



IIT Madras

ONLINE DEGREE

Programming, Data Structures and Algorithms using Python

Professor Madhavan Mukund Class and Objects

So, continuing with our discussion of slightly more exotic aspects of Python, let us look at classes and objects.

(Refer Slide Time: 0:15)

The image shows two sequential frames of a video lecture. The slide title is "Classes and objects" and the presenter is Professor Madhavan Mukund, IIT Madras BSc Degree. The slide content is as follows:

- **Abstract datatype**
 - Stores some information
 - Designated functions to manipulate the information
 - For instance, stack: last-in, first-out, `push()`, `pop()`

Handwritten diagrams illustrate a stack data structure with "push" and "pop" operations, and a sequence of elements $x_1, x_2, \dots, x_n, x_{n+1}$.

The second frame of the slide adds the following content:

- Separate the (private) implementation from the (public) specification
- **Class**
 - Template for a data type
 - How data is stored
 - How public functions manipulate data
- **Object**
 - Concrete instance of template

So, most often classes and objects arise in the context of what are called abstract data types. So, we have data types as we know, in Python, we have lists, we have dictionaries. And when we have a data type, we have certain permitted operations on these. For a list, for example, you can append to it, or you can combine two lists using plus you can concatenate them, with a dictionary, you can create a new entry with the key, you can update it, and so on.

You can get X, extract all the keys of a dictionary, extract all the values and so on. Now, sometimes we need to create our own data type. And this data type will typically have two parts; it will have some information that is stored in it. But there may also be some discipline or some required way of controlling access to this information.

So, a typical example that most people use for this is a stack. So, what is a stack? A stack is what you think in English, it is just a pile of things come one on top of the other. Now, if I have a stack of books, for example, in a table, what can I do? I can add one more to the top of the stack. So, this is what is called in stack terminal, you 'push', or I can take the top most book of the stack, and this is called a 'pop'.

Now, I cannot take out this book, until I take out the box on top of it. Otherwise, things will fall down haphazardly. So, the idea of a stack is that I have a sequence of values. So, x_1, x_2 up to say some x_n , and when I push I add a value on the end, and when I pop, I can only take out the last value. Now I may represent this information as a list.

And if represented as a list, Python will allow me to take out or update, say the second element in this list. But as a stack, that is not allowed. If I do not respect this last in first out thing as it is called, then I cannot guarantee that the information that I stored has whatever property I would like to associate with it being a stack.

So what we want to do is have an implementation of a stack, a way of storing the information and a way of implementing this push and pop. But this implementation should only be manipulated using this public specification, I can only use pop and push, I cannot start inserting into the stack or extracting a value from the stack or updating a value just because I know it is a list. So, so the implementation should be private and you should only use the publicly allowed functions on it.

So, this is what an abstract data type essentially means. So, one of the ways of implementing abstract data types is to use this idea of a class. So, a class is a template, it tells you an abstract description of this abstract data type in terms of how the data is stored, and how the functions manipulate the stored data. And now you can make as many copies of this function as of this template as you want. And these are what are called objects.

So, I can describe what a stack should look like. And then I can create multiple stacks, it is just like I can use multiple lists, multiple dictionaries, if I update one dictionary, it does not change the other dictionary. So, when I have built in data types in Python, I can create as

many of each as I want. Similarly, a class is like a user defined it, I want to define a stack. And now I want having defined the stack, I want you to be able to create as many stacks as you want those are the objects.

(Refer Slide Time: 3:35)

Example: 2D points

- A point has coordinates (x, y)
 - `__init__()` initializes internal values x, y
 - First parameter is always `self`
 - Here, by default a point is at $(0, 0)$

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

The diagram shows a 2D coordinate system with x and y axes. A point is plotted at coordinates (a, b) . A vector from the origin to the point is labeled with a and b . A second point is shown at $(a+\Delta x, b+\Delta y)$, representing a shift from the first point.

Madhavan Mukund | Classes and objects

So, let us look at a concrete example, supposing we want to look at geometric points. So, geometric point, by geometric point just mean that we normally you would have seen this, of course in maths that you have this x, y coordinate and I take a point. So, how do I describe this point? Well, it will have some x coordinate a and some y coordinate B , which says that basically, I am a distance away from here on this direction.

And I am b distance away from here in this direction. So, the second thing tells me how I am. The first thing tells me how much to the right or left of the center. So, this is my point. So, I want to now have a way of storing these points and manipulating them. What do I do manipulating I might want to take this point and I may want to shift it there.

I might want to shift it by some a plus Δx and B plus Δy , I may want to shift it to a new position where I tell you move it 2 steps, right and 4 steps up. So, these are the kinds of manipulations that I might want to do. So, that is called for example translation. So now, in Python, the way we will do this as we use this class definition.

So, we say that I want to define a class point and the first and most important thing you need to do is to create points, so for that we have this kind of special function, which is called a constructor, which is always called `init` it with underscore and score on both sides. So, that it is a special function is not just so this underscore underscore is part of the name of the

function. Now one of the things that we need to do is remember we have many points, so each point has an identity.

So, it has to talk about its internal values and the values of other points that it might encounter. So, there is this parameter called self, which every function inside a class has, which is an identification of itself. I mean, so every object has a notion of myself or itself and other points. And now when we create a point, we have to provide its location. So, we provide two arguments a and b and internally, it stores it as x and y.

So we have self, my copy of x, my copy of y, so self dot x self dot y, and I initialize them to a and b and as in other functions in Python, if I provide some default values, I can create a point without providing a and b and then it will be at the origin. So, this is the starting point of our class point, we have this constructor called init which takes two parameters, the x coordinate and the y coordinate and initializes this point to have that x coordinate and that y coordinate.

(Refer Slide Time: 6:15)

Example: 2D points

- A point has coordinates (x, y)
 - `__init__()` initializes internal values x, y
 - First parameter is always `self`
 - Here, by default a point is at $(0, 0)$
- Translation: shift a point by $(\Delta x, \Delta y)$
 - $(x, y) \rightarrow (x + \Delta x, y + \Delta y)$

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

    def translate(self, delftax, delftay):
        self.x += delftax
        self.y += delftay
```

Handwritten notes: $l += 7$ and $l = l + 7$

Madhavan Mukund Classes and objects

So, as I decided this, discuss just now one of the things you might want to do is to take a point and shift it. So, we want to make take a point at x comma y, and shift it to x plus delta x y plus delta y. So now, inside this point, this is now a public description. So, the functionality of point permits you to translate as it is called in geometry, translate a point from here to there, translate means shift it by a certain quantity.

So, I have a point already. So, I already have a self dot x self dot way, and I want to shift it by delta x delta y, remember that this parameter is always, so I take self dot x and update it to

self dot x plus delta y. So, remember, this notation, i plus equal to 7 is a shortcut for i equal to i plus 7. So, whenever you are updating the same value on the right hand side, you can collapse this, this part into this plus equal to.

So, this is just a shortcut for saying self dot x gets updated to self dot x plus delta x, self dot y gets updated to self dot y plus delta y. So, this is my translation.

(Refer Slide Time: 7:21)

Example: 2D points

- A point has coordinates (x, y)
 - `__init__()` initializes internal values x, y
 - First parameter is always `self`
 - Here, by default a point is at $(0, 0)$
- Translation: shift a point by $(\Delta x, \Delta y)$
 - $(x, y) \rightarrow (x + \Delta x, y + \Delta y)$
- Distance from the origin
 - $d = \sqrt{x^2 + y^2}$

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

    def translate(self, deltax, deltay):
        self.x += deltax
        self.y += deltay

    def odistance(self):
        import math
        d = math.sqrt(self.x*self.x + self.y*self.y)
        return d
```

The diagram shows a 2D coordinate system with a point (a, b) in the first quadrant. A right-angled triangle is formed with the origin $(0,0)$ as one vertex, and the point (a,b) as the other. The horizontal side is labeled a , the vertical side is labeled b , and the hypotenuse (distance from origin) is labeled d . The formula $d = \sqrt{a^2 + b^2}$ is written below the diagram.

Madhavan Mukund | Classes and objects

Now, I might ask some other question, which is, how far is this point a comma b? How far is it from the origin? If I draw a straight line from $(0,0)$ to this how far is it? So, by Pythagoras theorem, this is going to be a, this is going to be b, so I am going to get square root of a squared plus b squared, because that that distance line is like the hypotenuse of a right hand, right angle triangle.

So, d is given by the square root of self dot x squared self dot y squared. And remember that this is a function in math. So, I take self dot x self dot x, which is the x squared, self dot y times self dot y, which is the y squared and apply square root of it, and return that value d. So, so far, so good, we have defined kind of our class, which can allow us to store two dimensional points and do a couple of things with it.

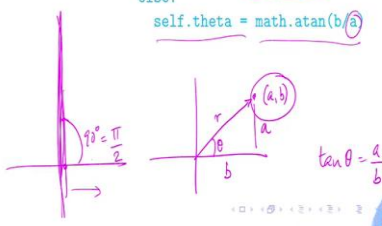
One is to translate that point by a certain displacement and the second one is to compute the distance from the origin. So now, what more can I do with this?

(Refer Slide Time: 8:20)

Polar coordinates

- (r, θ) instead of (x, y)
- $r = \sqrt{x^2 + y^2}$
- $\theta = \tan^{-1}(y/x)$

```
import math
class Point:
    def __init__(self, a=0, b=0):
        self.r = math.sqrt(a*a + b*b)
        if a == 0:
            self.theta = math.pi/2
        else:
            self.theta = math.atan(b/a)
```



Madhavan Mukund

Classes and objects

So, I might, for various reasons, decided to represent this point differently. So, if I have a point, a comma b, an equivalent way to describe it is to actually compute its distance. Okay, and compute this angle that it makes with the x axis. So, this is what is called polar coordinates. So, in polar coordinates, now notice that this is again unique, because the angle fixes at what angle it is an r tells me exactly where it is.

So, if I know r and theta, it is not difficult to imagine that it fixes a point uniquely. And this is equivalent to giving the x and the y coordinate. So, these two are obviously then interchangeable. So, we know that r is given by this x squared plus y squared square root as we saw before, because it is nothing more than the distance from the origin. And if you know trigonometry, then this height divided by the adjacent thing is tan of theta.

So, this is the definition of the trigonometric function tangent, the tangent is the opposite against the adjacent. So, the inverse of that ratio, tan inverse of y by x, which is the theta for which this is the tangent gives me the value of theta. So, I can start with x, y, and I can compute r and theta. So, that is what is happening here now. Now, the point that the thing that we are trying to emphasize is that this is a different private implementation of the same public point.

So, as a user, you still create the point giving this a comma b value, you have no idea whether internally storing it as r comma theta is storing it as a comma b. So, inside the definition of this class, now this init function takes the same values, but what it does is it instant instead of initializing self dot x it initializes self dot r, by using the square root of a squared plus b squared, remember that this is math.

So, it is math dot square root. Similarly, it initializes that theta to be the tan inverse, which is in the math functions called a tan, arc tangent sometimes, a tan of b by a. But there is a problem because a could be 0. So, if a is 0, then what we are saying is that we are somewhere on this y axis, because a is the displacement in the x direction. So, if a is 0, we are on the y axis. So, the angle is really 90 degrees, which all these angles are actually represented in radians. So, 90 degrees pi by 2, so we will take it as pi by 2.

So, if the a coordinate, that is the x coordinate of the point I am trying to create is along the y axis, then I will declare the angle to be 90, otherwise I will take it to be the tan inverse. So, at this point, from a user's perspective they should not care whether it is r theta or x, y, because the information is actually interchangeable. So, now let us look at the two functions that we had defined, the two functions defined what translate and this distance from the origin.

(Refer Slide Time: 11:29)

Polar coordinates

- (r, θ) instead of (x, y)
 - $r = \sqrt{x^2 + y^2}$
 - $\theta = \tan^{-1}(y/x)$
- Distance from origin is just r

```
import math
class Point:
    def __init__(self, a=0, b=0):
        self.r = math.sqrt(a*a + b*b)
        if a == 0:
            self.theta = math.pi/2
        else:
            self.theta = math.atan(b/a)

    def odistance(self):
        return(self.r)
```

Madhavan Mukund | Classes and objects

So, distance from the origin is very easy now, because I am actually representing it explicitly as this r quantity. So, if I am asked the diff, the distance from the origin of a point, instead of having to compute this x squared plus y squared and taking the square root, I can just return the current value of self dot r. So, this becomes easier.

So, this could be one justification, if you are going to often ask the distance, then this representation makes that calculation easier because you calculate the self dot r once and for all, every time the point changes, after every call to distance just report set value, you never have to do this calculation again. So, that these are some of the reasons why you might choose a different representation, depending on how the functions are going to be called the work that you have to do down the line might improve.

(Refer Slide Time: 12:13)

Polar coordinates

- (r, θ) instead of (x, y)
 - $r = \sqrt{x^2 + y^2}$
 - $\theta = \tan^{-1}(y/x)$
- Distance from origin is just r
- Translation
 - Convert (r, θ) to (x, y)
 - $x = r \cos \theta$, $y = r \sin \theta$
 - Recompute r, θ from $(x + \Delta x, y + \Delta y)$

```
def translate(self, deltax, deltay):  
    x = self.r * math.cos(self.theta)  
    y = self.r * math.sin(self.theta)  
    x += deltax  
    y += deltay  
    self.r = math.sqrt(x*x + y*y)  
    if x == 0:  
        self.theta = math.pi/2  
    else:  
        self.theta = math.atan(y/x)
```

Diagram: A right triangle with hypotenuse r , horizontal side x , and vertical side y . The angle at the origin is θ . Handwritten notes: $\sin \theta = \frac{y}{r}$, $\cos \theta = \frac{x}{r}$.

Polar coordinates

- (r, θ) instead of (x, y)
 - $r = \sqrt{x^2 + y^2}$
 - $\theta = \tan^{-1}(y/x)$
- Distance from origin is just r
- Translation
 - Convert (r, θ) to (x, y)
 - $x = r \cos \theta$, $y = r \sin \theta$
 - Recompute r, θ from $(x + \Delta x, y + \Delta y)$
- Interface has not changed
 - User need not be aware whether representation is (x, y) or (r, θ)

```
def translate(self, deltax, deltay):  
    x = self.r * math.cos(self.theta)  
    y = self.r * math.sin(self.theta)  
    x += deltax  
    y += deltay  
    self.r = math.sqrt(x*x + y*y)  
    if x == 0:  
        self.theta = math.pi/2  
    else:  
        self.theta = math.atan(y/x)
```

On the other hand, now if I have to translate, I have to do a lot of work. Because the translation, remember is going to be still expressed in terms of this x plus Δx and y plus Δy , but I do not have x and y . So, I have to go backwards, I have to go from r θ to x , y . Now again, using trigonometry, if this is x , and this is y , and this is θ .

Then \sin of θ . And this is say the hypotenuse, the \sin of θ is this large, \sin of θ is y by r , and \cos of θ is x by r . So, from this, you get the x is $r \cos \theta$ and y is $r \sin \theta$. So, you can translate backwards from r to x , r θ to x comma y , then you can apply this transformation at the x y level and then convert back so you have to do this conversion in both directions to go from r θ to x y , apply this plus Δx plus Δy and go back.

So, this is what is happening here. So, you compute a new value for x and y , given the current values inside for r and θ . By applying \cos and \sin , then you translate. But x and y are not my internal representation, I am just using it as an intermediate thing. So, you have to convert back. So, this code is essentially the same code that we had when we did the `init`.

So, the distance from the origin becomes simpler, translation becomes more difficult. In the x, y thing, the translation was easier. But this is a margin required computing the square root. So, it is a trade-off, you have to decide which is which. But the important thing is that from the user's perspective, nothing has changed, you still create a point by providing the x coordinate and the y coordinate.

You still provide the translation function by giving the x displacement in the y displacement, you still ask for the distance, and get it back as a distance from the origin. So, the interface, as it is called, has not changed. So, what the user sees, as far as the class is concerned, has not changed. And this is the whole point of this whole discipline of using classes and objects that you can change the internals of a class, but provide the same public face.

So, the other person's code does not change. So you, you do not end up. So, this is something that we do not want to do that. Because I change my code, somebody else's code stops working. It might work better or worse, it might be more efficient, less efficient, that is a different matter, but it should not give wrong answers. So, that is one of the things that we get.

(Refer Slide Time: 14:41)

Special functions

- `__init__()` — constructor
- `__str__()` — convert object to string
 - `str(o) == o.__str__()`
 - Implicitly invoked by `print()`

```
class Point:
    ...
    def __str__(self):
        return(
            '(' + str(self.x) + ', '
            + str(self.y) + ')'
        )
```

Handwritten notes: `print(p)` points to `__str__`; `str(x)` points to `str(self.x)`; `print(x)` points to the return value.

Madhavan Mukund Classes and objects

So, to wind up this discussion of classes this look at some special functions, so we already saw one the constructor function has this underscore underscore init. Another useful function is this function called str, which converts an object to a human readable form as a string.

So, for instance, we have might want to take our point and write it out, if I want, if I ask for the value of p, you might want to see it in this form as a string with open bracket, close bracket and x comma y if separated by commas as you would normally read it in a math notation, right. But we have to do that explicitly.

So, we will have to take define a string function, which will take self dot x self dot y, convert them into strings, and then using the normal string operations, take these special symbols, open bracket, comma, close bracket and put them around and in between, so we are really assembling a string by taking the open bracket, the string value of x, comma, the string value of y, and close bracket.

Now, we normally do not use this explicitly, but it is always used implicitly by print, whenever we print a value of a variable, whenever I say print x, for any x, k, what happens is that this is actually converted to a string of x. So, if I print a number, what happens is the number is first converted to a string and then print. So, print can only print strings in some sense. So, a string is called implicitly.

So, the same thing happens here. So, if I want to actually if I say print p, where p is a point, so what will happen is that it will go through this process, and it will come out. So, that is why the string is an important function.

(Refer Slide Time: 16:25)

Special functions

- `__init__()` — constructor
- `__str__()` — convert object to string
 - `str(o) == o.__str__()`
 - Implicitly invoked by `print()`
- `__add__()`
 - Implicitly invoked by `+`

```
class Point:
    ...
    def __str__(self):
        return(
            '+' + str(self.x) + ', ' +
            str(self.y) + ')'
    def __add__(self, p):
        return(Point(self.x + p.x,
                     self.y + p.y))
```

Diagram illustrating vector addition: $(3, 4) + (5, 7) = (8, 11)$. The vectors are shown as arrows originating from the origin, and their sum is shown as a longer arrow.

Presenter: Madhavan Mukund, Classes and objects

Similarly, you can do functions which overload operators, supposing I want to have the possibility of taking a point p, and taking a point q and now defining a point p plus q. So, what does p plus q do it will have this plus this in the x axis and this plus this in the y axis. So, if I have, for example, 3 comma 4 as p, and I have 5 comma 7 as q, then I will end up with a point which is 3 plus 5, 8, and 4 plus 7, 11. This is what p plus q is.

So, how do I define that? Well, I define it using this function called add. So, now this is a situation where you see this no use of self. So, I am a point self and I get another point p and I want to add myself to it, or p to me. So, I take my x and I take the px that is the new x coordinate, I take my y and I take the new y that is the new y coordinate. But now this is a different point. This is not myself, this is not p.

So, I have to create a new point. So, I have to construct a point out of this by passing this new x and this new y value that I have calculated to point which creates a new point, this class creates a new point and returns a new object. So, when I say p plus q, p does not change q does not change, I get back a new point whose coordinates are the sum of the coordinates in each direction of p and q.

(Refer Slide Time: 17:49)

Special functions

- `__init__()` — constructor
- `__str__()` — convert object to string
 - `str(o) == o.__str__()`
 - Implicitly invoked by `print()`
- `__add__()`
 - Implicitly invoked by `+`
- Similarly
 - `__mult__()` invoked by `*`
 - `__lt__()` invoked by `<`
 - `__ge__()` invoked by `>=`
 - ...

```
class Point:
    ...
    def __str__(self):
        return(
            '+' + str(self.x) + ', '
            + str(self.y) + ')'
    )
    def __add__(self, p):
        return(Point(self.x + p.x,
                     self.y + p.y))
```

Madhavan Mukund Classes and objects

So, just like add, you can also do multiply, you can also use, you can define how to compare points. For instance, you might say that one point is smaller than other point if it is smaller, and both the x and the y. So, you can use less than, greater than, greater than equal to, so you can define all these functions.

So, then you can manipulate these user defined objects, and put them into conditions and print statements and all that exactly as you would a built in type in Python. So, it gives you the flexibility of using your objects, you can sort them for example, you can take points and sort them in increasing order because you have a way of doing these comparisons directly on points.

So, you do not have to write a complicated thing each time. You can just use a normal notation that you use for sorry if this is less than that exchange and so on. So, it is very convenient to be able to work with these objects and classes directly.

