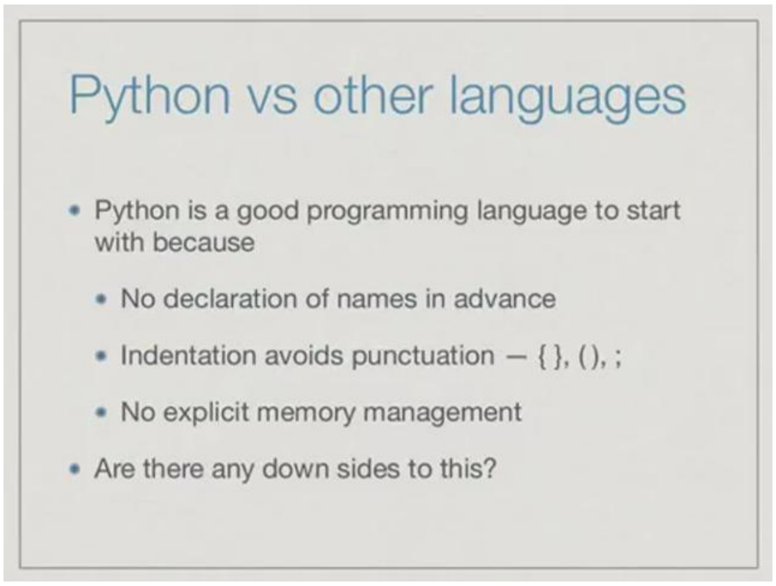
Programming, Data Structures and Algorithms in Python
Prof. Madhavan Mukund
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Week - 08
Lecture - 05
Wrap-up, Python vs. other languages

(Refer Slide Time: 00:02)



Python vs other languages

- Python is a good programming language to start with because
  - No declaration of names in advance
  - Indentation avoids punctuation — { }, ( ), ;
  - No explicit memory management
- Are there any down sides to this?

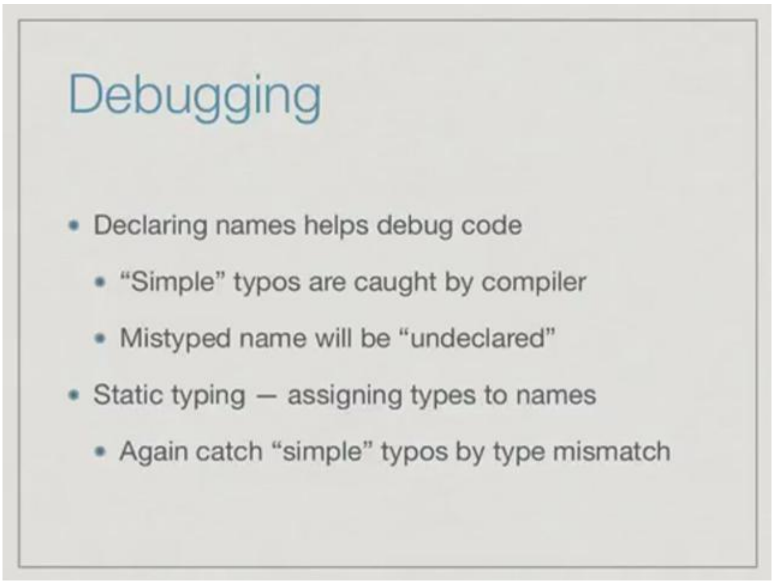
We have come to the last lecture of this course. So, instead of going into more features of Python of which there are many that we have not described. Let us take some time instead to reflect about Python as a language and compare it to some of the other languages which exist. Though you may not be familiar with them, I would like to highlight some aspects of Python which are different from other languages and also argue whether they are better or worst.

Why did we choose Python to do this course? Well, Python is a very good language to start programming for many reasons. One reason is that it’s syntax is very simple and part of the simplicity comes from the fact that you do not have to declare names in advance, so you do not have to start doing a lot of things before you write your code. You can jump into the Python interpreter for example, and keep writing statements without worrying about what x is and what y is, as you go along you can define values and manipulate them. So, that makes it very convenient and very nice to start programming because you do not have to learn a lot of things in order to write even basic programs.

The way in which Python programs are laid out with indentation to indicate when a loop begins and when an 'if' block begins. We do not have to worry about a lot of punctuation: braces, brackets, semicolons, which are part and parcel of more conventional languages. Once again this makes a language little easier to learn and a little less formidable for a new comer to look at.

The other thing which is little more technical which we will... I will talk about a little bit in this lecture is that we do not have to worry about storage beyond understanding what is mutable and immutable. If you need to use a name x, we use it; if we need a list l we use it, if we need to add something to an l, we just say append. We never bother about where the space is coming from or where it is going. These are all plus points of Python. So, what are the minus points? So are there things that are bad about Python?

(Refer Slide Time: 02:05)



Debugging

- Declaring names helps debug code
  - "Simple" typos are caught by compiler
  - Mistyped name will be "undeclared"
- Static typing — assigning types to names
  - Again catch "simple" typos by type mismatch

The first thing is to do with the lack of declarations - the lack of declarations is often a very good thing, because you do not have to worry about writing a lot of stuff before you start programming, but it is also a bad thing because many programming errors come from simple typing mistakes. Very often you need to write x and you write y. Now in Python if you write a y and you assign it a value somewhere where you meant an x, Python will not know that you were supposed to use x and not y because every new name that comes along is just happily added to the family of names which your program manipulates.

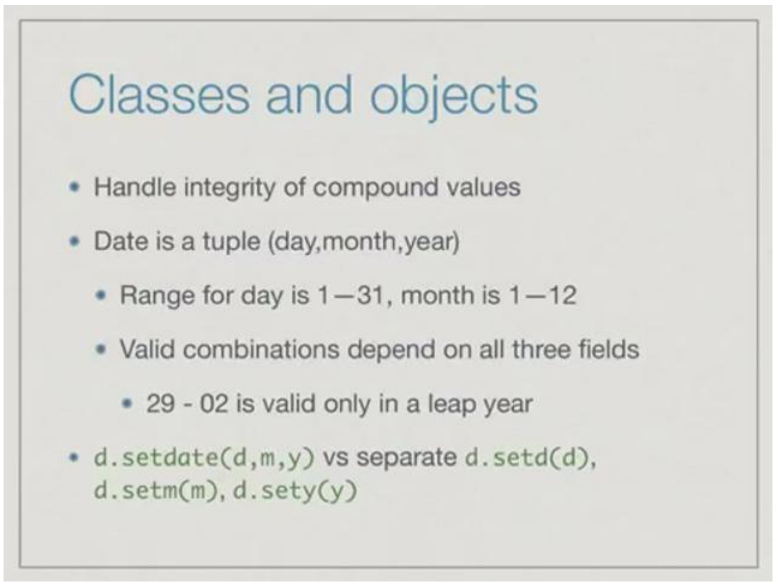
These kind of typos can be very very hard to find, and because you have this kind of dynamic name introduction with no declarations, Python makes it very difficult for you as a programmer to spot these errors. On the other hand if you declare these names in advance which happens in other languages like C or C++ or Java. Then if you use a name which you have not declared then the compiler will tell you that this name has not been seen before and therefore something is wrong. So, a miss typed name can be easily caught as an undeclared name if you have declarations.

Whereas, in Python it will just happily go ahead and create a new value for that name and pretend that there are two names now while you think there is only one and create all sorts of unpredictable errors in your later code. The other side of this is typ ing. So, in Python we have seen that names do not have types they only inherit the type from the value that they have. You could say at some point x equal to 5 and later on assign x to a string and later on assign x to a list and Python will not complain, at each point given the current value of x legal operations are defined as per that value.

Now this is again nice and convenient but it can also lead to errors for the same reason you might be thinking of x as an integer, but somewhere halfway through your program you forgot that it is an integer and start assigning it some different type of value. Now if you had announced to Python that x must always be an integer then this name must only store an integer value, presumably as a compiler it would catch it internally.

A lot of errors are either typos in variable names or misguided usage of names from one type to another, both of these can be caught very easily by compilers if you have declarations of names, both of these get uncaught or they are left uncaught by Python and they allow you to propagate errors and these errors can be very difficult to find. So the down side of having this flexibility about adding names and changing their types, values as you go along is that debugging large programs requires a lot more care on your part, writing large programs requires care, debugging is very difficult.

(Refer Slide Time: 04:48)



Classes and objects

- Handle integrity of compound values
- Date is a tuple (day,month,year)
  - Range for day is 1—31, month is 1—12
  - Valid combinations depend on all three fields
    - 29 - 02 is valid only in a leap year
- `d.setdate(d,m,y)` vs separate `d.setd(d)`, `d.setm(m)`, `d.sety(y)`

The other part has to do with the discussion that we had towards the second half of the course about user defined data types in terms of Classes and Objects. So, the first thing that is a direct consequence of not having declarations is that we cannot assert that a name has a type. In particular we saw that if we want to use something as a list we have to first declare it to be an empty list, so we have to write something like l is equal to this to say that hence forth I can append to it.

The first append or the first operation on the list that I do will have to be legal, for that I have to tell it that it is a list. This is more or less like a declaration except it is not quite a declaration. I am actually creating an empty object of that type. In the same way if I want a dictionary I have to give it a completely new name like this. I have to say d is equal to an empty dictionary. So, there is no way to assign a type to a name without creating an object of that type.

This is actually a problem with the kind of user defined types that we have, for instance it is very convenient to be able to define an empty tree without having to create a tree, a node. For instance, if you had type declarations you can say that the name t is of type tree and then you can use this value like none - all programming languages have such a value which denotes something that does not exist. You can say that there is not one none, but many nones and by context this none is a none of type tree.

Python also uses it. There is only one value none and you can use none for anything, but when it has none, it has no type - that is the difference. If in Python a name has the value none, it has by definition no type, whereas if you had declarations you can say that this name has a type, it just does not happen to have a value.

And this is typically what you want for an empty node or an empty tree. We had if you remember a very cumbersome way of dealing with this in order to make recursive tree exploration work better, we actually added a layer of empty nodes at the frontier and extended our tree by one layer just so that we could easily detect when we reach the end of a path. Now this can be avoided actually, if you have type declarations, so this is another feature which makes actually... the lack of declaration makes things a little bit more complicated in Python which one doesn't normally come across in beginning programming.

The other thing is much more serious. So, this is more to do with convenience and representation of empty objects, but without declarations you really cannot implement the kind of separation of public and private that you want in an object. Remember our goal in an abstract data type was to have a public interface or a set of functions which are legal for a data type and have a private implementation. For instance, we would have a stack, we might implement as a list, but we would only like pop and push. Same way a queue may be also a list, but we only want the add and remove queue at opposite ends and we do not want to confuse this with the list operations; we do not want things like append and extend to be used indiscriminately.

The other part of it is that we do not want the data itself to be accessible, we do not want to say that if given a point p, I can use p dot x and p dot y from outside and directly update the values. Because, this is sensitive to the fact that tomorrow I might change from x dot y... x and y representation to r and theta, we said that we might have situations where we might prefer to represent our point using r and theta. Now if some programmer started using point in the days when we are using x and y and started manipulating p dot x and p dot y directly outside the code.

See if it is inside the class code, then as the maintainers of the class we would make sure that wherever we used to use p dot x and p dot y we now use p dot r and p dot theta. So, it is an internal thing we change from x, y to r, theta we internally update all the code within the class to be in terms of r, theta and not x, y. But if somebody outside is using x and y and these values no longer exist then what happens is that for the outside person their code stops working, because we have changed an internal implementation of point.

And this is a very dangerous situation which happens quite often, and this is where it is important to separate public from private if they do not have any access to the private implementation of the point then they cannot use p dot x and p dot y. Outside this problem is avoided. So just to reiterate, supposing we have a stack implemented as a list and we only allow public methods push and pop, a person who is exploiting this fact that it is a list could directly add something to the end and violate the heap property for instance or the stack property.

So, we could get situations where the data structure is compromised because the person is using operations which are legal for the private implementation, but illegal for the abstract data type which we are trying to present to the user. So for this, the only way we can get around this is to actually have some way of saying that these names are private and these names are public. Now it is a not that Python does not have declarations.

If you remember Python has this global declaration which allows you to take an immutable value inside a function and say it refers to the same immutable value outside. There are situations in which Python allows declarations, but there are many other things where it could allow declarations and make things more usable, but it does not and this is one example.

In languages like java or... you will find a lot of declarations saying private and public, and this looks very bureaucratic but it is really required in order to separate out the implementation from the interface. Actually, in an ideal world, the implementation must be completely private. So, you should never be able to look at x and y directly. For instance if you want x there should be a function called get x which gives you the x value, there is function called set x which sets the x value.

This may look superficially the same as saying p dot x equal to v, but the difference is that we do not know actually there is an x, x is an abstract concept for us. So, x coordinate is just... it is a property of the point which we can set. Now how we set it is through this function. So, if inside the functions we start setting r theta as the representation, then changing the x value will correspondingly change r and theta indirectly. So, when we say get x, for instance, it does not actually read the x value, it gets r cos theta and we say set x it will change the r to account for the new x and recompute the theta.

In this way if we have only these functions to access the thing then we have only conceptual values inside and these conceptual values are manipulated through these functions and we do not know the actual representation. So this is the ideal world, everything is hidden and we only have these functions, but this is cumbersome for every part of the data type we have to use these functions. And partly the reason why we have to have this private public declaration is that the programmers are not happy with having to always invoke a function, sometimes they would like to directly assign. They would like some parts of the data type to be public. So, they can say p dot x equal to 7.

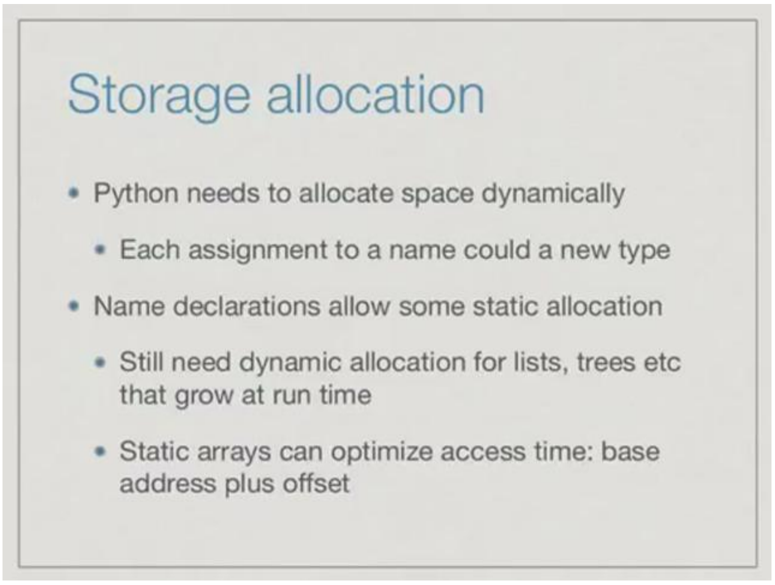
But in general this is the style that one would ideally advocate for object oriented programming - make all the internal names and variables private and only allow restricted access, so that they are used in the appropriate way and the use does not get compromised if the internal representation changes. If we move x, y to r, theta, get x and set x will still work, whereas p dot x may not be meaningful anymore.

Another reason to have this style of accessing values is sometimes you do not want individual values to be actually accessed individually. Supposing, we have a date, a date is typically a three component field which has the day, month and year. And these have their own ranges: the valid days for a month range from 1 to 31 and the months range from 1 to 12, but not every month has 31 days.

The valid combination for a day and month depends on both these quantities and in fact it depends on the year also because we cannot have 29th of February unless it is a leap year. So, if we update day or month we have to be careful that we are updating it legally, we can start with the month February with the legal date like 15 and then change the date from 15 to 31, and now end up with an illegal date which is 31st of February. So what we need actually is to have a composite update operation which sets the date by providing all three values, rather than supplying three separate operations to update the three fields separately.

Even if you have these functions you do not always want to give these functions individually, because there might be some constraints between the values which have to be preserved and you can preserve those by controlling access to them. So you can say, you cannot set the day separately and the month separately, you must set them together. This is how one other reason why it is good to keep all the implementation private, not allow direct update and then control the way in which you update the values.

(Refer Slide Time: 14:09)



Storage allocation

- Python needs to allocate space dynamically
  - Each assignment to a name could a new type
- Name declarations allow some static allocation
  - Still need dynamic allocation for lists, trees etc that grow at run time
  - Static arrays can optimize access time: base address plus offset

Now let us come to Storage allocation. How the names and values we use in our program are actually allocated and stored in memory so that we can look them up. Because in python we use names on the fly we keep coming up with names and the values keep changing, Python cannot decide in advance that it needs a space for x, for one integer because tomorrow this x might be a list and it does not even know which names are coming. So Python has to allocate space always in a dynamic manner, it has to keep as you use a name, it has to find space for it and the space requirement may change if the value changes it's type.

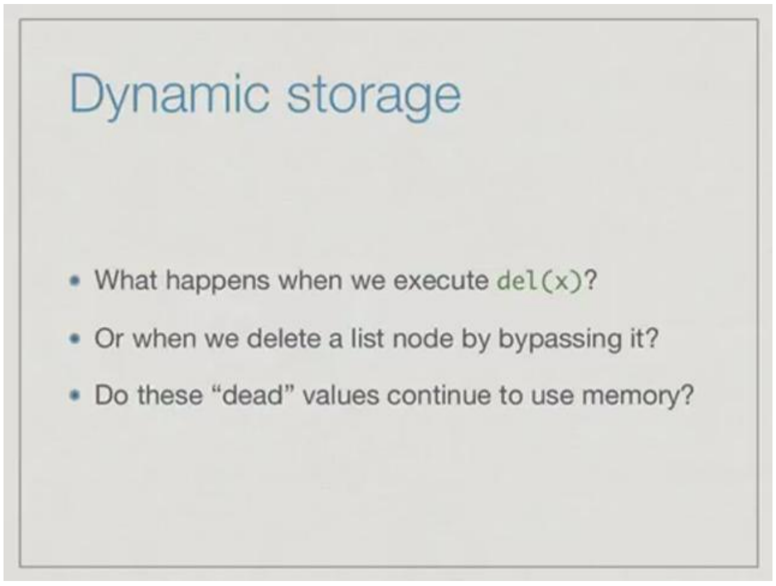
On the other hand if I have a static declaration, then if I say that i is an integer and j is an integer and k is an integer then the compiler can directly declare in advance some space in the memory to be reserved for i, j and k. Now this is particularly useful for arrays, we mentioned that in an earlier lecture, the difference between arrays and lists.

In a statically declared situation, an array of size hundred will actually be allocated as a block of hundred contiguous values without gaps. This means that I can get to any entry in the array by looking at... the knowing the first value and then how many values to skip. So, if I want the 75th value, I can go to the first value and calculate where the 75th value will be if I just jump over that many values and get there directly. This gives us what is called a Random Access Array. It does not take any more time to get to the first element than the last element.

Whereas in a list, as we saw even in our object based implementation to get to the ith element we have to start with the head and go to the first, second, third, so it takes time proportional to the position. If we have static allocation we can also exploit the fact that we can get peculiar random access arrays which Python actually does not have. Now just because we have declaration does not mean everything is declared statically.

Even in languages like Java and C++, we would do this object oriented style where we would have a template for a class and then we would create object of this class dynamically as a tree or list grows and shrinks, we will create more objects or remove them. So there is a dynamic part also, but it is exclusively for this kind of user defined data types and the static part takes care of all the standard data types that you use for counting and various standard things and particular arrays very often. If you know in advance, you need a block of data of a particular size, arrays are much more efficient than lists.

(Refer Slide Time: 16:39)



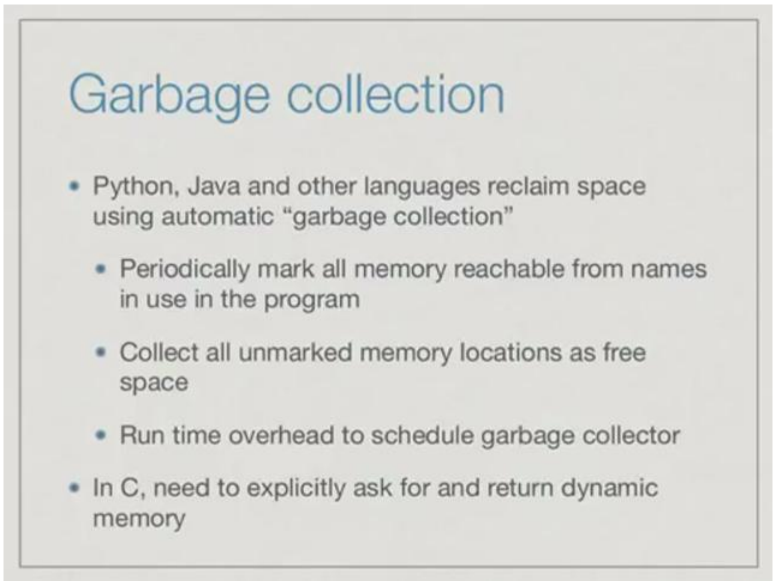
Dynamic storage

- What happens when we execute `del(x)`?
- Or when we delete a list node by bypassing it?
- Do these "dead" values continue to use memory?

One part of storage allocation is getting it, the other part is de allocation, giving it up. So, we saw that in Python you can remove an element from the list by saying del(l[0]). In general you can un allocate any name's value by saying del(x). So what happens when we... does this give up the storage? When we say del(x) we are saying that the space that x is currently using for it's value is no longer needed. So what happens to that space? Who takes care of it? Similarly when we had a list, if you remember, when we wanted to delete a list, if we had a sequence of things and we wanted to delete a list, we did not actually delete it, we just bypassed it, we said that the first element points to the third.

Now, logically this thing is no longer part of my list but where has it gone? Has it gone anywhere or is it still there in my memory, blocking space? So what happens with the dead values which we have unset by using del or we have bypassed by manipulating the links inside an object and so on.

(Refer Slide Time: 17:40)



Garbage collection

- Python, Java and other languages reclaim space using automatic "garbage collection"
  - Periodically mark all memory reachable from names in use in the program
  - Collect all unmarked memory locations as free space
  - Run time overhead to schedule garbage collector
- In C, need to explicitly ask for and return dynamic memory

So it turns out that languages like Python, also other languages like Java… this has nothing to do with declaring variables, it has to do with how space is allocated. They use something called Garbage collection. Garbage in the programming language sense is memory which has been allocated once, but it is not being used anymore and therefore is not useful anymore, because we cannot reach it in some sense. So, it is like that list example: there was an element in our list which we could reach from the head. At some point we deleted it, now we can no longer reach that value nor can we reuse it because it has been declared to be allocated, so it is garbage.

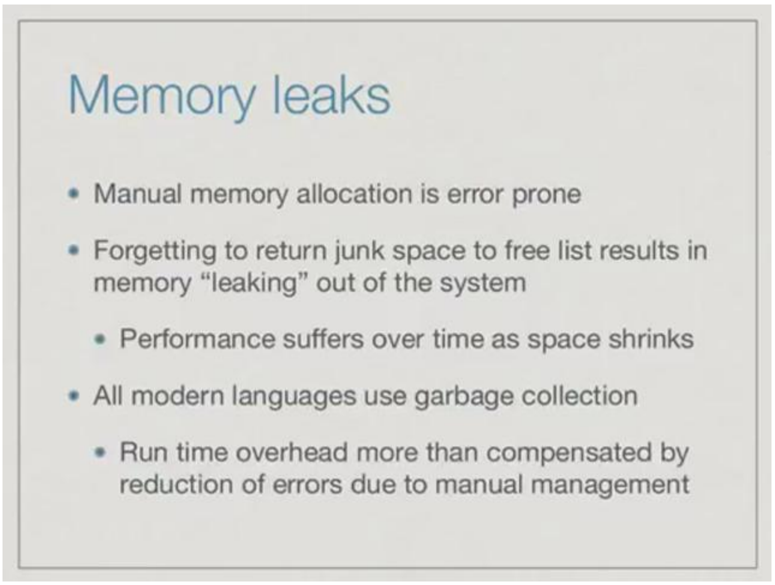
So roughly speaking how does garbage collection work? Well, what you do is you imagine that you have your memory and then you have somewhere some names, n, m, x, l and so on in your program and you know where these things point, this is somewhere here, this is somewhere here, this is somewhere here, this is somewhere here. So, you start with the names that you have and you go on... you mark this thing, you say this is mine, this is mine, this is mine, and this is mine.

Now this could be a list, l could be a list, it could be that this in turn points to the next element in the list, so it goes here so then you point to this and say this is also mine. It is what the names point to plus what their pointed values point to, you keep following this until you mark all the memory that you can actually reach from your name, so the names in your program and all the memory that they can indirectly reach. Now I can go through and say that everything which has not been marked is not in use, so I can go and explicitly free it. This is the second phase. You collect all the unmarked memory locations and make them free and proceed.

Now, this is a process which runs in parallel with your program, at some point logically speaking you have to stop your program and mark the memory, release space and then resume your program. So there is an overhead. Some languages like C do not have this garbage collection built in. So in C if you need dynamic memory, so remember that all the things that you declare in advance are allocated statically, you cannot undo them, you cannot say hence forth I do not need them.

But if you have something like a list or a tree where you are growing and shrinking, you will ask for a memory, saying, I want memory worth one node, then you will populate it. When you delete something it is your responsibility as a programmer to say this was given to me sometime back, please take it back. You have to, as a programmer, ask for dynamic memory and more importantly when you are no longer using it, return it back. So, C has this kind of programmer driven manual memory allocation.

(Refer Slide Time: 20:23)



Memory leaks

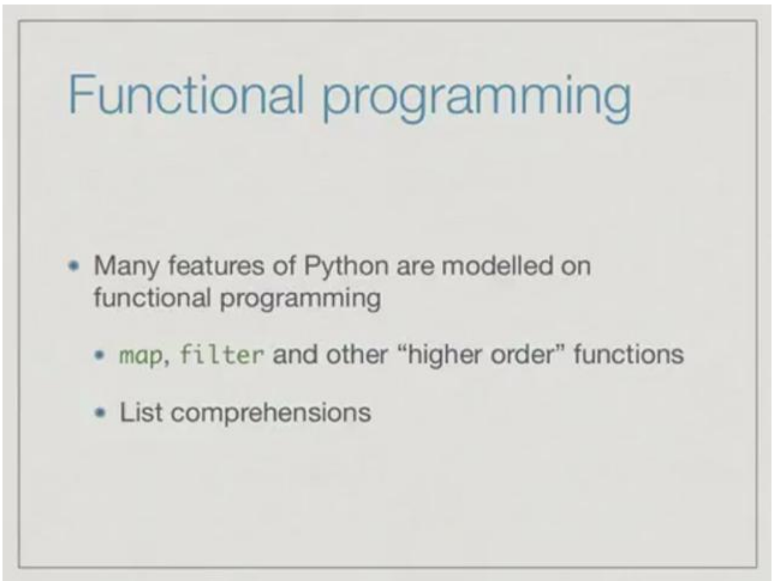
- Manual memory allocation is error prone
- Forgetting to return junk space to free list results in memory "leaking" out of the system
  - Performance suffers over time as space shrinks
- All modern languages use garbage collection
  - Run time overhead more than compensated by reduction of errors due to manual management

This is quite error prone. As you can imagine there might be just a simple case where a programmer writes a delete operation in a tree or a list and forgets to give the memory back. So, this means that every time something is added and removed from a list or a tree, in such a program, that thing will be residing in memory leaving it to be used when it is actually not used. So over a period of time this thing will keep filling up. If you take the flip side and you look at the free memory you think of the free memory as a fluid then the free memory is shrinking. So this is called a Memory leak, the memory is leaking out of your system, it is not really leaking out, it is getting filled out.

This is the terminology. So you might see somewhere in some text, the word memory leak. It is just refering to the fact that memory that has been allocated to a program is not being properly de allocated when it is no longer in use, and the symptom of this will be that as the program runs longer and longer, it will start taking up more and more memory and making the whole program very much more sluggish. So, the performance will shrink, will suffer over time as the space shrinks.

So, virtually speaking all modern languages use garbage collection because it is so much simpler. Though there is a runtime overhead with this mark and release kind of mechanism, the advantages that you get from not having to worry about it and not trusting the programmer as such, that you avoid the memory leaks by making sure that your garbage collection works rather than relying on the good sense of the programmer to make sure that all the memory allocated is actually de-allocated.

(Refer Slide Time: 21:58)



Functional programming

- Many features of Python are modelled on functional programming
    - `map`, `filter` and other "higher order" functions
    - List comprehensions

For a final point of this last lecture, let us look at a completely different style of programming which is called Functional programming. So, Python and other languages we have talked about are what are called imperative programming languages. So in an imperative programming language you actually give a step by step process to compute things. You assign names to keep track of intermediate values, you put things in lists and then you have to basically do the mechanical computation: we have to simulate it more or less, so a sequence of instructions. So you have to know a mechanical process for computing something more or less before you can implement it.

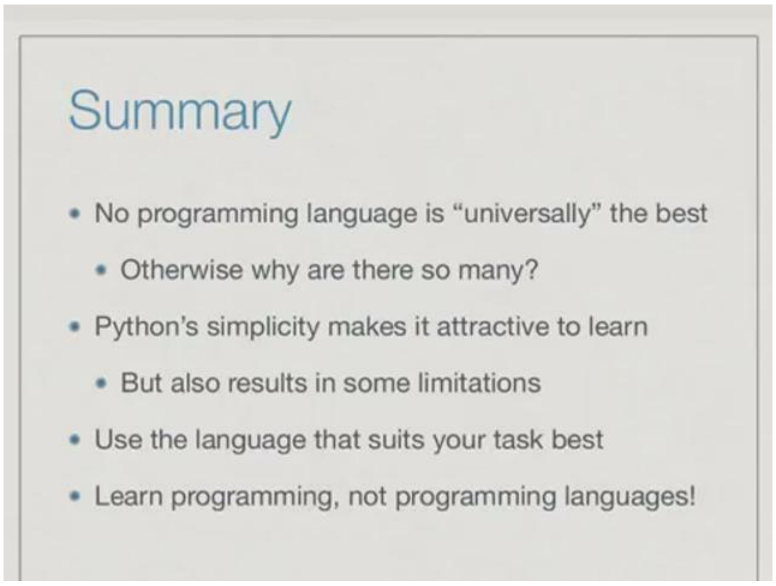
Functional programming is something which tries to take the kind of inductive definitions that we have seen directly at heart and just use these as the basis for computation. So, you will directly specify inductive functions in a declarative way what to conclude, so here is a typical declaration. For example, this is a Haskell style declaration. So, the first line with this double colon says that factorial is a function and this is it's type, it takes an integer as input and produces an integer as output. This is saying that factorial is a box that looks like this. It takes ints and produces ints, so this is the thing that you have. And then it gives you the rule to compute it. It says factorial of 0 is 1 and then the rules are processed in order, so if it comes to the second rule, it means at this point that n is not 0. If n is not 0, then n times factorial n minus 1 is the answer.

The actual way that Haskell works is by rewriting. We will not get into that, but the main point is that there is no mention here about the intermediate names, it is just taking the parameters passed to the function and inductive or recursive calls and how to combine them. So here is another example. If you want to add up the elements in a list, so this Haskell's type for a list of integers, so it takes a list of integers and produces an integer. So, you would say that for a list which is empty, the sum is 0 and again coming here sum is not... this is not empty if it comes here because we go in order.

If the first list does not match, the second list must be not empty, then it has functions such as head or tail to take the first element or the last element. So, if you have a non empty list, the sum is given by taking the first element and then inductively adding it to the inductive result of computing the rest. This is a completely different style of programming which is called Declarative programming and you can look it up. It is very elegant and it has it's own uses and features.

Python has actually borrowed some of it's nice features from functional programming. And one of them in particular that we have seen is this use of map and filter and in general the idea that you can pass functions to other functions. So, these are all naturally coming from functional programming and map and filter which allows to take a function and apply to every element of a list. And then resulting from this, very compact way of writing lists using list comprehensions in one line, combining map and filter. These are features of functional programming which are integrated into Python.

(Refer Slide Time: 25:12)



Summary

- No programming language is "universally" the best
    - Otherwise why are there so many?
- Python's simplicity makes it attractive to learn
    - But also results in some limitations
- Use the language that suits your task best
- Learn programming, not programming languages!

To conclude, one can never say that one programming language is the best programming language, because if there were a best programming language, then obviously everybody would be using that language. The very fact that there are so many programming languages around, it is obvious that no programming language is universally better than every other. So what happens is that, you choose a language which is best for you.

In particular here we were trying to learn programming and various aspects of programming data structures, algorithms. And Python being a simple language to start working with and to use and it's nice built-in things like dictionaries and stuff like that, exception handling, are done in a very nice way, so it makes it very attractive to learn. But as we saw, the same things that make it attractive to learn, the lack of declarations also limit it's expressiveness. We cannot talk about privacy in terms of implementations or objects and so on.

So the moral of the story is that when you have a programming task, you know what you want to do, look around for the language that suites your task the best. Do not be afraid of learning programming languages to do the task, because once you have learnt one programming language, it is actually not that much difference between one and the other, it is just minor differences.

Of course, there are different styles like functional programming which look very different, but more or less if you have a little bit of background, you can switch programs from one language to another by just looking up the reference manual and working with it. Of course, if you use a programming language long enough, then you should be careful to learn it well, but for many things you can just get by on the fly by just translating one language to another.

So, the main message is that you should focus on learning programming. Learning how to translate, first of all coming up with good algorithms, how to translate algorithms into effective implementations, what are the good data structures, so your focus should be on algorithms, data structures, the most elegant way in which you can phrase your instructions. And then worry about the programming language.

It is a mistake to sit and learn a programming language; nobody learns a programming language, you learn features of a programming language and put them to use as you come up with good programs. So, with this I wish you all the best and I hope that you have a fruitful career ahead in programming.

Thank you.