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

```
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
```

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

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

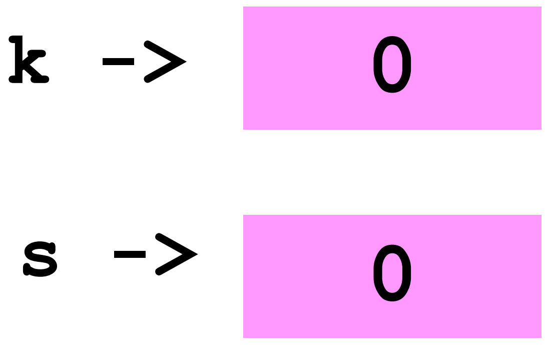
```
k = 0
s = 0
while k < 5:
    k = k + 1
    s = s + k
    print k,s
```

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

We call this the “loop body”

Trace the Execution

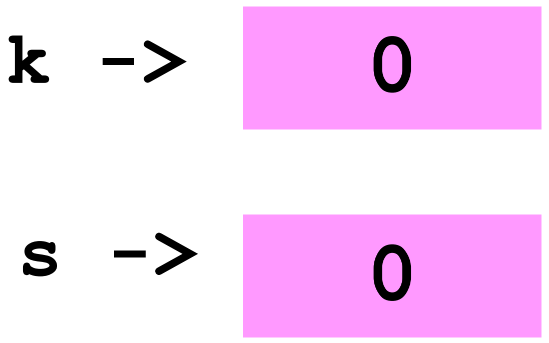
```
k = 0
s = 0
while k < 5:
    k = k + 1
    s = s + k
    print k,s
```



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
```

k ->	0
s ->	0

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
```

```
k -> 1

s -> 1
```

```
1 1
```

Trace the Execution

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

```
k -> 1

s -> 1
```

```
1 1
```

Is the boolean condition true?

Trace the Execution

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

```
k -> 1

s -> 1
```

```
1 1
```

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
```

k -> 2

s -> 3

```
1  1
2  3
```

Trace the Execution

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

k -> 2

s -> 3

```
1  1
2  3
```

Is the boolean condition true?

Trace the Execution

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

k -> 2

s -> 3

```
1  1
2  3
```

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
```

k -> 3

s -> 6

```
1  1
2  3
3  6
```

Trace the Execution

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

k -> 3

s -> 6

```
1  1
2  3
3  6
```

Is the boolean condition true?

Trace the Execution

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

```
k -> 3

s -> 6
```

```
1 1
2 3
3 6
```

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
```

k -> 4

s -> 10

```
1  1
2  3
3  6
4  10
```

Trace the Execution

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

```
k -> 4

s -> 10
```

```
1  1
2  3
3  6
4  10
```

Is the boolean condition true?

Trace the Execution

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

k -> 4

s -> 10

```
1  1
2  3
3  6
4  10
```

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
```

k -> 5

s -> 15

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

Trace the Execution

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

k -> 5

s -> 15

```
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.

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
```

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

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

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

Suppose this is the situation:

```
k ->   9

s ->  45
```

“Discovering” When to Quit

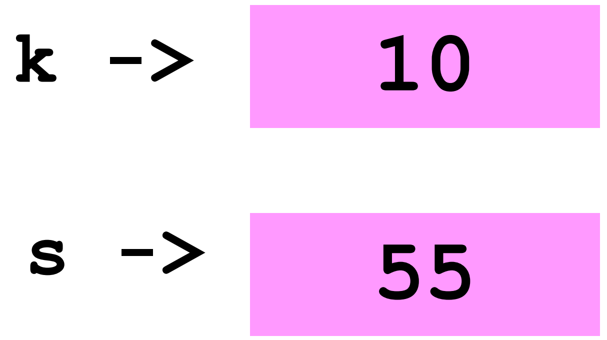
```
k = 0
s = 0
while s < 50:
    k = k + 1
    s = s + k
print k,s
```

The boolean condition says “OK”

k ->	9
s ->	45

“Discovering” When to Quit

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



"Discovering" When to Quit

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

The boolean condition now says "stop"

k ->	10
s ->	55

"Discovering" When to Quit

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

```
k ->  10

s ->  55
```

```
10   55
```

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

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.

Let's Revisit the sqrt 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
```

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

```
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:

```
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
```

What we Really Want

```
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

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

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 = ')
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
```

nSteps keeps track of the number of steps

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
```

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

`s = 'HHTHTTHHTHHTHTTH'`

Made of of `H`'s and `T`'s

Generating a Coin Toss String

```
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
```

Repeated concatenation with random choice for H and T

Let's Look for 'Sandwiches' in a CoinToss String

t is length-m sandwich string 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,

`HTTTTTH` `THHHHHHHHHHHHT`

A Boolean-Valued Function

```
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
```

Type1 and Type2 are Boolean Variables

Boolean Variables

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

```
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

```
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<=n AND
t is NOT a sandwich.

When the Loop Ends

```
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]
```

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

Look for a Length-5 Sandwich

```
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'
```