10. Iteration: The while-Loop

Topics:
- repetition
- the while statement
- generating sequences
- summation
- looking for patterns in strings

Open-Ended Iteration
So far, we have only addressed iterative problems in which we know (in advance) the required number of repetitions.

Not all iteration problems are like that.
Some iteration problems are open-ended

Stir for 5 minutes vs Stir until fluffy.

Examples

Keep tossing a coin until the number of heads and the number of tails differs by 10.

Repeat this until |L-W| <= .000001:
  L = (L + W)/2
  W = x/L

In both cases, we do not know the number of iterations that will be required.

The While Loop

We introduce an alternative to the for-loop called the while-loop.

The while loop is more flexible and is essential for ``open ended'' iteration.

How Does a While-Loop Work?

A simple warm-up example: sum the first 5 whole numbers and display the summation process.

Two Solutions

\[
\begin{align*}
k &= 0 \\
s &= 0 \\
\text{while } k < 5: \\
&\quad k = k + 1 \\
&\quad s = s + k \\
&\quad \text{print } k, s
\end{align*}
\]

\[
\begin{align*}
s &= 0 \\
\text{for } k \text{ in range}(1,6): \\
&\quad s = s + k \\
&\quad \text{print } k, s
\end{align*}
\]
The While-Loop Solution

```python
k = 0
s = 0
while k < 5:
    k = k + 1
    s = s + k
print k, s
```

Observation: k is used for counting, s is used for the running sum, and the while is used to control the repetition of the indented code.

The Solution

```python
k = 0
s = 0
while k < 5:
    k = k + 1
    s = s + k
print k, s
```

We call this the "loop body"

Trace the Execution

```python
k = 0
s = 0
while k < 5:
    k = k + 1
    s = s + k
    print k, s
```

At the start, k and s are initialized

Is the boolean condition true?

Trace the Execution

```
k = 0
s = 0
while k < 5:
    k = k + 1
    s = s + k
    print k, s
```

Yes, so execute the loop body
Trace the Execution

\begin{align*}
\text{k = 0} & \quad \text{s = 0} \\
\text{while k < 5:} & \\
\text{\quad k = k + 1} & \quad \text{\quad s = s + k} \\
\text{\quad print k, s} & \\
\end{align*}

Is the boolean condition true?

Yes, so execute the loop body

Trace the Execution

\begin{align*}
\text{k = 0} & \quad \text{s = 0} \\
\text{while k < 5:} & \\
\text{\quad k = k + 1} & \quad \text{\quad s = s + k} \\
\text{\quad print k, s} & \\
\end{align*}

Yes, so execute the loop body

Trace the Execution

\begin{align*}
\text{k = 0} & \quad \text{s = 0} \\
\text{while k < 5:} & \\
\text{\quad k = k + 1} & \quad \text{\quad s = s + k} \\
\text{\quad print k, s} & \\
\end{align*}

Is the boolean condition true?

Trace the Execution

\begin{align*}
\text{k = 0} & \quad \text{s = 0} \\
\text{while k < 5:} & \\
\text{\quad k = k + 1} & \quad \text{\quad s = s + k} \\
\text{\quad print k, s} & \\
\end{align*}
Trace the Execution

\[
\begin{align*}
k &= 0 \\
s &= 0 \\
\text{while } k < 5: \\
&\quad k = k + 1 \\
&\quad s = s + k \\
&\quad \text{print } k, s
\end{align*}
\]

Is the boolean condition true?

\[
\begin{align*}
k &\rightarrow 3 \\
s &\rightarrow 6 \\
\end{align*}
\]

Yes, so execute the loop body

\[
\begin{align*}
k &= 0 \\
s &= 0 \\
\text{while } k < 5: \\
&\quad k = k + 1 \\
&\quad s = s + k \\
&\quad \text{print } k, s
\end{align*}
\]

\[
\begin{align*}
k &\rightarrow 4 \\
s &\rightarrow 10 \\
\end{align*}
\]

Is the boolean condition true?

\[
\begin{align*}
k &= 0 \\
s &= 0 \\
\text{while } k < 5: \\
&\quad k = k + 1 \\
&\quad s = s + k \\
&\quad \text{print } k, s
\end{align*}
\]

Yes, so execute the loop body

\[
\begin{align*}
k &\rightarrow 5 \\
s &\rightarrow 15 \\
\end{align*}
\]
Trace the Execution

```
k = 0
s = 0
while k < 5:
    k = k + 1
    s = s + k
    print k, s
```

```
      1  1
      2  3
      3  6
      4 10
      5 15
```

Is the boolean condition true?
NO! The loop is over.

The While-Loop Mechanism

```
while A Boolean Expression:
    The Loop Body
```

The Boolean expression is checked. If it is true, then the loop body is executed. The process is repeated until the Boolean expression is false. At that point the iteration terminates.

The Broader Context

```
Code that comes before the loop
while A Boolean Expression:
    The Loop Body
Code that comes after the loop
```

Every variable involved in the Boolean expression must be initialized.

```
Code that comes before the loop
while A Boolean Expression:
    The Loop Body
Code that comes after the loop
```

After the loop terminates the next statement after the loop is executed.

The Broader Context

```
Code that comes before the loop
while A Boolean Expression:
    The Loop Body
Code that comes after the loop
```

Indentation defines the loop body.

Back to Our Example

```
k = 0
s = 0
while k < 5:
    k = k + 1
    s = s + k
    print k, s
```

```
      1  1
      2  3
      3  6
      4 10
      5 15
```

Let's move the print statement outside the loop body.
Back to Our Example

```python
k = 0
s = 0
while k < 5:
    k = k + 1
    s = s + k
print k, s
```

A Modified Problem

Print the smallest $k$ so that the sum of the first $k$ whole numbers is greater than 50.

The answer is 10 since

1+2+3+4+5+6+7+8+9 = 45
and
1+2+3+4+5+6+7+8+9+10 = 55

"Discovering" When to Quit

```python
k = 0
s = 0
while s < 50:
    k = k + 1
    s = s + k
print k, s
```

While loops can handle iterative situations even if we do not know the required number of repetitions.

"Discovering" When to Quit

```python
k = 0
s = 0
while s < 50:
    k = k + 1
    s = s + k
print k, s
```

"Discovering" When to Quit

The boolean condition says "OK"

```python
k = 0
s = 0
while s < 50:
    k = k + 1
    s = s + k
print k, s
```

"Discovering" When to Quit

```python
k = 0
s = 0
while s < 50:
    k = k + 1
    s = s + k
print k, s
```

"Discovering" When to Quit

```python
k = 0
s = 0
while s < 50:
    k = k + 1
    s = s + k
print k, s
```
"Discovering" When to Quit

\[
\begin{align*}
k &= 0 \\
\text{s} &= 0 \\
\text{while } s < 50: & \quad k \rightarrow 10 \\
& \quad k = k + 1 \\
& \quad s = s + k \\
& \quad \text{print } k, s \\
\text{s} & \rightarrow 55
\end{align*}
\]

The boolean condition now says "stop"

Control passes to the next statement after the end of the loop body

Defining Variables

\[
\begin{align*}
k &= 0 \\
\text{s} &= 0 \\
\text{while } s < 50: & \quad k \rightarrow 10 \\
& \quad k = k + 1 \\
& \quad s = s + k \\
& \quad \text{print } k, s
\end{align*}
\]

The "property" that \( s \) is the sum of the first \( k \) whole numbers is invariant throughout the iteration. Defining variables in this fashion promotes correctness.

Let's Revisit the sqrt Problem Again!

For-Loop Solution

```python
def sqrt(x):
    x = float(x)
    L = x
    W = 1
    for k in range(5):
        L = (L + W)/2
        W = x/L
    return L
```

The number of iterations is "hardwired" into the implementation.
5 may not be enough -- an accuracy issue.
5 may be too big -- efficiency issue.

What we Really Want

```python
def sqrt(x):
    x = float(x)
    L = x
    W = 1
    for k in range(5):
        L = (L + W)/2
        W = x/L
    return L
```

Iterate until \( L \) and \( W \) are really close.
**What we Really Want**

*Not this:*

```python
for k in range(5):
    L = (L + W)/2
    W = x/L
```

*But this:*

```python
while abs(L-W)/L > 10**-12
    L = (L + W)/2
    W = x/L
```

**What we Really Want**

This says "keep iterating as long as the discrepancy relative to \( L \) is bigger than \( 10^{-12} \)"

**Template for doing something an Indefinite number of times**

```python
# Initializations
while not stopping condition:
    # do something
```

When the loop terminates, the discrepancy relative to \( L \) will be less than \( 10^{-12} \)

**A Common Mistake**

```python
while abs(L-W)/L < 10**-12
    L = (L + W)/2
    W = x/L
```

Forgetting that we want a NOT stopping condition

**The Up/Down Sequence Problem**

Pick a random whole number between one and a million. Call the number \( n \) and repeat this process until \( n == 1 \):

- if \( n \) is even, replace \( n \) by \( n/2 \).
- if \( n \) is odd, replace \( n \) by \( 3n+1 \)
The Up/Down Sequence Problem

<table>
<thead>
<tr>
<th>99</th>
<th>741</th>
<th>157</th>
<th>20</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>298</td>
<td>2224</td>
<td>472</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>149</td>
<td>1112</td>
<td>136</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>438</td>
<td>556</td>
<td>68</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>219</td>
<td>278</td>
<td>34</td>
<td>8</td>
<td>etc</td>
</tr>
<tr>
<td>658</td>
<td>139</td>
<td>17</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>329</td>
<td>418</td>
<td>52</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>988</td>
<td>209</td>
<td>26</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>494</td>
<td>628</td>
<td>13</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>247</td>
<td>314</td>
<td>40</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

The Central Repetition

```python
if m%2 == 0:
    m = m/2
else:
    m = 3*m+1
```

Note cycling once m==1:

1, 4, 2, 1, 4, 2, 1, 4, 2, 1, 4, 2, 1, ...

Shuts Down When m==1

```python
n = input('m = ')
m = n
nSteps = 0
while m > 1:
    if m%2==0:
        m = m/2
    else:
        m = 3*m + 1
    nSteps = nSteps+1
print n,nSteps,m
```

Avoiding Infinite Loops

```python
nSteps = 0
maxSteps = 200
while m > 1 and nSteps<maxSteps:
    if m%2==0:
        m = m/2
    else:
        m = 3*m + 1
    nSteps = nSteps+1
```

Introduce Boolean-Valued Functions

The Boolean condition that controls a while loop can be very complicated.

It is sometimes a good idea to simplify things using Boolean-valued functions.

An Example: Looking for Patterns in a “Coin Toss” String

```python
S = 'HHTHTTHHTHTHTHTH'
Made of of H's and T's
```
Generating a Coin Toss String

```python
def GenCoinToss(n):
    s = ''
    for k in range(n):
        i = randi(1,2)
        if i==1:
            s = s + 'H'
        else:
            s = s + 'T'
    return s
```

Repetitive concatenation with random choice for H or T

Let's Look for 'Sandwiches' in a Coin Toss String

A length-\(m\) sandwich string is if either
- its first and last characters are 'H' and all the rest are T's
- or its first and last characters are T and the rest are H's.

```
HTTTTHT  THHHHHHHHHHHHT
```

A Boolean-Valued Function

```python
def isSandwich(t):
    n = len(t)
    Meat = t[1:n-1]
    Type1 = s[0]=='H' and s[n-1]=='H'
    Type1 = Type1 and Meat.count('T')==n-2
    Type2 = s[0]=='T' and s[n-1]=='T'
    Type2 = Type2 and Meat.count('H')==n-2
    return Type1 or Type2
```

This is an assignment statement:

```
Type1 = s[0]=='H' and s[n-1]=='H'
```

This expression evaluates to True or False.
The result is stored in Type1

Look for a Length-5 Sandwich

```python
s = some long coin toss string
k = 0
n = len(s)
t = s[0:5]
while k+5<=n and (not isSandwich(t)):
    k+=1
    t = s[k:k+5]
if k+5==n+1:
    print 'there is no sandwich'
else:
    print 'there is a sandwich'
```

The While Condition

```python
s = some long coin toss string
k = 0
n = len(s)
T = s[0:5]
while k+5<=n and (not isSandwich(t)):
    k+=1
    t = s[k:k+5]
```

Keep iterating as long as \(k+5\leq n\) AND \(t\) is NOT a sandwich.
When the Loop Ends

\[
s = \text{some long coin toss string} \\
k = 0 \\
n = \text{len}(s) \\
T = s[0:5] \\
\text{while } k+5 \leq n \text{ and (not isSandwich(t))}: \\
\quad k+=1 \\
\quad t = s[k:k+5] \\
\]

Either \(k+5 = n+1\) or \(t\) is a sandwich

Look for a Length-5 Sandwich

\[
s = \text{some long coin toss string} \\
k = 0 \\
n = \text{len}(s) \\
t = s[0:5] \\
\text{while } k+5 \leq n \text{ and (not isSandwich(t))}: \\
\quad k+=1 \\
\quad t = s[k:k+5] \\
\text{if } k+5 = n+1: \\
\quad \text{print 'there is no sandwich'} \\
\text{else} \\
\quad \text{print 'there is a sandwich'} \\
\]