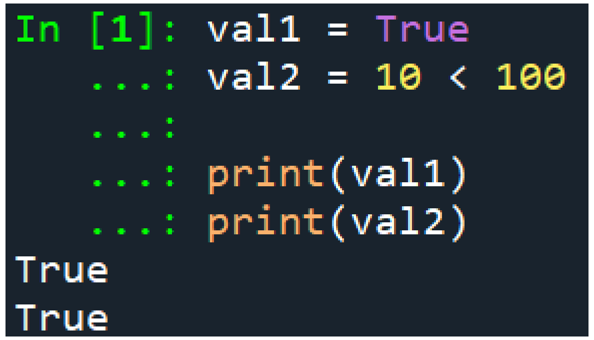
Lecture 4: Loops and Logic

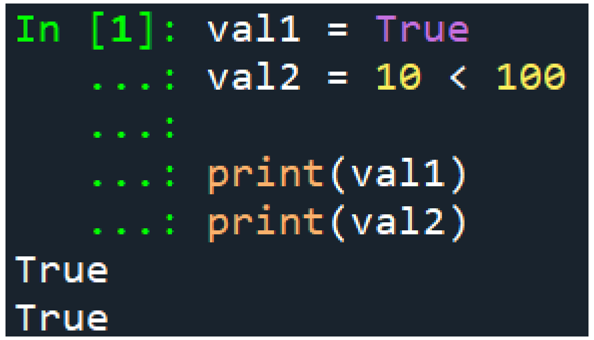
Round and round we go!

First, recall Booleans



```
In [1]: val1 = True
   ...: val2 = 10 < 100
   ...:
   ...: print(val1)
   ...: print(val2)
True
True
```

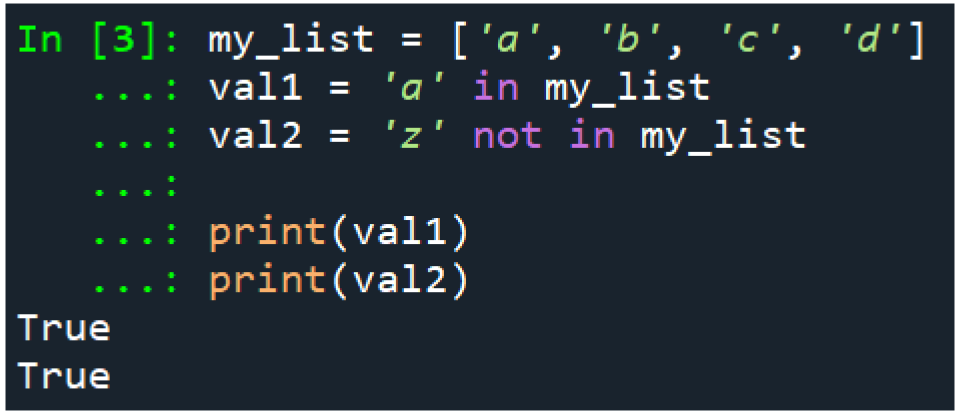
First, recall Booleans



```
In [1]: val1 = True
   ...: val2 = 10 < 100
   ...:
   ...: print(val1)
   ...: print(val2)
True
True
```

==, !=, >, <, >=, <=

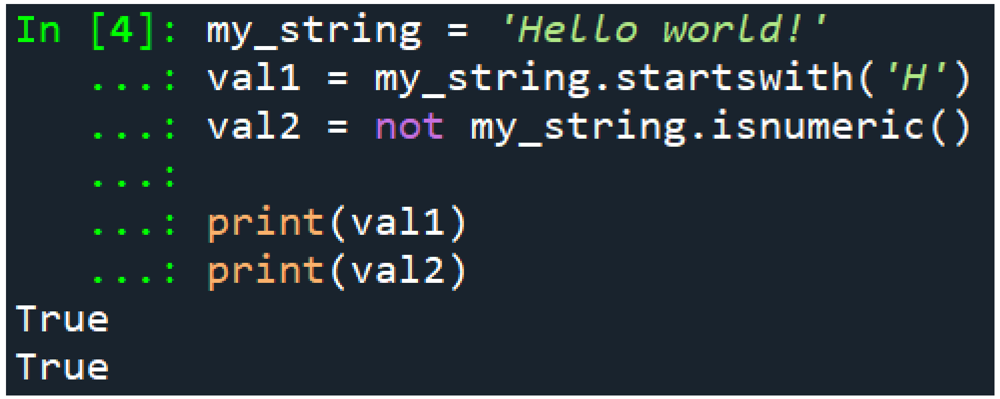
First, recall Booleans



```
In [3]: my_list = ['a', 'b', 'c', 'd']
   ...: val1 = 'a' in my_list
   ...: val2 = 'z' not in my_list
   ...:
   ...: print(val1)
   ...: print(val2)
True
True
```

in, not

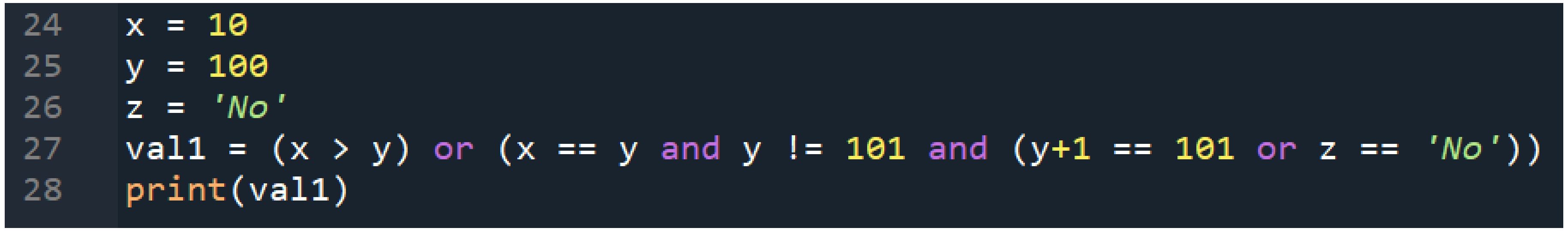
First, recall Booleans



```
In [4]: my_string = 'Hello world!'
   ...: val1 = my_string.startswith('H')
   ...: val2 = not my_string.isnumeric()
   ...:
   ...: print(val1)
   ...: print(val2)
True
True
```

Some special string methods

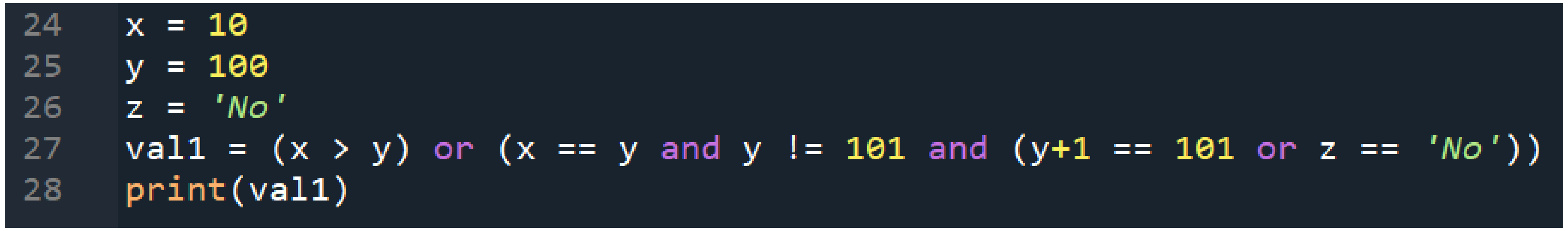
Complex chains of logic



```python
x = 10
y = 100
z = 'No'
val1 = (x > y) or (x == y and y != 101 and (y+1 == 101 or z == 'No'))
print(val1)
```

$$\underbrace{(x > y)}\ \text{or}\ (\underbrace{x == y}\ \text{and}\ \underbrace{y \mathrel{!=} 101}\ \text{and}\ (\underbrace{y+1 == 101}\ \text{or}\ \underbrace{z == \text{‘No’}}))$$

Complex chains of logic

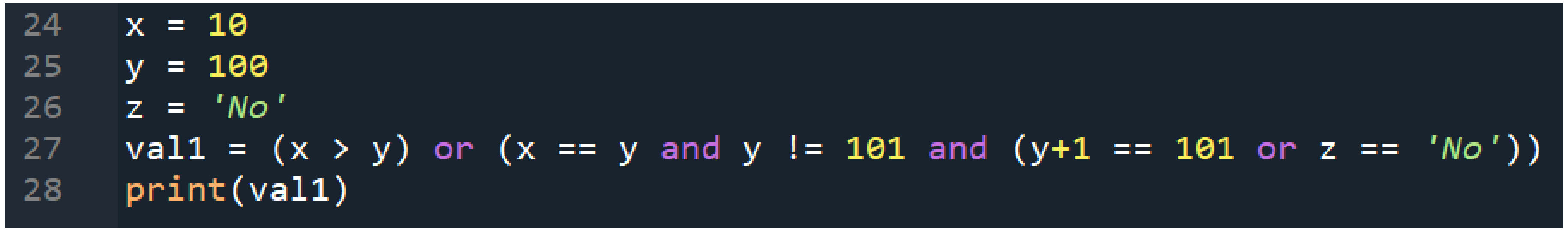


```
x = 10
y = 100
z = 'No'
val1 = (x > y) or (x == y and y != 101 and (y+1 == 101 or z == 'No'))
print(val1)
```

$$\underbrace{(x > y)}_{}\ \text{or}\ (\underbrace{x == y}_{}\ \text{and}\ \underbrace{y\ !=\ 101}_{}\ \text{and}\ (\underbrace{y+1 == 101}_{}\ \text{or}\ \underbrace{z == \text{‘No’}}_{}))$$

$$\underbrace{(\text{False})}_{}\ \text{or}\ (\ \underbrace{\text{False}}_{}\ \text{and}\ \ \underbrace{\text{True}}_{}\ \ \text{and}\ (\ \underbrace{\text{True}\quad \text{or}\quad \text{True}}_{}))$$

Complex chains of logic



```
x = 10
y = 100
z = 'No'
val1 = (x > y) or (x == y and y != 101 and (y+1 == 101 or z == 'No'))
print(val1)
```

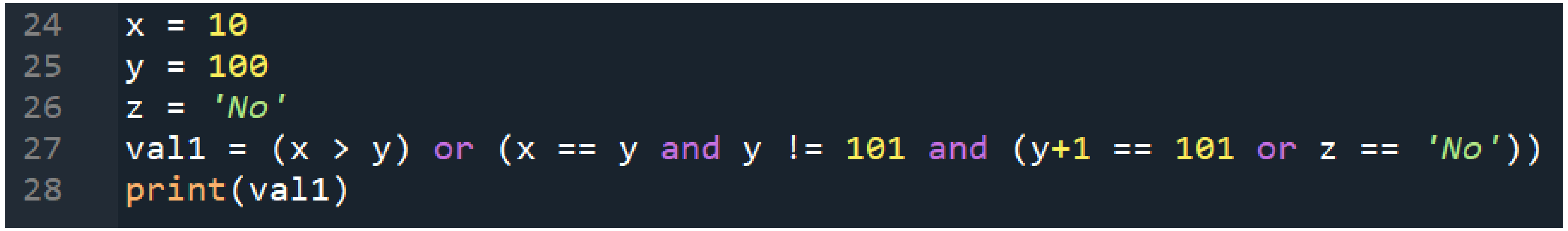
(x > y) or (x == y and y != 101 and (y+1 == 101 or z == ‘No’))

(False) or (False and True and (True or True))

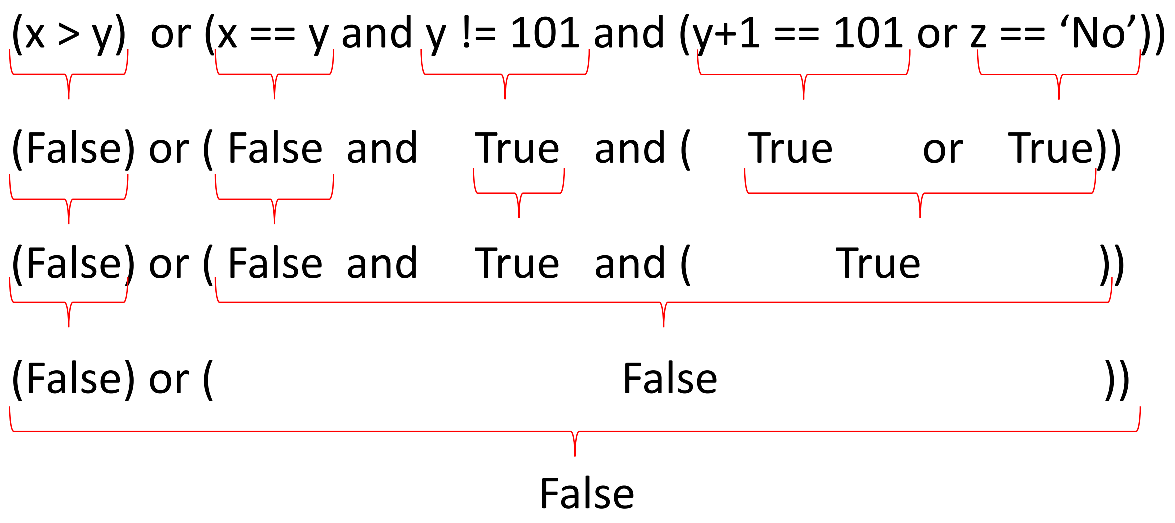
(False) or (False and True and (True))

(False) or (False))

Complex chains of logic

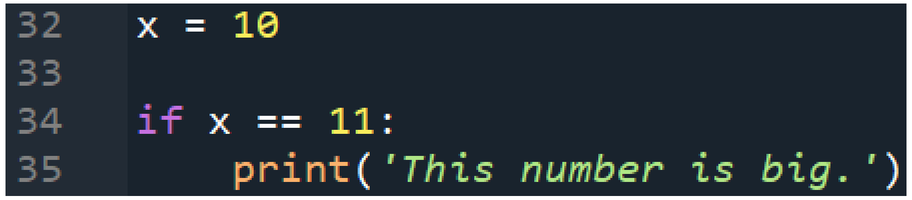


```python
x = 10
y = 100
z = 'No'
val1 = (x > y) or (x == y and y != 101 and (y+1 == 101 or z == 'No'))
print(val1)
```



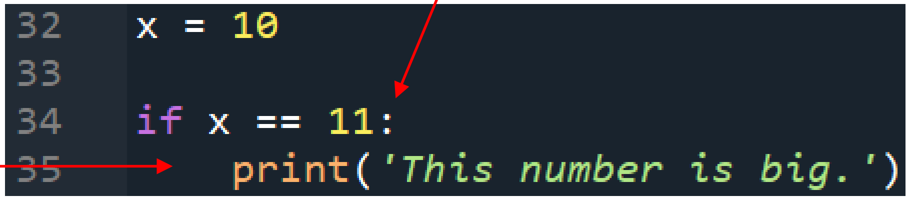
(x > y) or (x == y and y != 101 and (y+1 == 101 or z == ‘No’))
(False) or (False and True and (True or True))
(False) or (False and True and (True))
(False) or (False))
False

The “if” statement block



```python
x = 10

if x == 11:
    print('This number is big.')
```

The “if” statement block

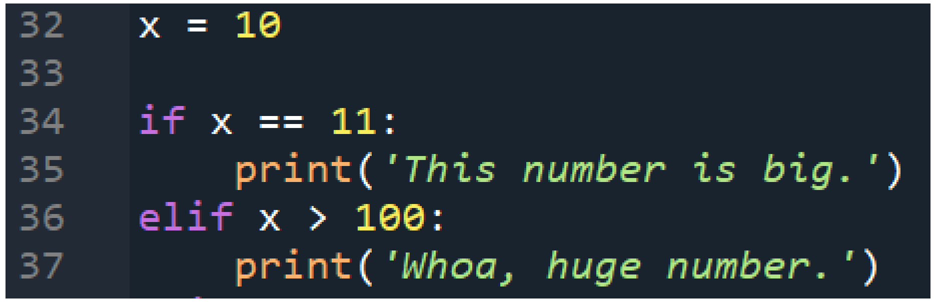


Colon at end of line

Mandatory:
one tab/four spaces

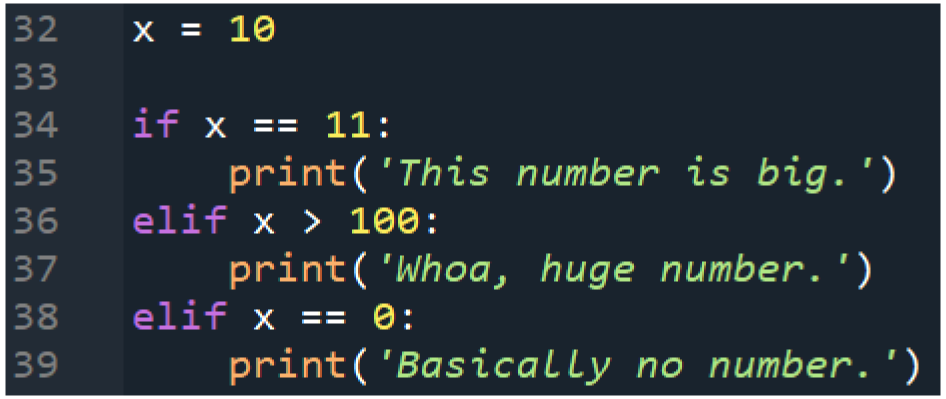
```python
x = 10

if x == 11:
    print('This number is big.')
```

The “if” statement block



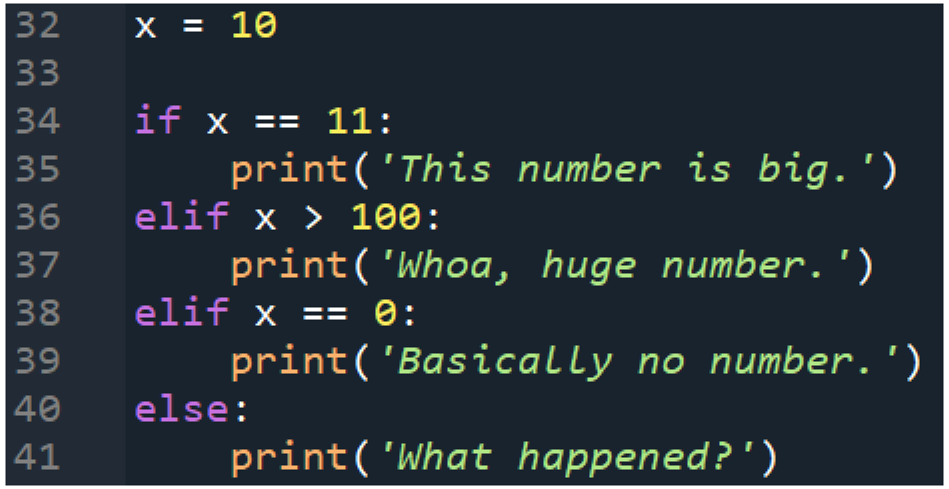
```python
x = 10

if x == 11:
    print('This number is big.')
elif x > 100:
    print('Whoa, huge number.')
```

The “if” statement block



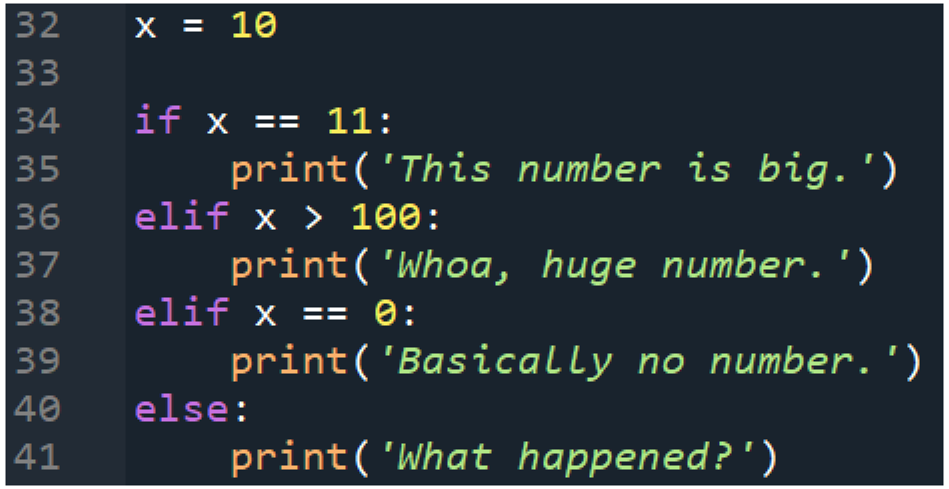
```python
x = 10

if x == 11:
    print('This number is big.')
elif x > 100:
    print('Whoa, huge number.')
elif x == 0:
    print('Basically no number.')
```

The “if” statement block



```python
x = 10

if x == 11:
    print('This number is big.')
elif x > 100:
    print('Whoa, huge number.')
elif x == 0:
    print('Basically no number.')
else:
    print('What happened?')
```

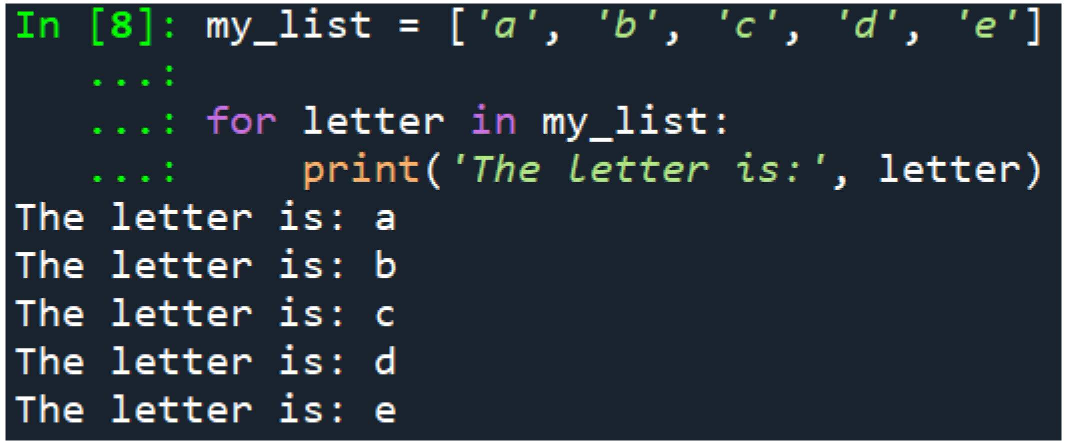
The “if” statement block



```python
x = 10

if x == 11:
    print('This number is big.')
elif x > 100:
    print('Whoa, huge number.')
elif x == 0:
    print('Basically no number.')
else:
    print('What happened?')
```

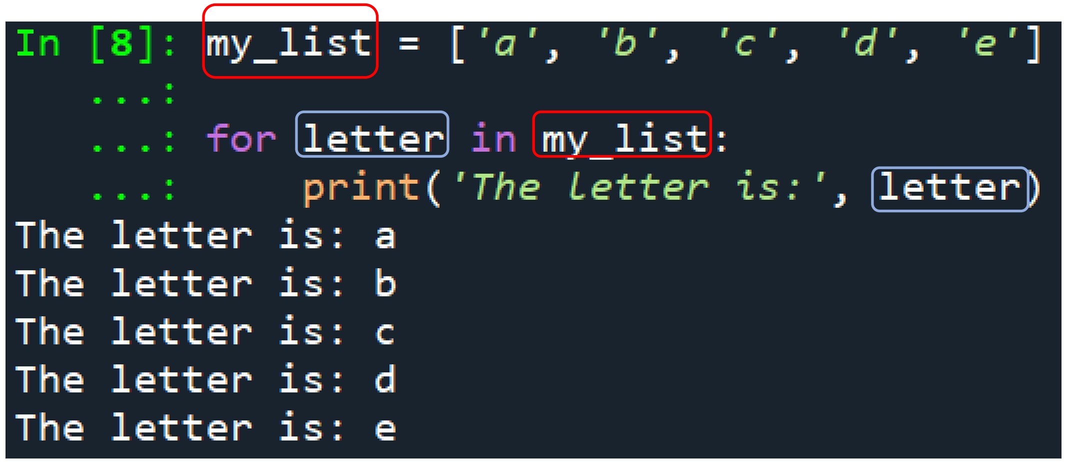
1 “if” statement
0-N “elif” statements
0-1 “else” statements

“For” loops



```
In [8]: my_list = ['a', 'b', 'c', 'd', 'e']
   ...:
   ...: for letter in my_list:
   ...:     print('The letter is:', letter)
The letter is: a
The letter is: b
The letter is: c
The letter is: d
The letter is: e
```

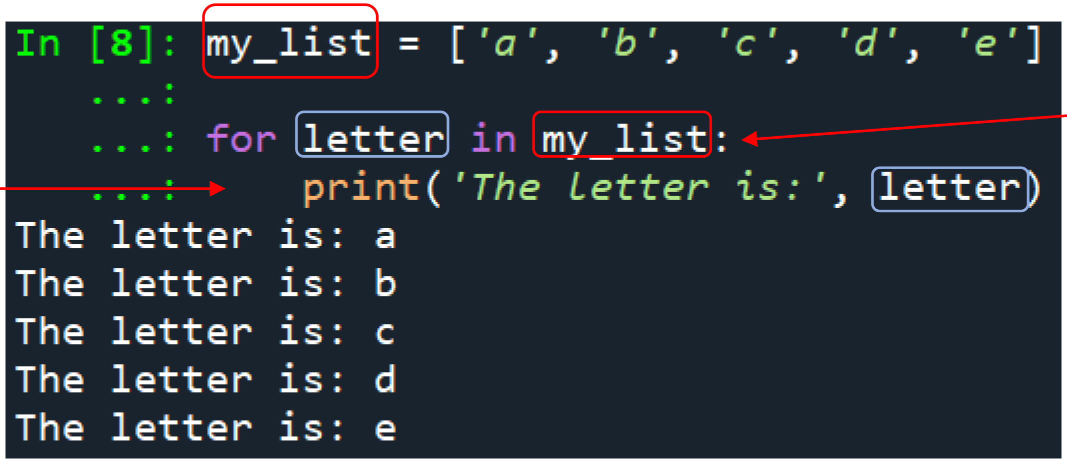
“For” loops



```
In [8]: my_list = ['a', 'b', 'c', 'd', 'e']
   ...:
   ...: for letter in my_list:
   ...:     print('The letter is:', letter)
The letter is: a
The letter is: b
The letter is: c
The letter is: d
The letter is: e
```

“For” loops

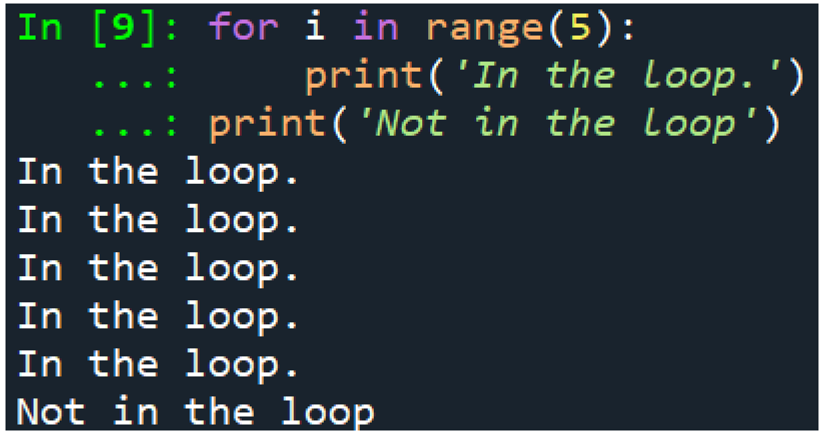
Same as “if” code block: mandatory indent



```
In [8]: my_list = ['a', 'b', 'c', 'd', 'e']
   ...:
   ...: for letter in my_list:
   ...:     print('The letter is:', letter)
The letter is: a
The letter is: b
The letter is: c
The letter is: d
The letter is: e
```

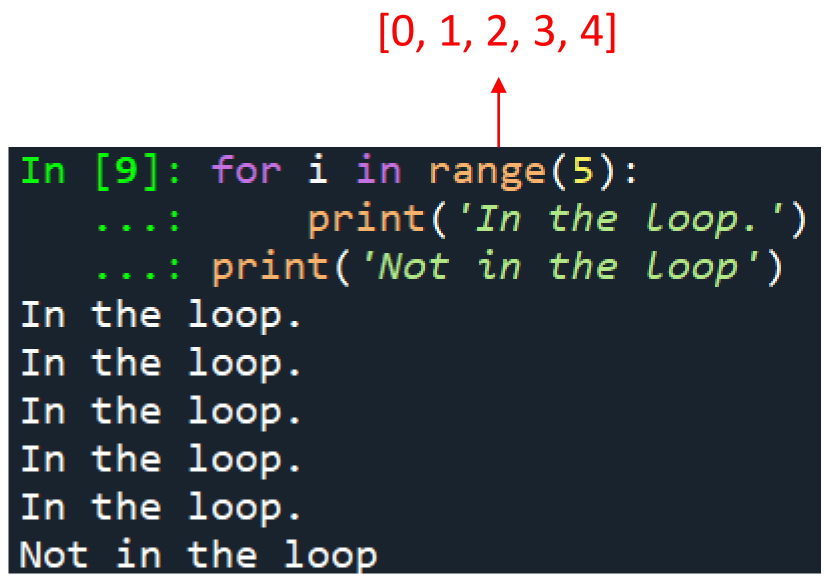
Same as “if” code block: mandatory colon

“For” loops



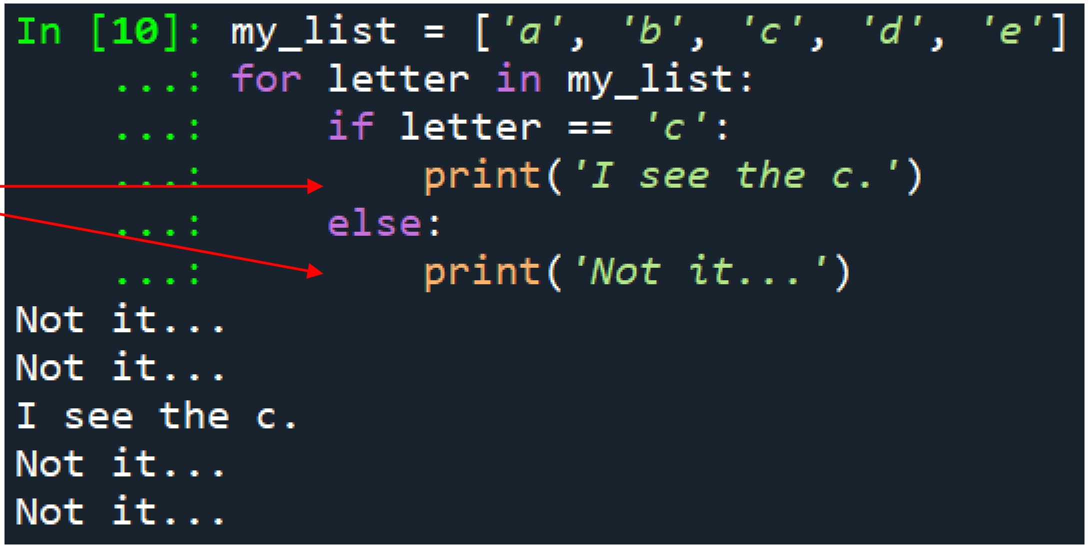
```
In [9]: for i in range(5):
   ...:     print('In the loop.')
   ...: print('Not in the loop')
In the loop.
In the loop.
In the loop.
In the loop.
In the loop.
Not in the loop
```

“For” loops



```
In [9]: for i in range(5):
   ...:     print('In the loop.')
   ...: print('Not in the loop')
In the loop.
In the loop.
In the loop.
In the loop.
In the loop.
Not in the loop
```

Combing “for” and “if”



```
In [10]: my_list = ['a', 'b', 'c', 'd', 'e']
    ...: for letter in my_list:
    ...:     if letter == 'c':
    ...:         print('I see the c.')
    ...:     else:
    ...:         print('Not it...')
Not it...
Not it...
I see the c.
Not it...
Not it...
```

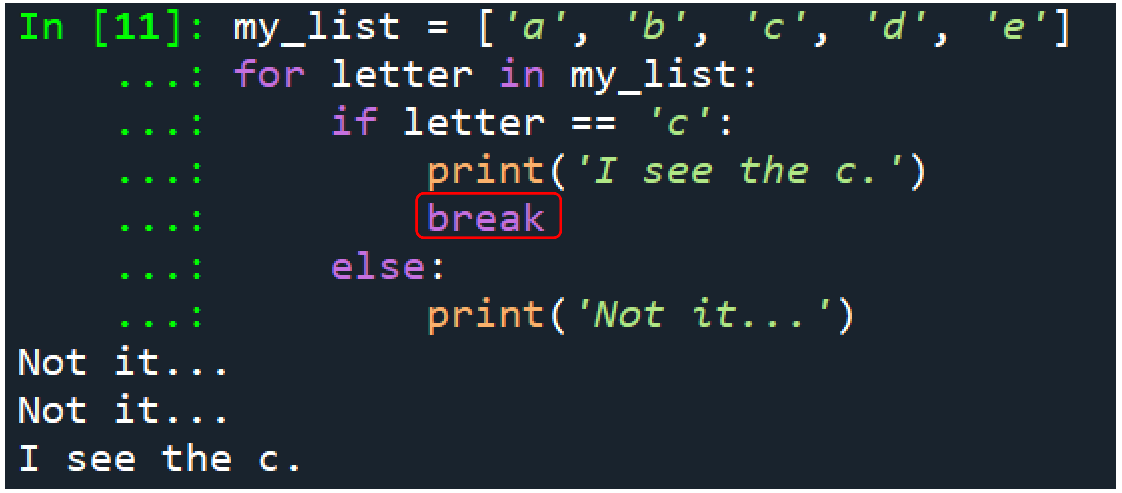
Two levels
of indenting

More control over your loops

“break”: Immediately exits the current loop
“continue”: Immediately goes to the next iteration of the current loop

More control over your loops

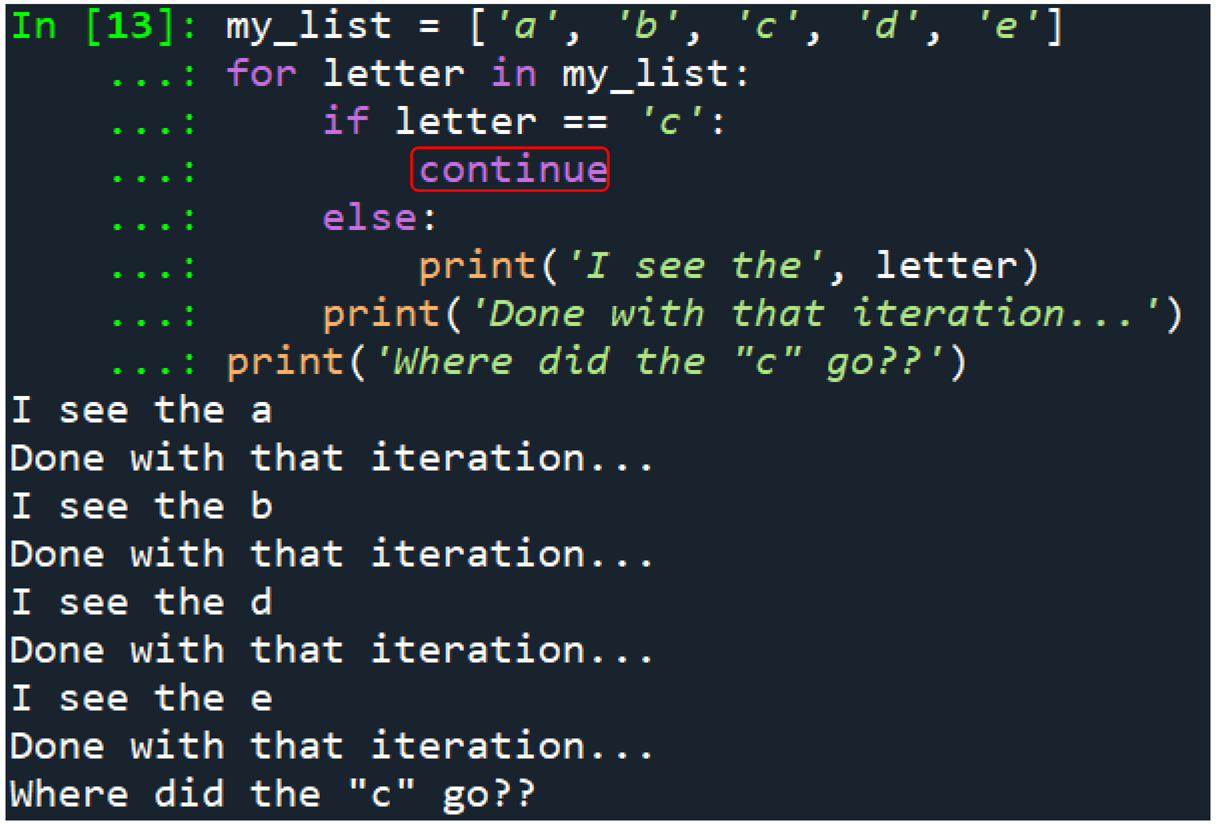
“break”: Immediately exits the current loop
“continue”: Immediately goes to the next iteration of the current loop



```
In [11]: my_list = ['a', 'b', 'c', 'd', 'e']
    ...: for letter in my_list:
    ...:     if letter == 'c':
    ...:         print('I see the c.')
    ...:         break
    ...:     else:
    ...:         print('Not it...')
Not it...
Not it...
I see the c.
```

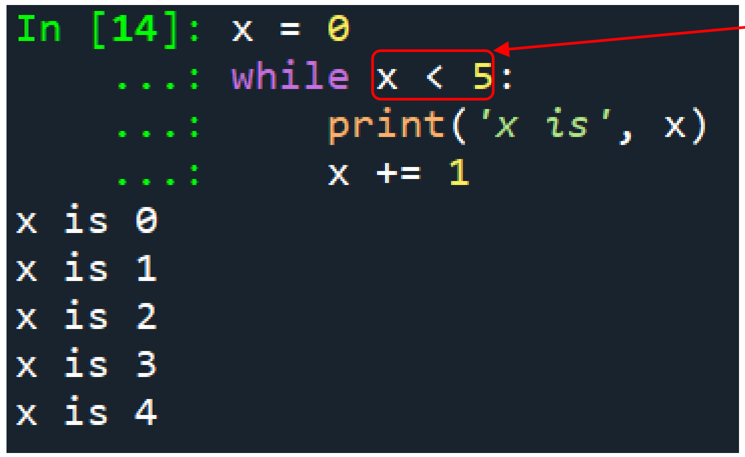
More control over your loops

“break”: Immediately exits the current loop
“continue”: Immediately goes to the next iteration of the current loop



```
In [13]: my_list = ['a', 'b', 'c', 'd', 'e']
    ...: for letter in my_list:
    ...:     if letter == 'c':
    ...:         continue
    ...:     else:
    ...:         print('I see the', letter)
    ...:     print('Done with that iteration...')
    ...: print('Where did the "c" go??')
I see the a
Done with that iteration...
I see the b
Done with that iteration...
I see the d
Done with that iteration...
I see the e
Done with that iteration...
Where did the "c" go??
```

The “while” loop



```
In [14]: x = 0
    ...: while x < 5:
    ...:     print('x is', x)
    ...:     x += 1
x is 0
x is 1
x is 2
x is 3
x is 4
```

Continues as long as expression is True

List comprehensions

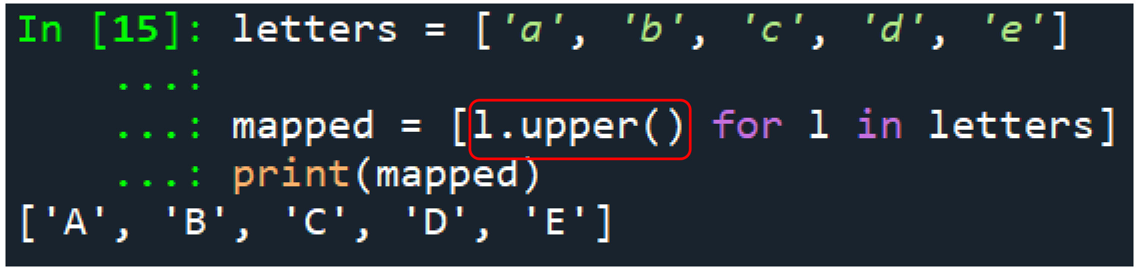
When you're iterating over a list(s), and the desired result is a different list.

```
[f(v) for v in starting_list]
[f(v) for v in starting_list if <condition>]
```

List comprehensions

When you’re iterating over a list(s), and the desired result is a different list.

[f(v) for v in starting_list]
[f(v) for v in starting_list if <condition>]

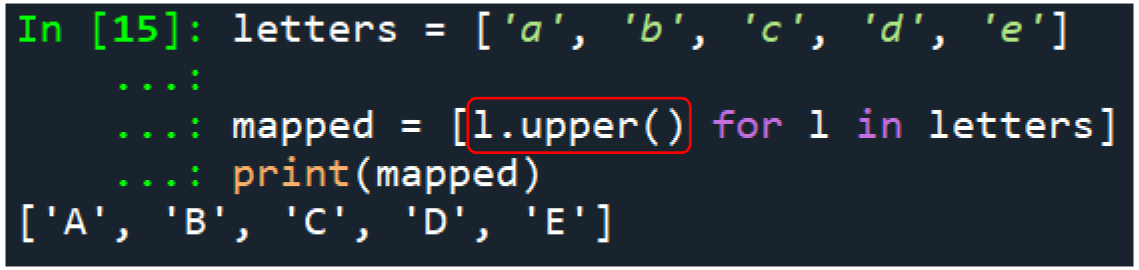


```
In [15]: letters = ['a', 'b', 'c', 'd', 'e']
    ...:
    ...: mapped = [l.upper() for l in letters]
    ...: print(mapped)
['A', 'B', 'C', 'D', 'E']
```

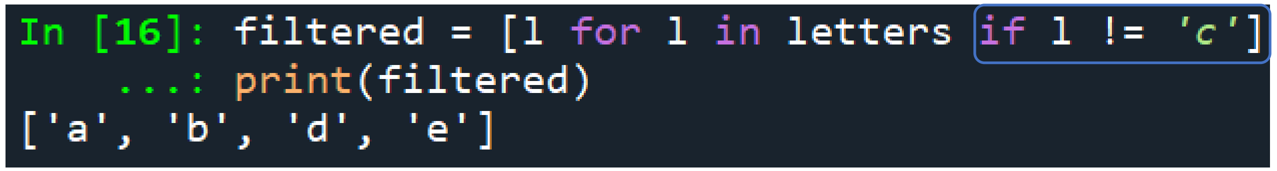
List comprehensions

When you're iterating over a list(s), and the desired result is a different list.

[f(v) for v in starting_list]
[f(v) for v in starting_list if <condition>]



```
In [15]: letters = ['a', 'b', 'c', 'd', 'e']
    ...:
    ...: mapped = [l.upper() for l in letters]
    ...: print(mapped)
['A', 'B', 'C', 'D', 'E']
```



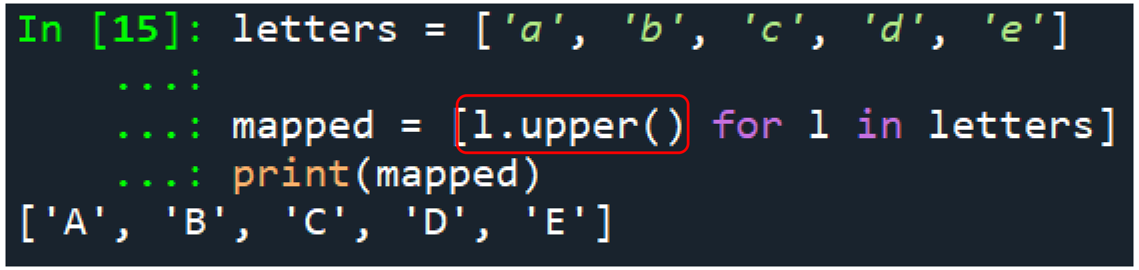
```
In [16]: filtered = [l for l in letters if l != 'c']
    ...: print(filtered)
['a', 'b', 'd', 'e']
```

List comprehensions

When you're iterating over a list(s), and the desired result is a different list.

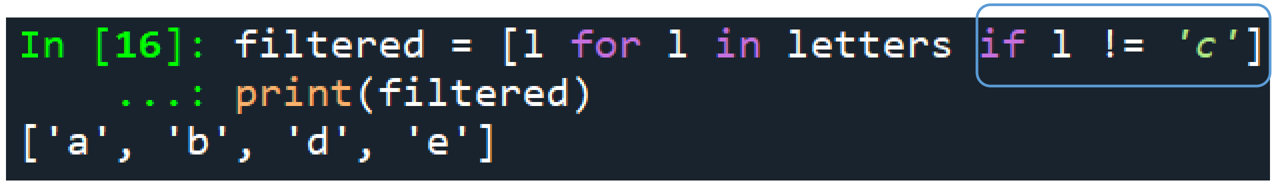
[f(v) for v in starting_list]
[f(v) for v in starting_list if <condition>]

Mapping



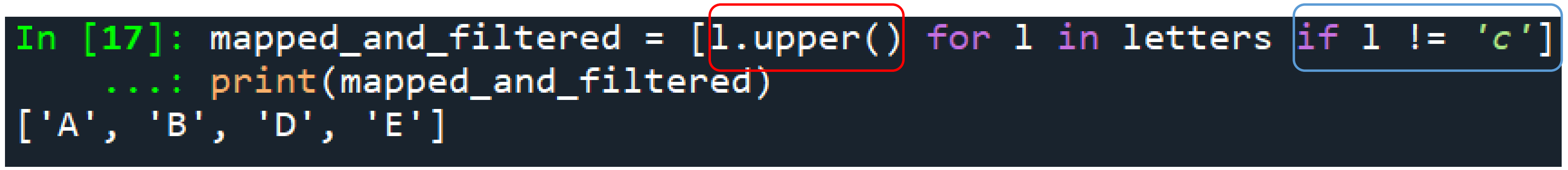
```
In [15]: letters = ['a', 'b', 'c', 'd', 'e']
    ...:
    ...: mapped = [l.upper() for l in letters]
    ...: print(mapped)
['A', 'B', 'C', 'D', 'E']
```

Filtering



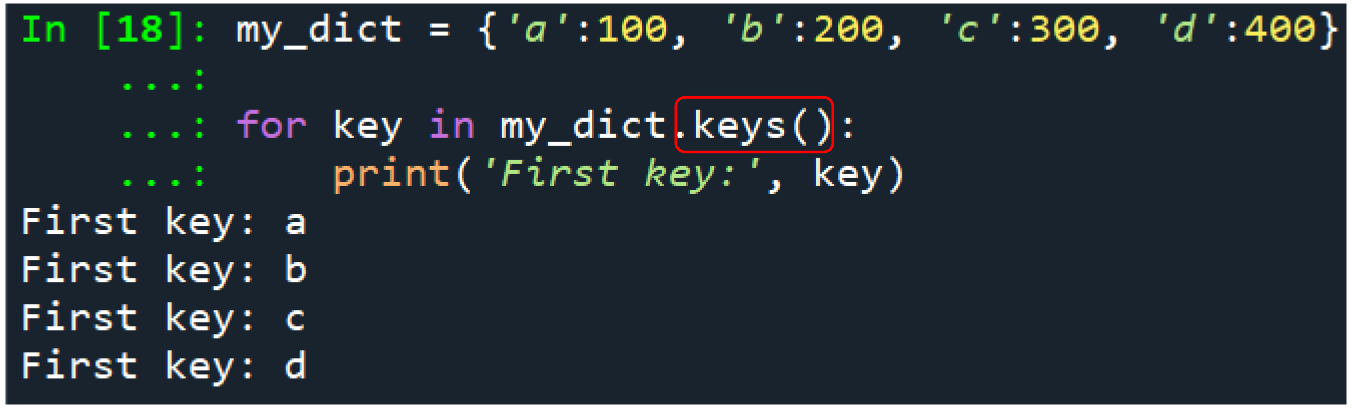
```
In [16]: filtered = [l for l in letters if l != 'c']
    ...: print(filtered)
['a', 'b', 'd', 'e']
```

Mapping and Filtering

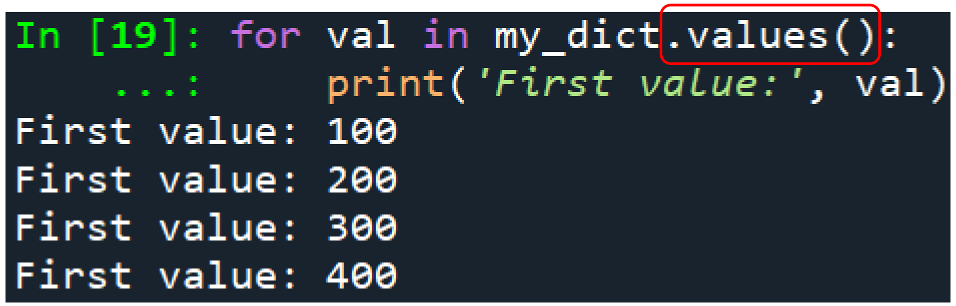


```
In [17]: mapped_and_filtered = [l.upper() for l in letters if l != 'c']
    ...: print(mapped_and_filtered)
['A', 'B', 'D', 'E']
```

Iterating over dictionaries

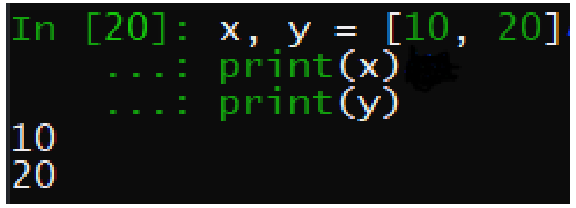


```
In [18]: my_dict = {'a':100, 'b':200, 'c':300, 'd':400}
    ...:
    ...: for key in my_dict.keys():
    ...:     print('First key:', key)
First key: a
First key: b
First key: c
First key: d
```



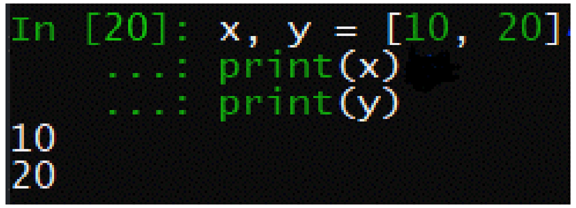
```
In [19]: for val in my_dict.values():
    ...:     print('First value:', val)
First value: 100
First value: 200
First value: 300
First value: 400
```

Interlude: unpacking notation

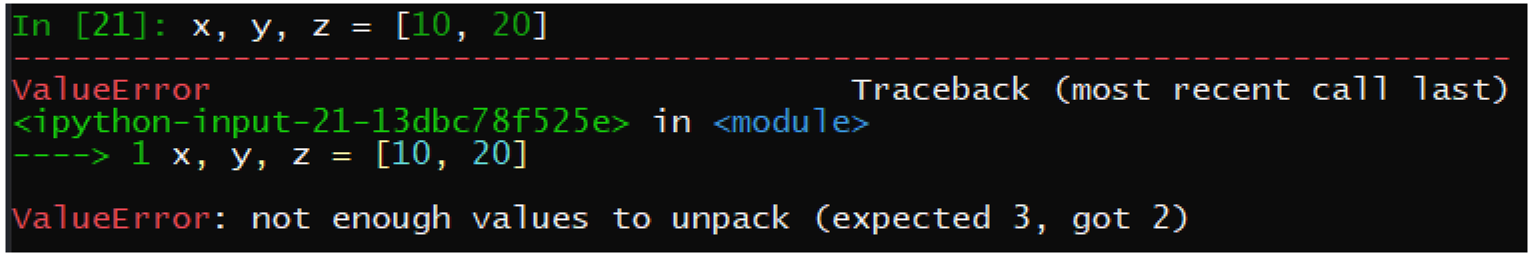


```
In [20]: x, y = [10, 20]
    ...: print(x)
    ...: print(y)
10
20
```

Interlude: unpacking notation

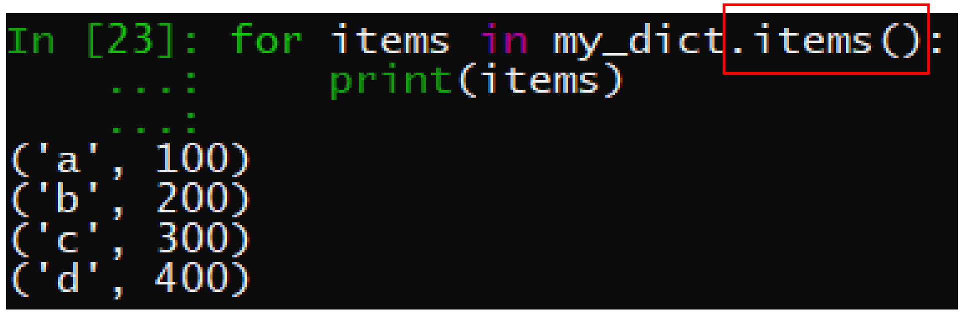


```
In [20]: x, y = [10, 20]
    ...: print(x)
    ...: print(y)
10
20
```



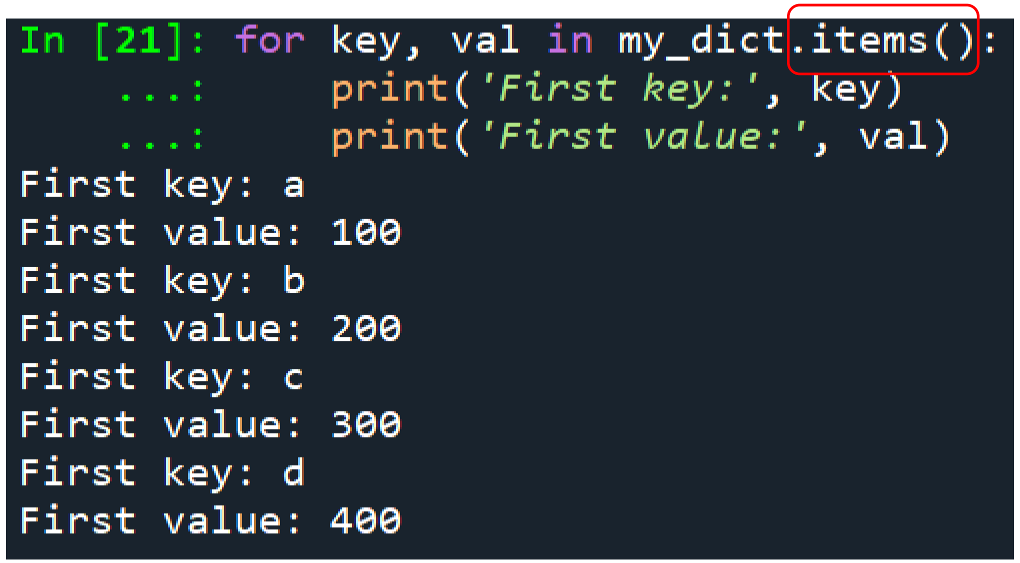
```
In [21]: x, y, z = [10, 20]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-21-13dbc78f525e> in <module>
----> 1 x, y, z = [10, 20]

ValueError: not enough values to unpack (expected 3, got 2)
```

Iterating over dictionaries



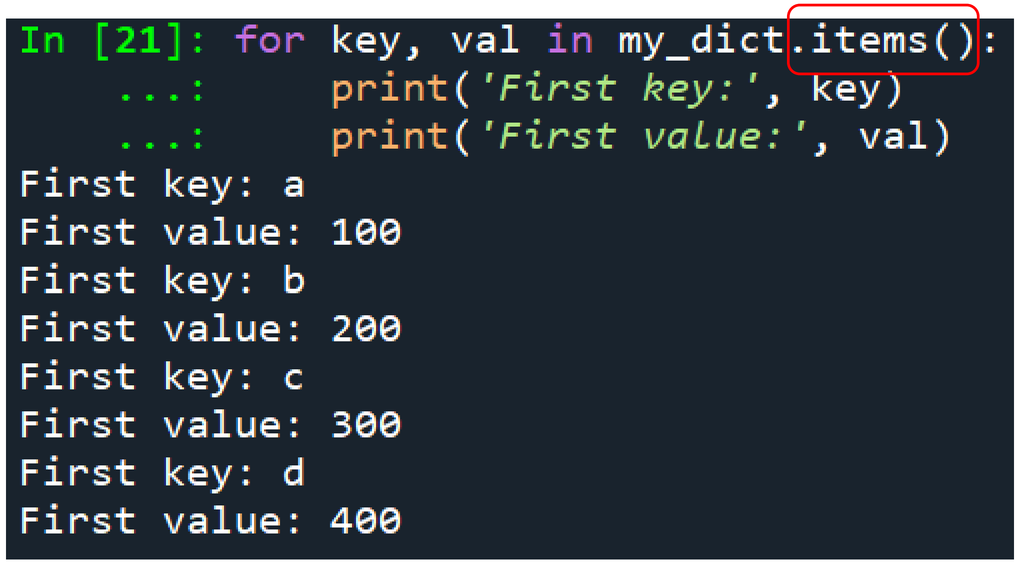
```
In [23]: for items in my_dict.items():
    ...:     print(items)
    ...:
('a', 100)
('b', 200)
('c', 300)
('d', 400)
```

Iterating over dictionaries

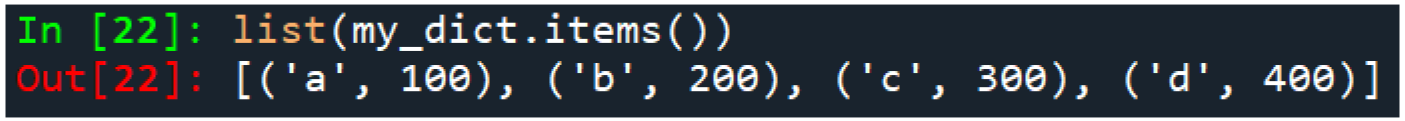


```
In [21]: for key, val in my_dict.items():
    ...:     print('First key:', key)
    ...:     print('First value:', val)
First key: a
First value: 100
First key: b
First value: 200
First key: c
First value: 300
First key: d
First value: 400
```

Iterating over dictionaries



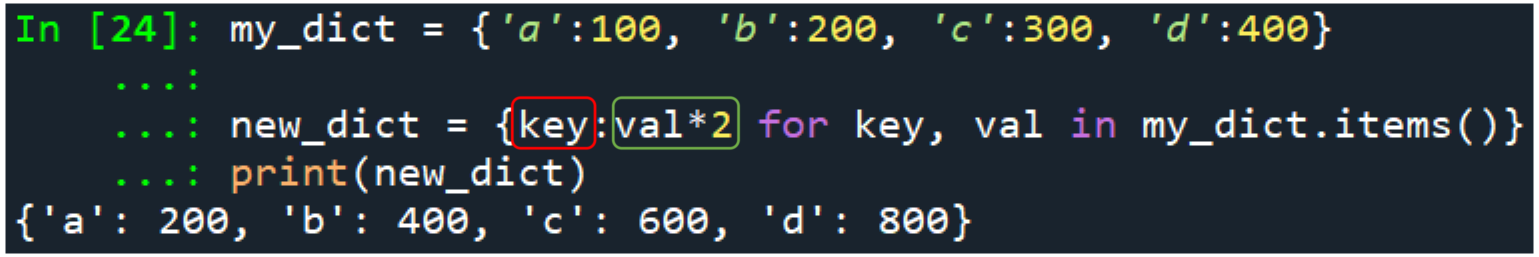
```
In [21]: for key, val in my_dict.items():
    ...:     print('First key:', key)
    ...:     print('First value:', val)
First key: a
First value: 100
First key: b
First value: 200
First key: c
First value: 300
First key: d
First value: 400
```



```
In [22]: list(my_dict.items())
Out[22]: [('a', 100), ('b', 200), ('c', 300), ('d', 400)]
```

Dictionary comprehensions

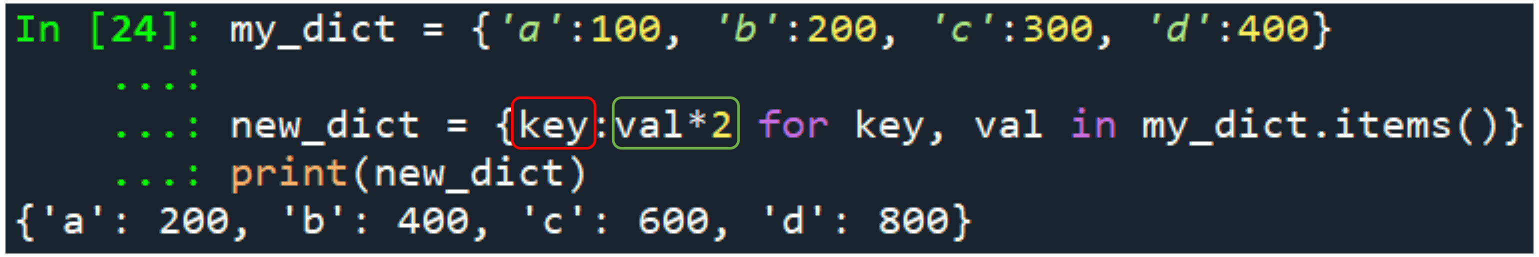
{f(key):f(val) for key, val in my_dict.items()}



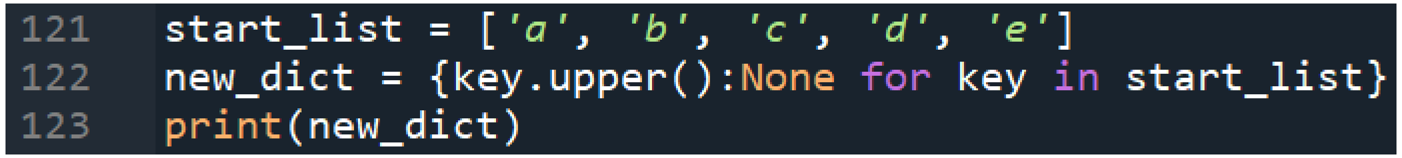
```
In [24]: my_dict = {'a':100, 'b':200, 'c':300, 'd':400}
    ...:
    ...: new_dict = {key:val*2 for key, val in my_dict.items()}
    ...: print(new_dict)
{'a': 200, 'b': 400, 'c': 600, 'd': 800}
```

Dictionary comprehensions

{f(key):f(val) for key, val in my_dict.items()}



```
In [24]: my_dict = {'a':100, 'b':200, 'c':300, 'd':400}
    ...:
    ...: new_dict = {key:val*2 for key, val in my_dict.items()}
    ...: print(new_dict)
{'a': 200, 'b': 400, 'c': 600, 'd': 800}
```

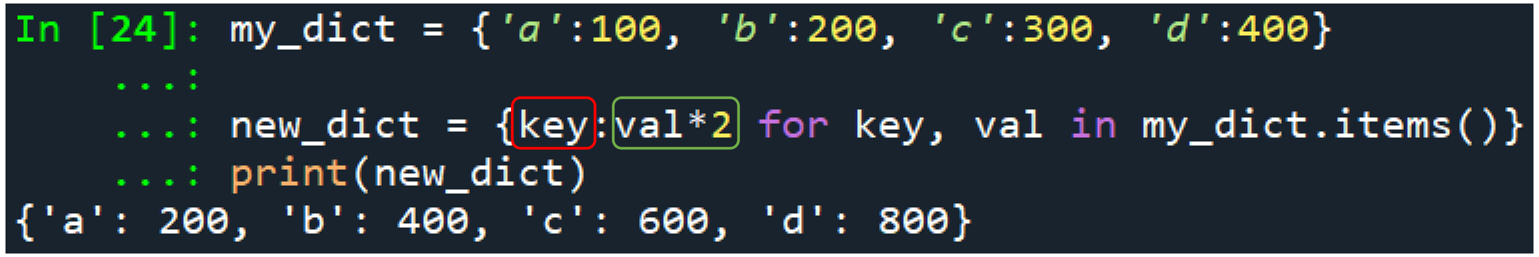


```
start_list = ['a', 'b', 'c', 'd', 'e']
new_dict = {key.upper():None for key in start_list}
print(new_dict)
```

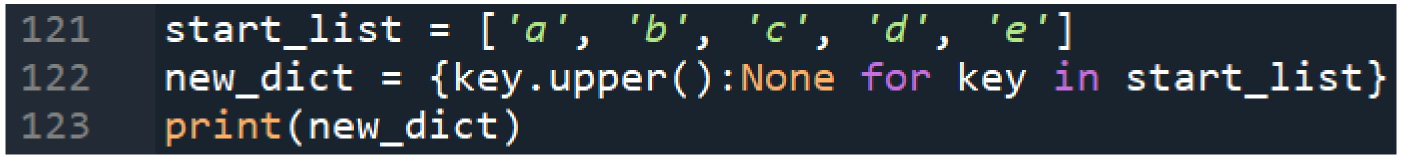
What will this do?

Dictionary comprehensions

{f(key):f(val) for key, val in my_dict.items()}

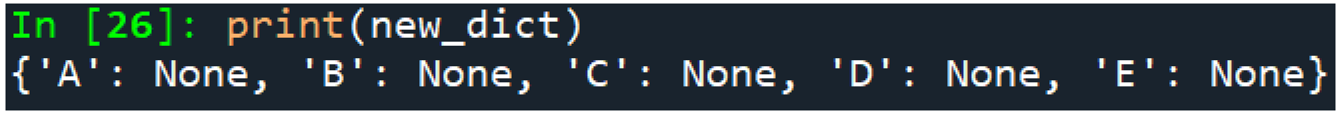


```
In [24]: my_dict = {'a':100, 'b':200, 'c':300, 'd':400}
    ...:
    ...: new_dict = {key:val*2 for key, val in my_dict.items()}
    ...: print(new_dict)
{'a': 200, 'b': 400, 'c': 600, 'd': 800}
```



```
start_list = ['a', 'b', 'c', 'd', 'e']
new_dict = {key.upper():None for key in start_list}
print(new_dict)
```

What will this do?



```
In [26]: print(new_dict)
{'A': None, 'B': None, 'C': None, 'D': None, 'E': None}
```