

22nd June 2019

To

Mr. L Venkata Rama Raju,
Founder & CEO,
Data Jango Technologies Pvt Ltd,
Hyderabad.

Respected sir,

Sub: **Guest Speaker Invitation**

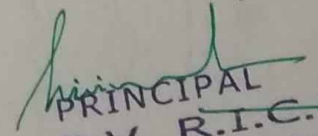
The department of Computer Science wishes to conduct a One-Day workshop on "Python for Data Sciences" for III BSc I Semester students of our college on 29-06-2019 from 9 AM to 5 PM.

We are inviting you as a resource person to deliver an expert lecture on "Python for Data Science". We believe that your contribution to this field is unparalleled and a workshop on this topic will be of great benefit.

Thanking you.



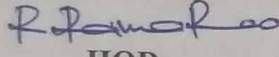
Yours Sincerely

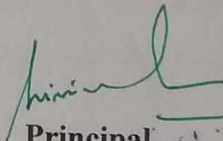

PRINCIPAL
Dr. B.V. R.I.C.E.
Vishnupur, BHIMAVARAM-534 202

CIRCULAR

Date: 25th June 2019

It is informed to that; the department of Computer Science is conducting a One-Day workshop on “**Python for Data Science**” for III BSc I Semester Computer Science students by **Mr. L Venkata Rama Raju, Founder & CEO of Datajango** on 29th June 2019 from 9 AM to 5 PM. Interested students could consult Mr. B Naresh to enrol your names.


HOD


Principal
Dr. B.V. R.I.C.E.
Vishnupur, BHIMAVARAM-534 202.

29th June 2019

To

Mr VenkataRamaraju L,
Founder & CEO,
Datajango.

Dear Sir,

Sub: Letter of Appreciation.

Thank you very much for delivering an informative and thought provoking lecture on "Python for Data Science" held on 29th June 2019 at B V Raju College, Vishnupur, Bhimavaram.

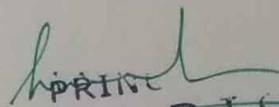
It is really a splendid lecture that exposed our students to the field practices. All the students appreciated and got benefitted from your views on the subject.

Looking forward for your cooperation for the promotion of compute education in future as well.



Thanking you.

Yours Sincerely,


PRINC
Dr. B.V. R.I.C.
Vishnupur, BHIMAVARAM-534 202.

B V Raju College
Vishnupur::Bhimavaram
Workshop on Python for Data Science
Department of Computer Science

Date: 29-06-2019

III BSc (MECs, MPCs & MSCs)

Attendance Sheet

S No	Roll No	Student Name	Section	Signature
1	173117102062	Alluri .valli pravallika	MPCs	A.V. Pravalika
2	173117102073	Darapureddy.HariniSri	MPCs	D. Harini Sri
3	173117102067	Ballari.Shabreen	MPCs	B. Shabreen
4	173117102095	Moturi.Navya	MPCs	M. Navya
5	173117102112	Roulo.Priyanka	MPCs	R. Priyanka
6	173117102109	Ragu.vasaviSaianjali	MPCs	R. Vasavi Saianjali
7	173117102097	mukku.anusha	MPCs	M. Anusha
8	173117102098	Mukku.sitamahalakshmi	MPCs	M. Sitamahalakshmi
9	173117102074	Denaboina.Udaykiran	MPCs	D. Udaykiran
10	173117137308	M.V.S.Chandu	MECs-B	M.V.S. Chandu
11	173117137310	M.Yesu Babu	MECs-B	M. Yesu Babu
12	173117137312	M.Meghana	MECs-B	M. Meghana
13	173117137313	M.Dharani	MECs-B	M. Dharani
14	173117137319	N.Sirisha	MECs-B	N. Sirisha
15	173117137321	N.N.V.S.Ganesh	MECs-B	N.N.V.S. Ganesh
16	173117137332	P.G.T.Durga	MECs-B	P. Durga
17	173117137339	S.S.L.Rasmitha	MECs-B	S.S.L. Rasmitha
18	173117137355	V.Aditya kumar	MECs-B	V. Aditya
19	173117137138	G.Sindhu	MSCs	G. Sindhu
20	173117137143	K.Vidya Devi	MSCs	K. V. Devi

21	173117137144	K.Thomas	MSCs	K.Thomas
22	173117137153	L.Yashwanth	MSCs	L.Yashwanth
23	173117137160	N.Gowthami	MSCs	N.Gowthami
24	173117137161	P.Sailaja	MSCs	P.Sailaja
25	173117137164	P.J.Sai Ram	MSCs	P.Sai Ram
26	173117137165	P.N.D.Bhavani	MSCs	P.N.D.Bhavani
27	173117137166	P.Jyothi	MSCs	P.Jyothi
28	173117137243	A.RAMYA	MECS-A	A.Ramya
29	173117137250	A.ROSHINI	MECS-A	A.Roshini
30	17311713251	A.KAVERI	MECS-A	A.Kaveri
31	17311713252	B.BHARATHI	MECS-A	B.Bharathi
32	17311713272	G.DIVYA	MECS-A	G.Divya
33	173117137277	G.SIRISHA	MECS-A	G.Sirisha
34	173117137279	I.BHASKAR	MECS-A	I.Bhaskar
35	173117137282	J.SWATHI LAKSHMI BHAVANI	MECS-A	J.S.L. Bhavani
36	173117137294	K.TEJA	MECS-A	K.Teja



Python for Data Science



Victor Emmanuel Rios
Founder & CEO
DataCamp

Python's Benevolent Dictator For Life



"Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered."

- Guido van Rossum

Running Python



Overview

- History
- Installing & Running Python
- Names & Assignment
- Sequences types: Lists, Tuples, and Strings
- Mutability

<http://docs.python.org/>



The Python Interpreter

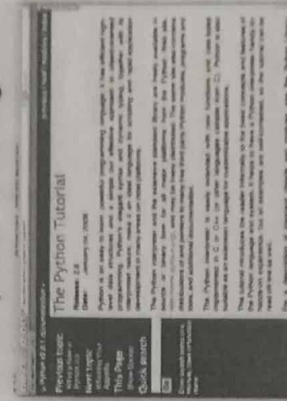
- Typical Python implementations offer both an interpreter and compiler
- Interactive interface to Python with a read-eval-print loop

```
lin@linuz2 ~]$ python
Python 2.4.3 (rt1, Jan 14 2008, 18:32:40)
[GCC 4.1.2 20070926 (Red Hat 4.1.2-14)] on linux2
Type "help", "copyright", "credits" or "license()" for more information
>>> del square(4)
-- return x * x
--
>>> map(square, [1, 2, 3, 4])
[1, 4, 9, 16]
>>>
```

Brief History of Python

- Invented in the Netherlands, early 90s by Guido van Rossum
- Named after Monty Python
- Open sourced from the beginning
- Considered a scripting language, but is much more
- Scalable, object oriented and functional from the beginning
- Used by Google from the beginning
- Increasingly popular

The Python tutorial is good!

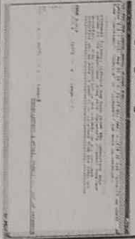


Installing

- Python is pre-installed on most Linux systems, including Linux and MAC OS X
- The pre-installed version may not be the most recent one (2.6.2 and 3.1.1 as of Sept 09)
- Download from <http://python.org/download/>
- Python comes with a large library of standard modules
- There are several options for an IDE
 - IDLE - works well with Windows
 - Emacs with python-mode or your favorite text editor
 - Eclipse with Pydev (<http://pydev.sourceforge.net/>)

IDLE Development Environment

- IDLE is an Integrated Development Environment for Python, typically used on Windows
- Multi-window text editor with syntax highlighting, auto-completion, smart indent and other.
- Python shell with syntax highlighting.
- Integrated debugger with stepping, persistent breakpoints, and call stack visibility



Editing Python in Emacs

- Emacs `python-mode` has good support for editing Python, enabled by default for `.py` files
- Features: completion, symbol help, eldoc, and inferior interpreter shell, etc.



Running Interactively on UNIX

```
On Unix...
% python
>>> 3+3
6
• Python prompts with '>>>'.
• To exit Python (not Idle):
  • In Unix, type CONTROL-D
  • In Windows, type CONTROL-Z + <Enter>
• Evaluate exit()
```

Running Programs on UNIX

- Call python program via the python interpreter
% python fact.py
- Make a python file directly executable by
 • Adding the appropriate path to your python interpreter as the first line of your file
 #!/usr/bin/python
- Making the file executable
 % chmod a+x fact.py
- Invoking file from Unix command line
 % fact.py

Example 'script': fact.py

```
#!/usr/bin/python
def fact(x)
    """Returns the factorial of the argument, assumed to be a positive"""
    if x == 0:
        return 1
    return x * fact(x - 1)

print
print 'N fact(N):'
print "*****"
for n in range(10):
    print n, fact(n)
```

Python Scripts

- When you call a python program from the command line the interpreter evaluates each expression in the file
- Familiar mechanisms are used to provide command line arguments and/or redirect input and output
- Python also has mechanisms to allow a python program to act both as a script and as a module to be imported and used by another python program

Example of a Script

```
#!/usr/bin/python
""" reads text from standard input and outputs any email addresses it finds, one to a line """
import re
from sys import stdin

# a regular expression - for a valid email address
pat = re.compile(r'[w]{1,4}[a-zA-Z]{2,4}')
for line in stdin.readlines():
    for address in pat.findall(line):
        print address
```

results

```
python> python email0.py <email.txt
bill@mstf.com
gates@microsoft.com
steve@apple.com
bill@mstf.com
python>
```

Getting a unique, sorted list

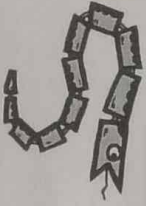
```
import re
from sys import stdin

pat = re.compile(r'[w]{1,4}[a-zA-Z]{2,4}')
# found is an initially empty set (a list of discriminators)
found = set()
for line in stdin.readlines():
    for address in pat.findall(line):
        found.add(address)
# sorted() takes a sequence returns a sorted list of its elements
print address
```

results

```
python> python email2.py <email.txt
bill@mstf.com
gates@microsoft.com
steve@apple.com
python>
```

The Basics



Basic Datatypes

- Integers (default for numbers)
`z = 5 / 2` # Answer 2, integer division
- Floats
`x = 3.456`
- Strings
 - Can use "" or ' to specify with "abc" == 'abc'
 - Unmatched can occur within the string
"att's"
 - Use triple double-quotes for multi-line strings or strings that contain both ' and " inside of them:
"""a'b'c"""

Simple functions: ex.py

```
"""factorial done recursively and iteratively"""
def fact(n):
    ans = 1
    for i in range(2, n):
        ans = ans * i
    return ans
def fact2(n):
    if n < 1:
        return 1
    else:
        return n * fact2(n - 1)
```

A Code Sample (in IDLE)

```
x = 34 - 23          # A comment.
y = "Hello"         # Another one.
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World" # String concat.
print x
print y
```

Whitespace

- Whitespace is meaningful in Python, especially indentation and placement of newlines
- Use a newline to end a line of code
 - Use \ when must go to next line prematurely
 - No braces {} to mark blocks of code, use consistent indentation instead
 - First line with less indentation is outside of the block
 - Colons start of a new block in many constructs, e.g. function definitions, then clauses

Simple functions: ex.py

```
671> python
Python 2.5.2
>>> import ex
>>> ex.fact(6)
1296
>>> ex.fact2(200)
7986578672947895035528321393218507...000000L
>>> ex.fact
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'fact' is not defined
>>>
```

Enough to Understand the Code

- Indentation matters to code meaning
- Block structure indicated by indentation
- First assignment to a variable creates it
 - Variable types don't need to be declared.
 - Python figures out the variable types on its own.
- Assignment is = and comparison is ==
- For numbers + - * / % are as expected
 - Special use of + for string concatenation and % for string formatting (as in C's printf)
- Logical operators are words (and, or, not) not symbols
- The basic printing command is print

Comments

- Start comments with #, rest of line is ignored
 - Can include a "documentation string" as the first line of a new function or class you define
 - Development environments, debugger, and other tools use it: it's good style to include one
- ```
def fact(n):
 """fact(n) assumes n is a positive
 integer and returns factorial of n."""
 return 1 if n==1 else n*fact(n-1)
```



## Assignment

- Binding a variable in Python means setting a name to hold a reference to some object
  - Assignment creates references, not copies
- Names in Python do not have an intrinsic type, objects have types
- Python determines the type of the reference automatically based on what data is assigned to it
- You create a name the first time it appears on the left side of an assignment expression

```
x = 3
```
- A reference is deleted via garbage collection after any names bound to it have passed out of scope
- Python uses *reference semantics* (more later)

## Naming Rules

- Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.
- There are some reserved words:

```
and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while
```

## Naming conventions

- The Python community has these recommended naming conventions
- `joined_lower` for functions, methods and attributes
  - `joined_lower` or `ALL_CAPS` for constants
  - `StudyCaps` for classes
  - `camelCase` only to conform to pre-existing conventions
  - Attributes: `interface`, `_internal`, `__private`

## Assignment

- You can assign to multiple names at the same time

```
>>> x, y = 2, 3
>>> x
2
>>> y
3
```
- This makes it easy to swap values
- ```
>>> x, y = y, x
>>> a = b = x = 2
```
- Assignments can be chained

Accessing Non-Existent Name

- Accessing a name before it's been properly created (by placing it on the left side of an assignment), raises an error
- ```
>>> y
Traceback (most recent call last):
 File "<pyshell>", line 1, in <module>
 y
NameError: name 'y' is not defined
>>> y = 3
>>> y
3
```

## Sequence types: Tuples, Lists, and Strings



## Sequence Types

1. **Tuple:** ('John', 32, ['CMSC1'])
  - A simple *immutable* ordered sequence of items
  - Items can be of mixed types, including collection types
2. **Strings:** "John Smith"
  - *Immutable*
  - Conceptually very much like a tuple
3. **List:** [1, 2, 'John', ('up', 'down')]
  - *Mutable* ordered sequence of items of mixed types

## Similar Syntax

- All three sequence types (tuples, strings, and lists) share much of the same syntax and functionality.
- Key difference:
  - Tuples and strings are *immutable*
  - Lists are *mutable*
- The operations shown in this section can be applied to *all* sequence types
- most examples will just show the operation performed on one

## Sequence Types 1

- Define tuples using parentheses and commas

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
```
- Define lists are using square brackets and commas

```
>>> li = ["abc", 34, 4.34, 23]
```
- Define strings using quotes ('', ' ', or """)

```
>>> st = "Hello World"
>>> st = 'Hello World'
>>> st = """This is a multi-line
string that uses triple quotes."""
```

### Lists are mutable

```
>>> l1 = ['abc', 23, 4.34, 23]
>>> l1[l1] = 45
>>> l1
['abc', 45, 4.34, 23]
```

- We can change lists in place.
- Name *l1* still points to the same memory reference when we're done.

### Tuples are immutable

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
>>> t[2] = 3.14
Traceback (most recent call last):
 File "python11.py", line 1, in <module>
TypeError: object doesn't support item assignment
```

- You can't change a tuple.
- You can make a fresh tuple and assign its reference to a previously used name.
- *The immutability of tuples means they're faster than lists.*

### Operations on Lists Only

```
>>> l1 = [1, 11, 3, 4, 5]
>>> l1.append('a') # Note the method syntax
>>> l1
[1, 11, 3, 4, 5, 'a']
>>> l1.insert(2, '1')
>>> l1
[1, 11, '1', 3, 4, 5, 'a']
```

### The extend method vs +

- + creates a fresh list with a new memory ref
  - *extend* operates on list *l1* in place.
- ```
>>> l1.extend([9, 8, 7])
>>> l1
[1, 2, '1', 3, 4, 5, 'a', 9, 8, 7]
```
- *Potentially confusing:*
 - *extend* takes a list as an argument
 - *append* takes a singleton as an argument
- ```
>>> l1.append([10, 11, 12])
>>> l1
[1, 2, '1', 3, 4, 5, 'a', 9, 8, 7, [10, 11, 12]]
```

### Operations on Lists Only

```
Lists have many methods, including index, count,
remove, reverse, sort
>>> l1 = ['a', 'b', 'c', 'b']
>>> l1.index('b') # index of 1st occurrence
1
>>> l1.count('b') # number of occurrences
2
>>> l1.remove('b') # remove 1st occurrence
>>> l1
['a', 'c', 'b']
```

### Operations on Lists Only

```
>>> l1 = [5, 2, 6, 8]
>>> l1.reverse() # reverse the list 'in place'
>>> l1
[8, 6, 2, 5]
>>> l1.sort() # sort the list 'in place'
>>> l1
[2, 5, 6, 8]
>>> l1.sort(reverse=True) # sort in place using user-defined comparison
```

### Tuple details

- The comma is the tuple creation operator, not parens
- ```
>>> ()
()
>>> (1)
(1)
>>> (1)
(1)
>>> (1)
(1)
```
- Don't forget the comma!
 - Trailing comma only required for singletons others
 - Empty tuples have a special syntactic form
- ```
>>> ()
()
>>> tuple()
()
```

### Summary: Tuples vs. Lists

- Lists slower but more powerful than tuples
  - Lists can be modified, and they have lots of handy operations and methods
  - Tuples are immutable and have fewer features
  - To convert between tuples and lists use the *list()* and *tuple()* functions.
- ```
l1 = list(tu)
tu = tuple(l1)
```

Sequence Types 2

- Access individual members of a tuple, list, or string using square bracket "array" notation
 - Note that all are 0 based...
- ```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
>>> tu[1] # Second item in the tuple, 'abc'
'abc'
>>> li = ['abc', 34, 4.34, 23]
>>> li[1] # Second item in the list.
34
>>> st = "Hello World"
>>> st[1] # Second character in string.
'e'
```

## Slicing: return copy of a subset

- ```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
Omit first index to make copy starting from beginning of the container
>>> t[:2]
(23, 'abc')
Omit second index to make copy starting at first index and going to end
>>> t[2:]
(4.56, (2,3), 'def')
```

The + Operator

- The + operator produces a new tuple, list, or string whose value is the concatenation of its arguments.
- ```
>>> (1, 2, 3) + (4, 5, 6)
(1, 2, 3, 4, 5, 6)
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]
>>> "Hello" + " " + "World"
'Hello World'
```

## Positive and negative indices

- ```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
Positive index: count from the left, starting with 0
>>> t[1]
'abc'
Negative index: count from right, starting with -1
>>> t[-3]
4.56
```

Copying the Whole Sequence

- [:] makes a copy of an entire sequence
- ```
>>> t[:]
(23, 'abc', 4.56, (2,3), 'def')
 - Note the difference between these two lines for mutable sequences

```
>>> l2 = l1 # Both refer to 1 ref,
# Changing one affects both
>>> l2 = l1[:] # Independent copies, two refs
```


```

## The \* Operator

- The \* operator produces a new tuple, list, or string that "repeats" the original content.
- ```
>>> (1, 2, 3) * 3
(1, 2, 3, 1, 2, 3, 1, 2, 3)
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> "Hello" * 3
'HelloHelloHello'
```

Slicing: return copy of a subset

- ```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
Return a copy of the container with a subset of the original members. Start copying at the first index, and stop copying before second.
>>> t[1:4]
('abc', 4.56, (2,3))
Negative indices count from end
>>> t[1:-1]
('abc', 4.56, (2,3))
```

## The 'in' Operator

- Boolean test whether a value is inside a container.
- ```
>>> 5 in t
False
>>> (2, 4) in t
True
>>> 4 in t
True
>>> 'not in t'
False
```
- For strings, tests for substrings
- ```
>>> 'abcde' in 'abcde'
True
>>> 'abc' in 'abcde'
True
>>> 'def' in 'abcde'
False
```
- Be careful: the in keyword is also used in the syntax of for loops and list comprehensions

## Mutability: Tuples vs. Lists



**B V RAJU COLLEGE**

**VISHNUPUR::BHIMAVARAM**

---

**DEPARTMENT OF COMPUTER SCIENCE**

**EVENT NAME:** Python for Data science

**DATE:** 29/06/19

**PARTICIPANT FEEDBACK FORM**

Name of the Student : B. shabreen  
Register Number : 173117102067  
Course & Group : III BSC MPC8.  
Contact Number : 9398661250  
Email ID : Ballari.shabreen@gmail.com  
Future events you are expecting : We need these type of workshop  
in future also  
How do you rate the event conducted: 1/2/3/4/5✓  
Are you satisfied with event conduction: Yes/No  
Comments or Suggestions : Nothing.

B. shabreen.  
Signature of the student

B V RAJU COLLEGE

VISHNUPUR::BHIMAVARAM

DEPARTMENT OF COMPUTER SCIENCE

EVENT NAME: Python for Data Science

DATE: 29/6/19

PARTICIPANT FEEDBACK FORM

Name of the Student : M. Seetha Mahalakshmi  
Register Number : 173117102098  
Course & Group : III B.Sc (MPCS)  
Contact Number : 7981473097  
Email ID : mseetha895@gmail.com  
Future events you are expecting : Need workshop on Cloud Computing  
How do you rate the event conducted: 1/2/3/4/5  
Are you satisfied with event conduction: Yes/No  
Comments or Suggestions : No

M. Seetha Mahalakshmi  
Signature of the student

**B V RAJU COLLEGE**

**VISHNUPUR::BHIMAVARAM**

---

**DEPARTMENT OF COMPUTER SCIENCE**

EVENT NAME: Python for Data Science

DATE: 29/6/19

**PARTICIPANT FEEDBACK FORM**

Name of the Student : M. Meghana  
Register Number : 173117137312  
Course & Group : III BSC (MECS-B)  
Contact Number : 7306345077  
Email ID : meghanamegi143@gmail.com  
Future events you are expecting : online Quiz  
How do you rate the event conducted: 1/2/3/4/5 ✓  
Are you satisfied with event conduction: Yes/No ✓  
Comments or Suggestions : No

M. Meghana  
Signature of the student

**B V RAJU COLLEGE**

**VISHNUPUR::BHIMAVARAM**

---

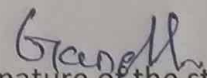
**DEPARTMENT OF COMPUTER SCIENCE**

EVENT NAME: Python

DATE: 29-06-19

**PARTICIPANT FEEDBACK FORM**

Name of the Student : N.N.U.S. Ganesh  
Register Number : 173117137321  
Course & Group : ITI BSC & M.E.S  
Contact Number : 9182550064  
Email ID : ganeshnimmala996@gmail.com  
Future events you are expecting :  
How do you rate the event conducted: 1/2/3/4/5  
Are you satisfied with event conduction: Yes/No  
Comments or Suggestions : Nothing

  
Signature of the student

B V RAJU COLLEGE

VISHNUPUR::BHIMAVARAM

DEPARTMENT OF COMPUTER SCIENCE

EVENT NAME: Python for Data Science

DATE: 29/06/19

PARTICIPANT FEEDBACK FORM

Name of the Student : P. G. T. Dugga  
Register Number : 173117137332  
Course & Group : III MEG-13  
Contact Number : 9642762479  
Email ID : geethikachinni@gmail.com  
Future events you are expecting : yes  
How do you rate the event conducted: 1/2/3/4/5  
Are you satisfied with event conduction: Yes/No  
Comments or Suggestions : NO

P. G. T. Dugga  
Signature of the student



**B V RAJU COLLEGE**

**VISHNUPUR::BHIMAVARAM**

---

**DEPARTMENT OF COMPUTER SCIENCE**

EVENT NAME: *Data Science*

DATE: *29.06.19*

**PARTICIPANT FEEDBACK FORM**

Name of the Student : *S. S. L. Rasmita*

Register Number : *173117137339*

Course & Group : *MES-B*

Contact Number : *9398622045*

Email ID : *lalithasamavedam99@gmail.com*

Future events you are expecting : *—*

How do you rate the event conducted: *1/2/3/4/5* ✓

Are you satisfied with event conduction: *Yes/No* ✓

Comments or Suggestions : *—*

*S.S.L. Rasmita*  
Signature of the student