Programming with Python

Dr. Soharab Hossain Shaikh
BML Munjal University

https://github.com/soharabhossain/Python

1

Introduction to Python

- Python is a high-level programming language
- General purpose and object-oriented language
- Open source and community driven
- "Batteries Included" - a standard distribution includes many modules
- Dynamic typed
- Source can be compiled or run just-in-time

2

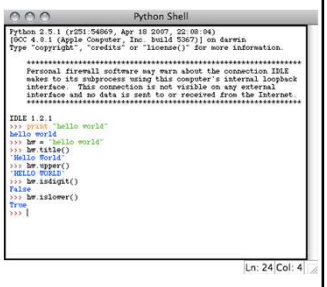
More than just printing

- Python is an object-oriented language
- Practically everything can be treated as an object
- "hello world" is a string
- Strings, as objects, have methods that return the result of a function on the string

3

String Methods

- Assign a string to a variable
- In this case "`hw`"
- `hw.title()`
- `hw.upper()`
- `hw.isdigit()`
- `hw.islower()`



4

String Methods

- The string held in your variable remains the same
- The method returns an altered string
- Changing the variable requires reassignment
  - `hw = hw.upper()`
  - `hw` now equals "HELLO WORLD"

5

Python DS

- **Lists** (mutable sets of strings)
  - `var = [] # create list`
  - `var = ['one', 2, 'three', 'banana']`
- **Tuples** (immutable sets)
  - `var = ('one', 2, 'three', 'banana')`
- **Dictionaries** (associative arrays or 'hashes')
  - `var = {} # create dictionary`
  - `var = {'lat': 40.20547, 'lon': -74.76322}`
  - `var['lat'] = 40.2054`
- **Set** (mutable collection of unique elements)
  - `var = {'one', 2, 'three', 'banana'}`
- Each has its own set of methods

6

Lists

- Think of a list as a stack of cards, on which your information is written
- The information stays **in the order** you place it in until you modify that order
- Methods return a string or subset of the list or modify the list to add or remove components
- Written as var[*index*], index refers to order within set (think card number, starting at 0)
- You can step through lists as part of a loop

7

List Methods

- Adding to the List
  - var[*n*] = *object*
    - replaces *n* with *object*
  - var.append(*object*)
    - adds *object* to the end of the list
- Removing from the List
  - var[*n*] = []
    - empties contents of card, but preserves order
  - var.remove(*n*)
    - removes first item with the specified value *n*
  - var.pop(*n*)
    - Removes element from position *n* and returns its value
  - var.extend()
  - Adds the elements of a list (or any iterable) to the end of the current list

8

Tuples

- Like a list, tuples are iterable arrays of objects
- Tuples are immutable – once created, unchangeable
- To add or remove items, you must redeclare
- Example uses of tuples
  - Country Names
  - Ordered set of functions

9

Dictionaries

- Dictionaries are sets of key & value pairs
- Allows you to identify values by a descriptive name instead of order in a list
- Keys are unordered unless explicitly sorted
- Keys are unique:
  - var['item'] = "apple"
  - var['item'] = "banana"
  - print var['item'] prints just banana

10

Set

- **Set** is a collection which is unordered and unindexed.
- No duplicate members.
- ```
  set1 = {"apple", "banana", "cherry"}
  set2 = {1, 5, 7, 9, 3}
  set3 = {True, False, False}
  ```

11

Indentation and Blocks

- Python uses **whitespace** and **indents** to denote blocks of code
- Lines of code that begin a block end in a colon:
- Lines within the code block are indented at the same level
- To end a code block, remove the indentation
- You'll want blocks of code that run only when certain conditions are met

12
  ```

Conditional Branching

- **if** and **else**

```
if variable == condition:
        #do something based on v == c
else:
        #do something based on v != c
```

- **elif** allows for additional branching

```
if condition:
elif another condition:
…
else: #none of the above
```

13

Looping with For

- For allows you to loop over a block of code a set number of times
- For is great for manipulating lists:

```
a = ['cat', 'window', 'defenestrate']
for x in a:
    print x, len(x)
```

Results:

```
cat 3
window 6
defenestrate 12
```

14

Looping with while

```
while expression:
    statement(s)

a = ['cat', 'window', 'defenestrate']
i=0
while i<len(a):
    print(a[i])
    i+=1
```

15

Modules

- Modules are additional pieces of code that further extend Python's functionality
- A module typically has a specific function
  - additional math functions, databases, network…
- Python comes with many useful modules

16

Modules

- Modules are accessed using import
  - **import sys, os** # imports two modules
- Modules can have subsets of functions
  - **os.path** is a subset within os
- Modules are then addressed by modulename.function()
  - **sys.argv** # list of arguments
  - **filename = os.path.splitext("points.txt")**
  - **filename[1]** # equals ".txt"

17

Files

- Files are manipulated by creating a file object
  - **f = open("points.txt", "r")**
- The file object then has new methods
  - **print f.readline()** *# prints line from file*
- Files can be accessed to read or write
  - **f = open("output.txt", "w")**
  - **f.write("Important Output!")**
- Files are iterable objects, like lists

18

Error Capture

- Check for type assignment errors, items not in a list, etc.
- Try, Except and Finally

try:
a block of code that might have an error
except:
code to execute if an error occurs in "try"
finally:
this part of the code is executed after either try or catch block

- Allows for graceful failure

19

Command Line Arguments

```
# Python program to demonstrate Command-Line arguments
# This script is saved fith the following name: cmdarg.py

import sys

# Total arguments
n = len(sys.argv)
print("\n Total arguments passed:", n)

# Arguments passed
print("\n Name of Python script:", sys.argv[0])

print("\n Arguments passed:", end = " ")
for i in range(1, n):
    print(sys.argv[i], end = " ")

# Addition of numbers
Sum = 0

# Using argparse module
for i in range(1, n):
    Sum += int(sys.argv[i])

print("\n\n Result (sum of all the arguments) : ", Sum)
```

20

Command Line Arguments

```
C:\Windows\System32\cmd.exe
C:\>python cmdarg.py 1 4 5
Total arguments passed: 4
Name of Python script: cmdarg.py
Arguments passed: 1 4 5
Result (sum of all the arguments) :  10
```

21

argparse

```
# import the necessary packages  CMD_line_Args.py
import argparse

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()

# Mandatory argument
ap.add_argument("-n", "--name", required=True, help="name of the user")
# Optional argument
ap.add_argument("-s", "--surname", required=False, help="Surname of the user")

args = vars(ap.parse_args())

# display a friendly message to the user
print("Hi {}, how are you?".format(args["name"]))

# If the argument is specified
if args["surname"]:
    print(args["surname"])
else:
    print('Surname not mentioned.!!!') # If the argument is not specified
```

```
# Usage of the code from the command prompt
# python CMD_line_Args.py -- help
# python CMD_line_Args.py -n Soharab -s Shaikh
# python CMD_line_Args.py -n Soharab
```

Please check the documentation of the argparse module :
https://docs.python.org/3/library/argparse.html

22

Function & Default Argument

```
def greet(name, msg="Good morning!"):
    """
    This function greets the person with the provided message.

    If the message is not provided, it defaults to "Good morning!"
    """

    print("Hello", name + ', ' + msg)

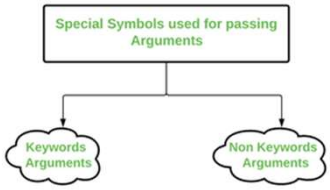
# Calling function with a single argument
greet("BMU")

# Calling function with two arguments
greet("BMU", "How do you do?")
```

```
Hello BMU, Good morning!
Hello BMU, How do you do?
```

23

Function Arguments



Special Symbols Used for passing arguments:-
1.)*args (Non-Keyword Arguments)
2.)**kwargs (Keyword Arguments)

24

```
# Python program to illustrate function arguments

# ------------------------------------------------------------
# *args with first extra argument
def myFun1(arg1, *argv):
    print ("\n\n First argument :", arg1)
    for arg in argv:
        print(" Next argument through *argv :", arg)

# ------------------------------------------------------------
# **kargs for variable number of keyword arguments
def myFun2(**kwargs):
    print("\n\n")
    for key, value in kwargs.items():
        print ("%s == %s" %(key, value))

# ------------------------------------------------------------
# *args, **kargs for variable number of keyword arguments

def myFun(*args,**kwargs):
    print("\n\n")
    print("args: ", args)
    print("kwargs: ", kwargs)


# Driver code
myFun1('Hello', 'Welcome', 'to', 'BMU')
myFun2(first ='soharab', mid ='hossain', last='shaikh')
myFun('life','is','good',first="soharab",mid="hossain",last="shaikh")
```

25

```
 First argument : Hello
 Next argument through *argv : Welcome
 Next argument through *argv : to
 Next argument through *argv : BMU


first == soharab
mid == hossain
last == shaikh


args:  ('life', 'is', 'good')
kwargs:  {'first': 'soharab', 'mid': 'hossain', 'last': 'shaikh'}
```

26

Plan for Today

- **Object Oriented Programming** with Python
  > Class & Object
  > Encapsulation
  > Inheritance
  > Polymorphism
  > Operator Overloading – Magic Methods
- **Module**
- **Package**

27

OOP Concepts

35

Why OOP?

- Often we describe real-life objects in code (which is easier with OOP)
- Emphasis on data protection and access control
- Code get's more manageable,
- With OOP, it is easy to reuse previously written code,
- Every bigger real-life project will be written in OOP

36

Object oriented programming

- Object in Python is a representation of a person, a place, a bank account, a car, or any item that the program should handle.
- Object Oriented Programming Recipie:
  - (1) define **classes** (these are descriptions)
  - (2) **make object instances** out of classes.

Class Car

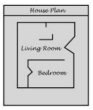
Car instances

37

Class

3 objects / instances / individuals

Blueprint that describes a house



Instances of the house described by the blueprint

38

OOP with a Taxi Example

- To learn OOP, we will use an example of a Taxi.

39

Example Object - Taxi

Every object has two main components:
Attributes (the data about it)
Behavior (the methods)

ATTRIBUTES	BEHAVIOR
• DriverName	• PickUpPassenger
• OnDuty	• DropOffPassenger
• NumPassenger	• SetDriverName
• Cities	• GetDriverName

40

Taxi
• DriverName: **string**
• OnDuty: **Boolean**
• NumPassenger: **int**
• Cities:**list**
• PickUpPassenger():**int**
• DropOffPassenger(): **int**
• SetDriverName(**string**)
• GetDriverName:**string**

41

Creating a simple class in Python

- In a class, we describe (1) how the object will look like, and (2) what behavior it will have.
- Classes are **the blueprints** for a new data type in your program!
- A class should have at minimum*:
  - A name (e.g. **Class Taxi**)
  - A constructor method (that describes how to create a new object, **__init__**)

42

CLASS NAME

```
class Taxi(object):
    '''This is a blueprint for a taxi! '''
    pass
```

43

Example Class

CLASS NAME

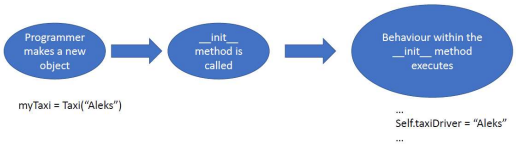
```
class Taxi:
    '''This class describes how a taxi may look like'''
    def __init__(self,driverName, onDuty, cities):
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0
```

Object Variables

44

The __init__ method

- __init__ is a special method in Python classes,
- The __init__ method is the constructor method for a class
- __init__ is called when ever an object of the class is constructed.



myTaxi = Taxi("Aleks")

...
Self.taxiDriver = "Aleks"
...

45

Example Object - Taxi

DATA

- DriverName
- OnDuty
- NumPassenger
- Cities

46

Creating an object from the class

- From one class, you make objects (instances).

```
class Taxi:
    '''This class describes how a taxi may look like'''
    def __init__(self,driverName, onDuty, cities):
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0


ourFirstTaxi = Taxi("Aleks", True, ['Lund', 'Malmo'])
print ourFirstTaxi.cities
```

47

Object Creation Flow

- To create a new object, use the class name

```
ourFirstTaxi = Taxi("Aleks", True, ['Lund', 'Malmo'])
```

- When you create a new object, the __init__ method from the class is called with the parameters that were passed.

48

Object Creation Flow

```
Taxi("Aleks", True, ['Lund', 'Malmo'])
```

- The __init__method Is called

```
class Taxi:
    '''This class describes how a taxi may look like'''
    def __init__(self,driverName, onDuty, cities):
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0


ourFirstTaxi = Taxi("Aleks", True, ['Lund', 'Malmo'])
print ourFirstTaxi.cities
```

49

Example Object - Taxi

DATA	BEHAVIOR
• DriverName	• PickUpPassenger
• OnDuty	• DropOffPassenger
• NumPassenger	• SetDriverName
• Cities	• GetDriverName

50

Behavior of a class

- classes/objects can have methods just like functions except that they have an extra **self** variable at the beginning.
- An object method takes as the first parameter the object (self) and can accept any number of other parameters.

51

Example Object Method

```
#We Describe the beaviour of the class with methods
def changeDriverName(self, newDriverName):
    ''' a simple method that updates the name of the driver'''
    self.dname = newDriverName
```

This method changes the name of the taxi driver for the passed object (self).

52

Another example of an object method

```
def pickUpPassengers(self, numOfPickedUpPassengers):
    ''' a method that increases the number of passengers'''
    self.numPassengers += numOfPickedUpPassengers
```

53

Exercise: Write a class that describes a **Bus**.

- A bus is created with a *number of* ***seats***, *a* ***color***, and is driven by a **bus driver** that *has a name*. New passengers can get in the bus, and existing bus passengers can leave their seat and get of the bus. Number of buss passengers can't be smaller than 0.

Instructions

- Start by drawing a class diagram
- Create a class for the bus in python
- Create two objects of your class

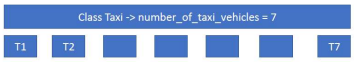
```
class Taxi:
    '''This class describes how a taxi may look like'''
    def __init__(self,driverName, onDuty, cities):
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0


ourFirstTaxi = Taxi("Aleks", True, ['Lund', 'Malmo'])
print ourFirstTaxi.cities
```

54

Object vs. Class Variables

- Most variables are **object specific** (for example, the variable number of passengers in a taxi is different for every taxi that we create).
- Some variables are **common to all objects** (for example, if we want to count the number of taxis that we have in our company)



55

Accessing class variables

- To access a class variable within a method, we use the @classmethod decorator, and pass the class to the method.

```
@classmethod
def how_many(cls):
    """returns the current TAXI inventory."""
    return cls.numberOfTaxis
```

56

Example use of class variable

```
class Taxi:
    '''This class describes how a taxi may look like'''

    numberOfTaxis = 0
    @classmethod
    def how_many(cls):
        """returns the current TAXI inventory."""
        return cls.numberOfTaxis

    def __init__(self,driverName, onDuty, cities):
        ''' The method that is called when a new object is created'''
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0
        Taxi.numberOfTaxis = Taxi.numberOfTaxis +1
```

```
ourFirstTaxi = Taxi("Aleks", True, ['Lund', 'Malmo'])
ourSecondTaxi = Taxi("Mladen", True, ['Lund', 'Malmo'])
ourFirstTaxi.changeDriverName("Jacob")
ourFirstTaxi.pickUpPassengers(4)
print ourFirstTaxi.dname,"is driving", ourFirstTaxi.numPassengers,
    "passenger(s). Currently, we have: ", Taxi.how_many(), " taxi(s)."
```

57

Inheritance

Code reusability

58

Example Class

CLASS NAME

```
class Taxi:
    '''This class describes how a taxi may look like'''
    def __init__(self,driverName, onDuty, cities):
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0
```

Object Variables

59

Comparing Bus & Taxi

```
class Taxi:
    '''This class describes how a taxi may look like'''
    def __init__(self,driverName, onDuty, cities):
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0
```

```
class Bus:
    '''This is my first class that describes a bus'''
    def __init__(self,busDriverName, colorParam, numberOfSeats):
        self.bdname = busDriverName
        self.color = colorParam
        self.nseats = numberOfSeats
```

60

Comparing Bus & Taxi

```
class Taxi:
    '''This class describes how a taxi may look like'''
    def __init__(self,driverName, onDuty, cities):
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0
```

```
class Bus:
    '''This is my first class that describes a bus'''
    def __init__(self,busDriverName, colorParam, numberOfSeats):
        self.bdname = busDriverName
        self.color = colorParam
        self.nseats = numberOfSeats
```

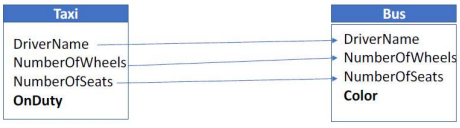
Classes share similar variables

61

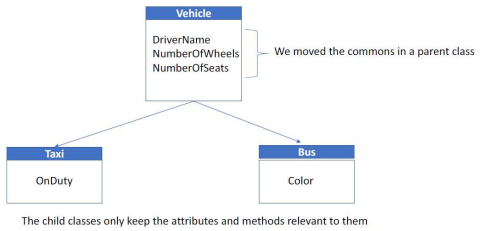
Inheritance

- Inheritance simplifies our code through reuse of the code that has been already written.
  - **Think about the Taxi and Bus, and what they have in common.**
- Inheritance is a relation between a **parent class** (e.g. *Vehicle*) and **children classes** (e.g. Taxi, Bus, Truck, etc.)
- A class inherits **attributes** and **behavior** methods from its parent classes.

62



63



64

OOP Inheritance in Python

1. Create a parent class (e.g. Vehicle) with the **common attributes** and **common methods**.

2. Create child classes (e.g. Bus and Taxi) with the **extended attributes** and **extended methods**.
   - Pass the class definition to the child (e.g. **Class Bus(Vehicle):** …)
   - Use the parent attributes and methods through **super().**

65

In Python code

```
class Vehicle():
    '''My class representing a vehicle'''
    def __init__(self, DriverName, NumberOfWheels, NumberOfSeats):
        '''This method initiates a new Vehicle (set's the parameters to object variables)'''
        self.dname = DriverName
        self.nwheels = NumberOfWheels
        self.nseats = NumberOfSeats
```

```
class Taxi(Vehicle):
    ''' This class inherits from Vehicle and adds OnDuty as a parameter'''
    def __init__(self,DriverName, NumberOfWheels, NumberOfSeats,OnDuty):
        #Vehicle.__init__(self, DriverName, NumberOfWheels, NumberOfSeats)
        super().__init__(DriverName, NumberOfWheels, NumberOfSeats)
        self.tduty=OnDuty
```

66

Method Overriding

- Method overriding is an object-oriented programming feature that allows a subclass to provide a different implementation of a method that is already defined by its superclass or by one of its superclasses.

- __init__ in the child class (e.g. Taxi) overrides the __init__ method from the parent class.

67

Example of overriding __str__

Let's add a __str__ method that nicely prints our Vehicle details on the screen.

```
class Vehicle:
    '''My class representing a vehicle'''
    def __init__(self, DriverName, NumberOfWheels, NumberOfSeats):
        '''This method initiates a new Vehicle (set's the parameters to object variables)'''
        self.dname = DriverName
        self.nwheels = NumberOfWheels
        self.nseats = NumberOfSeats
    def __str__(self):
        '''This method return's the vehicle details for printing on screen'''
        return "This vehicle is driven by: " + self.dname + " and it has " + str(self.nwheels) + " wheels."
```

```
# We create one instance of a vehicle and print it.
ourfirstvehicle = Vehicle("Aleksander", 4, 5)
print(ourfirstvehicle)
```

Output:

```
This vehicle is driven by: Aleksander and it has 4 wheels.
```

68

Example of overriding __str__

```
class Vehicle:
    '''My class representing a vehicle'''
    def __init__(self, DriverName, NumberOfWheels, NumberOfSeats):
        '''This method initiates a new Vehicle (set's the parameters to object variables)'''
        self.dname = DriverName
        self.nwheels = NumberOfWheels
        self.nseats = NumberOfSeats
    def __str__(self):
        '''This method return's the vehicle details for printing on screen'''
        return "This vehicle is driven by: " + self.dname + " and it has " + str(self.nwheels) + " wheels."
```

```
class Taxi(Vehicle):
    ''' This class inherits from Vehicle and adds OnDuty as a parameter'''
    def __init__(self,DriverName, NumberOfWheels, NumberOfSeats,OnDuty):
        super().__init__(DriverName, NumberOfWheels, NumberOfSeats)
        self.tdutysOnDuty
    def __str__(self):
        return super().__str__() + "Also, this taxi duty state is: " + str(self.tduty)
```

69

Example of overriding __str__

```
# We create one instance of a vehicle and print it.
ourfirstvehicle = Vehicle("Aleksander", 4, 58)
print(ourfirstvehicle)

# We create one instance of a taxi and print it.
ourfirstTaxi = Taxi("James", 4, 2, True)
print(ourfirstTaxi)
```

```
This vehicle is driven by: Aleksander and it has 4 wheels.
This vehicle is driven by: James and it has 4 wheels.Also, this taxi duty state is: True
```

70

Polymorphism

Ability of a method to show different behaviour depending on the type of the object being operated on

71

Polymorphism
Operator Polymorphism

Polymorphism simply means that we can call the **same method name** with parameters, and depending on the parameters, it will do different things. For example:

```
>>> print(6 * 5)
```
← 30

```
>>> print("Hello" * 5)
```
← HelloHelloHelloHelloHello

72

Polymorphism
Operator Polymorphism

For integer data types, + operator is used to perform arithmetic addition operation.

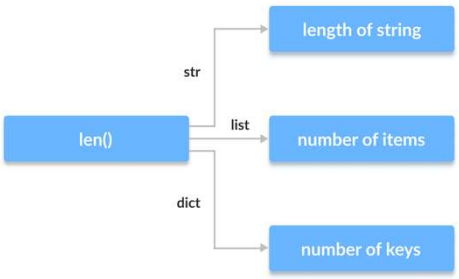
```
num1 = 1
num2 = 2
print(num1+num2)
```

Similarly, for string data types, + operator is used to perform concatenation.

```
str1 = "Python"
str2 = "Programming"
print(str1+" "+str2)
```

73

Polymorphism
Function Polymorphism



74

Polymorphism
Class Polymorphism

We can use the concept of polymorphism while creating the methods of a class as Python allows different classes to have methods with the same name.
We can then later generalize calling these methods by disregarding the object we are working with. Let's look at an example:

```
class Cat:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def info(self):
        print(f"I am a cat. My name is {self.name}. I am {self.age} years old.")

    def make_sound(self):
        print("Meow")


class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def info(self):
        print(f"I am a dog. My name is {self.name}. I am {self.age} years old.")

    def make_sound(self):
        print("Bark")
```

75

Polymorphism
Class Polymorphism

```
cat1 = Cat("Kitty", 2.5)
dog1 = Dog("Fluffy", 4)

for animal in (cat1, dog1):
    animal.make_sound()
    animal.info()
    animal.make_sound()
```

Output

```
Meow
I am a cat. My name is Kitty. I am 2.5 years old.
Meow
Bark
I am a dog. My name is Fluffy. I am 4 years old.
Bark
```

76

Polymorphism & Inheritance
(Method Overriding)

Child classes inherit methods and attributes from the parent class. We can redefine certain methods and attributes specifically to fit the child class, which is known as **Method Overriding**.

Polymorphism allows us to access these overridden methods and attributes that have the same name as the parent class.

```
from math import pi


class Shape:
    def __init__(self, name):
        self.name = name

    def area(self):
        pass

    def fact(self):
        return "I am a two-dimensional shape."

    def __str__(self):
        return self.name
```

77

Polymorphism & Inheritance
(Method Overriding)

```
class Square(Shape):
    def __init__(self, length):
        super().__init__("Square")
        self.length = length

    def area(self):
        return self.length**2

    def fact(self):
        return "Squares have each angle equal to 90 degrees."


class Circle(Shape):
    def __init__(self, radius):
        super().__init__("Circle")
        self.radius = radius

    def area(self):
        return pi*self.radius**2
```

```
a = Square(4)
b = Circle(7)
print(b)
print(b.fact())
print(a.fact())
print(b.area())
```

Output

```
Circle
I am a two-dimensional shape.
Squares have each angle equal to 90 degrees.
153.93804002589985
```

78

Magic/Dunder Methods

- Dunder or magic methods in Python are the methods having two prefix and suffix underscores in the method name. Dunder here means **“Double Under (Underscores)”**.
- These are commonly used for operator overloading.

79

Operator Overloading

(Magic/Dunder Methods)

```
# Python Program illustrate how to overload an binary + operator

class A:
    def __init__(self, arg):
        self.a = arg

print(dir(A))
```

```
['__add__', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__'
, '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne
__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__']
```

- When we use an operator on user defined data types then automatically a special function or magic function associated with that operator is invoked.
- Changing the behavior of operator is as simple as changing the behavior of method or function. You define methods in your class and operators work according to that behavior defined in methods.
- When we use + operator, the magic method __add__ is automatically invoked in which the operation for + operator is defined. There by changing this magic method's code, we can give extra meaning to the + operator.

80

Operator Overloading

```
# Python Program illustrate how to overload an binary + operator

class A:
    def __init__(self, arg):
        self.a = arg

    # Adding two objects (to overload the + operator)
    def __add__(self, obj):
        return self.a + obj.a

ob1 = A(1)
ob2 = A(2)
print(ob1 + ob2)

ob3 = A("BMU")
ob4 = A("Gurgaon")
print(ob3 + ob4)
```

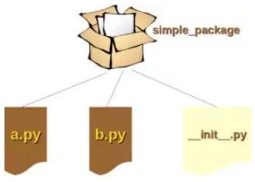
Output

```
3
BMUGurgaon
```

81

Package & Module

- A package is basically a directory with Python files and a file with the name __init__.py.
- This means that every directory inside of the Python path, which contains a file named __init__.py, will be treated as a package by Python.
- It's possible to put several modules into a Package.



Let's go to the coding demo…

82

End of Presentation

83