while (mahself.stillAwake())
{
    sheep++;
}

Lecture 17: while Loops and Sentinel Loops
Adapted from slides by Marty Stepp and Stuart Reges
Categories of loops

• **definite loop**: Executes a known number of times.
  • The `for` loops we have seen are definite loops.
    • Print "hello" 10 times.
    • Find all the prime numbers up to an integer \( n \).
    • Print each odd number between 5 and 127.

• **indefinite loop**: One where the number of times its body repeats is not known in advance.
  • Prompt the user until they type a non-negative number.
  • Print random numbers until a prime number is printed.
  • Repeat until the user has typed "q" to quit.
The **while** loop

- **while loop**: Repeatedly executes its body as long as a logical test is true.

  ```python
  while test:
      statement(s)
  ```

- Example:

  ```python
  num = 1
  while num <= 200:
      print(str(num) + " ", end='')
      num = num * 2
  
  # output:  1  2  4  8  16  32  64  128
  ```
Example **while** loop

```python
# finds the first factor of 91, other than 1
n = 91
factor = 2
while n % factor != 0:
    factor += 1
print("First factor is", factor)
```

# output: First factor is 7

- **while** is better than **for** because we don't know how many times we will need to increment to find the factor.
Sentinel values

- **sentinel**: A value that signals the end of user input.
  - **sentinel loop**: Repeats until a sentinel value is seen.

- Example: Write a program that prompts the user for text until the user types "quit", then output the total number of characters typed.
  - (In this case, "quit" is the sentinel value.)

  Type a word (or "quit" to exit): hello
  Type a word (or "quit" to exit): yay
  Type a word (or "quit" to exit): quit
  You typed a total of 8 characters.
Solution?

```python
sum = 0
response = "dummy"  # "dummy" value, anything but "quit"

while response != "quit":
    response = input("Type a word (or \"quit\" to exit): ")
    sum += len(response)

print("You typed a total of " + str(sum) + " characters.")
```

• This solution produces the wrong output. Why?
  You typed a total of 12 characters.
The problem with our code

• Our code uses a pattern like this:
  
  \[\begin{align*}
  \text{\emph{sum}} &= 0 \\
  \text{while input is not the sentinel:} \\
  \text{prompt for input; read input.} \\
  \text{add input length to the sum.}
  \end{align*}\]

• On the last pass, the sentinel’s length (4) is added to the sum:
  
  \[\begin{align*}
  \text{prompt for input; read input ("quit").} \\
  \text{add input length (4) to the sum.}
  \end{align*}\]

• This is a fencepost problem.
  • Must read \(N\) lines, but only sum the lengths of the first \(N-1\).
A fencepost solution

\[ \text{sum} = 0. \]

```python
prompt for input; read input. # place a "post"
```

```python
while (input is not the sentinel):
    add input length to the sum. # place a "wire"
    prompt for input; read input. # place a "post"
```

• Sentinel loops often utilize a fencepost "loop-and-a-half" style solution by pulling some code out of the loop.
Correct code

```python
sum = 0

# pull one prompt/read ("post") out of the loop
response = input("Type a word (or \"quit\" to exit): ")

while (response != "quit"): # moved to top of loop
    sum += len(response)
    response = input("Type a word (or \"quit\" to exit): ")

print("You typed a total of " + str(sum) + " characters.")
```
Sentinel as a constant

```
SENTINEL = "quit"
...
sum = 0

# pull one prompt/read ("post") out of the loop
response = input("Type a word (or " + SENTINEL + " to exit): ")

while response != SENTINEL:
    sum += len(response)  # moved to top of loop
    response = input("Type a word (or " + SENTINEL + " to exit): ")

print("You typed a total of " + str(sum) + " characters.")
```