* **Types of Errors**
  + Syntax Errors
  + Run-Time Errors
  + Logical Errors
* **Syntax Errors**
  + Errors that occur during compilation are called Syntax Errors also known as compile-time errors.
  + That means there are some issues with the grammar of the program.
  + It would be reported to you by the interpreter or compiler.
  + You need to correct the program.
    - Some Examples:
      * print “Hello” 🡨 ( ) Missing.
      * city = ‘Ahmedabad’
      * Total = 5 + city 🡨 city contains string, can’t add with 5.
      * print(math.pow()) 🡨 pow() requires min 2 arguments, 0 given.
    - Other Common Syntax Errors
      * Forgetting to put a symbol. such as a colon, comma or brackets
      * Misspelling a keyword
      * Incorrect Indentation
      * Missing :
      * Incorrect number of positional arguments
      * Empty if, else, while, for, function, class, method, etc.
    - ^ indicates the position in the line where an error has been detected.
    - Filename and line numbers are also displayed to help you locate the erroneous statement easily.
* **Run-Time Errors**
  + Errors that occur during execution of a program are known as Run-Time Errors. Also known as Exceptions. It means something unforeseen has happened.
  + It would be reported to the user via Python Runtime.
  + You can overcome these types of errors using Try.
    - Some Examples:
      * a = int(input(“Enter a number.”)
      * b = int(input(“Enter another number.”)
      * c = a / b
      * Memory Related: Stack/Heap overflow, Variable doesn’t exist.
      * Arithmetic Related: Divide By Zero
      * File Handling: File not found, Unable to open the file, Unable to write into a file.
    - The stack trace prints the names of the files, line numbers starting from the first file that got executed, up to the point of exception.
* **Handling Exceptions**
  + Use **try** and **except** block to deal with an exception.
  + Statement(s) which you suspect may go wrong at runtime should be enclosed within a **try** block.
  + An exceptional condition can be tackled in two ways:
    - Pack exception information in an object and raise an exception.
    - Let Python Runtime pack exception-information in an object and raise an exception.
  + Raising an exception is same as throwing an exception in C++/Java.
  + Once an exception is raised, two things can be done:
    - Catch the raised exception object in except block.
    - Raise the exception further.
  + Once we catch the exception object, we can either perform a graceful exit or rectify the exceptional situation and continue execution.
  + If we raise the exception object further, default exception handler catches the object, prints stack trace and terminate.
  + Two ways to create exception objects:
    - From ready-made exception classes (like ZeroDivisionError)
    - From user-defined exception classes
  + Python facilitates exception handling by providing:
    - Keywords **try, except, else, finally, raise**
    - Readymade exception classes.
* **How to use try-except?**
  + **try** block: Enclose in it the code that you anticipate will cause an exception.
  + **except** block: Catch the raised exception in it. It must immediately follow the try block.
  + There can be multiple **except** blocks for one **try** block.
  + If no exception occur, control goes to first line beyond the **except** block.
* **Examples of try-except**

# A normal function without try-except.

def try1():

a = int(input("Enter an integer"))

b = int(input("Enter another integer"))

c = a / b

print ("Division = ", c)

# Example of Divide by zero error.

def try2():

try:

a = int(input("Enter an integer"))

b = int(input("Enter another integer"))

c = a / b

print ("Division = ", c)

except ZeroDivisionError:

print ("Denominator 0 ????")

try3()

# Example of Divide by Zero and value error.

def try3():

try:

a = int(input("Enter an integer"))

b = int(input("Enter another integer"))

c = a / b

print ("Division = ", c)

except ZeroDivisionError as zde:

print ("Denominator 0 ????")

print (zde.args)

x = input("Press a key.")

print (zde)

except ValueError:

print ("Unable to convert string to Int")

except:

print ("Some unknown error")

# Example for Syntax Error

def try4():

try:

date = eval(input("Enter Date:"))

except SyntaxError:

print ("invalid date entered.")

else:

print ("You entered ", date)

# Example for IOError

def try5():

try:

name = input("enter a file name:")

f = open(name,'r')

except IOError:

print ("File not found :", name)

else:

n = len(f.readlines())

print (name, 'has', n, 'lines')

f.close()

* **Types of Exceptions**

|  |  |
| --- | --- |
| **Exception**  **Class Name** | **Description** |
| Exception | Represents any type of exception. All exceptions are sub classes of this class. |
| ArithmeticError | Represents the base class for arithmetic errors like OverflowError, ZeroDivisionError, FloatingPointError. |
| AssertionError | Raised when an assert statement gives error. |
| AttributeError | Raised when an attribute reference or assignment fails. |
| EOFError | Raised when input() function reaches end of file condition without reading any data. |
| FloatingPointError | Raised when a floating point operation fails. |
| GeneratorExit | Raised when generator's close() method is called. |
| IOError | Raised when an input or output operation failed. It raises when the file opened is not found or when writing data disk is full. |
| ImportError | Raised when an import statement fails to find the module being imported. |
| IndexError | Raised when a sequence index or subscript is out of range. |
| KeyError | Raised when a mapping (dictionary) key is not found in the set of existing keys. |
| KeyboardInterrupt | Raised when the user hits the interrupt key (normally Control-C or Delete). |
| NameError | Raised when an identifier is not found locally or globally. |
| NotImplementedError | Derived from 'RuntimeError'. In user defined base classes, abstract methods should raise this exception when they require derived classes to override the method. |
| OverFlowError | Raised when the result of an arithmetic operation is too large to be represented. This cannot occur for long integers (which would rather raise 'MemoryError") |
| RuntimeError | Raised when an error is detected that doesn't fall in any of the other categories. |
| StopIteration | Raised by an iterator's next() method to signal that there are no more elements. |
| SyntaxError | Raised when the compiler encounters a syntax error. Import or exec statements and input() and eval() functions may raise this exception. |
| IndentationError | Raised when indentation is not specified properly. |
| SystemExit | Raised by the sys.exit() function. When it is not handled, the Python interpreter exits. |
| TypeError | Raised when an operation or function is applied to an object of inappropriate datatype. |
| UnboundLocalError | Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable. |
| ValueError | Raised when a built-in operation or function receives an argument that has right datatype but wrong value. |
| ZeroDivisionError | Raised when the denominator is zero in a division or modulus operation. |

* **User-defined Exceptions**
  + Since all exceptional conditions can’t be anticipated, for every exceptional condition there can’t be a class in Python library.
  + In such cases, we can define our own exception.

Write a program that receives 10 integers and stores them and their cubes in a dictionary. If the number entered is less than 3, raise a user-defined exception **NumTooSmall,** and if the number entered is more than 30, then raise a user-defined exception **NumTooBig**. Whether an exception occurs or not, at the end print the contents of the dictionary.

“Just an introduction to a class in below-mentioned program:”

class NumTooSmall(Exception):

def \_\_init\_\_(self, arg):

self.msg = arg

print ("Inside NumTooSmall")

class NumTooBig(Exception):

def \_\_init\_\_(self, arg):

self.msg = arg

print ("Inside NumTooBig")

def try6():

d = {}

for x in range(10):

try:

a = int(input("Enter a number:"))

d[a] = a\*a\*a

if a < 3:

raise NumTooSmall(a)

elif a > 30:

raise NumTooBig(a)

except ValueError:

print("Error")

except NumTooSmall as me:

print (a , "<3")

except NumTooBig as me:

print (a , ">10")

print ("now in try6 ...")

print (d)

* + More on it after learning class-objects.
* **else Block**
  + The **try…except** statement may also have an optional **else** block.
  + If it is present, it must occur after all the except blocks.
  + Control goes to **else** block if no exception occurs during execution of the **try** block.
  + **examples : try4(), try5()**
* **finally Block**
  + Code in **finally** always runs, no matter what!
  + Even if a **return** or **break** occurs first.
  + **finally** block is placed after **except** blocks (if they exist).
  + **try** block must have **except** block and/or **finally** block.
  + **finally** block is commonly used for releasing external resources like files, network connections or database connections, irrespective of whether the use of the resource was successful or not.

def try7():

try:

name = input("enter a file name:")

f = open(name,'a')

a, b = [int(x) for x in input("Enter two numbers").split()]

c = a / b

print (a,b,c)

f.write("Writing %d into file." %c)

except IOError:

print ("File not found :", name)

except ZeroDivisionError as zde:

print ("Denominator 0 ????")

print (zde.args)

finally:

f.close()

print (name,'file closed')

try7()

* **Logical Errors:**
  + Computer can’t detect logical errors. This type of error is due to human being only. Let me explain with one simple example: We want to add two numbers. We wrote a function to do that but instead of addition we used subtraction operator. Now, this type of error can be corrected only and only by the programmers themselves.
* **Write a program for the following:**
  + Write a program that receives an integer an input. If a string is entered instead of an integer, then report an error and give another chance to user to enter an integer. Continue this process till correct input is supplied.