Indefinite Loop Worksheet

1. Compare and contrast the following pairs of terms:
   a. Definite loop vs. indefinite loop

   A definite loop executes a specific number of times that is known *a priori*. The exception would be if a break statement is nested inside a definite loop.

   In the declaration of an indefinite loop, the number of execution times is not specified.

   b. Interactive loop vs. Sentinel Loop

   A sentinel loop uses the same input command to ask the user to enter a value or exit the loop. An interactive loop has a separate input command that asks the user whether or not to continue.

2. Give a truth table that shows the Boolean value of each of the following expressions:
   a. not (P and Q)

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>not(P and Q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

   b. (not P) and Q

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>(not P) and Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

   c. (not P) or (not Q)

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>(not P) or (not Q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
d. \((P \lor R) \land (Q \lor R)\)

<table>
<thead>
<tr>
<th>P</th>
<th>R</th>
<th>Q</th>
<th>((P \lor R) \land (Q \lor R))</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

3. Write a while loop fragment that calculates the following values:
   a. Sum of the first \(n\) counting numbers: \(1 + 2 + 3 + \ldots + n\)

   ```python
   sum = 0
   i = 1
   n = eval(input("n: "))
   while i <= n:
       sum = sum + i
       i = i + 1
   ```

   b. Sum of a series of numbers entered by the user until the value 999 is entered. Note: 999 should not be part of the sum.

   ```python
   sum = 0
   while True:
       n = eval(input("Enter a value: "))
       if n == 999:
           break
       sum = sum + n
   ```
c. The Syracuse (also called Collatz or Hailstone) sequence is generated by starting with a natural number and repeatedly applying the following functions until reaching 1:

\[
\text{syr}(x) = \begin{cases} 
\frac{x}{2} & \text{if } x \text{ is even} \\
3x + 1 & \text{if } x \text{ is odd}
\end{cases}
\]

\[
x = \text{eval(input("x: "))}
\]
\[
\text{while } x \neq 1:
\]
\[
\quad \text{if } x \% 2 == 0:
\quad \quad x = x/2
\]
\[
\quad \text{else:}
\quad \quad x = 3\times x + 1
\]

4. A positive whole number \( n > 2 \) is prime if no number between 2 and \( \sqrt{n} \) (inclusive) evenly divides \( n \). Write a program that accepts a value of \( n \) as input and determines if the value is prime. If \( n \) is not prime, your program should quit as soon as it finds a value that evenly divides \( n \).

```python
import math
n = eval(input("Enter n (>2): "))
i = 2
prime = True
while i <= math.sqrt(n):
    if n % i == 0:
        prime = False
        break
    i = i + 1
print(prime)
```

5. Modify the previous program to find every prime number less than or equal to \( n \).

```python
import math
max_n = eval(input("Enter n (>2): "))
primes = []
for n in range(3,max_n+1):
    i = 2
    prime = True
    while i <= math.sqrt(n):
        if n % i == 0:
            prime = False
            break
        i = i + 1
    if prime:
        primes.append(n)
print(primes)
```
i = i + 1
if prime:
    primes.append(n)
print(primes)

6. Write a program that computes the fuel efficiency of a multi-leg journey. The program will first prompt for the starting odometer reading and then get information about a series of legs. For each leg, the user enters the current odometer reading and the amount of gas used (separated by a space). The user signals the end of the trip with a blank line. The program should print out the miles per gallon achieved on each leg and the total MPG for the trip.

last_odometer = eval(input("Odometer reading: "))
sum = 0
legs = 0
while True:
    in = input("Enter odometer reading and amount of gas (blank line signals end of trip)"
    if in.strip() == "":
        break
    fields = in.split(" ")
    odometer = eval(fields[0])
    gas = eval(fields[1])
    print((last_odometer - odometer)/gas)
    sum = sum + (last_odometer - odometer)/gas
    legs = legs + 1
    last_odometer = odometer

print(sum/legs)