COL1000
Introduction to Programming

Priyanka Golia

Most (if not all) of the content is borrowed from Prof. Subodh Kumar's slides

Quiz

L =[[10,20],30,40]
allowed or syntax error

L = [[10,20],30,40]
print(L[0][1])

L =[[10,20],30,40]
$L[0][0] = 100$

print(L[0])

L =[(10,20),30,[40,50],$''\,hello''$]
$L[0][0] = 100$

print(L[0])

$letters = ['a','b','c','d','e']$

$print(letters[-2:-3:-1])$

nums = list(range(0,10))

result = ['odd' if x % 2 else 'even' for x in nums if x > 7]

print(result)

Quiz

L =[[10,20],30,40]
allowed or syntax error

Allowed

L = [[10,20],30,40]
print$(L[0][1])$

L =[[10,20],30,40]
$L[0][0] = 100$

print$(L[0])$

L =[(10,20),30,[40,50],$''hello''$]
$L[0][0] = 100$

print$(L[0])$

$letters = ['a','b','c','d','e']$

$print(letters[-2 : -3 : -1])$

nums = list(range(0,10))

result = ['odd' if x % 2 else 'even' for x in nums if x > 7]

print(result)

Quiz

L =[[10,20],30,40]
allowed or syntax error

Allowed

L = [[10,20],30,40]
$print(L[0][1])$

20

L =[[10,20],30,40]
$L[0][0] = 100$

$print(L[0])$

L =[(10,20),30,[40,50],"hello"]
$L[0][0] = 100$

$print(L[0])$

$letters = ['a','b','c','d','e']$

$print(letters[-2 : -3 : -1])$

nums = list(range(0,10))

result = ['odd' if x % 2 else 'even' for x in nums if x > 7]

print(result)

Quiz

L =[[10,20],30,40]
allowed or syntax error

Allowed

L = [[10,20],30,40]
print(L[0][1])

20

L =[[10,20],30,40]
L[0][0] = 100

print(L[0])

[100,20]

L =[(10,20),30,[40,50],"hello"]
L[0][0] = 100

print(L[0])

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

print(letters[-2 : -3 : -1])
```

```
nums = list(range(0,10))

result = ['odd' if x % 2 else 'even' for x in nums if x > 7]

print(result)
```

Quiz

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

allowed or syntax error

Allowed

```
L = [[10,20],30,40]
print(L[0][1])
```

20

```
L =[[10,20],30,40]
L[0][0] = 100

print(L[0])
```

[100,20]

```
L =[(10,20),30,[40,50],"hello"]
L[0][0] = 100

print(L[0])
```

TypeError: 'tuple' object does not support item assignment

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

print(letters[-2 : -3 : -1])
```

```
nums = list(range(0,10))

result = ['odd' if x % 2 else 'even' for x in nums if x > 7]

print(result)
```

Quiz

L =[[10,20],30,40]
allowed or syntax error

Allowed

L = [[10,20],30,40]
print($L[0][1]$)

20

L =[[10,20],30,40]
$L[0][0] = 100$

print($L[0]$)

[100,20]

L =[(10,20),30,[40,50],"*hello*"]
$L[0][0] = 100$

print($L[0]$)

TypeError: 'tuple' object does not support item assignment

$letters = ['a','b','c','d','e']$

$print(letters[-2:-3:-1])$

'd'

nums = list(range(0,10))

result = ['odd' if x % 2 else 'even' for x in nums if x > 7]

print(result)

Quiz

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

allowed or syntax error

Allowed

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

print(L[0][1])

20

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

$L[0][0] = 100$

print(L[0])

[100,20]

L =[(10,20),30,[40,50],"$hello$"]

$L[0][0] = 100$

print(L[0])

TypeError: 'tuple' object does not support item assignment

$letters = ['a','b','c','d','e']$

$print(letters[-2 : -3 : -1])$

'd'

nums = list(range(0,10))

result = ['odd' if x % 2 else 'even' for x in nums if x > 7]

print(result)

'even', 'odd'

```
nums = list(range(0,10))
result = ['odd' if x % 2 else 'even' for x in nums if x > 7]
print(result)
```

ternary expression (also called the conditional expression or inline if).

value_if_true if condition else value_if_false

```
age = 15
msg = "Adult" if age >= 18 else ("Teenager" if age >= 13 else "Child")
print(msg)    # Teenager
```

Nested Ternary

Matrix in Python — list of lists.

matrix = [[1,2,3], [4,5,6], [7,8,9]]

Notice the comma (,) between lists.
Each inner list is an element of outer list.

```
[
  [1,2,3],
  [4,5,6],
  [7,8,9]
]
```

- Number of outer list elements — number of rows.
  That is, len(matrix) is the number of rows.
- Now, each row is a list of elements (inner lists).
  That is, len(matrix[0]) is number of column.
- To access the i^{th} row — matrix[i].
- To access the element at i^{th} row and j^{th} column — matrix[i][j]

Matrix in Python — list of lists.

If you have a flat list and want to reshape it into an n by m matrix (list of lists)

a =[1,2,3,4,5,6,7,8,9,10,11,12] $\longrightarrow$ [[1,2,3,4], [5,6,7,8], [9,10,11,12]]

Matrix in Python — list of lists.

If you have a flat list and want to reshape it into an n by m matrix (list of lists)

a =[1,2,3,4,5,6,7,8,9,10,11,12] $\longrightarrow$ [[1,2,3,4], [5,6,7,8], [9,10,11,12]]

```
n, m = 3,4  # desired matrix dimensions: 3 rows, 4 columns
matrix = [a[i : i + m] for i in range(0,n × m, m)]
print(matrix)
# Output: [[1,2,3,4], [5,6,7,8], [9,10,11,12]]
```

Matrix in Python — list of lists.

Matrix Transpose.

1	2	3
4	5	6
7	8	9

$\longrightarrow$

1	4	7
2	5	8
3	6	9

M[0][0]	M[0][1]	M[0][2]
M[1][0]	M[1][1]	M[1][2]
M[2][0]	M[2][1]	M[2][2]

$\longrightarrow$

M[0][0]	M[1][0]	M[2][0]
M[0][1]	M[1][1]	M[2][1]
M[0][2]	M[1][2]	M[2][2]

Matrix in Python — list of lists.

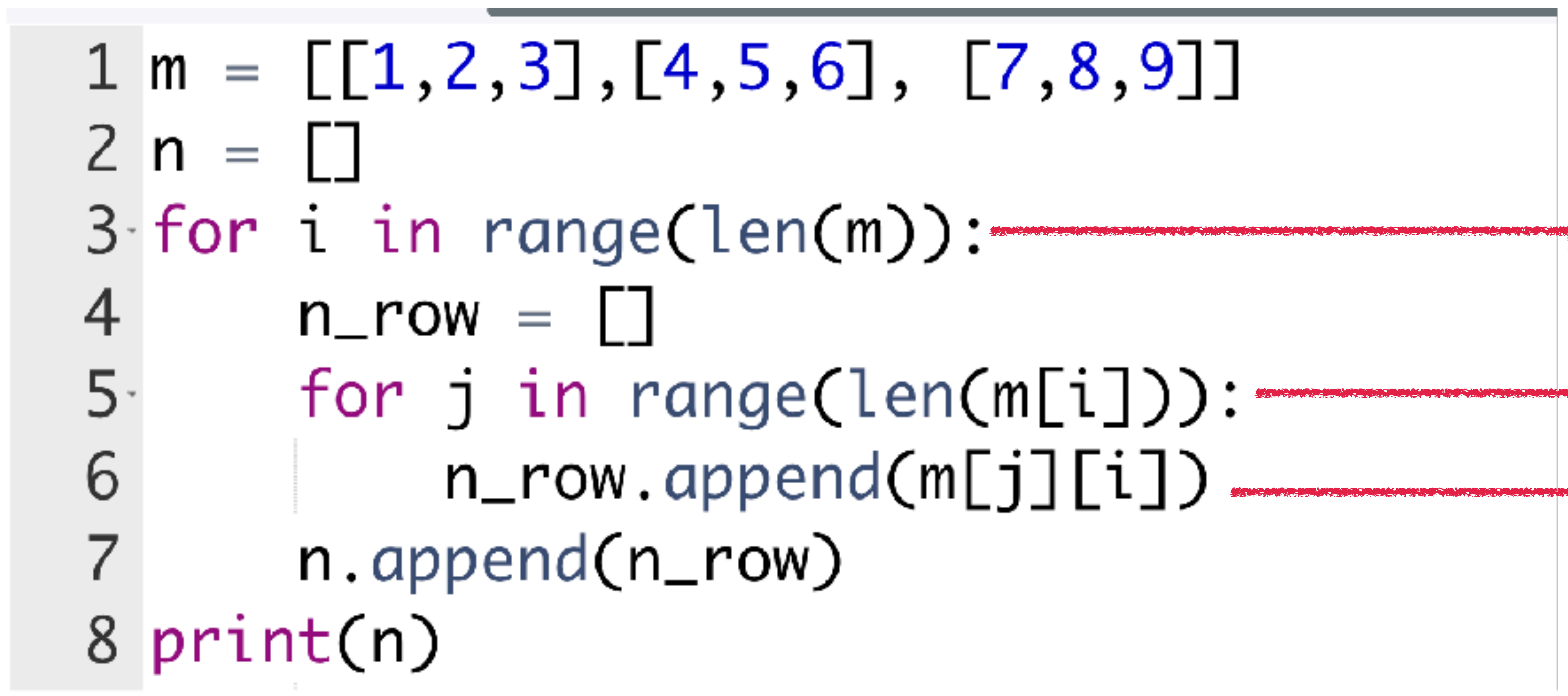
Matrix Transpose.

1	2	3
4	5	6
7	8	9

1	4	7
2	5	8
3	6	9

M[0][0]	M[0][1]	M[0][2]
M[1][0]	M[1][1]	M[1][2]
M[2][0]	M[2][1]	M[2][2]

M[0][0]	M[1][0]	M[2][0]
M[0][1]	M[1][1]	M[2][1]
M[0][2]	M[1][2]	M[2][2]



```python
m = [[1,2,3],[4,5,6], [7,8,9]]
n = []
for i in range(len(m)):
    n_row = []
    for j in range(len(m[i])):
        n_row.append(m[j][i])
    n.append(n_row)
print(n)
```

Number of rows

Number of columns

$M[j][i]$ — for each i, j varies from 0-2.

Matrix in Python — list of lists.

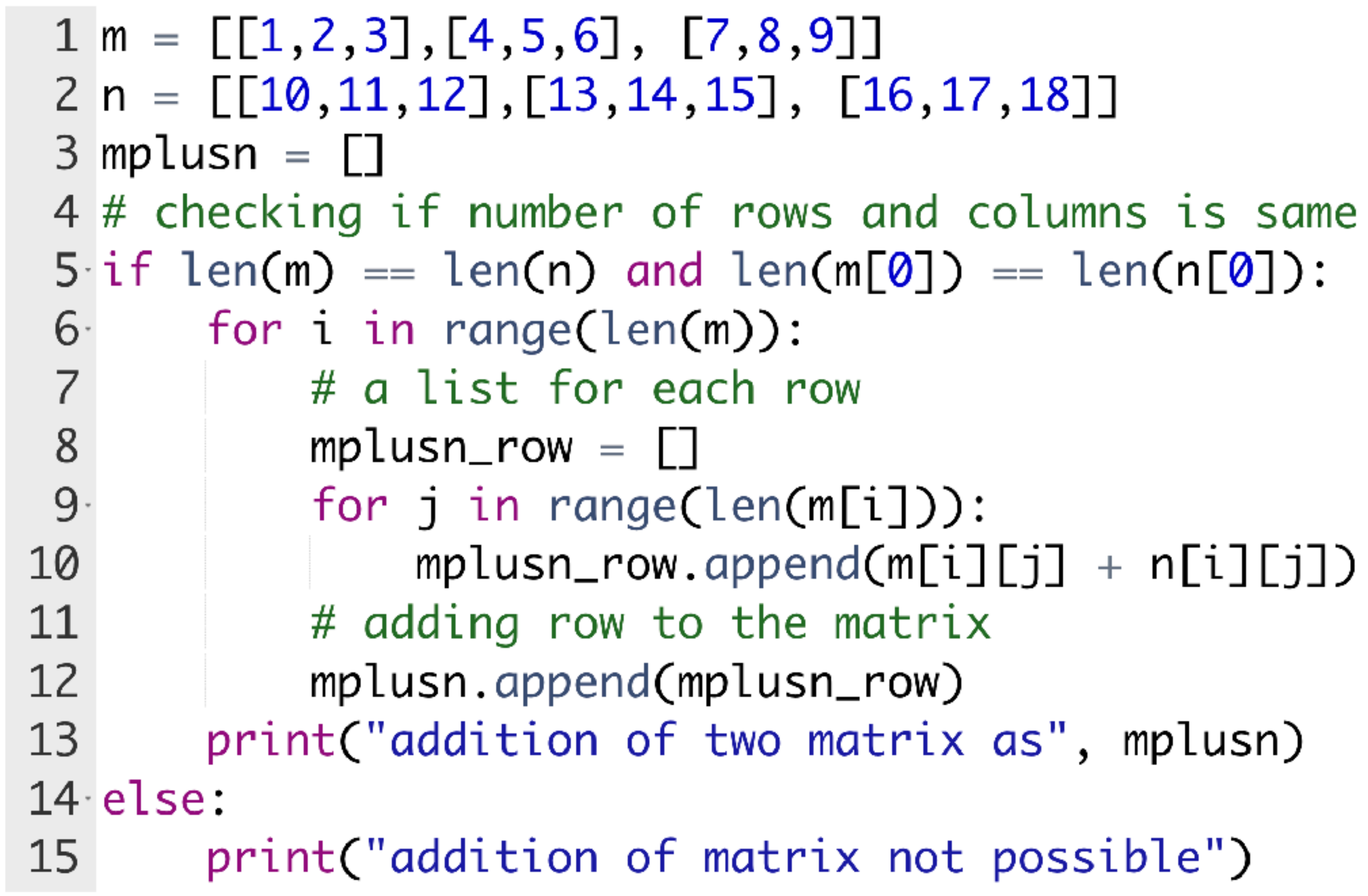
Matrix Transpose. Homework exercise:

Check if a given matrix is a symmetric matrix or not.

Symmetric matrix is **a square matrix** that is equal to its transpose, meaning **its elements are mirrored across the main diagonal**

Matrix in Python — list of lists.

Addition of Matrix



```python
m = [[1,2,3],[4,5,6], [7,8,9]]
n = [[10,11,12],[13,14,15], [16,17,18]]
mplusn = []
# checking if number of rows and columns is same
if len(m) == len(n) and len(m[0]) == len(n[0]):
    for i in range(len(m)):
        # a list for each row
        mplusn_row = []
        for j in range(len(m[i])):
            mplusn_row.append(m[i][j] + n[i][j])
        # adding row to the matrix
        mplusn.append(mplusn_row)
    print("addition of two matrix as", mplusn)
else:
    print("addition of matrix not possible")
```

Inside “if” only if number of rows and columns are same

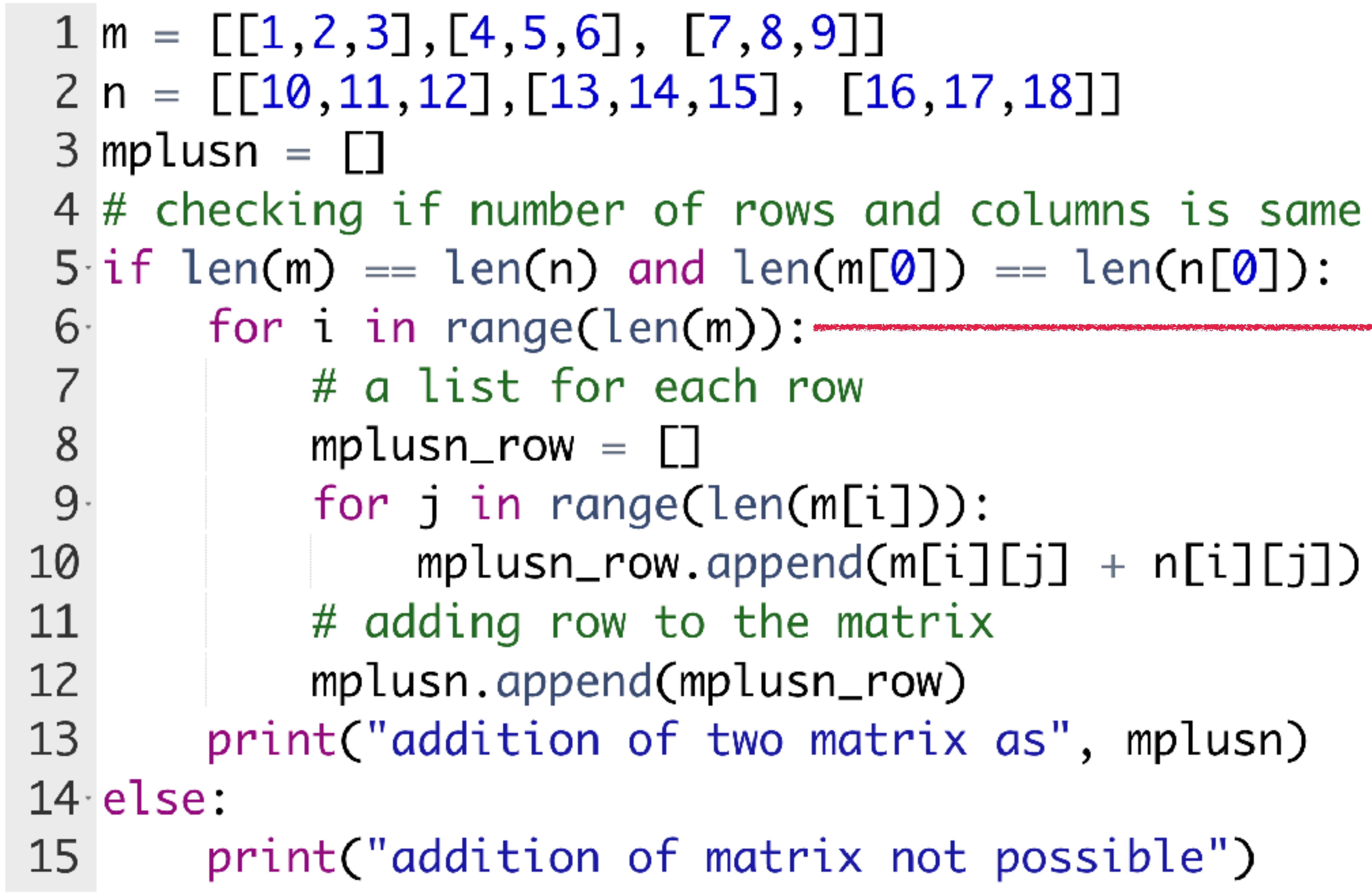
Creating rows for addition matrix

Outside inner loop

Outside outer loop

Matrix in Python — list of lists.

Addition of Matrix



```python
m = [[1,2,3],[4,5,6], [7,8,9]]
n = [[10,11,12],[13,14,15], [16,17,18]]
mplusn = []
# checking if number of rows and columns is same
if len(m) == len(n) and len(m[0]) == len(n[0]):
    for i in range(len(m)):
        # a list for each row
        mplusn_row = []
        for j in range(len(m[i])):
            mplusn_row.append(m[i][j] + n[i][j])
        # adding row to the matrix
        mplusn.append(mplusn_row)
    print("addition of two matrix as", mplusn)
else:
    print("addition of matrix not possible")
```

Inside “if” only if number of rows and columns are same

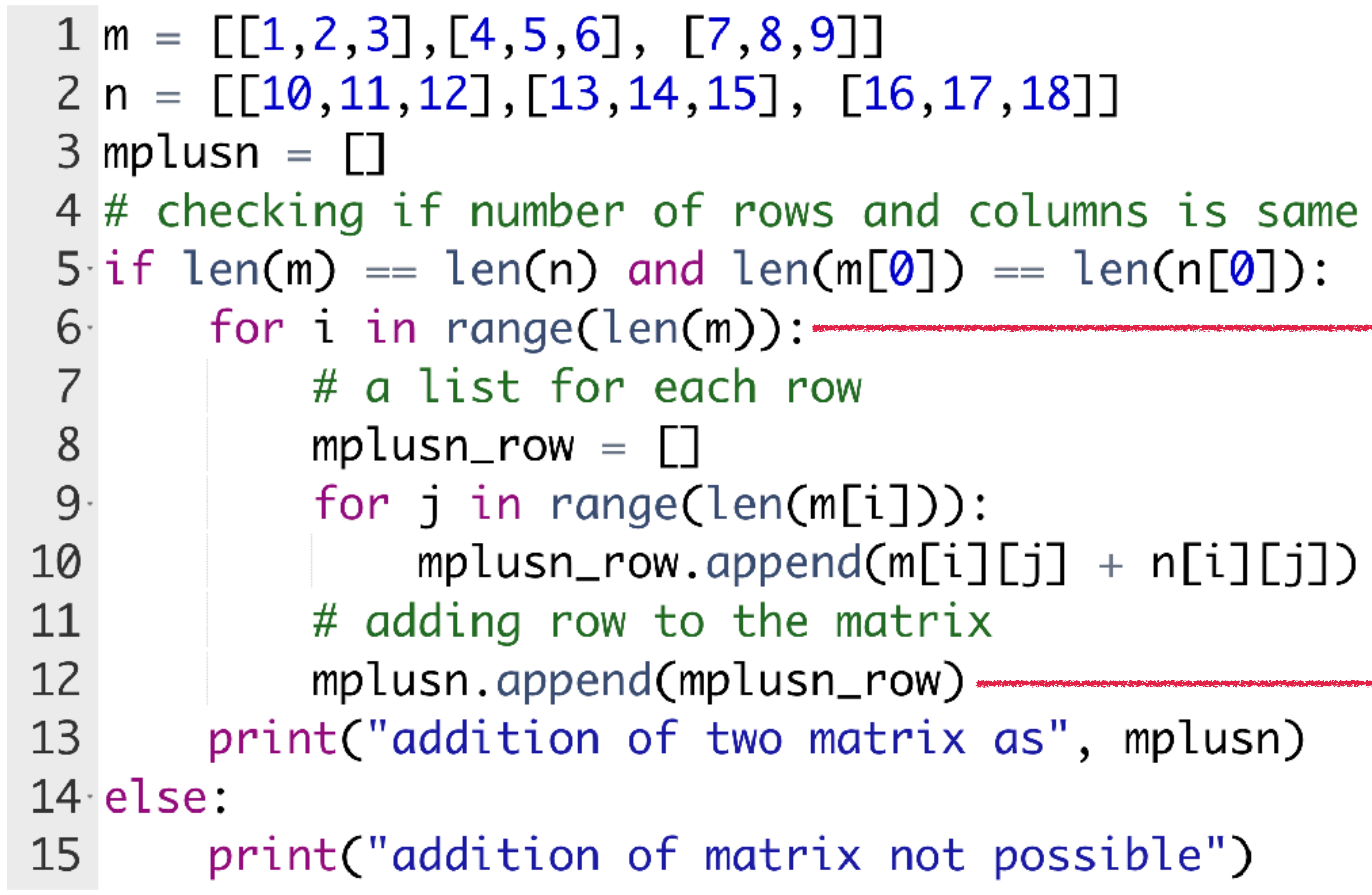
Creating rows for addition matrix

Outside inner loop

Outside outer loop

Matrix in Python — list of lists.

Addition of Matrix



```python
m = [[1,2,3],[4,5,6], [7,8,9]]
n = [[10,11,12],[13,14,15], [16,17,18]]
mplusn = []
# checking if number of rows and columns is same
if len(m) == len(n) and len(m[0]) == len(n[0]):
    for i in range(len(m)):
        # a list for each row
        mplusn_row = []
        for j in range(len(m[i])):
            mplusn_row.append(m[i][j] + n[i][j])
        # adding row to the matrix
        mplusn.append(mplusn_row)
    print("addition of two matrix as", mplusn)
else:
    print("addition of matrix not possible")
```

Inside "if" only if number of rows and columns are same

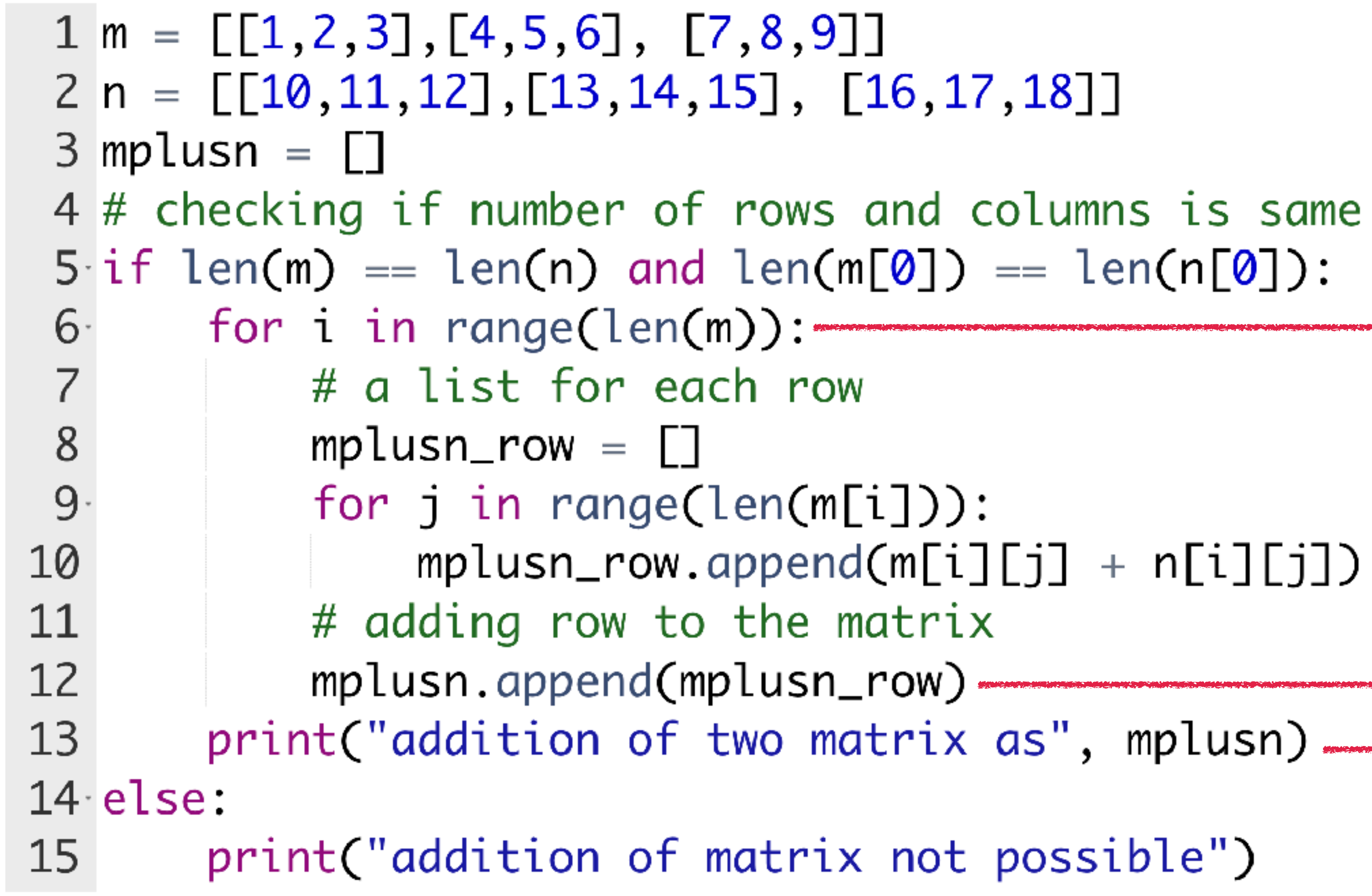
Creating rows for addition matrix

Outside inner loop

Outside outer loop

Matrix in Python — list of lists.

Addition of Matrix



```python
m = [[1,2,3],[4,5,6], [7,8,9]]
n = [[10,11,12],[13,14,15], [16,17,18]]
mplusn = []
# checking if number of rows and columns is same
if len(m) == len(n) and len(m[0]) == len(n[0]):
    for i in range(len(m)):
        # a list for each row
        mplusn_row = []
        for j in range(len(m[i])):
            mplusn_row.append(m[i][j] + n[i][j])
        # adding row to the matrix
        mplusn.append(mplusn_row)
    print("addition of two matrix as", mplusn)
else:
    print("addition of matrix not possible")
```

Inside “if” only if number of rows and columns are same

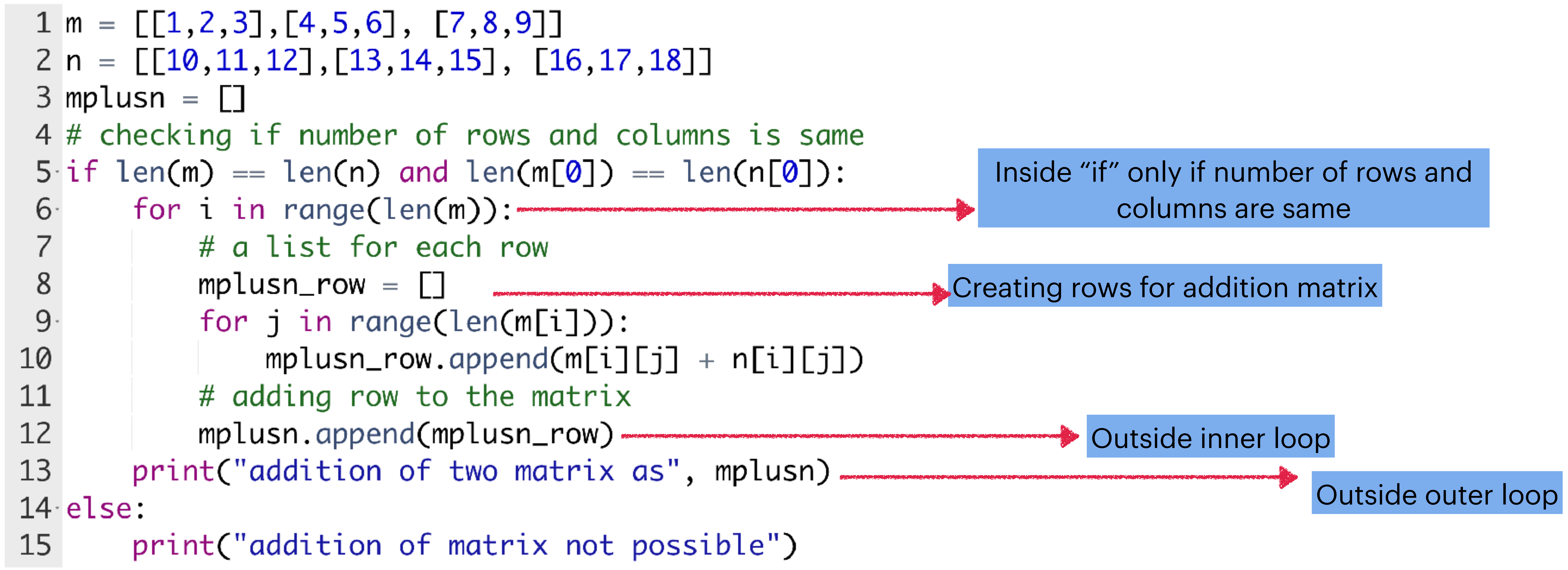
Creating rows for addition matrix

Outside inner loop

Outside outer loop

Matrix in Python — list of lists.

Addition of Matrix



```
m = [[1,2,3],[4,5,6], [7,8,9]]
n = [[10,11,12],[13,14,15], [16,17,18]]
mplusn = []
# checking if number of rows and columns is same
if len(m) == len(n) and len(m[0]) == len(n[0]):
    for i in range(len(m)):
        # a list for each row
        mplusn_row = []
        for j in range(len(m[i])):
            mplusn_row.append(m[i][j] + n[i][j])
        # adding row to the matrix
        mplusn.append(mplusn_row)
    print("addition of two matrix as", mplusn)
else:
    print("addition of matrix not possible")
```

Inside "if" only if number of rows and columns are same
Creating rows for addition matrix
Outside inner loop
Outside outer loop

Deep Copy vs Shallow Copy

Shallow Copy
A shallow copy creates a new outer container object, but does not create copies of the nested (inner) objects. Instead, it copies references to the same inner objects.
→ **Any change to a nested object through the original will also be reflected in the shallow copy.**

Deep Copy
A deep copy creates a new outer container object and recursively creates new copies of all nested objects.
→ **The original and the deep copy become completely independent: changes in one do not affect the other.**

```python
import copy
L = [0,[1,2],3]
print("original list", L)
L1 = L    # could have used L1 = copy.copy(L)
L2 = copy.deepcopy(L)
L[1][0] = 4
L[0] = 5
L.append(6)
print("original list after modification",L)
print("shallow copy",L1)
print("deep copy",L2)
```

Notice how we are creating deep copy.

Shallow copy: new outer object, inner references shared.
Deep copy: new outer object, new inner objects

```
original list [0, [1, 2], 3]
original list after modification [5, [4, 2], 3, 6]
shallow copy [5, [4, 2], 3, 6]
deep copy [0, [1, 2], 3]
```

String Operations

String — again an ordinal container.

Immutable.

Can be indexed, iterated, has len() (and other op.)

Mainly used for multi-line strings.

```
s = '''This is
a string that
spans multiple lines.'''
```

“”, ‘’, ‘’’ ‘’’ Single quotes, double quotes, and triple quotes are allowed.

each character internally represented by a numeric code. print(ord(‘a’)) = 97

How do you have quotes inside string ? Hello ‘hi’ ?

```
S = “Hello \’ hi \’”
print(S)
```

\ is the escape character to designate especial type — “\n” — end of line,
\’ , \” — for quotes.
\\ — for \

String Operations

Method	Description	Example	Output
isupper()	Checks if **all characters** are uppercase	"HELLO".isupper()	TRUE
islower()	Checks if **all characters** are lowercase	"hello".islower()	TRUE
isdecimal()	Checks if **all characters** are decimal digits	"123".isdecimal()	TRUE
count(x)	Counts occurrences of substring `x`	"banana".count("na")	2
find(x)	Finds **first index** of substring `x` **(`-1` if not found)**	"banana".find("na")	2
find(x, start)	Start searching from `start` index	"banana".find("na", 3)	4
replace(a, b)	Replace all occurrences of `a` with `b`	"hello".replace("l","x")	"hexxo"
replace(a, b, n)	Replace only first `n` occurrences	"hello".replace("l","x",1)	"hexlo"
title()	Capitalizes first letter of each word	"hello world".title()	"Hello World"

String Operations

Method	Example	Output	Explanation
strip()	" hello “.strip()	"hello"	Default: removes whitespace from both ends
strip(chars)	"hello".strip("ho")	"ell"	Removes all `h`/`o` from both ends
strip(chars)	"hello".strip("lo")	"he"	Strips `l` and `o` from both ends
lstrip(chars)	"hello".lstrip("he")	"llo"	Removes `h` and `e` only from **left**
lstrip(chars)	"hello".lstrip("oleh")	""	Strips `h,e,l,o` set, leaves nothing
rstrip(chars)	"hello".rstrip("elh")	“hello"	right blocked with ‘o’
split()	"a b c".split()	['a', 'b', 'c']	Default splits on any whitespace, ignores extra
split(sep)	"a,b,c".split(",")	['a','b','c']	Split on `,`
split(sep, n)	"a,b,c".split(",",1)	['a','b,c']	Only 1 split done
join(list)	",".join(["a","b","c"])	"a,b,c"	Joins with `,`
join(tuple)	"-".join(("x","y","z"))	"x-y-z"	Works with tuples/iterables