

Information, Computation, Communication

Learning Python

Loops – Part II

Agenda

- `enumerate()`
- Nested loops
- Early exit or interrupting loops
 - `break`
 - `continue`
- Nested lists
 - Matrices
 - Nested list comprehension

Next: **Functions (Intro)** 

Function Enumerate

enumerate()

- ...is a built-in Python function that returns not only the **element** of a list but also its corresponding **index** in the list

```
for index, value in enumerate(my_list): # traverse the list
    # We can now do some computation involving
    # the element index and/or value
```

Example: enumerate()

- Write a piece of code that reads a list of days in the week and outputs the **index+1** and the **value** (a string) for each of the first five days; you can assume that the list contains all weekdays

```
my_list = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]

for index, value in enumerate(my_list): # traverse the list

    if index < 5: # focus on the first five elements
        print(index + 1, value, end="; ") # print

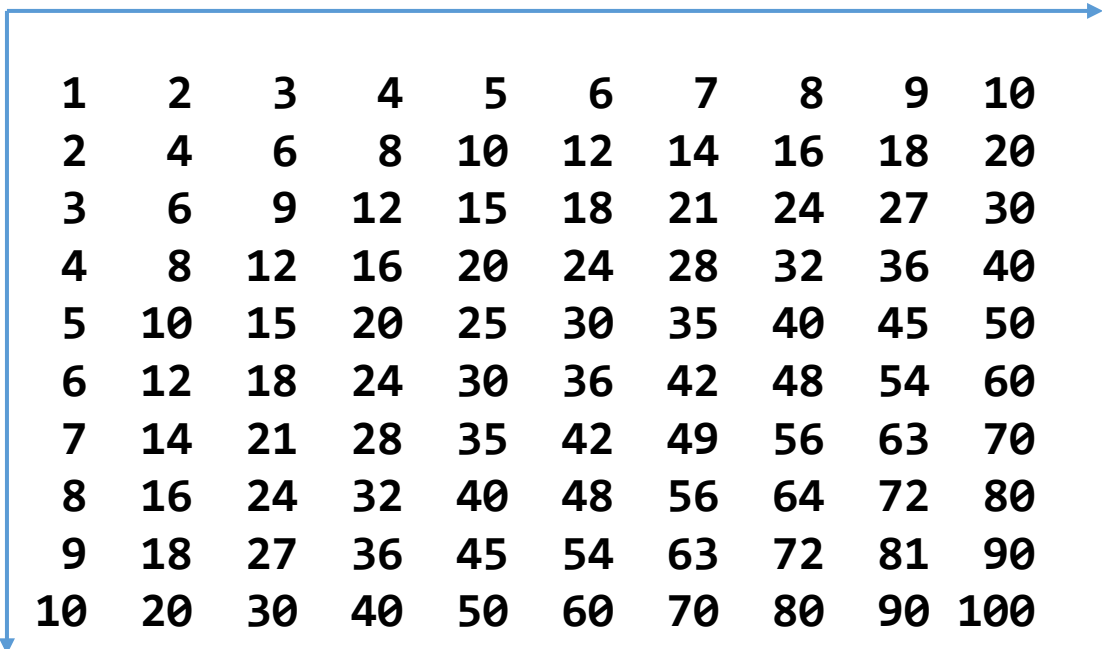
# Output:
# 1 Mon; 2 Tue; 3 Wed; 4 Thu; 5 Fri;
```

Nested Loops

Nested Loops

- ...are loops that contain other loops
- Example: **Multiplication table**

Columns = 0, 1, 2, ..., 9



1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Rows = 0, 1, 2, ..., 9

Example: Multiplication Table

Outer loop (over rows) :

- **$r = 0$**

- **Inner loop (over columns)**

- Inner $c = 0, 1, 2, 3, \dots, 8, 9$

- Output: $(r+1) * (c+1)$

- **$r = 1$**

- Inner $c = 0, 1, 2, 3, \dots, 8, 9$

- Output: $(r+1) * (c+1)$

...

- **$r = 9$**

- Inner $c = 0, 1, 2, 3, \dots, 8, 9$

- Output: $(r+1) * (c+1)$

$c = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Example: Multiplication Table

Outer loop

Inner loop

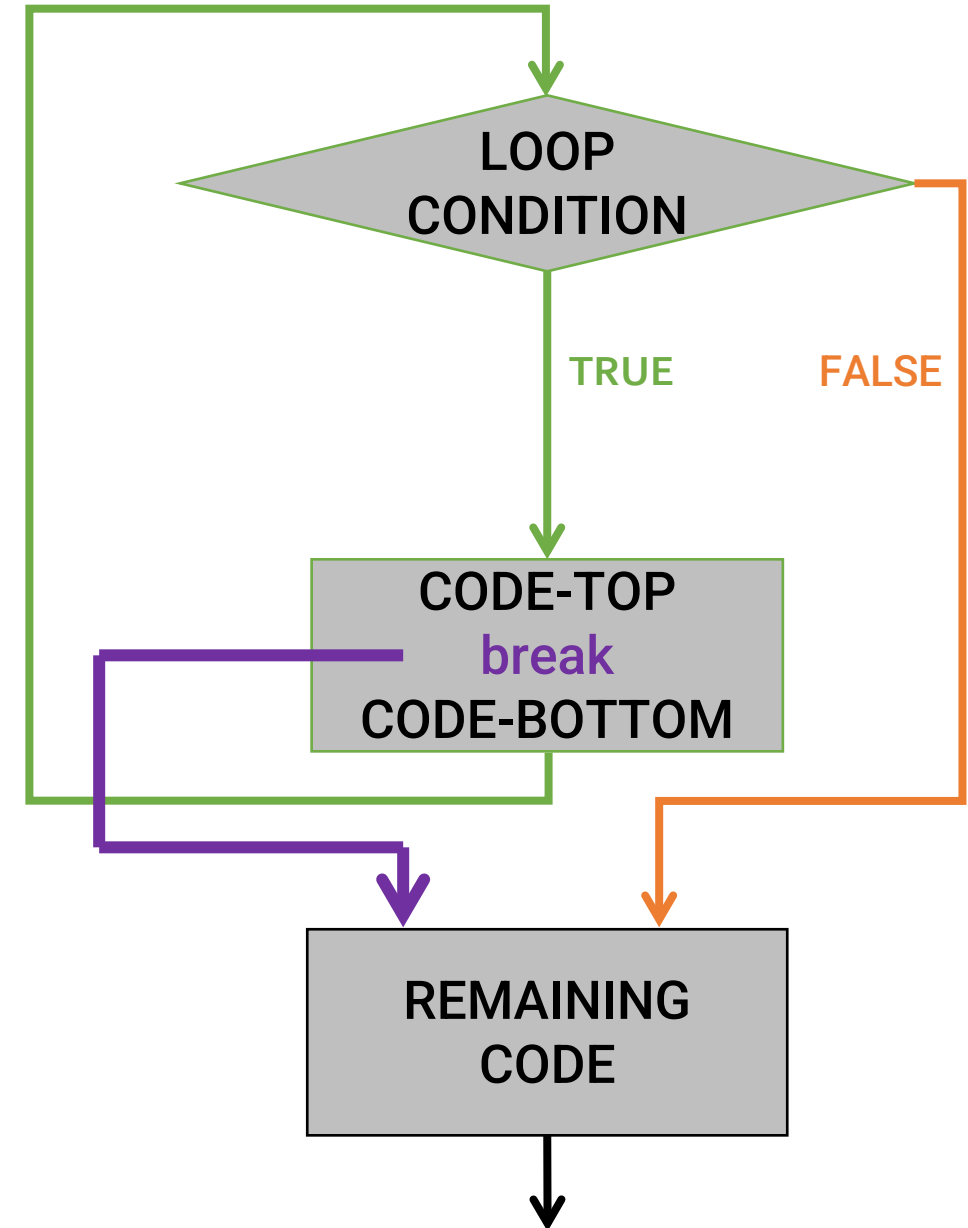
```
for r in range(10):  
    for c in range(10):  
        v = (r + 1) * (c + 1)  
        # Print the result 'v', formatted to take up to 4 spaces  
        # for alignment, and stay on the same line  
        print(f"{v:4}", end=" ")  
  
    # Move to the next line after completing a row of the table  
    print()
```

Early Exit from Loops

- break

Loops: Early Exit

- **break** keyword is used to terminate the loop before its end is due
- Can be used in **while** and **for**
- If used inside a nested loop, **break** affects the **closest** of the loops

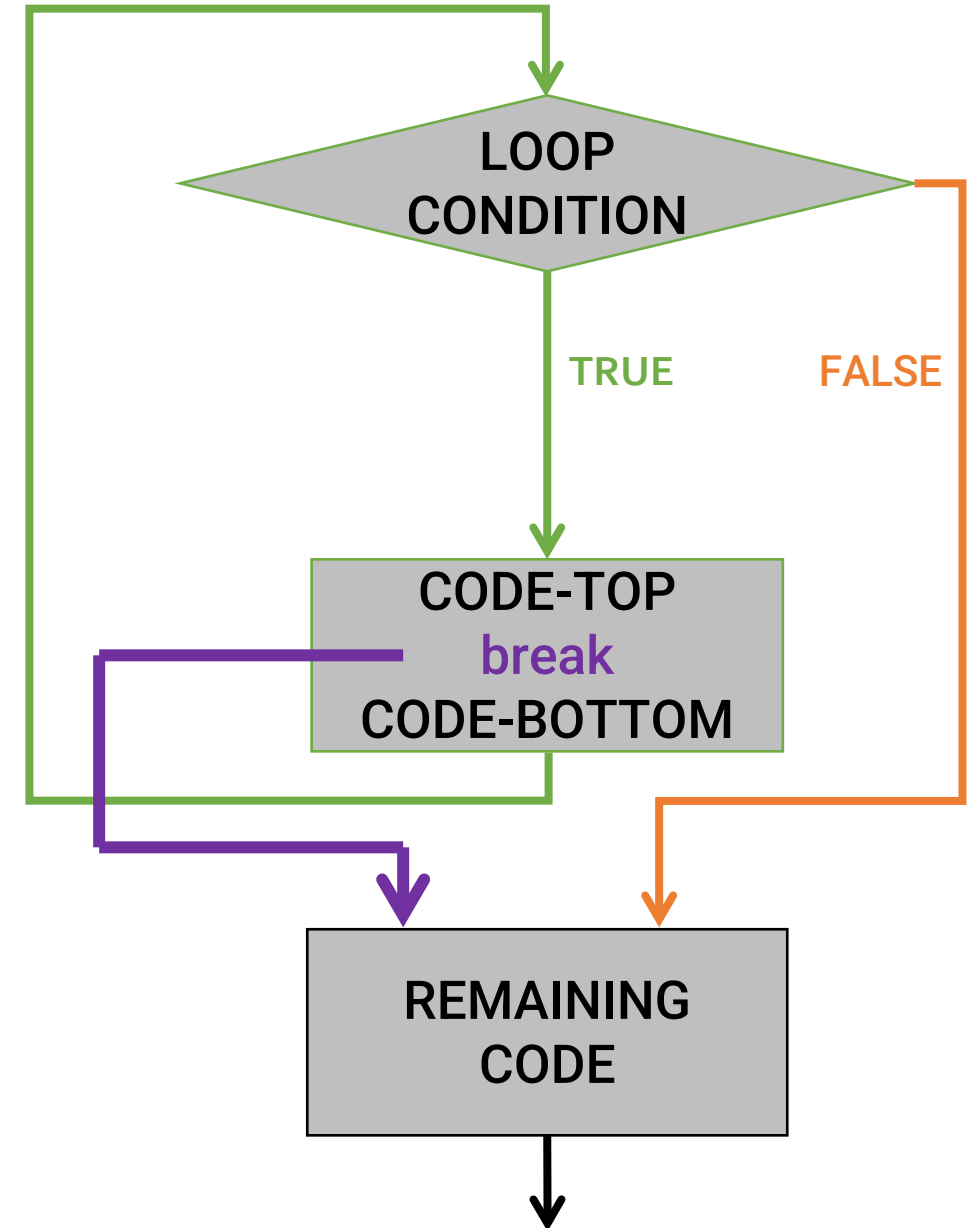


Loops: Early Exit

```
# while or for loop
while loop-again:
    # If loop-again is true,
    # repeat the loop
    code-top

    # If should-exit-early is true,
    # end the loop
    if should-exit-early:
        break

    code-bottom
```



EXAMPLES

Example: $in_list = [9, 10, 14, 17, 25, 28, 29, 44, 46, 54, 56, 57, 59, 61, 64, 71, 74, 90, 94, 95]$

Output: `out_list = [9, 10, 14, 17, 25, 28, 29, 44, 46]`

13

Example: Early Exit from Loops

```
# Create an empty list to store the output
out_list = []

# Iterate through each element in the sorted input list
for elem in in_list:

    # Check if the current element exceeds the threshold
    if elem > threshold:
        # If it does, exit the loop
        break
    out_list.append(elem) # copy

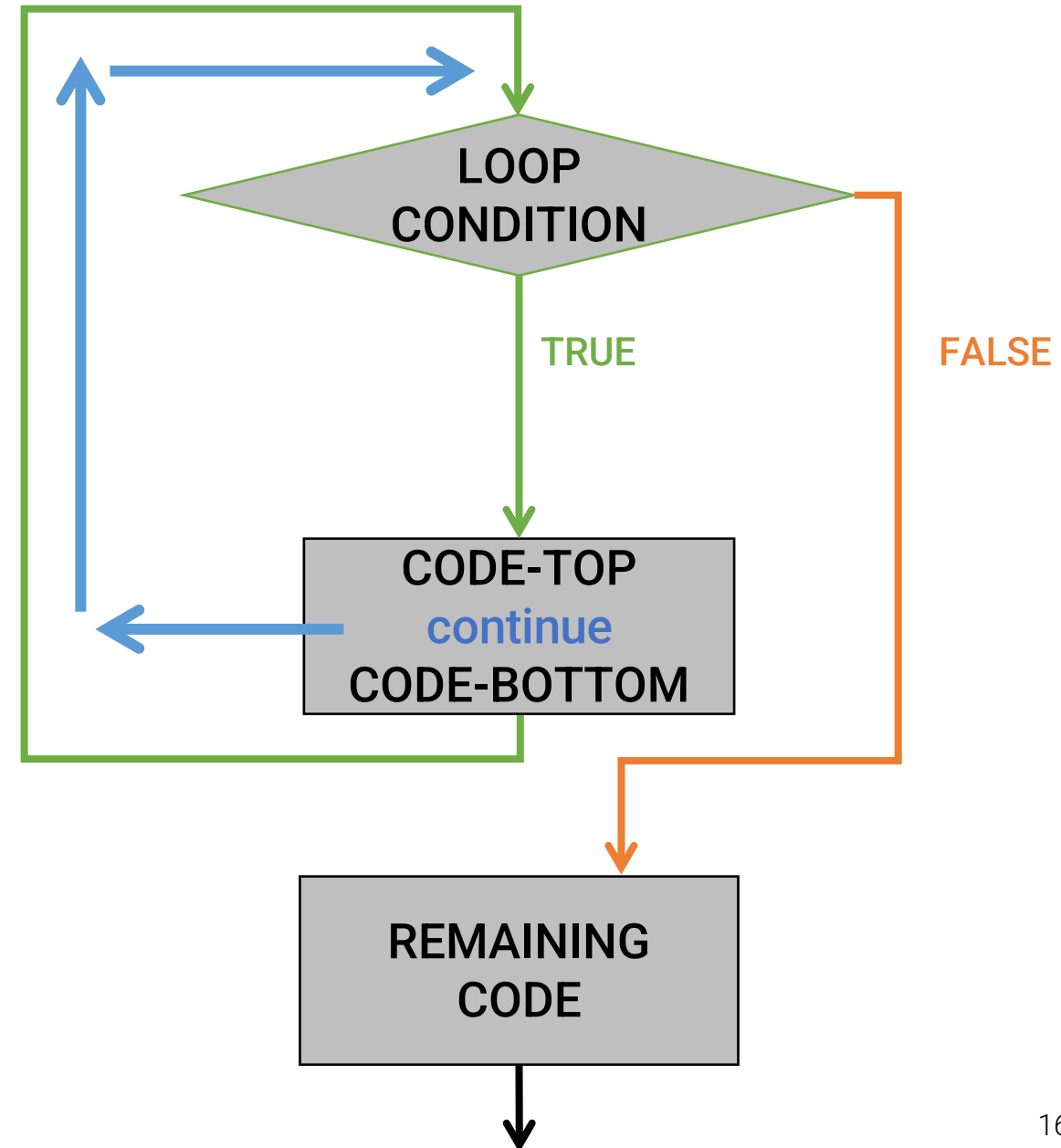
# Print
print(out_list)
print(f"Output list has {len(out_list)} elements.")
```

Skipping Loop Iterations

- `continue`

Loops: Skipping (Part of) an Iteration

- **continue** keyword is used to **interrupt** the current loop iteration and continue the loop by starting the **next** iteration provided that the loop condition is still True
- If used inside a nested loop, **continue** affects the **closest** of the loops

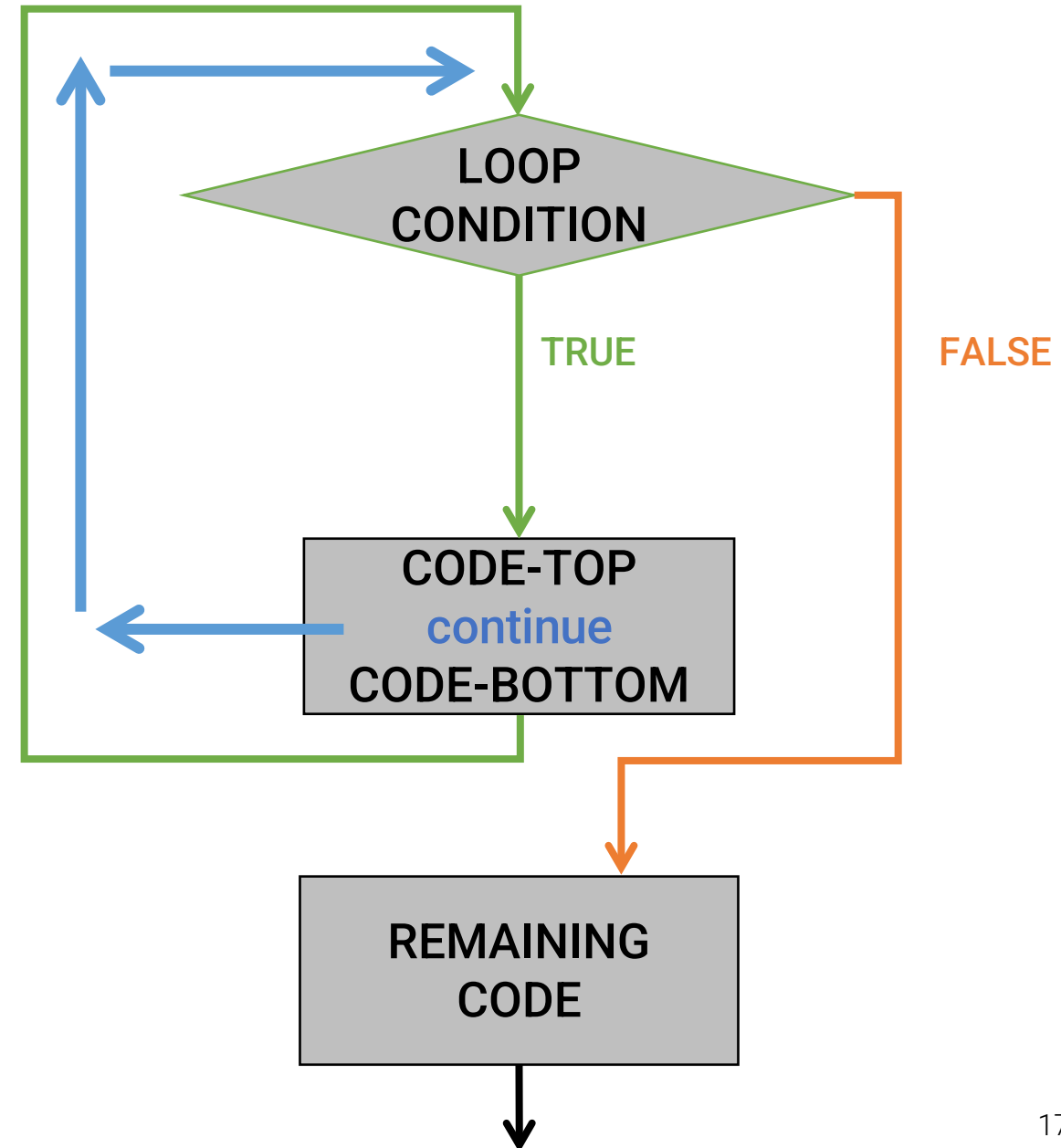


Loops: Skipping (Part of) an Iteration

```
# while or for loop
while loop-again:
    # If loop-again is true,
    # repeat the loop
    code-top

    # If should-skip is true,
    # skip code-bottom
    if should-skip:
        continue

    code-bottom
```



EXAMPLES

Example: `in_list = [9, 10, 14, 17, 25, 28, 29, 44, 46, 54, 56, 57, 59, 61, 64, 71, 74, 90, 94, 95]`

Example: Skipping (Part of) an Iteration

```
# Create an empty list to store the output
```

```
out_list = []
```

```
for num in in_list: # Iterate through the list of numbers
```

```
    # If the number is even, skip the remaining instructions
```

```
    # and move to the next iteration
```

```
    if not (num % 2):
```

```
        continue
```

```
    # If the number is odd, append it to the output list
```

```
    out_list.append(num)
```

```
print(out_list)
```

Nested Lists

...and Matrices

Nested Lists

- ... are lists that contain other lists
- Nested lists are how we represent matrices, for example, but they are not limited to two dimensions

Example: Creating a Matrix

- Write a block of code that creates a 5×5 matrix using a nested list, such that each row contains values 0, 1, 2, 3, 4.

```
matrix = [] # Create an empty list
for row in range(5): # For every row
    matrix.append([]) # Append an empty sublist to the list
    for col in range(5): # For every column
        matrix[row].append(col) # Append to the row

print(matrix)
# Output:
# [[0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4],
#   [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]
```

Creating a Matrix using List Comprehension

- Write a block of code that creates a 5×5 matrix using a nested list, such that each row contains values 0, 1, 2, 3, 4.

```
# Nested list comprehension
matrix = [[col for col in range(5)] for row in range(5)]
```

```
print(matrix)
```

```
# Output:
# [[0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4],
#   [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]
```

Example: Filtering Matrices

- Write a block of code that traverses an input matrix and generates a list of odd numbers found in the matrix.

Example: *input matrix:* `[[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

Output: *output list:* `[1, 3, 5, 7, 9]`

Example: Filtering Matrices

- Write a block of code that traverses an input matrix and generates a list of odd numbers found in the matrix.

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
odd_numbers = []
# Loop through each row in the matrix
for row in matrix:
    # Loop through each element in the current row
    for element in row:
        if element % 2:
            odd_numbers.append(element)
print(odd_numbers)
# Output:
# [1, 3, 5, 7, 9]
```

Filtering Matrices Using List Comprehension

- Write a block of code that traverses an input matrix and generates a list of odd numbers found in the matrix.

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

```
odd_numbers = [element for row in matrix  
                for element in row  
                if element % 2]
```

```
print(odd_numbers)
```

```
# Output:
```

```
# [1, 3, 5, 7, 9]
```



Next topic: Intro to Functions