KOÇ ÜNİVERSİTESİ

08 – Lists

COMP 125 Programming with Python

Recording Disclaimer

The synchronous sessions are recorded (audiovisual recordings). The students are not required to keep their cameras on during class.

The audiovisual recordings, presentations, readings and any other works offered as the course materials aim to support remote and online learning. They are only for the personal use of the students. Further use of course materials other than the personal and educational purposes as defined in this disclaimer, such as making copies, reproductions, replications, submission and sharing on different platforms including the digital ones or commercial usages are strictly prohibited and illegal.

The persons violating the above-mentioned prohibitions can be subject to the administrative, civil, and criminal sanctions under the Law on Higher Education Nr. 2547, the By-Law on Disciplinary Matters of Higher Education Students, the Law on Intellectual Property Nr. 5846, the Criminal Law Nr. 5237, the Law on Obligations Nr. 6098, and any other relevant legislation.

The academic expressions, views, and discussions in the course materials including the audio-visual recordings fall within the scope of the freedom of science and art.

How can we store and organize data?

- **Collection**: A data structure used to store multiple values in a single unit
  - In most other programming languages, the most basic collection is an array, which stores values of the same type

- In Python, the most basic, and arguably one of the most useful, collection is the **`List`** data structure

- **`List`** is also a *sequence* (recall the `range` function and `strings`)

Sequences

- **Sequence**: an object that contains multiple items of data
  - The items are stored in sequence one after another

- Python provides different types of sequences, including lists and tuples
  - The difference between these is that a list is mutable, and a tuple is immutable

Lists

- A data type for storing values in a linear collection
- Python declaration

```
[1, 2, 3, 4, 5]
```

  - Using **brackets [ ]** to write in the code
  - List items are separated with commas
  - The **length** of a list is the number of items it contains (just like strings)
  - Python function `len()` to return the length of any sequence, also including a list

```
my_list = [1, 2, 3, 4, 5]
```

`len(my_list)` will return 5

Lists

- Can be defined to keep items of different types

```
['a', 'b', 'c', 'd', 'e′]
[True, False, False]
```

- Can have a varying number of items including 0 (empty list) and 1

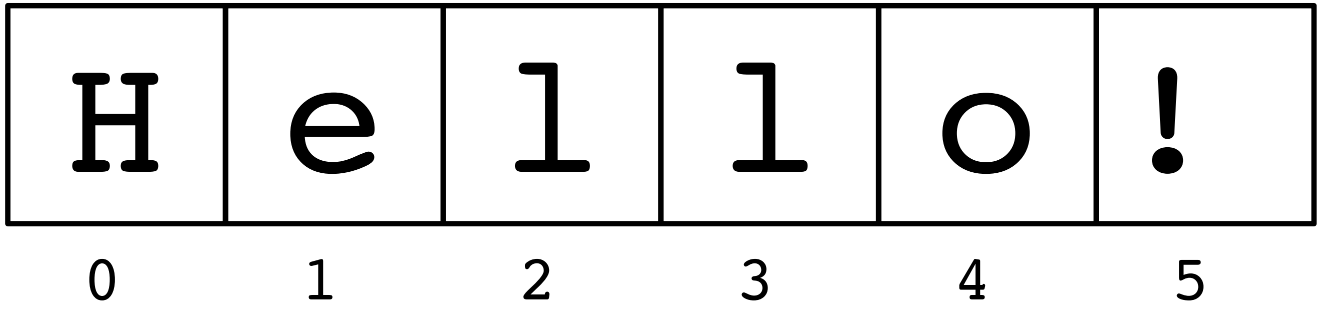
```
[ ]
[3.2]
```

- Can include items of different data types

```
[ 1, 3.2, 'a', 'b', 'c', True]
```

(revisited) String indexing

- Each character in the string is associated with an **`index`**
  - **index**: An integer representing the location of a character in a string
- Zero-based indexing
  - Indices start with 0 (not 1)
  - The **first character** of a string exists at **index 0**
  - The **last character** of a string exists at **index `len(s) - 1`**
  - Index **`len(s)`** is NOT a valid index



H	e	l	l	o	!
0	1	2	3	4	5

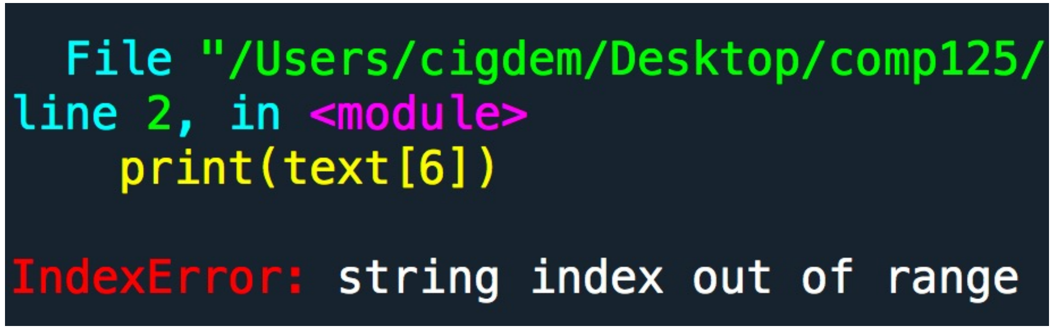
(revisited) **String indexing**

- You can access a character in the sequence (string) by specifying its index in **square brackets [ ]**
- The character at index `i` of string `s` can be accessed with the expression `s[i]`

```
text = 'Hello!'
my_character = text[1]    # my_character = 'e'
print(text[4])            # displays o on the screen
```

- You MUST use valid index values

```
text = 'Hello!'
print(text[6])
```



```
  File "/Users/cigdem/Desktop/comp125/
line 2, in <module>
    print(text[6])

IndexError: string index out of range
```

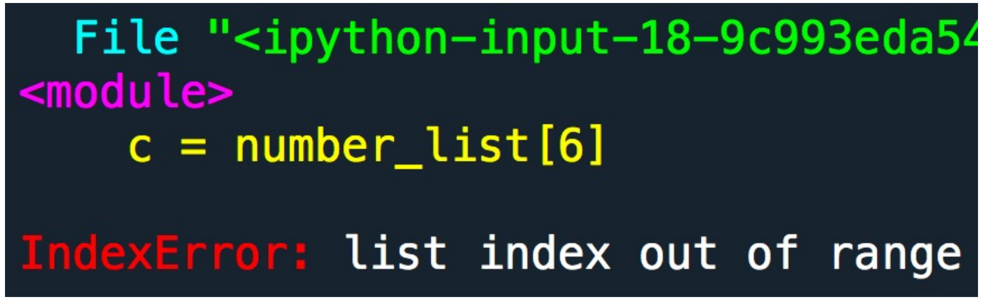
List indexing

- Indexing lists is like indexing strings

```
number_list = [10, 20, 30, 40]
another_list = [10, 'comp 125', True]
a = number_list[2]                       # a = 30
b = another_list[1]                      # b = 'comp 125'
```

- You MUST use valid index values

```
c = number_list[6]
```



```
  File "<ipython-input-18-9c993eda54
<module>
    c = number_list[6]

IndexError: list index out of range
```

- The index of last item is `len(list)-1`

Iterating over a list

- Write a code fragment that calculates the sum of the items in a given list

```
numbers = [10, 20, 30, 40]
my_sum = 0
for i in range(len(numbers)):
    my_sum += numbers[i]
```

- Use the `len()` function to prevent *IndexError* when iterating over a list with a loop

Iterating over a list (using the `in` operator)

- Write a code fragment that calculates the sum of the items in a given list

```
numbers = [10, 20, 30, 40]
my_sum = 0
for i in numbers:
     my_sum += i
```

Displaying lists

```
fruits = ['apple', 'banana', 'mango']
print(fruits)
```

Output

```
['apple', 'banana', 'mango']
```

How to display list items one by one?

```
fruits = ['apple', 'banana', 'mango']
for current in range(len(fruits)):
     print(fruits[current])
```

Output

```
apple
banana
mango
```

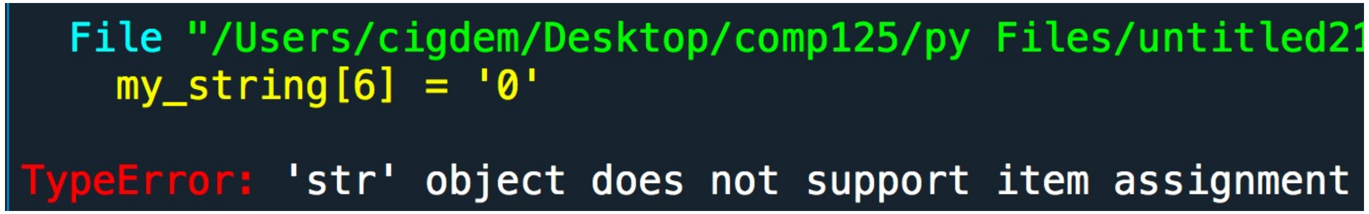
Alternative implementation

```
for current in fruits:
   print(current)
```

(revisited) Strings are immutable

- After the strings are created, their characters cannot be modified through indexing
- Indexing with immutable sequences can only be used for **accessing**

```
my_string = 'COMP125'
print(my_string[5])                  # Works
my_character = my_string[6]          # Also works
my_string[6] = '0'                   # Gives an error
```



```
  File "/Users/cigdem/Desktop/comp125/py Files/untitled21
    my_string[6] = '0'

TypeError: 'str' object does not support item assignment
```

- To change a string, you must first build a new string and then re-assign the string variable

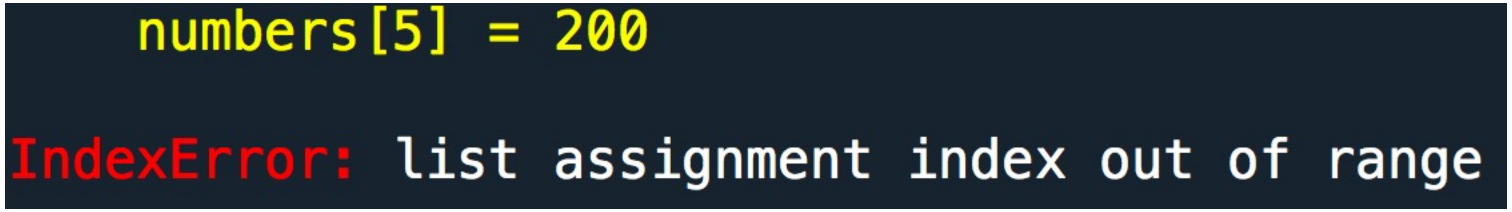
```
my_string = 'COMP125'
my_string = 'COMP105'
```

Lists are mutable

- Items in a mutable sequence can be modified after the sequence is created

- Lists are mutable, and so their items can be modified through indexing

```
numbers = [10, 20, 30, 40]
print(numbers)
numbers[2] = 100
print(numbers)
numbers[5] = 200
```



```
    numbers[5] = 200

IndexError: list assignment index out of range
```

Example

- Write a program which replaces list items of a string data type with -1

```
my_list = ['a', 1, 4.4, 'b', 'comp', 6, True]
for i in range(len(my_list)):
    if type(my_list[i]) == str:
        my_list[i] = -1
```

Adding an item to a list – `append()`

- The `append()` function adds a single item to the end of a list

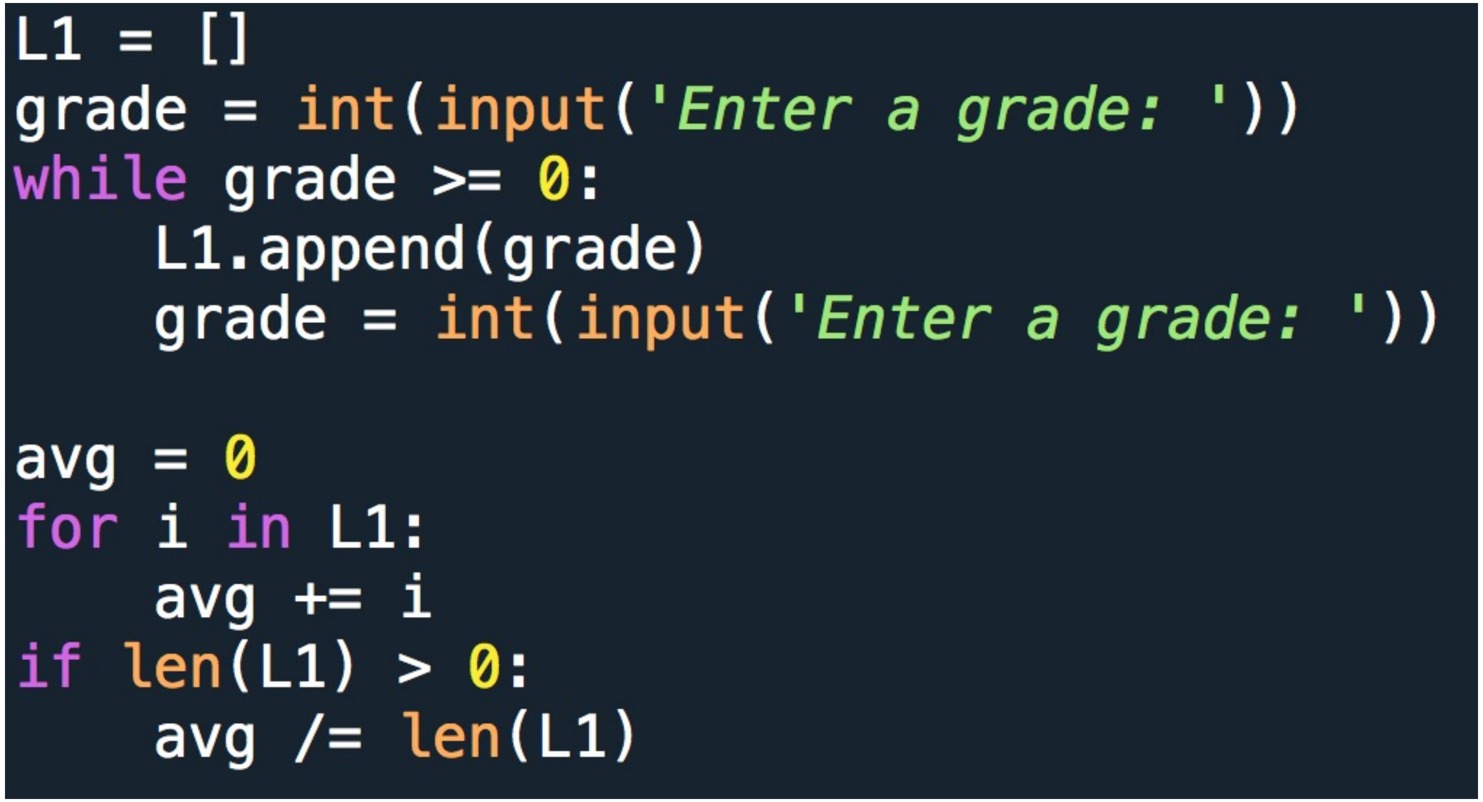
```
L1 = [10, 20, 30, 40]
L1.append(50)
print(L1)
```

Output

```
[10, 20, 30, 40, 50]
```

Example

- Write a program that takes student grades from a user, keeps these grades in a list, and calculates the average grade. The program should continue taking grades until the user enters a negative value.



```
L1 = []
grade = int(input('Enter a grade: '))
while grade >= 0:
    L1.append(grade)
    grade = int(input('Enter a grade: '))

avg = 0
for i in L1:
    avg += i
if len(L1) > 0:
    avg /= len(L1)
```

Removing the last item of a list – `pop()`

- The **`pop()`** function removes the last item in a list and returns this removed item

```
L2 = [10, 20, 30, 40]
removed = L2.pop()
print(L2)
print(removed)
```

Output

```
[10, 20, 30]
40
```

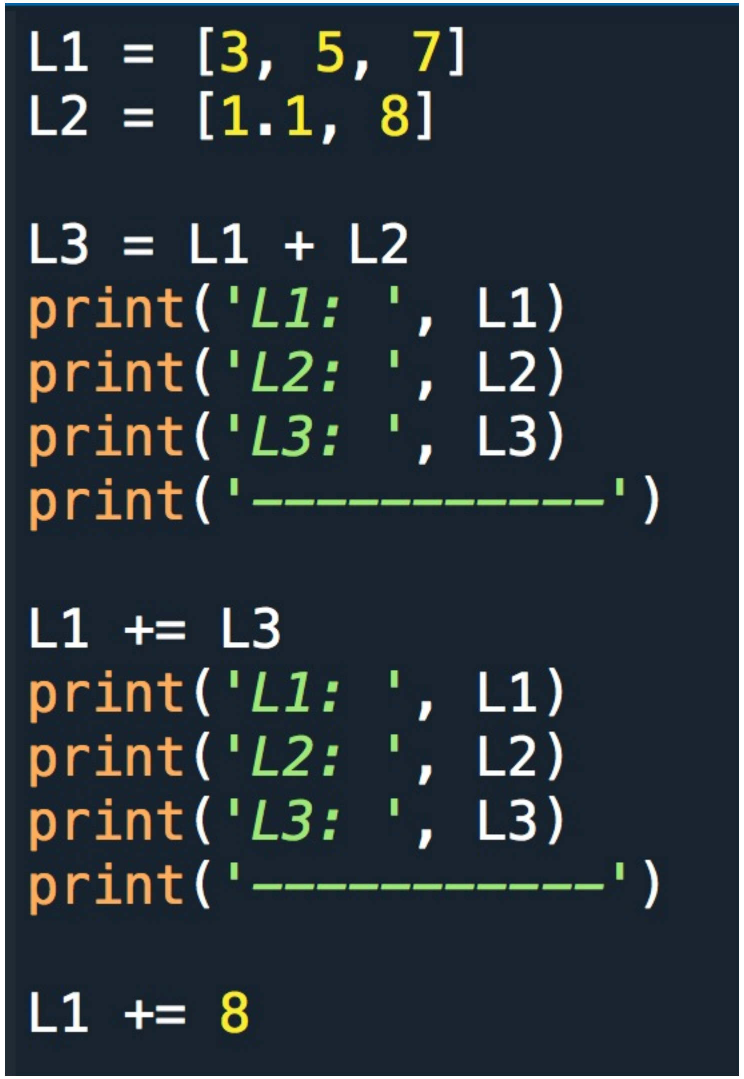
Concatenating lists

- Concatenate: Join two things together (you have seen the string version)

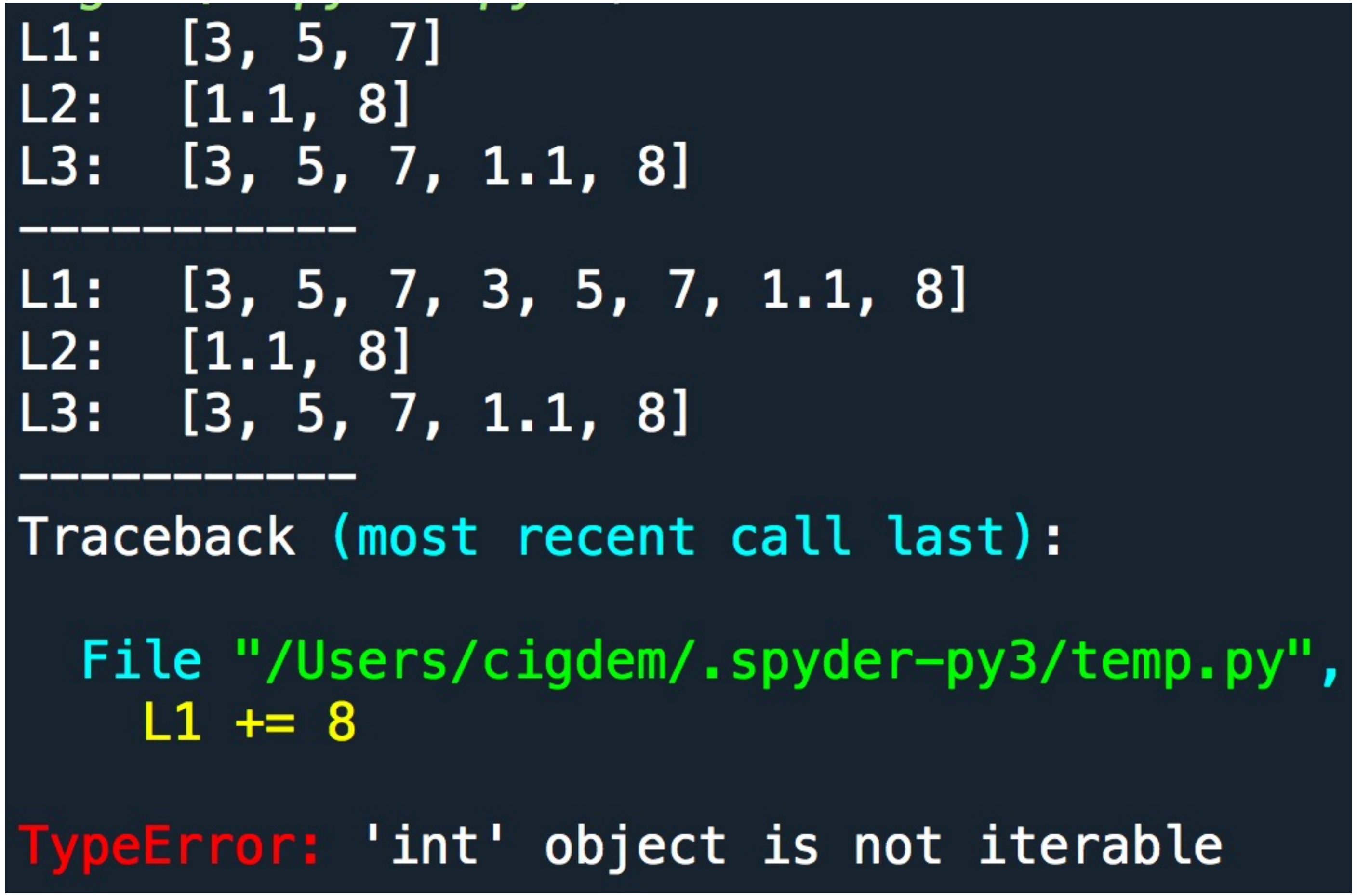
- The + operator can be used to concatenate two lists
    - **Cannot concatenate a list with another data type, such as a number**

- The += augmented assignment operator can also be used to concatenate lists

Concatenating lists



```
L1 = [3, 5, 7]
L2 = [1.1, 8]

L3 = L1 + L2
print('L1: ', L1)
print('L2: ', L2)
print('L3: ', L3)
print('------------')

L1 += L3
print('L1: ', L1)
print('L2: ', L2)
print('L3: ', L3)
print('------------')

L1 += 8
```



```
L1:  [3, 5, 7]
L2:  [1.1, 8]
L3:  [3, 5, 7, 1.1, 8]
------------
L1:  [3, 5, 7, 3, 5, 7, 1.1, 8]
L2:  [1.1, 8]
L3:  [3, 5, 7, 1.1, 8]
------------
Traceback (most recent call last):

  File "/Users/cigdem/.spyder-py3/temp.py",
    L1 += 8

TypeError: 'int' object is not iterable
```

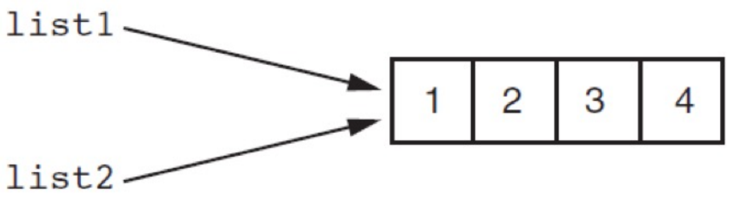
You can use += only for concatenating other lists!

Copying lists

- In Python, assigning one variable to another simply makes both variables reference the same object in memory

```
list1 = [1, 2, 3, 4]
list2 = list1
list1[0] = 99
print(list1)
print(list2) # refers to the same list
```

Figure 7-4 list1 and list2 reference the same list



Output

```
[99, 2, 3, 4]
[99, 2, 3, 4]
```

We haven't created a copy of the original list yet!

Copying lists

- You may use one of the following to create a separate copy of a list
  1. Create a copy of the list using list.copy() method
  2. Create a new empty list and add a copy of each item in the original list to the new list
  3. Create a new empty list and concatenate the original list to the new empty list

```
list1 = [1, 2, 3, 4]
list2 = list1.copy()

list2[0] = 99
print(list1)
print(list2)
```

Output

```
[1, 2, 3, 4]
[99, 2, 3, 4]
```

Copying lists

- You may use one of the following to create a separate copy of a list
  1. Create a copy of the list using list.copy() method
  2. Create a new empty list and add a copy of each item in the original list to the new list
  3. Create a new empty list and concatenate the original list to the new empty list

```
list1 = [1, 2, 3, 4]
list2 = []
for i in list1:
    list2.append(i)

list2[0] = 99
print(list1)
print(list2)
```

Output

```
[1, 2, 3, 4]
[99, 2, 3, 4]
```

Copying lists

- You may use one of the following to create a separate copy of a list
  1. Create a copy of the list using list.copy() method
  2. Create a new empty list and add a copy of each item in the original list to the new list
  3. Create a new empty list and concatenate the original list to the new empty list

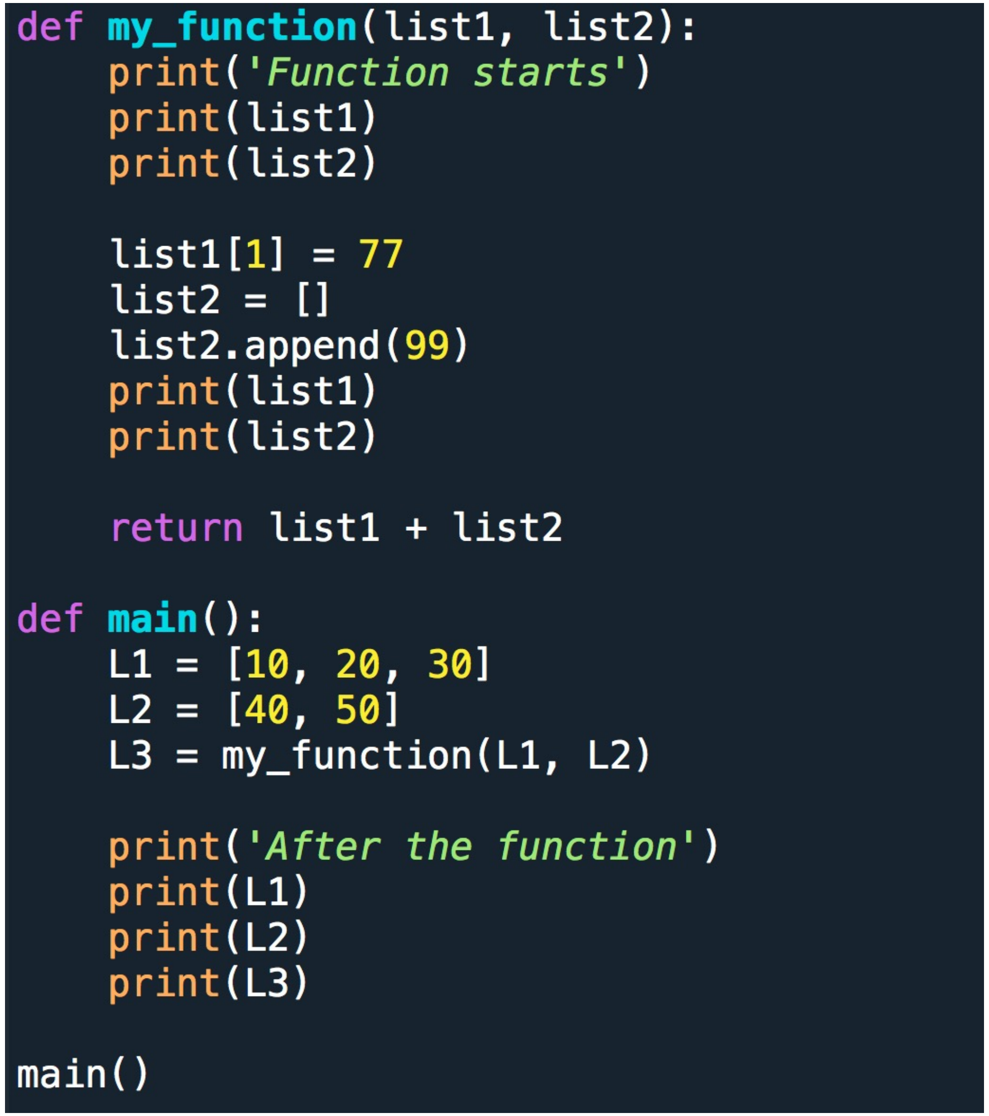
```
list1 = [1, 2, 3, 4]
list2 = []
list2 += list1

list2[0] = 99
print(list1)
print(list2)
```

<u>Output</u>

```
[1, 2, 3, 4]
[99, 2, 3, 4]
```

Passing/returning lists to/from functions

- If you change the value of a list parameter in a function, you cannot see this change after this function returns
- However, if you change the value of a list item, you will see this change after the function returns



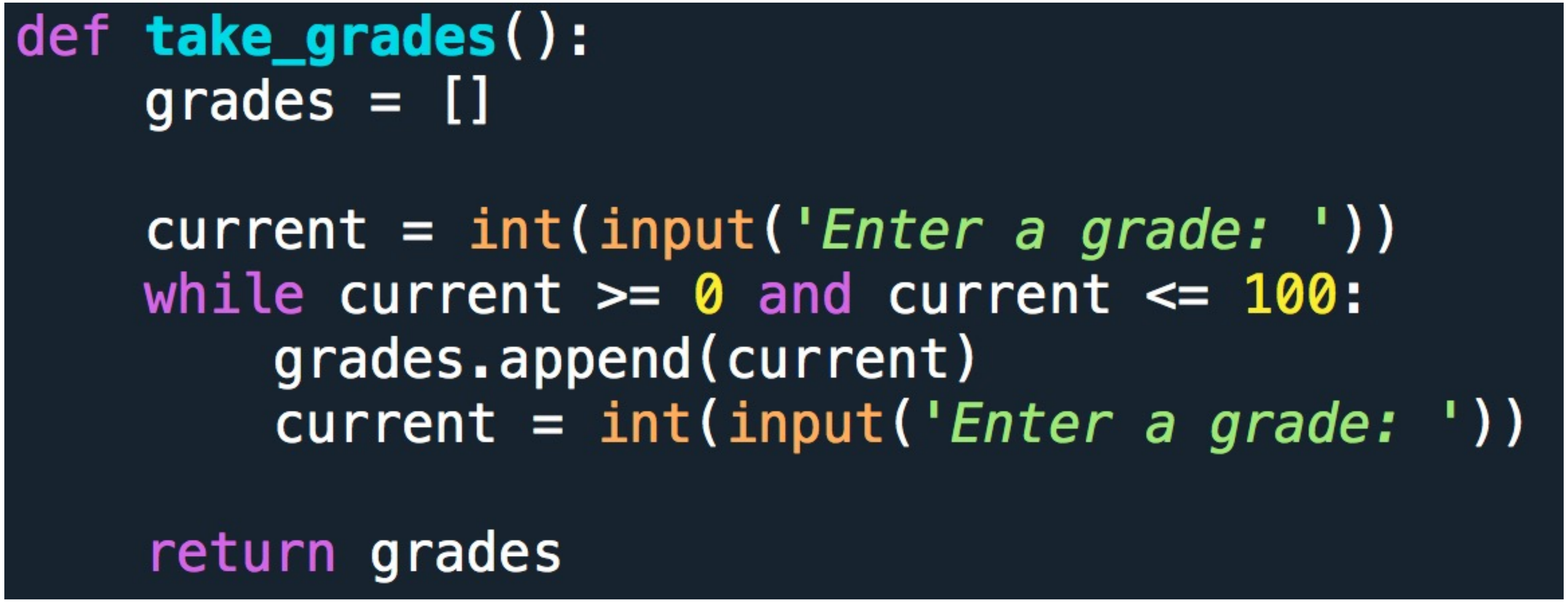
```python
def my_function(list1, list2):
    print('Function starts')
    print(list1)
    print(list2)

    list1[1] = 77
    list2 = []
    list2.append(99)
    print(list1)
    print(list2)

    return list1 + list2

def main():
    L1 = [10, 20, 30]
    L2 = [40, 50]
    L3 = my_function(L1, L2)

    print('After the function')
    print(L1)
    print(L2)
    print(L3)

main()
```

Output

```
Function starts
[10, 20, 30]
[40, 50]
[10, 77, 30]
[99]
After the function
[10, 77, 30]
[40, 50]
[10, 77, 30, 99]
```

Example

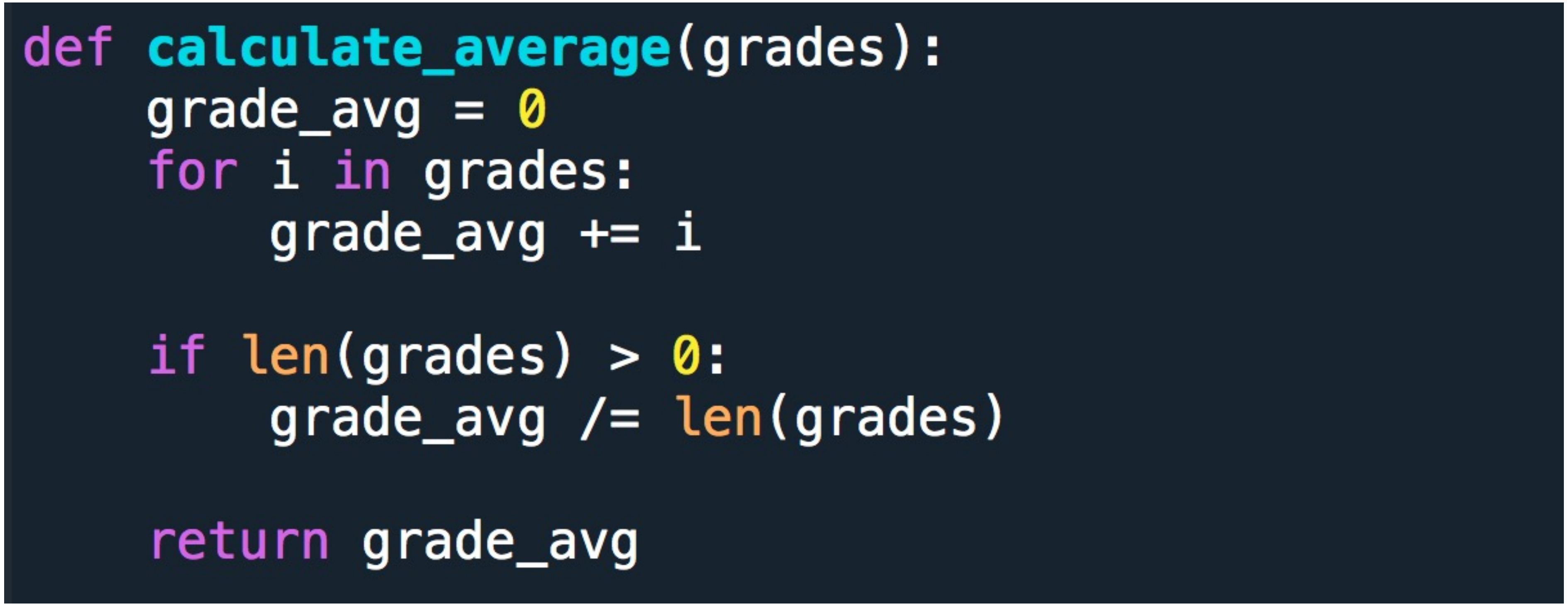
- Write a function that takes the students’ grade from the keyboard and returns them. A grade is valid only if it is between 0 and 100. The function continues taking the grades until an invalid grade is entered.



```python
def take_grades():
    grades = []

    current = int(input('Enter a grade: '))
    while current >= 0 and current <= 100:
        grades.append(current)
        current = int(input('Enter a grade: '))

    return grades
```

Example

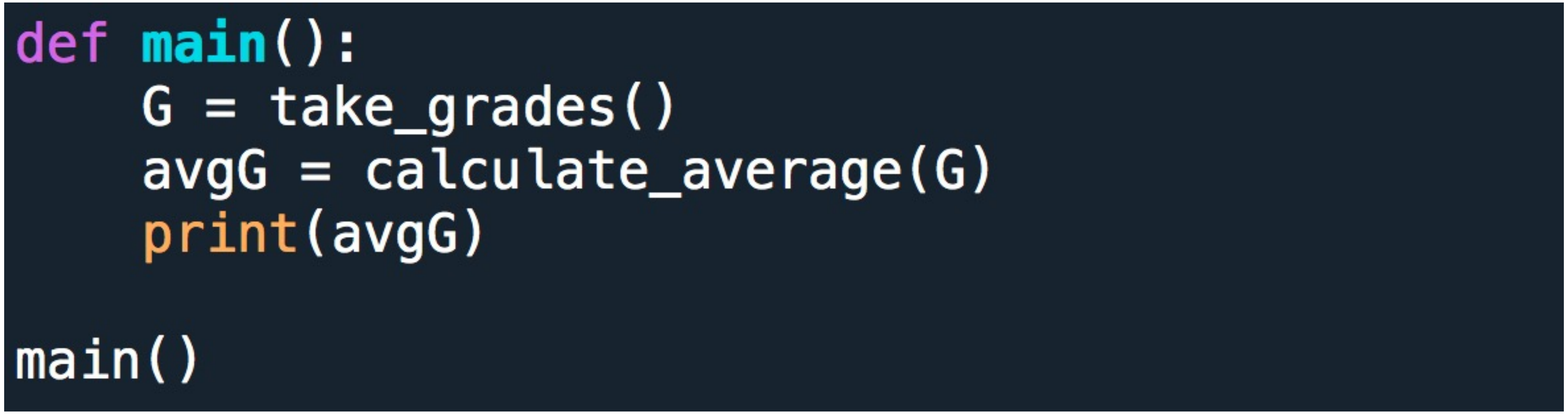
- Write another function that takes the students’ grade as its parameter and returns the average grade.



```python
def calculate_average(grades):
    grade_avg = 0
    for i in grades:
        grade_avg += i

    if len(grades) > 0:
        grade_avg /= len(grades)

    return grade_avg
```

Example

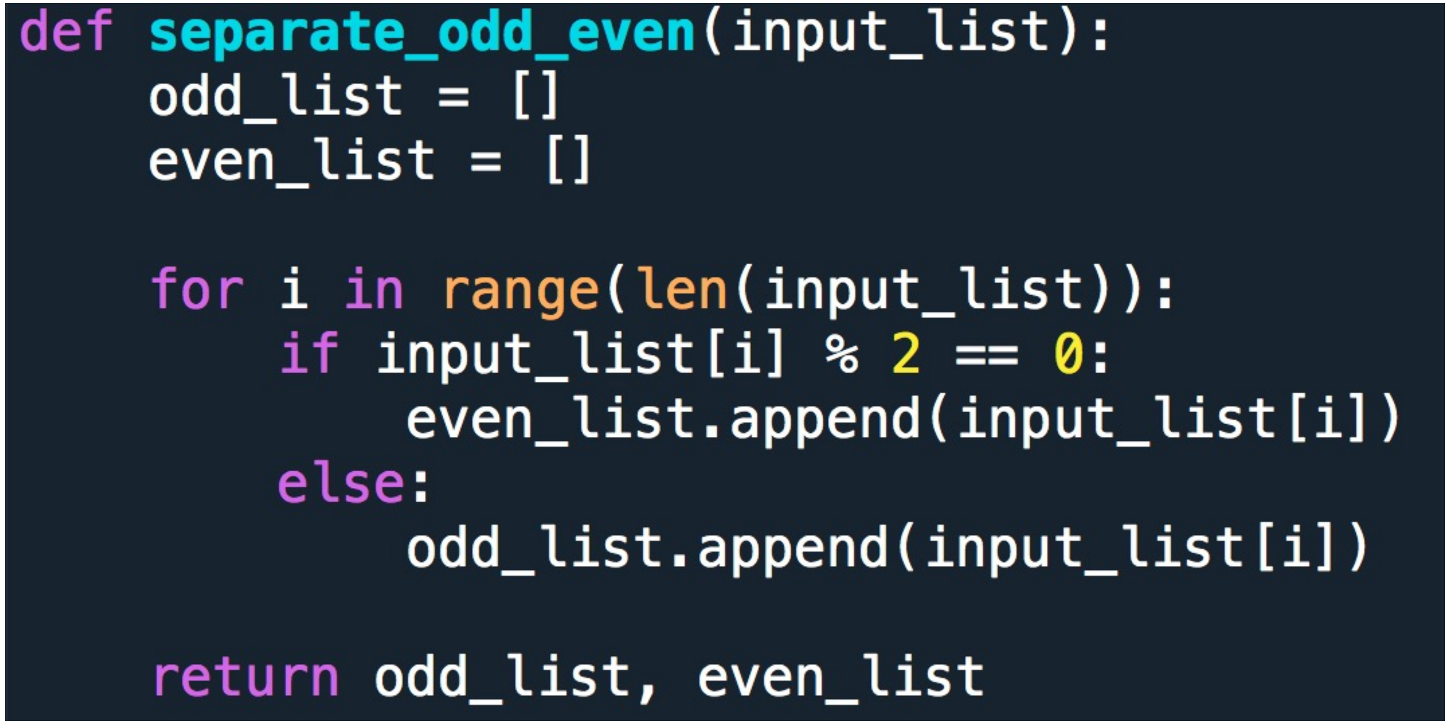
- Write a main function that calls these two functions to get the students’ grade from the user and display their average on the screen. Also call the main function.



```
def main():
    G = take_grades()
    avgG = calculate_average(G)
    print(avgG)

main()
```

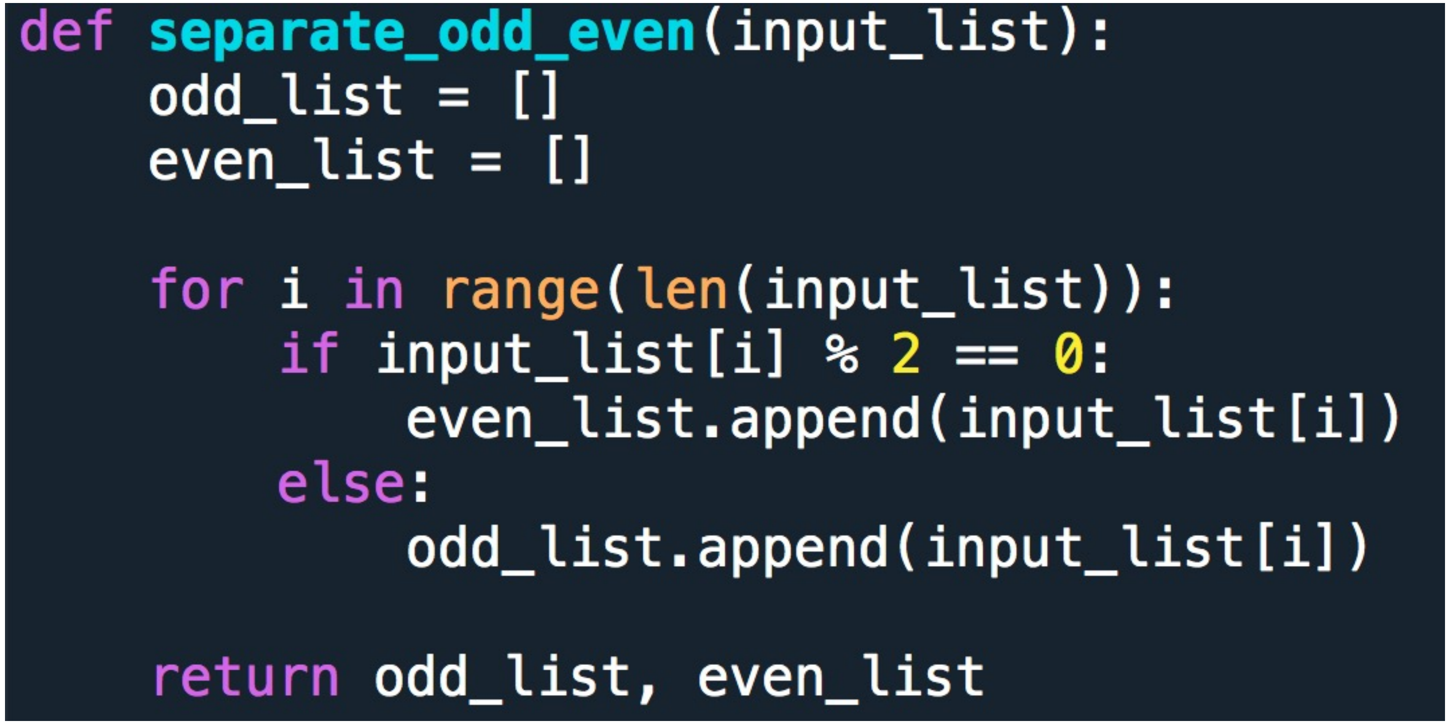
Example

- Write a function that takes a list as its parameter and returns two lists containing odd and even items in the input list. You may assume that all list items are integers.



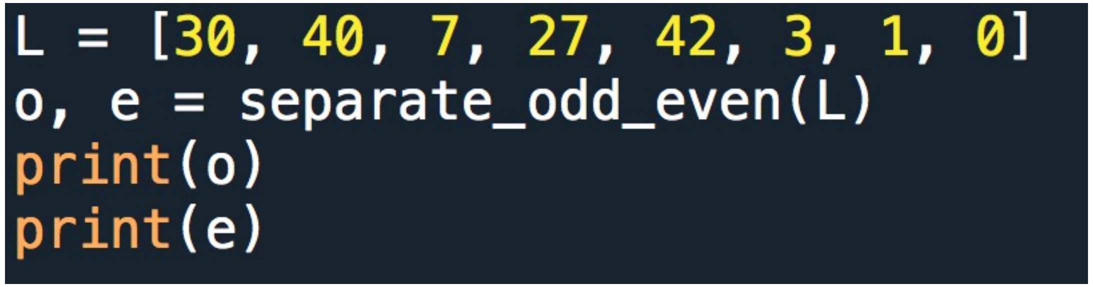
```python
def separate_odd_even(input_list):
    odd_list = []
    even_list = []

    for i in range(len(input_list)):
        if input_list[i] % 2 == 0:
            even_list.append(input_list[i])
        else:
            odd_list.append(input_list[i])

    return odd_list, even_list
```

Example

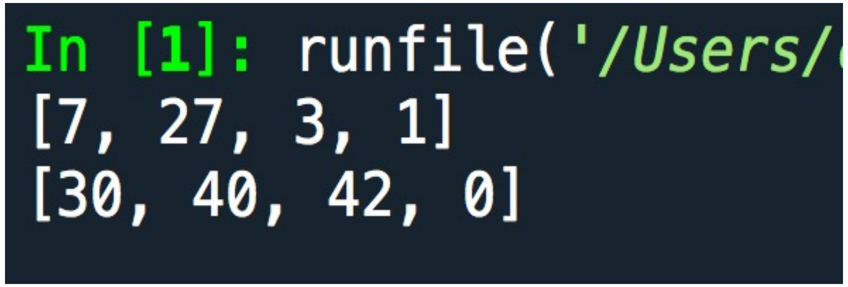


```python
def separate_odd_even(input_list):
    odd_list = []
    even_list = []

    for i in range(len(input_list)):
        if input_list[i] % 2 == 0:
            even_list.append(input_list[i])
        else:
            odd_list.append(input_list[i])

    return odd_list, even_list
```

What is the output?



```python
L = [30, 40, 7, 27, 42, 3, 1, 0]
o, e = separate_odd_even(L)
print(o)
print(e)
```



```
In [1]: runfile('/Users/
[7, 27, 3, 1]
[30, 40, 42, 0]
```

(revisited) Slicing strings

- **Slice (or substring)** of a string is a consecutive block of characters that has been extracted from the original string
- Specify a range of indices in square brackets (remember the rules for the range function)

```
alphabet = 'abcdefghijklmnopqrstuvwxyz'
s1 = alphabet[3:6]          → def
s2 = alphabet[3:1]          → empty substring
s3 = alphabet[3:12]         → defghijkl
s4 = alphabet[3:12:2]       → dfhjl
```

Slicing lists

- Slice: a span of items that are taken from a sequence
- **List slicing format**

```
list[start:end]
```

  - Result is a list containing copies of items from `start` up to, but not including, `end`
  - If `start` is not specified, 0 is used for the start index
  - If `end` is not specified, `len(list)` is used for the end index
  - Slicing expressions can include a step value and negative indices relative to the end of the list

Slicing lists

```
my_list = [44, 55, 66, 77, 88]

my_list[2:4] →       [66, 77]
my_list[1:] →        [55, 66, 77, 88]
my_list[:2] →        [44, 55]
my_list[1:4:2] →     [55, 77]
my_list[1:3:2] →     [55]
my_list[3:1] →       []
my_list[3:1:-1] →    [77, 66]
```

Slicing lists -- more

```
list1 = ['a', 'b', 'c', 'd', 'e']

list1[::-1] → ['e', 'd', 'c', 'b', 'a']

list1[::2] →  ['a', 'c', 'e']

list1[::-2] → ['e', 'c', 'a']
```

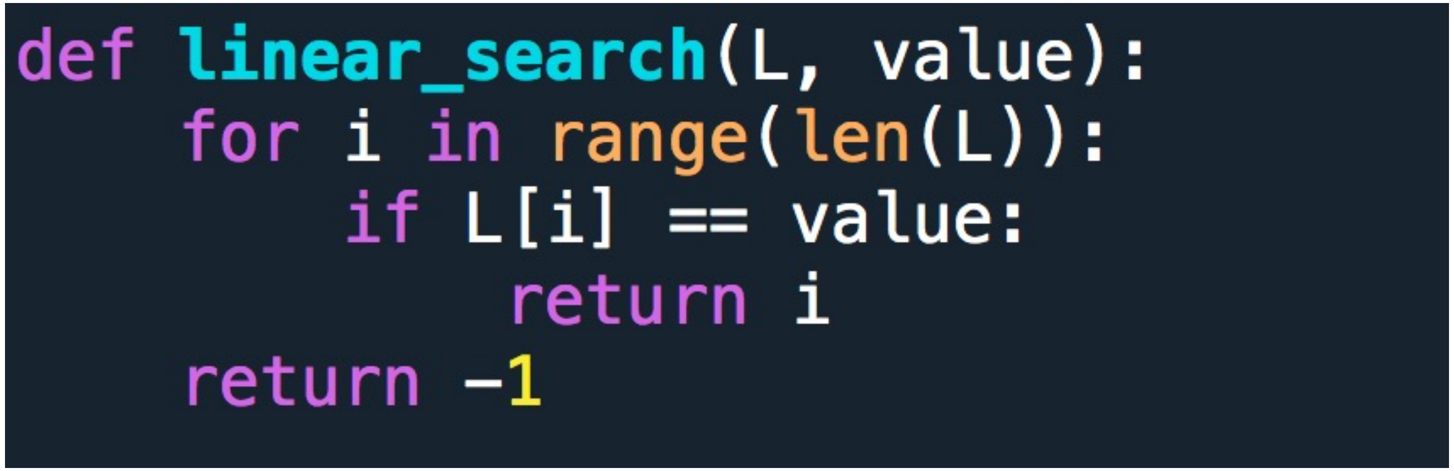
Searching an item in a list

- The `in` operator can be used to determine whether an item is in the list
- It returns `True` if the item is in the list, `False` otherwise
- The `not in` operator can be used to determine whether an item is not in the list

```
fruits = ['apple', 'banana', 'mango', 'kiwi']

'mango' in fruits
→ True

'broccoli' in fruits
→ False

'broccoli' not in fruits
→ True
```

Searching an item in a list

- Now let’s write our own search function. This function takes a list and an item value as its parameters and returns the index of this item if it is found in the list. Otherwise, if the item is not in the list, it returns -1. You may assume that the list contains unique item values.



```
def linear_search(L, value):
    for i in range(len(L)):
        if L[i] == value:
            return i
    return -1
```

More on lists

Repetition operator

- The * operator makes multiple copies of a list and joins them all together

```
numbers = [0] * 5
print(numbers)
numbers = [1, 2, 3] * 3
print(numbers)

L1 = [5, 6]
L2 = L1 * 3
L3 = [3.14, 2.71]
L3 *= 2
print(L1)
print(L2)
print(L3)
```

Output

```
[0, 0, 0, 0, 0]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
[5, 6]
[5, 6, 5, 6, 5, 6]
[3.14, 2.71, 3.14, 2.71]
```

Splitting strings to make a list

- The `split()` function splits a string where there is a white space

```
s = 'I am comprised of       words'
words = s.split()

words → ['I', 'am', 'comprised', 'of', 'words']
s → 'I am comprised of       words'
```

Splitting strings to make a list

- `split(delimeter)` splits a string where there is the delimeter

```
s = 'do,re,mi,fa,sol,la,ti'
notes = s.split(',')
notes → ['do', 're', 'mi', 'fa', 'sol', 'la', 'ti']
```

```
s = 'do, re, mi, fa, sol, la, ti'
notes = s.split(', ')
notes → ['do', 're', 'mi', 'fa', 'sol', 'la', 'ti']
```

Creating a list from the `range` function

- Recall the range function that creates an implicit sequence

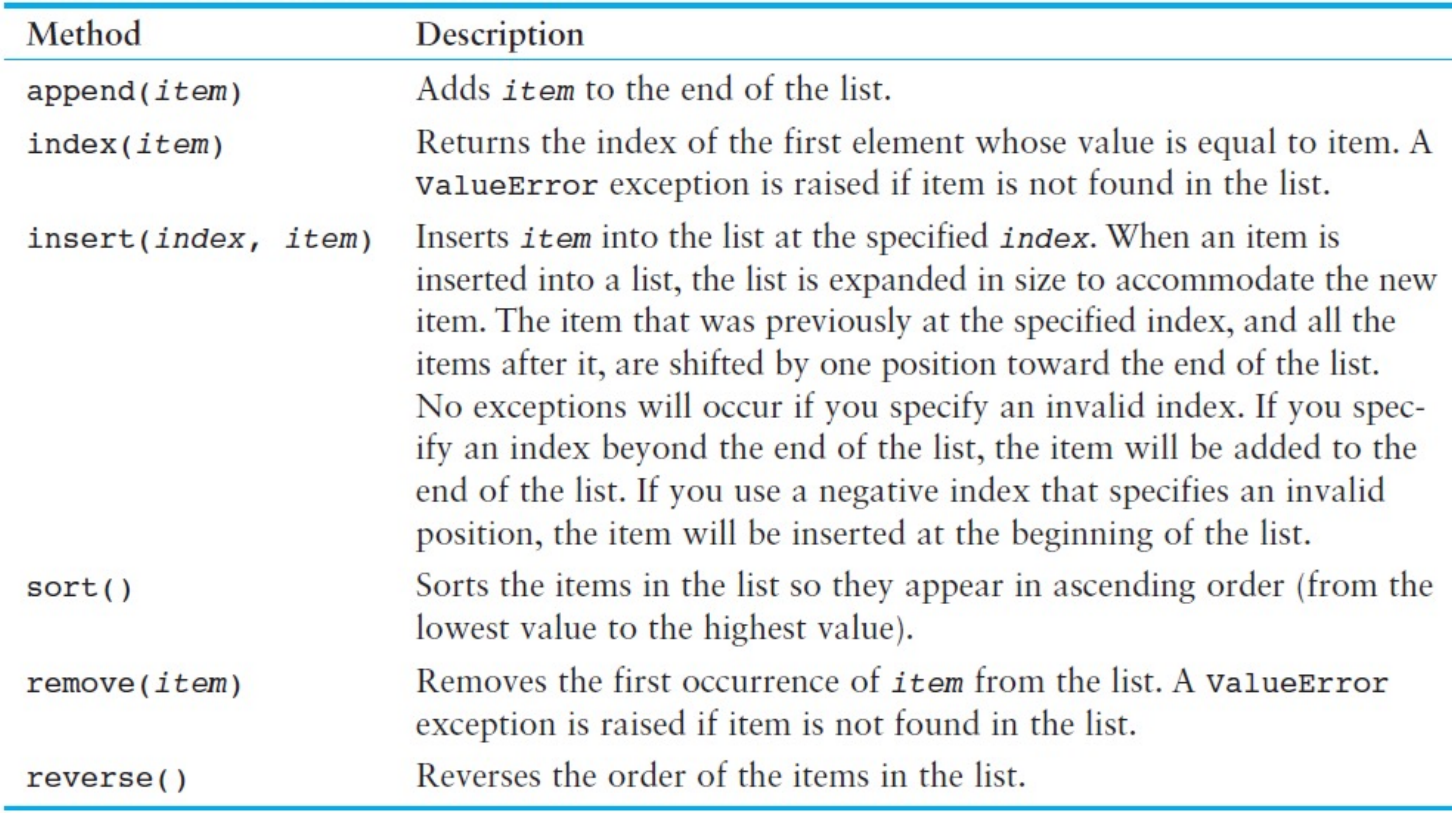
```
numbers = list (range(5))
[0, 1, 2, 3, 4]
```

Type Conversion!

```
numbers = list (range(0, 10, 2))
[0, 2, 4, 6, 8]

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

Useful list methods

Table 7-1 A few of the list methods



Method	Description
`append(item)`	Adds `item` to the end of the list.
`index(item)`	Returns the index of the first element whose value is equal to item. A `ValueError` exception is raised if item is not found in the list.
`insert(index, item)`	Inserts `item` into the list at the specified `index`. When an item is inserted into a list, the list is expanded in size to accommodate the new item. The item that was previously at the specified index, and all the items after it, are shifted by one position toward the end of the list. No exceptions will occur if you specify an invalid index. If you specify an index beyond the end of the list, the item will be added to the end of the list. If you use a negative index that specifies an invalid position, the item will be inserted at the beginning of the list.
`sort()`	Sorts the items in the list so they appear in ascending order (from the lowest value to the highest value).
`remove(item)`	Removes the first occurrence of `item` from the list. A `ValueError` exception is raised if item is not found in the list.
`reverse()`	Reverses the order of the items in the list.

Example

```
L = [1, 3, 4, 2, 6, 3, 2, 3]

idx = L.index(3)
print(idx)

L.insert(3, 7)
print(L)

L.remove(3)
print(L)

L.reverse()
print(L)

L.sort()
print(L)
```

Output

```
1
[1, 3, 4, 7, 2, 6, 3, 2, 3]
[1, 4, 7, 2, 6, 3, 2, 3]
[3, 2, 3, 6, 2, 7, 4, 1]
[1, 2, 2, 3, 3, 4, 6, 7]
```

del Statement

- In general **del** statement is used to delete object references
- Within the list context **del** removes items at specified *indices*. The indices of the remaining part are updated!

```
L = [10, 20, 30, 40, 50]

del L[1]
print(L)

del L[2]
print(L)

del L[:]
print(L)
```

Output

```
[10, 30, 40, 50]
[10, 30, 50]
[]
```

Useful built-in functions: min, max

- The built-in `min` and `max` functions return the item with the minimum and the maximum value in a sequence, respectively

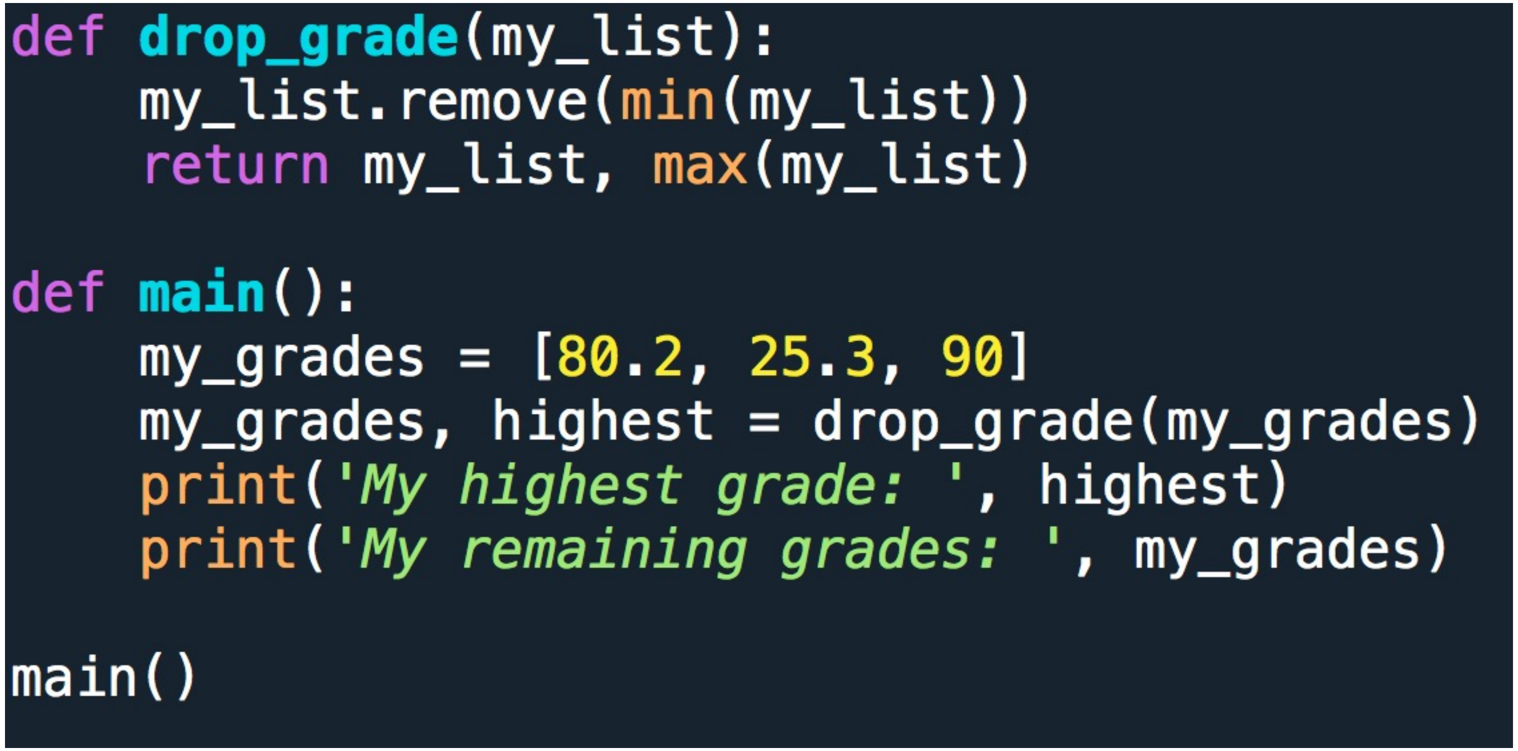
```
L = [88, 11, 22, 99, 77]

print(min(L))
print(max(L))
```

Output

```
11
99
```

Example

- Write a function that takes a list of exam grades as an input, drops the lowest grade from the list, and returns the remaining list and also the highest grade. You may assume that the list includes at least two grades.

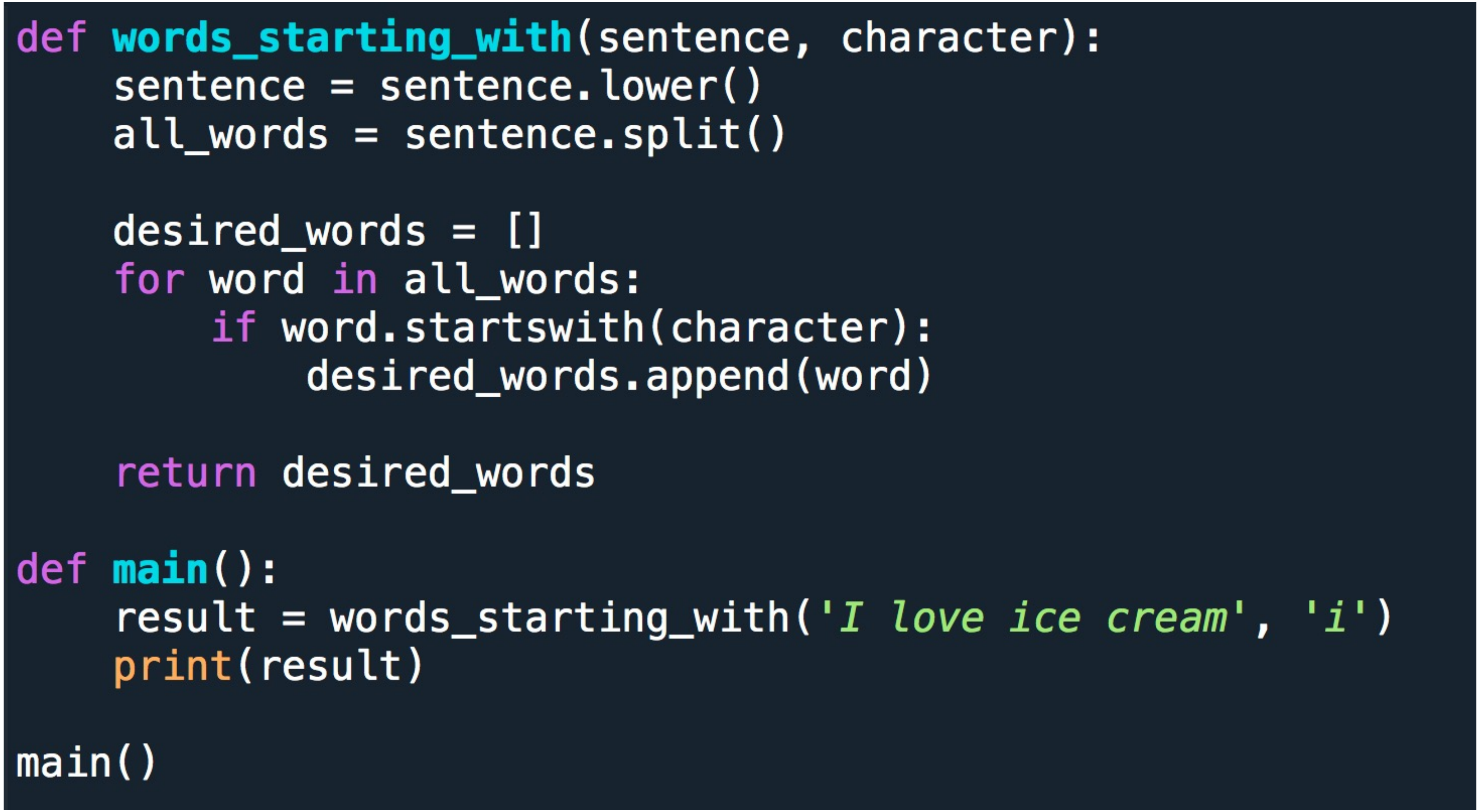


```python
def drop_grade(my_list):
    my_list.remove(min(my_list))
    return my_list, max(my_list)

def main():
    my_grades = [80.2, 25.3, 90]
    my_grades, highest = drop_grade(my_grades)
    print('My highest grade: ', highest)
    print('My remaining grades: ', my_grades)

main()
```

Example

- Write a function that takes a string and a character and returns all words in the string starting with the given character.

```
def words_starting_with(sentence, character):
    ...


e.g.,
my_words = words_starting_with('I love ice cream', 'i')
my_words → ['i', 'ice']
```



```python
def words_starting_with(sentence, character):
    sentence = sentence.lower()
    all_words = sentence.split()

    desired_words = []
    for word in all_words:
        if word.startswith(character):
            desired_words.append(word)

    return desired_words

def main():
    result = words_starting_with('I love ice cream', 'i')
    print(result)

main()
```

Useful built-in functions

- The str `join` method takes all items in a list (or a sequence) and joins them into one string. List items should be of the string type. It uses a given string as the separator

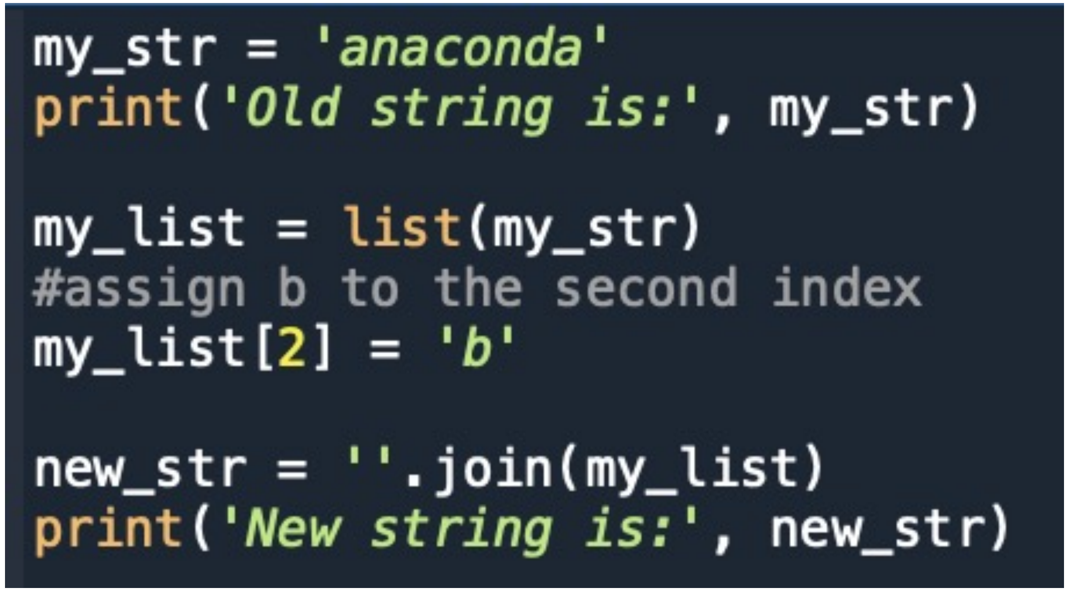
```
my_list = ['comp', '125', 'midterm']
my_string = '##'.join(my_list)
print(my_string)
```

Output

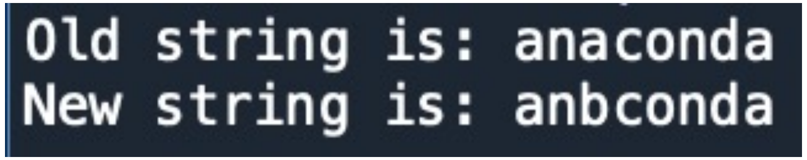
```
comp##125##midterm
```

Example

- Recall that strings are immutable! What if you want to change a specific character within the string (let’s say the character at the second index)?



```
my_str = 'anaconda'
print('Old string is:', my_str)

my_list = list(my_str)
#assign b to the second index
my_list[2] = 'b'

new_str = ''.join(my_list)
print('New string is:', new_str)
```



```
Old string is: anaconda
New string is: anbconda
```

Strings vs. Lists

Strings

- `len()`, `print()`
- slicing, indexing
- for loops
- `in`
- concatenation
- **immutable**
  - Cannot add, remove, or update elements

Lists

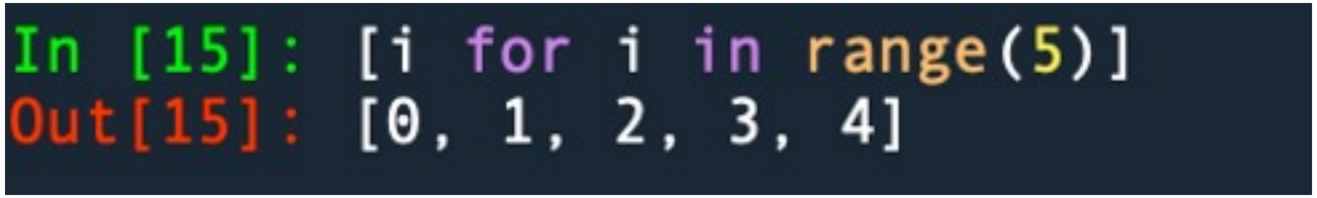
- `len()`, `print()`
- slicing, indexing
- for loops
- `in`
- concatenation
- **mutable**
  - `append()`
  - `pop()`
  - assign with indexing
  - `del` item

List Comprehension

- An efficient and compact way to create lists

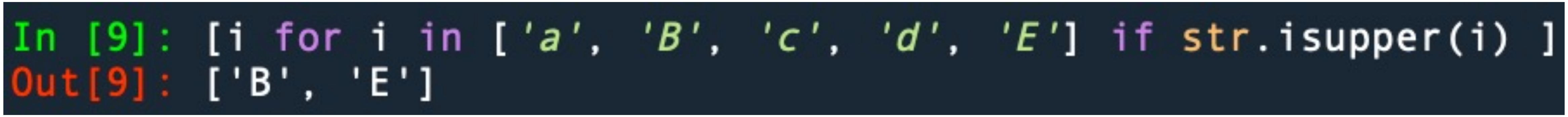
  Syntax:

  new_list = [*expression* for *member* in *iterable (if condition)*]



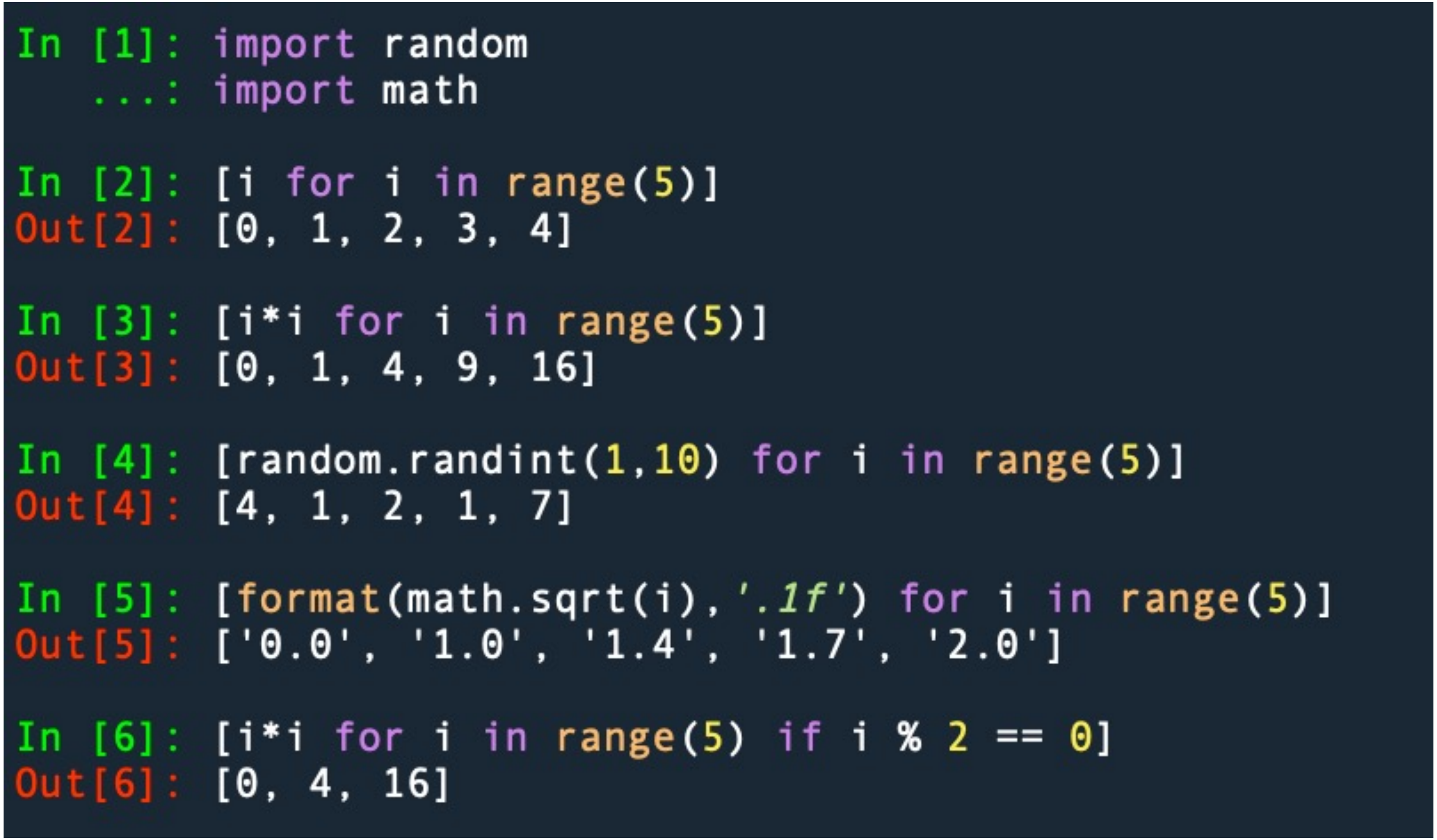
```
In [15]: [i for i in range(5)]
Out[15]: [0, 1, 2, 3, 4]
```

- List comprehension with conditional



```
In [9]: [i for i in ['a', 'B', 'c', 'd', 'E'] if str.isupper(i) ]
Out[9]: ['B', 'E']
```

List Comprehension Examples



```
In [1]: import random
   ...: import math

In [2]: [i for i in range(5)]
Out[2]: [0, 1, 2, 3, 4]

In [3]: [i*i for i in range(5)]
Out[3]: [0, 1, 4, 9, 16]

In [4]: [random.randint(1,10) for i in range(5)]
Out[4]: [4, 1, 2, 1, 7]

In [5]: [format(math.sqrt(i),'.1f') for i in range(5)]
Out[5]: ['0.0', '1.0', '1.4', '1.7', '2.0']

In [6]: [i*i for i in range(5) if i % 2 == 0]
Out[6]: [0, 4, 16]
```

Two-dimensional lists

Two-dimensional lists

- **Two-dimensional list** is a list that contains other lists as its items
  - "List of lists"
  - Also known as nested list
  - Common to think of two-dimensional lists as having **rows** and **columns**
  - Useful for working with multiple sets of data

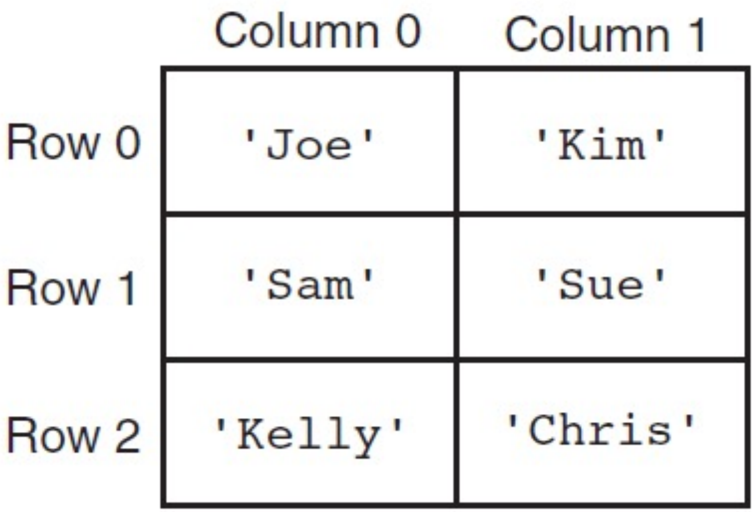
- To process data in a two-dimensional list, you need to use **two indices**
  - **Have to use valid indices for all of the dimensions**

- Typically nested loops are used to process a two-dimensional list

Two-dimensional lists

```
students = [['Joe', 'Kim'], ['Sam', 'Sue'], ['Kelly', 'Chris’]]
```

```
print(students)
print(students[1])
print(students[2])
print(students[2][1])
```



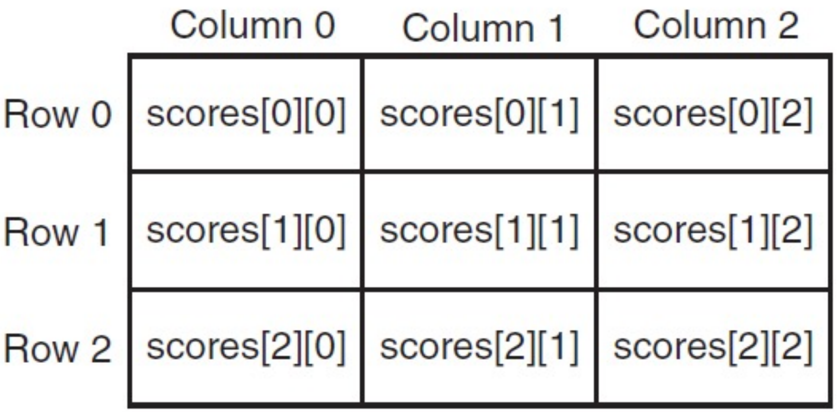
	Column 0	Column 1
Row 0	'Joe'	'Kim'
Row 1	'Sam'	'Sue'
Row 2	'Kelly'	'Chris'

Output

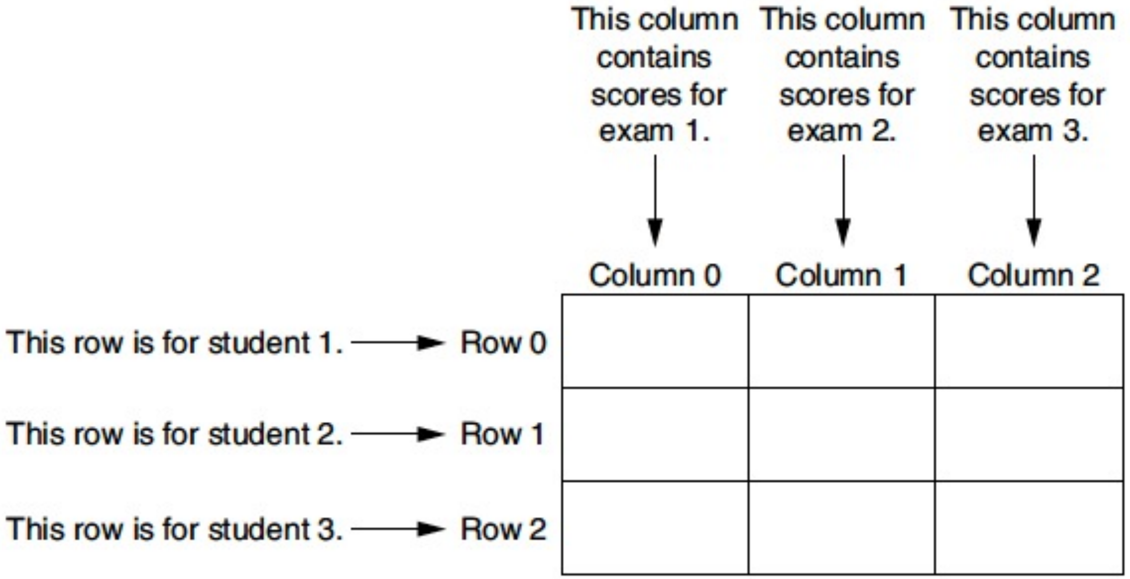
```
[['Joe', 'Kim'], ['Sam', 'Sue'], ['Kelly', 'Chris']]
['Sam', 'Sue']
['Kelly', 'Chris']
Chris
```

Two-dimensional lists

- For example, suppose you are writing a grade-averaging program for an instructor. The instructor has three students, and each student takes three exams during the semester.



	Column 0	Column 1	Column 2
Row 0	scores[0][0]	scores[0][1]	scores[0][2]
Row 1	scores[1][0]	scores[1][1]	scores[1][2]
Row 2	scores[2][0]	scores[2][1]	scores[2][2]


This column contains scores for exam 1.
This column contains scores for exam 2.
This column contains scores for exam 3.
Column 0
Column 1
Column 2
This row is for student 1.
Row 0
This row is for student 2.
Row 1
This row is for student 3.
Row 2

The elements in row 0 are referenced as follows:

```
scores[0][0]
scores[0][1]
scores[0][2]
```

The elements in row 1 are referenced as follows:

```
scores[1][0]
scores[1][1]
scores[1][2]
```

And, the elements in row 2 are referenced as follows:

```
scores[2][0]
scores[2][1]
scores[2][2]
```

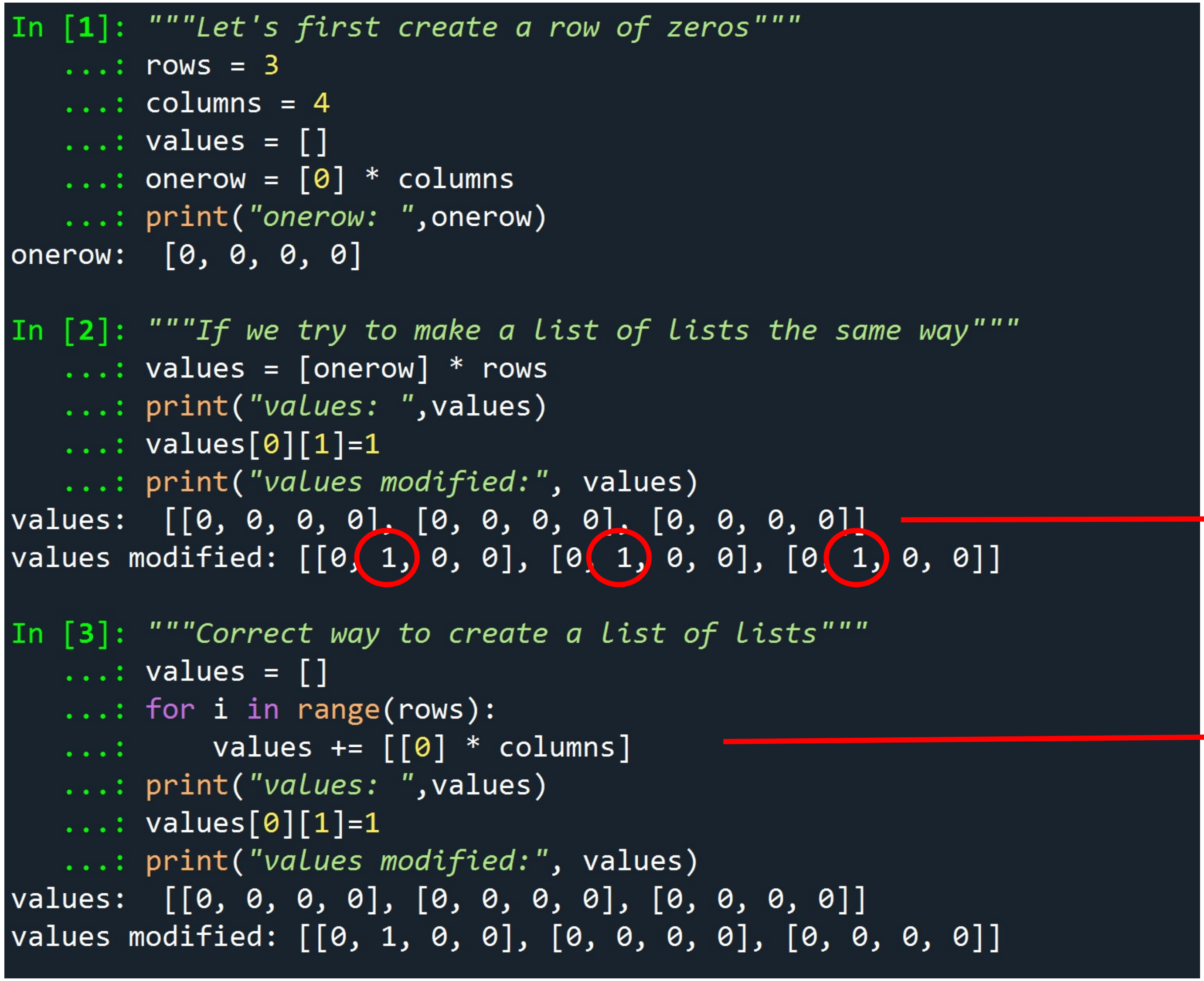
Example

```
a = [[1, 2, 3], [4, 5, 6]]

print(a[0])      [1, 2, 3]
print(a[1])      [4, 5, 6]

b = a[0]
print(b)         [1, 2, 3]

print(a[0][2])  3
```

```
a[0][1] = 7
print(a)  [[1, 7, 3], [4, 5, 6]]
print(b)  [1, 7, 3]
```

Repetition operator for List of lists

- Let's create a list of lists with 3 rows and 4 columns



```
In [1]: """Let's first create a row of zeros"""
   ...: rows = 3
   ...: columns = 4
   ...: values = []
   ...: onerow = [0] * columns
   ...: print("onerow: ",onerow)
onerow:  [0, 0, 0, 0]

In [2]: """If we try to make a list of lists the same way"""
   ...: values = [onerow] * rows
   ...: print("values: ",values)
   ...: values[0][1]=1
   ...: print("values modified:", values)
values:  [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
values modified: [[0, 1, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0]]

In [3]: """Correct way to create a list of lists"""
   ...: values = []
   ...: for i in range(rows):
   ...:     values += [[0] * columns]
   ...: print("values: ",values)
   ...: values[0][1]=1
   ...: print("values modified:", values)
values:  [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
values modified: [[0, 1, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

References the same list object, they are not independent

Every iteration creates a separate list object, they are all independent

List Comprehension for List of Lists

- Remember creating a list with list comprehension

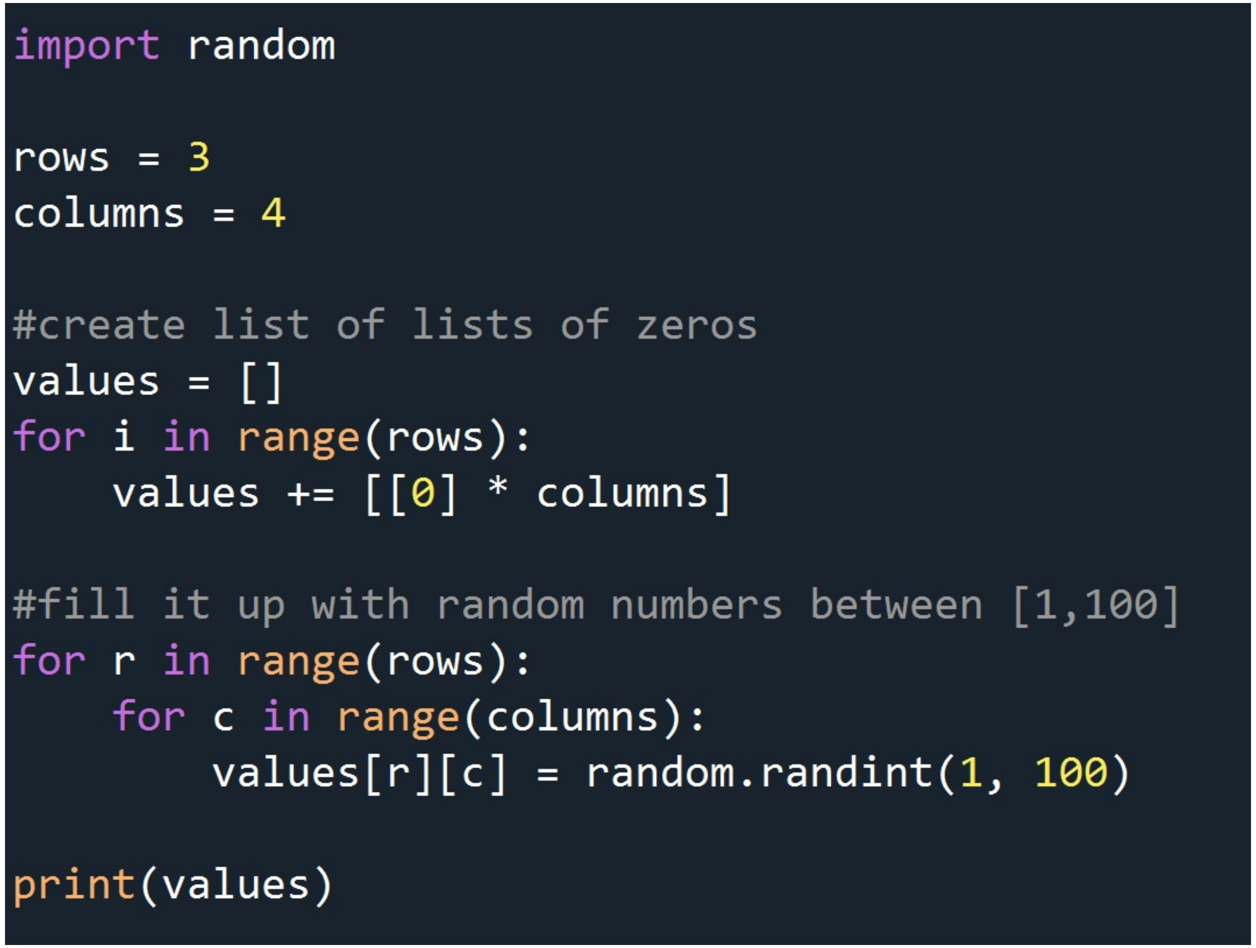
```
my_list = [0 for i in range(5)]
```

- Now letis create a 5 x 5 matrix with filled with 0s

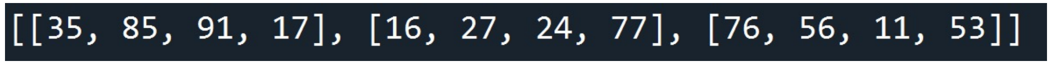
```
my_matrix = [[0 for c in range(5)] for r in range(5)]
```

Example

- Write a program that creates a two-dimensional list with the dimensions of 3 and 4 and fills it with random numbers from1 to 100.



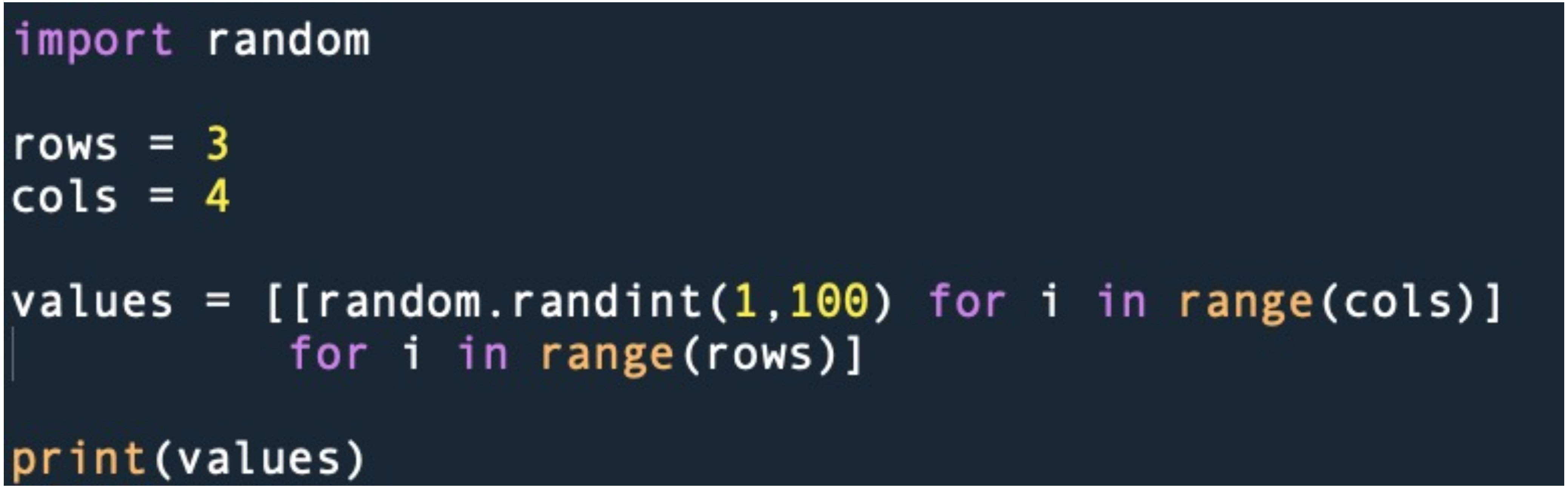
```python
import random

rows = 3
columns = 4

#create list of lists of zeros
values = []
for i in range(rows):
    values += [[0] * columns]

#fill it up with random numbers between [1,100]
for r in range(rows):
    for c in range(columns):
        values[r][c] = random.randint(1, 100)

print(values)
```



```
[[35, 85, 91, 17], [16, 27, 24, 77], [76, 56, 11, 53]]
```

Take 2 with List Comprehension

- Write a program that creates a two-dimensional list with the dimensions of 3 and 4 and fills it with random numbers from1 to 100.



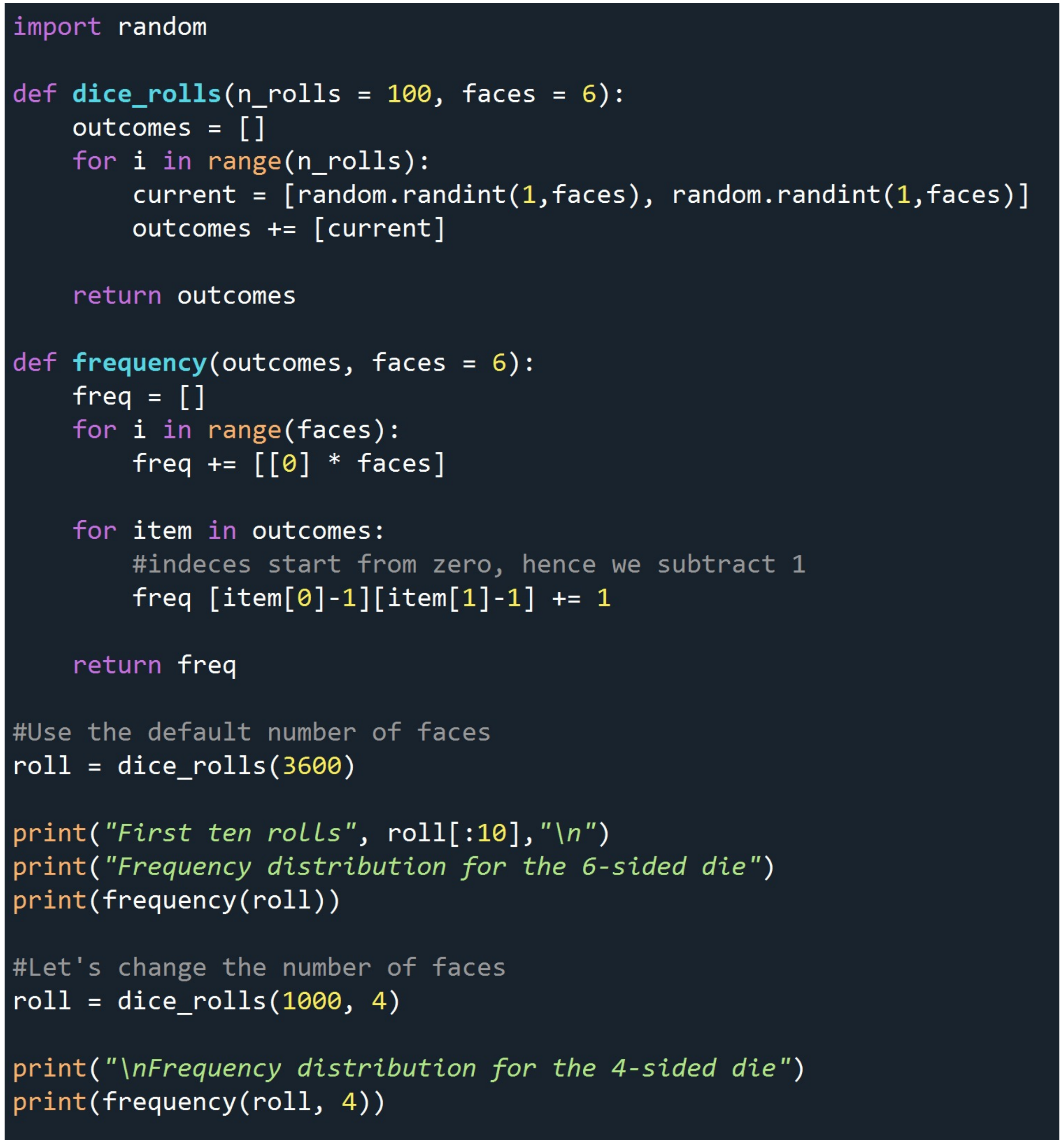
```
import random

rows = 3
cols = 4

values = [[random.randint(1,100) for i in range(cols)]
          for i in range(rows)]

print(values)
```

Example

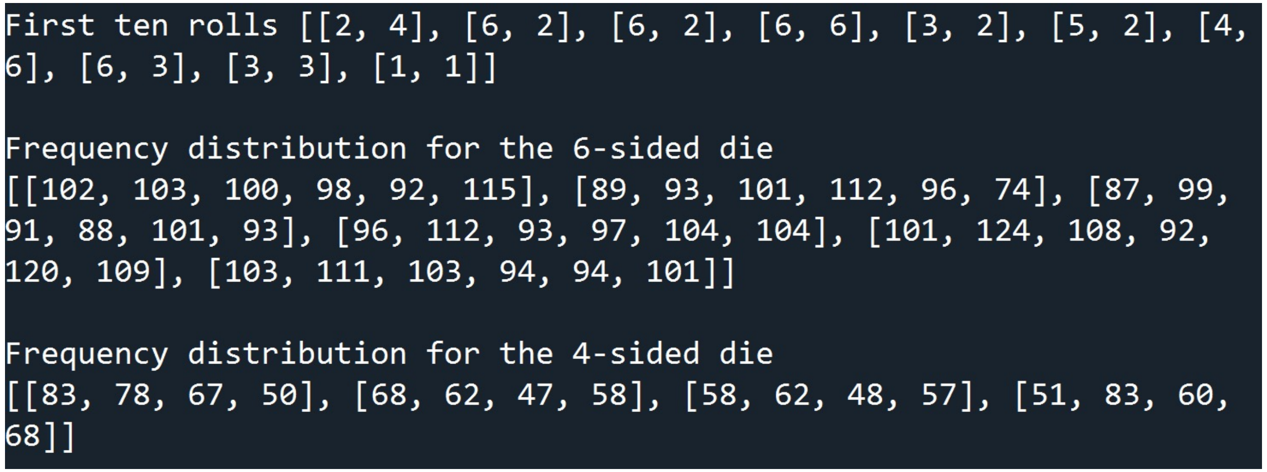
- Write a function (dice_rolls) that emulates n-rolls of a pair of dice
    - It should return the outcome as a list of lists, where each roll's outcome is a list of size two
    - By default n-rolls should be 100, and number of faces should be 6

- Write a function (frequency) that takes the outcome of dice_rolls and number of faces (by default 6) as input

- It should return the frequency of occurrence for each outcome



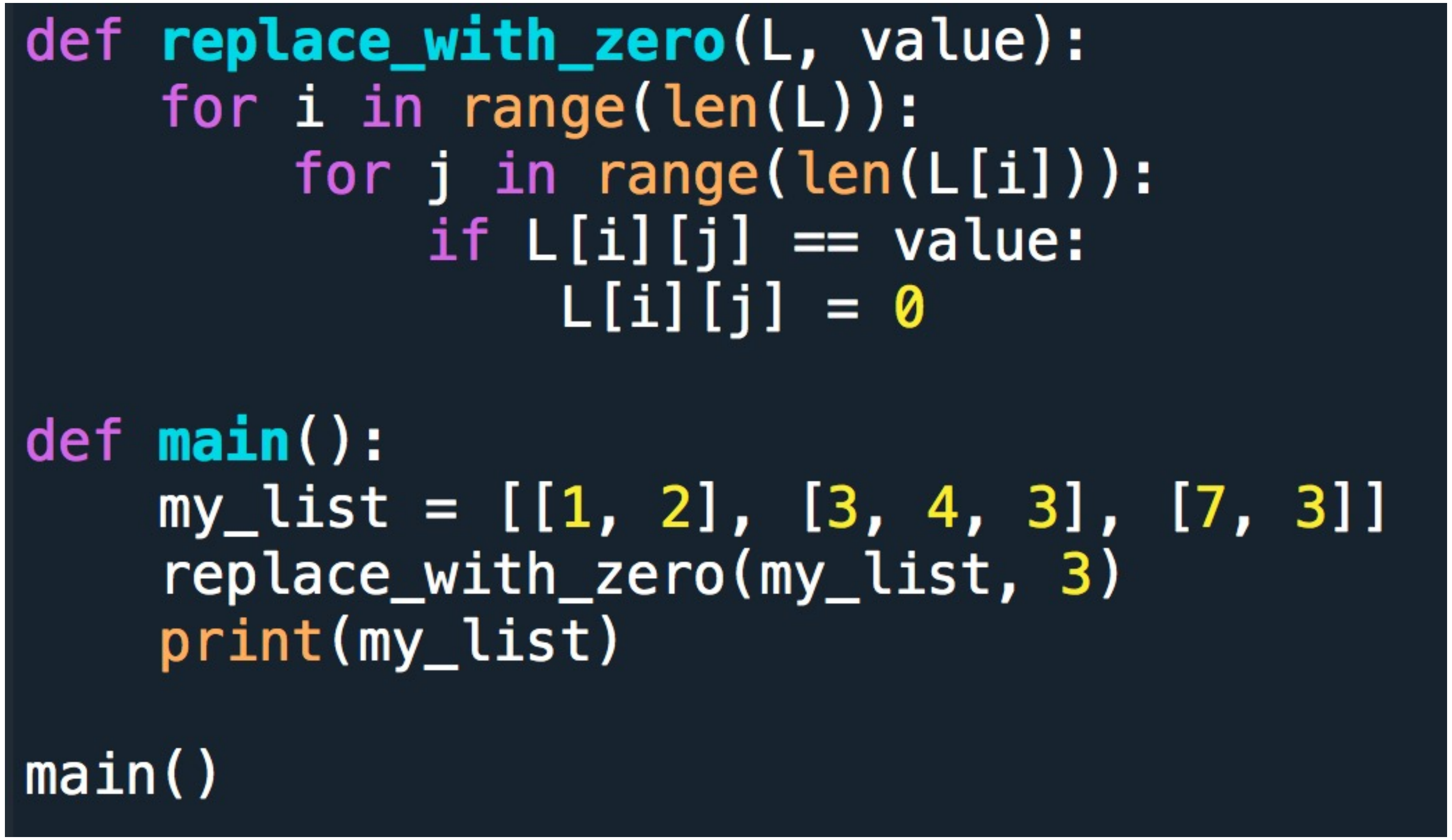
```python
import random

def dice_rolls(n_rolls = 100, faces = 6):
    outcomes = []
    for i in range(n_rolls):
        current = [random.randint(1,faces), random.randint(1,faces)]
        outcomes += [current]

    return outcomes

def frequency(outcomes, faces = 6):
    freq = []
    for i in range(faces):
        freq += [[0] * faces]

    for item in outcomes:
        #indeces start from zero, hence we subtract 1
        freq [item[0]-1][item[1]-1] += 1

    return freq

#Use the default number of faces
roll = dice_rolls(3600)

print("First ten rolls", roll[:10],"\n")
print("Frequency distribution for the 6-sided die")
print(frequency(roll))

#Let's change the number of faces
roll = dice_rolls(1000, 4)

print("\nFrequency distribution for the 4-sided die")
print(frequency(roll, 4))
```



```
First ten rolls [[2, 4], [6, 2], [6, 2], [6, 6], [3, 2], [5, 2], [4,
6], [6, 3], [3, 3], [1, 1]]

Frequency distribution for the 6-sided die
[[102, 103, 100, 98, 92, 115], [89, 93, 101, 112, 96, 74], [87, 99,
91, 88, 101, 93], [96, 112, 93, 97, 104, 104], [101, 124, 108, 92,
120, 109], [103, 111, 103, 94, 94, 101]]

Frequency distribution for the 4-sided die
[[83, 78, 67, 50], [68, 62, 47, 58], [58, 62, 48, 57], [51, 83, 60,
68]]
```

Example

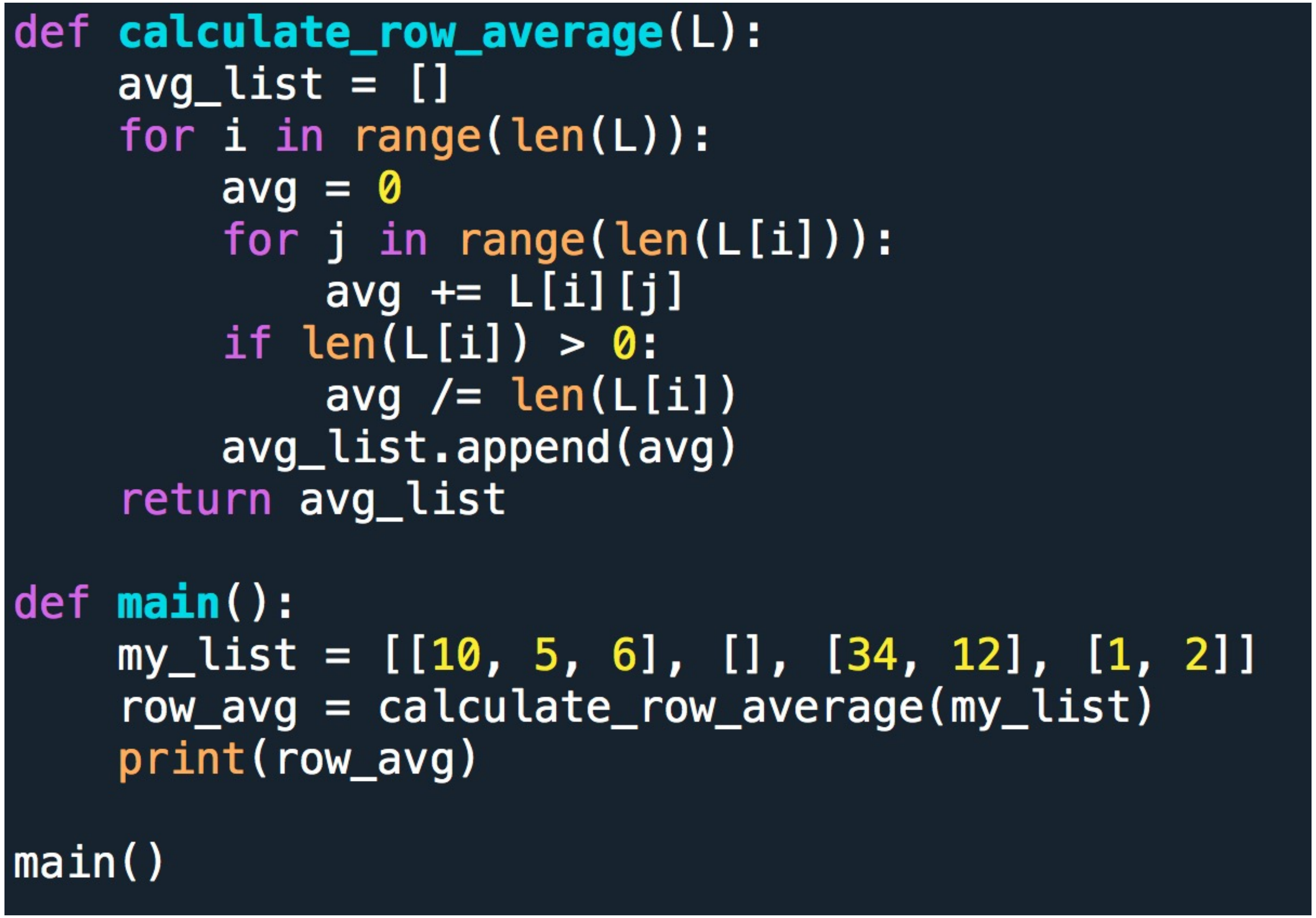
- Write a function that takes a 2D list and a value as its inputs and replaces all items with the given value with 0.



```python
def replace_with_zero(L, value):
    for i in range(len(L)):
        for j in range(len(L[i])):
            if L[i][j] == value:
                L[i][j] = 0

def main():
    my_list = [[1, 2], [3, 4, 3], [7, 3]]
    replace_with_zero(my_list, 3)
    print(my_list)

main()
```

Example

- Write a function that takes a 2D list as its input and returns a 1D list containing the average of each row.



```
def calculate_row_average(L):
    avg_list = []
    for i in range(len(L)):
        avg = 0
        for j in range(len(L[i])):
            avg += L[i][j]
        if len(L[i]) > 0:
            avg /= len(L[i])
        avg_list.append(avg)
    return avg_list

def main():
    my_list = [[10, 5, 6], [], [34, 12], [1, 2]]
    row_avg = calculate_row_average(my_list)
    print(row_avg)

main()
```