Beyond sequential execution

• So far, all our programs have looked like this:

<do thing 1>
<do thing 2>
<do thing 3>
...

Start with first command. Execute commands in order until there are no more.

But often sequential execution is not enough.

if <something>:
  <do thing 1>
else:
  <do thing 2>

If something is true, execute the first command. Otherwise, execute the second command.
Control statements

• **Conditionals**: control **which** set of statements is executed.
  – if / else

• **Iteration**: control **how many** times a set of statements is executed.
  – while loops
  – for loops
The if statement

- If the condition is True, the body gets executed.
- Otherwise, nothing happens.

```python
if x < 0:
    print 'x is negative'
```

- NOTE: IDLE editor helps with indentation.
The if/else statement

- If the condition is True, body1 gets executed.
- Otherwise, **body2** gets executed.

```python
if CONDITION:
   BODY1
else:
   BODY2
```

```python
if x < 0:
   print 'x is negative'
else:
   print 'x is positive or zero'
```
Chained conditionals

- If the condition1 is True, body1 gets executed.
- Otherwise, if condition2 is True, body2 gets executed.
- If neither condition is True, body3 gets executed.
Chained conditionals

- if temp $x < 0$:
  - print "x is negative"
- elif $x == 0$:
  - print "x is zero"
- else:
  - print "x is positive"
An example

```python
a = False
b = True
if a and b:
    print 'I love red.'
elif a or b:
    print 'I love green.'
else:
    print 'I love blue.'
    print 'I also love purple.'
```

What does this output?  I love green.
An example

```python
a = False
b = True
if a and b:
    print 'I love red.'
elif a or b:
    print 'I love green.'
else:
    print 'I love blue.'
print 'I also love purple.'
```

What does this output?

I love green.
I also love purple.
Nested conditionals

```python
if is_adult:
    if is_senior_citizen:
        print 'Admission $2 off.'
    else:
        print 'Full price.'
else:
    print 'Admission $5 off.'
```

- Can get confusing. Indentation helps to keep the code readable and the python interpreter happy!
Another example

```python
x = 4
y = -3
if x < 0:
    if y > 0:
        print x + y
    else:
        print x - y
else:
    print x
```

What does this output? 

-12
Common if errors

• Syntax errors
  – Mixing up = and == in the condition

```python
b = False
if b = False
    print b
print 'inside if maybe'
```

SyntaxError: invalid syntax

IndentationError: unindent does not match any outer indentation level
The while loop

- As long as the condition is true, the body gets executed repeatedly.
- The first time the condition is false, execution ends.
The while loop

```python
i = 0
while i < 3:
    print i
    i = i + 1
```

• What does this output?

0
1
2

Side note: if the condition is false the first time it is tested, the body is never executed
The break statement

- Immediately exits the innermost loop.

```python
i = 0
while True:
    i+=1
    line = raw_input('>>> ')
    if line == 'done':
        break
    print i
print 'Done!'
```

(An if statement is not a loop!)
What' will happen with this code?

```
i = 0
while i < 3:
    print i
```

- It will loop forever (aka Infinite loop)! How do we fix it?

```
i = 0
while i < 3:
    print i
    i = i + 1
```
The infinite loop

This code also loops forever! Why? And how do you fix this?

```
i = 4
while i > 0:
   print i
   i = i + 1
```

```
i = 4
while i > 0:
   print i
   i = i - 1
```
The for loop

- Example:

```python
for i in [0, 1, 2, 3):
    print i
```

Sequence of values – list, string, etc.

Sequence element

Any set of statements

Indentation is important

For ELEMENT in SEQUENCE:
BODY

Sequence element

Sequence of values – list, string, etc.
Using range

index variable

for INDEX in range(n):
   BODY

any set of statements

generates sequence of n values starting at 0 and incrementing by 1

• What does this output?

```
for i in range(4):
   sq = i * i
   print i, sq
```

<table>
<thead>
<tr>
<th>i</th>
<th>sq</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>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>
Using range

index variable

for INDEX in range([start], stop, [step]):
   BODY

any set of statements
generates sequence of values
start and step are optional

• What does this output?

for i in range(1, 7, 2):
   print i
For loop and strings

• Iterating through the characters of a string

```python
str1 = 'stressed'
for c in str1:
    print c,
```

stressed
For loop and strings

• What is the output?

```python
str1 = 'stressed'
res = ''
for c in str1:
    res = c + res
print res
```

desserts

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>c</th>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>1</td>
<td>t</td>
<td>ts</td>
</tr>
<tr>
<td>2</td>
<td>r</td>
<td>rts</td>
</tr>
<tr>
<td>3</td>
<td>e</td>
<td>erts</td>
</tr>
<tr>
<td>4</td>
<td>s</td>
<td>serts</td>
</tr>
<tr>
<td>5</td>
<td>s</td>
<td>sserts</td>
</tr>
<tr>
<td>6</td>
<td>e</td>
<td>esserts</td>
</tr>
<tr>
<td>7</td>
<td>d</td>
<td>desserts</td>
</tr>
</tbody>
</table>
Combining for and if

for i in range(6):
    if i % 2 == 0:
        print i, 'is even.'
    else:
        print i, 'is odd.'

• What does this output?

0 is even.
1 is odd.
2 is even.
3 is odd.
4 is even.
5 is odd.
Nested for loops

```python
for i in range(1, 6):
    for j in range(1, 6):
        prod = i * j
        # use comma to print all on one line
        print prod,
    print
```

must use new index variable for inner loop

- What does this output?

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25
```
For vs While

• For loop is primarily used
  • for iterating over a sequence of values
  • when we know the number of iterations in advance

• While loop is primarily used
  • when we don't know the number of iterations in advance (they could be controlled by user input)
Questions?