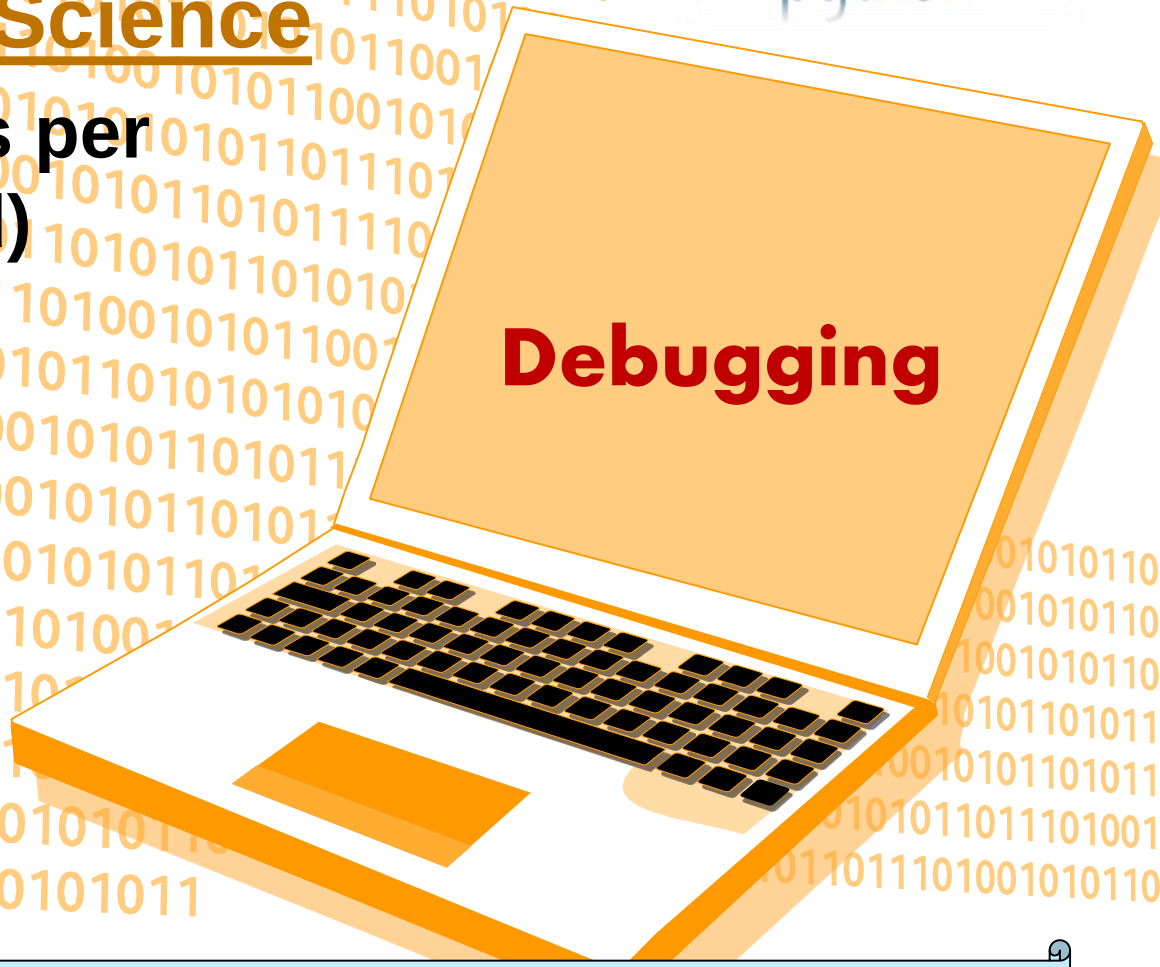
Chapter 9 :

Computer Science

Class XI (As per CBSE Board)



New Syllabus 2019-20

Visit : python.mykvs.in for regular updates

Errors and Exceptions

In Python, there are two kinds of errors: syntax errors and exceptions.

A syntax error is an error in the syntax of a sequence of characters or tokens that is intended to be written in a particular programming language.

e.g.

```
>>> while True print 'Hello world'
SyntaxError: invalid syntax
```

Errors and Exceptions

The other kind of errors in Python are exceptions.

Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it.

Errors detected during execution are called exceptions.
e.g.

```
 >>> 10 * (1/0)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    10 * (1/0)
ZeroDivisionError: integer division or modulo by zero
```

Errors and Exceptions

Standard Exceptions available in Python are

Exception, SystemExit,OverflowError, FloatingPointError, ZeroDivisonError, EOFError, KeyboardInterrupt, IndexError, IOError, SyntaxError , IndentationError , SystemExit , ValueError, TypeError , RuntimeError

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

Errors and Exceptions

try

Syntax:

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

e.g.

```
 try:
   fh = open("testfile", "r")
   fh.write("This is my test file for exception handling!!")
except IOError:
   print ("Error: can\'t find file or read data")
else:
   print ("Written content in the file successfully")
```

Debugging

• “print line debugging”

– At various points in your code, insert print statements that log the state of the program

• You will probably want to print some strings with some variables

• You could just join things together like this:

```
>>>x=9
>>>print 'Variable x is equal to ' + str(x)
Output : Variable x is equal to 9
```

• … but that gets unwieldy pretty quickly

• The format function is much nicer:

```
>>>x=3
>>>y=4
>>>z=9
>>>print 'x, y, z are equal to {}, {}, {}'.format(x,y,z)
Output : x, y, z are equal to 6, 4, 8
```

Debugging

• Python Debugger: pdb
– insert the following in your program to set a breakpoint
– when your code hits these lines, it'll stop running and launch an interactive prompt for you to inspect variables, step through the program, etc.

```
import pdb
pdb.set_trace()
```

n to step to the next line in the current function
s to step into a function
c to continue to the next breakpoint
you can also run any Python command, like in the interpreter

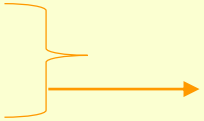
Debugging

Create a.py file with below code and run it in python use n to step next line.

```
num_list = [500, 600, 700]
alpha_list = ['x', 'y', 'z']

import pdb
pdb.set_trace()          #debugging code
def nested_loop():
    for number in num_list:
        print(number)
        for letter in alpha_list:
            print(letter)

if __name__ == '__main__':
    nested_loop()
```



While executing above code whole program will be traced.

Another way is to invoke the pdb module from the command line.

```
$ python -m pdb mycode.py
```