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

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
```

k -> 0

s -> 0

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
```

k -> 0

s -> 0

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
```

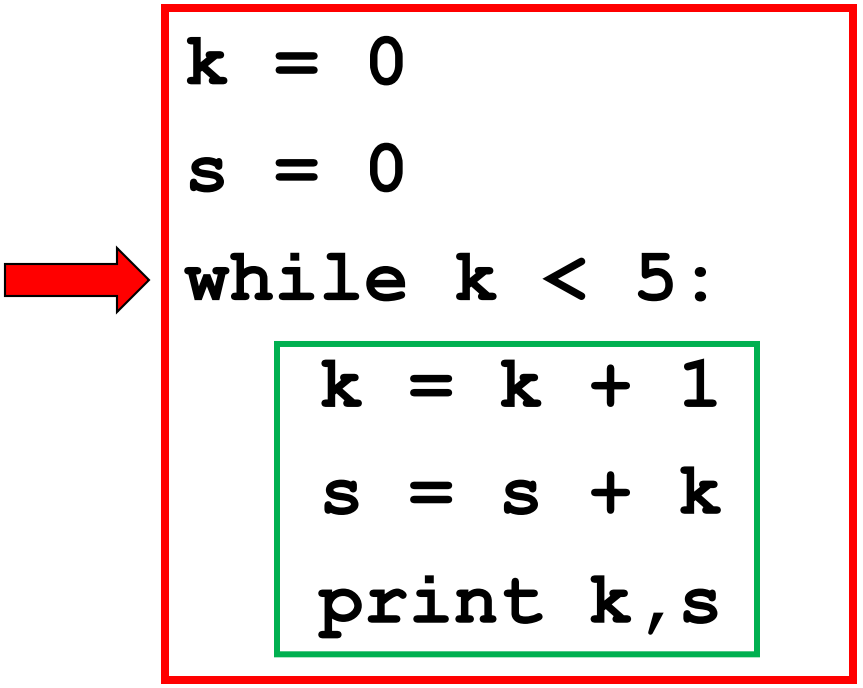
k -> 1

s -> 1

1 1

Is the boolean condition true?

Trace the Execution



k -> 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 ->

s ->


```
1  1
2  3
```

Trace the Execution

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

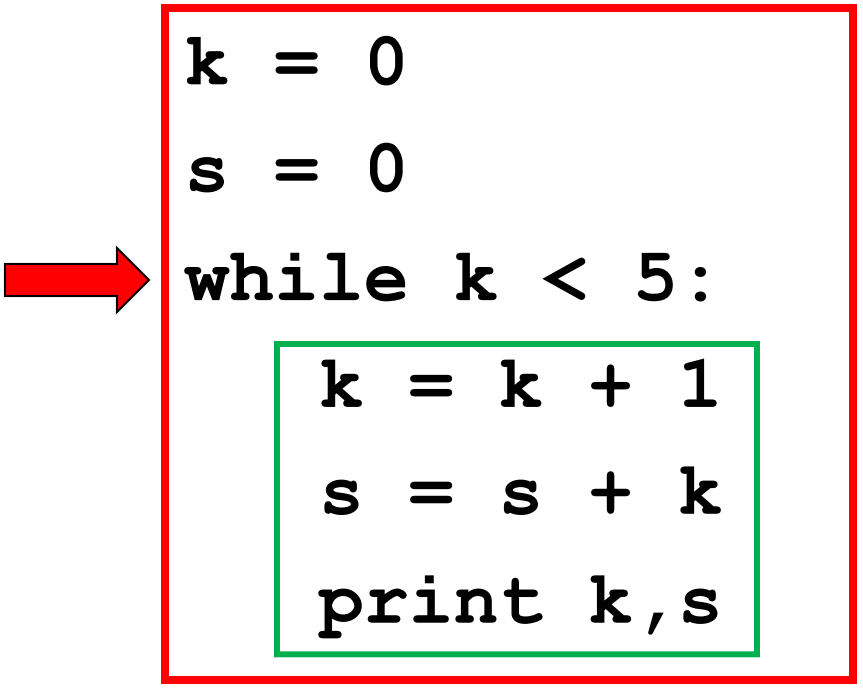
k -> 2

s -> 3

```
1  1
2  3
```

Is the boolean condition true?

Trace the Execution



k -> 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 ->

s ->


```
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 ->

s ->


```
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
```

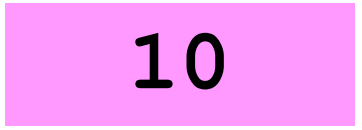
Yes, so execute the loop body

Trace the Execution

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

k ->

s ->




```
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
```

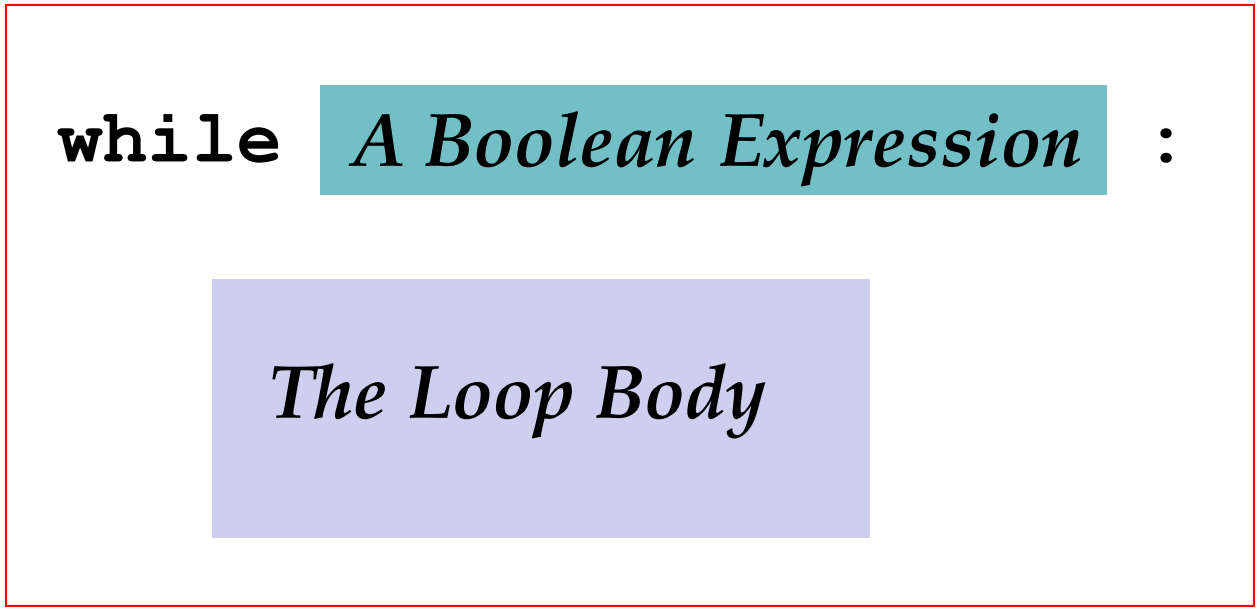
k ->

s ->


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

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

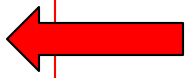
The While-Loop Mechanism



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.

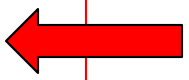
The Broader Context

Code that comes before the loop

`while` *A Boolean Expression* `:`

The Loop Body

Code that comes after the loop



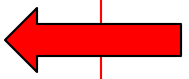
After the loop terminates the next statement after the loop is executed.

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
```

k -> 10

s -> 55

"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

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

```
10   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
```

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

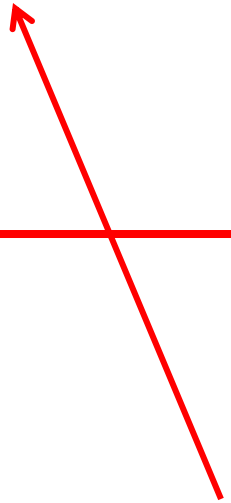
```
# Initializations
```

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

Example 2

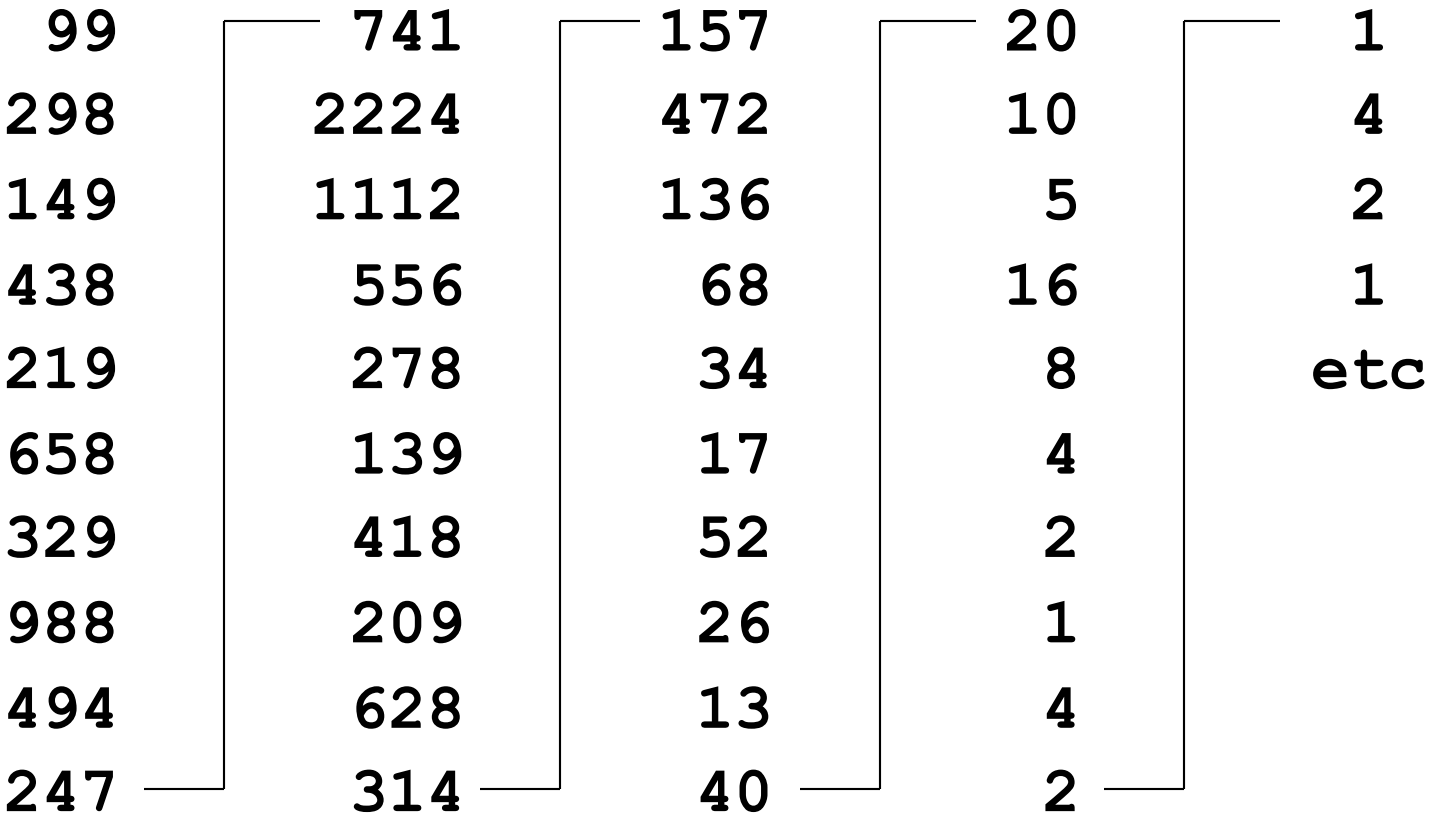
The "Up/Down" Sequence

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

The Up/Down Sequence Problem



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
```

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"

$$\text{phi} = (1 + \text{sqrt}(5))/2$$

Fibonacci Ratios 2/1, 3/2, 5/3, 8/5

1

2

2

3

3

5

5

8

Generating Fibonacci Numbers

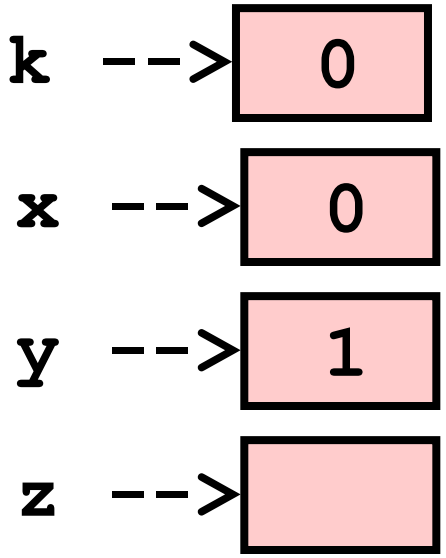
Here are the first 12 Fibonacci Numbers

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

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

Generating Fibonacci Numbers

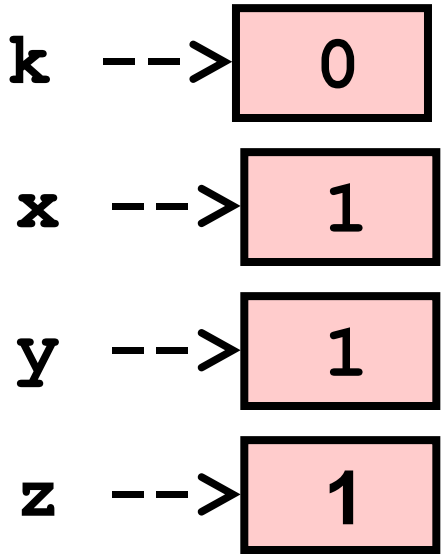
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144




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

Generating Fibonacci Numbers

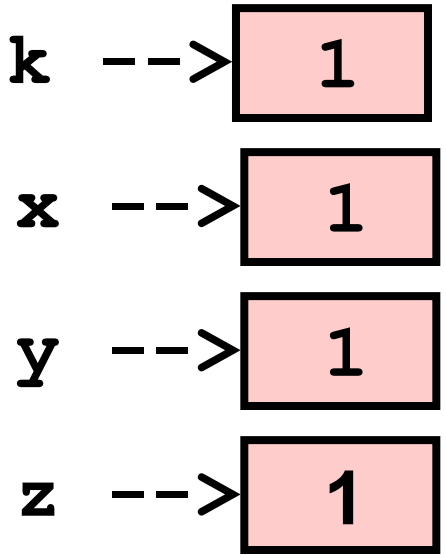
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144




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

Generating Fibonacci Numbers

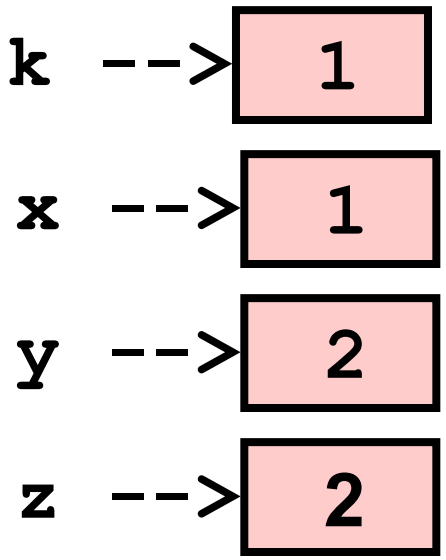
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144




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

Generating Fibonacci Numbers

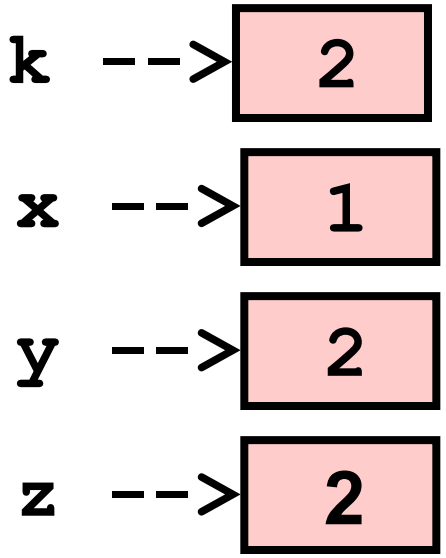
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144




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

Generating Fibonacci Numbers

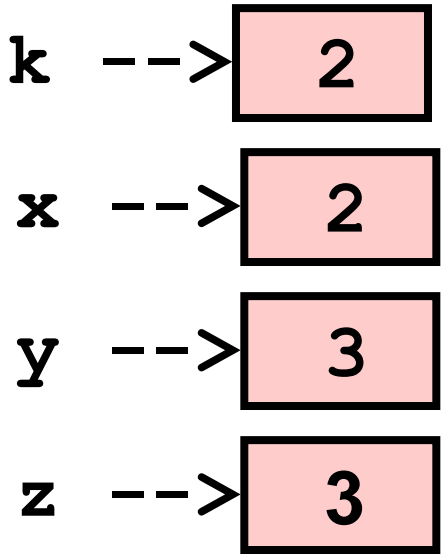
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144




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

Generating Fibonacci Numbers

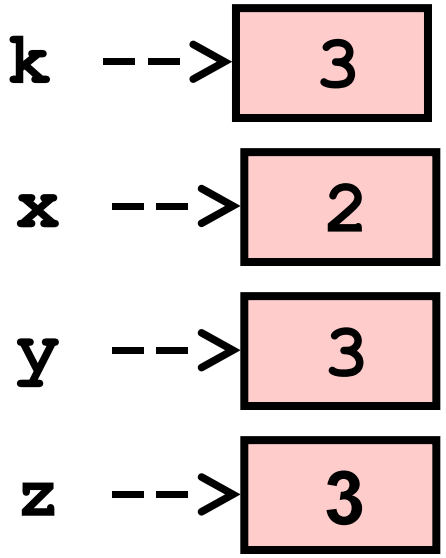
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144




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

Generating Fibonacci Numbers

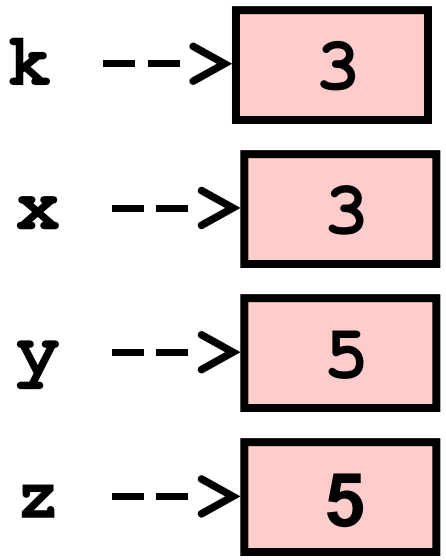
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144




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

Generating Fibonacci Numbers

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




```
x = 0
y = 1
for k 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
```

```
0
1
1
2
3
5
8
13
```

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
while y < 1000000:
    x = y
    y = z
    z = x + y
print y
```

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
```

Print First Fibonacci Number >= 1000000

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

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

Print Largest Fibonacci Number < 1000000

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

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
```

```
1.000000000000
2.000000000000
1.500000000000
1.666666666667
1.600000000000
1.625000000000
1.615384615385
1.619047619048
1.617647058824
1.618181818182
1.617977528090
1.618055555556
1.618025751073
1.618037135279
1.618032786885
      :
```

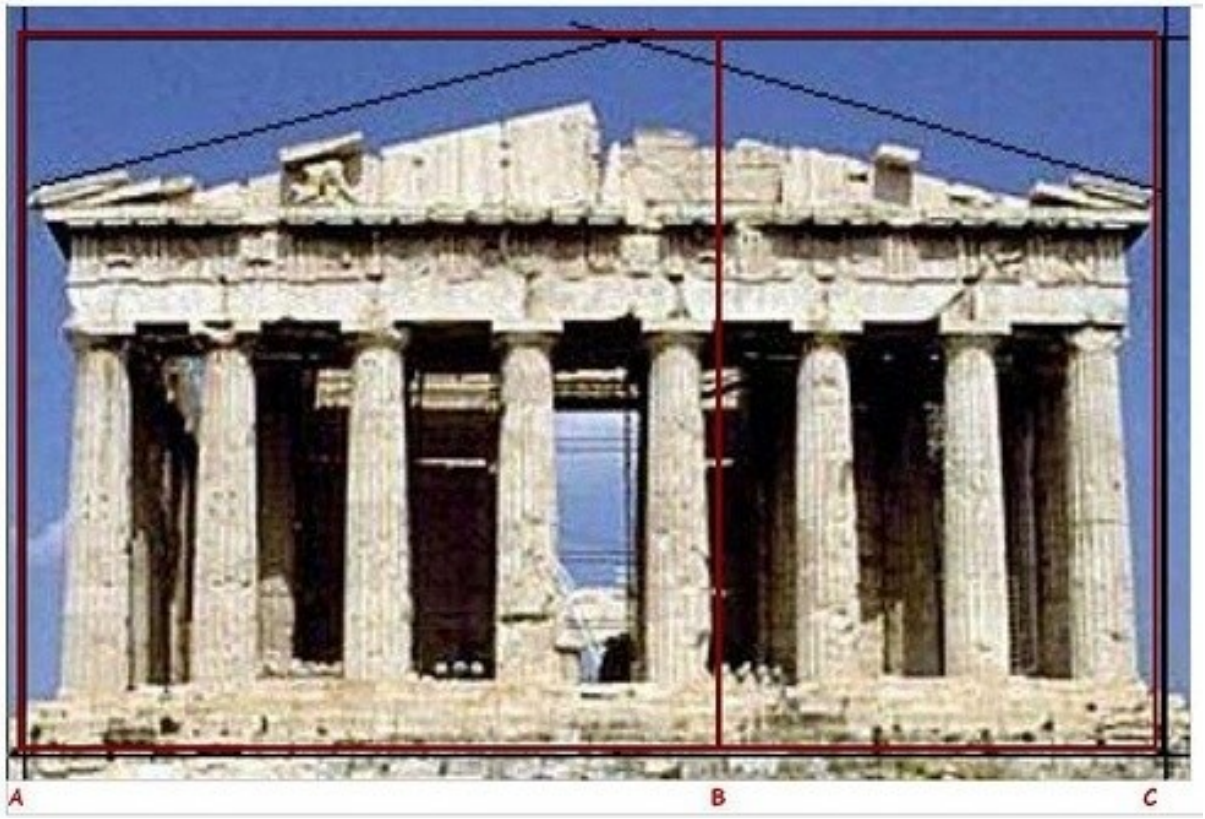
Heading towards the
Golden ratio = (1+sqrt(5))/2

Fibonacci Ratios

```
past = 0
current = 1
next = past + current
k = 1
phi = (1+math.sqrt(5))/2
while abs(next/current - phi) > 10**-9
    past = current
    current = next
    next = past + current
    k = k+1
print k,next/current
```

23 1.618033988749

Most Pleasing Rectangle



1

(1+sqrt(5))/2