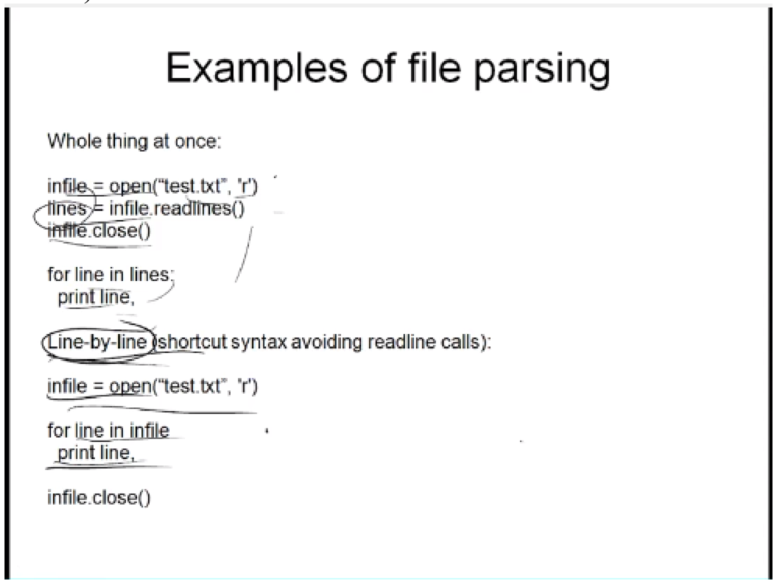
Hi everyone ,once again ,welcome to this class, we are continuing our class, on the programming class, today we will continue, our discussion on Python, we were looking at several things basically like a lot of basic constructs , a lot of data structures ,we looked at several things and in fact ,in the last lecture .
(Refer Slide Time: 00:32)



Examples of file parsing

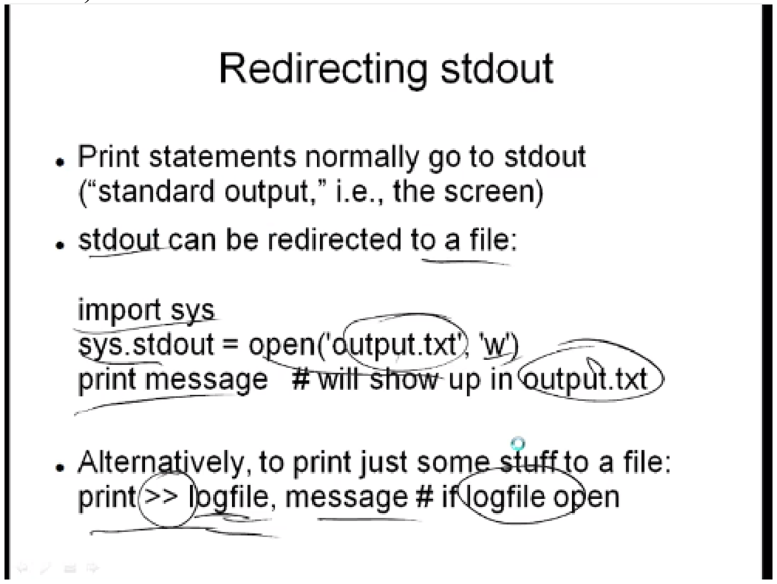
Whole thing at once:

```
infile = open("test.txt", 'r')
lines = infile.readlines()
infile.close()

for line in lines:
  print line,
```

Line-by-line (shortcut syntax avoiding readline calls):

```
infile = open("test.txt", 'r')

for line in infile
  print line,

infile.close()
```

We also did some of the file i/o , how to open a file ,how to close the file, how to write into a file things like that ,we saw all the details , regarding how to use them, we also did a lot of functions, the built-in functions essentially and then.
(Refer Slide Time: 00:53)



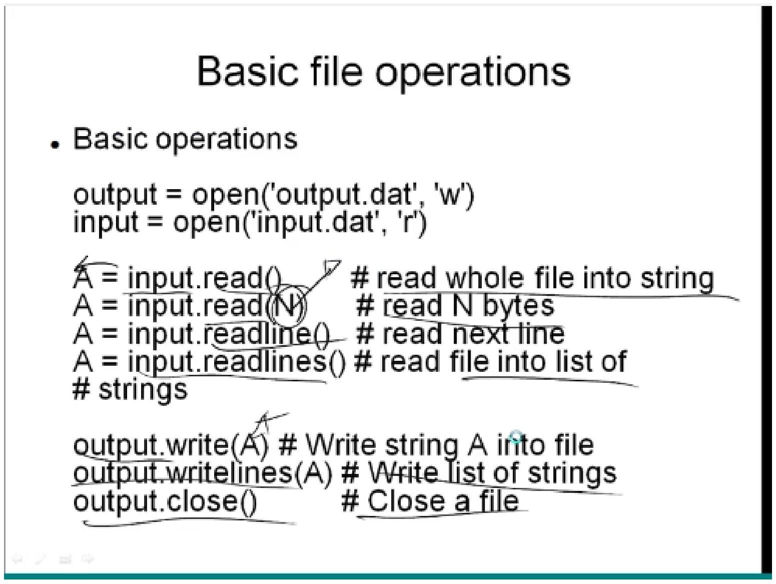
Redirecting stdout

- Print statements normally go to stdout ("standard output," i.e., the screen)
- stdout can be redirected to a file:

```
import sys
sys.stdout = open('output.txt', 'w')
print message   # will show up in output.txt
```

- Alternatively, to print just some stuff to a file:
  `print >> logfile, message # if logfile open`

(Refer Slide Time: 00:54)



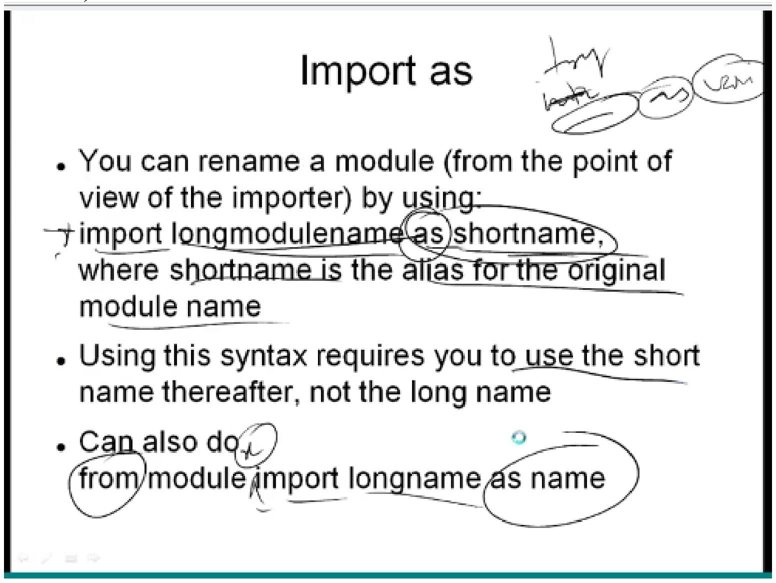
Basic file operations

- Basic operations

```
output = open('output.dat', 'w')
input = open('input.dat', 'r')

A = input.read()          # read whole file into string
A = input.read(N)         # read N bytes
A = input.readline()      # read next line
A = input.readlines() # read file into list of
# strings

output.write(A) # Write string A into file
output.writelines(A) # Write list of strings
output.close()        # Close a file
```

We also kind of understood ,some of the concepts with underscore, then the name underscore , that is again another built-in functions , that we can call into, also in the last class.

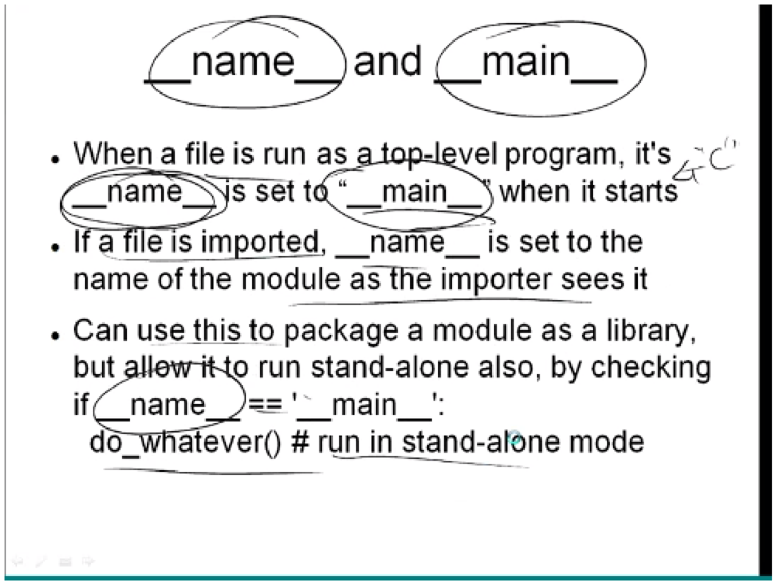
(Refer Slide Time: 01:07)



Import as

- You can rename a module (from the point of view of the importer) by using:
  import longmodulename as shortname, where shortname is the alias for the original module name
- Using this syntax requires you to use the short name thereafter, not the long name
- Can also do
  from module import longname as name

(Refer Slide Time: 01:09)

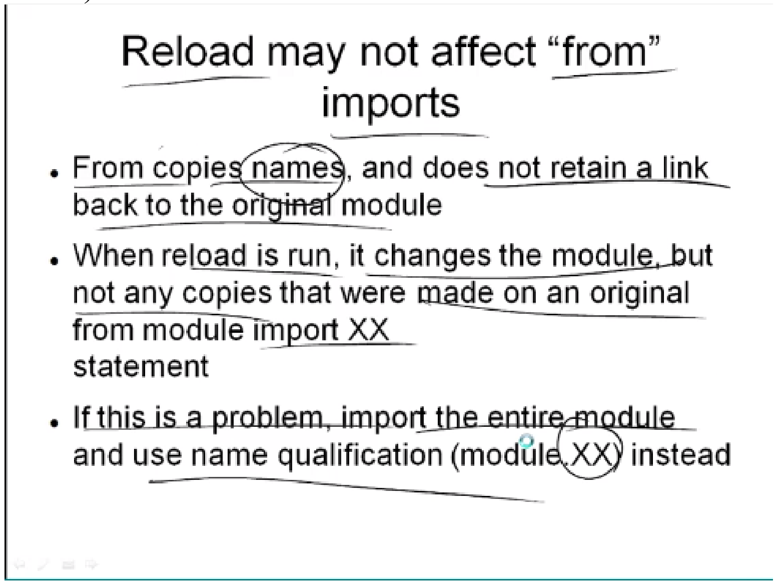

__name__ and __main__

- When a file is run as a top-level program, it's __name__ is set to "__main__" when it starts
- If a file is imported, __name__ is set to the name of the module as the importer sees it
- Can use this to package a module as a library, but allow it to run stand-alone also, by checking

```
if __name__ == '__main__':
  do_whatever() # run in stand-alone mode
```

We talked about lot of these importing, Python scripts from ,some other location ,or we import them, how do we will import a whole bunch of, functions and then how do .we actually do use some of them, things like that these are like several more ,often available in Python language to support that.

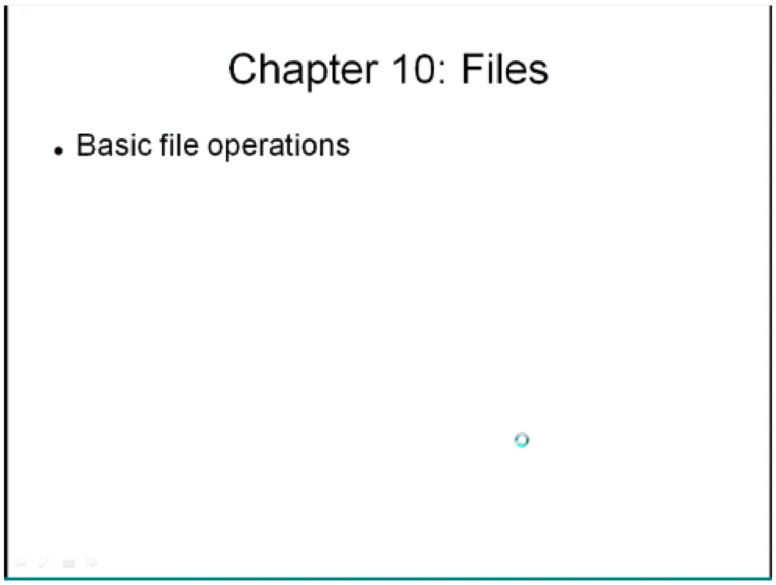
(Refer Slide Time: 01:31)


Reload may not affect "from" imports

- From copies names, and does not retain a link back to the original module
- When reload is run, it changes the module, but not any copies that were made on an original from module import XX statement
- If this is a problem, import the entire module and use name qualification (module.XX) instead

So today we will shift our gears, actually we will be covering couple of chapters.

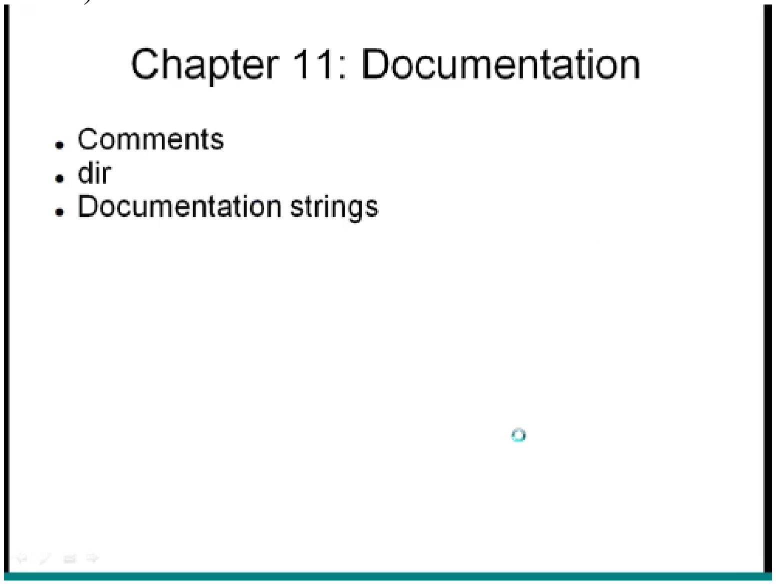
(Refer Slide Time: 01:43),



Chapter 10: Files

- Basic file operations

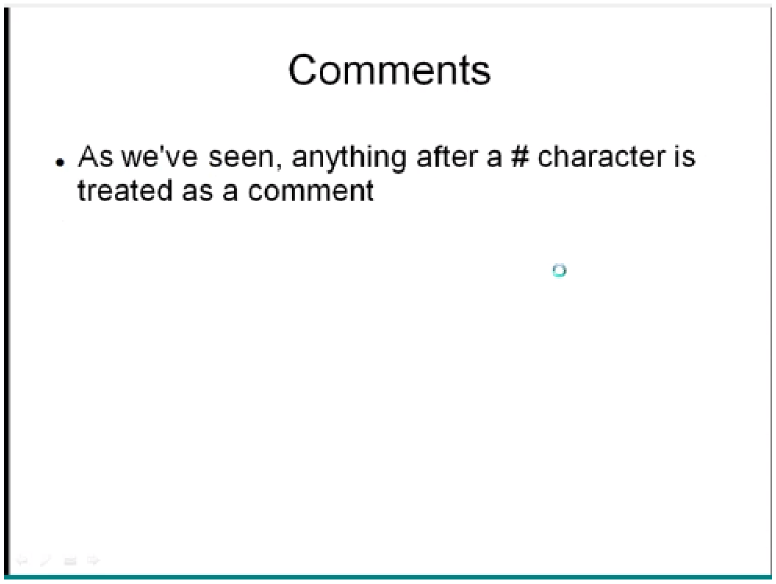
The first chapter is going to be, just on the documentation,.
(Refer Slide Time: 01:51)



Chapter 11: Documentation

- Comments
- dir
- Documentation strings

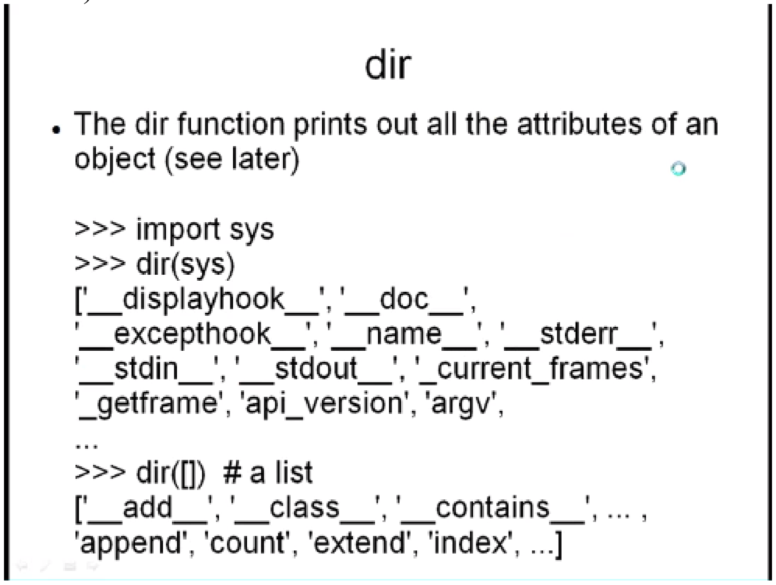
We will talk about the comments, dir, the directory and then documentation strings, what are the different things, that we can do here and then we will also like, look at some of the object-oriented functions, that are available in Python, this is mainly the class and the class libraries this is something, that we will go through okay, so without a lot of other, intro let me go to the comment string.
(Refer Slide Time: 02:27)


Comments

- As we've seen, anything after a # character is treated as a comment

So comments , as we actually saw this ,much earlier probably in the first class ,anything after a # or character is a comment, this can be basically on a line or at the beginning of the line wherever you put the # and then type in something else , that is treated as a comment, so nothing special about it, I mean so you can write comment ,say everywhere and urge you to write comments lot of comments, that is it becomes readable for the next person who supports these strings.

(Refer Slide Time: 03:10)


dir

- The dir function prints out all the attributes of an object (see later)

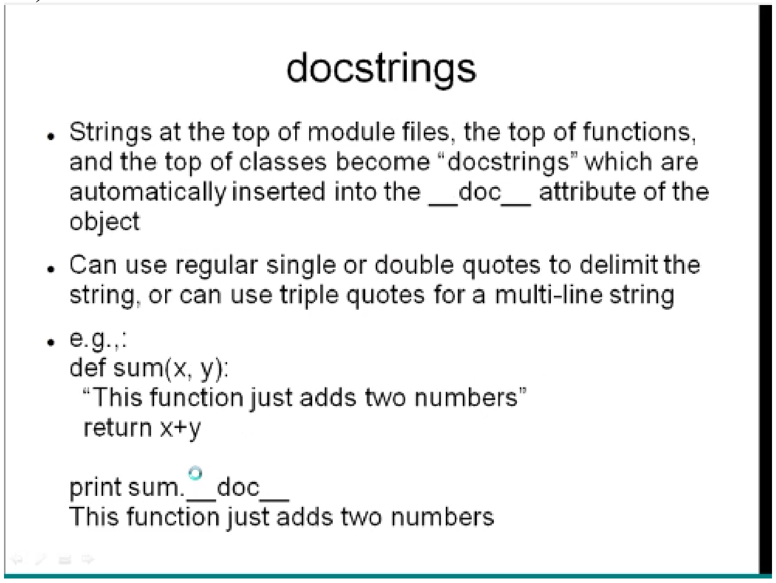
```
>>> import sys
>>> dir(sys)
['__displayhook__', '__doc__',
'__excepthook__', '__name__', '__stderr__',
'__stdin__', '__stdout__', '_current_frames',
'_getframe', 'api_version', 'argv',
...
>>> dir([])  # a list
['__add__', '__class__', '__contains__', ... ,
'append', 'count', 'extend', 'index', ...]
```

The dir function essentially prints out ,all the attributes of an object ,so this object and attributes actually like we will ,study later and we talk about the class and the class libraries, but just think about like, I mean if you have some objects ,we can just use dir to report all its attributes,. so for example here we are importing this sys , basically this is again the import functions that we studied in the last lecture.

And then when we say like a dir sys ,it prints out all the various attributes for this , for example _display hook _,_doc, _except hook ,_name ,_stderr, and all those different functions that are there , that are all like displayed in this one, form and then if you specify dir as [],instead of this and then basically there is a list , so here it displays the attributes for the list ,which is add ,class ,contains , append ,count, extend ,index etc.

So all the list attributes are displayed, when you say like they are list, the [] denotes that it is a list.

(Refer Slide Time: 04:51)



docstrings

- Strings at the top of module files, the top of functions, and the top of classes become "docstrings" which are automatically inserted into the __doc__ attribute of the object
- Can use regular single or double quotes to delimit the string, or can use triple quotes for a multi-line string
- e.g.,:

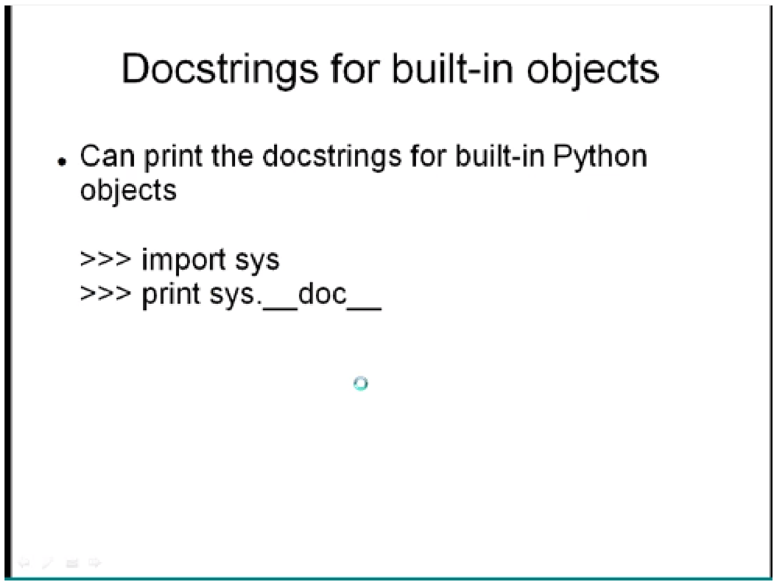
```
def sum(x, y):
  "This function just adds two numbers"
  return x+y

print sum.__doc__
This function just adds two numbers
```

Now comes the doc strings, the doc strings are nothing ,but documentation strings ,which are essentially used as documentation ,when we write our functions, essentially , so strings at the top of the module files, the top of functions and top of classes and all become doc strings which are automatically ,inserted into the _doc_ attribute of the object, so and then we can use regular, single or double quotes to, delimit the string or we can also use , triple quotes for multi-line string.

So these are all like I mean the standard ones, that we saw, we need to get to this okay ,so here as an example ,we have a function here ,which is basically sum x ,y, this we know like , how to define the function with a colon, usually we only put this , the statement ,where we just return x+y, but in this case ,we will just add this string here with the quotes ,this function just adds, two numbers and this becomes the doc string, for this particular function.

So when we say like print, sum _doc_ it prints the doc string ,which is essentially like its attribute ,so this is the way, that that we can specify the doc string.

(Refer Slide Time: 07:30)



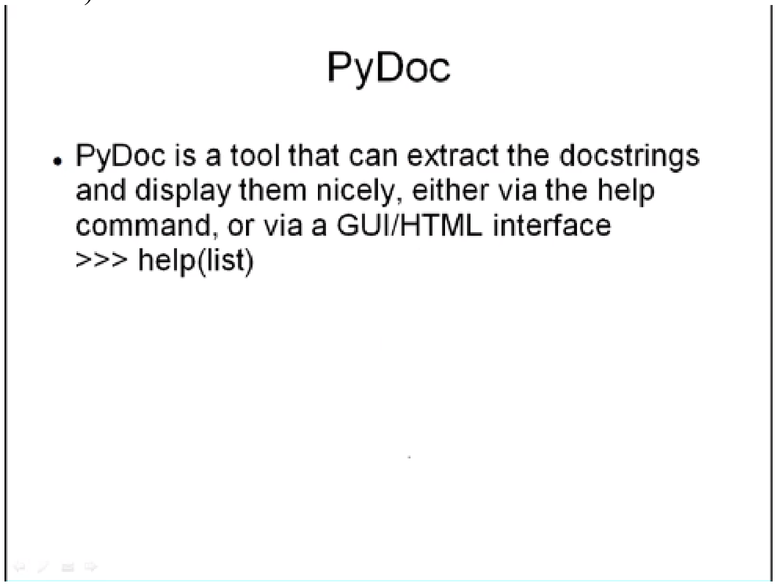
Docstrings for built-in objects

- Can print the docstrings for built-in Python objects

```
>>> import sys
>>> print sys.__doc__
```

Now there is also like, doc string available for built-in objects, for example, we can print these doc strings for built-in Python objects by just, using the relevant _doc_ ,for example if we import sys ,you can do print sys. _doc_ , so this is something that is useful ,so I think you can find out what this prints out .

(Refer Slide Time: 08:07)



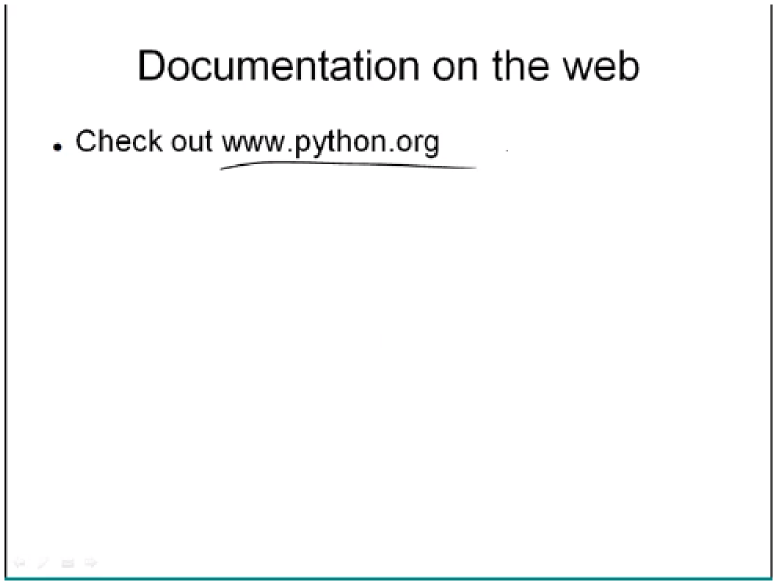
PyDoc

- PyDoc is a tool that can extract the docstrings and display them nicely, either via the help command, or via a GUI/HTML interface

```
>>> help(list)
```

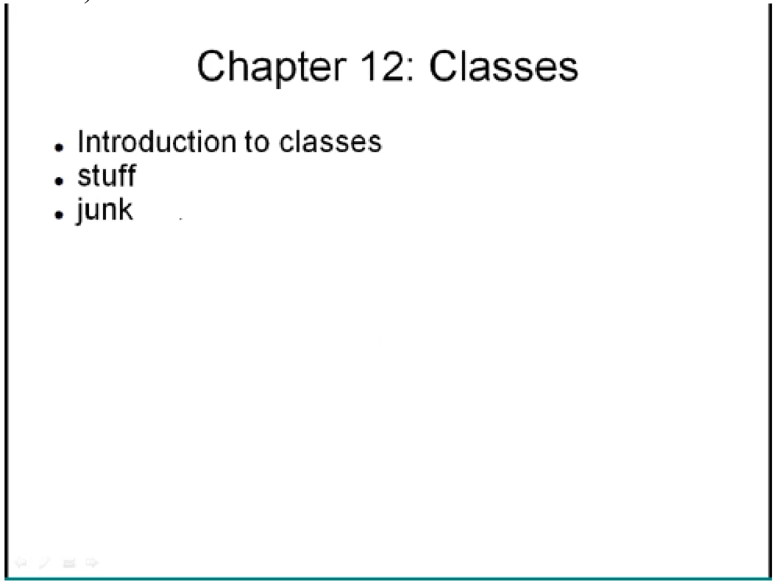
Now , there is also a tool called PyDoc, the pydoc is a tool ,that can extract the doc strings and display them nicely , either via the help command or via GUI or HTML interface, so you can do like help, list and then basically print out all those different doc strings .

(Refer Slide Time: 08:35)


Documentation on the web

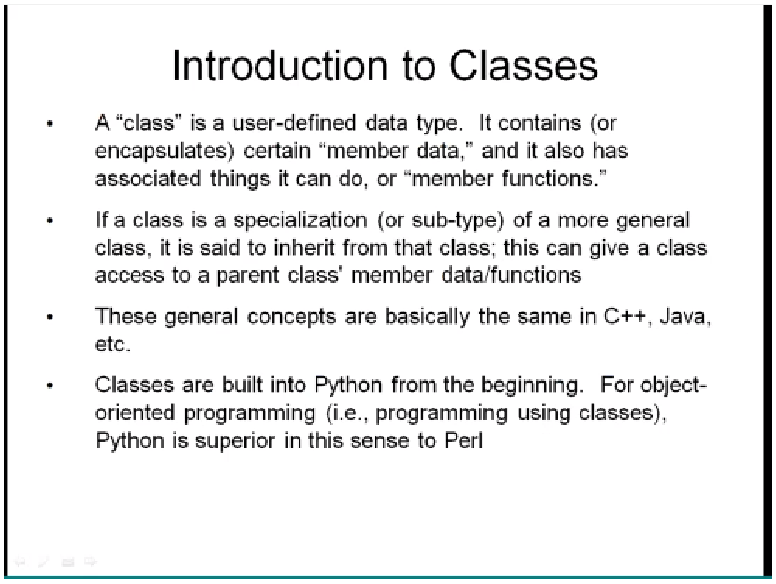
- Check out www.python.org

And then for any of the documentation , you can go to the web, and then search on www.python.org and can get a lot of information from this website.
(Refer Slide Time: 08:52)


Chapter 12: Classes

- Introduction to classes
- stuff
- junk

Now , we will go into the next topic, which is essentially the classes and this is one of the big topics that, we will be talking about today, mainly like I mean ,we will we will first do an introduction to classes, what are classes and we will talk about ,how do we again do ,what does it mean to actually have a class and this is again like I mean, as I mentioned , this is object-oriented functions, so we will talk about that and then we will also once , so we introduce to the classes and then we will talk about stuff and junk okay.
(Refer Slide Time: 09:48)



Introduction to Classes

- A “class” is a user-defined data type. It contains (or encapsulates) certain “member data,” and it also has associated things it can do, or “member functions.”
- If a class is a specialization (or sub-type) of a more general class, it is said to inherit from that class; this can give a class access to a parent class' member data/functions
- These general concepts are basically the same in C++, Java, etc.
- Classes are built into Python from the beginning. For object-oriented programming (i.e., programming using classes), Python is superior in this sense to Perl

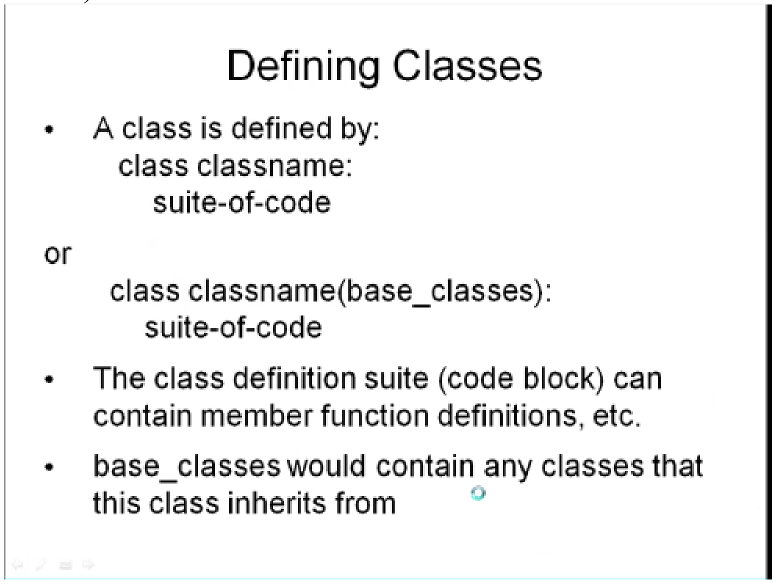
So the class is essentially ,it is a user-defined data type, it contains certain member data , which we will define ,what it is ,it also has associated things, that it can do ,called member functions, so anytime that class is defined ,should have a member data and functions , if a class is a specialization or a subtype of a more general class ,it is set to inherit from that class and this can give a class access to the parents class member both data and functions.

So these are all the concepts , essentially like many of the object oriented functions, first one is class, so class as we saw, it is basically, it is a user-defined data type ,the class will have some member functions and member data type, now a class can also ,have what are called sub classes, which are nothing but specialized classes and these subclasses, once you have, they can inherit this member function and data plus ,its own specialized functions and specialized data. So this is the way, that you can actually look into, what are classes and how the various term functions can be defined ,so the member function and the member data ,that the subclass gets from its parent class ,that is called the inheritance or this is basically inherited class, it is also called inherited class and these pretty much ,these terminologies are all pretty much same as C++, Java etc, like general concepts in other languages as well .

The classes are built into Python, from the very beginning ,we actually like use this term class, many times before I hope you remember ,those things , even though we did not never talked about, the object oriented programming itself and the object oriented programming basically ,that is the programming with , using all these classes and the members or member functions on the data, the python is superior to Perl in the sense okay.

So this is something ,that you may want to keep in mind, we also saw object oriented, which has all these things ,but Perl was developed as a non object oriented from beginning, so it may not have all these things , whereas Python has all the things, that are already there.

(Refer Slide Time: 13:12)



Defining Classes

- A class is defined by:

```
class classname:
    suite-of-code
```

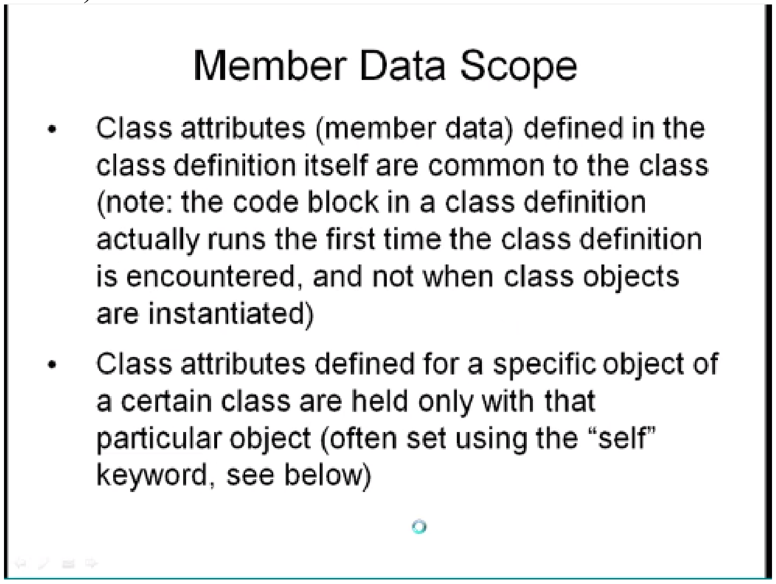
or

```
class classname(base_classes):
    suite-of-code
```

- The class definition suite (code block) can contain member function definitions, etc.
- base_classes would contain any classes that this class inherits from

Now, how do we define a class, the class is defined by two things ,one is the class name and the suite of code ,essentially or you can also define it as class room class name , base classes and then suite of code , the class definition suite or the code block can contain, member function definitions ,so you can define all the member function , what if this means , basically like whenever, you define a data, taken only they are restricted to member functions and the member data. The base classes go, would contain any class that ,this class inherits from okay.

(Refer Slide Time: 14:05)



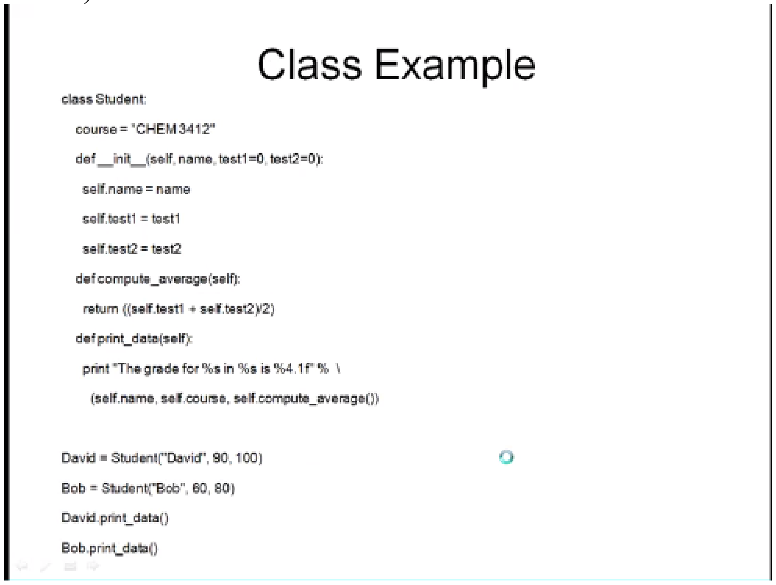
Member Data Scope

- Class attributes (member data) defined in the class definition itself are common to the class (note: the code block in a class definition actually runs the first time the class definition is encountered, and not when class objects are instantiated)
- Class attributes defined for a specific object of a certain class are held only with that particular object (often set using the “self” keyword, see below)

Now ,what is the member data scope or scope of the member data ,the class attributes, that is the member data, defined in the class definition itself are common to the class ,so the code block in the class definition actually, runs the first time , that the class definition is encounter and not when the class objects ,are this instantiated ,so this is another thing that you may want keep in mind.

So when the class objects are instantiated, the code block does not get drawn, it runs only when the first time, the class definition is encountered, the class attributes defined for a specific object ,object of a certain class are held only with that particular object ,often set using the self keyword, so we will see an example later, but again, so this is basically once you define a specific object, it has only held only ,that particular object .So when you go into another object , this property will already be gone.

(Refer Slide Time: 15:25)



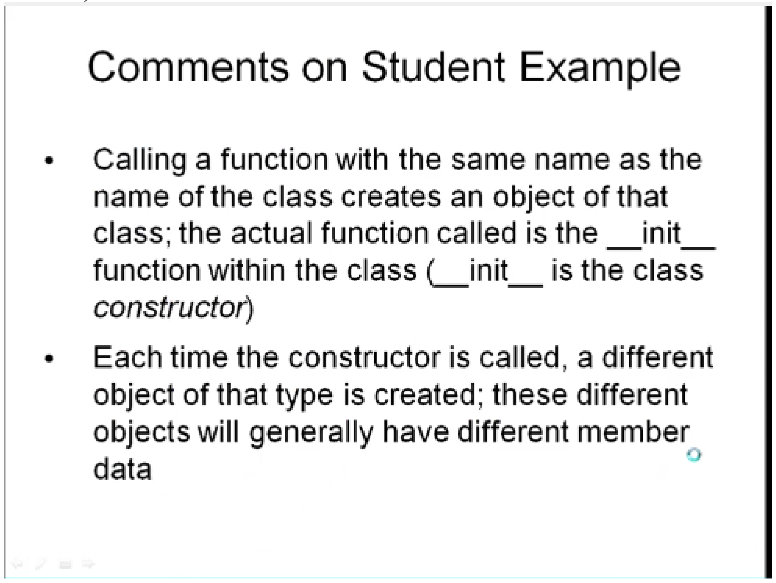
Class Example

```
class Student:
  course = "CHEM 3412"
  def __init__(self, name, test1=0, test2=0):
   self.name = name
   self.test1 = test1
   self.test2 = test2
  def compute_average(self):
   return ((self.test1 + self.test2)/2)
  def print_data(self):
   print "The grade for %s in %s is %4.1f" % \
    (self.name, self.course, self.compute_average())

David = Student("David", 90, 100)
Bob = Student("Bob", 60, 80)
David.print_data()
Bob.print_data()
```

So here is an example , here we define a class ,called student , the student has various attributes one is the course ,which is we define as CHEM 3412 and then, we define self functions ,so first one is the _init_, so here we give self.name, test1 = 0 and test2, so self name is the name, that we assign ,self-test 1 is called test 1 ,from arbitrary function or self-test 2 is also another test 2, now we can say like okay .

What is the computer average of self and then basically that will return, basically it ,will add like test 1 + test 2 / by 2 and then gives you ,that average and then printed data is essentially like I mean , so even as to print, the grade for this particular course, in this or wait for a particular name in this particular course, is sum , the complete average of the test1 and test2 .

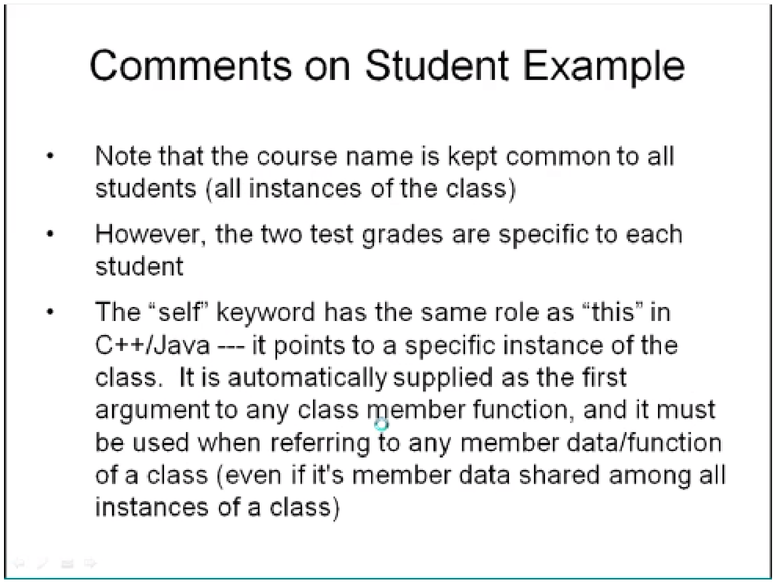
But if you have like multiple students, so you can say like , David is a student ,with attribute David as his name ,90 ,100 as Bob is a student like this ,is the base class , basically as a student + Bob the name and it's test1 and test2 scores, so now ,we can say like David ,print data and then it will print, the grade for David in CHEM 3412 , is 95.0 %, so it takes this computed average and post it and for same thing for Bob in CHEM 3412 , is 70.0 %.So the David. Print _data and Bob .print _data, both of them generate these kinds of results.
(Refer Slide Time: 18:22)



Comments on Student Example

- Calling a function with the same name as the name of the class creates an object of that class; the actual function called is the __init__ function within the class (__init__ is the class *constructor*)
- Each time the constructor is called, a different object of that type is created; these different objects will generally have different member data

So ,some more comments on the student example, the calling a function with the same name, as the name of the class ,creates an object of that class, the actual function is called is the _init _function within that particular class, _init _is also known as the class constructor ,so whenever we talk about the object-oriented programming , there are two things that you need to understand, one is what is called a constructor and then the other one is called the destructor.
So the destructor, will come the later on stage and each time a constructor is called a different object of that type is created , these different objects will ,generally have different member data.
(Refer Slide Time: 19:22)



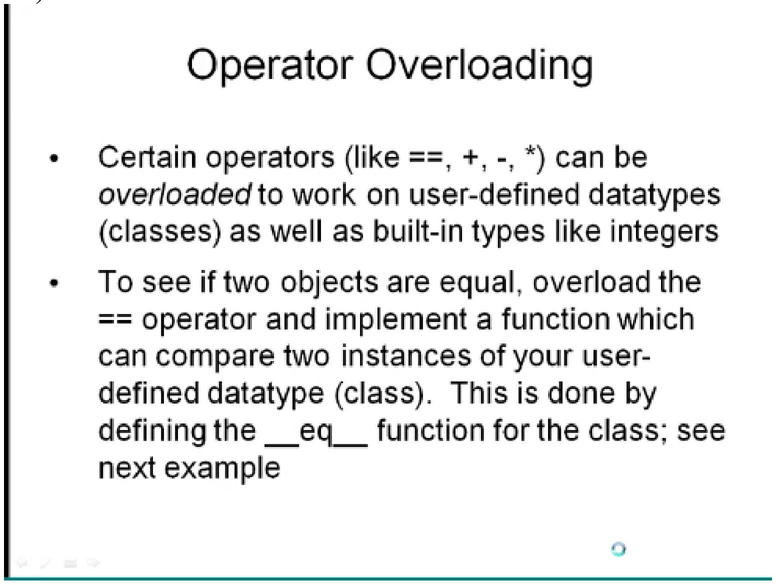
Comments on Student Example

- Note that the course name is kept common to all students (all instances of the class)
- However, the two test grades are specific to each student
- The "self" keyword has the same role as "this" in C++/Java --- it points to a specific instance of the class. It is automatically supplied as the first argument to any class member function, and it must be used when referring to any member data/function of a class (even if it's member data shared among all instances of a class)

So one thing if you had noticed in the previous example ,basically the class name was held constant ,it was the HEM 3412 to the class ,but over the 2 test grades are specific to each student, so the self keyword has the same role as this in C++ or Java, it points to a specific instance of the class, it automatically supplied as the first argument to any class member, so it is automatically supplied as a first class ,member first argument to any class member function.

And it must be used, than referring to any member data or function of particular class , even if the members data ,shared among all the instances of the class okay.

(Refer Slide Time: 20:33)

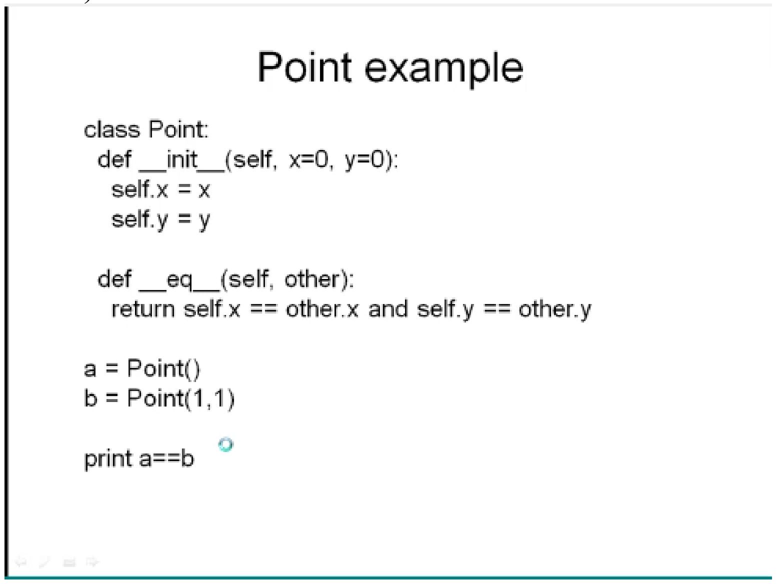


Operator Overloading

- Certain operators (like ==, +, -, *) can be *overloaded* to work on user-defined datatypes (classes) as well as built-in types like integers
- To see if two objects are equal, overload the == operator and implement a function which can compare two instances of your user-defined datatype (class). This is done by defining the __eq__ function for the class; see next example

So now ,we come to what, is known as the operator overloading, this is again another important concept essentially , where you can use the pre-existing functions and then all load them, so that suit your own customized , so for example some certain operators like = ,==, +,-,* can be

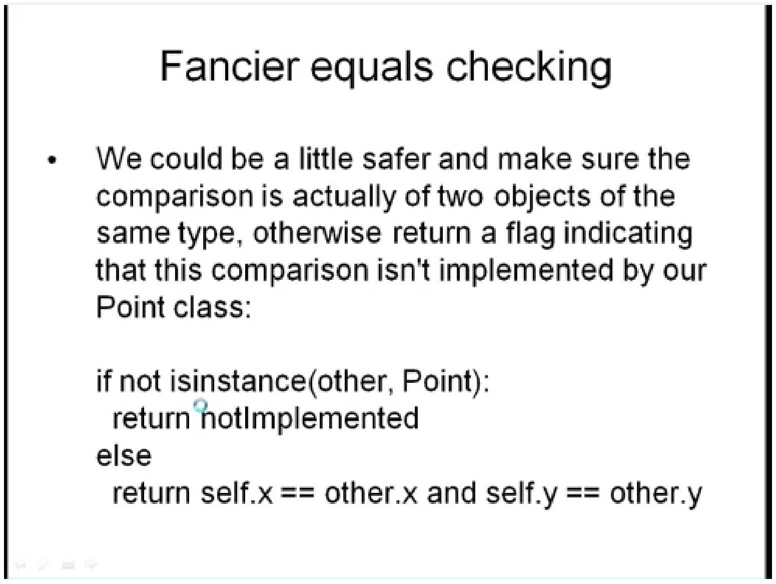
overloaded to work on a user-defined event, as well as a built-in data types like integers, so today is to see, if two objects are the same ,if they are integers, we typically use the == function on that operator .
But if you are a ,user defined data type and then if you want to compare those ,two user defined data types, that they match we can overload the = operator and implement a function which can compare , the two instances of the user-defined data type ,so this is done by actually defining _eq_ function or the class and let us see the example.
(Refer Slide Time: 22:03)



Point example

```
class Point:
  def __init__(self, x=0, y=0):
    self.x = x
    self.y = y

  def __eq__(self, other):
    return self.x == other.x and self.y == other.y

a = Point()
b = Point(1,1)

print a==b
```

So here we, define a class called point and then the point has its _ init _basically the self and x=0 and y=0 and then basically it return self x and self y ,now we can also define the =to testing and basically in this ,_self_ and other and then so we just basically, say that return self .x =other .x and self .y=other. y , so it basically does this testing and so here there is a array ,so a = points nothing and then b = point 1 ,1 and then now it will print a ==b.
(Refer Slide Time: 23:11)



Fancier equals checking

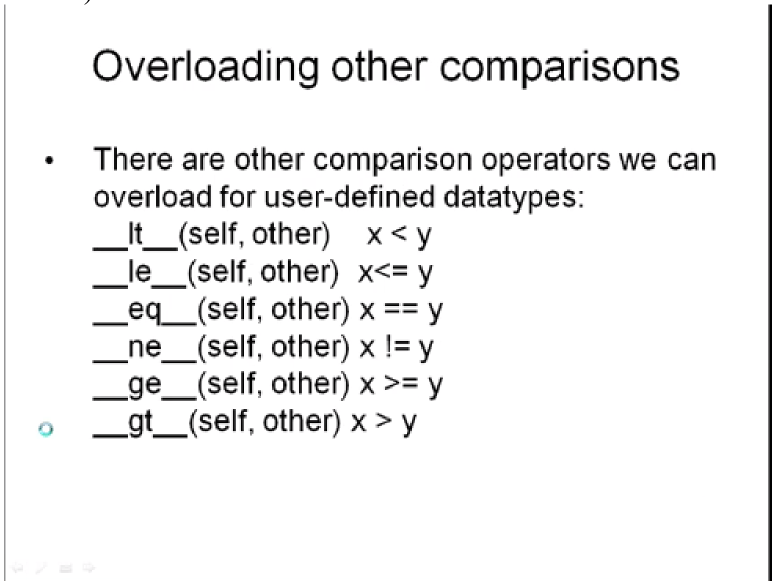
- We could be a little safer and make sure the comparison is actually of two objects of the same type, otherwise return a flag indicating that this comparison isn't implemented by our Point class:

```
if not isinstance(other, Point):
  return notImplemented
else
  return self.x == other.x and self.y == other.y
```

So one thing ,that we have to do whenever ,we do the comparison ,we need to actually make sure ,that they are the two objects, that are being compared on the same type , so we could be a little safer and make sure that the comparison is actually between the two objects ,of the same type and otherwise return a flag indicating ,that the comparison isn't implemented by our point class, so if that is the case then how do we do it .

So here, we just say basically if not instance other point ,return not implemented ,else return self .x== other .x and self. y== other. y , so we can take the same functions and then basically an item closed, within this if else to basically like test, for whether the comparison is between the two objects of the same type or not .

(Refer Slide Time: 24:53)



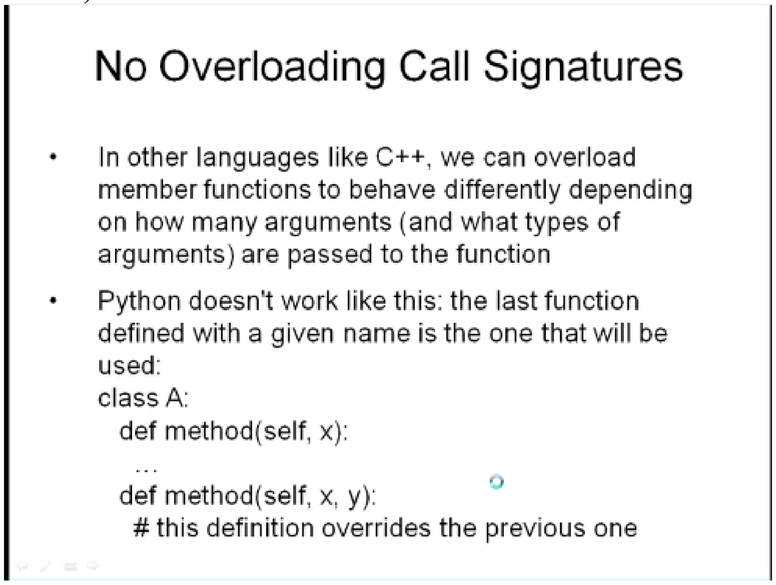
Overloading other comparisons

- There are other comparison operators we can overload for user-defined datatypes:

```
__lt__(self, other)    x < y
__le__(self, other)  x<= y
__eq__(self, other) x == y
__ne__(self, other) x != y
__ge__(self, other) x >= y
__gt__(self, other) x > y
```

Now we do not have to stop, at overloading the = to sign, we can also overload other operators, comparison operators like <, the _, x<y or _le _which is essentially stand for < or = and then same thing for eq which is x== y and then any is the !=or non eq ,which is essentially != and then there is a ge and gt ,ge stands for > or = form, and then gt stands for > than ,so all the logical operators, so basically the testing operators, we can overload and then use it .

(Refer Slide Time: 25:52)



No Overloading Call Signatures

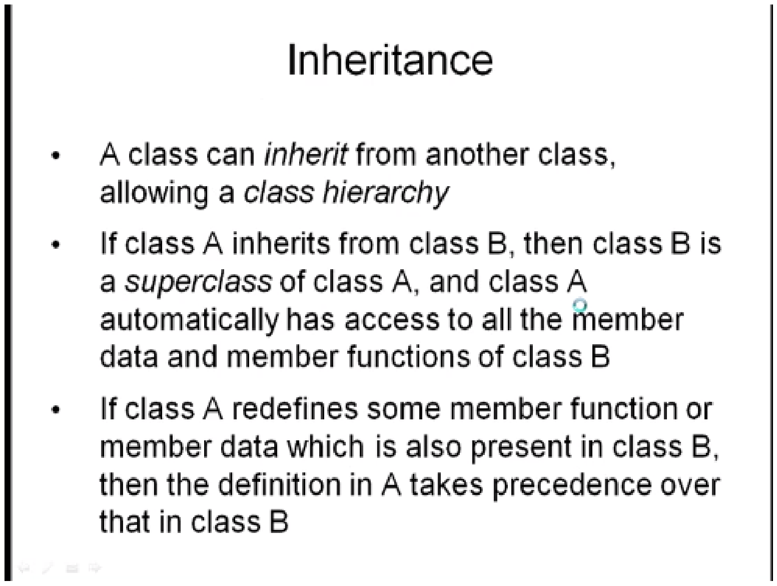
- In other languages like C++, we can overload member functions to behave differently depending on how many arguments (and what types of arguments) are passed to the function
- Python doesn't work like this: the last function defined with a given name is the one that will be used:

```
class A:
  def method(self, x):
    ...
  def method(self, x, y):
    # this definition overrides the previous one
```

So one thing ,that you may want to notice , there is no overloading the call signature ,stuff in other languages like C++ , we can overload the member functions ,to behave differently depending on how, many arguments are passed to the function, but Python does not work like that the last function ,defined with a given name is the one that will be used, so the function itself we cannot overload in Python.

And then basically , just uses the last defined function, so it does not use the ones , so a small example again here ,we have class A and then we define method and then we will say like self x, but then we want to change it ,we have to write a new function, which is now here def method is self x and y ,so this def overwrite the previous one and that is how you can get it.

(Refer Slide Time: 27:14)



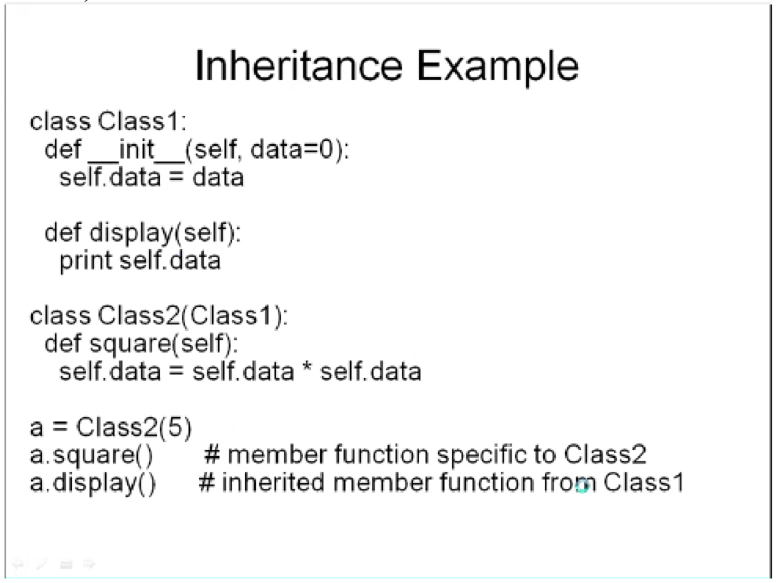
Inheritance

- A class can *inherit* from another class, allowing a *class hierarchy*
- If class A inherits from class B, then class B is a *superclass* of class A, and class A automatically has access to all the member data and member functions of class B
- If class A redefines some member function or member data which is also present in class B, then the definition in A takes precedence over that in class B

So now, we come to the inheritance essentially, we talked about this, in a couple of slides ago, so essentially a class can inherit from another class, allowing at class hierarchy, if a class-A inherits from class B, then class B is the super class of class A and says A automatically, so the class a automatically has access, to all the member data and the member function for classes.

So this is another thing, that we briefly talked about, so I just wanted to explain this somehow, in the class A inherit from the Class B, the class B is the super class of class A ,and then class A gets all the goodies from the class B, basically all the functions are automatically accessible by class A and if the class A redefine some member function or member data which is also present in class B,then the definition in A, takes precedence over the class B .

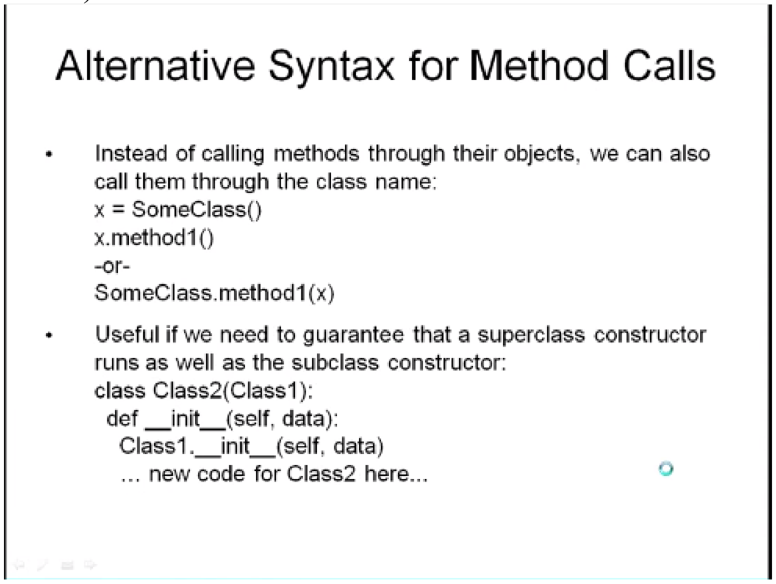
(Refer Slide Time: 28:49)



Inheritance Example

```
class Class1:
  def __init__(self, data=0):
    self.data = data

  def display(self):
    print self.data

class Class2(Class1):
  def square(self):
    self.data = self.data * self.data

a = Class2(5)
a.square()      # member function specific to Class2
a.display()     # inherited member function from Class1
```

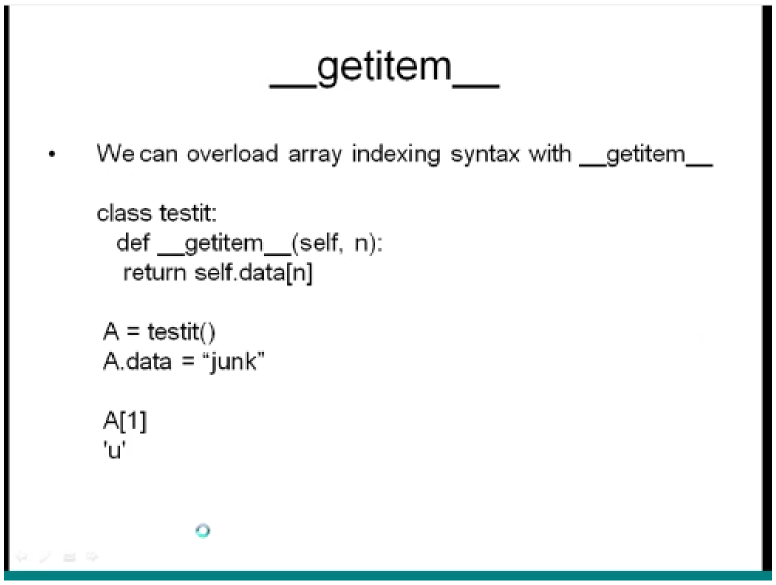
So now let us look at an inheritance example , how to do the inheritance , so here we define the class as class 1 and then _ init we define as self data=0 and those are the starting conditions and then self. Data is data, now we just print out the data and then we define a class 2 which is an inherited class of class 1, so see how we link these 2 and then when we define a square self.
So we can basically say like self. Data = self. Data* self. data ,which is essentially like now, start the multiplication ,so one thing to note here is essentially ,since we defined the class 2 as class 1 it gets all these things essentially even the print or the display command is also there, so we can specify like A as class 2 with 5 and then we can also specify a. square as well as ignore display , so a .square is defined here and then a. display .So the first one is just a member function ,with +2 and the second function is an inherited from class1.
(Refer Slide Time: 31:09)



Alternative Syntax for Method Calls

- Instead of calling methods through their objects, we can also call them through the class name:
```
x = SomeClass()
x.method1()
-or-
SomeClass.method1(x)
```
- Useful if we need to guarantee that a superclass constructor runs as well as the subclass constructor:
```
class Class2(Class1):
  def __init__(self, data):
    Class1.__init__(self, data)
    ... new code for Class2 here...
```

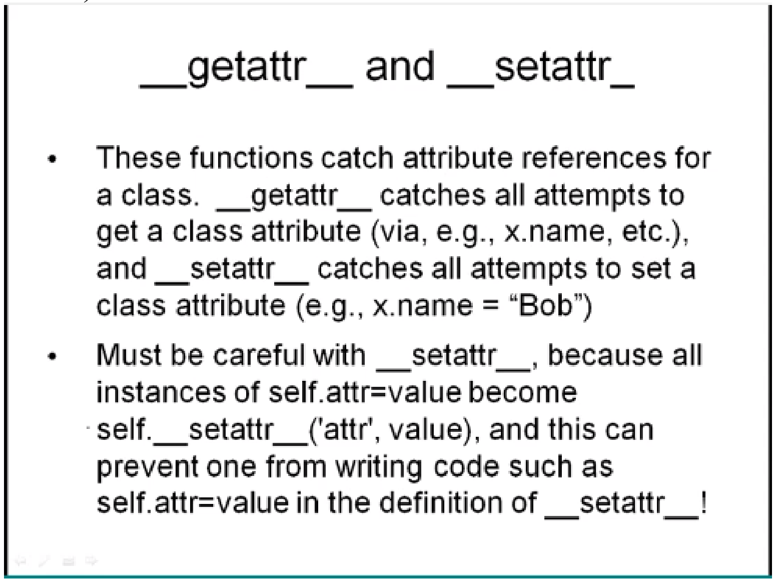
So we saw like some of the methods, so let us now explore, some of the alternative syntax for the method calls , so instead of calling the methods through their objects., you can also call them through their class name, so here is an example ,so we define x as this class variable some x, some class, now we can call the method 1 as just x . method 1 or some class . method 1 these both syntax are perfectly legal syntax.
Now this alternate method ,may be useful to guarantee ,that the super class constructor, runs as well as the subclass constructor , so here is another example ,we define class 2 as inherited class from +1 and then here we can define ,like _init _ for self data and then we can also like class 1 . _init _ which directly refers to the 1 before the thing and then you can say like insert the class 2 code here.
(Refer Slide Time: 33:14)


__getitem__

- We can overload array indexing syntax with __getitem__

```
class testit:
  def __getitem__(self, n):
   return self.data[n]

A = testit()
A.data = “junk”

A[1]
'u'
```

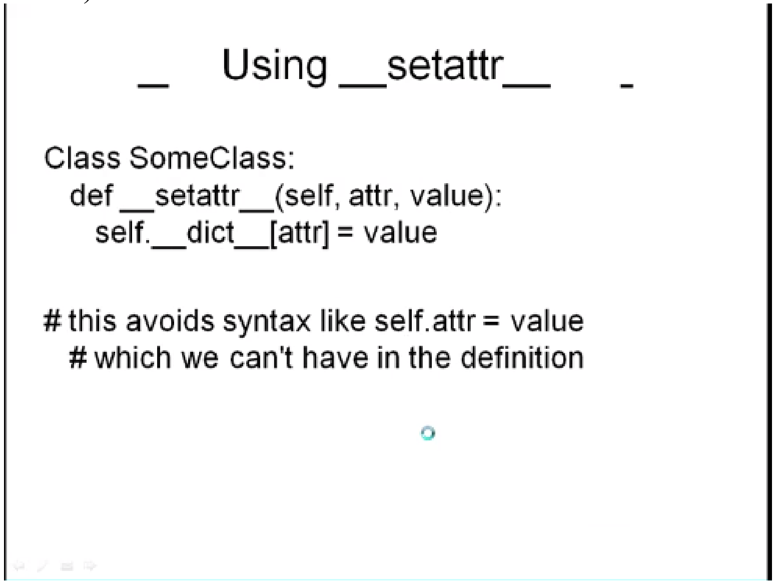
So we can overload an array indexing ,syntax with _get item, so how do we ,do that ,we define a class for test it and then we say like define the _get item is self n and then basically the return self. Data and then the n variable , and now, when we wanted to use this ,test it we can just say like A= test it and A. data = junk, so now we get this violation here.
(Refer Slide Time: 34:27)


__getattr__ and __setattr_

- These functions catch attribute references for a class. __getattr__ catches all attempts to get a class attribute (via, e.g., x.name, etc.), and __setattr__ catches all attempts to set a class attribute (e.g., x.name = “Bob”)
- Must be careful with __setattr__, because all instances of self.attr=value become self.__setattr__('attr', value), and this can prevent one from writing code such as self.attr=value in the definition of __setattr__!

Now there are more, to the system defined functions ,one such is get attribute and set attribute so the get attribute, these functions catch the attribute references for class ,so the get attribute catches all attempts, to get the class attribute, so this is the example x. name etc and then the set attribute can just all the ,attempts to set a class attribute for example x. name = Bob , so we need to be careful ,with the set attribute, because all this stuff for the self attribute value become self _attribute .

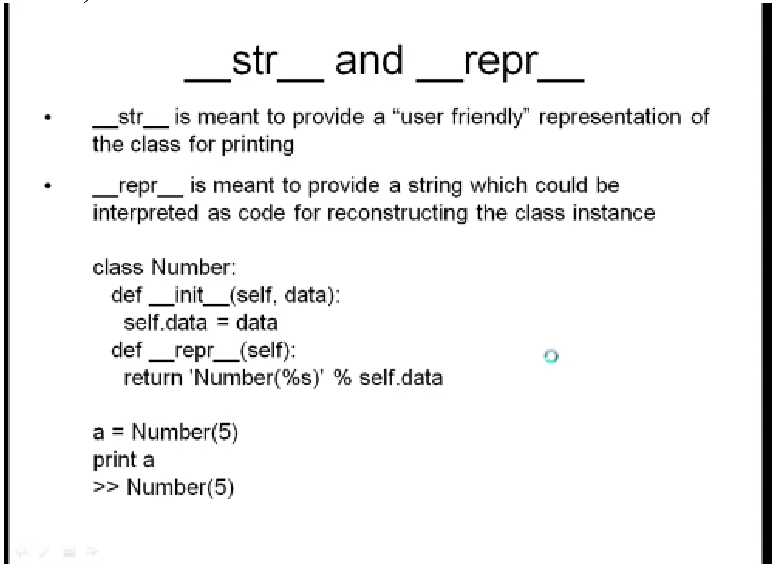
And then the value will be the actual value and this can prevent one from writing code such as self .attribute = value in the definition of set attribute.
(Refer Slide Time: 35:46)



Using __setattr__

```
Class SomeClass:
  def __setattr__(self, attr, value):
    self.__dict__[attr] = value

# this avoids syntax like self.attr = value
  # which we can't have in the definition
```

So now, how do we use the set attribute, so using the _set or set attribute , so here you another example they say like class, some class and then define _ set attribute, for example so this is basically we capture the initial value and then say like self. _dict_ attribute = value then it will print and this avoids the syntax like self. attr = value ,which we cannot have this definition.
(Refer Slide Time: 36:41)



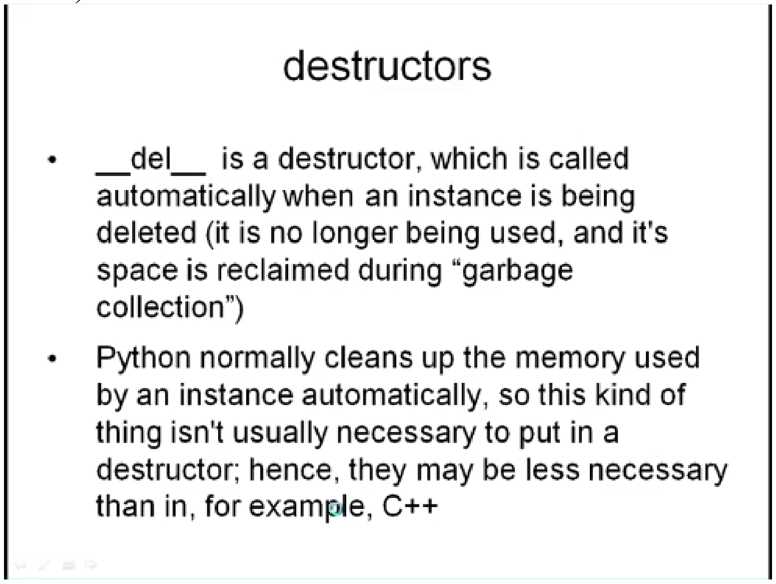
__str__ and __repr__

- __str__ is meant to provide a "user friendly" representation of the class for printing
- __repr__ is meant to provide a string which could be interpreted as code for reconstructing the class instance

```
class Number:
  def __init__(self, data):
    self.data = data
  def __repr__(self):
    return 'Number(%s)' % self.data

a = Number(5)
print a
>> Number(5)
```

Now there is another member function _str_ which is essentially, it is meant to prevent a user-friendly representation of a class ,for printing and then the other one is the _ repr _that is meant to provide a string, which could be interpreted as a code for reconstructing the class instances, so

now here is one example ,so here the class called number ,we define the function basically and then self .data =data and then we say repr self and then the return the number with self data .
So now, then you see this repr, essentially that is constructed as , string is in the code , so here when we say this that is interpreted as a code, so we say a = number 5 and then we say print a and then finally like the number is 5.r
(Refer Slide Time:39:01)

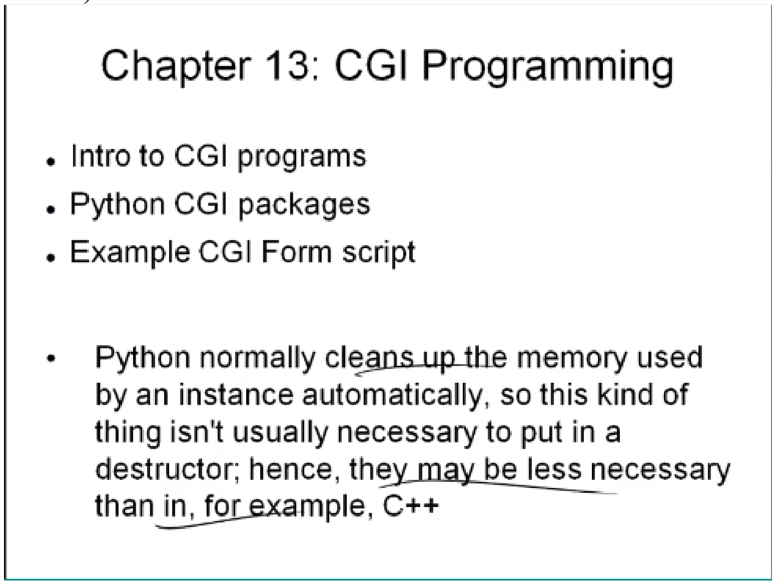


destructors

- __del__ is a destructor, which is called automatically when an instance is being deleted (it is no longer being used, and it's space is reclaimed during "garbage collection")
- Python normally cleans up the memory used by an instance automatically, so this kind of thing isn't usually necessary to put in a destructor; hence, they may be less necessary than in, for example, C++

Now I mentioned in the very, beginning of this lecture ,basically structures essentially ,let along with that, with the constructor, but now let us look at ,what is a destructor, the _del _ is also known as the destructor, it is called automatically, whenever an extent an instance is being deleted, so whenever the instance get blown off , transistor limitation, so whenever you want to delete something basically the destructor is called .
And essentially in Python, it automatically does this garbage collection, once that instance is being limited, so the garbage collection is basically where the space is reclaimed, so I think like I mean you probably have heard about ,there are different types of memories, the most common one or basically they are all the variables are kept, is called the heap okay, so essentially like you assign a variable, or assign a value to a variable essentially like , some portion of each taken and then the particular used is written there .
So when the heap gets filled up ,you cannot write ,anything more into the disk , so then you remove something basically, so every time like you allocate or you call the constructor which we saw earlier a portion of the memory, is being allocated to this constructor and then that is when like the memory come out ,everything is kept as a heap and it won't touch any other program.
So with other layers that we saw earlier, very beginning also like, the programs are kept at one level ,the operating system is in another level and then there are other supervisor functions will

be kept in the other directory, so whenever a constructor is called basically it gets allocated now the destructor basically, comes automatically and then basically retains this and there is the job of the destructor and then Python also normally cleans up the memory automatically.
(Refer Slide Time: 42:27)



Chapter 13: CGI Programming

- Intro to CGI programs
- Python CGI packages
- Example CGI Form script

- Python normally cleans up the memory used by an instance automatically, so this kind of thing isn't usually necessary to put in a destructor; hence, they may be less necessary than in, for example, C++

So it is not necessary to put this in the destructor ,so there may be like , then in another for example C++ ,so that is pretty much ,what I wanted to cover for today ,so we went through like 2 important topics ,one is this documentation basically, like comments ,the dir command and then doc strings, then we went on to talk about the classes for Python, so basically like those are essentially it is a user-defined data type and essentially like it is like any other object oriented language .
So it has a member function and member data and then that is for the class and then you can also define subclasses, which and we can inherit, it inherits all the properties of class ,the original class ,so it also inherits the member data and the member functions and then you can add to it some specialized functions and specialized data and then we went through like some examples of the class and how it works okay, I think that is pretty much all for today we will meet and start the CGI programming I mean in the next class thank you.