SECTOR: ICT

SUB-SECTOR: SOFTWARE DEVELOPMENT

SFDBA 501: DEVELOP A BACKEND APPLICATION

Learning Unit:

1. Introduction to Python Programming Language
2. **Apply objects, methods and flow controls**
3. Small Applications in Python

UNIT1: **INTRODUCE PYTHON LANGUAGE**

Purpose statement:

Coding or writing computer programs is a passion because programmers spend much time for writing instructions to abstract real world things into computerized problem solving and decision making. Recent programming languages contributed to achieving this by implementing Function-Driven programming approach. Nowadays, Object-Oriented Programming approach overcomes the problems of traditional approaches such as program extensibility, scalability, static-typed, platform limitations, and portability.

That is why **python** is selected for writing instructions, and **django** framework to implement Rapid Application Development (RAD).

PYTHON PROGRAMMING LANGUAGE (CONT…)

Objectives to achieve.

At the end of the session, participants will be able to:

- Install Python and Pycharm IDE.
- Declare python variables and assign values to them.
- Apply **input()** and **print()** functions.
- Implement comments.
- Define python **test** and **loop** block of instructions.

STEPS TO INSTALL PYTHON AND PyCharm IDE.

PyCharm is a cross-platform editor developed by JetBrains. Pycharm provides all the tools you need for productive Python development.

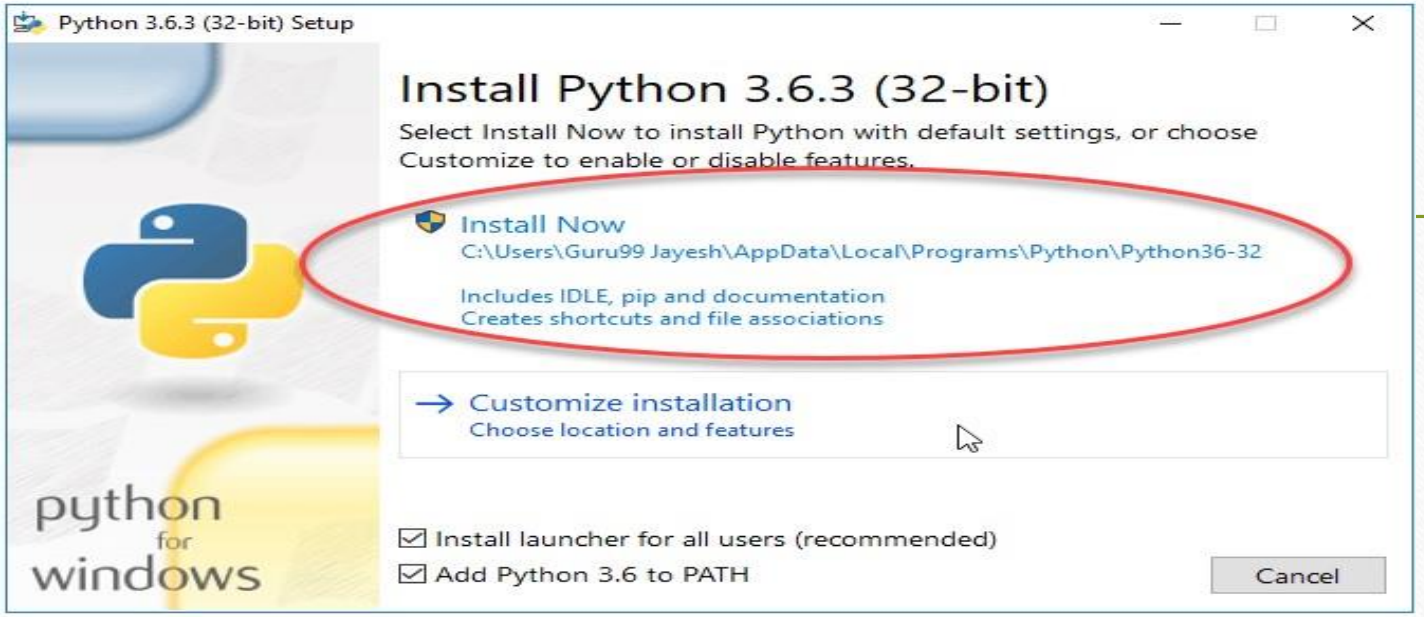
Below are the detailed steps for installing Python and PyCharm:

1. Installing Python

Step 1) To download and install Python visit the official website of Python http://www.python.org/downloads/ and choose your version


python™
About
Downloads
Documentation
Community
Download the latest version for Windows
Download Python 3.6.3
Download Python 2.7.14
Wondering which version to use? Here's more about the difference between Python 2 and 3.
Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other
Want to help test development versions of Python? Pre-releases

Step 2) Once the download is complete, run the exe for install Python. Now click on Install Now.

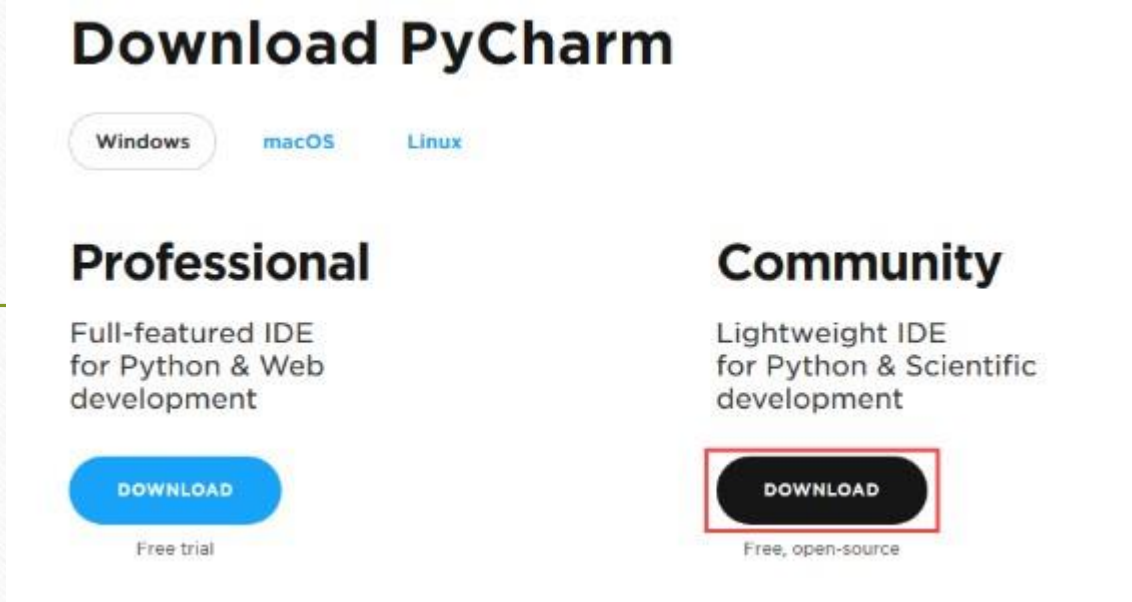


Note: remember to check “Add Python 3.6 to PATH”, to create language advanced system environment variable.

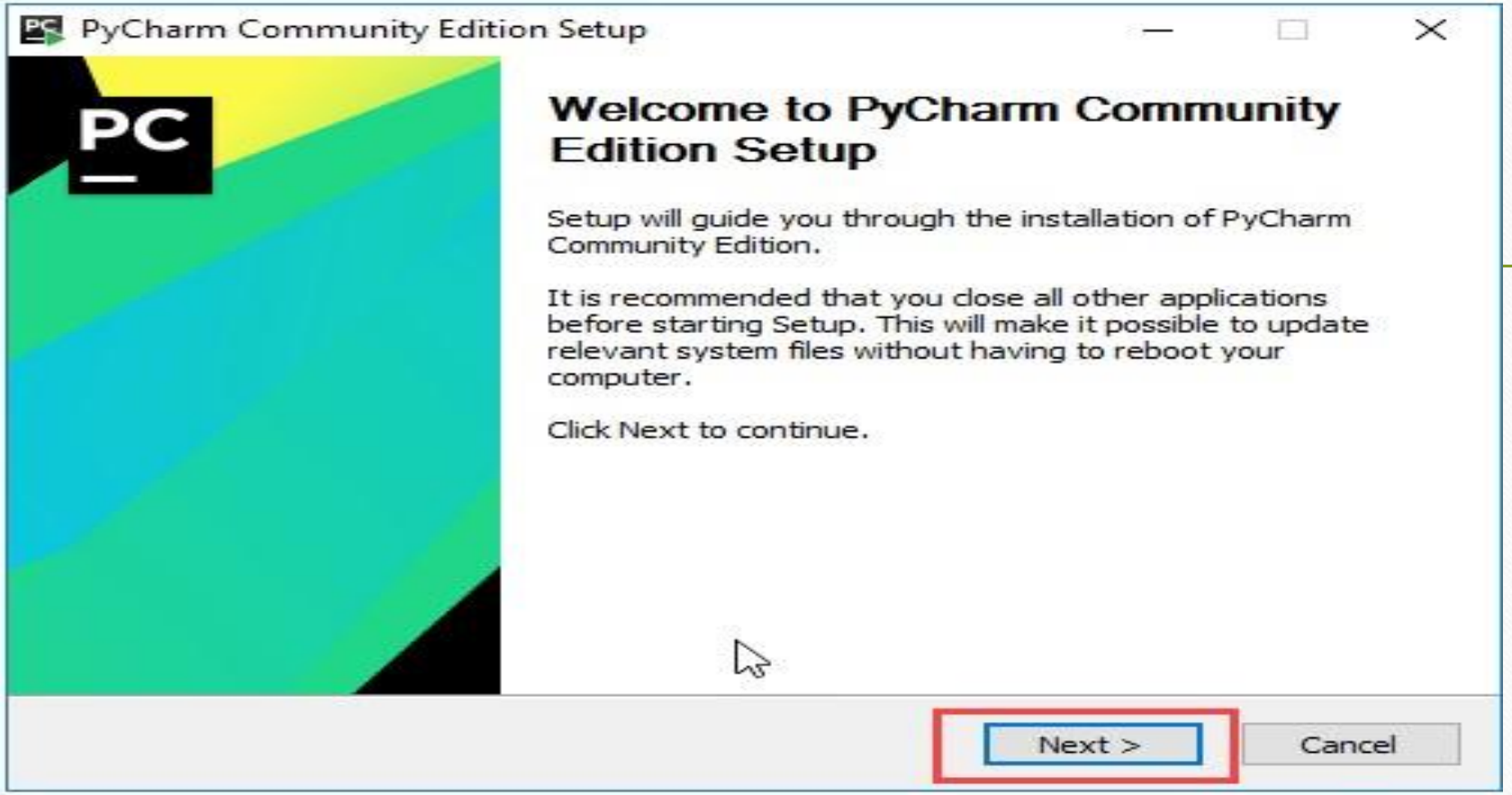
Step 3) You can see Python installing at this point. The remaining steps, you follow on screen instructions.

2. Installing Pycharm IDE

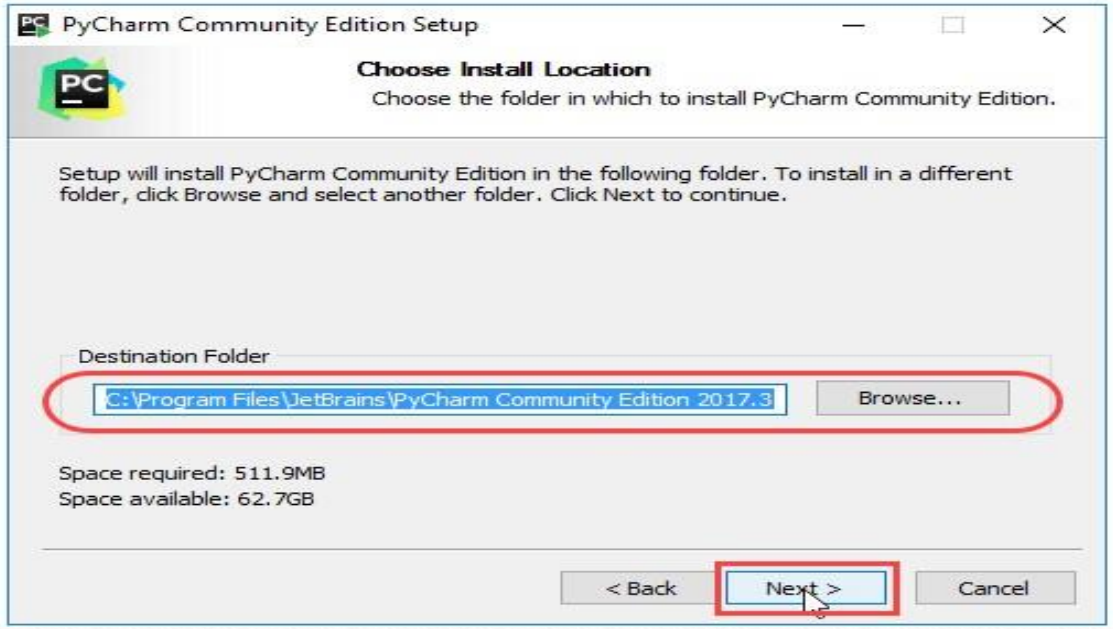
Step 1) To download PyCharm visit the website https://www.jetbrains.com/pycharm/download/ and Click the "DOWNLOAD" link under the Community Section.



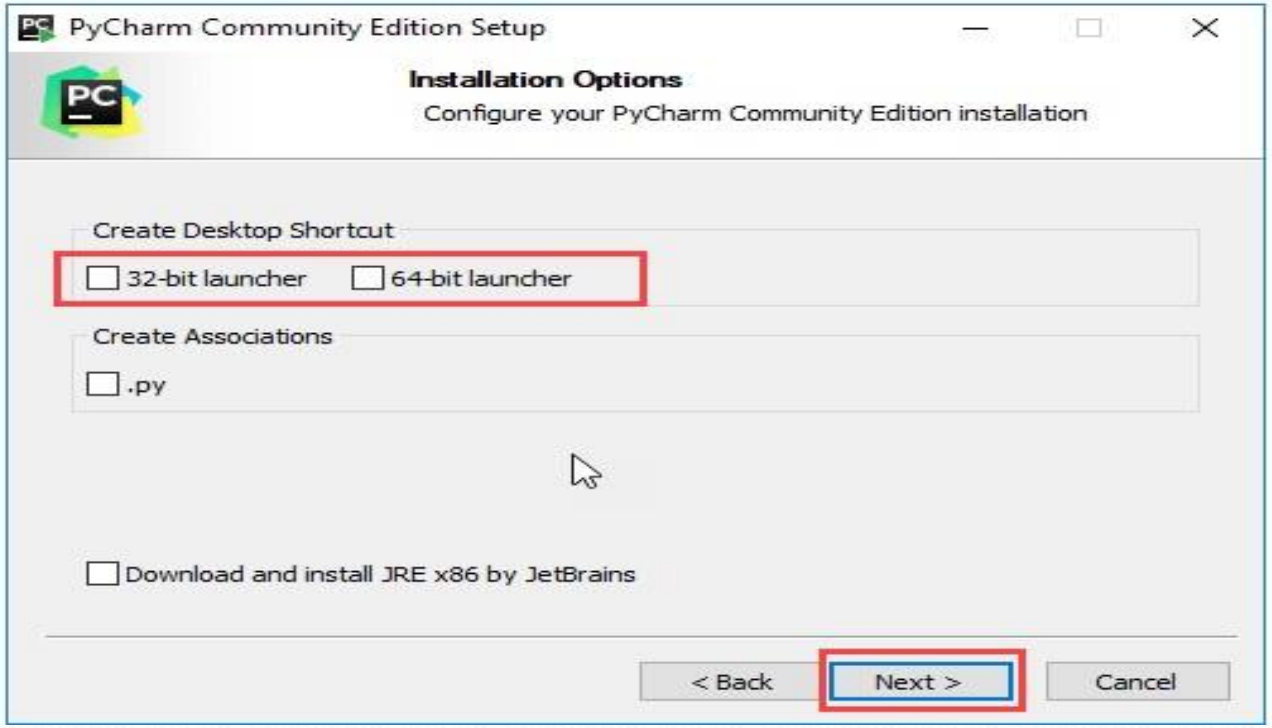
Step 2) Once the download is complete, run the exe for install PyCharm. The setup wizard should have started. Click "Next".


PyCharm Community Edition Setup
PC
Welcome to PyCharm Community Edition Setup
Setup will guide you through the installation of PyCharm Community Edition.
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Next >
Cancel

Step 3) On the next screen, Change the installation path if required. Click “Next”.



Step 4) On the next screen, you can create a desktop shortcut if you want and click on "Next". Select a launcher according to your system type. And then follow on screen Instructions.



PYTHON EDITORS AND IDEs

A. Let start with Python IDEs:

An integrated development environment (**IDE**) is a software application that provides comprehensive facilities to **computer programmers** for software development. An **IDE** normally consists of a source code editor, build automation tools, and a debugger. Some **IDEs** contain a compiler, interpreter, or both.

- PyCharm IDE (installed in previous slides). It is one of popular Python IDE
- Wing IDE: you can get it from (https://wingware.com › downloads

Wing Pro. The full-featured Python **IDE** for professional developers. Powerful Debugger; Intelligent Editor with Code Warnings; Extensive Code Inspection

). In our lesson, we have selected Pycharm IDE (community edition).

PYTHON FRAMEWORKS

A **framework** is a collection of program that you can use to develop your own application. It is built on top of a **programming language**. **Framework** is a set of pre-written code libraries designed to be used by developers.

Examples: Django, TurboGears, Flask, CherryPy. Etc

For our lesson, we have selected Django (Popular and lightweight framework).

PYTHON EDITORS AND IDEs

B. Python Editors

Editors or text **editors** are software programs that enable the user to create and edit text files. In the field of **programming**, the term **editor** usually refers to source code **editors** that include many special features for writing and **editing** code. Examples:

Sublime text, VIM, Visual Studio Code. Etc In our lesson, we have selected Sublime text 3.

- **Coding or programming strategies:**

1. Collecting data from individuals or institutions.
2. Analyzing and interpreting data inline with community problems.
3. Choosing appropriate programming tools (Language, framework, IDE, and Editor).
4. Implementing the solution.
5. Presenting and testing prototypes to system users and get new inputs.

6. Handing over the solution (program) with the client.

CODE FOR CUSTOMERS

PROGRAMMING IN PYTHON

What is Python?

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library.

This means, you can develop any program using python. The language involves objectoriented features and early procedural oriented programming features.

PROGRAMMING LANGUAGE INSTRUCTION CATEGORIES

- ✓ **Variables** as value containers.
- ✓ **Read/Input** and **Write/Output** functions
- ✓ **Tests**
- ✓ **Loops or iterative structures**

i. **Variables declaration and value assignment.**

Variable declaration syntax: <variable name> = <value>

Examples: Age=25

1.VARIABLES IN PYTHON

Two types/categories of variables exist: **Global** and **Local** variables.

Global variables are the one that are defined and declared outside a function and we need to use them inside a function. Example: # this function f() uses global variable

s def f():

print (s)

global scope

VARIABLES IN PYTHON (Cont…)

s="I am proud to be Rwandan"

f()

Output: I am proud to be Rwandan

If a variable with same name is defined inside the scope of function as well then it will print the value given inside the function only and not the global value.

```
# this function has a variable with
# name as same as s def f():
    s=" Me too!"
    print (s)          # global scope
    s="I am proud to be Rwandan"
    f()
    print (s)
```

Output:

```
Me too!
I am proud to be Rwandan
```

➢ Manipulating variables

1) Re-declare a variable

- ~~You can re-declare the variable even after you have declared it once.~~
- Here we have variable initialized to f=0.
- Later, we re-assign the value "John" to variable f.
- The source code can be the following:

➢ Manipulating variables (Cont…)

declare a variable and initialize it

```
f=0
print (f)
# re-declaring the variable here
f="John"
(f)
```

Output: 0 print

John

➢ Manipulating variables (Cont…)

2) Concatenate variables

You cannot concatenate different data types like string and number together. For example, we will concatenate "TV" with the number "10". To do so, you

have to apply conversion method (str) like: a=“TV” b=10 print (a+str(b))
+(plus)=concatenation operator.

Output: TV10

➢ Manipulating variables (Cont…)

3) Delete a variable

You can also delete variable using the command **del** "variable name". **Example:**

a=15

print(a) # **Output**: 15 del (a)

a has been deleted! print (“a has been deleted!”)

VARIABLES IN PYTHON ARE DYNAMICALLY DECLARED (CONT…)

Firstname="John"

Pi=3.14

Checked=true

Here, the data type is not specified at all. Value data type is recognized while the application is being interpreted.

Variables values are coming from one of the following pools/domain called data types:

- Numeric (Numbers): 13, 75.08. etc.

VARIABLES IN PYTHON ARE DYNAMICALLY DECLARED

(CONT…)

- Alphabetic: a, tony, James. Etc.
- Alphanumeric: TV5, Afrique24, 50cent. Etc.
- Boolean: True/False

Receiving values into variables:

Variable assignment: x=2.15. Variable assignment works from left to right.

You can also assign a single value to several variables simultaneously.

a = b = c = 1 or multiple assignment such as : **i, k,j=10,8,11**

VARIABLES IN PYTHON ARE DYNAMICALLY DECLARED (CONT…)

Printing variable value.

Example: i,k,j=10,8,11

print("THE VALUE OF i=",i) **Output** print("THE VALUE OF k=",k) **THE VALUE OF i=10** print("THE VALUE OF j=",j) **THE VALUE OF k=8**

THE VALUE OF j=11

VARIABLES IN PYTHON ARE DYNAMICALLY DECLARED (CONT…)

Arithmetic operation on variable value.

Arithmetic operations (addition, substraction, multiplication, and division,) can modify variable value while printing it on computer console/screen. For instance: Age=20

print("Your age has been modified to:", Age+3)

Output

Your age has been modified to:23

VARIABLES IN PYTHON ARE DYNAMICALLY DECLARED (CONT…)

User inputs

To get input from the user, use the **input ()** function. Example:

name = input("What is your name? ")

input () function can be used together with conversion/parse methods to capture user input with specified data type.

Example: age=int(input("Enter your age here"))

Print("your age is:",age)

VARIABLES IN PYTHON ARE DYNAMICALLY DECLARED (CONT…)

Recall: Rules for variable naming:

1. ~~Variables names must start with a letter or an underscore~~.

x = True # valid

_y = True # valid

9x = False # starts with numeral

=> SyntaxError: invalid syntax

$y = False # starts with symbol or special character

=> SyntaxError: invalid syntax

VARIABLES IN PYTHON ARE DYNAMICALLY DECLARED (CONT…)

2. **The remainder of your variable name may consist of letters, numbers and underscores.**

has_0_in_it = "Still Valid" **3.**

Names are case sensitive.

x = 9 y

= X*5

=>NameError: name 'X' is not defined

COMMENTS IN PYTHON

- Comments in Python are denoted with the pound or hash mark (#).When that is the first
- character of a line of code, the entire line is deemed a comment.The # can also appear in • the middle of the line; this means from the point where it is found, the rest of the same
- line is a comment. For example:
- # this entire line is a comment
- foo = 1 # short comment: assign int 1 to 'foo'
- print 'Python and %s are number %d' % ('Django', foo)

- Comments are not only used to explain nearby code, but also to prevent what would
- otherwise be working code from executing.

PYTHON DATA PRIMITIVE/ STANDARD TYPES

They include scalars or literals (such as numbers and strings).

1) Numbers

- Python has two primary numeric types: int (for integer) and float (for floating point number). Python has only one integer type, int, as opposed to many other languages that have multiple integer types.

- In addition to normal base-10 notation, integers can be represented in hexadecimal (base 16) and octal(base 8). Floats are double-precision floating-point real numbers you should be familiar.

PYTHON DATA PRIMITIVE/ STANDARD TYPES(Cont…)

2. Strings are sequence of characters or a mixture of characters and numbers.

When assigned to variables, they have to be either single- quoted or doublequoted. Both mean the same thing in python. For instance, the following strings have been assigned to variables:

Name=‘Kalisa’ or Name=“Kalisa”

TV=‘TV10’ or TV=“TV10” . The same quotation marks are applied to string literals which may be displayed to users for communication.

PYTHON DATA PRIMITIVE/ STANDARD TYPES(Cont…)

3. **Boolean Type**: Like most other languages, exactly two Boolean values can be expressed: True and False. All Python values can be represented as a Boolean value, regardless of their data values.

For example, any numeric type equal to zero is considered False while all nonzero numeric values are True. Similarly, empty containers are False while nonempty containers are True.

PYTHON USER DEFINED DATA TYPES

1. LISTS

The list type is probably the most commonly used collection type in Python. Despite its name, a list is more like an array in other languages, mostly JavaScript. In Python, a list is merely an ordered collection of valid Python values. A list can be created by enclosing values, separated by commas, in square brackets:

Syntax: list_identifier=[element1, element2, element3, …, element n]

Example1: names=['Jules','Tony','Boston','Robison']

The elements of a list are not restricted to a single data type, which makes sense given that Python is a dynamic language:

PYTHON USER DEFINED DATA TYPES (Cont…)

Example2: mixed_list = [1, ‘abc’, True, 2.34,’Peter’] . Lists

can be nested like: nested_list = [['a', 'b', 'c'], [1, 2, 3]] Lists

are mutable, so you can change the values in a list:

Mixed_list[1]=‘Zoo’

Print(mixed_list)

2. Tuples

A tuple is similar to a list except that it is fixed-length and immutable.

PYTHON USER DEFINED DATA TYPES (Cont…)

2. Tuples (Cont…)

Example: ip_address = ('10.20.30.40', 8080)

The same indexing rules for lists also apply to tuples.

3. Dictionaries

A dictionary in Python is a collection of key-value pairs. The dictionary is surrounded by curly braces. Each pair is separated by a comma and the key and value are separated by a colon. Here is an example:

PYTHON USER DEFINED DATA TYPES (Cont…)

3. **Dictionaries (Cont…)**

Example: a dictionary to list the months of the year

Months={'Jan': 'January', 'Feb':'February','Mar':'March','Apr':'April','Ma':'May',

‘Ju’:’June’,’Jul’:’July’:’Au’:’August’,’Spt’:’September’,’Oct’:’October’,’Nov’:’November’, ‘Dec’:’December’}

Print(Months[‘Au’]) #display August

4. **Sets.** Sets are defined in a similar way like dictionaries.

OPERATORS IN PYTHON

Python language supports the following types of operators.

- Arithmetic **Operators**.
- Comparison (Relational) **Operators**.

- Assignment **Operators**.
- Logical **Operators**. • Bitwise **Operators**.
- Membership **Operators**.

- Identity **Operators**.

OPERATORS IN PYTHON (Cont…)

- You are familiar with Arithmetic operators, Relational operators, Assignment operators, and Logical Operators.

Remarks: 1. Assignment operator is used to generate compound operators such as: **+=**, x+=3 is equivalent to x=x+3, **-=, *=, /=, %=.** And so on and so forth.

2. Logical Operators, use **(and, or, not)** as Boolean logical functions.

Here we are going to discuss about other operators which sound new to you.

OPERATORS IN PYTHON (Cont…)

- **Membership Operators or Belonging Operators**

Membership operators are used to test if a sequence is presented in an object:

They are: in, not in. they are useful to set a criterion/condition for both **tests** and **loops.** For instance: **x=20** for x **in** (12,17,20,23,30):

do this (define your block task)

OPERATORS IN PYTHON (Cont…)

- **Identity Operators**

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location.

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

OPERATORS IN PYTHON (Cont…)

- **Bitwise Operators**

Bitwise operators are used to compare (binary) numbers and are common in such languages: C, C++, JAVA. etc

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

OPERATORS IN PYTHON (Cont…)

- **Operator precedence**

With arithmetic operators, programmers use “My Dear Aunt Sally” jargon. To mean Multiplication , Division,, Addition, Substraction (High to lower priorities) respectively. Remainder or Modulas (%) is next to Division in priority.

~~For more details, visit~~

https://www.w3schools.com/python/python_operators.asp or https://www.tutorialspoint.com/python/python_basic_operators.htm

ADVANTAGES OF PYTHON OVER C,C++, PHP, JAVA, and ALGORITHMS

- Python is an interpreted language which makes its program easier to debug.
- In python, programmers don't recall reserved words before declaring variables.
- Literals or sequence of characters are either enclosed in double quotation mark or single quotation mark.
- Python uses comprehensive library because its reserved
- words seem to be English terms familiar to programmers.

ADVANTAGES OF PYTHON OVER C,C++, PHP, JAVA, and ITS ALGORITHMS (Cont…)

like **print()** function used to write a piece of information and **input()** function used to read data from users. This makes python easy to learn

2. INPUT & OUTPUT FUNCTIONS

. **Displaying a request, information, and variable content via print()function**

Example1: print('Enter your age:') #print Enter your age literal

2. x=25

Print('The value of x equals to ': x)

Reading user inputs. Print("ENTER YOUR WEIGHT:")

Weight=int(input()) # input can play both tasks

Weight=int(input("ENTER YOUR WEIGHT:")

Block indentation philosophy.

Python uses indentation to define control and loop constructs. This contributes to Python's readability, however, it requires the programmer to pay close attention to the use of whitespace. For instance, defining a function:

```
def my_function(): # This is a function definition. Note the colon (:)
    a = 2 # This line belongs to the function because it's indented
    return a # This line also belongs to the same function
print(my_function())
```

Hash symbol begins a single line comment.

3. CONDITIONALS

Conditional expressions, involving keywords such as if, elif, and else, provide Python programs with the ability to perform different actions depending on a Boolean condition: True or False.

- **Ternary operator**

The ternary operator is used for inline conditional expressions. It is best used in simple, concise operations that are easily read.

BLOCK OF INSTRUCTIONS (Cont…)

Ternary operator (Cont…)

n = 5

"Greater than 2" if n > 2 else "Smaller than or equal to 2“

- **if, elif, and else**

In Python you can define a series of conditionals using if for the first one, elif for the rest, up until the final

(optional) else for anything not caught by the other conditionals.

BLOCK OF INSTRUCTIONS (Cont…)

<u>if, elif, and else</u> (Cont…)

```
number = 5 if number > 2:
print("Number is bigger than 2.")
elif number < 2:
    print("Number is smaller than 2.")
else:
    print("Number is 2.")
```

Outputs Number is bigger than 2

BLOCK OF INSTRUCTIONS (Cont…)

4. LOOPS

Loops or iterative structures enable developers to set certain portions of their code to repeat through a number of loops which are referred to as iterations.

Example1: **while loop**

```
i = 0
while i < 7: print(i)
print("end of the loop")
```

BLOCK OF INSTRUCTIONS (Cont…)

2. **For loop**

Python provides a more convenient way to express a definite loop. The for statement iterates over a range of values. These values can be a numeric range, or elements of a data structure like a string, list, or tuple. For instance **for n in range(1, 11):**

print(n)

The expression range(1, 11) creates an object known as an iterable that allows the for loop to assign to the variable n the values 1, 2, . . . , 10.

BLOCK OF INSTRUCTIONS (Cont…)

For loop(Cont…)

Range function syntax: range(begin,end,step). **where**

- begin is the first value in the range; if omitted, the default value is 0
- end is one past the last value in the range; the end value may not be omitted
- change is the amount to increment or decrement; if the change parameter is omitted, it defaults to 1 (counts up by ones)

begin, end, and step must all be integer values; floating-point values and other types are not allowed.

BLOCK OF INSTRUCTIONS (Cont…)

For loop(Cont…)

Consider the following loop that counts down from 21 to 3 by threes:

```
for n in range(21, 0, -3):
```

```
        print(n, '', end='')
```

It prints: **21 18 15 12 9 6 3.** The following code computes and prints the sum of all the positive integers less than 100:

```
sum = 0 # Initialize sum
for i in range(1, 100):
    sum += i
print(sum)
```

UNIT 3: **CREATE SMALL APPLICATION IN PYTHON**

MANIPULATE EXCEL SPREADSHEET IN PYTHON

✓ Excel files in Python using openpyxl library

The **openpyxl** is a Python library to read and write Excel 2010(and later versions) xlsx/xlsm/xltx/xltm files.

Install openpyxl by executing **pip install openpyxl** command in your command prompt. **pip**= python installer package. It is used to install python libraries and frameworks.

1. **Openpyxl create new file (.xlsx)**

MANIPULATE EXCEL SPREADSHEET IN PYTHON(Cont…)

Openpyxl create new file (.xlsx) Cont…

```python
from openpyxl import Workbook
import time

book = Workbook()
sheet = book.active

sheet['A1'] = 56
sheet['A2'] = 43

now = time.strftime("%x")
sheet['A3'] = now

book.save("sample.xlsx")
```

MANIPULATE EXCEL SPREADSHEET IN PYTHON(Cont…)

In the example above, we created a new xlsx file. We wrote data into three cells.

From openpyxl module, we imported **Workbook** class. A Workbook is the container of all other parts of the document.

2. Openpyxl write to a cell

There are two basic ways to write to a cell: using cell references such as **A1** or **D3,** or using a row and column notation with **cell()** method. both ways are shown below:

MANIPULATE EXCEL SPREADSHEET IN PYTHON(Cont…)

2. Openpyxl write to a cell (Cont…)

```python
from openpyxl import Workbook

book = Workbook()
sheet = book.active

sheet['A1'] = 1
sheet.cell(row=2, column=2).value = 2

book.save('write2cell.xlsx')
```

MANIPULATE EXCEL SPREADSHEET IN PYTHON(Cont…)

2. Openpyxl write to a cell (Cont…)

In the above example, we write two values to two cells:

sheet['A1']=1 # uses cell reference

sheet.cell(row=2,column=2).value=2 # uses cell ()method. You can also work with workbook containing many sheets. In that case sheets are given indices starting from sheet1, sheet2, …

MANIPULATE EXCEL SPREADSHEET IN PYTHON(Cont…)

3. Openpyxl append values

With the **append()** method, you can append a group of values at the bottom of the current sheet. In this case, a new row (s) is created.

Here is an illustration of the code:

MANIPULATE EXCEL SPREADSHEET IN PYTHON(Cont…)

3. Openpyxl append values (cont…)

```python
from openpyxl import Workbook

book = Workbook()
sheet = book.active

rows = (
    (88, 46, 57),
    (89, 38, 12),
    (23, 59, 78),
    (56, 21, 98),
    (24, 18, 43),
    (34, 15, 67)
)

for row in rows:
    sheet.append(row)

book.save('appending.xlsx')
```

MANIPULATE EXCEL SPREADSHEET IN PYTHON(Cont…)

4. Openpyxl read cell

In the following example, we read the previous written data from the sample.xlsx file.

```
import openpyxl

book = openpyxl.load_workbook('sample.xlsx')

sheet = book.active

a1 = sheet['A1']
a2 = sheet['A2']
a3 = sheet.cell(row=3, column=1)

print(a1.value)
print(a2.value)
print(a3.value)
```

MANIPULATE EXCEL SPREADSHEET IN PYTHON(Cont…)

5. Openpyxl iterate by rows

The **iter_rows()** method returns cells from the worksheet as rows**.** This is achieved through the iterations of a loop such as for. The illustration below iterates over row by row:

5. Openpyxl iterate by rows (Cont…)

```python
from openpyxl import Workbook

book = Workbook()
sheet = book.active

rows = (
    (88, 46, 57),
    (89, 38, 12),
    (23, 59, 78),
    (56, 21, 98),
    (24, 18, 43),
    (34, 15, 67)
)

for row in rows:
    sheet.append(row)

for row in sheet.iter_rows(min_row=1, min_col=1, max_row=6, max_col=3):
    for cell in row:
        print(cell.value, end=" ")
    print()

book.save('iterbyrows.xlsx')
```

MANIPULATE EXCEL SPREADSHEET IN PYTHON(Cont…)

5. Openpyxl iterates by columns

The **iter_cols ()** method can be used to iterate over column by column. The illustration is below:

Further reading of statistics and charts with openpyxl is required.

MANIPULATE EXCEL SPREADSHEET IN PYTHON(Cont…)

```python
from openpyxl import Workbook

book = Workbook()
sheet = book.active

rows = (
    (88, 46, 57),
    (89, 38, 12),
    (23, 59, 78),
    (56, 21, 98),
    (24, 18, 43),
    (34, 15, 67)
)

for row in rows:
    sheet.append(row)

for row in sheet.iter_cols(min_row=1, min_col=1, max_row=6, max_col=3):
    for cell in row:
        print(cell.value, end=" ")
    print()

book.save('iterbycols.xlsx')
```

PYTHON DATE, TIME, and CALENDAR

A Python program can handle date and time in several ways. There is a popular **time** module available in Python which provides functions for working with times, and for converting between representations. The function**time.time()**returns the current system time. Example:

- import time; # This is required to include time module.
- ticks = time.time()
- print ("Number of ticks :", ticks)

PYTHON DATE, TIME, and CALENDAR (Cont…)

Getting current time

- import time
- ~~localtime = time.localtime(time.time())~~
- print ("Local current time :", localtime)

Getting calendar for a month

PYTHON DATE, TIME, and CALENDAR (Cont…)

- The calendar module gives a wide range of methods to play with yearly and monthly calendars. Here, we print a calendar for a given month (Jan 2008)

- import calendar

- cal = calendar.month(2008, 1)

- print("Here is the calendar:“)

- print (cal)

PYTHON DATE, TIME, and CALENDAR (Cont…)

Remarks: 1. to work with time, date, and calendar, **time** and **calendar** modules have to be imported as shown above.

2. A further reading of time and calendar functions such as **calendar.firstweekday( ), calendar.isleap(year), calendar.monthcalendar(year,month),** time.timezone**,…Etc is required JSON IN PYTHON**

- JSON is a syntax for storing and exchanging data.
- JSON is text, written with **J**ava**S**cript **O**bject **N**otation.

❖Parse JSON - Convert from JSON to Python

If you have a JSON string, you can convert it by using **json.loads()** method. The result will be a python dictionary as shown below:

JSON IN PYTHON(CONT…)

Parse JSON - Convert from JSON to Python (cont…)

Example

Convert from JSON to Python:

```
import json

# some JSON:
x =  '{ "name":"John", "age":30, "city":"New York"}'

# parse x:
y = json.loads(x)

# the result is a Python dictionary:
print(y["age"])
```

JSON IN PYTHON(CONT…)

- Convert from Python to JSON

If you have a python object (i.e dictionary), you can convert it into a JSON string by using **json.dumps()** method. The result is a JSON string:

Try the following example

JSON IN PYTHON(CONT…)

Convert from Python to JSON (Cont…)

Example

Convert from Python to JSON:

```
import json

# a Python object (dict):
x = {
  "name": "John",
  "age": 30,
  "city": "New York"
}

# convert into JSON:
y = json.dumps(x)

# the result is a JSON string:
print(y)
```


Purpose statement:

Modern Web application applications development mostly requires general purpose programming and framework. It is in this essence, python **python** programming language and **django** framework are selected.

The choice encounters important features of the above tools such as **easy** and **fast learning**, **lightweight framework**, **simplicity**, and **scalability**. Not only this but also taking advantages of online platforms in the business like e-commerces, elearning, online games, and social networks platforms.

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

Objectives to achieve.

At the end of the session, participants will be able to:

- Configure virtual environment and install Django framework .
- Create projects and applications.
- Create application model through ORM and python classes.
- Create class objects and modify their values
- Run development server and customize admin site.

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

Contents:

- What is Django?
- Projects, Apps, Models, and Views
- URL Structure
- Templates
- Admin Interface/site
- Forms

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

Django is a shiny web framework that allows one to build dynamic, professionallooking websites in python.

Django Pros:

- ✓ Projects or "apps" are pluggable.
- ✓ Object relational mapper: combines the advantages of having a database with the advantages of using an object oriented programming language.
- ✓ Database allows for efficient data storage and retrieval
- ✓ Python allows for cleaner and more readable code

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

- ✓ Django follows MTV-Model-Template-View application design.
- ✓ Django does not work quite like PHP, or other server-side scripting languages.
  - ✓ Django organizes your website into apps.
- ✓ An app represents one component of a website. Example: a simple web poll, blog, etc
- ✓ Apps can be used in multiple different. websites/projects (“pluggable”), and a website can have multiple apps.

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

❑ Creating an isolated Python environment.

It is recommended that you use **virtualenv** to create isolated Python environments, so that you can use different package versions for different projects.

Run the following command in your **shell** or **command prompt** to install virtualenv by issuing: **pip install virtualenv.** # pip-Python Installer Package.

Or **pip install virtualenvwrapper-win** for windows users.

After you install virtualenv, create an isolated environment with the following command: **mkvirtualenv my_env.** # my_env is your environment name**.**

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

❑ Installing Django with pip.

After setting virtual environment, the next step is to install the framework. Run the following command at the shell prompt to install Django with pip:

pip install Django.

❑ Creating your first project.

First Django project will be building a complete blog. Django provides a command that allows you to create an initial project file structure. Run the following command from your shell:

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

- Creating your first project(Cont…). **django-admin startproject mysite**

This will create a Django project with the name **mysite**.

A project structure is generated:

mysite/ # project directory which inside there is:

mysite. # directory

manage.py # python file.

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

Creating your first project (Cont…).

Inside the second mysite directory there are the following python files:

__init__.py

settings.py

urls.py

wsgi.py

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

To complete the project setup, we will need to create the tables in

the database required by the applications listed in INSTALLED_APPS of project settings.py . Open the shell and run the following commands:

cd mysite python manage.py migrate. And note migrate

output lines.

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

- Running the development server

Start the development server by typing the following command from your project's root folder:

python manage.py runserver

Now, open http://127.0.0.1:8000/ in your browser. You should see a page stating that the project is successfully running as illustrated by the following web page:

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

❑ Creating an application

Now, let's create first Django application. We will create a blog application from scratch. From the project's root directory, run the following command: **python manage.py startapp blog**

This will create the basic structure of blog application, which looks like this:

blog/

__init__.py

admin.py

apps.py

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

- Creating an application (Cont…)

Application structure:

migrations/

__init__.py

models.py

tests.py

views.py

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

❑Creating blog data schema (Model)

Application models are python classes which maps objects 'attributes and on database side, classes will be **tables** while attributes form **column set**. This automatic correspondence between Object Oriented Programming and Relation Database design is known as Object Relational Mapping (ORM). Open application Models.py and modify it as follow: from django.db import models class BlogPost(models.Model):

title = models.CharField(max_length=150)

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

Blog model (Cont…) body = models.TextField() timestamp = models.DateTimeField()

This code create new class BlogPost with three attributes: **title**, **body**, and **timestamp**,. The next step is to register the model to admin site as shown below:

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

- **Registering application model to admin site.**

Open application **admin.py** file and edit it as:

```
from  django.contrib  import  admin
from  .models  import  BlogPost
admin.site.register(BlogPost)
```

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT (CONT…)

Making migrations

✓ Make project migrations by running the following command:

Inside project directory, write: **python manage.py makemigrations app name.**

✓ Validate existing migrations by running: python **manage.py migrate**.

❑ **Trying Out the admin site**

Note admin site page in the browser below:

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT (CONT…)

The admin login screen

Python-Gjango Web Application Development(Cont…)

Creating superuser

Before logging in admin login screen. You have to issue the following command in your shell prompt or command prompt interface.

cd mysite

Python manage.py createsuperuser. And then configure superuser username, email(optional), and password.

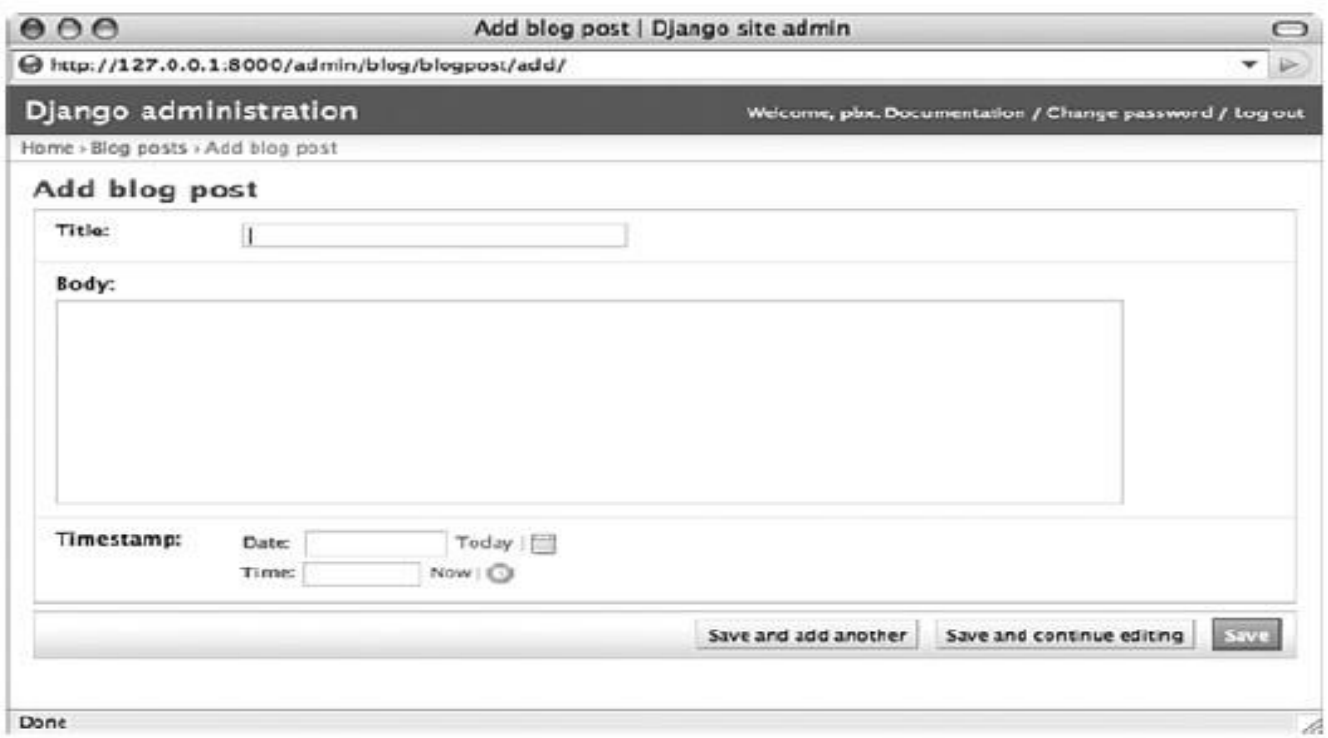
If successfully done, you will get the following screen including your application name and its model.

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT (CONT…)

The admin home page

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

Adding new content via the admin screen



PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

Customizing admin page from django.contrib import admin from .models import BlogPost

from django.contrib.auth.models import Group admin.site.unregister(Group) admin.site.site_header='Admin Dashboard' admin.site.site_title='Blog Management'

PYTHON-GJANGO WEB APPLICATION DEVELOPMENT(CONT…)

RECOMMENDATIONS

Further research

- Django **Views** (these are processes between web pages and the database). They define business logic.
- Django **Templates** (these contain HTML files, CSS files, and Javascript files). They define application layout known as front-end.
- Django and **Content Management System (CMS).**