User Defined Exceptions:
Some time we have to define and raise exceptions explicitly to indicate that something goes wrong ,such type of exceptions are called User Defined Exceptions or Customized Exceptions.

raise (throw) keyword:
We can raise an exception explicitly with the raise keyword. You can manually throw (raise) an exception in Python with the keyword raise.

Syntax:
```
raise [Exception [, args ]]
```

Exception is the type of exception (for example, NameError) and argument is a value for the exception argument. The argument is optional; if not supplied, the exception argument is None.

Example:
```
>>> raise NameError("Raju")
>>> raise ValueError("InvalidData")
```

Example:
```
x=int(input("Enter Any Number: "))
if x<=10:
    raise ValueError('x should not be less than 10!')
else:
    print("Great Value")
```

Example:
```
try:
    x=int(input("Enter Any Number: "))
    print(x)
except:
    raise ValueError("Invalid Input")
```

Example:
```
try:
    a = int(input("Enter a negative integer: "))
    if a >= 0:
        raise ValueError("That is not a negative number!")
except ValueError as ve:
    print(ve)
```

Assertions in Python(Automatic Error Detection)
Assert is the PYTHON Keyword. Python's assert statement helps you find bugs more quickly and with less pain. When it encounters an assert statement, Python evaluates the accompanying expression, which is hopefully true. If the expression is false, Python raises an AssertionError exception.

Syntax
```
assert expression, argument
```

expression
Required. Expression to evaluate.

argument
Optional. Argument passed to the exception raise

Asserts VS Try...Except:
Software Errors are Two Categories:
1. Recoverable Errors (try ... except)
==> User can take corrective action(Try Again or Choose Another Option)
2. Un-Recoverable Errors(assert)
==> Not Enough information to fix or no alternative action is possible

Places to consider putting assertions:
1 Checking parameter types, classes, or values
2 Checking data structure invariants (never changed)
3 After calling a function, to make sure that its return is reasonable

NOTE:
Assertions are like airbags for your software. Always there, work automatically when you need them.

Example:
```
assert 2 + 2 == 4
assert 2 + 2 == 3
assert 2 + 2 == 3,"That can't be right."
```

Example:
```
def power(x,y):
  assert x>0,"x Must be Positive Number not {0}"
  assert y>0,"y Must be Positive Number not {0}"
  return x**y
print(power(1,-2))
```

Example:
```
def GetAge(age):
    assert age>18,"Age Must not Be less than 18Years"
    print("You are Allow to Access: ",age)
GetAge(19)
```

Ignore Errors
Errors can be ignored without handling them in the program. We can do this using 'pass' in except block of error handling section like below.

Syntax:
```
try:
    data="Something_that_Can_go_wrong"
except:
    pass
```

Example:
```
try:
    pass
except:
    pass
finally:
    pass
```

```
Example:
try:
  x=int(input("Enter Required Number: "))
  y=int(input("Enter Required Number: "))
  print("The Result is: ",int(x/y))
except:
    pass
    print("Error: Arithmetic Operation Ignored. Pass Block")
else:
    print("SuccessfullyDone")

Example:
try:
    x=int(input("Enter Any: "))
    print(x)
except:
    pass
finally:
    pass
```