**Python Loop Examples -**

Here are some examples of loops in Python, including `for` and `while` loops:  
  
---  
  
### \*\*1. For Loop Example\*\*  
The `for` loop is used to iterate over a sequence (like a list, tuple, string, or range).  
  
```python  
# Example: Iterating through a list  
fruits = ["apple", "banana", "cherry"]  
for fruit in fruits:  
 print(fruit)  
  
# Example: Using range()  
for i in range(5): # Loops from 0 to 4  
 print(i)  
```  
  
---  
  
### \*\*2. While Loop Example\*\*  
The `while` loop continues as long as a condition is `True`.  
  
```python  
# Example: Counting with a while loop  
count = 0  
while count < 5:  
 print("Count is:", count)  
 count += 1  
```  
  
---  
  
### \*\*3. Nested Loops\*\*  
You can nest loops inside each other.  
  
```python  
# Example: Nested for loop  
for i in range(3): # Outer loop  
 for j in range(2): # Inner loop  
 print(f"i = {i}, j = {j}")  
```  
  
---  
  
### \*\*4. Loop with `break` and `continue`\*\*  
You can use `break` to exit a loop and `continue` to skip the current iteration.  
  
```python  
# Example: Using break  
for i in range(10):  
 if i == 5:  
 break # Exit the loop when i is 5  
 print(i)  
  
# Example: Using continue  
for i in range(10):  
 if i % 2 == 0:  
 continue # Skip even numbers  
 print(i)  
```  
  
---  
  
### \*\*5. Looping Through a String\*\*  
You can loop through each character in a string.  
  
```python  
# Example: Iterating through a string  
word = "Python"  
for char in word:  
 print(char)  
```  
  
---  
  
### \*\*6. Using `else` with Loops\*\*  
The `else` block in a loop runs if the loop completes without hitting a `break`.  
  
```python  
# Example: For loop with else  
for i in range(5):  
 print(i)  
else:  
 print("Loop completed!")  
  
# Example: While loop with else  
count = 0  
while count < 3:  
 print(count)  
 count += 1  
else:  
 print("While loop completed!")  
```  
  
---  
  
### \*\*7. Infinite Loop\*\*  
Be cautious with infinite loops; they run forever unless stopped.  
  
```python  
# Example: Infinite loop (use with caution)  
while True:  
 print("This will run forever!")  
 break # Add a break to stop it  
```  
  
---  
  
### \*\*8. Looping Through a Dictionary\*\*  
You can loop through keys, values, or both in a dictionary.  
  
```python  
# Example: Iterating through a dictionary  
person = {"name": "Alice", "age": 25, "city": "New York"}  
for key, value in person.items():  
 print(f"{key}: {value}")  
```  
  
---  
  
These examples cover common use cases for loops in Python. Let me know if you'd like more advanced examples!

Some advance examples

Here are some \*\*advanced examples of loops in Python\*\*, demonstrating more complex use cases and techniques:  
  
---  
  
### \*\*1. Loop with List Comprehension\*\*  
List comprehensions are a concise way to create lists using loops.  
  
```python  
# Example: Squaring numbers in a range  
squares = [x\*\*2 for x in range(10)]  
print(squares)  
  
# Example: Filtering even numbers  
evens = [x for x in range(20) if x % 2 == 0]  
print(evens)  
```  
  
---  
  
### \*\*2. Loop with Dictionary Comprehension\*\*  
You can use loops to create dictionaries dynamically.  
  
```python  
# Example: Creating a dictionary with squares  
squares\_dict = {x: x\*\*2 for x in range(5)}  
print(squares\_dict)  
  
# Example: Filtering dictionary items  
original\_dict = {"a": 1, "b": 2, "c": 3, "d": 4}  
filtered\_dict = {k: v for k, v in original\_dict.items() if v % 2 == 0}  
print(filtered\_dict)  
```  
  
---  
  
### \*\*3. Using `zip()` to Loop Through Multiple Iterables\*\*  
The `zip()` function allows you to iterate over multiple sequences in parallel.  
  
```python  
# Example: Looping through two lists  
names = ["Alice", "Bob", "Charlie"]  
ages = [25, 30, 35]  
for name, age in zip(names, ages):  
 print(f"{name} is {age} years old.")  
  
# Example: Looping through three lists  
scores = [85, 90, 95]  
for name, age, score in zip(names, ages, scores):  
 print(f"{name}, age {age}, scored {score}.")  
```  
  
---  
  
### \*\*4. Loop with `enumerate()`\*\*  
The `enumerate()` function provides both the index and the value during iteration.  
  
```python  
# Example: Using enumerate with a list  
fruits = ["apple", "banana", "cherry"]  
for index, fruit in enumerate(fruits):  
 print(f"Index {index}: {fruit}")  
```  
  
---  
  
### \*\*5. Looping Through a Generator\*\*  
Generators are memory-efficient and can be looped through like lists.  
  
```python  
# Example: Using a generator  
def generate\_numbers(n):  
 for i in range(n):  
 yield i\*\*2  
  
for num in generate\_numbers(5):  
 print(num)  
```  
  
---  
  
### \*\*6. Loop with `itertools`\*\*  
The `itertools` module provides advanced looping utilities.  
  
```python  
import itertools  
  
# Example: Infinite loop with itertools  
for i in itertools.count(10, 2): # Start at 10, step by 2  
 if i > 20:  
 break  
 print(i)  
  
# Example: Cartesian product  
colors = ["red", "blue"]  
sizes = ["S", "M", "L"]  
for combo in itertools.product(colors, sizes):  
 print(combo)  
```  
  
---  
  
### \*\*7. Loop with `try` and `except`\*\*  
You can handle exceptions inside a loop.  
  
```python  
# Example: Handling errors in a loop  
numbers = [10, 0, 5]  
for num in numbers:  
 try:  
 result = 100 / num  
 print(f"100 / {num} = {result}")  
 except ZeroDivisionError:  
 print("Cannot divide by zero!")  
```  
  
---  
  
### \*\*8. Looping Through a 2D List (Matrix)\*\*  
You can use nested loops to iterate through a 2D list.  
  
```python  
# Example: Iterating through a matrix  
matrix = [  
 [1, 2, 3],  
 [4, 5, 6],  
 [7, 8, 9]  
]  
  
for row in matrix:  
 for value in row:  
 print(value, end=" ")  
 print() # Newline after each row  
```  
  
---  
  
### \*\*9. Loop with Custom Iterators\*\*  
You can create your own iterator class and loop through it.  
  
```python  
# Example: Custom iterator  
class MyRange:  
 def \_\_init\_\_(self, start, end):  
 self.current = start  
 self.end = end  
  
 def \_\_iter\_\_(self):  
 return self  
  
 def \_\_next\_\_(self):  
 if self.current >= self.end:  
 raise StopIteration  
 value = self.current  
 self.current += 1  
 return value  
  
for num in MyRange(1, 5):  
 print(num)  
```  
  
---  
  
### \*\*10. Loop with `async` and `await`\*\*  
Asynchronous loops are used in asynchronous programming.  
  
```python  
import asyncio  
  
# Example: Asynchronous loop  
async def print\_numbers():  
 for i in range(5):  
 print(i)  
 await asyncio.sleep(1) # Simulate async task  
  
asyncio.run(print\_numbers())  
```  
  
---  
  
### \*\*11. Loop with `collections`\*\*  
The `collections` module provides advanced data structures for looping.  
  
```python  
from collections import Counter  
  
# Example: Counting elements in a list  
fruits = ["apple", "banana", "apple", "cherry", "banana", "apple"]  
fruit\_count = Counter(fruits)  
for fruit, count in fruit\_count.items():  
 print(f"{fruit}: {count}")  
```  
  
---  
  
### \*\*12. Loop with Progress Bar\*\*  
You can use the `tqdm` library to display a progress bar during a loop.  
  
```python  
from tqdm import tqdm  
import time  
  
# Example: Loop with progress bar  
for i in tqdm(range(10)):  
 time.sleep(0.5) # Simulate a task  
```  
  
---  
  
These examples demonstrate advanced techniques for using loops in Python.