10-loops-problem-python

0.1 Loops in Python - Solving 10 problems

0.1.1 10 Problems

1. Counting Positive Numbers Problem: Given a list of numbers, count how many are positive.

```
numbers = [1, -2, 3, -4, 5, 6, -7, -8, 9, 10]
```

2. Sum of Even Numbers Problem: Calculate the sum of even numbers up to a given number n.

3. Multiplication Table Printer Problem: Print the multiplication table for a given number up to 10, but skip the fifth iteration.

4. Reverse a String Problem: Reverse a string using a loop.

5. Find the First Non-Repeated Character Problem: Given a string, find the first non-repeated character.

6. Factorial Calculator Problem: Compute the factorial of a number using a while loop.

7. Validate Input Problem: Keep asking the user for input until they enter a number between 1 and 10.

8. Prime Number Checker Problem: Check if a number is prime.

9. List Uniqueness Checker Problem: Check if all elements in a list are unique. If a duplicate is found, exit the loop and print the duplicate.

```
items = ["apple", "banana", "orange", "apple", "mango"]
```

10. Exponential Backoff Problem: Implement an exponential backoff strategy that doubles the wait time between retries, starting from 1 second, but stops after 5 retries.

1. Counting Positive Numbers Problem: Given a list of numbers, count how many are positive.

```
numbers = [1, -2, 3, -4, 5, 6, -7, -8, 9, 10]
```

[ ]:
```
numbers = [1, -2, 3, -4, 5, 6, -7, -8, 9, 10]
count = 0
for i in numbers:
  if i>0:
    count+=1
print(count)
```

```
6
```

2. Sum of Even Numbers Problem: Calculate the sum of even numbers up to a given number n.

[ ]:
```
n = int(input("Enter the value of n: "))
sum = 0
for i in range(1,n+1):
  if i%2==0:
    sum += i
print(sum)
```

```
Enter the value of n: 8
20
```

3. Multiplication Table Printer Problem: Print the multiplication table for a given number up to 10, but skip the fifth iteration.

[ ]:
```
n = int(input("Enter the number whose table you want to print: "))
for i in range (1,11):
  if i==5:
    continue
  print(n*i)
```

```
Enter the number whose table you want to print: 4
4
8
12
16
24
28
32
36
40
```

4. Reverse a String Problem: Reverse a string using a loop.

[ ]:
```
club = "manchester united"
reversed = ""

for char in club:
  reversed = char+reversed
print(reversed)
```

```
detinu retsehcnam
```

5. Find the First Non-Repeated Character Problem: Given a string, find the first non-repeated character.

[ ]:
```
text = "teeteracdacd"
for char in text:
  if text.count(char) == 1:
    print(char)
```

```
    break
```

r

6. Factorial Calculator Problem: Compute the factorial of a number using a while loop.

[ ]:
```
n = 5
f = 1
while n!=0:
  f = f*(n)
  n = n-1
print(f)
```

120

7. Validate Input Problem: Keep asking the user for input until they enter a number between 1 and 10.

[ ]:
```
while True:
  n = int(input("Enter value between 1-10: "))
  if 1<=n<=10:
    print("Thanks")
    break
  else:
    print("Invalid Number, Try Again")
```

```
Enter value between 1-10: 24
Invalid Number, Try Again
Enter value between 1-10: 9
Thanks
```

8. Prime Number Checker Problem: Check if a number is prime.

[ ]:
```
n = int(input("Enter a number for prime checking: "))
count = 0
if n>1:
  for i in range(2,n):
    if(n%i==0):
      count+=1;
      break
  if(count==0):
    print("prime")
  else:
    print("Not prime")
else:
  print("Neither prime Nor composite")
```

```
Enter a number for prime checking: 72
Not prime
```

[ ]:
```
n = int(input("Enter a number for prime checking: "))
is_Prime = True
if n>1:
  for i in range(2,n):
    if(n%i==0):
      is_Prime = False
      break
  print(is_Prime)
else:
  print("Neither prime nor composite")
```

```
Enter a number for prime checking: 1
Neither prime nor composite
```

9. List Uniqueness Checker Problem: Check if all elements in a list are unique. If a duplicate is found, exit the loop and print the duplicate.

```
items = ["apple", "banana", "orange", "apple", "mango"]
```

[ ]:
```
items = ["apple", "banana", "orange", "apple", "mango"]

unique_item = set()

for item in items:
    if item in unique_item:
        print("Duplicate: ", item)
        break
    unique_item.add(item)
```

```
Duplicate:  apple
```

10. Exponential Backoff Problem: Implement an exponential backoff strategy that doubles the wait time between retries, starting from 1 second, but stops after 5 retries.

[ ]:
```
import time

wait_time = 1
max_retries = 5
attempts = 0

while attempts < max_retries:
    print("Attempt", attempts + 1, "- wait time", wait_time, )
    time.sleep(wait_time)
    wait_time *= 2
    attempts += 1
```

```
Attempt 1 - wait time 1
Attempt 2 - wait time 2
Attempt 3 - wait time 4
```

```
Attempt 4 - wait time 8
Attempt 5 - wait time 16
```