

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

Week - 05
Lecture - 06
Pass, del () and None

(Refer Slide Time: 00:01)

Doing nothing

- Blocks such as `except:`, `else:`, ...cannot be empty
- Use `pass` for a null statement

```
while(True):  
    try:  
        userdata = input("Enter a number: ")  
        usernum = int(userdata)  
    except ValueError:  
        pass  
    else:  
        break
```

For the last lecture this week we look at some useful things which will crop up and which do not fit anywhere else specific so we just combined a couple of them into this one single presentation.

So, we had an **example** when we are doing the input and print about how to prompt the user in case they had an invalid entry. We were trying to read a number and what we said was that we would input some string that the user provides, so give them a message saying enter the number they provide us with the string and then we try to convert it using the `int` function. And this `int` function will fail if the user has provided a string which is not a valid integer, in which case we will get a value error. And we get a value error we print out a message saying try again and we go back.

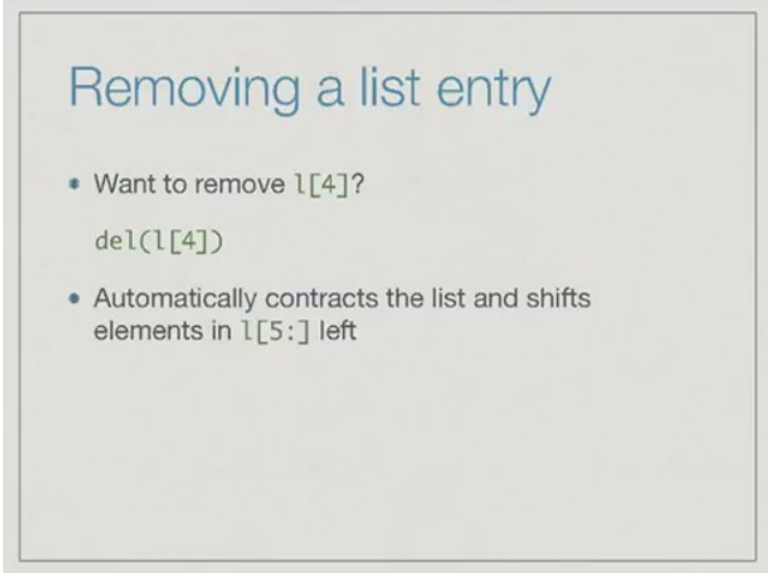
Finally, if we succeed, that is this try succeeds the `int` works then we will exit from this try block go to the `else` and the `else` will break out to of the loops. So, this we had already seen. Now the question is, what if we want to change this so that we do nothing we do

not want to do this. In other words if the user does not present a valid value instead of telling them why we just keep saying enter a number I mean we have seen this any number of times right, you go to some user interface and you type something wrong it does not tell you what is wrong it just keeps going back and back and back and asking to type again. So, how would you actually program something as unfriendly as that?

What we want to say is, if I come to this value error do nothing. Now the Problem with python is that wherever you put this kind of a colon it expects something after that, you cannot have an empty block. So if I put a colon there must be at least one statement after that. This is a syntactic rule of Python you cannot have an empty block. But here I want to do nothing, I want to recognize there is a value error and then go back here that is fine, but I do not want to do anything else.

How do I do nothing in Python? So the answer is, that there is a special statement called pass. Pass is a statement which exists only to fill up spaces which need to be filled up. So when you have blocks like except or else, if you put the block name there you cannot leave it empty. In this case you can use the word pass in order to do nothing.

(Refer Slide Time: 02:28)



Removing a list entry

- Want to remove `l[4]`?
`del(l[4])`
- Automatically contracts the list and shifts elements in `l[5:]` left

Supposing, I have a list and I want to remove an element from the middle of the list. So one way to do this of course, is to take the slice up to that position this slice from that position then glue them together using plus and so on. But what if I want to directly do this. It turns out that that there is a command called del. If I say `del l 4` and what it does is

effectively removes 1 4 from the current set of values, and this automatically contracts the list and shifts everything from position 5 on wards to the left by 5. So let us just verify with this works the way it claims to work.

(Refer Slide Time: 03:09)

```
>>> l = list(range(10))
>>> l
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> del(l[4])
>>> l
[0, 1, 2, 3, 5, 6, 7, 8, 9]
>>>
```

Supposing, we set our list to be the range of values from 0 to 10, 0 to 9 say, and now I say del l 4 then l becomes 0, 1, 2, 3, 5, 6, 7, 8, 9, because l 4 was the value 4 remember the position start from 0. And so it has actually deleted the value at the fourth position. This also works for dictionaries. If we want to remove the value associated with the key k then you can del d k and it will remove the key k and whatever values associated with it and removal from it. So, the key will now it considered being an undefined key.

(Refer Slide Time: 03:56)

Undefining a value

- In general, `del(x)` removes the value associated with `x`, makes `x` undefined

```
x = 7
del(x)
y = x+5

NameError: name 'x' is not defined
```

In general we can take a name like `x` and say `del x` and what this will do is make `x` undefined. Supposing, you wrote some junk code like this we set `x` equal to 7 and then we say `del x`, and then we ask `y` equal to `x` plus 5. Now this point since `x` has been undefined even though it had a value of 7 this expression `x` plus 5 cannot be calculated because `x` no longer has a value, and what you will end up with is error that we saw before namely a name error saying that the name `x` is not defined.

(Refer Slide Time: 04:29)

Checking undefined name

- Assign a value to `x` only if `x` is undefined

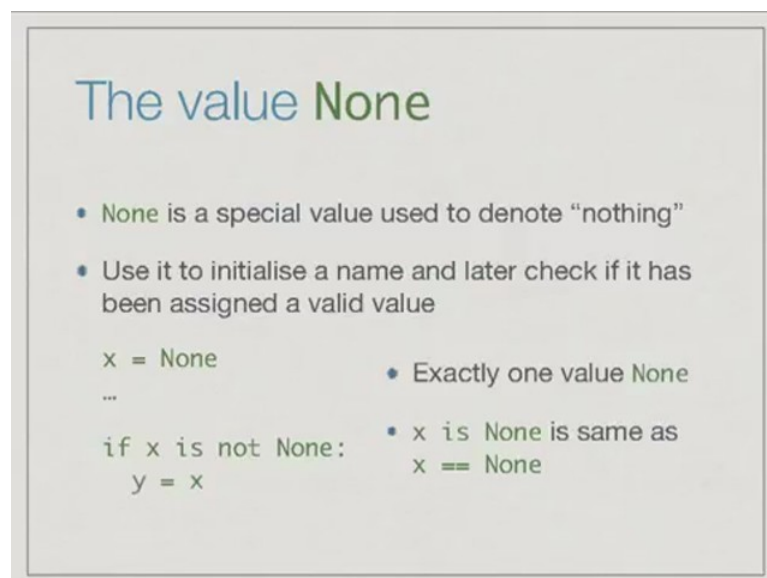
```
try:
    x
except NameError:
    x = 5
```

How would you go about checking if a name is defined, well of course you could use

exception handling. Supposing, we want to assign a value to a name `x` only if `x` is undefined, then we can say `try x` so this is trying to do something with the `x`.

Remember that names can be anything, it could be functions. So, you can just write `x` because if it is currently in name of a function which takes no arguments we will try to execute that function, so it is perfectly valid to just write `x`. But `x` if it has no value it will give a name error. So, you can say `try x` and if you happened to find a name error set it to 5 otherwise leave it untouched. If `x` already has a value this will do nothing to `x`, if `x` does not have a value then it will update the value of `x` to 5.

(Refer Slide Time: 05:15)



The slide is titled "The value None" in a blue and green font. It contains two bullet points: "None is a special value used to denote 'nothing'" and "Use it to initialise a name and later check if it has been assigned a valid value". Below these, there is a code snippet: `x = None`, followed by an ellipsis, and then `if x is not None:` with `y = x` indented. To the right of the code, there are two more bullet points: "Exactly one value None" and "x is None is same as x == None".

- None is a special value used to denote "nothing"
- Use it to initialise a name and later check if it has been assigned a valid value

```
x = None
...
if x is not None:
    y = x
```

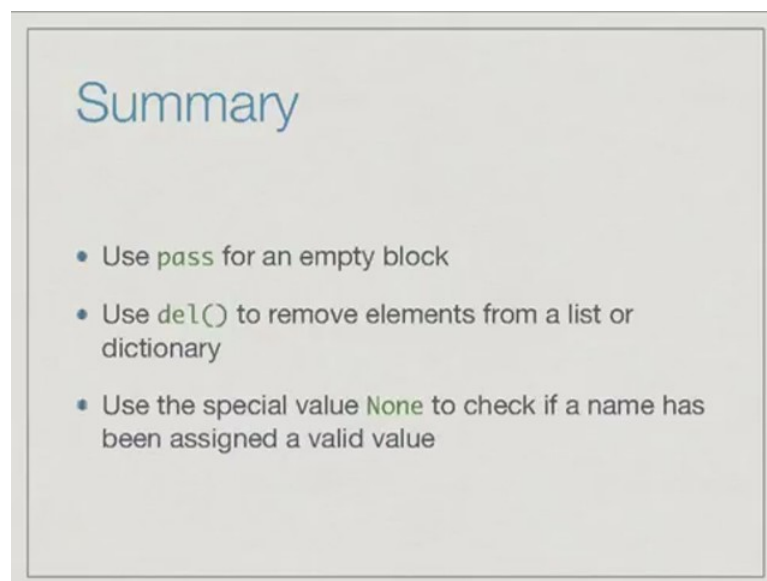
- Exactly one value None
- x is None is same as x == None

Now, usually what happens is that we want to check whether something has been defined or not so far. It is not a good idea to just leave it undefined and then use exception handling to do it, because you might actually find **strange** things happening. So, Python provides us with a special value called `None` with the capital `N`, which is the special value used to define nothing - an empty value or a null value.

We will find great use for it later on when we are defining our own data structures, but right now just think of `none` as a special value, there is only one value in `none` and it denotes nothing. So, the typical use is that when we want to check whether name has a valid value we can initialize it to `none` and later on we can check if it is still `none`. So, initially we say `x` is equal to `none` and finally we go ahead and say, if `x` is not `none` then set `y` equal to `x`, Another words `y` equal to `x` is will not be executed if `x` is still `none`.

Now, notice the peculiar word is not. So, we are using is not equal to. So there is exactly one value none in Python in the space. All **Nones** point to the same thing. Remember was checking when we say l 1 is l 2, we are checking whether l 1 and l 2 point to the same list object. We were asking whether x and the constant none point to the same none object and there is only one value none. So, x is none as the same as x equal to equal to none, so x is not none is the same as x not equal to none. So, x is not none is much easier to read the next not equal to none. That is why we will write it this say.

(Refer Slide Time: 06:58)



What we have seen our three useful things which we will use from time to time as we go **along**. One is the statement `pass` which is the special statement that does nothing and can be used whenever you need an empty block. Then we saw the command `del` which takes the value and undefined it. The most normal way to use `del` is to remove something from a list or a dictionary, you would not normally want to just undefine name which is holding simple value.

But from a dictionary or a list we might want to remove a key or if might want to remove a position and `del` is very useful for that. Finally, we have seen that there is a special value `none` which denotes a null value, it is a unique null value and this can be used to initialize variables to check whether they have been assigned sensible values **later** in the code.