Exception Handling

Exceptions

Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them:

- **Exception Handling:**
- **Assertions:**

Exceptions

What is Exception?

- An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
- In general, when a Python script encounters a situation that it can't cope with, it raises an exception. An exception is a Python object that represents an error.
- When a Python script raises an exception, it must either handle the exception immediately otherwise it would terminate and come out.

Handling Exceptions

Handling an exception:

- If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

Handling Exceptions

Syntax:

```
try:
  You do your operations here;
  ......................
except Exception I:
  If there is ExceptionI, then execute this block.
except Exception II:
  If there is ExceptionII, then execute this block.
  ......................
else:
  If there is no exception then execute this block.
```

Example

```
try:
  for number in range(10):
    # use a if the number is a multiple of 3, otherwise use b
    if (number % 3) == 0:
        message = message + a
    else:
        message = message + b
    print(message)

except(NameError):
    print("One or more variables are not defined")
```

Example with multiple exceptions

```
try:
    #print(x)
    print("qwe"/2)
except NameError:
    print("Error occured and handeled")
except:
    print("Something went wrong")
```

The *except* clause with no exceptions:

```
try:
    a=b+10
except:
    print("Something is incorrect in try block")
```

Except block with multiple exceptions

```
try:
    a=b+10
except(NameError,ZeroDivisionError,TypeError,ValueError):
    print("Something is incorrect in try block")
```

Try Finally Block

The try-finally clause:

You can use a **finally:** block along with a **try:** block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not. The syntax of the try-finally statement is this:

```
try:
  You do your operations here;
  Due to any exception, this may be
  skipped.
finally:
  This would always be executed.
```

Example

```
try:
    a = int(input("Enter a number (a): "))
    b = int(input("Enter a number (b): "))
except (ValueError,NameError):
    print("Data is not properly read")
finally:
   print("Program Ends")
```

Example

```
try:
    a = int(input("Enter a number (a): "))
    b = int(input("Enter a number (b): "))
except (ValueError,NameError):
    print("Data is not properly read")
    a, b=1, 10
finally:
    print("Values in the range",a," to ",b," are: ")
    for i in range(a,b+1):
        print(i,end=" ")
    print("\nProgram Ends")
```

Else Clause

- try/except statement may include an optional else clause, which appears after all the except clauses
  - Aligned with try and except clauses
  - Syntax similar to else clause in decision structure
  - Else suite: block of statements executed after statements in try suite, only if no exceptions were raised
    - If exception was raised, the else suite is skipped

Example

```
try:
    num = int(input("Enter a number: "))
except:
    print("Not a valid number!")
else:
    reciprocal = 1/num
    print(reciprocal)
```

Raise Exceptions

- The raise statement allows the programmer to force a specified exception to occur.
- The sole argument to raise indicates the exception to be raised.
- A simpler form of the raise statement allows one to re-raise the exception (if you don’t want to handle it):

Example

```
try:
    no = 10 + "Hello"
    raise TypeError
except TypeError:
    print("Unsupported Operation")
```

Example

```
try:
    num = int(input("Enter a positive integer: "))
    if(num <= 0):
# we can pass the message in the raise statement
        raise ValueError("That is  a negative number!")
except ValueError as e:
    print(e)
```