**While Loop Day 13**

**Example 1: Basic while Loop**

**Definition:** This code initializes a to 1. The while loop continues to run as long as the condition a != 2 (a is not equal to 2) is true.

1. **Iteration 1:** a is 1. 1 != 2 is True. The loop prints 's' and increments a to 2.
2. **Iteration 2:** a is 2. 2 != 2 is False. The loop terminates.

**Example Code:**

Python

a = 1

while a!= 2:

print('s')

a+=1

**Output:**

Plaintext

s

**Example 2: Reverse a Number**

**Definition:** This program reverses an integer entered by the user.

1. It takes an integer n and stores a copy of it in n1.
2. The while n != 0 loop runs as long as n is not 0.
3. **rem = n % 10**: The **modulo operator (%)** gets the last digit of n (the remainder).
4. **rev = (rev \* 10) + rem**: This line builds the reversed number. It shifts the existing rev digits one place to the left (by multiplying by 10) and adds the new remainder to the ones place.
5. **n = n // 10**: The **floor division operator (//)** removes the last digit from n.
6. The loop repeats until all digits are processed.

Example Code:

(Note: The input for this cell was likely 1234, based on the next cell's example)

Python

n = int(input("Enter a No:- "))

n1 = n

rev = 0

while n != 0:

rem = n % 10

rev = (rev \* 10) + rem

n = n // 10

print(rev)

print(n1)

**Output:**

Plaintext

Enter a No:- 1234

4321

1234

**Example 3: Modulo Operator**

**Definition:** This is a standalone example showing how the modulo operator (% 10) works. It calculates the remainder when 1234 is divided by 10, which effectively isolates the last digit.

**Example Code:**

Python

1234 % 10

**Output:**

Plaintext

4

**Example 4: Palindrome Check (True Case)**

**Definition:** This code checks if a number is a **palindrome** (a number that reads the same backward as forward, like 121). It uses the same logic as Example 2 to reverse the number, and then uses an if-else statement to compare the original number (n1) with the reversed number (rev).

**Example Code:**

Python

n = int(input("Enter a No:- "))

n1 = n

rev = 0

while n != 0:

rem = n % 10

rev = (rev \* 10) + rem

n = n // 10

print(rev)

print(n1)

if n1 == rev:

print("Palindrome")

else:

print("Not a Palindrome")

**Output:**

Plaintext

Enter a No:- 121

121

121

Palindrome

**Example 5: Palindrome Check (False Case)**

**Definition:** This is the same program as Example 4, but this time it's given a non-palindrome number. Since the original (n1 = 1212) and the reversed (rev = 2121) are not equal, it prints "Not a Palindrome".

**Example Code:**

Python

n = int(input("Enter a No:- "))

n1 = n

rev = 0

while n != 0:

rem = n % 10

rev = (rev \* 10) + rem

n = n // 10

print(rev)

print(n1)

if n1 == rev:

print("Palindrome")

else:

print("Not a Palindrome")

**Output:**

Plaintext

Enter a No:- 1212

2121

1212

Not a Palindrome

**Example 6: Palindrome Check with Nested Loop**

**Definition:** This is the same palindrome-checking logic, but with an unnecessary for i in range(1, 2): loop nested inside the while loop. This for loop runs only once (for i=1) during *each* iteration of the while loop, printing 1 each time. The input 12121 has 5 digits, so the while loop runs 5 times, and 1 is printed 5 times.

**Example Code:**

Python

n = int(input("Enter a No:- "))

n1 = n

rev = 0

while n != 0:

for i in range(1, 2):

print(i, end=" ")

rem = n % 10

rev = (rev \* 10) + rem

n = n // 10

print(rev)

print(n1)

if n1 == rev:

print("Palindrome")

else:

print("Not a Palindrome")

**Output:**

Plaintext

Enter a No:- 12121

1 1 1 1 1 12121

12121

Palindrome

**Example 7: Complex Nested Loop**

**Definition:** This code shows a complex interaction of loops and an if-else statement.

1. a starts at 1. The outer while a <= 5 is True.
2. The for i in range(1, 6, -1): loop does **not** run because its range is empty (it's impossible to go from 1 to 6 with a step of -1).
3. The inner while a <= 5: loop starts.
4. It prints + and increments a. This continues until a becomes 6. (Outputs: +, +, +, +, +)
5. The inner while loop terminates.
6. The if a == 2: check is evaluated. a is now 6, so this is False.
7. The else: block runs, printing 3.
8. It prints the current value of a (which is 6).
9. The outer while loop's condition a <= 5 is checked. a is 6, so the condition is False, and the loop terminates.
10. The final print(a + 2) (6 + 2) is executed, printing 8.

**Example Code:**

Python

a = 1

while a <= 5:

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

print('\*')

while a <= 5:

print('+')

a+=1

if a == 2:

print(2)

else:

print('3')

print(a)

print(a + 2)

**Output:**

Plaintext

+

+

+

+

+

3

6

8

**Example 8: if Conditions and "Truthiness"**

**Definition:** This demonstrates how Python evaluates "truthiness" in if statements.

1. a = "1234".
2. if a[2]:: a[2] is the character '3'. In Python, any non-empty string is considered True. The condition is met, so it prints "x".
3. if len(list([a[0], a[3]])): This creates a list ['1', '4']. len() of this list is 2. In Python, any non-zero integer is considered True. The condition is met, so it prints "Q".

**Example Code:**

Python

a = "1234"

if a [2]:

print("x")

else:

print("y")

if len(list([a[0], a[3]])):

print("Q")

else:

print("S")

**Output:**

Plaintext

x

Q

**Example 9: if Condition with len()**

**Definition:** This is a partial repeat of the previous example, isolating the second if statement.

1. a = "1234".
2. list([a[0], a[3]]) creates ['1', '4'].
3. len(['1', '4']) returns 2.
4. if 2: is evaluated. Since 2 is a non-zero number, it is treated as True.
5. The code prints "Q".

**Example Code:**

Python

a = "1234"

if len(list([a[0], a[3]])):

print("Q")

else:

print("S")

**Output:**

Plaintext

Q