

**NPTEL MOOC**

# **PROGRAMMING, DATA STRUCTURES AND ALGORITHMS IN PYTHON**

**Week 3, Lecture 2**

**Madhavan Mukund, Chennai Mathematical Institute**

**<http://www.cmi.ac.in/~madhavan>**



# Lists

- \* Lists are mutable
  - \* `list1 = [1,3,5,6]`  
`list2 = list1`  
`list1[2] = 7`
  - \* `list1` is now `[1,3,7,6]`
  - \* So is `list2`



# Lists

- \* On the other hand
  - \* `list1 = [1,3,5,6]`  
`list2 = list1`  
`list1 = list1[0:2] + [7] + list1[3:]`
  - \* `list1` is now `[1,3,7,6]`
  - \* `list2` remains `[1,3,5,6]`
- \* Concatenation produces a new list



# Extending a list

- \* Adding an element to a list, in place
  - \* `list1 = [1,3,5,6]`  
`list2 = list1`  
`list1.append(12)`
  - \* `list1` is now `[1,3,5,6,12]`
  - \* `list2` is also `[1,3,5,6,12]`



# Extending a list ...

- \* On the other hand
  - \* `list1 = [1,3,5,6]`  
`list2 = list1`  
`list1 = list1 + [12]`
  - \* `list1` is now `[1,3,5,6,12]`
  - \* `list2` remains `[1,3,5,6]`
- \* Concatenation produces a new list



# List functions

- \* `list1.append(v)` — extend `list1` by a single value `v`
- \* `list1.extend(list2)` — extend `list1` by a list of values
- \* In place equivalent of `list1 = list1 + list2`
- \* `list1.remove(x)` — removes first occurrence of `x`
- \* Error if no copy of `x` exists in `list1`



# A note on syntax

- \* `list1.append(x)` rather than `append(list1,x)`
- \* `list1` is an object
- \* `append()` is a function to update the object
- \* `x` is an argument to the function
- \* Will return to this point later



# Further list manipulation

- \* Can also assign to a slice in place
  - \* `list1 = [1,3,5,6]`  
`list2 = list1`  
`list1[2:] = [7,8]`
  - \* `list1` and `list2` are both `[1,3,7,8]`
- \* Can expand/shrink slices, but be sure you know what you are doing!
  - \* `list1[2:] = [9,10,11]` produces `[1,3,9,10,11]`
  - \* `list1[0:2] = [7]` produces `[7,9,10,11]`



# List membership

- \* `x in l` returns `True` if value `x` is found in list `l`

```
# Safely remove x from l
```

```
if x in l:
```

```
    l.remove(x)
```

```
# Remove all occurrences of x from l
```

```
while x in l:
```

```
    l.remove(x)
```



# Other functions

- \* `l.reverse()` — reverse `l` in place
- \* `l.sort()` — sort `l` in ascending order
- \* `l.index(x)` — find leftmost position of `x` in `l`
  - \* Avoid error by checking if `x` in `l`
- \* `l.rindex(x)` — find rightmost position of `x` in `l`
- \* Many more ... see Python documentation!



# Initialising names

- \* A name cannot be used before it is assigned a value

```
y = x + 1 # Error if x is unassigned
```

- \* May forget this for lists where update is implicit

```
l.append(v)
```

- \* Python needs to know that `l` is a list



# Initialising names ...

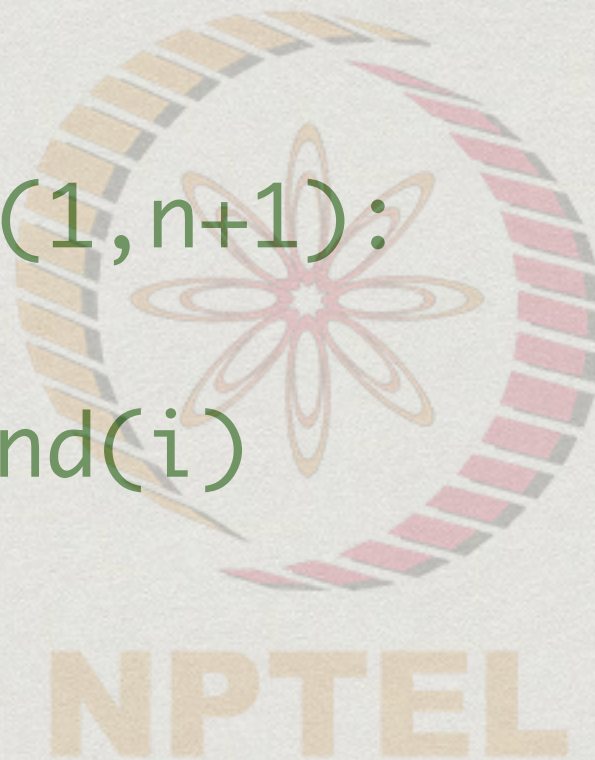
```
def factors(n):
```

```
    for i in range(1,n+1):
```

```
        if n%i == 0:
```

```
            flist.append(i)
```

```
    return(flist)
```





# Initialising names ...

```
def factors(n):
```

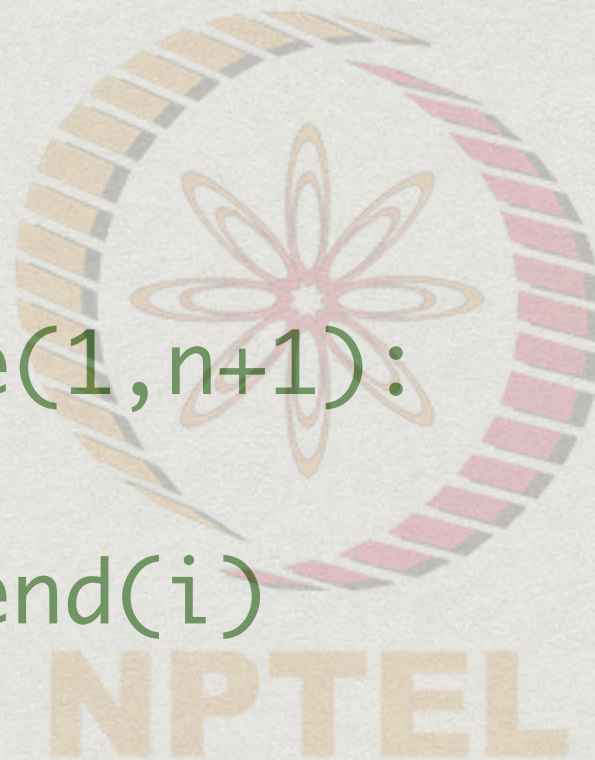
```
    flist = []
```

```
    for i in range(1,n+1):
```

```
        if n%i == 0:
```

```
            flist.append(i)
```

```
    return(flist)
```





# Summary

- \* To extend lists in place, use `l.append()`, `l.extend()`
- \* Can also assign new value, in place, to a slice
- \* Many built in functions for lists — see documentation
- \* Don't forget to assign a value to a name before it is first used