Error handling for exceptions

- Error handling for exceptions
  - Debugging procedures can contribute to writing correct codes, exception errors are still likely to occur in your scripts. Exception refer to errors that detected as the script is running.
  - When Python encounters an error, it raises, or throws, an exception. This typically means the script stops running with a runtime error, sometimes also referred to as traceback

- Use try-except statement to handle exceptions

```
try:
    x=input(“First number:”)
    y=input(“Second number:”)
    print x/y
except ZeroDivisionError:
    print “The second number cannot be zero”
except TypeError:
    print “Only numbers are valid entries”
```

- Use try-except statement to handle exceptions

```
try:
    x=input(“First number:”)
    y=input(“Second number:”)
    print x/y
except (ZeroDivisionError, TypeError) as e:
    print e
```

- Use try-except statement to handle exceptions
  - To catch all the exceptions, no matter what type

```
try:
    x=input(“First number:”)
    y=input(“Second number:”)
    print x/y
except Exception as e:
    print e
```

- Use try-except statement to handle exceptions
  - The try-except statement can also include an else statement

```
While True
  try:
      x=input(“First number:”)
      y=input(“Second number:”)
      print x/y
  except Exception as e:
      print e
  else:
      break
```

- Handling geoprocession exceptions
  - When a geoprocessing tool fails to run, it throws an ExecuteError exception. It is not the built-in Python exception classes, but it s generated by ArcPy and thus the arcpy.ExecuteError class has to be used.

```
import arcpy
arcpy.env.workspace=“c:/data”
in_features=“streams.shp”
out_features=“stream.shp”
try:
   arcpy.CopyFeatures_management(in_features, out_features)
except arcpy.ExecuteError:
   print arcpy.GetMessages(2)
except:
   print “There has been a nontool error”
```