"Python Technology"

A

Industrial Training Project
Submitted
in partial fulfillment
for the award of the Degree of
Bachelor of Technology
in Department of Computer Science & Engineering
(With specialization in Computer Science & Engineering)

Submitted to:
Mr. Shashi Kant
Assistant Prof.(CSE)

Submitted By:
Prachi Khandelwal
17EMCCS076

Department of Computer Science & Engineering
Modern Institute of Technology & Research Centre
Rajasthan Technical University , Kota
October,2019


Certificate of Completion
This is to certify that Prachi Khandelwal successfully completed 32 hours of Python and Django Full Stack Web Developer Bootcamp online course on Sept. 27, 2019
Jose Portilla
Jose Portilla, Instructor
&
Udemy
Certificate no: UC-4PNNHQOS
Certificate url: ude.my/UC-4PNNHQOS
#BeAble

Acknowledgement

I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude I give to our Prof. S.K Sharma of MITRC, Alwar . Whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project especially in writing this report.

Furthermore I would also like to acknowledge with much appreciation the crucial role of Mr. Manish Kumar Mukhijia, for creating constant push to move forward who gave the permission to use all required equipment and the necessary materials to complete the task and I am also grateful to our HOD-CSE Mr. Arvind Sharma for the prompt and very valuable support to necessary infrastructure required for the project work.

Last but not least, many thanks go to the head of the project,Mr. Shashikant who have invested his full effort in guiding the team in achieving the goal. I have to appreciate the guidance given by other supervisor as well as the panels especially in our project presentation that has improved our presentation skills thanks to their comment and advice.

Thank you
Prachi Khandelwal

Abstract

Python Technology

With the advent of Information Technology in the last decade, the major focus has shifted from manual systems to computerised systems. Now this time is the era of connecting people's from one place to another ,so that they can share their ideas , interest of knowledge and can help in grooming of other person . My " Tech Blog Website" is an online platform to do such things here people can share their ideas regarding the technical trends and science emergencing world .I have developed this website using Python , Django and Javascript as backend and Sql for data structure. HTML, CSS and Bootstrap for Frontend.

Contents

List of Figures

List of Tables

Chapter 1

Python Introduction

1.1 What is Python ?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.
It is used for

- web development (server-side)
- software development
- mathematics
- system scripting

1.2 What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.

- Python can be used for rapid prototyping, or for production-ready software development.

1.3 Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

1.4 Python Enviroment Setup

We have set up the Python Programming environment online, so that you can compile and execute all the available examples online. It will give you the confidence in what you are reading and will enable you to verify the programs with different options. Feel free to modify any example and execute it online. Try the following example using our online compiler available at Coding Ground

$$\boxed{/user/bin/python3/print(\text{"}Hello, Python!\text{"})}$$

For most of the examples given in this tutorial, you will find a Try it option on our website code sections, at the top right

corner that will take you to the online compiler. Just use it and enjoy your learning.
Python 3 is available for Windows, Mac OS and most of the flavors of Linux operating system. Even though Python 2 is available for many other OSs, Python 3 support either has not been made available for them or has been dropped

Chapter 2

Python Syntax

2.1 Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

Example

$$if \quad 5 > 2 : print(" Five is greater than two!")$$

2.2 Python Variables

In Python variables are created the moment you assign a value to it:

Python has no command for declaring a variable.

2.3 Comments

Python has commenting capability for the purpose of in-code documentation.
Comments start with a # and Python will render the rest of the line as a comment:

2.4 Reserved Keyword

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

Table 2.1: Reserved Keyword

Chapter 3

Python Data Types

3.1 Built-in Data Types

In programming, data type is an important concept.
Variables can store data of different types, and different types can do different things.
Python has the following data types built-in by default, in these categories:

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`

Figure 3.1: Builtin Datatype

Chapter 4

Python Numbers

4.1 Numbers

There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them:

Example

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

To verify the type of any object in Python, use the type() function:

Example

```
print(type(x))
print(type(y))
print(type(z))
```

4.1.1 Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example

Integers:

```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

4.1.2 Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Example

Floats:

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

4.1.3 Complex

Complex numbers are written with a "j" as the imaginary part:

Example

Complex:

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

4.2 Type Conversion

You can convert from one type to another with the int(), float(), and complex() methods:

Example

Convert from one type to another:

```
x = 1 # int
y = 2.8 # float
z = 1j # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
```

4.3 Random Number

Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

Example

Import the random module, and display a random number between 1 and 9:

```
import random

print(random.randrange(1,10))
```

Chapter 5

Python Casting

5.1 Specify a Variable Type

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.
Casting in python is therefore done using constructor functions:

- int() - **constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number)**
- **float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)**
- **str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals**

Chapter 6

Python Strings

6.1 String Literals

String literals in python are surrounded by either single quotation marks, or double quotation marks.
'hello' is the same as "hello".
You can display a string literal with the print() function:

Example

```
print("Hello")
print('Hello')
```

6.2 Strings are Arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.
However, Python does not have a character data type, a single character is simply a string with a length of 1.
Square brackets can be used to access elements of the string.

Example

Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, World!"
print(a[1])
```

6.3 Slicing

You can return a range of characters by using the slice syntax. Specify the start index and the end index, separated by a colon, to return a part of the string.

Example

Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!"
print(b[2:5])
```

6.4 Negative Indexing

Use negative indexes to start the slice from the end of the string:

Example

Get the characters from position 5 to position 1, starting the count from the end of the string:

```
b = "Hello, World!"
print(b[-5:-2])
```

6.5 String Length

To get the length of a string, use the len() function.

Example

The `len()` function returns the length of a string:

```
a = "Hello, World!"
print(len(a))
```

6.6 String Methods

Python has a set of built-in methods that you can use on strings.

6.6.1 Strip

Example

The `strip()` method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

6.6.2 Lower

Example

The `lower()` method returns the string in lower case:

```
a = "Hello, World!"
print(a.lower())
```

6.6.3 Upper

Example

The `upper()` method returns the string in upper case:

```
a = "Hello, World!"
print(a.upper())
```

6.6.4 Concatenation

Example

Merge variable `a` with variable `b` into variable `c`:

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

6.7 Methods

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string

Figure 6.1: Built-in methods

Chapter 7

Python Operators

Operators are used to perform operations on variables and values.
Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

7.1 Arithmetic Opertaors

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x+y
-	Subtraction	x-y
*	Multiplication	x*y
/	Division	x/y
**	Exponentiation	x**y

Table 7.1: Arithmetic Operators

7.2 Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example
=	x=5
+=	x+=3
-=	x-=3
*=	x*=3
/=	x/=5

Table 7.2: Assignment Operator

7.3 Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x==y
!=	Not equal	x!=y
¿	Greater than	x¿y
¡	less than	x¡y
¿=	Greater than or equal to	x¿=y
¡=	Less than or equal to	x¡=y

Table 7.3: Comparison Operators

7.4 Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x ¡ 5 and x ¡ 10
or	Returns True if one of the statements is true	x ¡ 5 or x ¡ 4
not	Reverse the result, returns False if the result is true	(x ¡ 5 and x ¡ 10)

Table 7.4: Logical Operators

7.5 Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Opertaor	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
¡¡	Zero fill left shift	Shift left by pushing zeros
¿¿	Signed right shift	Shift right by pushing bits fall off

Table 7.5: Bitwise Operators

Chapter 8

Python Lists

8.1 Python Collections (Arrays)

There are four collection data types in the Python programming language:

- List: **It is a collection which is ordered and changeable. Allows duplicate members.**
- **Tuple: It is a collection which is ordered and unchangeable. Allows duplicate members.**
- **Set: It is a collection which is unordered and unindexed. No duplicate members.**
- **Dictionary: It is a collection which is unordered, changeable and indexed. No duplicate members.**

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

8.2 List

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

Example

Create a List:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

8.2.1 Negative Indexing

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

8.2.2 Loop Through a List

You can loop through the list items by using a for loop:

Example

Print all items in the list, one by one:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

8.2.3 List Length

To determine how many items a list has, use the len() method:

Example

Print the number of items in the list:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

8.3 List Methods

Method	Description
append( )	Adds an element at the end of the list
clear( )	Removes all the elements from the list
copy( )	Returns a copy of the list
count( )	Returns the number of elements with the specified value
extend( )	Add the elements of a list (or any iterable), to the end of the current list
index( )	Returns the index of the first element with the specified value
insert( )	Adds an element at the specified position
pop( )	Removes the element at the specified position
remove( )	Removes the item with the specified value

Table 8.1: List Method

Chapter 9

Python Tuples

9.1 Tuple

A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.

Example

Create a Tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

9.1.1 Access tuple Items

You can access tuple items by referring to the index number, inside square brackets:

Example

Print the second item in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

9.1.2 Negetive Indexing

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

Example

Return the third, fourth, and fifth item:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

9.1.3 Range Of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.
When specifying a range, the return value will be a new tuple with the specified items.

Example

Return the third, fourth, and fifth item:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

Chapter 10

Python Sets

10.1 Set

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

Example

Create a Set:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

10.1.1 Access Items

You cannot access items in a set by referring to an index, since sets are unordered the items has no index.
But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Example

Loop through the set, and print the values:

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

10.1.2 Add Items

To add one item to a set use the add() method.
To add more than one item to a set use the update() method.

Example

Add an item to a set, using the `add()` method:

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

10.2 Set Methods

Python has a set of built-in methods that you can use on sets.

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set

Table 10.1: Set Methods

Chapter 11

Python Dictionaries

11.1 Dictionary

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

Example

Create and print a dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

11.1.1 Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

Example

Get the value of the "model" key:

```
x = thisdict["model"]
```

11.1.2 Loop Through a Dictionary

You can loop through a dictionary by using a for loop.
When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

Example

Print all key names in the dictionary, one by one:

```
for x in thisdict:
  print(x)
```

11.1.3 Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

Example

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

11.1.4 Removing Items

There are several methods to remove items from a dictionary:

Example

The `pop()` method removes the item with the specified key name:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

11.1.5 Copy a Dictionary

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.
There are ways to make a copy, one way is to use the built-in Dictionary method copy().

Example

Make a copy of a dictionary with the `copy()` method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

11.1.6 Nested Dictionary

A dictionary can also contain many dictionaries, this is called nested dictionaries.

Example

Create a dictionary that contain three dictionaries:

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
```

11.2 Dictionary Methods

Method	Description
clear( )	Removes all the elements from the dictionary
copy( )	Returns a copy of the dictionary
fromkeys( )	Returns a dictionary with the specified keys and values
get( )	Returns the value of the specified key
items( )	Returns a list containing the a tuple for each key value pair
keys( )	Returns a list containing the dictionary's keys
pop( )	Removes the element with the specified key
popitem( )	Removes the last inserted key-value pair

Table 11.1: Dictionary Methods

Chapter 12

Python If ... Else

12.0.1 Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a ¡ b
- Less than or equal to: a ¡= b
- Greater than: a ¿ b
- Greater than or equal to: a ¿= b

These conditions can be used in several ways, most commonly in "if statements" and loops.
An "if statement" is written by using the if keyword.

Example

If statemert:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

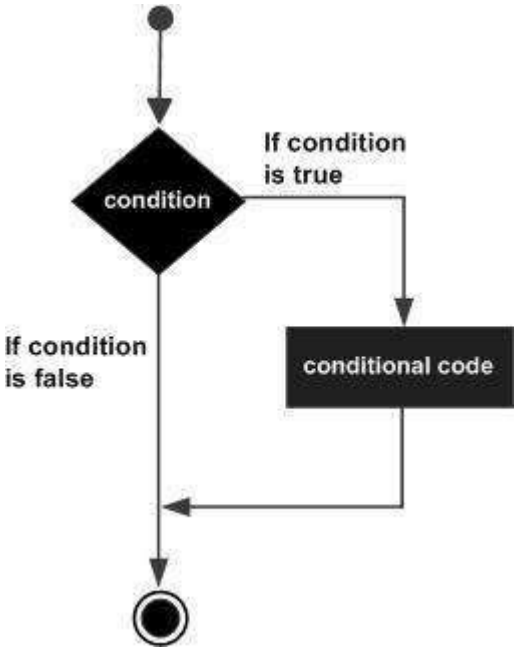

Figure 12.1: If Condition

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

12.0.2 Elif

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

Example

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

In this example a is equal to b, so the first condition is not true,

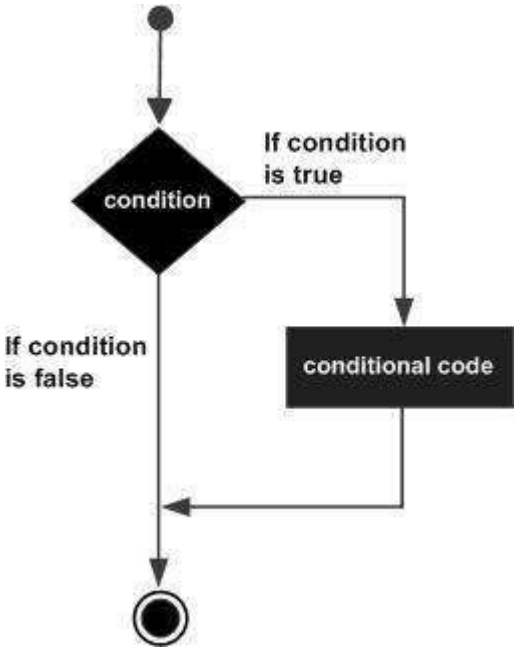

Figure 12.2: If-Else Condition

but the elif condition is true, so we print to screen that "a and b are equal".

12.0.3 Else

The else keyword catches anything which isn't caught by the preceding conditions.

Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

In this example a is greater than b, so the first condition is not true, also the elif condition is not true

Chapter 13

Python Loops

Python has two primitive loop commands:

- While Loops
- For Loops

13.1 The while Loop

Here, statement(s) may be a single statement or a block of statements with uniform indent. The condition may be any expression, and true is any non-zero value.
The loop iterates while the condition is true. Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

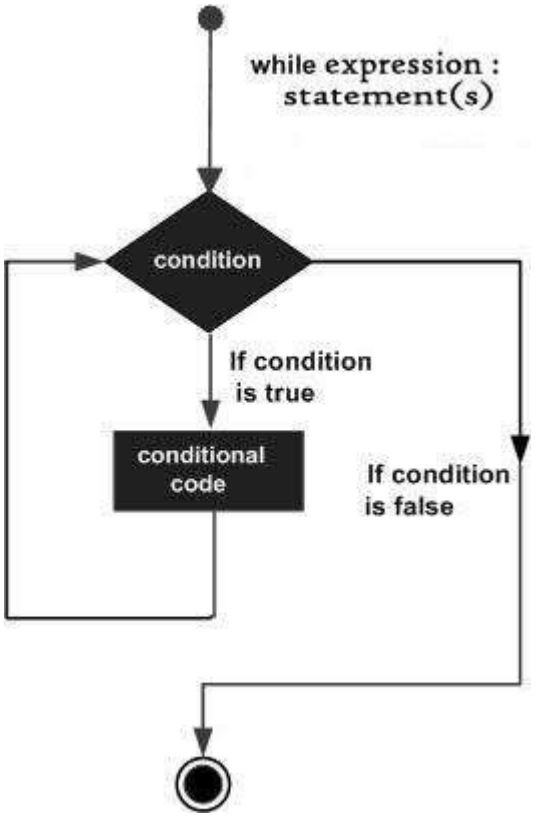

Figure 13.1: While Loop

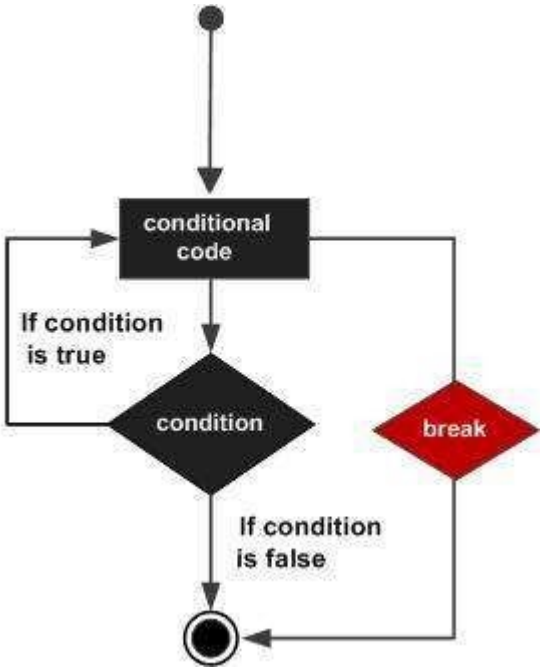

Figure 13.2: Break

13.2 The break Statement

The break statement is used for premature termination of the current loop. After abandoning the loop, execution at the next statement is resumed, just like the traditional break statement in C.
The most common use of break is when some external condition is triggered requiring a hasty exit from a loop. The break statement can be used in both while and for loops.

13.3 For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
This is less like the for keyword in other programming

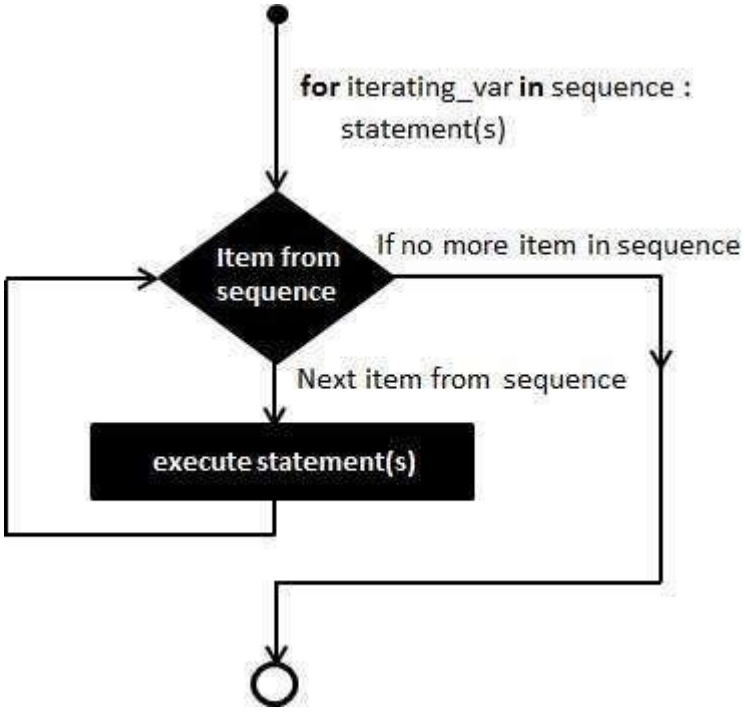

Figure 13.3: For Loops

languages, and works more like an iterator method as found in other object-orientated programming languages.
With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Chapter 14

Python Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.
As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.

14.1 Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.

- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

14.2 Recursion

Python also accepts function recursion, which means a defined function can call itself.
Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.
The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.
In this example is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).
To a new developer it can take some time to work out how exactly this works, best way to find out is by testing and modifying it.

Chapter 15

Python Classes and Object

15.1 Class

Compared with other programming languages, Python's class mechanism adds classes with a minimum of new syntax and semantics. It is a mixture of the class mechanisms found in C++ and Modula-3. Python classes provide all the standard features of Object Oriented Programming: the class inheritance mechanism allows multiple base classes, a derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name. Objects can contain arbitrary amounts and kinds of data. As is true for modules, classes partake of the dynamic nature of Python: they are created at runtime, and can be modified further after creation.
In C++ terminology, normally class members (including the data members) are public (except see below Private Variables), and all member functions are virtual. As in Modula-3, there are no shorthands for referencing the object's members from its methods: the method function is declared with an explicit first argument representing the object, which is provided implicitly by the call. As in Smalltalk, classes themselves are objects. This provides semantics for importing and renaming. Unlike

C++ and Modula-3, built-in types can be used as base classes for extension by the user. Also, like in C++, most built-in operators with special syntax (arithmetic operators, subscripting etc.) can be redefined for class instances. (Lacking universally accepted terminology to talk about classes, I will make occasional use of Smalltalk and C++ terms. I would use Modula-3 terms, since its object-oriented semantics are closer to those of Python than C++, but I expect that few readers have heard of it.)

Example

Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5
```

15.2 Class Object

Class objects support two kinds of operations: attribute references and instantiation.
Attribute references use the standard syntax used for all attribute references in Python: obj.name. Valid attribute names are all the names that were in the class's namespace when the class object was created. So, if the class definition looked like this:

Example

Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)
```

Chapter 16

Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.
Parent class is the class being inherited from, also called base class.
Child class is the class that inherits from another class, also called derived class.
The name BaseClassName must be defined in a scope containing the derived class definition. In place of a base class name, other arbitrary expressions are also allowed. This can be useful, for example, when the base class is defined in another module:

Execution of a derived class definition proceeds the same as for a base class. When the class object is constructed, the base class is remembered. This is used for resolving attribute references: if a requested attribute is not found in the class, the search proceeds to look in the base class. This rule is applied recursively if the base class itself is derived from some other class.
There's nothing special about instantiation of derived classes: DerivedClassName() creates a new instance of the class.
Method references are resolved as follows: the corresponding class attribute is searched, descending down the chain of base

classes if necessary, and the method reference is valid if this yields a function object.
Derived classes may override methods of their base classes. Because methods have no special privileges when calling other methods of the same object, a method of a base class that calls another method defined in the same base class may end up calling a method of a derived class that overrides it. (For C++ programmers: all methods in Python are effectively virtual.) An overriding method in a derived class may in fact want to extend rather than simply replace the base class method of the same name. There is a simple way to call the base class method directly: just call BaseClassName.methodname(self, arguments). This is occasionally useful to clients as well. (Note that this only works if the base class is accessible as BaseClassName in the global scope.)

Example

Create a class named Person, with firstname and lastname properties, and a printname method:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname method:

x = Person("John", "Doe")
x.printname()
```

Chapter 17

OOP's Terminology

- **Class : A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.**
- **Class Variable : A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.**
- **Data Member : A class variable or instance variable that holds data associated with a class and its objects.**
- **Function overloading : The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects or arguments involved.**
- **Instance Variable : A variable that is defined inside a method and belongs only to the current instance of a class.**

- **Inheritance : The transfer of the characteristics of a class to other classes that are derived from it.**
- **Object : A unique instance of a data structure that is defined by its class. An object comprises both data members (class variables and instance variables) and methods.**

Chapter 18

Python Module

If you quit from the Python interpreter and enter it again, the definitions you have made (functions and variables) are lost. Therefore, if you want to write a somewhat longer program, you are better off using a text editor to prepare the input for the interpreter and running it with that file as input instead. This is known as creating a script. As your program gets longer, you may want to split it into several files for easier maintenance. You may also want to use a handy function that you've written in several programs without copying its definition into each program.
To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module; definitions from a module can be imported into other modules or into the main module (the collection of variables that you have access to in a script executed at the top level and in calculator mode).

Example

Save this code in a file named mymodule.py

```
def greeting(name):
  print("Hello, " + name)
```

18.0.1 Module Search Path

When a module named spam is imported, the interpreter first searches for a built-in module with that name. If not found, it then searches for a file named spam.py in a list of directories given by the variable sys.path. sys.path is initialized from these locations:
the directory containing the input script (or the current directory). PYTHONPATH (a list of directory names, with the same syntax as the shell variable PATH). the installation-dependent default.
After initialization, Python programs can modify sys.path. The directory containing the script being run is placed at the beginning of the search path, ahead of the standard library path. This means that scripts in that directory will be loaded instead of modules of the same name in the library directory. This is an error unless the replacement is intended. See section Standard Modules for more information.

18.0.2 Built-in Module

Python comes with a library of standard modules, described in a separate document, the Python Library Reference ("Library Reference" hereafter). Some modules are built into the interpreter; these provide access to operations that are not part of the core of the language but are nevertheless built in, either for efficiency or to provide access to operating system primitives such as system calls. The set of such modules is a configuration option which also depends on the underlying platform. For example, the winreg module is only provided on Windows systems. One particular module deserves some attention: sys, which is built into every Python interpreter. The variables sys.ps1 and sys.ps2 define the strings used as primary and secondary prompts:

Example

Import and use the `platform` module:

```
import platform

x = platform.system()
print(x)
```

Each module has its own private symbol table, which is used as the global symbol table by all functions defined in the module. Thus, the author of a module can use global variables in the module without worrying about accidental clashes with a user's global variables. On the other hand, if you know what you are doing you can touch a module's global variables with the same notation used to refer to its functions, modname.itemname.

Modules can import other modules. It is customary but not required to place all import statements at the beginning of a module (or script, for that matter). The imported module names are placed in the importing module's global symbol table.

There is a variant of the import statement that imports names from a module directly into the importing module's symbol table.

Chapter 19

DJANGO -Basics

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Django makes it easier to build better web apps quickly and with less code.

19.1 History Of Django

- **2003 : Started by Adrian Holovaty and Simon Willison as an internal project at the Lawrence Journal-World newspaper**
- **2005 : Released July 2005 and named it Django, after the jazz guitarist Django Reinhardt.**
- **2005 : Mature enough to handle several high-traffic sites.**
- **Current : Django is now an open source project with contributors across the world**

19.2 Advanatges of Django

Here are few advantages of using Django which can be listed out here

- Django provides a bridge between the data model and the database engine, and supports a large set of database systems including MySQL, Oracle, Postgres, etc. Django also supports NoSQL database through Django-nonrel fork. For now, the only NoSQL databases supported are MongoDB and google app engine.
- Django supports multilingual websites through its built-in internationalization system. So you can develop your website, which would support multiple languages.
- Django has built-in support for Ajax, RSS, Caching and various other frameworks.
- Django provides a nice ready-to-use user interface for administrative activities.
- Django comes with a lightweight web server to facilitate end-to-end application development and testing .

19.3 Django-Enviroment

Django development environment consists of installing and setting up Python, Django, and a Database System. Since Django deals with web application, it's worth mentioning that you would need a web server setup as well.
Django is written in 100 percent pure Python code, so you'll need to install Python on your system. Latest Django version requires Python 2.6.5 or higher for the 2.6.x branch or higher than 2.7.3 for the 2.7.x branch.
If you're on one of the latest Linux or Mac OS X distribution, you probably already have Python installed. You can verify it by typing python command at a command prompt. If you see something like this, then Python is installed.

Installing Django is very easy, but the steps required for its installation depends on your operating system. Since Python is a platform-independent language, Django has one package that works everywhere regardless of your operating system. You can download the latest version of Django from the link http://www.djangoproject.com/download.
Django supports several major database engines and you can set up any of them based on your comfort.

- MySQL (http://www.mysql.com/)
- PostgreSQL (http://www.postgresql.org/)
- SQLite 3 (http://www.sqlite.org/)
- MongoDb (https://django-mongodb-engine.readthedocs.org)

Django comes with a lightweight web server for developing and testing applications. This server is pre-configured to work with Django, and more importantly, it restarts whenever you modify the code.
However, Django does support Apache and other popular web servers such as Lighttpd. We will discuss both the approaches in coming chapters while working with different examples.

19.4 Creating Models

A model is the single, definitive source of truth about your data. It contains the essential fields and behaviors of the data you're storing. Django follows the DRY Principle. The goal is to define your data model in one place and automatically derive things from it.
This includes the migrations - unlike in Ruby On Rails, for example, migrations are entirely derived from your models file,

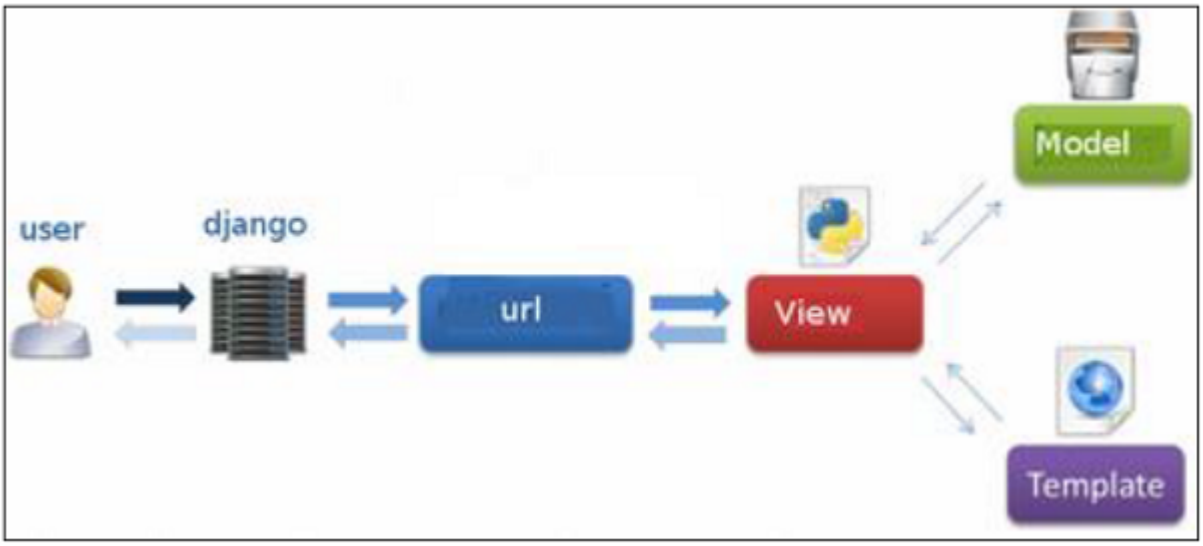

Figure 19.1: Django WebServer

and are essentially just a history that Django can roll through to update your database schema to match your current models.

```
from django.db import models


class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')


class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

Bibliography

[1] W3School *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[2] Python Documentation