CS331

PROGRAMMING LANGUAGE LAB

Group 7 -

1. *B.Sathya Prakash (1401CS06)*
2. *Anupam Das (1401CS52)*
3. *Jitendra Kumar (1401CS19)*

INTRODUCTION

- Full name of the language is **Python**.
- Python is a widely used high-level, general-purpose, interpreted, dynamic programming language.
- Python was conceived in the late 1980s, and its implementation began in December 1989 by its creator **Guido van Rossum** at Centrum Wiskunde & Informatica (CWI) in the Netherlands.
- Many of Python's features originated from an interpreted language called ABC.
- Python was first released in February 1991 and was developed by **Python Software Foundation.**
- Python's design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java.
- Python is influenced by languages like ABC,C,C++,Haskell,ALGOL 68,Java,etc.
- Top Websites like Youtube, Dropbox, Google, Quora, Yahoo Maps, Instagram are made upon Python(Django framework).
- Languages influenced by python are Ruby,JavaScript,Falcon,Swift,Nim and many more developing languages.
- Python is also used for web development and for making games. Pygame is the tool which one can use to make game easily and pygame contains all libraries.

- Python 2.0 released in 2000 had major new features
    1. Garbage collector
    2. Support for unicode
- Python 3.0 - major,backwards incompatible release in 2008.

Python Programming Paradigms -

- Python is multi-paradigms language.you can write programs or libraries that are largely procedural, object-oriented, or functional in python.

- The notion of **programming paradigms** is a way to classify programming language according to the style of computer programming.

- Python is an object oriented where one can create different classes and objects(instances of class) and many methods.

- Python is procedural language. Procedural is also known as routines or subroutines. A procedure contains a series of computational steps to be carried out.Procedural programming is also referred to as imperative programming. Procedural programming languages are also known as top-down languages.

How to execute program -

- Major interpretations of python are

  CPython:- CPython is the default, most widely used implementation of the Python programming language. It is written in C. CPython is a source code interpreter.

  Jython:- Jython is a Java implementation of Python that combines expressive power with clarity. Jython is freely available for both commercial and non-commercial use and is distributed with source code.

IronPython:- IronPython is an open-source implementation of the Python programming language which is tightly integrated with the .NET Framework. IronPython can use the .NET Framework and Python libraries, and other .NET languages can use Python code just as easily.

PyPy:- PyPy is a Python interpreter and just-in-time compiler. PyPy focuses on speed, efficiency and compatibility with the original CPython interpreter.

- Then depending on operating system we have different ways of execution.

- Python interpreter program translate Python code into computer instructions.

- ***<u>For linux :-</u>***
    1. *Write code in text file and save it with ".py" extension(eg. first.py).*
    2. *Now open the terminal and change the current directory to the directory where the first.py file is located.*
    3. *Now just run python first.py.*
       $ python <filename.py>
       Eg - $ python first.py
    4. *If python interpreter is installed in computer this command will execute without any error and will give the output otherwise first download python interpreter and then install it and then run above command.*
    5. *In most of the linux systems python interpreter is installed by default.*

- ***For Windows : -***

    1. *Write code in text file and save it with .py extension (eg. first.py).*
    2. *Open command prompt and navigate to directory in which first.py is saved using cd command.*
    3. *Try >python first.py.*
    4. *If you get message like python is not recognized then python interpreter isn't on path.Then try calling it like this (assuming Python2.6, installed in the usual location):*
       > C:\Python26\python.exe first.py
    5. *In order to run program your operating looks in various places and tries to match the name of the program / command you typed with some programs along the way.*
    6. *In windows : control panel > system > advanced > |Environment Variables| > system variables -> Path. this needs to include: C:\Python26; (or equivalent). If you put it at the front, it will be the first place looked. You can also add it at the end, which is possibly saner.Then restart your prompt, and try typing 'python'. If it all worked, you should get a ">>>" prompt.*

- *We can also use PyCharm to executive python code. PyCharm is python ide .*

Rules to construct variables and different keywords in python -

- Variables names must start with a letter or an underscore.
  Eg. _flag , flag_
- The remainder of your variable name may consist of letters, numbers and underscores.
  Eg variable_1 ,_777,etc
- Names are case sensitive. _flag_one , _FLAG_ONE , _Flag_One are each a different variable.
- Variables are used to store some value so each variable takes some space according to it's data type.
- Python variable do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to variable. The equal sign(=) is used to assign a value to variable.Variable assignment works left to right.

  ```
  Eg. var1 = 25 #integer assignment
      var2 = "string" #string assignment
      var3 = 10.25 #float assignment
  ```
- We can also assign same value to multiple variables
  Eg. p=q=r=1;
- We can also assign as follows
  Var1,var2,var3 = 100,"string",10.25
- We can't use any keyword as a variable in python.
- Python swap values in single line and this applies to all objects in python.
  Eg. var1,var2 = var2,var1
- In Python, variables that are only referenced inside a function are implicitly global. If a variable is assigned a value anywhere within the function's body, it's assumed to be a local unless explicitly declared as global.

Keywords in python-

Keywords	Description
True,false	Results of logical and comparison operation in python
None	Absence of value or null value
and,or,not	Logical operators
as	To create alias while importing module
assert	Used for debugging purpose
break	Used to come outside loop
continue	To continue the loop
class	Used to define new user defined class
def	Used to define new method/function
del	Used to delete reference of an object
if,else,elif	Used for conditional branching
except,raise,try	Used with exceptions in python
finally	Used with try..except to close up resource
for,while	Used for looping
global	To declare variable inside function which is global
in	To test sequence contains value or not
is	For testing object identity
lambda	Used to create anonymous function
return	For returning from function
pass	Null statement in python

Type Systems in Python:-

- **Duck Typing :** Duck typing is concerned with establishing the suitability of an object for some purpose. With normal typing, suitability is assumed to be determined by an object's type only. In duck typing, an object's suitability is determined by the presence of certain *methods and properties* (with appropriate meaning), rather than the actual type of the object.

  For example, Consider the code

```
class Person:
def help(self):
      print("Heeeelp!")

class Duck:
def help(self):
      print("Quaaaaaack!")

class SomethingElse:
pass
donald = Duck()
john = Person()
who = SomethingElse()
for thing in [donald, john, who]:
Try:
       InTheForest(thing)
except AttributeError:
      print 'Meeowww!'
```

  Output is :

```
Quaaaaaack!
Heeeelp!
Meeowww!
```

  Here, the equivalent function would take an object of any type and call that object's help method. If the object does not have the method that are called

then the function signals a run-time error. If the object does have the method, then it is executed no matter the type of the object.

- **Dynamic Typing :** A language has dynamic typing when it does not associate values strictly with a specific type, but it is designed to "decide" what the type of a value should be at runtime, based on how you are attempting to use it.

  For example,

```
Count = "5";   //defines a string variable
Count = count * 2   //this is legal and has obvious result

Var = 11
Var  = " programming languages"
// It will print programming languages
```

  Here, the compiler did not complain that you are attempting to multiply a string by a number and refuse to compile the program (such as would happen in languages like C, C++, C# and Java). It produced code to forward the arguments `count` and `2` to the multiplication operator just like you asked and moved on.
  With the program now compiled, dynamic typing comes into effect *at runtime*. When the multiplication operator gets around to look at its operands, it checks to see what is the current type of each one. As before, it's a string and an int. But the operator knows that it can only multiply two integers (let's ignore floats for simplicity), so it has produced an integer value from the string.

- Strong Typing : Strong typing probably means that variables have a well-defined type and that there are strict rules about combining variables of different types in expressions.
  For example,

```
a  = 9
b = "9"
c = concatenate(a, b)  // produces "99"
d = add(a, b)          // produces 18
```

This is not possible in python as it is a strongly typed language. To do something like this, we need to do casting.

```
a  = 9
b = "9"
c = concatenate(  str(a),  b)    // produces "99"
d = add(a,  int(b)  )            //produces 18
```

Python code is written in this way

Type casting:

Python is moderately type-checked. Implicit conversion is defined for numeric types (as well as booleans), so one may validly multiply a complex number by a long integer (for instance) without explicit casting. However, there is no implicit conversion between (e.g.) numbers and strings; a string is an invalid argument to a mathematical function expecting a number.

Example ::

It does support
a=35+56.23

but it does not support
a='35'+56.23

Scoping rules:

Python follows the LGB Rule

- Name references search at most three scopes: local, then global, then built-in.
- Name assignments create or change local names by default.
- “Global” declarations map assigned names to an enclosing module’s scope.

In other words, all names assigned inside a function def statement are locals by default; functions can use globals, but they must declare globals to change them. Python’s name resolution is sometimes called the LGB rule, after the scope names:

When you use an unqualified name inside a function, Python searches three scopes—the local (L), then the global (G), and then the built-in (B)—and stops at the first place the name is found.

When you assign a name in a function (instead of just referring to it in an expression), Python always creates or changes the name in the local scope, unless it’s declared to be global in that function.

When outside a function (i.e., at the top-level of a module or at the interactive prompt), the local scope is the same as the global—a module’s namespace.

Example::

- ```
  # global scope
  X = 99              # X and func assigned in module: global

  def func(Y):        # Y and Z assigned in function: locals
      # local scope
      Z = X + Y       # X is not assigned, so it's a global
      return Z
  ```
  ```

func(1) # func in module: result=100

Global names: X, func

X is a global because it's assigned at the top level of the module file; it can be referenced inside the function without being declared global. func is global for the same reason; the def statement assigns a function object to the name func at the top level of the module.

Local names: Y, Z

Y and Z are local to the function (and exist only while the function runs), because they are both assigned a value in the function definition; Z by virtue of the = statement, and Y because arguments are always passed by assignment (more on this in a minute).

- *y, z = 1, 2 # global variables in module*

```
def all_global():
    global x       # declare globals assigned
    x = y + z      # no need to declare y,z: 3-scope rule
```

Here, x, y, and z are all globals inside function all_global.

Bindings:

In computing, a binding from a programming language to a library or operating system service is an application programming interface (API) providing glue code to use that library or service in a particular programming language.

Binding generally refers to a mapping of one thing to another. In the context of software libraries, bindings are wrapper libraries that bridge two

programming languages so that a library written for one language can be used in another language. Many software libraries are written in system programming languages such as C or C++; in order to use these libraries from another (usually higher-level) language such as Python, a binding to the library must be created in that language, possibly requiring the recompilation of the language's code depending on the amount of modification necessary.

There are several ways to call code written in C from Python.First, there is the *ctypes* module in the standard library. It allows you to load a dynamic-link library (DLL on Windows, shared libraries .so on Linux) and call functions from these libraries, directly from Python. Such libraries are usually written in C.

Second, in case of CPython there is the *Python/C API*. It can be used in two major ways:

Examples::

- *ctypes exports the cdll, and on Windows windll and oledll objects, for loading dynamic link libraries.*

  >>>
  *>>> from ctypes import **
  >>> print windll.kernel32
  <WinDLL 'kernel32', handle ... at ...>
  >>> print cdll.msvcrt
  <CDLL 'msvcrt', handle ... at ...>
  >>> libc = cdll.msvcrt
  >>>
  Windows appends the usual .dll file suffix automatically.

- On Linux, it is required to specify the filename including the extension to load a library, so attribute access can not be used

to load libraries. Either the LoadLibrary() method of the dll loaders should be used, or you should load the library by creating an instance of CDLL by calling the constructor:

```
>>>
>>> cdll.LoadLibrary("libc.so.6")
<CDLL 'libc.so.6', handle ... at ...>
>>> libc = CDLL("libc.so.6")
>>> libc
<CDLL 'libc.so.6', handle ... at ...>
```

Expressions ::

Boolean Expressions :: A **boolean expression** (or logical expression) evaluates to one of two states true or false. Python provides the boolean type that can be either set to False or True.

Examples:: Assume variable a holds 10 and variable b holds 20, then –

Operator	**Description**	**Example**
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.

or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

Arithmetic Expressions : Expressions containing arithmetic operators.

Examples :: Assume variable *a* holds *10* and variable *b* holds *20*, then −

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2

% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Assignment Expressions : Expressions containing assignment operators.

Operator	**Description**	**Example**
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a

-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Examples::

Assume variable a holds 10 and variable b holds 20, then −

```
#!/usr/bin/python
```

```
a = 21
b = 10
c = 0

c = a + b
print "Line 1 - Value of c is ", c

c += a
print "Line 2 - Value of c is ", c

c *= a
print "Line 3 - Value of c is ", c

c /= a
print "Line 4 - Value of c is ", c

c  = 2
c %= a
print "Line 5 - Value of c is ", c

c **= a
print "Line 6 - Value of c is ", c

c //= a
print "Line 7 - Value of c is ", c
```

When you execute the above program, it produces the following result −

```
Line 1 - Value of c is 31
Line 2 - Value of c is 52
Line 3 - Value of c is 1092
Line 4 - Value of c is 52
Line 5 - Value of c is 2
Line 6 - Value of c is 2097152
Line 7 - Value of c is 99864
```

Various Features And Constructs

1.Recursion:

-Python supports recursion as similar to all other languages.

Example:-

```
def factorial(n):
{    if n == 0:
        return 1
    else:
return n * factorial(n - 1)
}
```

2.Conditional Statement:

-In order to write useful programs the conditional statements are highly useful to check the constraints on the variables.In python we can write all the conditional statements .For example

“**IF LOOP**”

Syntax:

If expression:

statement(s)

```
food = 'spam'
if food == 'spam':
print('Ummmm, my favorite!')
print('I feel like saying it 100 times...')
print(100 * (food + '! '))
```

‘’**ELSE LOOP**”

Syntax:

If expression:

Statement s

Else:

Statement s

```
if food == 'spam':
 print('Ummmm, my favorite!')
else:
print("No, I won't have it. I want spam!")
```

Python even supports Chained Conditionals , Nested Conditionals.

3.Iterative Statements:-

Repeated execution of a set of statements is called “Iteration”. Python supports all sort of iterative statements.

Examples:

In a for loop , while loop , do while loop a certain set of instructions are executed repeatedly.

“For Loop”:

Syntax:

for iterating_var in sequence:
statements(s)

```
for f in ["Joe", "Zoe", "Brad", "Angelina", "Zuki", "Thandi", "Paris"]:
    invitation = "Hi " + f + ".  Please come to my party”.
    print(invitation)
```

Here we see that invitation is a given a new value each time and the print statement is executed as many number of times as of the array length.

“While Loop”:

```
count = 0
while (count < 9):
   print 'The count is:', count
   count = count + 1

print "Good bye!"
```

Here the loop executes until the count is 8.Once the count reaches 8 the loop terminates.

VARIOUS DATA STRUCTURES SUPPORTED:

Python has quite a few inbuilt data structures in it.
The builtin data structures are:

- **Lists**,

  Example :

  ```
  >>> l = [1, 2, 3]
  >>> l[0]
  1
  >>> l.append(1)
  >>> l
  [1, 2, 3, 1]
  ```

- **Tuples**

Example :

```
>>> d = dict([('jan', 1), ('feb', 2), ('march', 3)])
>>> d['feb']
2
```

➢ **Dictionaries**,

Example :

```
>>> d = {'first':'string value', 'second':[1,2]}
>>> d.keys()
['first', 'second']
>>> d.values()
['string value', [1, 2]]
```

➢ **Strings**

Example :

```
>>> mystr = "This is a string"
>>> len(mystr)
16
```

➢ **Sets**

Example :

```
>>> a = set([1, 2, 3, 4])
>>> b = set([3, 4, 5, 6])
>>> a | b # Union
{1, 2, 3, 4, 5, 6}
>>> a & b # Intersection
{3, 4}
```

➢ **frozensets**

Example :

```
>>> a = frozenset([1, 2, 3])
>>> b = frozenset([2, 3, 4])
>>> a.union(b)
```

frozenset([1, 2, 3, 4])

PROGRAM CONTAINING THE ABOVE FUNCTIONALITIES:

```
# Python program for implementation of heap Sort

# To heapify subtree rooted at index i.
# n is size of heap
def heapify(arr, n, i):
    largest = i  # Initialize largest as root
    l = 2 * i + 1 # left = 2*i + 1
    r = 2 * i + 2 # right = 2*i + 2

    # See if left child of root exists and is
    # greater than root
    if l < n and arr[i] < arr[l]:
        largest = l

    # See if right child of root exists and is
    # greater than root
    if r < n and arr[largest] < arr[r]:
        largest = r

    # Change root, if needed
    if largest != i:
        arr[i],arr[largest] = arr[largest],arr[i]  # swap

        # Heapify the root.
        heapify(arr, n, largest)

# The main function to sort an array of given size
def heapSort(arr):
    n = len(arr)

    # Build a maxheap.
    for i in range(n, -1, -1):
        heapify(arr, n, i)

    # One by one extract elements
    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]   # swap
        heapify(arr, i, 0)

# Driver code to test above
```

```
arr = [ 12, 11, 13, 5, 6, 7]
heapSort(arr)
n = len(arr)
print ("Sorted array is")
for i in range(n):
    print ("%d" %arr[i]),
```

APPLICATIONS OF PYTHON

❏ Python offers many choices for web development:
➢ Frameworks such as Django and Pyramid.
➢ Micro-frameworks such as Flask and Bottle.

❏ Python is widely used in scientific and numeric computing:
➢ SciPy is a collection of packages for mathematics, science, and engineering.
➢ Pandas is a data analysis and modeling library.
➢ IPython is a powerful interactive shell that features easy editing and recording of a work session, and supports visualizations and parallel computing.

❏ Python is often used as a support language for software developers, for build control and management, testing, and in many other ways.
➢ SCons for build control.
➢ Buildbot and Apache Gump for automated continuous compilation and testing.
➢ Roundup or Trac for bug tracking and project management.

❏ Python's standard library supports many Internet protocols:
➢ E-mail processing
➢ Support for FTP, IMAP, and other Internet protocols.

A SAMPLE PROGRAM TO ILLUSTRATE ALL THE ABOVE POINTS

```
#object-oriented

class compute:
    def __init__(self):                    #constructure
            print ("")

    def comp_add(self,x,y):                #functional
            print("-------------------------------------------------")
            print ("calculated using functional programing paradigm")
            print("-------------------------------------------------")
            return ( x + y )

    def sum_fifty(self):
            print("-------------------------------------------------")
            print ("Demonstrating Reflective programing paradigm")
            print("-------------------------------------------------")
            ans=0
            for i in range(50):
                    ans += i
            print ("the sum from 1 to 50 is", ans)

    def add_list(self,Mylist):             #procedural
            print("-------------------------------------------------")
            print ("Demonstrating procedural programing paradigm")
            print("-------------------------------------------------")
            ans=0
            if type(Mylist) is list:
                    for x in Mylist:
                            ans+=x
            return ans

print("**************************")
print ("Python Program Started")
print("**************************")
comp = compute()
sum_1_2 = comp.comp_add(1,2)
print(sum_1_2)
l = [1, 2, 3]
print(comp.add_list(l))

def fact_recur(z):          #recursion
    if(z==0):
            return 1
```

```
    return z*(fact_recur(z-1))

def fact_iter_for(z):
    ans=1;
    for y in range(z):
            ans = ans *(y+1)
    return ans
def fact_iter_while(z):
    ans=1;
    while(z > 1):
            ans = ans * z
            z= z-1
    return ans


class_name = "compute"
method = "sum_fifty"
obj = globals()[class_name]()
getattr(obj,method)()

print("\n\n\n")

print("-------------------------------------------------")
print ("Demonstrating Strongly Typed")
print("-------------------------------------------------")
#strongly typed
try:
    _hello = "hello" + 5 + goodbye
    print (_hello)
except:
    print ("error due to strongly typed")
print("\n\n\n")

#dynamically typed
print("-------------------------------------------------")
print ("Demonstrating dynamically typed")
print("-------------------------------------------------")
_hello = "hello python"
print ("Type of variable _hello is : ",type(_hello))
_hello = 5
print ("Now Type of variable _hello is : ",type(_hello))

print("\n\n\n")

#duck typing
print("-------------------------------------------------")
print ("Demonstrating duck typing")
print("-------------------------------------------------")
```

```
_hello += 10
print (_hello)

_hello = float(_hello)
print ("Now Type of variable _hello after type casting is : ",type(_hello))


print("-------------------------------------------------")
print ("Demonstrating recursion")
print("-------------------------------------------------")
print ("factorial by recursion ",fact_recur(5))
print("\n\n\n")

print("-------------------------------------------------")
print ("Demonstrating Iteration using For Loop")
print("-------------------------------------------------")
print ("factorial by iteration for loop", fact_iter_for(6))
print("\n\n\n")


print("-------------------------------------------------")
print ("Demonstrating Iteration using While Loop")
print("-------------------------------------------------")
print ("factorial by iteration while loop", fact_iter_while(7))
print("\n\n\n")
```

OUTPUT :


Menu
Programm...
CS332_pro...
PPL_lab_AS...
/media/jit...
jitendra@j...
100%
Monday September 12, 3:06 PM
jitendra@jitendra-Inspiron-5547:/media/jitendra/DATA/sem_5/2016/CS332_programming_languages_Lab
File Edit View Search Terminal Help
jitendra@jitendra-Inspiron-5547 /media/jitendra/DATA/sem_5/2016/CS332_programming_languages_Lab $ python all_features.py
****
Python Program Started
****
---
calculated using functional programing paradigm
---
sum of 1 and 2 is :
3

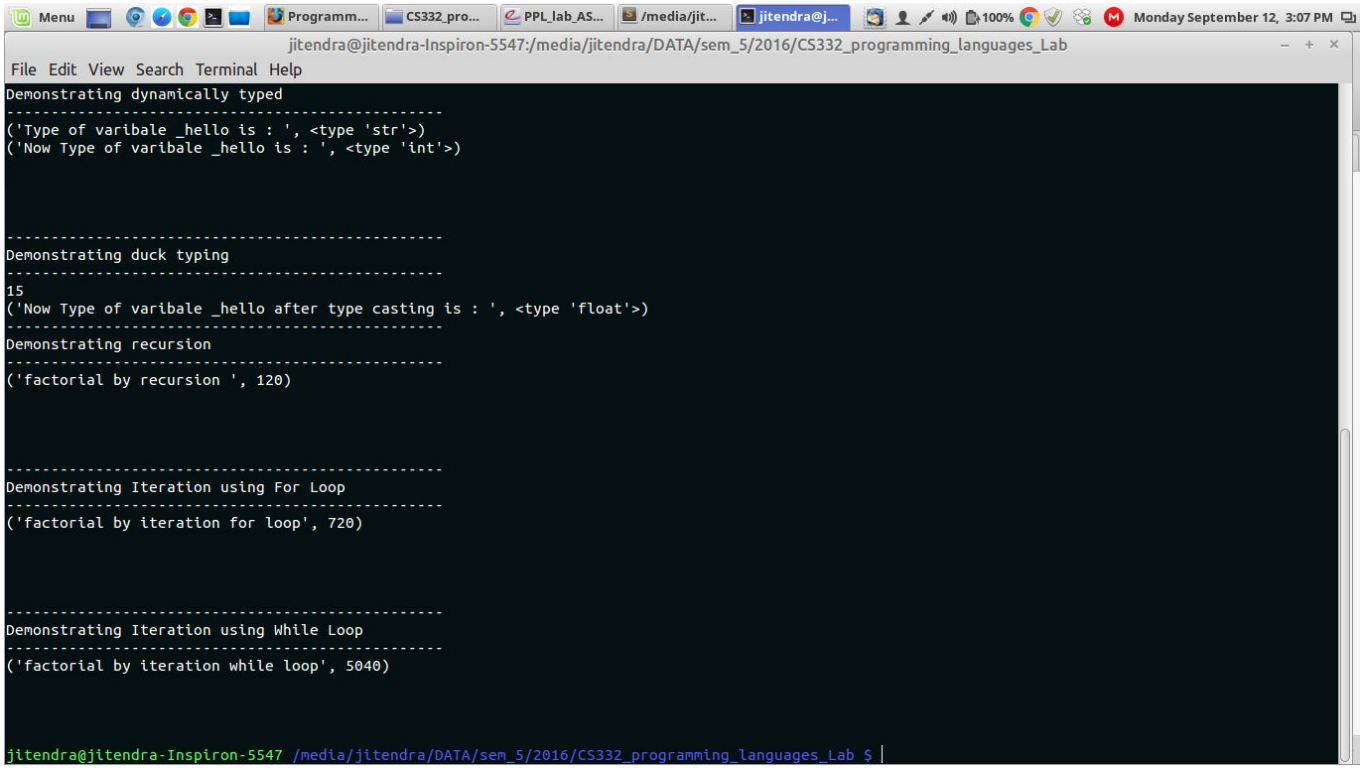
Demonstrating procedural programing paradigm

Sum of the element in the list :
6
---
demonstrating recursion
---
factorial of 5 is :
120

Demonstrating Reflective programing paradigm

('the sum from 1 to 50 is', 1225)

Demonstrating Strongly Typed
---
error due to strongly typed


Menu
Programm...
CS332_pro...
PPL_lab_AS...
/media/jit...
jitendra@j...
100%
Monday September 12, 3:07 PM
jitendra@jitendra-Inspiron-5547:/media/jitendra/DATA/sem_5/2016/CS332_programming_languages_Lab
File Edit View Search Terminal Help
Demonstrating dynamically typed

('Type of varibale _hello is : ', <type 'str'>)
('Now Type of varibale _hello is : ', <type 'int'>)

Demonstrating duck typing

15
('Now Type of varibale _hello after type casting is : ', <type 'float'>)

Demonstrating recursion

('factorial by recursion ', 120)

Demonstrating Iteration using For Loop

('factorial by iteration for loop', 720)

Demonstrating Iteration using While Loop

('factorial by iteration while loop', 5040)
jitendra@jitendra-Inspiron-5547 /media/jitendra/DATA/sem_5/2016/CS332_programming_languages_Lab $