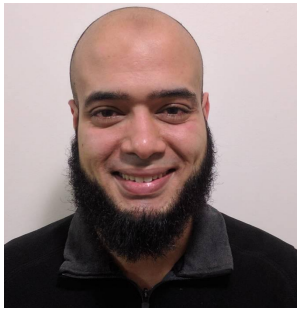
Python Programming

Multiple Exceptions Handling

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Multiple Exceptions

- Consider this file (data.txt) and the code
- Figure Out as much as u could from the exceptions that may occur during the run time

```
3   path, idx = input().split()
4   idx = int(idx)
5
6   file = open(path, 'r')
7   lst = file.read().splitlines()
8   print(lst[idx])
9
10  file.close()
11  """
12  How many possible exceptions?
13
```

data.txt

```
1   10
2   20
3   30
```

Multiple Exceptions

```
3   path, idx = input().split()
4   idx = int(idx)
5
6   file = open(path, 'r')
7   lst = file.read().splitlines()
8   print(lst[idx])
9
10  file.close()
11  """
12  How many possible exceptions?
13
```

```
data.txt 1          ==> 1
data.txt            ==> ValueError: unpack
not_exist.txt 1     ==> FileNotFoundError
/boot/efi/ 1        ==> PermissionError
data.txt hey        ==> ValueError
data.txt 1000       ==> IndexError
data.txt -1000      ==> IndexError
data.txt -1         ==> 30
```

Multiple Exceptions Handling

- We can use except <**specific exception**>

```
try:
    path, idx = input().split()
    idx = int(idx)

    file = open(path, 'r')
    lst = file.read().splitlines()
    print(lst[idx])

    file.close()
except ValueError:
    print('ValueError')
except IndexError:
    print('IndexError')
except FileNotFoundError:
    print('FileNotFoundError')
except:
    print('Something else')
```

Grouping exceptions

- To group exceptions: use a **tuple** of exceptions

```
2   try:
3       path, idx = input().split()
4       idx = int(idx)
5
6       file = open(path, 'r')
7       lst = file.read().splitlines()
8       print(lst[idx])
9
10      file.close()
11
12  except (ValueError, IndexError):    # observe ()
13      print('ValueError or IndexError')
14  except FileNotFoundError:
15      print('FileNotFoundError')
16  except:
17      print('Something else')
```

Proper resources Handling

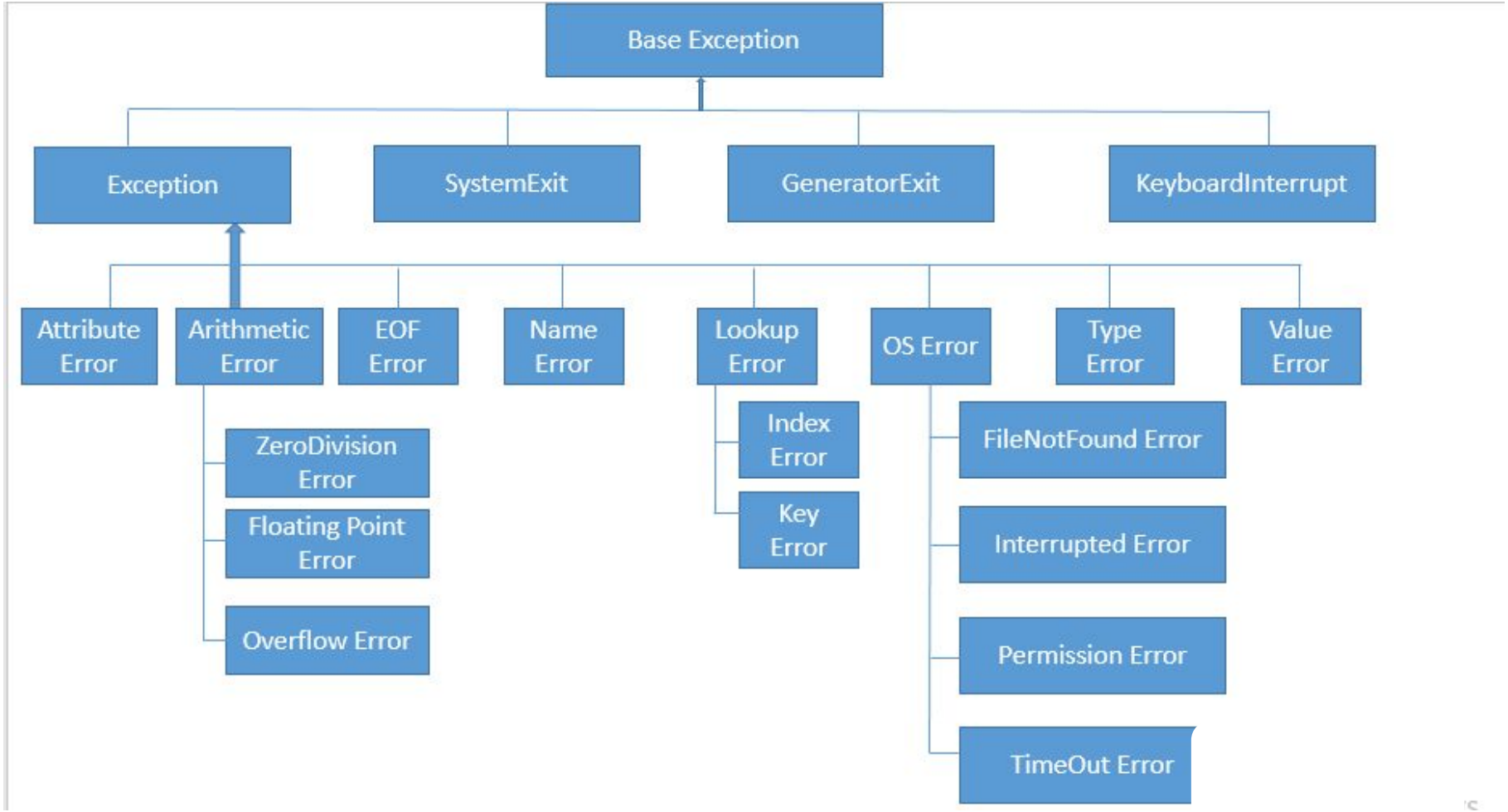
- Utilize the finally block!

```
2   file = None
3   try:
4       path, idx = input().split()
5       idx = int(idx)
6
7       file = open(path, 'r')
8       lst = file.read().splitlines()
9       print(lst[idx])
10
11  except OSError: # cover all sub-types
12      print('Catch all OS errors')
13  except:
14      print('Something else')
15
16  finally:
17      # In all previous codes we wrongly handled it
18      if file is not None:
19          file.close()
```

Use with statement with files

- Recall: With statement closes the file always if it was opened
  - Even with exceptions
- Also observe new syntax: BaseException, as e, str(e)

```
1
2   try:
3       path, idx = input().split()
4       idx = int(idx)
5
6       with open(path, 'r') as file:
7           lst = file.read().splitlines()
8           print(lst[idx])
9           # File will ALWAYS be closed
10  except BaseException as e:      # same as except without class
11      # as e: e is the created exception object
12      error = str(e)  # get the error msg
13      print(error)
14
```


Base Exception
Exception
SystemExit
GeneratorExit
KeyboardInterrupt
Attribute Error
Arithmetic Error
EOF Error
Name Error
Lookup Error
OS Error
Type Error
Value Error
ZeroDivision Error
Floating Point Error
Overflow Error
Index Error
Key Error
FileNotFound Error
Interrupted Error
Permission Error
TimeOut Error

Img [src]

Reading

- Built-in Exceptions (Exception hierarchy)
  - Short Summary
  - Skip what you don’t understand!
- Future reading: Using Python errno
  - We can actually get a number (**error code**) per exception
  - Caution: codes are platform dependent
- Future reading: Raising an Exception to Another Exception

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."