Objective: This worksheet explains how a Python programmer can write comments in the code that becomes useful for others. The worksheet also explains the exception handling for gracefully handling the errors using ***try/except*** block.

Commenting

- The comments can be written in Python code opening with triple double (or single) quotes and closing with the same triple double (or single) quote. Whatever is written in between becomes the ***docstring***. The ***docstrings*** also need to follow the indentation. In the example below, the ***docstring*** explains what the function does.

```
def factorial(n):
    """
    This function calcuates the factorial of n
    """
    if (n ==0) or (n==1):
        return 1;
    else:
        return n*factorial(n-1)
```

- ***Docstrings*** are particularly useful when someone else wants to read the description of the code as shown in the example below:

```
factorial?
```

```
Signature: factorial(n)
Docstring: This function calcuates the factorial of n
File:      c:\users\bits-pc\desktop\ism\python code\<ipython-input-12-d2daa8d64a21>
Type:      function
```

- Single line commenting can be done using a hash symbol (#). The text after # symbol will be ignored by the Python interpreter.

```
myString = input ("Enter your name")      #Asking the user to enter name
```

Error Handling

- In many situations, the Python code can throw errors. E.g. when a user enters an invalid input. In such situations, we do not want the functionality to break, instead we want to handle these errors gracefully and also want to alert the user. This can be achieved with error handling using ***try/except*** construct.
- In the example below, user is prompted to enter a float value, but user enters a string. When the cell is executed, it throws an error that *"could not convert string to float"*

```
myValue = input ("Enter a float value")
myValue = float(myValue)
```

```
Enter a float value Data Science
```

```
Enter a float valueData Science

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-1-b8d588ff9202> in <module>
      1 myValue = input ("Enter a float value")
----> 2 myValue = float(myValue)

ValueError: could not convert string to float: 'Data Science'
```

- Note that the user cannot be stopped to input unwanted value, but he can be alerted and the error can be handled in such a way that the execution does not break. Observe the ***try/except*** block below:

```
myValue = input ("Enter a float value")
try:
    myValue = float(myValue)
except ValueError:
    print("Enter a float value")
```

try/except block

```
Enter a float valueData Science
Enter a float value
```

- The statement that is susceptible to errors is covered under the ***try*** block and the error that it can throw is captured under the ***except*** block. In the example above, the code can throw ***ValueError***, so it is covered under ***except*** with an alert message to the user.

Python Keyword – None

- The ***None*** keyword is used to define a null value, or no value at all. None is not the same as 0, False, or an empty string.
- The code below throws an error, even with a ***try/except*** block.

```
def getMyValue():
    myValue = input ("Enter a float value")
    try:
        output = float(myValue)
    except ValueError:
        print("Enter a float value")
    return output
```

```
value = getMyValue()
print(value)
```

```
Enter a float valueData Science
Enter a float value

---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-19-c0d2be2fde8b> in <module>
----> 1 value = getMyValue()
      2 print(value)

<ipython-input-18-cba497747a78> in getMyValue()
      5     except ValueError:
      6         print("Enter a float value")
----> 7     return output

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

- Note that the user entered value is being attempted to convert into float and the converted value to store in the variable ***output***. But it does not happen so the value to ***output*** remains unassigned. When output is retuned from the function, it throws an error that "*local variable 'output' referenced before assignment".*
- This can be avoided if the value of output is assigned to ***None*** as shown below:

```
def getMyValue():
    myValue = input ("Enter a float value")
    try:
        output = float(myValue)
    except ValueError:
        print("Enter a float value")
        output = None
    return output
```

Exercise:

In the code snippet below, a user can enter a non-float value as well as a 0. We have learnt that non-float value can be handled using a ***ValueError*** in the ***try/except*** block. But how to gracefully handle divide by zero error? Make the necessary changes to the code to handle ***ZeroDivisionError.*** (Clue: multiple except conditions can be added to the code.)

```
myValue = input ("Enter a float value")
try:
    myValue = float(myValue)
    myValue = 1/myValue
except ValueError:
    print("Enter a float value")
```