Meet UTA Jarrett’s dog Greta, lying in her “nest”
Indefinite Loops

Based in part on notes from the CS-for-All curriculum developed at Harvey Mudd College
So far: Two Types of for Loops

vals = [3, 15, 17, 7]

def sum(vals):
    result = 0
    for x in vals:
        result += x
    return result

[element]-based loop

vals = [3, 15, 17, 7]

def sum(vals):
    result = 0
    for i in range(len(vals)):
        result += vals[i]
    return result

[index]-based loop

Both are examples of definite loops (i.e., fixed number of iterations)
**Indefinite Loops**

- Use an *indefinite loop* when the # of repetitions you need is:
  - not obvious or known
  - impossible to determine before the loop begins, e.g.,
    - Finding an element
    - Computing an estimate up to some error bound
    - Playing a game of rock, paper, scissors (as opposed to one round)

- Toy problem: `print_multiples(n, bound)`
  - should print all multiples of `n` that are less than `bound`
  - output for `print_multiples(9, 100)`: 9 18 27 36 45 54 63 72 81 90 99
def print_multiples(n, bound):
    mult = n
    while mult < bound:
        print(mult, end=" ")
        mult = mult + n
    print()
while Loops

while <loop test>:
    <body of the loop>

Steps:
1. evaluate the loop test (a boolean expression)
2. if it's True, execute the statements in the body, and go back to step 1
3. if it's False, skip the statements in the body and go to the statement after the loop
Tracing a while Loop

- Let's trace the loop for `print_multiples(15, 70):

  ```
  mult = n
  while mult < bound:
    print(mult, end=' ')
    mult = mult + n
  print()
  ```

  `mult < bound`  `output thus far`  `mult`
Tracing a while Loop

• Let's trace the loop for `print_multiples(15, 70):

  ```python
  mult = n
  while mult < bound:
    print(mult, end=' ')
    mult = mult + n
  print()
  ```

<table>
<thead>
<tr>
<th>mult &lt; bound</th>
<th>output thus far</th>
<th>mult</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 &lt; 70 (True)</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>30 &lt; 70 (True)</td>
<td>15 30</td>
<td>30</td>
</tr>
<tr>
<td>45 &lt; 70 (True)</td>
<td>15 30 45</td>
<td>45</td>
</tr>
<tr>
<td>60 &lt; 70 (True)</td>
<td>15 30 45 60</td>
<td>60</td>
</tr>
<tr>
<td>75 &lt; 70 (False)</td>
<td></td>
<td>75</td>
</tr>
</tbody>
</table>

so we exit the loop and `print()`
Important!

• In general, a while loop's test includes a key "loop variable".

• We need to update that loop variable in the body of the loop.

• Failing to update it can produce an infinite loop!

• Recall the loop in print_multiples:

```python
mult = n
while mult < bound:
    print(mult, end=' ')
    mult = mult + n
```

* What is the loop variable?
* Where is it updated?
Important!

• In general, a while loop's test includes a key "loop variable".

• We need to update that loop variable in the body of the loop.

• Failing to update it can produce an infinite loop!

• Recall the loop in print_multiples:

```python
mult = n
while mult < bound:
    print(mult, end=' ')
    mult = mult + n
```

What is the loop variable? **mult**
Where is it updated? **In the body of the loop**
Important!

• In general, a while loop's test includes a key "loop variable".

• We need to update that loop variable in the body of the loop.

• Failing to update it can produce an infinite loop!

• Showing every iteration makes progress towards making the while loop condition false is one way to show a while loop will terminate
Factorial Using a while Loop

• We don't need an indefinite loop, but we can still use while!

```python
def fac(n):
    result = 1
    while n > 0:
        result *= n
    return result
```

• Let's trace `fac(4)`:

<table>
<thead>
<tr>
<th>n</th>
<th>n &gt; 0</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>False</td>
<td></td>
</tr>
</tbody>
</table>

```python
fac(4)
```

`result` will be 24.
Factorial Using a while Loop

• We don't need an indefinite loop, but we can still use while!

```python
def fac(n):
    result = 1
    while n > 0:
        result *= n
        n = n - 1
    return result
```

• Let's trace fac(4):

<table>
<thead>
<tr>
<th>n</th>
<th>n &gt; 0</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>True</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>True</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>24</td>
</tr>
<tr>
<td>0</td>
<td>False</td>
<td>result</td>
</tr>
</tbody>
</table>

Factorial Using a \texttt{while} Loop

• We don't need an indefinite loop, but we can still use \texttt{while}!

\begin{verbatim}
def fac(n):
    result = 1
    while n > 0:
        result *= n
        n = n - 1
    return result
\end{verbatim}

• Let's trace \texttt{fac(4)}:

\begin{center}
\begin{tabular}{ccc}
\hline
\text{n} & \text{n > 0} & \text{result} \\
4 & 4 > 0 (True) & 1*4 = 4 \\
4 & 4 > 0 (True) & 12*2 = 24 \\
3 & 3 > 0 (True) & 4*3 = 12 \\
2 & 2 > 0 (True) & 12*2 = 24 \\
1 & 1 > 0 (True) & 24*1 = 24 \\
0 & 0 > 0 (False) & \\
\hline
\end{tabular}
\end{center}

so we exit the loop and return 24
Factorial Four Ways!

**recursion**

```python
def fac(n):
    if n == 0:
        return 1
    else:
        rest = fac(n-1)
        return n * rest
```

**for loop**

```python
def fac(n):
    result = 1
    for x in range(1, n+1):
        result *= x
    return result
```

**map**

```python
def fac(n):
    return reduce(lambda x,y : x*y, range(1,max(2,n+1)))
```

**while loop**

```python
def fac(n):
    result = 1
    while n > 0:
        result *= n
        n = n - 1
    return result
```

More on these later!
Extreme Looping!

- What does this code do?

```python
print('It keeps')
while True:
    print('going and')
print('Phew! Done!')
```
Extreme Looping!

• What does this code do?

```
print('It keeps')
while True:
    print('going and')
print('Phew! Done!')  # never gets here!
```

• An infinite loop!

    Use **Ctrl-C** to stop a program inside python

    Use **W-F2** to stop a program in PyCharm
import random

while True:
    print('Help!')
    if random.choice(range(10000)) == 111:
        break
    print('Let me out!')

print('At last!')

• What are the final two lines that are printed?
import random

while True:
    print('Help!')
    if random.choice(range(10000)) == 111:
        break
    print('Let me out!')

print('At last!')

• What are the final two lines that are printed?
  Help!
  At last!

• How could we count the number of repetitions?
Counting the Number of Repetitions

```python
import random

count = 1
while True:
    print('Help!')
    if random.choice(range(10000)) == 111:
        break
    print('Let me out!')
    count += 1

print('At last! It took', count, 'tries to escape!')
```
Important!

• In general, a while loop's test includes a key "loop variable".

• We need to update that loop variable in the body of the loop.

• Failing to update it can produce an \textit{infinite loop}!

• Can rely on a statistical argument (e.g., rock, paper, scissors)

• Counting the number of iterations and exiting after a maximum has been reached is a safer way to loop indefinitely
import random

count = 1
while count<=5000:
    print('Help!')
    if random.choice(range(10000)) == 111:
        break
    print('Let me out!')
    count += 1

print('At last! It took', count, 'tries to escape!')
How many values does this loop print?

```
a = 40
while a > 2:
    a = a // 2
    print(a - 1)
```

A. 2  
B. 3  
C. 4  
D. 5  
E. none of these
How many values does this loop print?

```python
a = 40
while a > 2:
    a = a // 2
    print(a - 1)
```

<table>
<thead>
<tr>
<th>a &gt; 2</th>
<th>a</th>
<th>prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>True</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>True</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>True</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>True</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>False</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A. 2  
B. 3  
C. 4  
D. 5  
E. none of these
For what inputs does this function return True?

```python
def mystery(n):
    while n != 1:
        if n % 2 != 0:
            return False
        n = n // 2
    return True
```

Try tracing these two cases:

- `mystery(12)`
- `mystery(8)`

A. odd numbers
B. even numbers
C. multiples of 4
D. powers of 2
E. none of these
For what inputs does this function return True?

def mystery(n):
    while n != 1:
        if n % 2 != 0:
            return False
        n = n // 2
    return True

Try tracing these two cases:

mystery(12)  mystery(8)

<table>
<thead>
<tr>
<th>n</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>False</td>
<td>1</td>
</tr>
<tr>
<td>True</td>
<td></td>
</tr>
</tbody>
</table>

A. odd numbers
B. even numbers
C. multiples of 4
D. powers of 2
E. none of these
Wesley says it’s break time so it’s break time
Nested Loops!

```python
for y in range(84):
    for m in range(12):
        for d in range(f(m,y)):
            for h in range(24):
                for mn in range(60):
                    for s in range(60):
                        tick()
```
Nested Loops!

- Nested Loops are loops where a loop appears inside the body of another loop.
  - The loop inside the body is called the inner loop. The other is called the outer loop.
- The inner loop completes all passes for a single pass of the outer loop
  - This is very useful for many types of algorithms, especially with data that has more than one dimension.
Repeating a Repetition!

```python
for i in range(3):
    for j in range(4):
        print(i, j)
```

inner loop

outer loop
Repeating a Repetition!

```python
for i in range(3):  # 0, 1, 2
    for j in range(4):  # 0, 1, 2, 3
        print(i, j)

0 0
```
Repeating a Repetition!

```python
for i in range(3):  # 0, 1, 2
    for j in range(4):  # 0, 1, 2, 3
        print(i, j)

0 0
0 1
```

0 0
0 1
Repeating a Repetition!

```python
for i in range(3):  # 0, 1, 2
    for j in range(4):  # 0, 1, 2, 3
        print(i, j)
```

```
0 0
0 1
0 2
```
Repeating a Repetition!

```python
for i in range(3):  # 0, 1, 2
    for j in range(4):  # 0, 1, 2, 3
        print(i, j)
```

0 0
0 1
0 2
0 3
Repeating a Repetition!

```python
for i in range(3):  # 0, 1, 2
    for j in range(4):  # 0, 1, 2, 3
        print(i, j)
```

```
0 0
0 1
0 2
0 3
1 0
1 1
1 2
1 3
```
Repeating a Repetition!

```python
for i in range(3):  # 0, 1, 2
    for j in range(4):  # 0, 1, 2, 3
        print(i, j)
```

0 0
0 1
0 2
0 3
1 0
1 1
1 2
1 3
2 0
2 1
2 2
2 3
Repeating a Repetition!

```python
for i in range(3):
    for j in range(4):
        print(i, j)
    print('---')
```

inner loop  

outer loop
Repeating a Repetition!

```python
for i in range(3):
    for j in range(4):
        print(i, j)
    print('---')
```

inner loop

outer loop

0 0
0 1
0 2
0 3
---
1 0
1 1
1 2
1 3
---
2 0
2 1
2 2
2 3
---
How many lines are printed?

for i in range(5):
    for j in range(7):
        print(i, j)

A. 4
B. 5
C. 7
D. 24
E. 35
How many lines are printed?

```python
for i in range(5):
    for j in range(7):
        print(i, j)
```

A. 4
B. 5
C. 7
D. 24
E. \(35 = 5 \times 7\) executions of inner code block
Tracing a Nested for Loop

for i in range(5):  # [0,1,2,3,4]
    for j in range(i):
        print(i, j)

i  range(i)  j  value printed
Tracing a Nested for Loop

```python
for i in range(5):  # [0,1,2,3,4]
    for j in range(i):
        print(i, j)
```

<table>
<thead>
<tr>
<th>i</th>
<th>range(i)</th>
<th>j</th>
<th>value printed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[]</td>
<td>none</td>
<td>nothing (we exit the inner loop)</td>
</tr>
<tr>
<td>1</td>
<td>[0]</td>
<td>0</td>
<td>1 0</td>
</tr>
<tr>
<td>2</td>
<td>[0,1]</td>
<td>0</td>
<td>2 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2 1</td>
</tr>
<tr>
<td>3</td>
<td>[0,1,2]</td>
<td>0</td>
<td>3 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>3 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>3 2</td>
</tr>
<tr>
<td>4</td>
<td>[0,1,2,3]</td>
<td>0</td>
<td>4 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>4 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>4 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>4 3</td>
</tr>
</tbody>
</table>

full output:
```
1 0
2 0
2 1
3 0
3 1
3 2
4 0
4 1
4 2
4 3
```
Second Example: Tracing a Nested for Loop

for i in range(4):
    for j in range(i, 3):
        print(i, j)
    print(j)

i     range(i, 3)     j     value printed
Second Example: Tracing a Nested for Loop

```python
for i in range(4):  # [0, 1, 2, 3]
    for j in range(i, 3):
        print(i, j)
        print(j)
    # would go here next

i  range(i, 3)  j  value printed
0  [0, 1, 2]  0  0  0
    1  0  0  1
    2  0  0  2
1  [1, 2]  1  1  1
    2  1  1  2
2  [2]  2  2  2
3  [] , so body of inner loop doesn't execute 2
```

full output:

```
0 0
0 1
0 2
0 2
1 1
1 2
2
2
2
2
```
Side Note: Staying on the Same Line When Printing

• By default, `print` puts an invisible `newline` character at the end of whatever it prints.
  • causes separate `print`s to print on different lines

• Example: What does this output?

```python
for i in range(7):
    print(i * 5)
```

0
5
10
15
20
25
30
Staying on the Same Line When Printing (cont.)

• To get separate prints to print on the same line, we can replace the newline with something else.

• Examples:

```python
for i in range(7):
    print(i * 5, end=' ')

0 5 10 15 20 25 30

for i in range(7):
    print(i * 5, end=',,')

0,5,10,15,20,25,30,
for row in range(3):
    for col in range(4):
        print('#', end=' ')  
    print()  # go to next line

<table>
<thead>
<tr>
<th>row</th>
<th>col</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
for row in range(3):
    for col in range(6):
        print(_____, end=' ')
    print()  # go to next line
for row in range(3):
    for col in range(6):
        print(col, end=' ')
    print()  # go to next line
for row in range(3):
    for col in range(6):
        print(_____, end=' ')
print()  # go to next line

<table>
<thead>
<tr>
<th>col</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>1 1 1 1 1 1</td>
</tr>
<tr>
<td>2 2 2 2 2 2</td>
</tr>
</tbody>
</table>
Fill in the Blank #2

```python
for row in range(3):
    for col in range(6):
        print(row, end=' ')
    print()  # go to next line
```

<table>
<thead>
<tr>
<th>row</th>
<th>col</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
What is needed in the blanks to get this pattern?

```python
for row in range(5):
    for col in __________: 
        print(_____, end=' ')  
    print()  # go to next line
```

<table>
<thead>
<tr>
<th>first blank</th>
<th>second blank</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. range(row) row</td>
<td>row</td>
</tr>
<tr>
<td>B. range(row) col</td>
<td>col</td>
</tr>
<tr>
<td>C. range(5 - row) row</td>
<td>row</td>
</tr>
<tr>
<td>D. range(5 - row) col</td>
<td></td>
</tr>
<tr>
<td>E. none of the above</td>
<td></td>
</tr>
</tbody>
</table>
What is needed in the blanks to get this pattern?

```
for row in range(5):
    for col in __________:
        print(_____, end=' ')
    print()  # go to next line
```

0 0 0 0 0
1 1 1 1
2 2 2
3 3
4

A. range(row) row  
B. range(row) col  
C. range(5 - row) row  
D. range(5 - row) col  
E. none of the above