample_string[20]

IndexError: string index out of range
```

In [11]:
```
# List
sample_list = []
sample_list[0]    # Since the list is empty
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-11-3ddb7685c247> in <module>
      1 # List
      2 sample_list = []
----> 3 sample_list[0]    # Since the list is empty

IndexError: list index out of range
```

In [13]:
```
sample_list = [1, 2, 3, 4, 5]
# Looping error in list due to miscalculation of index where list ends
for i in range(len(sample_list)):
```

```
    j = i+1
    print(sample_list[j])    # for the last iteration, j becomes greater than
```

```
2
3
4
5
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-13-fae4358f2833> in <module>
      3 for i in range(len(sample_list)):
      4     j = i+1
----> 5     print(sample_list[j])

IndexError: list index out of range
```

8. KeyError

1. It appears when a key is not found in a dictionary.
2. dict.get() method is the safe option when looking for a key in the dictionary.
3. .get() method returns a None value by default if key is absent. It can also return a user-defined value or message.

In [14]:
```
sample_dict = dict()
sample_dict['key_1'] = 'value_1'
```

In [15]:
```
sample_dict['key_1']
```

Out[15]:
```
'value_1'
```

In [16]:
```
sample_dict['key_2']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-16-cc8b3daba665> in <module>
----> 1 sample_dict['key_2']

KeyError: 'key_2'
```

In [20]:
```
# Using .get() method with default return value of None
print(sample_dict.get('key_2'))
```

```
None
```

In [19]:
```
# Using .get() method with user defined value
print(sample_dict.get('key_2', 'key is absent!'))
```

```
key is absent!
```

9. KeyboardInterrupt

1. It appears when the user hits the interrupt key (Ctrl+C or Delete).
2. The script or code that is under execution is stopped.
3. In Jupyter, pressing 'I' twice after selecting the cell stops the code.

In [22]:
```
while True:
```

```
    a = input('Enter a: ')    # Pressing I twice stops the execution
    b = input('Enter b: ')
    if a == 'stop':
        break
    print(a+b)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-22-e545bb9f1333> in <module>
      1 while True:
----> 2     a = input('Enter a: ')
      3     b = input('Enter b: ')
      4     if a == 'stop':
      5         break

~/.pyenv/versions/3.9.1/envs/data_science/lib/python3.9/site-packages/ipykernel/kernelbase.py in raw_input(self, prompt)
    846                 "raw_input was called, but this frontend does not support input requests."
    847             )
--> 848         return self._input_request(str(prompt),
    849             self._parent_ident,
    850             self._parent_header,

~/.pyenv/versions/3.9.1/envs/data_science/lib/python3.9/site-packages/ipykernel/kernelbase.py in _input_request(self, prompt, ident, parent, password)
    890             except KeyboardInterrupt:
    891                 # re-raise KeyboardInterrupt, to truncate traceback
--> 892                 raise KeyboardInterrupt("Interrupted by user") from None
    893             except Exception as e:
    894                 self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user
```

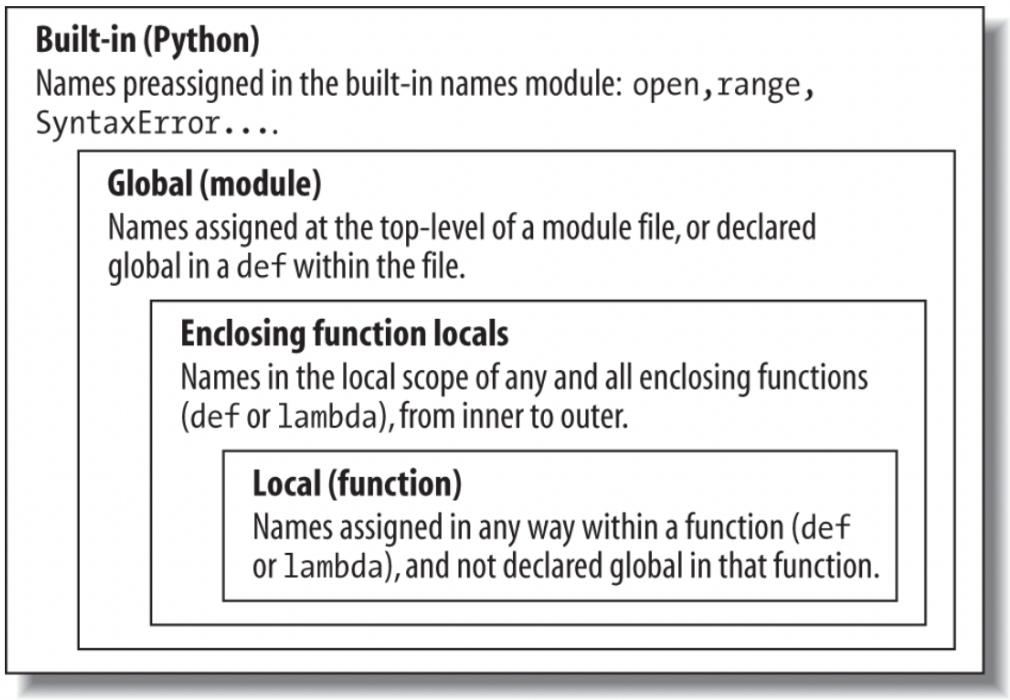
10. MemoryError

1. It occurs when an operation runs out of memory.
2. It may happend during an infinite loop or infinite recursion.
3. It may not be safe to product this error deliberately on a computer.
4. I tried producing this error using math.factorial() of a large number but it kept on freezing.

11. NameError

1. It is raised when a variable is not found in local or global scope.
2. It is again one of the more common erros a beginner with face.

3. Python uses Local-Enclosing-Global-Built-in (LEGB) scoping.



In [24]:
```
print(sample_name)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-c0f090fa820a> in <module>
----> 1 print(sample_name)

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

In [25]:
```
sample_name = 'hello python'
print(sample_name)
```

```
hello python
```

12. NotImplementedError

1. It is raised by abstract methods.
2. When a method is provided for in a class, say to standardise a method name, but the exact implementation is left vacant for the user to customise and implement as per their requirement, it may raise this error.
3. As per documentation, in user defined base classes, abstract methods should raise this exception when they require derived classes to override the method, or while the class is being developed to indicate that the real implementation still needs to be added.
4. For more details, refer to https://stackoverflow.com/questions/44315961/when-to-use-raise-notimplementederror

In [85]:
```
class BaseClassDemo:
    def __init__(self):
        self.result = dict()
    def get(self, key):
        raise NotImplementedError
    def set(self, key, value):
        self.result[key] = value
```

In [86]:
```
a = BaseClassDemo()
a.set('key_1', 'value_1')
```

In [29]:
```
a.result
```

Out[29]:
```
{'key_1': 'value_1'}
```

In [30]:
```
a.get('key_1')
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
<ipython-input-30-64285ca1a9fd> in <module>
----> 1 a.get('key_1')

<ipython-input-27-3cb636f7935f> in get(self, key)
      3         self.result = dict()
      4     def get(self, key):
----> 5         raise NotImplementedError
      6     def set(self, key, value):
      7         self.result[key] = value

NotImplementedError:
```

In [31]:
```
# Creating a derived class which inherits these methods and implements the ge
class DerivedClassDemo(BaseClassDemo):
    def get(self, key):
        return self.result.get(key, 'Key not found')
```

In [33]:
```
b = DerivedClassDemo()
b.set('key_2', 'value_2')
```

In [35]:
```
b.get('key_2')
```

Out[35]:
```
'value_2'
```

In [37]:
```
b.get('key_1')
```

Out[37]:
```
'Key not found'
```

13. OSError

1. It is raised when system operation causes system related error.
2. It is the error class for the os module, which is raised when an os specific system function returns a system-related error, including I/O failures such as “file not found” or “disk full”.
3. It may again be unsafe to produce this error deliberately!

14. OverflowError

1. It appears when the result of an arithmetic operation is too large to be represented.

In [38]:
```
```

```
import math
print("The exponential value is")
print(math.exp(1000))
```

```
The exponential value is
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
<ipython-input-38-7f4ce1496721> in <module>
      1 import math
      2 print("The exponential value is")
----> 3 print(math.exp(1000))

OverflowError: math range error
```

15. ReferenceError

1. It is raised when a weak reference proxy is used to access an attribute of the referent after the garbage collection.
2. Put simply, it helps us understand how referencing inside Python works.

In [46]:

```
import gc
import weakref

class ReferenceDemo(object):

    def __init__(self, value):
        self.value = value

    def __del__(self):
        print('(Deleting %s as sample_reference_instance assigned to some oth
```

In [47]:

```
sample_reference_instance = ReferenceDemo('value_1')
sample_reference_instance_proxy = weakref.proxy(sample_reference_instance)

print('Originally, before reassignment and garbage collection:', sample_refer
sample_reference_instance = None    # Reassigning instance to None - causes _
print ('After Reassignment - the weakly referenced proxy also changes and is
       sample_reference_instance_proxy.value)
```

```
Originally, before reassignment and garbage collection: value_1
(Deleting <__main__.ReferenceDemo object at 0x10a8b7130> as sample_reference_i
nstance assigned to some other value)
---------------------------------------------------------------------------
ReferenceError                            Traceback (most recent call last)
<ipython-input-47-25ef8ff81b22> in <module>
      4 print('Originally, before reassignment and garbage collection:', sampl
e_reference_instance_proxy.value)
      5 sample_reference_instance = None    # Reassigning instance to None - c
auses __del__ and garbage collection
----> 6 print ('After Reassignment - the weakly referenced proxy also changes
 and is unable to access its original reference:', sample_reference_instance_p
roxy.value)

ReferenceError: weakly-referenced object no longer exists
```

16. RuntimeError

1. It appears when an error does not fall under any other category.
2. It is like a miscellaneous 'else' statement to capture unforeseen and previously unseen error category.

3. it returns the error message with what went wrong.

17. StopIteration

1. It is raised by next() function to indicate that there is no further item to be returned by iterator.

In [87]:
```
count = 10
def print_integer(upper_limit):
    try:
        for i in range(0, upper_limit):
            yield i
    finally:
        print("Close is called")
```

In [54]:
```
a = print_integer(2)
```

In [55]:
```
next(a)
```

Out[55]:
```
0
```

In [56]:
```
next(a)
```

Out[56]:
```
1
```

In [57]:
```
next(a)
```

```
Close is called
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-57-15841f3f11d4> in <module>
----> 1 next(a)

StopIteration:
```

18. SyntaxError

1. It is raised by parser when syntax error is encountered.
2. It is again one of the most common erros encountered when you are starting to learn coding.

In [59]:
```
sample_syntax_error = "This is a string without closing quotes
```

```
  File "<ipython-input-59-92475ead9f36>", line 1
    sample_syntax_error = "This is a string without closing quotes
                                                                  ^
SyntaxError: EOL while scanning string literal
```

In [60]:
```
sample_syntax_error = "This is a string without closing quotes"
```

19. IndentationError

1. It appears when there is incorrect indentation.

In [62]:
```
for i in range(10):
print(i)
```

```
  File "<ipython-input-62-0c8aafc23d7e>", line 2
    print(i)
    ^
IndentationError: expected an indented block
```

20. TabError

1. It appears when indentation consists of inconsistent tabs and spaces.
2. In many code editors, there is a configuration to setup tabs and its equivalent in spaces.
3. During coding, if you use spaces and tabs inconsistently, i.e. 4 spaces by spacebar but 2 spaces by tab, it will cause this error.
4. Most of the code editors and Jupyter Notebook is smart enough nowadays to figure out that

21. SystemError

1. It occurs when interpreter detects internal error.
2. TypeError Raised when a function or operation is applied to an object of incorrect type.
3. UnboundLocalError Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
4. UnicodeError Raised when a Unicode-related encoding or decoding error occurs.

22. SystemExit

1. Raised by sys.exit([arg]) function.
2. The optional argument arg can be an integer giving the exit or another type of object. If it is an integer, zero is considered “successful termination”.

In [65]:
```
# Demo: sys.exit()
import sys

number_of_wheels = 4

if number_of_wheels < 18:
    # Program will be exited
    sys.exit("The vehicle is not a car!")
else:
    print("Vehicle appears to be a car")
```

```
An exception has occurred, use %tb to see the full traceback.

SystemExit: The vehicle is not a car!
/Users/paruljuniwal/.pyenv/versions/3.9.1/envs/data_science/lib/python3.9/site
-packages/IPython/core/interactiveshell.py:3445: UserWarning: To exit: use 'ex
it', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

23. TypeError

1. It is raised when a function or operation is applied to an object of incorrect type.

In [66]:
```
# Indexing an integer
a = 1
a[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-66-760df5eaa39c> in <module>
      1 a = 1
----> 2 a[0]

TypeError: 'int' object is not subscriptable
```

In [67]:
```
# Concatenating string with integer
x = 'a'
y = 2
x + y
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-67-bbf2c5980c1a> in <module>
      1 x = 'a'
      2 y = 2
----> 3 x + y

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

24. UnboundLocalError

1. It is raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.

In [74]:
```
def sample_local_error(val):
    if val <= 5:
        print(message)
    else:
        message = "val is greater than 5"
        print(message)
```

In [75]:
```
# The function above thows unbound local error for val <= 5 while it works fi
sample_local_error(2)
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-75-2397661056e9> in <module>
      1 # The function above thows unbound local error for val < 5 while it wo
rks fine for val > 5.
----> 2 sample_local_error(2)

<ipython-input-74-bbea1c6ed068> in sample_local_error(val)
      1 def sample_local_error(val):
      2     if val <= 5:
----> 3         print(message)
      4     else:
      5         message = "val is greater than 5"

UnboundLocalError: local variable 'message' referenced before assignment
```

In [77]:
```
sample_local_error(6)
```

```
val is greater than 5
```

25. UnicodeError

1. It occurs when a Unicode-related encoding or decoding error occurs.
2. As a data scientist, when loading raw data from internet or reading from a file, you will encounter this error very frequently.
3. This exception is a subclass of ValueError.
4. Often, functions will have a parameter of encoding or encode which can be set to 'utf-8' or other desirable encoding to resolve this error.

26. ValueError

1. It is raised when built-in operation or function receives an argument that has the right type but an invalid value.
2. One of the most common usage would be string to int conversion.

In [79]:
```
print(int('xyz'))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-79-127645585295> in <module>
----> 1 print(int('xyz'))

ValueError: invalid literal for int() with base 10: 'xyz'
```

27. ZeroDivisionError

1. It is raised when the second argument of a division or modulo operation is zero.

In [81]:
```
2 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-81-8b4ac6d3a3e1> in <module>
----> 1 2 / 0

ZeroDivisionError: division by zero
```

In [ ]:
```
```