11. Iteration: The while-Loop

**Topics:**
- Open-Ended repetition
- the while statement
- Example 1: The sqrt Problem
- Example 2: The UpDown Sequence
- Example 3: The Fibonacci Sequence

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.

Compute the square root of 2….

L = 2; W = 1
Repeat this until |L-W| <= .000001:
  L = (L + W)/2
  W = x/L

How Does a While-Loop Work?

A simple warm-up example:

Sum the first 5 whole numbers and display the summation process.

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.

Two Solutions

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

s = 0
for k in range(1,6):
  s = s + k
print k,s
The While-Loop Solution

```
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

```
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

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

At the start, k and s are initialized

Trace the Execution

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

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

\[ k = 0 \]
\[ s = 0 \]

while \( k < 5 \):
  \[ k = k + 1 \]
  \[ s = s + k \]

print \( k, s \)

Is the boolean condition true?

Yes, so execute the loop body

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

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

Is the boolean condition true?

Is the boolean condition true?

Yes, so execute the loop body

Yes, so execute the loop body
Trace the Execution

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

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.

The Broader Context

```
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
```

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

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

```
5  15
```

Only the final value of k and s are reported.

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

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

```
10  55
```

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

“Discovering” When to Quit

Suppose this is the situation:

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

```
k -> 9
s -> 45
```

“Discovering” When to Quit

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

```
k -> 10
s -> 55
```

The boolean condition says "OK"
"Discovering" When to Quit

The boolean condition now says "stop"

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

```
k -> 10
s -> 55
```

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

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

```
k -> 10
s -> 55
```

Defining Variables

```
k = 0
s = 0
while s < 50:
    # s is the sum 1+ ... + k
    k = k + 1
    s = s + k
print k, s
```

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.

Example 1

The Square Root Problem (Again!)

For-Loop Solution

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

```
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 be too big -- efficiency issue

5 may not be enough -- accuracy issue

What we Really Want

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

```
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

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

But this:

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

This says:
"Keep iterating as long as the discrepancy relative to L is bigger than 10**(-12)"

What we Really Want

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

When the loop terminates, the discrepancy relative to L will be less than 10**(-12)

Template for doing something an Indefinite number of times

# Initializations

while not-stopping condition :

    # do something

A Common Mistake

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

Forgetting that we want a "NOT stopping" condition

Example 2

The "Up/Down" Sequence
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 \( 3n + 1 \)

```
99      741     157      20       1
298     2224     472      10       4
149     1112     136       5       2
438      556      68      16       1
219     278     34       8      etc
658     139      17       4
329      418      52       2
988     209      26       1
494      628      13       4
247      314     40       2
```

The Central Repetition

```
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 \( m = 1 \)

```
n = input('m = ')  # nSteps keeps track of the number of steps
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

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

Example 3

Fibonacci Numbers and the Golden Ratio
Fibonacci Numbers and the Golden Ratio

Here are the first 12 Fibonacci Numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144

The Fibonacci ratios 1/1, 2/1, 3/2, 5/3, 8/5 get closer and closer to the “golden ratio”

\[ \phi = \frac{1 + \sqrt{5}}{2} \]

Generating Fibonacci Numbers

Here are the first 12 Fibonacci Numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144

Starting here, each one is the sum of its two predecessors

```
x = 0
y = 1
for k in range(10):
z = x+y
x = y
y = z
```

Generating Fibonacci Numbers

```
x = 0
y = 1
for k in range(10):
z = x+y
x = y
y = z
```

Generating Fibonacci Numbers

```
x = 0
y = 1
for k in range(10):
z = x+y
x = y
y = z
```

Generating Fibonacci Numbers

```
x = 0
y = 1
for k in range(10):
z = x+y
x = y
y = z
```
Generating Fibonacci Numbers

\[0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144\]

\[
x = 0 \\
y = 1 \\
\text{for } k \text{ in range(10)}: \\
z = x + y \\
x = y \\
y = z
\]

```
x = 0 
print x 
y = 1 
print y 
for k in range(6): 
    z = x + y 
    x = y 
    y = z
```

\[0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144\]

\[
x = 0 \\
y = 1 \\
\text{for } k \text{ in range(10)}: \\
z = x + y \\
x = y \\
y = z
\]

\[0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144\]

\[
x = 0 \\
y = 1 \\
\text{for } k \text{ in range(10)}: \\
z = x + y \\
x = y \\
y = z
\]

\[0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144\]

\[
x = 0 \\
y = 1 \\
\text{for } k \text{ in range(10)}: \\
z = x + y \\
x = y \\
y = z
\]
Generating Fibonacci Numbers

x = 0
print x
y = 1
print y
for k in range(6):
z = x+y
x = y
y = z
print z

x = 0
print x
y = 1
print y
k = 0
while k<6:
z = x+y
x = y
y = z
print z
k = k+1

Print First Fibonacci Number
>= 1000000

x = 0
y = 1
z = x+y
x = y
y = z
print y

x = 0
y = 1
z = x+y
x = y
y = z
print z
k = k+1

Print First Fibonacci Number
>= 1000000

past = 0
current = 1
next = past + current
while current < 1000000:
past = current
current = next
next = past + current
print current

1346269

Reasoning. When the while loop terminates, it will be the first time that current >= 1000000 is true. By print out current we see the first fib >= million

Print Largest Fibonacci Number < 1000000

past = 0
current = 1
next = past + current
while next < 1000000:
past = current
current = next
next = past + current
print current

832040

Reasoning. When the while loop terminates, it will be the first time that next >= 1000000 is true. print out current we see the largest fib with this property

Print Largest Fibonacci Number < 1000000

past = 0
current = 1
next = past + current
while next < 1000000:
past = current
current = next
next = past + current
print current

832040

Reasoning. When the while loop terminates, it will be the first time that next >= 1000000 is true. Current has to be < 1000000. And it is the largest fib with this property.
Fibonacci Ratios

past = 0
current = 1
next = past + current
while next <= 1000000:
    past = current
    current = next
    next = past + current
print next/current

Heading towards the Golden ratio = (1+sqrt(5))/2

Most Pleasing Rectangle

(1+sqrt(5))/2