EXCEPTION HANDLING

Learning Objective:

After Having Studied this Chapter, You will be able to Understand : -

- *Types of Errors.*
- *Difference between Error and Exception.*
- *How to Handle Exception.*
- *TRY EXCEPT FINALLY*

6.1 INTRODUCTION

A bug is an error in the software program. ***Debugging*** *is the process of finding the place of error, reason of error, rectifying them and then recompiling code till it is error free*. This chapter discusses various types of errors and exception handling.

6.2 TYPES OF ERRORS

An error is something which will prevent from program to running correctly. Error is also known as bug.

There are 3 types of errors in python.

1. Compile time errors.

2. Run time errors

3. Logical errors

1. COMPILE TIME ERRORS:

The errors that occur at compile time are known as compile time errors. When the program is compiled, its source code is checked if it follows programming language rules or not.

Two types of errors come under compile time errors.

Syntax errors: *Syntax error occurs, when any grammatical rules of programming language are violated.* Syntax error are the most common errors and easier to correct as compiler usually produce a helpful error message that gives an idea about what is wrong in the program.

Example - x = = y* z ; instead of x = y *z ;

x=I Instead of x=i;

Semantic errors:

Semantic errors occur when the statements are not meaningful.

Example x=y*z is the correct statement. This means that the product of y and z is assigned to x.

But if we write

y*z=x.

This statement is meaningless in programming.as x is assigned to product of y and z which is not a variable or any memory location.

2. RUN TIME ERRORS

Errors that occur during execution of program are called Run Times Errors. A run time error causes abnormal program termination during execution.

Example Divide error (occurs when any number is divided by zero.)

Infinite loop.

Consider the code given below. It is causing division by 0 error.

```
a=10
b=0
c=a/0
print(c)

Traceback (most recent call last):
  File "C:/Python37/divide_by_0error.py", line 3, in <module>
    c=a/0
ZeroDivisionError: division by zero
```

A good program should have the appropriate code as preventive measures (known as **guard code**) to handle such operational error. The ability of a program, to recover following an operational error and to continue with its execution is known as **Robustness.**

3. LOGICAL ERRORS

This is the error, which occurs when an algorithm is impleme[nted incorrectly. This could occur if a variable is used without initialization or there is no increment statement when a loop is used, leading to indefinite loop.

```
i=1;

while i<5:
```

```
print i;
```

Another example could be while finding sum of numbers:

```
sum ,a=3;

sum=sum + a;
```

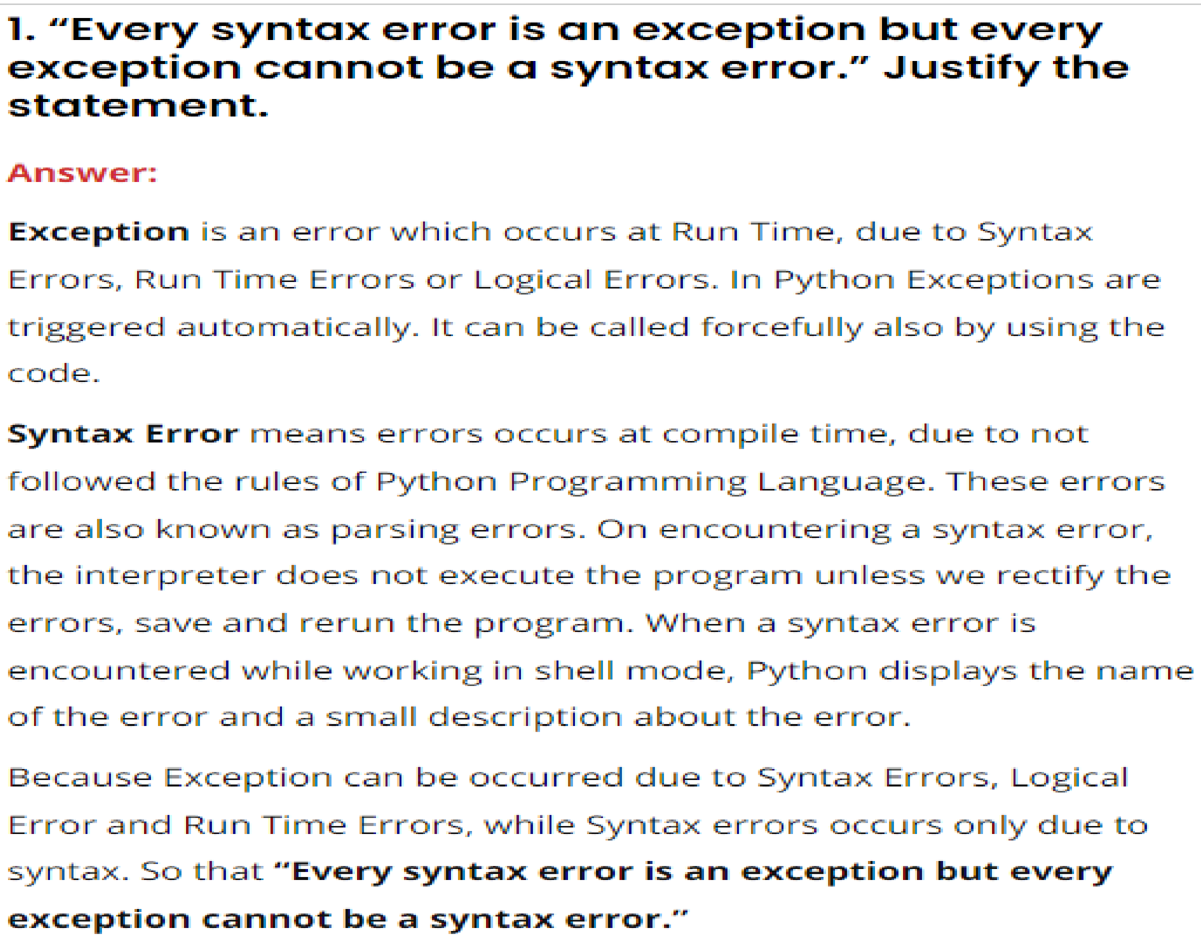
In the above example, since sum value is not initialized to any value, the result may not be proper.

Logical errors may be hard to locate as many at times program would run correctly but will not give the desired output. And also it may not be clear that which statement is causing this kind of output especially when the program is long.

6.3 EXCEPTIONS

Exception Is an error that occur at run time due to syntax, run time or logical error. In python exceptions are triggered automatically but they can be raised by programmer also.

QUESTION



1. "Every syntax error is an exception but every exception cannot be a syntax error." Justify the statement.

Answer:

Exception is an error which occurs at Run Time, due to Syntax Errors, Run Time Errors or Logical Errors. In Python Exceptions are triggered automatically. It can be called forcefully also by using the code.

Syntax Error means errors occurs at compile time, due to not followed the rules of Python Programming Language. These errors are also known as parsing errors. On encountering a syntax error, the interpreter does not execute the program unless we rectify the errors, save and rerun the program. When a syntax error is encountered while working in shell mode, Python displays the name of the error and a small description about the error.

Because Exception can be occurred due to Syntax Errors, Logical Error and Run Time Errors, while Syntax errors occurs only due to syntax. So that **"Every syntax error is an exception but every exception cannot be a syntax error."**

BUILT IN EXCEPTIONS

6.4 SOME BUILT IN EXCEPTIONS IN PYTHON

Exception	Cause of Error
EOFError	Raised when the input() functions hits end-of-file condition.
FloatingPointError	Raised when a floating point operation fails.
ImportError	Raised when the imported module is not found.
IndexError	Raised when index of a sequence is out of range.

KeyError	Raised when a key is not found in a dictionary.
MemoryError	Raised when an operation runs out of memory.
NameError	Raised when a variable is not found in local or global scope.
OverflowError	Raised when result of an arithmetic operation is too large to be represented.
RuntimeError	Raised when an error does not fall under any other category.
SyntaxError	Raised by parser when syntax error is encountered.
IndentationError	Raised when there is incorrect indentation.
SystemError	Raised when interpreter detects internal error.
TypeError	Raised when a function or operation is applied to an object of incorrect type.
ValueError	Raised when a function gets argument of correct type but improper value.
ZeroDivisionError	Raised when second operand of division or modulo operation is zero.

2. When are the following built-in exceptions raised? Give examples to support your answers.
a) ImportError
b) IOError
c) NameError
d) ZeroDivisionError

Answer:

(a) ImportError :- It is raised when the requested module definition is not found.

```
        >>> import maths
        Traceback (most recent call last):
          File "", line 1, in
            import maths
        ModuleNotFoundError: No module named
'maths'

        >>> import randoms
        Traceback (most recent call last):
          File "", line 1, in
            import randoms
        ModuleNotFoundError: No module named
'randoms'
```

(b) IOError :- It is raised when the file specified in a program statement cannot be opened.

```
>>> f = open("abc.txt",'r')
Traceback (most recent call last):
  File "", line 1, in
    f = open("abc.txt",'r')
FileNotFoundError: [Errno 2] No such
file or directory: 'abc.txt'
```

(c) NameError :- It is raised when a local or global variable name is not defined.

```
>>> print(name)
Traceback (most recent call last):
  File "", line 1, in
    print(name)
NameError: name 'name' is not defined
```

(d) ZeroDivisionError :- It is raised when the denominator in a division operation is zero.

```
>>> 10/0
Traceback (most recent call last):
  File "", line 1, in
    10/0
ZeroDivisionError: division by zero
```

3. What is the use of a raise statement?

Write a code to accept two numbers and display the quotient. Appropriate exception should be raised if the user enters the second number (denominator) as zero (0).

Answer: The raise statement can be used to throw an exception.

The syntax of raise statement is: *raise exception-name[(optional argument)]*

The argument is generally a string that is displayed when the exception is raised.

Code to accept two numbers and display the quotient.

```
n = int(input("Enter Number 1 :"))
m = int(input("Enter Number 2 : "))
if m == 0:
    raise ZeroDivisionError
else:
    print("Quotient : ", n / m)
```

4. Use assert statement in Question No. 3 to test the division expression in the program.

Answer: An assert statement in Python is used to test an expression in the program code. If the result after testing
comes false, then the exception is raised. This statement is generally used in the beginning of the function or after
a function call to check for valid input.

The syntax for assert statement is: *assert Expression[,arguments]*

On encountering an assert statement, Python evaluates the expression given immediately after the assert keyword. If this expression is false, an AssertionError exception is raised which can be handled like any other exception.

```
 n = int(input("Enter Number 1 :"))
 m = int(input("Enter Number 2 : "))
 assert (m == 0), "Oops , Zero Division Error
...."
 print("Quotient : ", n / m)
```

Output -1 :

```
Enter Number 1 :10
Enter Number 2 : 2
Traceback (most recent call last):
File "D:/PythonProg/ExceptionHandling/DivideByZero.py", line 4, in
assert (m == 0), "Opps, .... ZeroDivisionError"
AssertionError: Opps, .... ZeroDivisionError
```

Output – 2:

```
Enter Number 1 :10
Enter Number 2 : 0
Traceback (most recent call last):
File "D:/PythonProg/ExceptionHandling/DivideByZero.py", line 5, in
print("Quotient : ", n / m)
ZeroDivisionError: division by zero
```

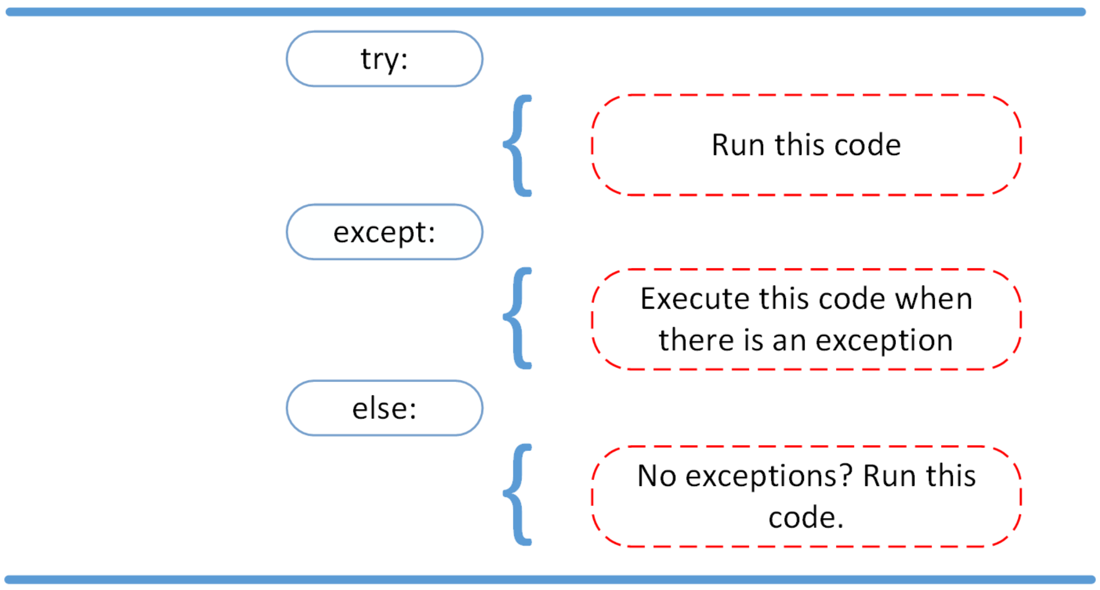
6.3.1 HANDLING AN EXCEPTION

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.

Exception Handling means writing additional code in the program which contains message or proper instructions to be given to the user incase exception has encountered.

Syntax

Here is simple syntax of *try....except...else* blocks –


try:
Run this code
except:
Execute this code when there is an exception
else:
No exceptions? Run this code.

Example:

```
a=20
b=0
try:
   print(a/b)
except ZeroDivisionError:
   print("division by 0 is not allowed")
else:
   print(a*b)
```

Output

```
division by 0 is not allowed
```

In the above example A/B will be equal to infinite and there is no way to represent infinite in our systems.so in **try** we write the possible problem code. Whatever is written in **try** causes program to halt, it will go to the **except** clause and if the statement in **try** runs successfully then the statement in **else** will be executed.

Lets see another example.

Example:

```
a=20
b=10
try:
   print(a/b)
except ZeroDivisionError:
   print("division by 0 is not allowed")
else:
   print(a*b)
```

Output

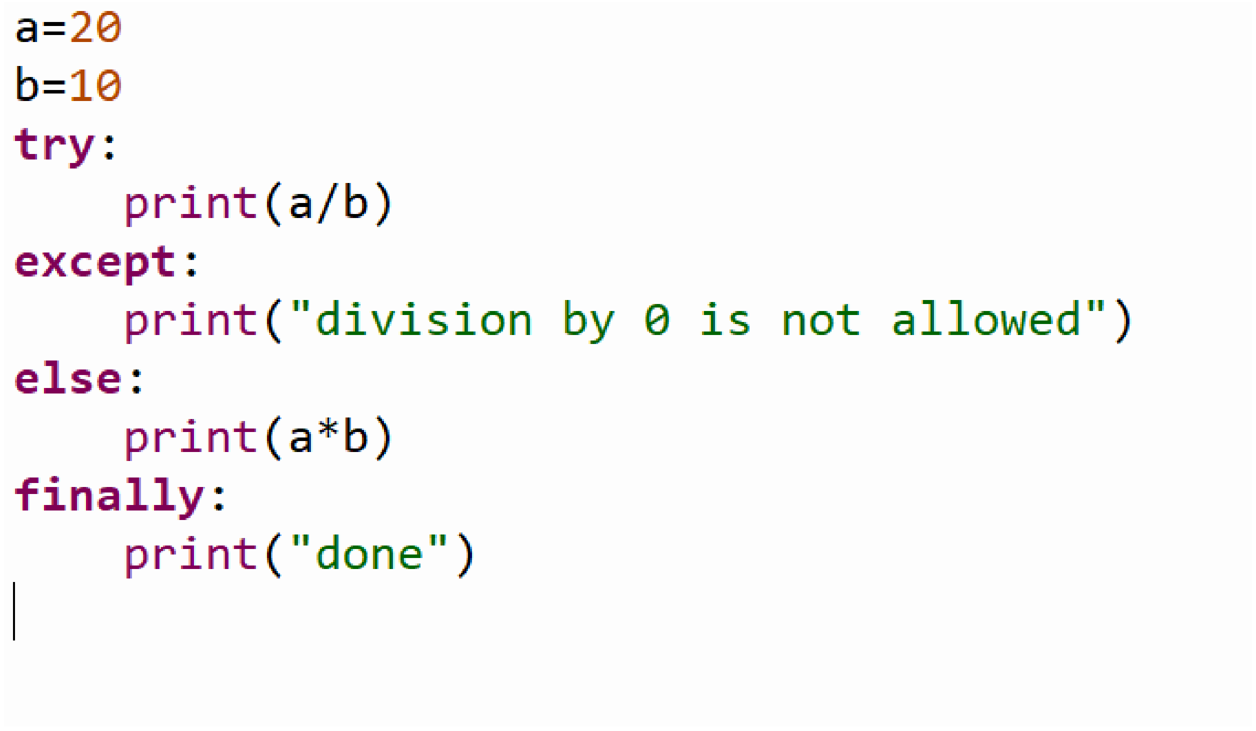
```
2.0

200
```

Considering there was no exception, so the try and else clause was executed.

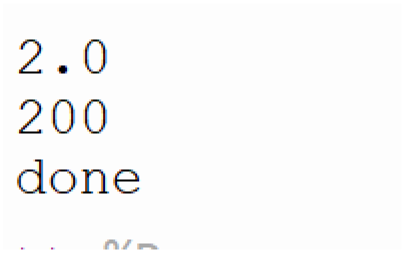
FINALLY

The try statement in Python can also have an optional finally clause. The statements inside the finally block are always executed regardless of whether an exception has occurred in the try block or not. Example



```
a=20
b=10
try:
    print(a/b)
except:
    print("division by 0 is not allowed")
else:
    print(a*b)
finally:
    print("done")
```

Output

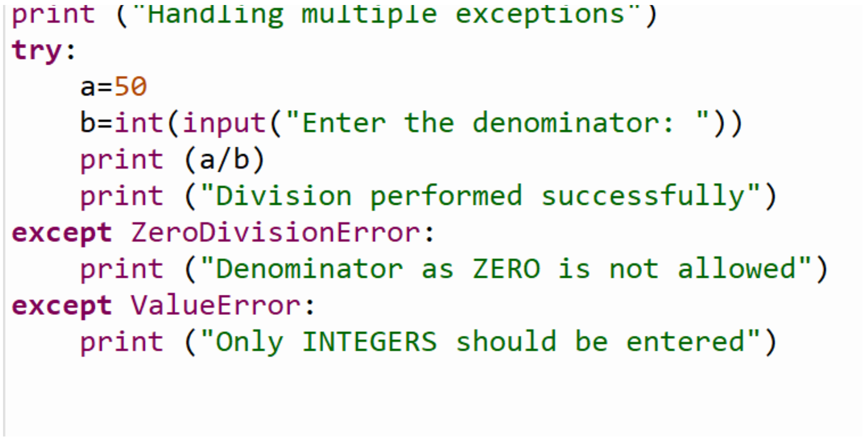


```
2.0
200
done
```

USE OF MULTIPLE EXCEPT CLAUSE

Sometimes, a single piece of code might be suspected to have more than one type of error. For handling such situations, we can have multiple except blocks for a single try block as shown in the Program 1-3.

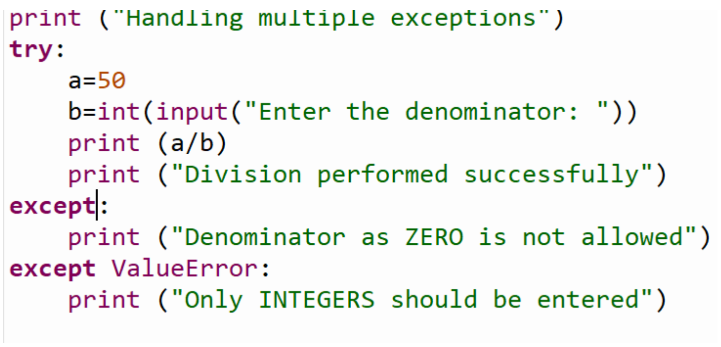
<u>Program Use of multiple except clauses</u>



```
print ("Handling multiple exceptions")
try:
    a=50
    b=int(input("Enter the denominator: "))
    print (a/b)
    print ("Division performed successfully")
except ZeroDivisionError:
    print ("Denominator as ZERO is not allowed")
except ValueError:
    print ("Only INTEGERS should be entered")
```

In the code, two types of exceptions (ZeroDivisionError and ValueError) are handled using two except blocks for a single try block. When an exception is raised, a search for the matching except block is made till it is handled. If no match is found, then the program terminates. However, if an exception is raised for which no handler is created by the programmer, then such an exception can be handled by adding an except clause without specifying any exception.

<u>Program Use of except without specifying an exception</u>



```
print ("Handling multiple exceptions")
try:
    a=50
    b=int(input("Enter the denominator: "))
    print (a/b)
    print ("Division performed successfully")
except:
    print ("Denominator as ZERO is not allowed")
except ValueError:
    print ("Only INTEGERS should be entered")
```

If the above code is executed, and the denominator entered is 0 (zero) , the handler for ZeroDivisionError exception will be searched. Since it is not present, the last except clause (without any specified exception) will be executed , so the message “ OOPS.....SOME EXCEPTION RAISED” will be displayed.

7. Consider the code given below and fill in the blanks.

```
print (" Learning Exceptions…")
try:
    num1= int(input (" Enter the first number")
    num2= int(input("Enter the second number"))
    quotient=(num1/num2)
    print ("Both the numbers entered were
correct")
except ________: # to enter only integers
    print (" Please enter only numbers")
except __________: # Denominator should not be
zero
    print(" Number 2 should not be zero")
else:
    print(" Great .. you are a good
programmer")
___________________ : # to be executed at the
end
print(" JOB OVER… GO GET SOME REST")
```

Answer:

```
print (" Learning Exceptions…")
try:
    num1= int(input (" Enter the first number")
    num2= int(input("Enter the second number"))
    quotient=(num1/num2)
    print ("Both the numbers entered were
correct")
except ValueError: # to enter only integers
    print (" Please enter only numbers")
except ZeroDivisionError: # Denominator should
not be zero
    print(" Number 2 should not be zero")
else:
    print(" Great .. you are a good
programmer")
finally : # to be executed at the end
print(" JOB OVER… GO GET SOME REST")
```

8. You have learnt how to use math module in Class XI. Write a code where you use the wrong number of arguments for a method (say sqrt() or pow()). Use the exception handling process to catch the ValueError exception.

Answer:

Code to show the wrong number of arguments with exception handling.

```
 import math
 try:
     print(math.sqrt(25,6))
 except TypeError:
     print("Wrong number of arguments used in
sqrt() ")
 finally:
     print("Okay, Correct it")
```

Output:

Wrong number of arguments used in sqrt()

Okay, Correct it

9. What is the use of finally clause? Use finally clause in the problem given in Question No. 7.

Answer: The try statement in Python can also have an optional finally clause. The statements inside the finally block are always executed regardless of whether an exception has occurred in the try block or not. It is a common practice to use finally clause while working with files to ensure that the file object is closed. If used, finally should always be placed at the end of try clause, after all except blocks and the else block.

Difference between error and an exception.

Exceptions are those which can be handled at the run time whereas errors cannot be handled. An exception is an Object of a type deriving from the System, Exception class. An Error is something that most of the time you cannot handle it.

Points To Remember.

1. A bug is an error in the software program. Debugging is the process of finding the place of error, reason of error, rectifying them and then recompiling code till it is error free.
2. An error is something which will prevent from program to running correctly. Error is also known as bug.
3. There are 3 types of errors in python.
   - Compile time errors.
   - Run time errors
   - Logical errors

4. The errors that occur at compile time are known as compile time errors.
5. Syntax error occurs, when any grammatical rules of programming language are violated.
6. Semantic errors occur when the statements are not meaningful.
7. Errors that occur during execution of program are called Run Times Errors. A run time error causes abnormal program termination during execution.
8. Logical Errors occurs when an algorithm is implemented incorrectly. This could occur if a variable is used without initialization or there is no increment statement when a loop is used, leading to indefinite loop.
9. An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
10. 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.
11. Exceptions are those which can be handled at the run time whereas errors cannot be handled.

6.8 SOLVED QUESTIONS

1.	What is the difference between exception and error in python? How an exception is handled in a python program?

Ans	Error: Errors sometimes called a bug, is anything in the code that prevents a program from compiling and running correctly. They are broadly three types: Compile time, Runtime, and Logical errors.
	Exception: An error at the run time is known as exception.
	An exception can be handled by **try** and **except** clauses in python.
	Ex: `try:`
	`    Print(a/b)`
	`  except:`
	`    print("Divide by Zero error")`

2.	What will be the output of following Python code? Justify your answer. x = 5 y = 0 **print** ('A') **try**: **print** ('B') A = x/y **print** ('C') **except ZeroDivisionError**:

	print ('F') except : print ('D') finally: print ('over')
Ans	The code will produce the following output: A B F over

Unsolved Questions

1. What are main error types? Which types are most dangerous and why?

2. What is a difference between an error and exception?

3. Name some common built-in exceptions in Python. When does these exception occur?

(a) Type Error (b) Index Error (c) Name Error

4. What is debugging and code tracing?

5. What do you understand by Syntax errors and Semantics errors?

6. Why are logical errors harder to locate?

7. What is an Exception?

8. Why Exception Handling is is required?

9. What is the need for debugger tool?