**Day 10 Notes**

**For Loops**

**Example 1: range() with a Negative Step**

**Definition:** This demonstrates using a range() function with a negative step (-1) to iterate backward. The loop starts at 10 and counts down to (but not including) 0. The second part shows the same concept using variables to define the start and end points.

**Example Code:**

Python

a = 5

for i in range(10,0,-1):

print(i, end=" ")

a = 5

for i in range((a+5),(a-5),-1):

print(i, end=" ")

**Output:**

Plaintext

10 9 8 7 6 5 4 3 2 1 10 9 8 7 6 5 4 3 2 1

**Example 2: Iterating Over a String**

**Definition:** A for loop can iterate directly over the characters of a string. The end=" " parameter in the print() function prints each character on the same line, separated by a space.

**Example Code:**

Python

a = 'Omkar'

for i in a:

print(i, end=" ")

**Output:**

Plaintext

O m k a r

**Example 3: range() with len()**

**Definition:** This loop iterates using range() based on the length of the string a. len(a) is 5. The range is (1, (5-2), 1), which simplifies to range(1, 3, 1). The loop, therefore, prints the numbers 1 and 2.

**Example Code:**

Python

a = 'omkar'

for i in range(1,(len(a)-2),1):

print(i,end=" ")

**Output:**

Plaintext

1 2

**Example 4: TypeError in Loop**

**Definition:** This code demonstrates a common error. The loop iterates through the string a = 'omkar'. In the first iteration, i is the string 'o'. The line i += 1 tries to add an integer (1) to a string ('o'), which is not a valid operation and raises a TypeError.

**Example Code:**

Python

a = 'omkar'

for i in a:

print(i)

i += 1

**Output:**

Plaintext

o

---------------------------------------------------------------------------

TypeError Traceback (most recent call last)

<ipython-input-1-04286d7515ec> in <cell line: 2>()

2 for i in a:

3 print(i)

----> 4 i += 1

TypeError: can only concatenate str (not "int") to str

**Example 5: Modifying the Loop Variable**

**Definition:** This example shows that modifying the loop variable (i) inside the loop does not affect the sequence generated by range(). The range(1, 5, 1) generates 1, 2, 3, 4. In each pass, i is set (e.g., to 1), then modified (i += 2 makes it 3), and then printed. On the *next* pass, i is set to the *next* value from the range (2).

**Example Code:**

Python

a = 5

for i in range(1,5,1):

i += 2

print(i)

**Output:**

Plaintext

3

4

5

6

**Example 6: Basic for Loop (Repetition)**

**Definition:** A simple for loop that iterates through the sequence generated by range(1, 10) (numbers 1 through 9) and prints each number on a new line.

**Example Code:**

Python

# make the loop for above code

for i in range(1,10):

print(i)

**Output:**

Plaintext

1

2

3

4

5

6

7

8

9

**Loop in the loop (Nested Loops)**

**Example 1: Square Pattern**

**Definition:** This is a nested loop. The outer loop (i) runs 4 times (for rows, as range(1, 5) goes 1, 2, 3, 4). For each row, the inner loop (j) also runs 4 times, printing a \* each time. The print() at the end of the outer loop moves the cursor to the next line.

**Example Code:**

Python

a = 5

for i in range(1,a,1):

for j in range(1,a,1):

print('\*',end =" ")

print()

**Output:**

Plaintext

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* ```

#### Example 2: Right-Angle Triangle (File Discrepancy)

\*\*Definition:\*\* This code is written to print a right-angle triangle of stars. The inner loop `j` runs from 1 up to `i+1`, increasing the number of items printed in each row.

\*\*Note on File Discrepancy:\*\* The code in your file is `print('\*',end =" ")`, but the saved output in the file shows numbers. This means the cell was likely run with `print(j, end=" ")`, saved, and then the code was changed to `print('\*', ...)` without re-running.

\*\*Example Code (as written in file):\*\*

```python

a = 5

for i in range(1,a,1):

for j in range(1,i+1,1):

print('\*',end =" ")

print()

**Output (as saved in file):**

Plaintext

1

1 2

1 2 3

1 2 3 4

**Example 3: Inverted Right-Angle Triangle**

**Definition:** This pattern is created by having the inner loop (j) run from the outer loop's current value (i) up to the maximum (a). As i (which goes 1, 2, 3, 4) increases, the inner loop runs fewer times.

**Example Code:**

Python

a = 5

for i in range(1,a,1):

for j in range(i,a ,1):

print('\*',end =" ")

print()

**Output:**

Plaintext

\* \* \* \* \* \* \* \* \* \* ```

#### Example 4: Inverted Number Triangle (Outer Variable `i`)

\*\*Definition:\*\* This uses the same loop structure as Example 3, but prints the value of the \*\*outer loop variable `i`\*\* in each step of the inner loop.

\*\*Example Code:\*\*

```python

a = 5

for i in range(1,a,1):

for j in range(i,a ,1):

print(i,end =" ")

print()

**Output:**

Plaintext

1 1 1 1

2 2 2

3 3

4

**Example 5: Inverted Number Triangle (Inner Variable j)**

**Definition:** This also uses the inverted triangle structure, but prints the value of the **inner loop variable j**, which increments.

**Example Code:**

Python

a = 5

for i in range(1,a,1):

for j in range(i,a ,1):

print(j,end =" ")

print()

**Output:**

Plaintext

1 2 3 4

2 3 4

3 4

4

**Example 6: Pyramid Pattern (File Discrepancy)**

**Definition:** This pattern uses two inner loops. The first (j) prints leading spaces to create the indentation. The second (k) prints the \* characters.

**Note on File Discrepancy:** The code in your file prints a space (print(" ",end= " ")), but the saved output shows underscores (\_). This is likely because the cell was run with print("\_", ...) and then the code was changed to print(" ", ...) without re-running.

**Example Code (as written in file):**

Python

a = 5

for i in range (1,a,1):

for j in range(i,a,1):

print(" ",end= " ")

for k in range(1,i+1):

print("\*",end=" ")

print()

**Output (as saved in file):**

Plaintext

\_ \_ \_ \_ \* \_ \_ \_ \* \* \_ \_ \* \* \* \_ \* \* \* \* ```

#### Example 7: Right-Angle Triangle

\*\*Definition:\*\* This is the correct code for a standard right-angle triangle. The inner loop `j` runs from 1 to `i+1`, printing an increasing number of `\*` symbols for each row.

\*\*Example Code:\*\*

```python

a = 5

for i in range(1,a,1):

for j in range(1,i+1,1):

print('\*',end =" ")

print()

**Output:**

Plaintext

\* \* \* \* \* \* \* \* \* \* ```

#### Example 8: Hourglass Pattern (Homework)

\*\*Definition:\*\* This code combines two sets of nested loops. The first set (loops `i` and `j`) prints an inverted triangle (like Example 3). The second set (loops `k` and `l`) prints a regular right-angle triangle (like Example 7).

\*\*Example Code:\*\*

```python

# Home work

a = 5

for i in range(1,a,1):

for j in range(i,a,1):

print('\*',end =" ")

print()

for k in range(1,a,1):

for l in range(1,k+1,1):

print('\*',end =" ")

print()

**Output:**

Plaintext

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* ```

<br>

## \*\*Day 11 Notes (Corrected)\*\*

\*(Based on `Py Day 11.ipynb`)\*

---

### \*\*Nested Loop Patterns (Continued)\*\*

#### Example 1: Hourglass Pattern with Underscore

\*\*Definition:\*\* This uses the same hourglass loop structure as Day 10, Example 8, but sets `end="\_"` in the `print()` function to use an underscore as a separator.

\*\*Example Code:\*\*

```python

a=5

for i in range(1,a,1):

for j in range(i,a,1):

print('\*',end="\_")

print()

for k in range(1,a,1):

for l in range(1,k+1,1):

print('\*',end="\_")

print()

**Output:**

Plaintext

\*\_\*\_\*\_\*\_

\*\_\*\_\*\_

\*\_\*\_

\*\_

\*\_

\*\_\*\_

\*\_\*\_\*\_

\*\_\*\_\*\_\*\_

**Example 2: Hourglass Pattern with '1' and Underscore**

**Definition:** This again uses the hourglass loop structure, but prints the number 1 followed by an underscore.

**Example Code:**

Python

a=5

for i in range(1,a,1):

for j in range(i,a,1):

print(1,end="\_")

print()

for k in range(1,a,1):

for l in range(1,k+1,1):

print(1,end="\_")

print()

**Output:**

Plaintext

1\_1\_1\_1\_

1\_1\_1\_

1\_1\_

1\_

1\_

1\_1\_

1\_1\_1\_

1\_1\_1\_1\_

**Example 3: Mixed Pattern ('+' and '5')**

**Definition:** This pattern has two inner loops. The first (j) prints an increasing number of + symbols. The second (k) prints a decreasing number of 5 symbols.

**Example Code:**

Python

a=6

for i in range(1,a,1):

for j in range(1,i+1,1):

print("+",end=" ")

for k in range(i,6,1):

print(5,end=" ")

print()

**Output:**

Plaintext

+ 5 5 5 5 5

+ + 5 5 5 5

+ + + 5 5 5

+ + + + 5 5

+ + + + + 5

**Example 4: Mixed Pattern ('+' and Decreasing Numbers)**

**Definition:** This pattern's outer loop (i) counts down from 5 to 1. The first inner loop (j) prints + symbols. The second inner loop (k) prints a decreasing sequence of numbers.

**Example Code:**

Python

for i in range(5,0,-1):

for j in range(5,i-1,-1):

print("+",end=" ")

for k in range(5,5-i,-1):

print(k,end=" ")

print()

**Output:**

Plaintext

+ 5 4 3 2 1

+ + 5 4 3 2

+ + + 5 4 3

+ + + + 5 4

+ + + + + 5

**Example 5: Mixed Number Pattern**

**Definition:** A complex pattern where the first inner loop (j) prints numbers from 1 up to i+1. The second inner loop (k) prints a decreasing sequence of even numbers.

**Example Code:**

Python

a=11

for i in range(0,5,1):

for j in range(1,i+2,1):

print(j,end=" ")

for k in range(2,a-(2\*i)+1,2):

print(k,end=" ")

print()

**Output:**

Plaintext

1 2 4 6 8 10

1 2 2 4 6 8

1 2 3 2 4 6

1 2 3 4 2 4

1 2 3 4 5 2

**Function: chr()**

**Definition:** The chr() function returns the character represented by a specified Unicode number (e.g., 65 is 'A', 66 is 'B'). This is often used to print letters in patterns.

**Example 1: chr() in a Simple Loop**

**Definition:** This loop runs 4 times, printing chr(65) (which is 'A') each time.

**Example Code:**

Python

for i in range(1,5,1):

print(chr(65),end=" ")

**Output:**

Plaintext

A A A A

**Example 2: chr() with a Variable**

**Definition:** This loop runs 4 times. In each pass, i becomes 1, 2, 3, and 4. chr(64+i) evaluates to chr(65), chr(66), etc., printing 'A', 'B', 'C', 'D'.

**Example Code:**

Python

for i in range(1,5,1):

print(chr(64+i),end=" ")

**Output:**

Plaintext

A B C D

**Example 3: chr() in a Nested Loop (Square)**

**Definition:** The outer loop (i) handles rows. The inner loop (j) handles columns, printing 'A', 'B', 'C', 'D' in each row.

**Example Code:**

Python

for i in range (1,5,1):

for j in range(1,5,1):

print(chr(64+j),end=" ")

print()

**Output:**

Plaintext

A B C D

A B C D

A B C D

A B C D

**Example 4: Mixed Pattern ('\*' and Letters)**

**Definition:** The first inner loop (j) prints an increasing number of \* symbols. The second inner loop (k) prints a decreasing sequence of letters in reverse order (D, C, B, A).

**Example Code:**

Python

for i in range (1,5,1):

for j in range(1,i+1,1):

print("\*",end=" ")

for k in range (i,5,1):

print(chr(69-k),end=" ")

print()

**Output:**

Plaintext

\* D C B A

\* \* C B A

\* \* \* B A

\* \* \* \* A

**Example 5: Letter Triangle**

**Definition:** This nested loop prints a right-angle triangle using letters. The inner loop (j) runs from 1 to i+1, printing 'A', then 'A B', and so on.

**Example Code:**

Python

for i in range(1,5,1):

for j in range(1,i+1,1):

print(chr(64+j),end=" ")

print()

**Output:**

Plaintext

A

A B

A B C

A B C D

**Example 6: Diamond Letter Pattern (File Discrepancy)**

**Definition:** A complex pattern that combines multiple loops to print \* symbols for padding and a centered, expanding sequence of letters. (Note: The variable a was set to 6 in a previous cell in the file.)

**Note on File Discrepancy:** The code in your file (specifically the last loop, for j in range(i, a):) does not match the saved output.

* The **code** for j in range(i, a): (which is range(i, 6)) should print 5 stars (\* \* \* \* \*) on the first line.
* The saved output shows 3 stars (\* \* \*), which would be produced by for j in range(i, 4):.

This is another case where the code was likely changed without re-running the cell.

**Example Code (as written in file):**

Python

for i in range(1, 5):

for j in range(i, 4):

print("\*", end=" ")

for j in range(i, 1, -1):

print(chr(64 + j), end=" ")

print("A", end=" ")

for j in range(2, i + 1):

print(chr(64 + j), end=" ")

for j in range(i, a):

print("\*", end=" ")

print()

**Output (as saved in file):**

Plaintext

\* \* \* A \* \* \* \* \* B A B \* \* \* C B A B C \* D C B A B C D

**Example 7: Inverted '\*' Triangle (Partial)**

**Definition:** This loop prints an inverted triangle of \* symbols. The inner loop j runs from i to 3.

* When i=1, range(1, 4) prints 3 stars.
* When i=2, range(2, 4) prints 2 stars.
* When i=3, range(3, 4) prints 1 star.
* When i=4, range(4, 4) is empty and prints nothing.

The print() at the end of the outer loop creates the new lines, including a blank one at the end.

**Example Code:**

Python

for i in range(1, 5):

for j in range(i, 4):

print("\*", end=" ")

print()

**Output:**

Plaintext

\* \* \* \* \* \* ```

---

### \*\*WHILE LOOP\*\*

\*\*Definition:\*\* A `while` loop repeatedly executes a block of code as long as a given condition is `True`.

#### Example 1: Basic `while` Loop

\*\*Definition:\*\* This loop initializes `a` to 1. It continues to run \*while\* `a < 5`. Inside the loop, it prints `a` and then increments `a` by 1 (`a += 1`). The loop stops when `a` becomes 5.

\*\*Example Code:\*\*

```python

a=1

while a<5 :

print(a,end=" ")

a+=1

**Output:**

Plaintext

1 2 3 4

**Example 2: Nested while and for Loops**

**Definition:** The outer while loop runs as long as a < 5 (4 times). In each pass of the while loop, the inner for loop runs completely, printing the numbers 1, 2, 3, 4.

**Example Code:**

Python

a=1

while a<5 :

for i in range(1,5,1):

print(i,end=" ")

print()

a+=1

**Output:**

Plaintext

1 2 3 4

1 2 3 4

1 2 3 4

1 2 3 4

**Example 3: while Loop for Sorting Even/Odd**

**Definition:** This loop iterates from 1 to 100. It uses the modulo operator (%) to check if a is even (a % 2 == 0). It then uses the append() function to add the number to the even list or the odd list.

**Functions:**

* **append()**: A list method that adds a single item to the end of a list.

**Example Code:**

Python

a=1

even=[]

odd=[]

while a<=100:

if a % 2 ==0:

even.append(a)

else:

odd.append(a)

a+=1

print(even)

print(odd)

**Output:**

Plaintext

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100]

[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99]