Exceptions VI

The `else` block, the `finally` block and nested `try/except` blocks

In this notebook, we will look at two additional blocks that are available in the `try/except` construct.

We have so far looked at:

1. the try block
2. the except block, and
3. multiple except blocks

There are two more blocks that we can include when handling exceptions.

1. else block: The else block is optional but it is sometimes easier and clearer to write code using the else block. The else block is also useful because it ensures that the statements in the try block are first executed. If an exception occurs, the statements in the except block is executed. If there are multiple except blocks, the appropriate except block is executed. The statements in the else block is executed only if there are no exceptions.
2. finally block: The statements in the finally block will be executed in all cases

Write a function that will ask the user to provides two integers and returns the exponent of the first number raised to the second. That is, given two numbers x and y, the function will return x raised to y. Suppose, we want to do this only for positive x values. If the user enters a negative value, we raise an exception.

In [2]:

```
'''
  Compute Exponent function
'''
import sys
def compute_exponent():
    try:
        n1 = int(input('Enter the first number: '))
        n2 = int(input('Enter the second number: '))
        if n1 <= 0:
            raise ValueError # This forces an execption whenever the user enters a negative value
for n1
        return n1**n2
    except ValueError:
        print('Only positive numbers are valid!!!')
        # sys.exit()

compute_exponent()
```

```
Enter the first number: -2
Enter the second number: 3
Only positive numbers are valid!!!
```

In the example above, the user is asked for n2 even if she enters a negative value for n1. This can be avoided by using the `else` block.

In [4]:

```
'''
  Statements in the else block is executed only if there are no exceptions
  Statements in the finally block will always be executed.
'''
import sys
def compute_exponent():
    try:
        n1 = int(input('Enter the first number: '))
        if n1 <= 0:
            raise ValueError # This forces an execption whenever the user enters a negative value
for n1
    except ValueError:
        print('Only positive numbers are valid!!!')
```

```
        # sys.exit()
    else:
        n2 = int(input('Enter the second number: '))
        print(n1**n2)
    finally:
        print('\nI will always be printed\nFrom the finally block')

compute_exponent()
```

```
Enter the first number: -2
Only positive numbers are valid!!!

I will always be printed
From the finally block
```

Note that we can also have nested `try/except` blocks.

In the following code, we have one `try/except` block to open the file. If the file is opened without error, the `else` block is executed. Note that the `else` block is a part of the first `try/except` code. There is a second `try/except` block inside the `else`.

In [5]:

```
'''
  Nested try/except blocks
  First try/except block: guards against errors encountered while opening the file
  Second try/except block: guards against errors encountered summing values
'''
try:
    f = open('mydata.txt')
except OSError:
    print('Could not open file')
else:
    sum = 0
    try:
        for line in f:
            sum += int(line)
    except Exception as er:
        print(er) # Show the problem
    finally:
        print('This finally block is part of the second (inner) try/except statement')
        f.close() # Close the file
    print('sum =', sum)
finally:
    print('This finally block is part of the first (outer) try/except statement')
```

```
This finally block is part of the second (inner) try/except statement
sum = 72
This finally block is part of the first (outer) try/except statement
```