ENGINEERS WITH
SOCIAL RESPONSIBILITY

Programming Lab

Autumn Semester

Course code: PC503

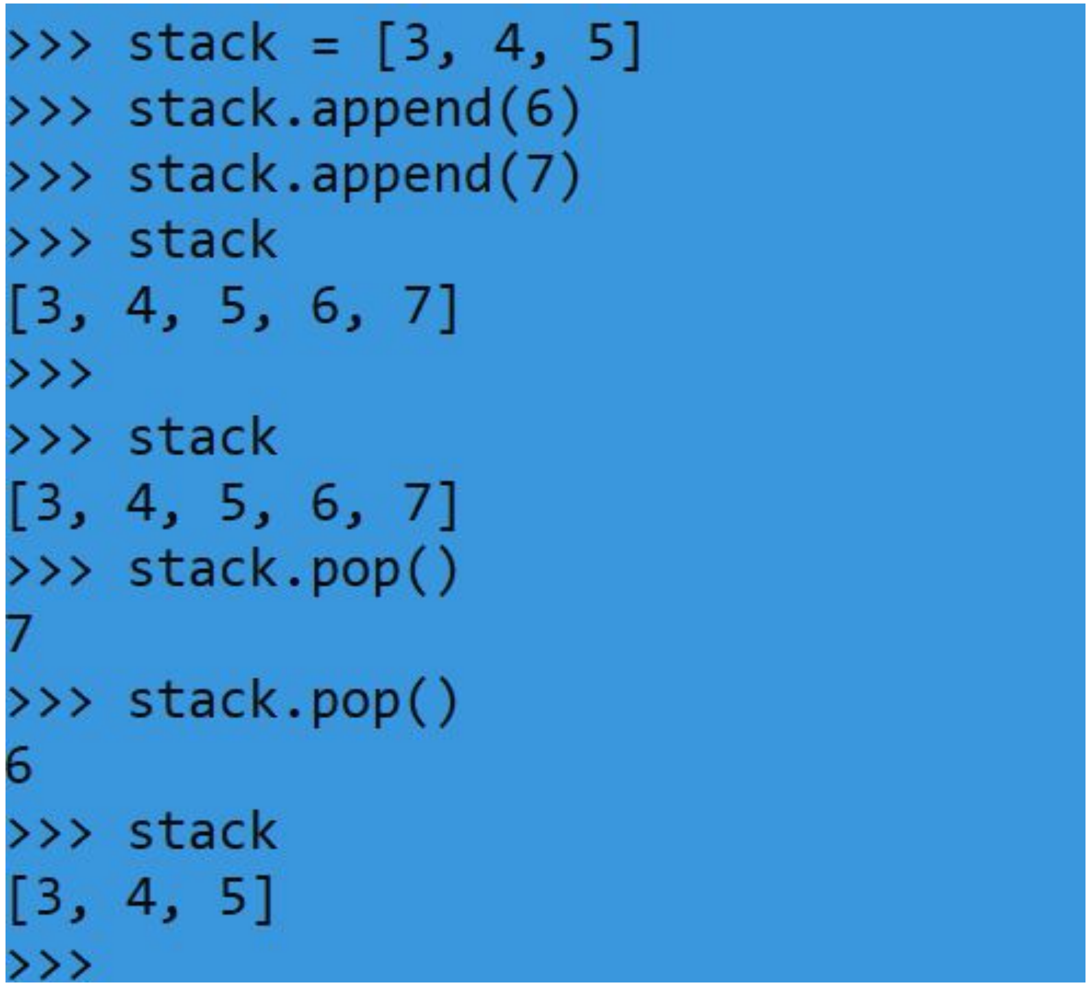
I can toast!
Let me rephrase that
I can do ANYTHING you tell me to...
... provided you speak my language

Dr. Rahul Mishra
Assistant Professor
DA-IICT, Gandhinagar

Lecture 9

Different Data Types

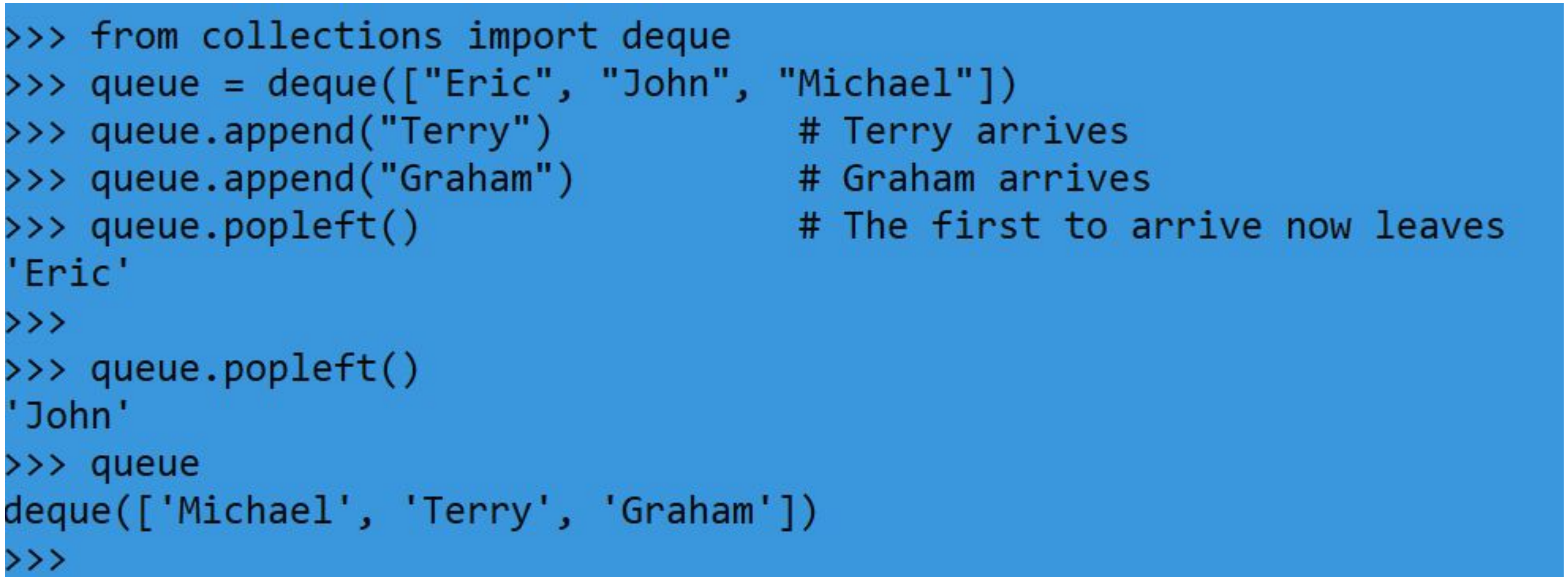
Using Lists as Stacks



```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>>
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack.pop()
6
>>> stack
[3, 4, 5]
>>>
```

Using Lists as Queue

To implement a queue, use *collections.deque* which was designed to have fast appends and pops from both ends.
For example:

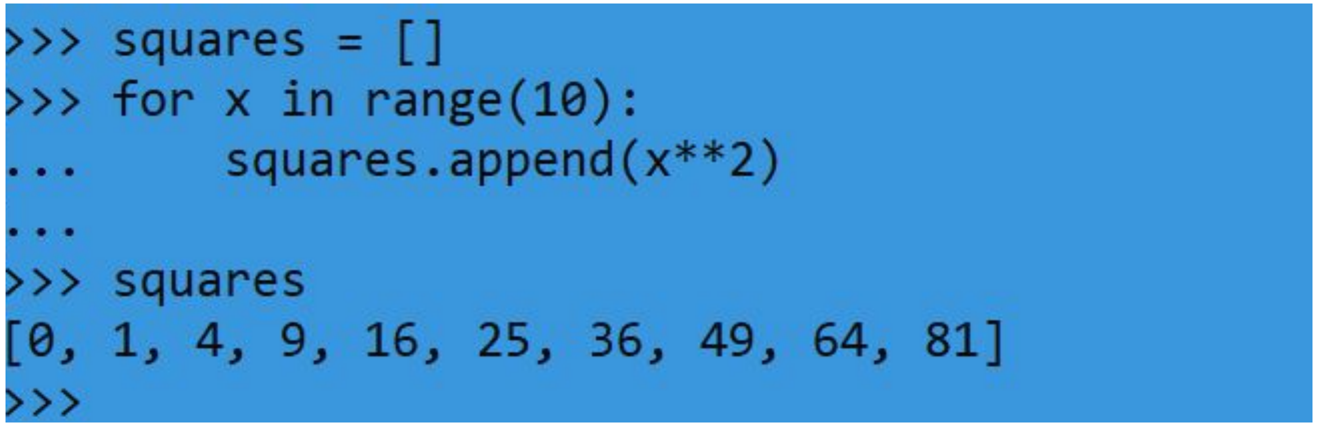


```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")          # Graham arrives
>>> queue.popleft()                 # The first to arrive now leaves
'Eric'
>>>
>>> queue.popleft()
'John'
>>> queue
deque(['Michael', 'Terry', 'Graham'])
>>>
```

List Comprehensions

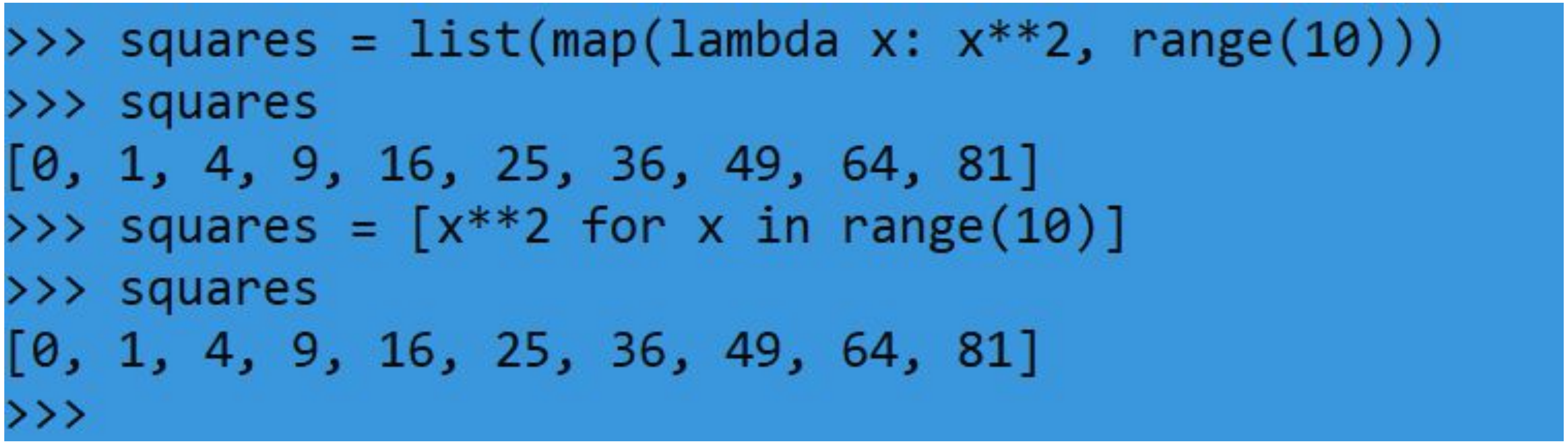
List comprehensions provide a concise way to create lists.

Common applications are to make new lists where each element is the result of some operations applied to each member of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.



```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
```

List Comprehensions



```
>>> squares = list(map(lambda x: x**2, range(10)))
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> squares = [x**2 for x in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
```

List Comprehensions

- A list comprehension consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses.

- The result will be a new list resulting from evaluating the expression in the context of the for and if clauses which follow it.

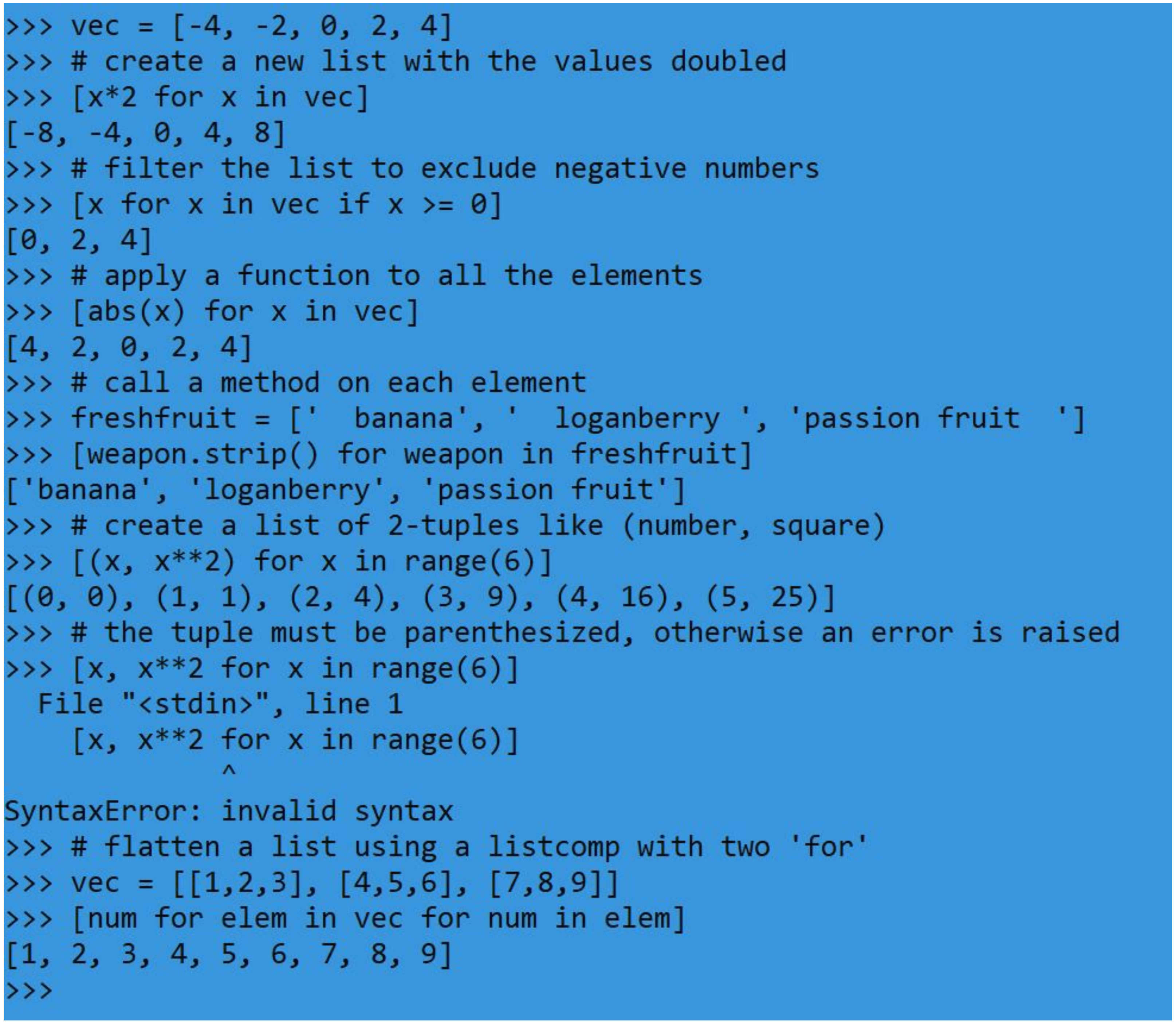
- For example, this listcomp combines the elements of two lists if they are not equal:that

```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
>>>
```

List Comprehensions

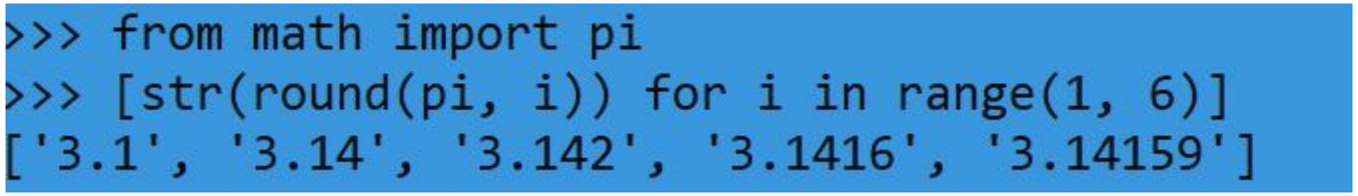
```
>>> combs = []
>>> for x in [1,2,3]:
...     for y in [3,1,4]:
...         if x != y:
...             combs.append((x, y))
...
>>> combs
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
>>>
```

List Comprehensions



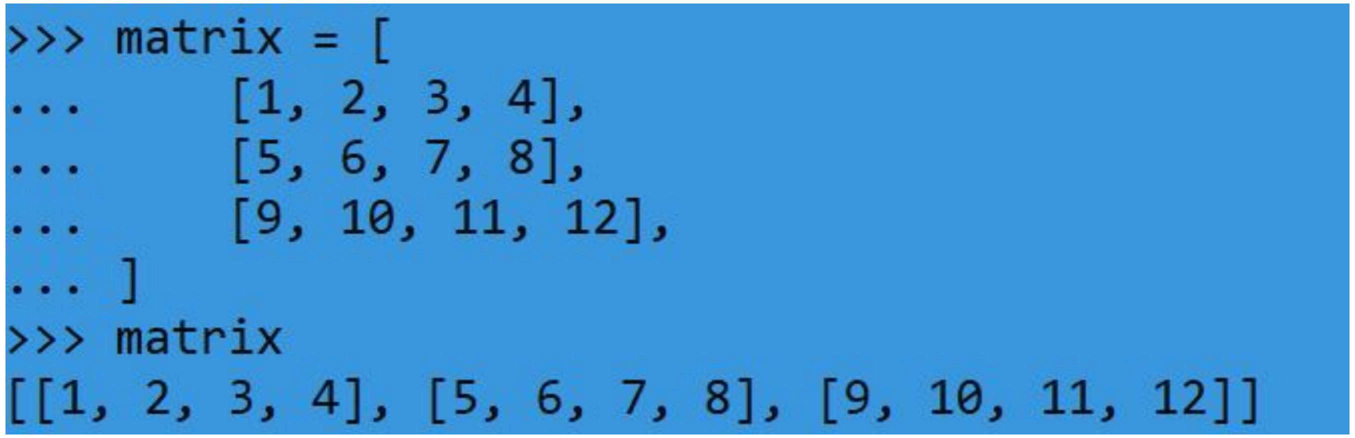
```
>>> vec = [-4, -2, 0, 2, 4]
>>> # create a new list with the values doubled
>>> [x*2 for x in vec]
[-8, -4, 0, 4, 8]
>>> # filter the list to exclude negative numbers
>>> [x for x in vec if x >= 0]
[0, 2, 4]
>>> # apply a function to all the elements
>>> [abs(x) for x in vec]
[4, 2, 0, 2, 4]
>>> # call a method on each element
>>> freshfruit = ['  banana', '  loganberry ', 'passion fruit  ']
>>> [weapon.strip() for weapon in freshfruit]
['banana', 'loganberry', 'passion fruit']
>>> # create a list of 2-tuples like (number, square)
>>> [(x, x**2) for x in range(6)]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
>>> # the tuple must be parenthesized, otherwise an error is raised
>>> [x, x**2 for x in range(6)]
  File "<stdin>", line 1
    [x, x**2 for x in range(6)]
             ^
SyntaxError: invalid syntax
>>> # flatten a list using a listcomp with two 'for'
>>> vec = [[1,2,3], [4,5,6], [7,8,9]]
>>> [num for elem in vec for num in elem]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

List Comprehensions



```
>>> from math import pi
>>> [str(round(pi, i)) for i in range(1, 6)]
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

Nested List Comprehensions



```
>>> matrix = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
>>> matrix
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

```
>>> [[row[i] for row in matrix] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
>>>
```

List Comprehensions

```
>>> transposed = []
>>> for i in range(4):
...     transposed.append([row[i] for row in matrix])
...
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

```
>>> transposed = []
>>> for i in range(4):
...     # the following 3 lines implement the nested
listcomp
...     transposed_row = []
...     for row in matrix:
...         transposed_row.append(row[i])
...     transposed.append(transposed_row)
...
 >>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

In the real world, you should prefer built-in functions to complex flow statements.
The zip() function would do a great job for this use case:

```
list(zip(*matrix))
[(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]
```