**Day 14 While in While Loop**

A nested while loop is a while loop placed inside another while loop. This structure is often used to create patterns or iterate over grid-like data. The outer loop typically controls the rows, and the inner loop controls the columns.

**Example 1: Simple while Loop with chr()**

**Definition:** This code uses a single while loop to print characters. a starts at 1 and increments up to 4. The chr(64 + a) function converts the numbers (65, 66, 67, 68) into their corresponding characters ('A', 'B', 'C', 'D').

**Example Code:**

Python

a = 1

while a <= 4:

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

a += 1

**Output:**

Plaintext

A B C D

**Example 2: Nested while Loop (Square Pattern)**

**Definition:** This is a classic nested loop.

* The **outer loop** (x) runs 5 times (for rows).
* The **inner loop** (a) runs 4 times for *each* iteration of the outer loop. It's important to note that a is reset to 1 every time the outer loop runs.
* This results in the line "A B C D" being printed 5 times.

**Example Code:**

Python

x = 1

while x <= 5:

a = 1

while a <= 4:

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

a += 1

print()

x += 1

**Output:**

Plaintext

A B C D

A B C D

A B C D

A B C D

A B C D

**Example 3: Right-Angle Triangle (Stars)**

**Definition:** This nested loop creates a right-angle triangle. The number of times the inner loop runs depends on the outer loop's current value.

* The outer loop (x) iterates from 1 to 4.
* The inner loop (a) runs x times (i.e., while a <= x:).
* This causes it to print 1 star, then 2 stars, then 3, then 4.

**Example Code:**

Python

x = 1

while x <= 4:

a = 1

while a <= x:

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

a += 1

print()

x += 1

**Output:**

Plaintext

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

---

### Example 4: Right-Angle Triangle (Numbers)

\*\*Definition:\*\* This code is identical in structure to Example 3, but instead of printing `\*`, it prints the inner loop's counter variable, `a`.

\*\*Example Code:\*\*

```python

x = 1

while x <= 4:

a = 1

while a <= x:

print(a,end = " ")

a += 1

print()

x += 1

**Output:**

Plaintext

1

1 2

1 2 3

1 2 3 4

**Example 5: Mixed Pattern**

**Definition:** This code uses two inner while loops, one after the other.

* The outer loop (x) runs 4 times.
* The first inner loop (a) prints an increasing number of \* symbols (1, then 2, 3, 4).
* The second inner loop (b) prints a decreasing sequence of numbers. The condition b <= 5 - x means it prints (1,2,3,4), then (1,2,3), then (1,2), then (1).

**Example Code:**

Python

x = 1

while x <= 4:

a = 1

while a <= x:

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

a += 1

b = 1

while b <= 5 - x:

print( b,end = " ")

b += 1

print()

x += 1

**Output:**

Plaintext

\* 1 2 3 4

\* \* 1 2 3

\* \* \* 1 2

\* \* \* \* 1

**Example 6: Inverted Triangle (File Discrepancy)**

**Definition:** This code is intended to print an inverted right-angle triangle of numbers. The outer loop (x) counts *down* from 5 to 1. The inner loop (a) should print numbers from 1 up to the current value of x.

**Expected Output:**

Plaintext

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

**Example Code:**

Python

x = 5

while x >= 1:

a = 1

while a <= x :

print(a,end = " ")

a += 1

print()

x -= 1

**Actual Output (from file):**

Plaintext

*(Note: The output in the file consists of 5 blank lines. This is inconsistent with the provided code and suggests the code was either changed after the cell was run or there was an execution issue.)*

**Function zip**

**Definition:** The zip() function takes two or more iterables (like tuples, lists, or strings) and combines them into a zip object. This object contains pairs (tuples) of corresponding elements from each iterable. The zipping stops as soon as the *shortest* iterable runs out of items.

**Example 1: zip Object**

**Definition:** This code calls zip() on two tuples. It doesn't print the pairs, but instead shows that the result of zip() is a zip object, which is an iterator.

**Example Code:**

Python

a = (1,2,3,4,5,6)

b = ('a','b','c','d','e','f')

zip(a,b)

**Output:**

Plaintext

<zip at 0x78a128d8f9c0>

**Example 2: zip with set()**

**Definition:** This code zips two tuples, a and b. b is longer than a. The zip() function pairs elements until a (the shorter tuple) is exhausted. The set() function then converts the resulting pairs into a set, which is an unordered collection of unique tuples. The extra item 'g' from tuple b is ignored.

**Example Code:**

Python

a = (1,2,3,4,5,6)

b = ('a','b','c','d','e','f','g')

set(zip(a,b))

**Output:**

Plaintext

{(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd'), (5, 'e'), (6, 'f')}

**Example 3: zip with dict()**

**Definition:** This demonstrates converting the output of zip() into a dictionary. The first iterable (a) provides the **keys**, and the second iterable (b) provides the **values**. Again, the extra item 'g' in b is ignored.

**Example Code:**

Python

a = (1,2,3,4,5,6)

b = ('a','b','c','d','e','f','g')

dict(zip(a,b))

**Output:**

Plaintext

{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f'}

**Example 4: zip with dict() (Swapped)**

**Definition:** This example is similar to the last one, but here the tuple of names (y) is used as the keys and the tuple of salaries (x) is used as the values.

**Example Code:**

Python

x = ('10k','20k','30k')

y = ('raj','karan','sahil')

z = dict(zip(y,x))

print(z)

**Output:**

Plaintext

{'raj': '10k', 'karan': '20k', 'sahil': '30k'}

**List Comprehension**

**Definition:** List comprehension is a concise, one-line syntax for creating lists. It combines a for loop, an operation, and (optionally) an if condition within square brackets [].

**Basic Syntax:** [expression for item in iterable]

**Example 1: Basic List Comprehension**

**Definition:** This is a simple list comprehension that creates a list of numbers from 1 to 5. It is equivalent to:

Python

a = []

for i in range(1, 6):

a.append(i)

**Example Code:**

Python

a = [i for i in range(1,6)]

print(a)

**Output:**

Plaintext

[1, 2, 3, 4, 5]

**Example 2: List Comprehension with an Expression**

**Definition:** This comprehension applies an expression (i\*\*2) to each item in the range. It creates a list containing the squares of numbers from 1 to 10.

**Example Code:**

Python

squ = [i\*\*2 for i in range(1,11)]

print(squ)

**Output:**

Plaintext

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

**Example 3: List Comprehension with if Condition**

**Definition:** This comprehension includes a conditional filter. The expression (i) is only added to the new list *if* the condition (i % 2 == 0) is true. This efficiently creates a list of all even numbers from 1 to 100.

**Syntax with Condition:** [expression for item in iterable if condition]

**Example Code:**

Python

even = [i for i in range(1,101) if i%2==0]

print(even)

**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]