

Python Design Patterns

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

About the Tutorial

This tutorial explains the various types of design patterns and their implementation in Python scripting language. This tutorial will take you through a roller coaster ride with different approaches and examples using Python concepts.

Audience

This tutorial is aimed to benefit both basic and intermediate levels of programmers and developers.

Prerequisites

Before you proceed with this tutorial, it is assumed that the user is already aware about basic python programming concepts.

Copyright & Disclaimer

© Copyright 2018 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial	i
Audience.....	i
Prerequisites.....	i
Copyright & Disclaimer	i
Table of Contents	ii
1. Python Design Patterns – Introduction	1
Structure of a design pattern	1
Why Python?	2
2. Python Design Patterns – Gist of Python.....	4
Features of Python Language	4
Important Points.....	4
How to download python language in your system?	6
The Important Tools in Python.....	6
What constitutes a design pattern in Python?.....	7
3. Python Design Patterns – Model View Controller Pattern.....	8
4. Python Design Patterns – Singleton Pattern.....	12
How to implement a singleton class?.....	12
5. Python Design Patterns – Factory Pattern.....	15
How to implement a factory pattern?.....	15
6. Python Design Patterns – Builder Pattern	17
How to implement builder pattern?	17
7. Python Design Patterns – Prototype Pattern.....	21
How to implement a prototype pattern?	21
8. Python Design Patterns – Facade Pattern	25
How to design a facade pattern?	25
9. Python Design Patterns – Command Pattern	32

How to implement the command pattern? 32

10. Python Design Patterns – Adapter Pattern..... 35

How to implement the adapter pattern? 35

11. Python Design Patterns – Decorator Pattern..... 39

How to implement decorator design pattern 39

12. Python Design Patterns – Proxy Pattern 44

How to implement the proxy pattern?..... 44

13. Python Design Patterns – Chain of Responsibility Pattern..... 46

How to implement the chain of responsibility pattern? 46

14. Python Design Patterns – Observer Pattern 49

How to implement the observer pattern? 49

15. Python Design Patterns – State Pattern 52

How to implement the state pattern? 52

16. Python Design Patterns – Strategy Pattern 55

How to implement the strategy pattern? 55

17. Python Design Patterns – Template Pattern 57

18. Python Design Patterns – Flyweight Pattern 60

How to implement the flyweight pattern?..... 60

19. Python Design Patterns – Abstract Factory Pattern..... 62

How to implement the abstract factory pattern? 62

20. Python Design Patterns – Object Oriented Pattern 66

How to implement the object oriented pattern?..... 66

21. Python Design Patterns – Object Oriented Concepts Implementation 68

How to implement classes and object variables? 68

22. Python Design Patterns – Iterator Pattern 71

How to implement the iterator pattern? 71

23. Python Data Structure – Dictionaries 74

How to implement dictionaries in Python?..... 74



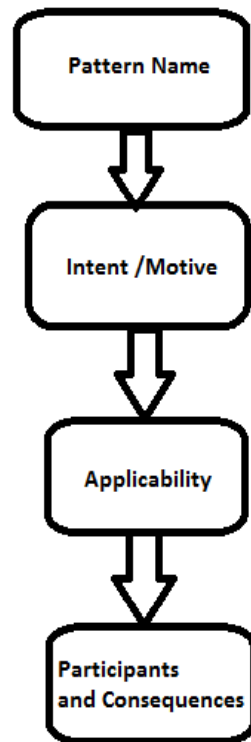
24. Python Design Pattern – The Lists Data Structure	76
How to implement lists?.....	76
25. Python Design Patterns – Sets.....	78
How to implement sets?	78
26. Python Design Patterns – Queues	82
How to implement the FIFO procedure?.....	82
How to implement the LIFO procedure?	83
What is a Priority Queue?.....	84
27. Python Design Patterns – Strings and Serialization	86
28. Python Design Patterns – Concurrency in Python	88
29. Python Design Patterns – Anti-Patterns	91
Important features of anti-patterns	91
30. Python Design Patterns – Exception Handling.....	94
Why use exceptions?.....	94

1. Python Design Patterns – Introduction

Design patterns are used to represent the pattern used by developers to create software or web application. These patterns are selected based on the requirement analysis. The patterns describe the solution to the problem, when and where to apply the solution and the consequences of the implementation.

Structure of a design pattern

The documentation of design pattern is maintained in a way that focuses more on the technology that is used and in what ways. The following diagram explains the basic structure of design pattern documentation.



Pattern Name

It describes the pattern in short and effective manner.

Intent/Motive

It describes what the pattern does.

Applicability

It describes the list of situations where pattern is applicable.

Participants and consequences

Participants include classes and objects that participate in the design pattern with a list of consequences that exist with the pattern.

Why Python?

Python is an open source scripting language. It has libraries that support a variety of design patterns. The syntax of python is easy to understand and uses English keywords.

Python provides support for the list of design patterns that are mentioned below. These design patterns will be used throughout this tutorial:

- Model View Controller Pattern
- Singleton pattern
- Factory pattern
- Builder Pattern
- Prototype Pattern
- Facade Pattern
- Command Pattern
- Adapter Pattern
- Prototype Pattern
- Decorator Pattern
- Proxy Pattern
- Chain of Responsibility Pattern
- Observer Pattern
- State Pattern
- Strategy Pattern
- Template Pattern
- Flyweight Pattern
- Abstract Factory Pattern
- Object Oriented Pattern

Benefits of using design pattern

Following are the different benefits of design pattern:

- Patterns provide developer a selection of tried and tested solutions for the specified problems.
- All design patterns are language neutral.

- Patterns help to achieve communication and maintain well documentation.
- It includes a record of accomplishment to reduce any technical risk to the project.
- Design patterns are highly flexible to use and easy to understand.

2. Python Design Patterns – Gist of Python

Python is an open source scripting language, which is high-level, interpreted, interactive and object-oriented. It is designed to be highly readable. The syntax of Python language is easy to understand and uses English keywords frequently.

Features of Python Language

In this section, we will learn about the different features of Python language.

Interpreted

Python is processed at runtime using the interpreter. There is no need to compile program before execution. It is similar to PERL and PHP.

Object-Oriented

Python follows object-oriented style and design patterns. It includes class definition with various features like encapsulation, polymorphism and many more.

Portable

Python code written in Windows operating system and can be used in Mac operating system. The code can be reused and portable as per the requirements.

Easy to code

Python syntax is easy to understand and code. Any developer can understand the syntax of Python within few hours. Python can be described as “programmer-friendly”

Extensible

If needed, a user can write some of Python code in C language as well. It is also possible to put python code in source code in different languages like C++. This makes Python an extensible language.

Important Points

Consider the following important points related to Python programming language:

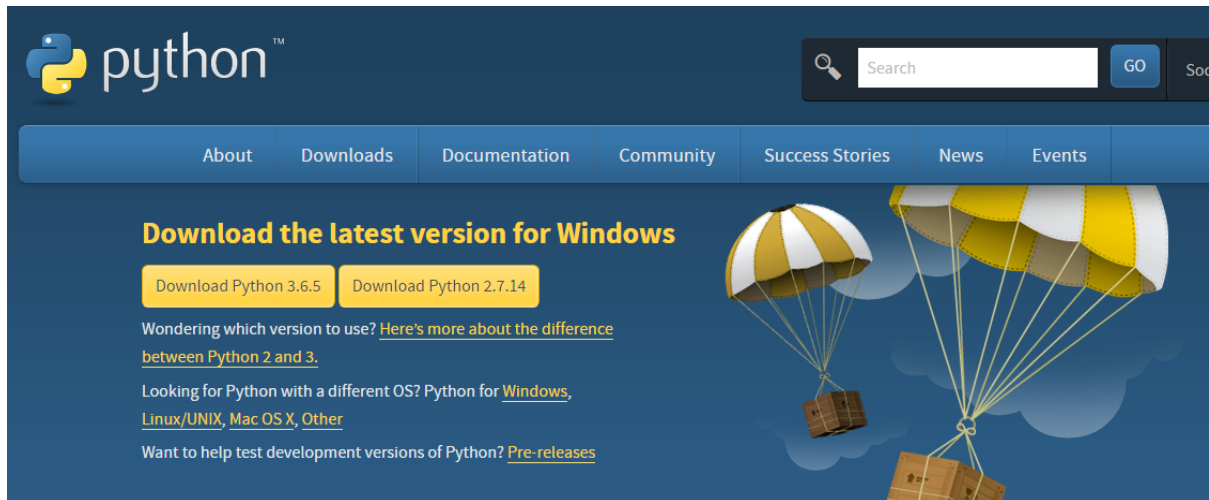
- It includes functional and structured programming methods as well as object-oriented programming methods.
- It can be used as scripting language or as a programming language.
- It includes automatic garbage collection.

- It includes high-level dynamic data types and supports various dynamic type checking.
- Python includes a feature of integration with C, C++ and languages like Java.

How to download python language in your system?

To download Python language in your system, follow this link:

<https://www.python.org/downloads/>



It includes packages for various operating systems like Windows, MacOS and Linux distributions.

The Important Tools in Python

In this section, we will learn in brief about a few important tools in Python.

Python Strings

The basic declaration of strings is as follows:

```
str = 'Hello World!'
```

Python Lists

The lists of python can be declared as compound data types separated by commas and enclosed within square brackets ([]).

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]  
tinylist = [123, 'john']
```

Python Tuples

A tuple is dynamic data type of Python, which consists of number of values separated by commas. Tuples are enclosed with parentheses.

```
tinytuple = (123, 'john')
```

Python Dictionary

Python dictionary is a type of hash table. A dictionary key can be almost any data type of Python. The data types are usually numbers or strings.

```
tinydict = {'name': 'omkar', 'code': 6734, 'dept': 'sales'}
```

What constitutes a design pattern in Python?

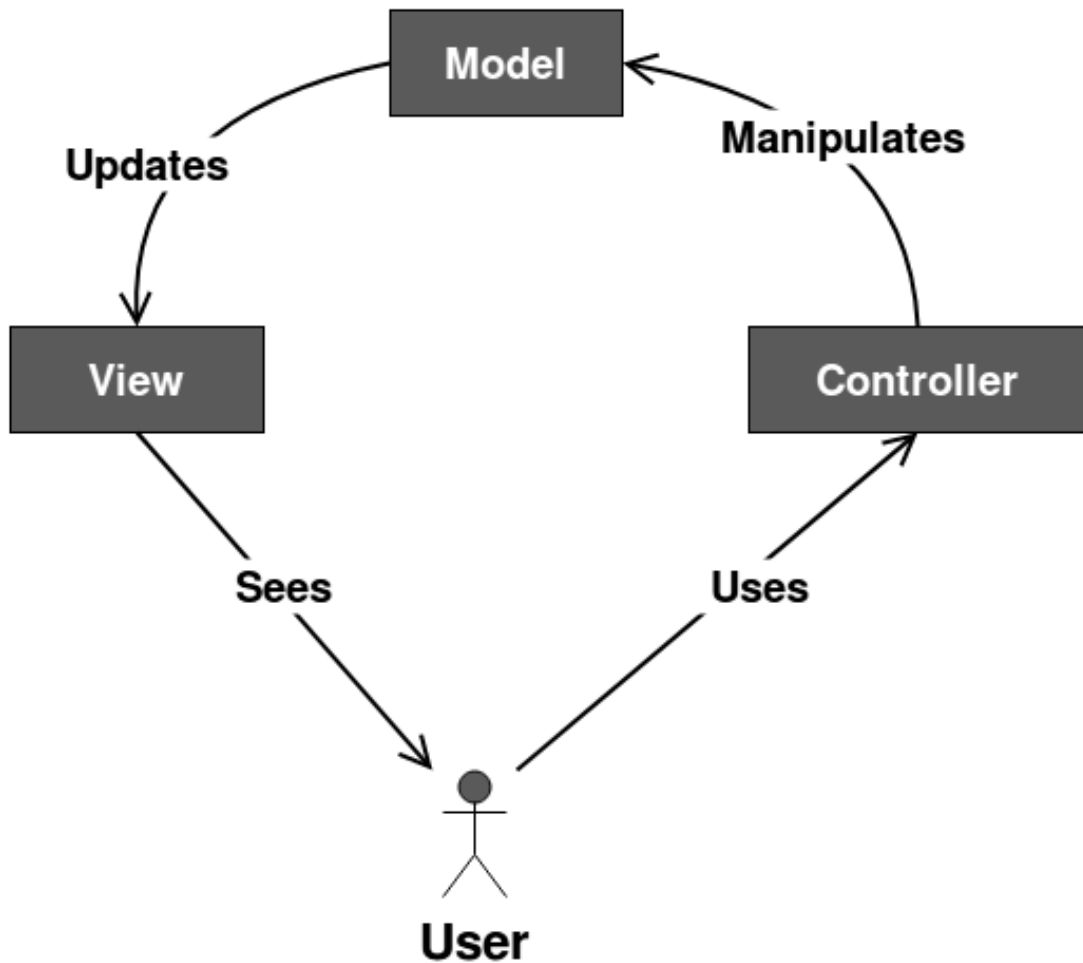
Python helps in constituting a design pattern using the following parameters:

- Pattern Name
- Intent
- Aliases
- Motivation
- Problem
- Solution
- Structure
- Participants
- Constraints
- Sample Code

3. Python Design Patterns – Model View Controller Pattern

Model View Controller is the most commonly used design pattern. Developers find it easy to implement this design pattern.

Following is a basic architecture of the Model View Controller:



Let us now see how the structure works.

Model

It consists of pure application logic, which interacts with the database. It includes all the information to represent data to the end user.

View

View represents the HTML files, which interact with the end user. It represents the model's data to user.

Controller

It acts as an intermediary between view and model. It listens to the events triggered by view and queries model for the same.

Python code

Let us consider a basic object called "Person" and create an MVC design pattern.

Model.py

```
import json

class Person(object):

    def __init__(self, first_name = None, last_name = None):
        self.first_name = first_name
        self.last_name = last_name
        #returns Person name, ex: John Doe
    def name(self):
        return ("%s %s" % (self.first_name,self.last_name))

    @classmethod
    #returns all people inside db.txt as list of Person objects
    def getAll(self):
        database = open('db.txt', 'r')
        result = []
        json_list = json.loads(database.read())
        for item in json_list:
            item = json.loads(item)
            person = Person(item['first_name'], item['last_name'])
            result.append(person)
        return result
```

It calls for a method, which fetches all the records of the Person table in database. The records are presented in JSON format.

View

It displays all the records fetched within the model. View never interacts with model; controller does this work (communicating with model and view).

```

from model import Person

def showAllView(list):
    print 'In our db we have %i users. Here they are:' % len(list)
    for item in list:
        print item.name()
def startView():
    print 'MVC - the simplest example'
    print 'Do you want to see everyone in my db?[y/n]'

def endView():
    print 'Goodbye!'

```

Controller

Controller interacts with model through the **getAll()** method which fetches all the records displayed to the end user.

```

from model import Person
import view

def showAll():
    #gets list of all Person objects
    people_in_db = Person.getAll()
    #calls view
    return view.showAllView(people_in_db)

def start():
    view.startView()
    input = raw_input()
    if input == 'y':
        return showAll()
    else:

```

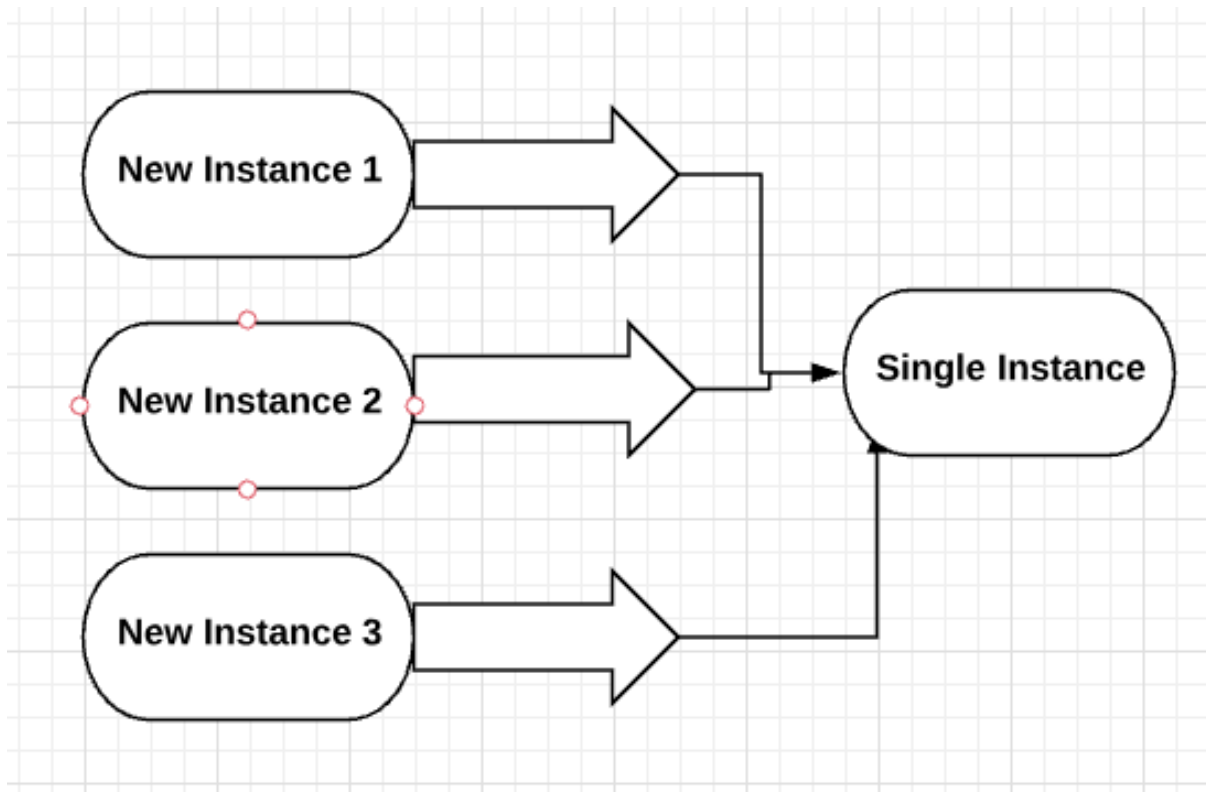
```
        return view.endView()

if __name__ == "__main__":
    #running controller function
    start()
```


4. Python Design Patterns – Singleton Pattern

This pattern restricts the instantiation of a class to one object. It is a type of creational pattern and involves only one class to create methods and specified objects.

It provides a global point of access to the instance created.



How to implement a singleton class?

The following program demonstrates the implementation of singleton class where it prints the instances created multiple times.

```
class Singleton:
    __instance = None

    @staticmethod
    def getInstance():
        """ Static access method. """
        if Singleton.__instance == None:
            Singleton()
```

12

```
        return Singleton.__instance

    def __init__(self):
        """ Virtually private constructor. """
        if Singleton.__instance != None:
            raise Exception("This class is a singleton!")
        else:
            Singleton.__instance = self

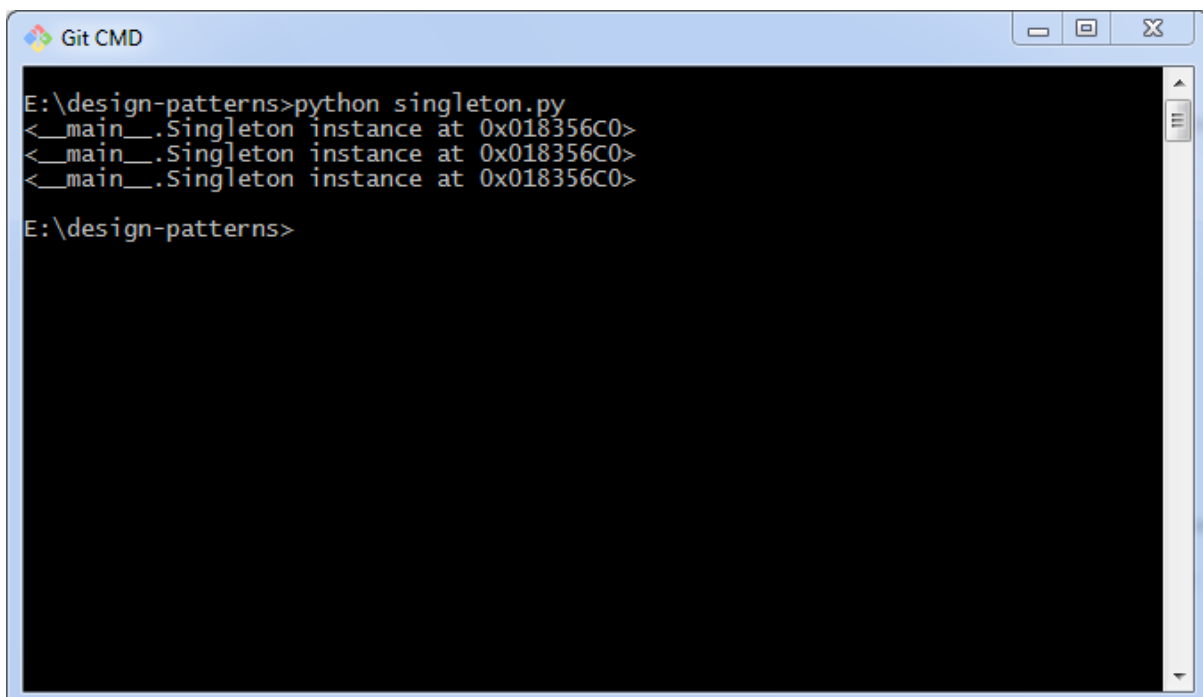
s = Singleton()
print s

s = Singleton.getInstance()
print s

s = Singleton.getInstance()
print s
```

Output

The above program generates the following output:



```
Git CMD
E:\design-patterns>python singleton.py
<__main__.Singleton instance at 0x018356C0>
<__main__.Singleton instance at 0x018356C0>
<__main__.Singleton instance at 0x018356C0>
E:\design-patterns>
```

The number of instances created are same and there is no difference in the objects listed in output.

5. Python Design Patterns – Factory Pattern

The factory pattern comes under the creational patterns list category. It provides one of the best ways to create an object. In factory pattern, objects are created without exposing the logic to client and referring to the newly created object using a common interface.

Factory patterns are implemented in Python using factory method. When a user calls a method such that we pass in a string and the return value as a new object is implemented through factory method. The type of object used in factory method is determined by string which is passed through method.

In the example below, every method includes object as a parameter, which is implemented through factory method.

How to implement a factory pattern?

Let us now see how to implement a factory pattern.

```
class Button(object):
    html = ""
    def get_html(self):
        return self.html

class Image(Button):
    html = "<img></img>"

class Input(Button):
    html = "<input></input>"

class Flash(Button):
    html = "<obj></obj>"

class ButtonFactory():
    def create_button(self, typ):
        targetclass = typ.capitalize()
        return globals()[targetclass]()

button_obj = ButtonFactory()
```

15

```
button = ['image', 'input', 'flash']
for b in button:
    print button_obj.create_button(b).get_html()
```

The button class helps to create the html tags and the associated html page. The client will not have access to the logic of code and the output represents the creation of html page.

End of ebook preview

If you liked what you saw...

Buy it from our store @ <https://store.tutorialspoint.com>