2-Python Loops:

1. Introduction to Loops

What are loops?
Loops allow you to repeat a block of code multiple times. This is essential for:

- Processing multiple items in a list
- Repeating operations until a condition is met
- Automating repetitive tasks

Real-life analogy:

- **WHILE** there are dirty dishes, **DO** wash one dish
- **FOR EACH** student in the class, **DO** take attendance

2. while Loop: The Conditional Loop

2.1 Basic while Loop Structure

python

```
# Example from document
x = 1
while x <= 5:
    # end=' ': print space not \n
    print(x, end=',')
    x = x + 1
# output: 1,2,3,4,5,
```

Detailed Explanation:

1. **Initialization**: x = 1 - Start with x equal to 1
2. **Condition Check**: while x <= 5: - Check if x is less than or equal to 5
3. **If condition is True:**
    - Execute the indented block
    - print(x, end=',') - Print x with a comma, not newline

- x = x + 1 - Increase x by 1 (this is CRITICAL!)

4. **After the block:** Go back to step 2 (check condition again)

Key Insight: Without x = x + 1, x would always be 1, the condition would always be true, and we'd get an **infinite loop**!

What happens step by step:

text

x = 1 → Check: 1 <= 5 → True → Print "1," → x becomes 2

x = 2 → Check: 2 <= 5 → True → Print "2," → x becomes 3

x = 3 → Check: 3 <= 5 → True → Print "3," → x becomes 4

x = 4 → Check: 4 <= 5 → True → Print "4," → x becomes 5

x = 5 → Check: 5 <= 5 → True → Print "5," → x becomes 6

x = 6 → Check: 6 <= 5 → False → Exit loop

2.2 The Infinite Loop Trap

python

```
# From document: shows the difference between if and while
if 2 < 5:
    print("2 < 5. Printed once")


while 2 < 5:
    print("2 < 5. Forever")  # INFINITE LOOP!
```

Explanation:

- if checks once: since 2 < 5 is always true, it prints once
- while keeps checking: 2 < 5 is ALWAYS true, so it prints forever

Why doesn't it stop? Because 2 and 5 are constants - they never change! The condition always evaluates to True.

How to create a useful while loop:

1. **Initialize** a variable before the loop
2. **Check** that variable in the condition
3. **Change** that variable inside the loop

3. Loop Control: break and continue

3.1 break: Exit the Loop Immediately

python

```
# Example from document: Division program
while True:
    x, y = map(float, input('enter two numbers').split())
    if y == 0:
        print("Y is zero!")
        break   # STOP the loop. Jump to line 10
    print(x / y)
print("Bye")
```

How break **works:**

1. while True: creates an infinite loop (condition always True)
2. User enters two numbers
3. If second number (y) is 0:
    - Print "Y is zero!"
    - break immediately exits the while loop
    - Jumps to line after the loop (print("Bye"))
4. If y is not 0:
    - Perform division
    - Loop continues (goes back to line 1)

Example Execution:

text

Input: 10 2

Output: 5.0

Input: 5 0

Output: Y is zero!

Bye

4. Practical while Loop Examples

4.1 Sum of Numbers 1 to 5

python

```
# Two approaches from document
# Approach 1: Counting up
x = 1
sum = 0
while x <= 5:
    sum += x
    x = x + 1
print(sum)  # 15

# Approach 2: Counting down
x = 5
sum = 0
while x >= 0:
    sum += x
    x -= 1  # Same as x = x - 1
```

print(sum) *# 15*

Explanation of sum += x**:**

- This is shorthand for sum = sum + x
- Takes current value of sum, adds x, stores result back in sum

Common Patterns:

- **Counting up**: Start at 1, increment until limit
- **Counting down**: Start at n, decrement until 0

4.2 Practice 1: Print Multiples of 3 الاعداد القابلة للقسمة على

python

```
end = int(input())
start = 1
while start <= end:
    if start % 3 == 0:
        print(start)
    start += 1
```

Explanation:

1. Get ending number from user
2. Start from 1
3. While current number ≤ end:
    - Check if divisible by 3 (start % 3 == 0)
    - If yes, print it
    - Increment number by 1

Example:

text

Input: 10

Output: 3

6

9

4.3 Practice 2: Calculate Power

```
#python
num, pow = map(int, input('Enter two numbers').split())
python_result = num ** pow
our_result = 1
while pow >= 1:
    our_result *= num
    pow -= 1
print(our_result)
```

How it works:

- num ** pow is Python's built-in power operator
- Our manual calculation:
    - Start with our_result = 1
    - Multiply by num, pow times
    - Decrement pow each time

Example with num = 2**,** pow = 3**:**

text

Initial: our_result = 1, pow = 3

Loop 1: our_result = 1 * 2 = 2, pow = 2

Loop 2: our_result = 2 * 2 = 4, pow = 1

Loop 3: our_result = 4 * 2 = 8, pow = 0

Exit loop: pow >= 1 is False

Result: 8 (same as 2 ** 3)

5. The Digit Counter Bug Example

```
num = int(input())
num2 = num  # copy
if num == 0:
    digits = 1
else:
    digits = 0
    if num < 0:
        num = -num  # Handle negatives
    while num > 0:
        digits += 1
        num //= 10
print('# digits of', num2, 'is', digits)
```

The Fixes:

1. **Copy the number**: num2 = num keeps original for display
2. **Handle 0 separately**: 0 has 1 digit
3. **Handle negatives**: Convert to positive for counting

6. for Loop: The Counting Loop

6.1 Basic for Loop with range() (understanding the range)

python

```
for pos in range(5):
    print(pos, end=' ')  # 0 1 2 3 4
print()

for pos in range(2, 5):
    print(pos, end=' ')  # 2 3 4
```

```
print()


rng = range(1, 21, 4)
for pos in rng:
    print(pos, end=' ')  # 1 5 9 13 17
print()
```

Understanding range()**:**

- range(stop): 0 to stop-1
- range(start, stop): start to stop-1
- range(start, stop, step): start to stop-1, incrementing by step

Key Difference from while**:**

- for loop automatically:
    - Initializes counter
    - Checks condition
    - Increments counter
- You don't need to manually manage the loop variable

6.2 Counting Backwards with range()

python

```
for pos in range(5, 0, -1):
    print(pos, end=' ')  # 5 4 3 2 1
print()
for pos in range(10, 0, -2):
    print(pos, end=' ')  # 10 8 6 4 2
print()
```

Explanation:

- range(5, 0, -1): Start at 5, go down to 1 (stop before 0)

- range(10, 0, -2): Start at 10, go down by 2: 10, 8, 6, 4, 2

6.3 Looping Through Strings

python

```
mystr = 'Me!'
for char in mystr:
    print(char)
```

Why this works: Strings in Python are **iterable** - you can loop through each character.

7. Nested Loops

7.1 Example: Sum Test Cases

python

```
n = int(input("Enter a number: "))
total = 0
for pos in range(1, n + 1):
    total += pos
print("Sum from 1 to", n, "=", total)
```

How it works:

1. Outer loop: For each test case
2. Inner loop: Calculate sum from 1 to n
3. Outer loop variable case tracks which test case we're on

Best Practices:

1. **Use meaningful variable names**: row instead of i for row counter
2. **Initialize variables** before loops
3. **Avoid deep nesting** - break complex loops into functions
4. **Use** break **and** continue judiciously - they can make code harder to read