More Python:
f-strings, list comprehension

String formatting- ”old” style, Python 3+

.format() method on a string object

```
#Example 1

fname = 'John'
sname = 'Smith'
print('Hello, {} {}!'.format(fname, sname))


# Example 2
print("The price is {:.2f} euros!".format(27))
```

String formatting- ”old” style, Python 3+

```
# Example 3
capitals = {'Afghanistan':'Kabul',
            'Albania': 'Tirana',
            'Algeria':'Algiers',
            'Andorra': 'Andorra la Vella'}

for country in capitals:
    print('{:>15} {:>20}'.format(country, capitals[country]))
```

Lots more options, see documentation!

String formatting new style, Python 3.7+

f-strings (”string interpolation)

Allows you to use embedded Python expressions inside string constants

```
name = 'John'
print(f'Hello, {name}!')
```

```
fname = 'John'
sname = 'Smith'
print(f'Hello, {fname} {sname}!')
```

f-strings

```
price = 21.7865

print(price)
print(f'Price is {price:.2f}')
print(f'Price is {price:>10.2f}')
```

- .2f – floating point number, 2 digits after dec point
- >10 – take 10 spaces, alight right

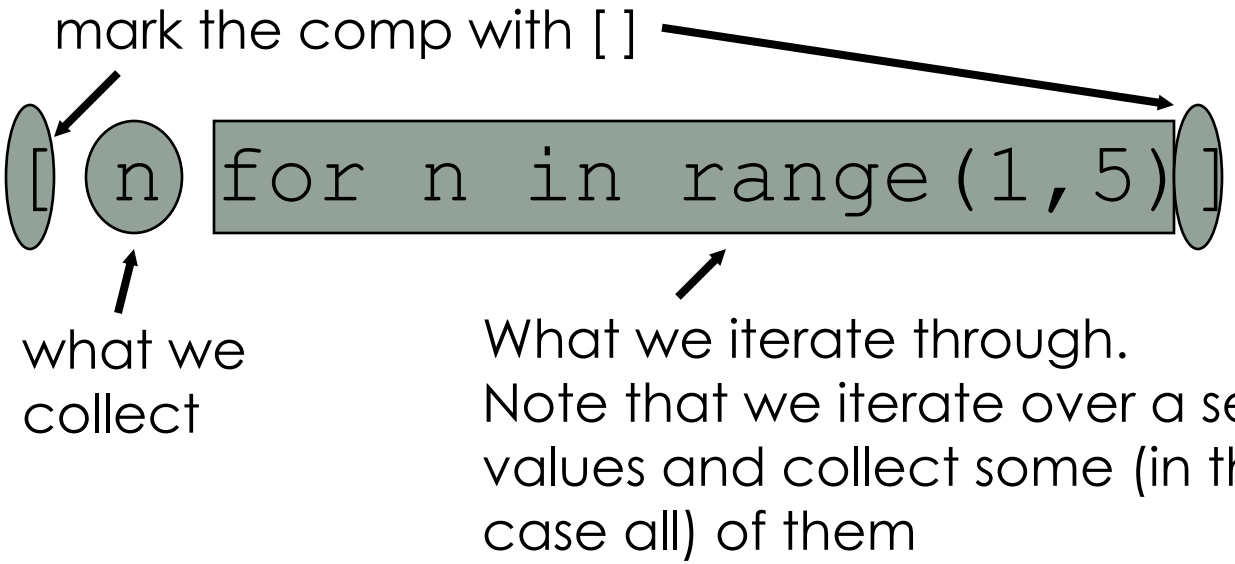
List comprehension

Comprehension refers to constructing a new collection by performing some operation on the elements of another collection

[expression for-clause condition]

Constructing lists using comprehension

```
[n for n in range(1,5)]
```



returns [1,2,3,4]

Constructing lists using comprehension

```
[ n**2 for n in range(1,6)]
```

returns [1,4,9,16,25]

Note that we can only change the values we are iterating over, in this case n

Multiple collects

[x+y for x in range(1,4) for y in range (1,4)]

It is the same as the following:

```
my_list = [ ]
for x in range (1,4):
    for y in range (1,4):
        my_list.append(x+y)
```

$\Rightarrow$ [2,3,4,3,4,5,4,5,6]

Modifying what gets collected

```
[c for c in "Hi There Mom" if c.isupper()]
```

The if part of the comprehensive controls which of the iterated values is collected at the end.

Only those values which make the if part true will be collected

→ ['H','T','M']

```
[i    for i in range(1,7)      if i%2==0]
```

→ [2,4,6]

List comprehension - conclusion

- An elegant way to create lists based on existing lists
- In general more compact and faster loops
- Avoid too long list comprehensions to ensure that code is user-friendly
- Every list comprehension can be rewritten as a for-loop, but not every for-loop can be rewritten as a list comprehension.