UNIVERSITÀ DI PADOVA

Prof. Mauro Conti

T.A. : Hossein Fereidooni & Moreno Ambrosin

< 2014 April >

OUTLINE

What you will become familiar with during the Python programming course are as follows:

- Basic Operators
- Variable Types
- Numbers
- String
- Lists
- Tuples
- Dictionary
- Decision Making
- Loops
- Functions
- Modules
- Files I/O
- Exceptions
- Classes/Objects

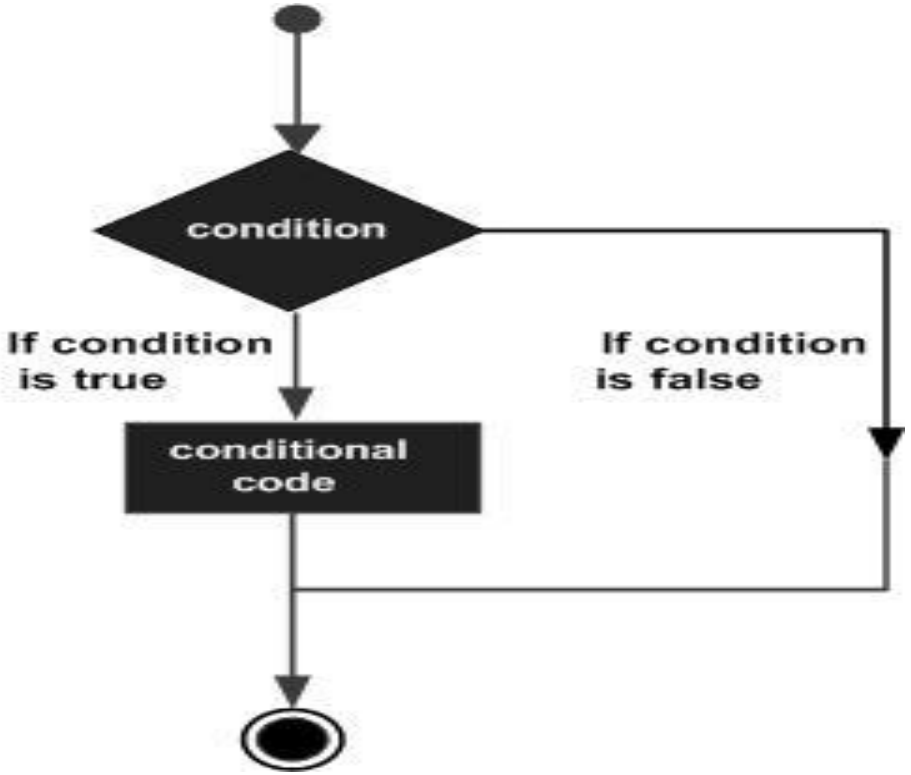
Comparison Operators

Python	*Meaning*
<	*Less than*
<=	*Less than or Equal*
==	*Equal to*
>=	*Greater than or Equal*
>	*Greater than*
!=	*Not equal*

Remember: "=" is used for assignment.

DECISION MAKING

- Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false



Indentation

- Leading whitespace at the beginning of a logical line, which in turn is used to determine the grouping of statements.
  - Increase indent: After an if statement or for statement (after :)
  - Maintain indent: Which lines are affected by the if/for
  - Reduce indent : To back to the level of the if statement or for statement in order to indicate the end of the block
  - Blank lines and comments are ignored. They do not affect indentation

Warning

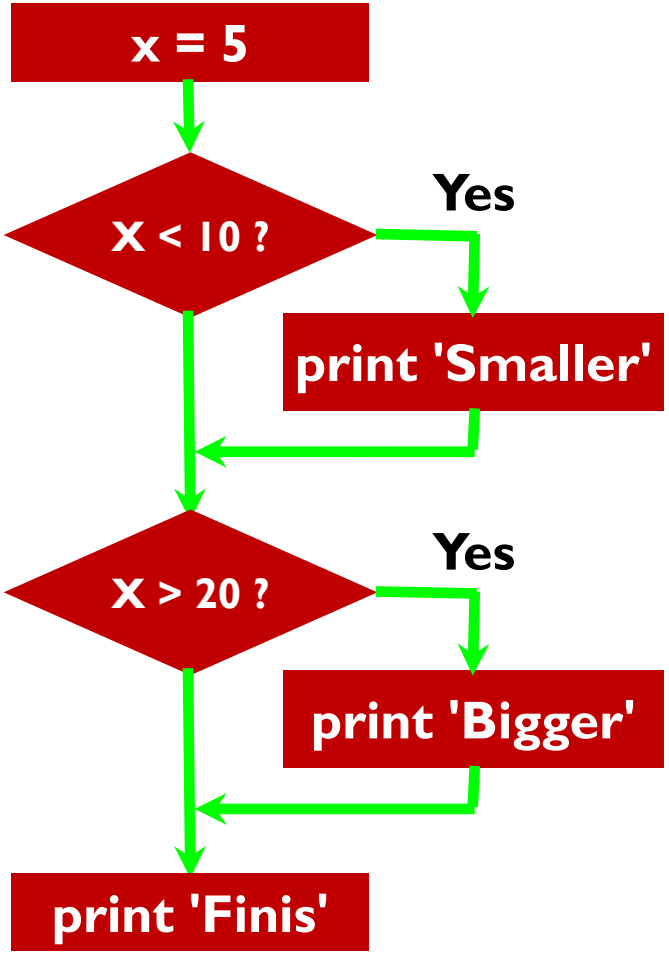
- Python cares a lot about how far line is indented. If you mix tabs and spaces, you may get "indentation errors" even if, everything looks fine

Indentation

```
x = 5
if x > 2 :
    print 'Bigger than 2'
    print 'Still bigger'
print 'Done with 2'

for i in range(5) :
    print i
    if i > 2 :
        print 'Bigger than 2'
    print 'Done with i', i
```

```
x = 5
if x > 2 :
# comments

    print 'Bigger than 2'
        # don't matter
    print 'Still bigger'
# but can confuse you

print 'Done with 2'
    # if you don't line
        # them up
```

DECISION MAKING



Program:

```
x = 5
if x < 10:
    print 'Smaller'

if x > 20:
    print 'Bigger'

print 'Finish'
```

Output:

```
Smaller
Finish
```

Comparison Operators

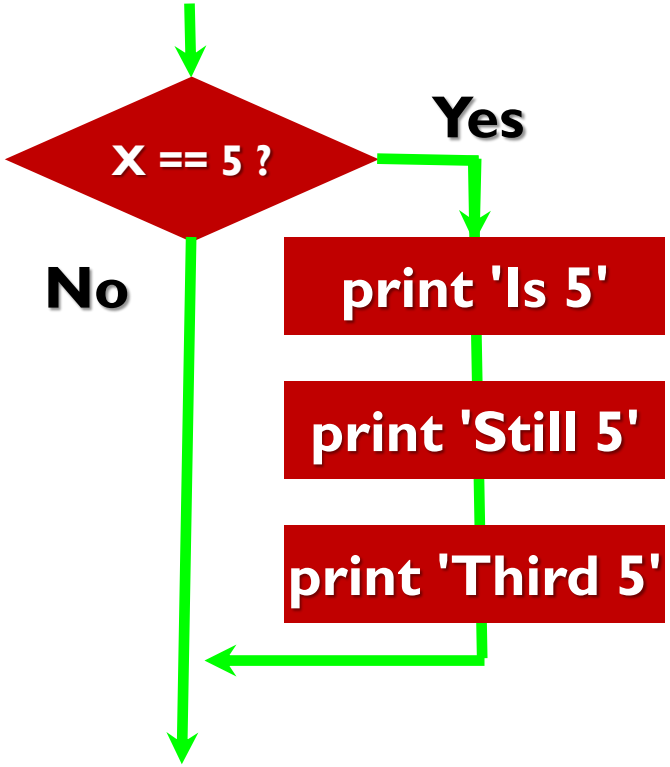
```
x = 5
if x == 5 :
    print 'Equals 5'
if x > 4 :
   print 'Greater than 4'
if  x >= 5 :
    print 'Greater than or Equal 5'
if x < 6 : print 'Less than 6'
if x <= 5 :
    print 'Less than or Equal 5'
if x != 6 :
    print 'Not equal 6'
```

```
Equals 5
Greater than 4
Greater than or Equal 5
Less than 6
Less than or Equal 5
Not equal 6
```

One-Way Decisions

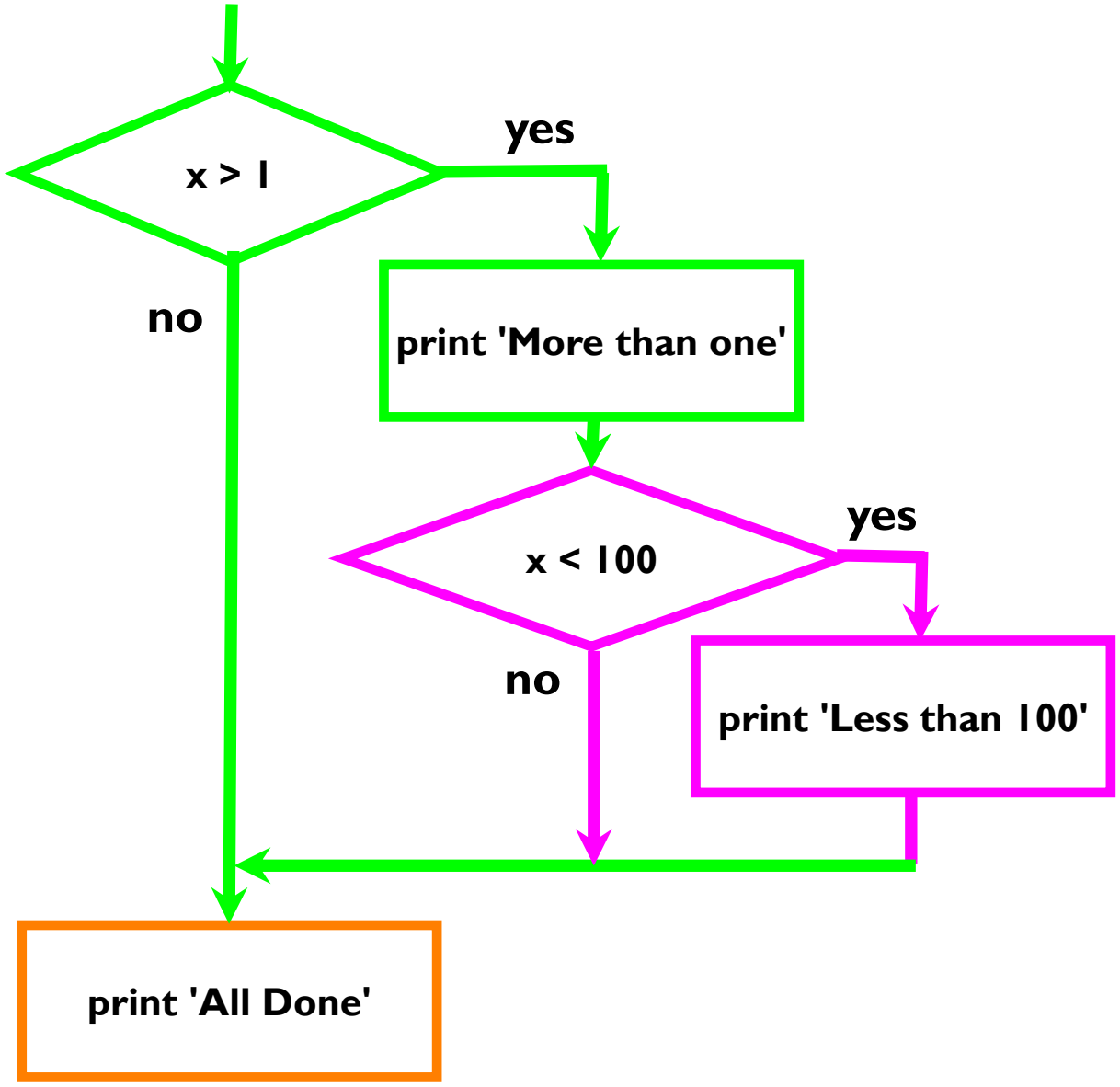
```
x = 5
print 'Before 5'
if  x == 5 :
    print 'Is 5'
    print ' Still 5'
    print 'Third 5'
print 'Afterwards 5'
print 'Before 6'
if x == 6 :
    print 'Is 6'
    print 'Is Still 6'
    print 'Third 6'
print 'Afterwards 6'
```

```
Before 5
Is 5
Still 5
Third 5
Afterwards 5
Before 6
Afterwards 6
```



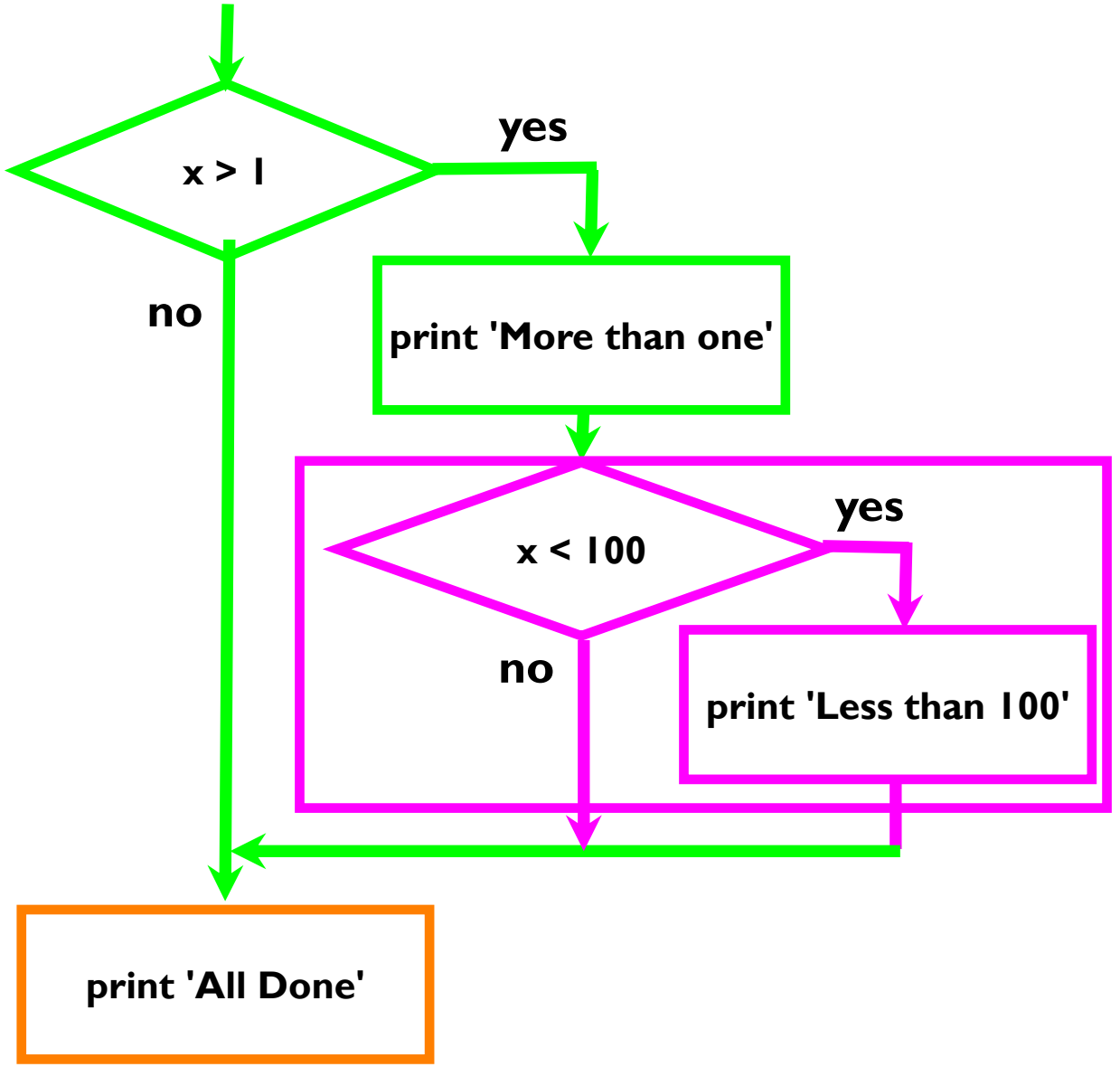
Nested Decisions

```
x = 42

if x > 1 :
    print 'More than one'
    if x < 100 :
        print 'Less than 100'

print 'All done'
```



Nested Decisions

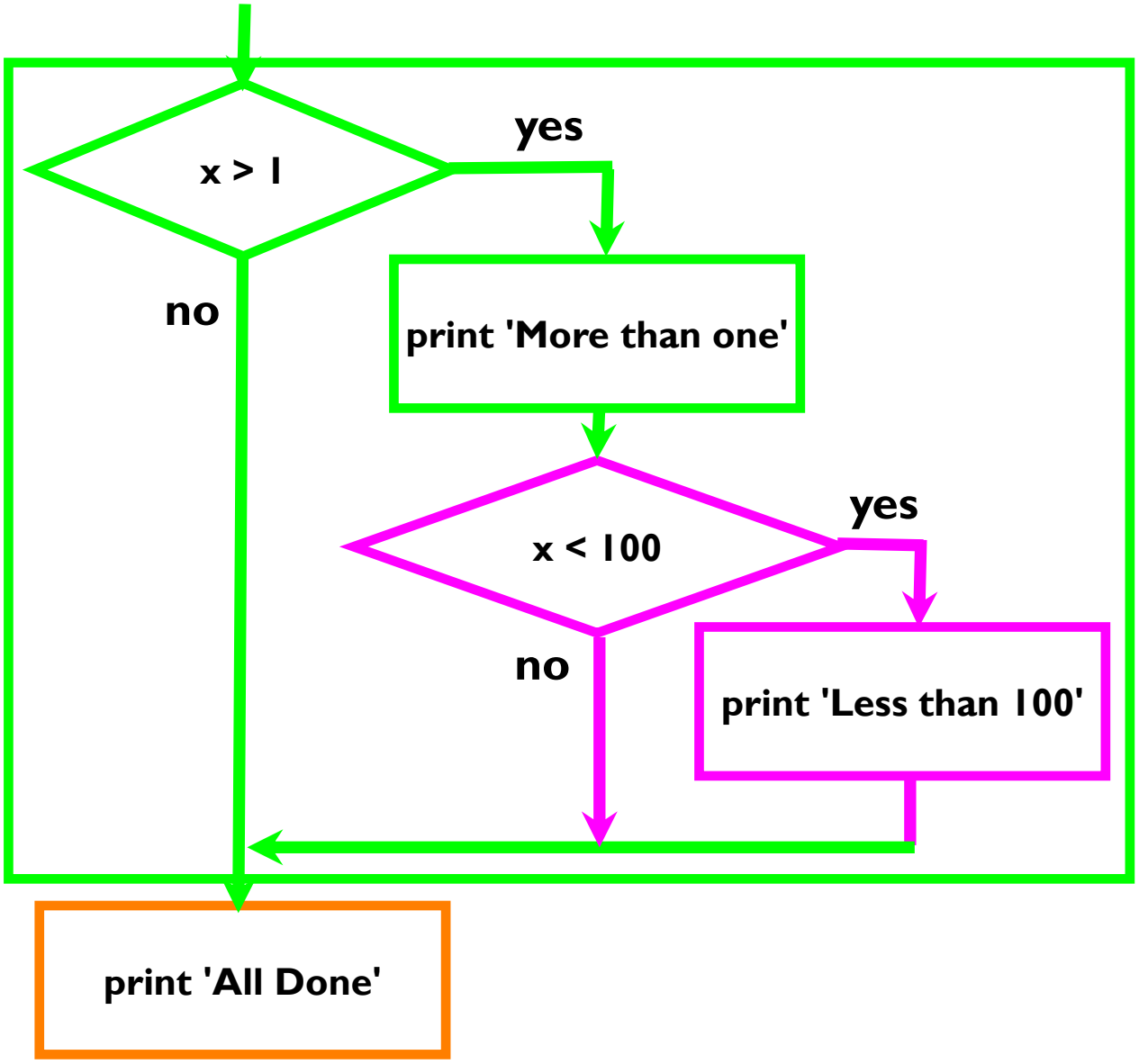
```
x = 42

if x > 1 :
    print 'More than one'
    if x < 100 :
        print 'Less than 100'

print 'All done'
```



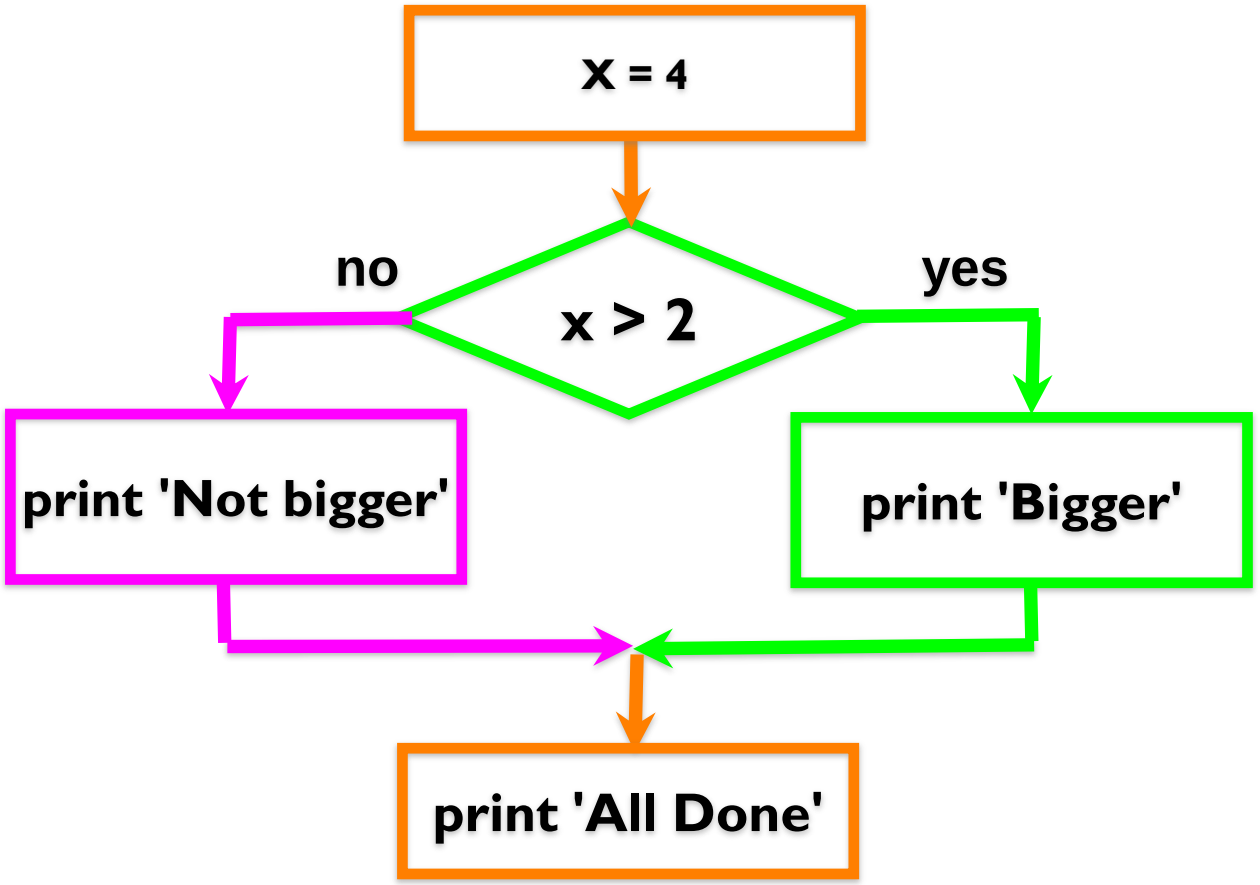
Nested Decisions

$x = 42$

```
if x > 1 :
    print 'More than one'
    if x < 100 :
        print 'Less than 100'

print 'All done'
```



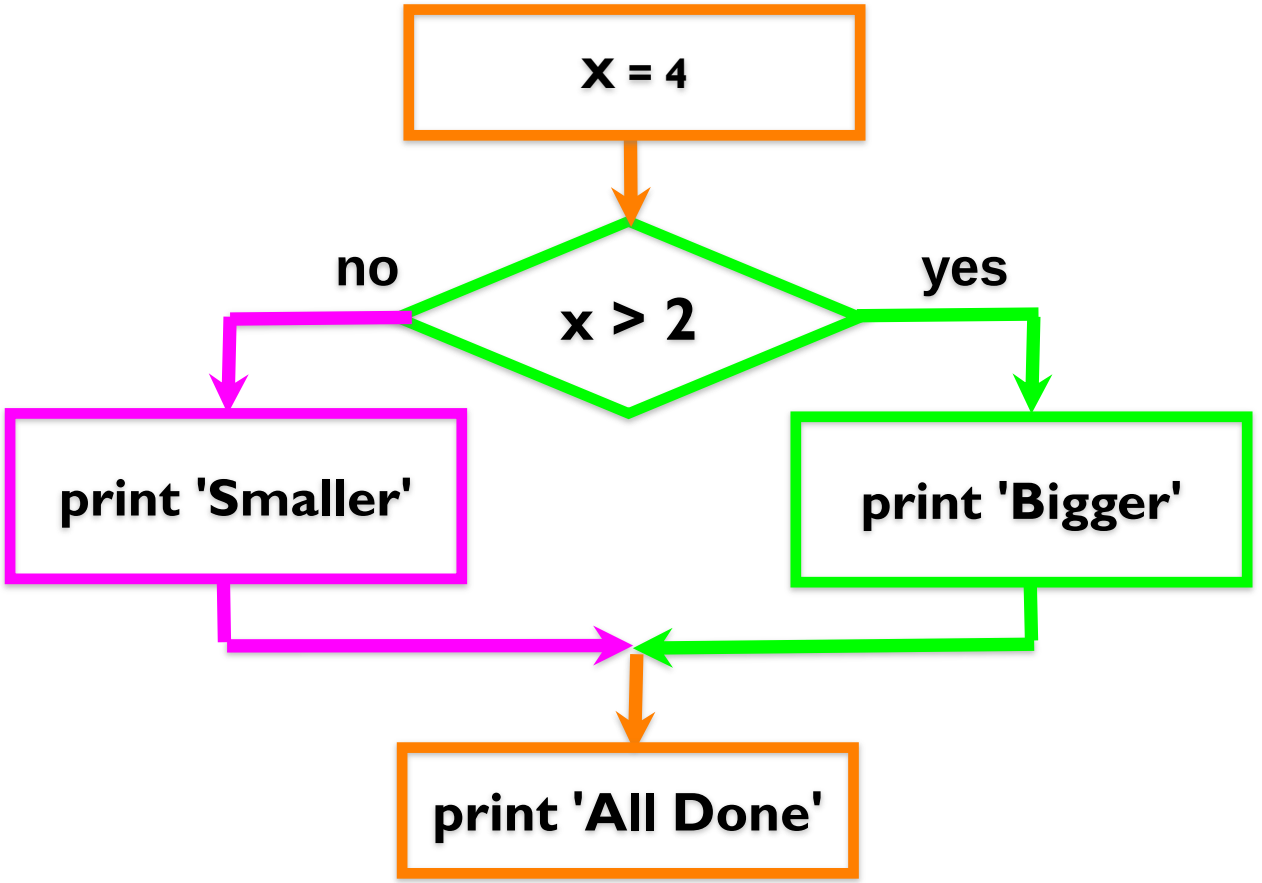
Two Way Decisions



- Sometimes we want to do one thing if a logical expression is true and something else if the expression is false

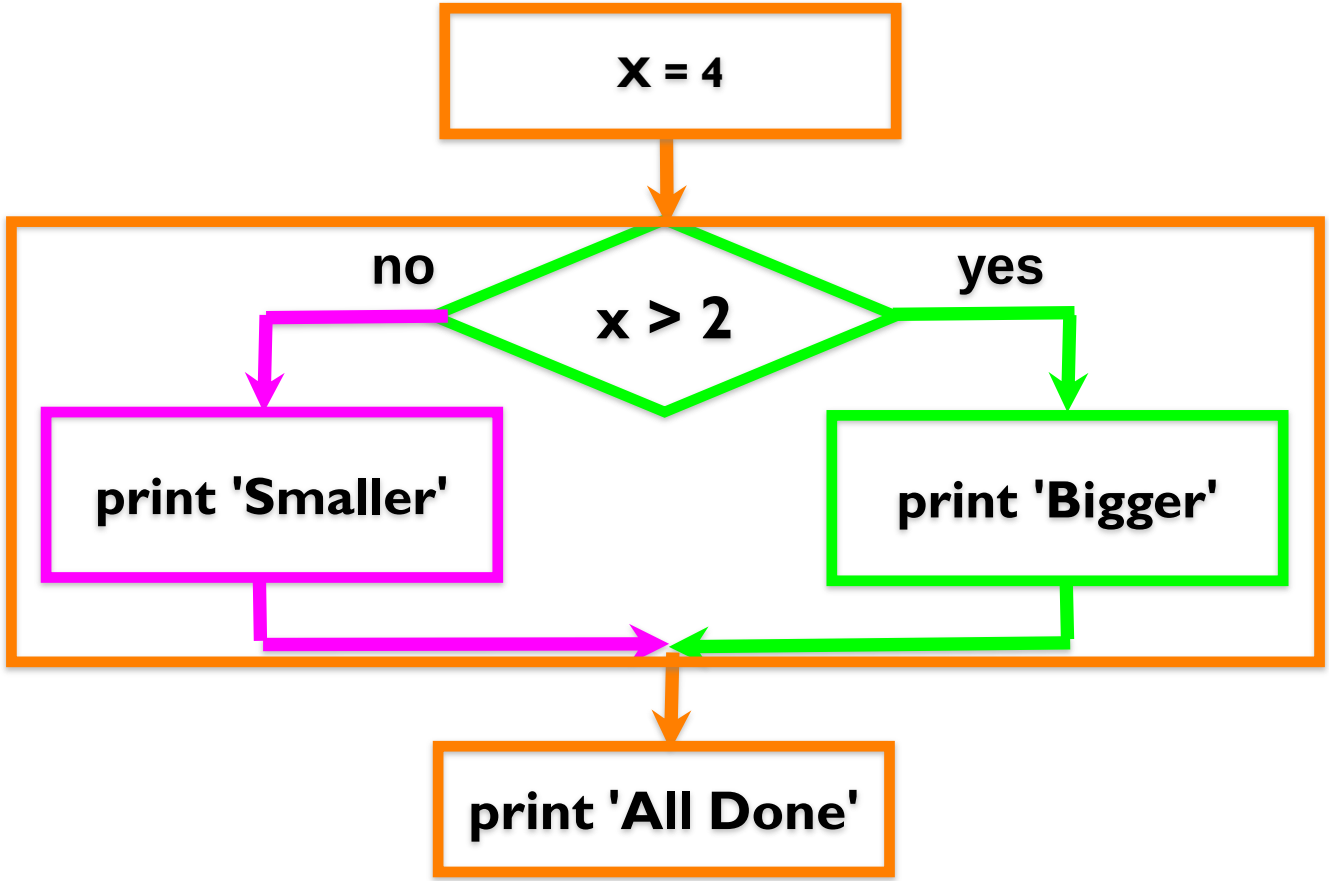
Two Way Decisions

```
x = 4

if x > 2 :
    print 'Bigger'
else :
    print 'Smaller'

print 'All done'
```



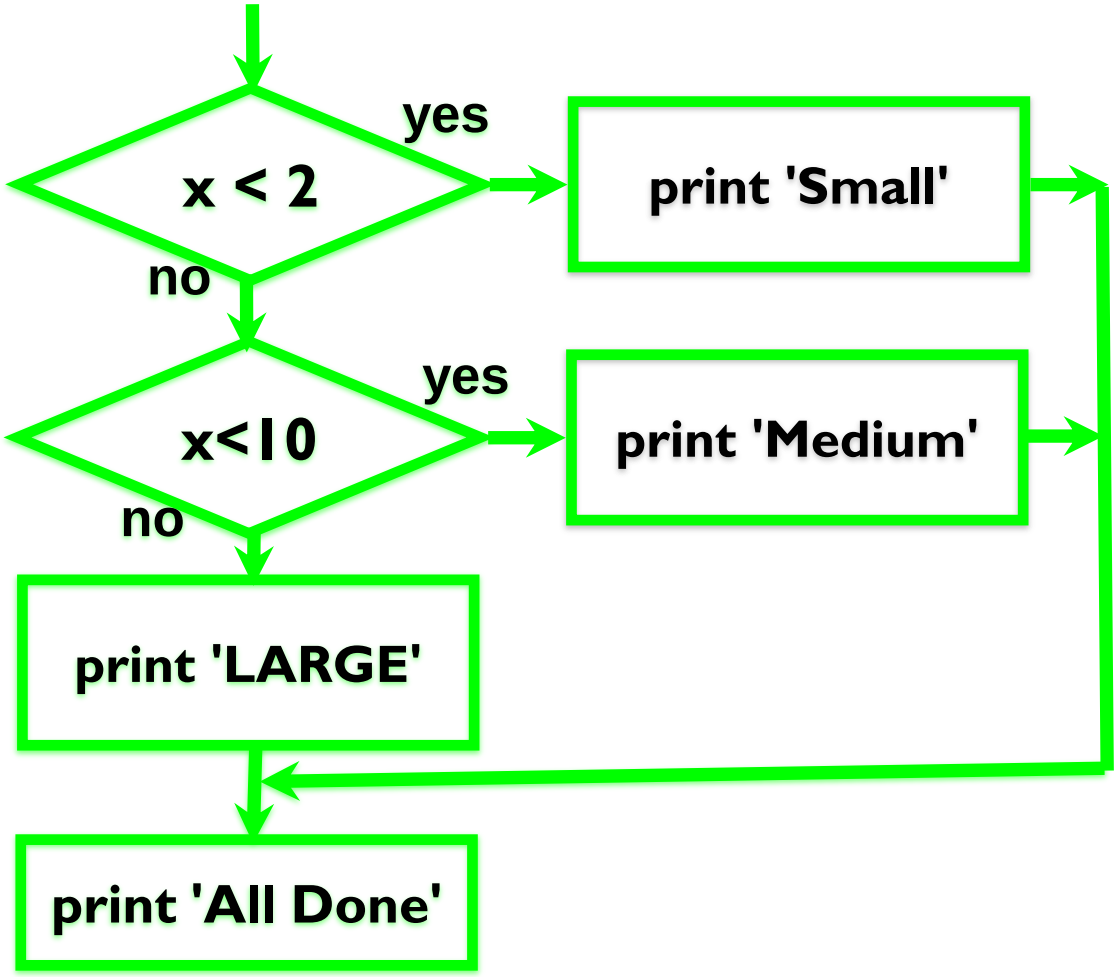
Two-way using else :

```
x = 4

if x > 2 :
    print 'Bigger'
else :
    print 'Smaller'

print 'All done'
```



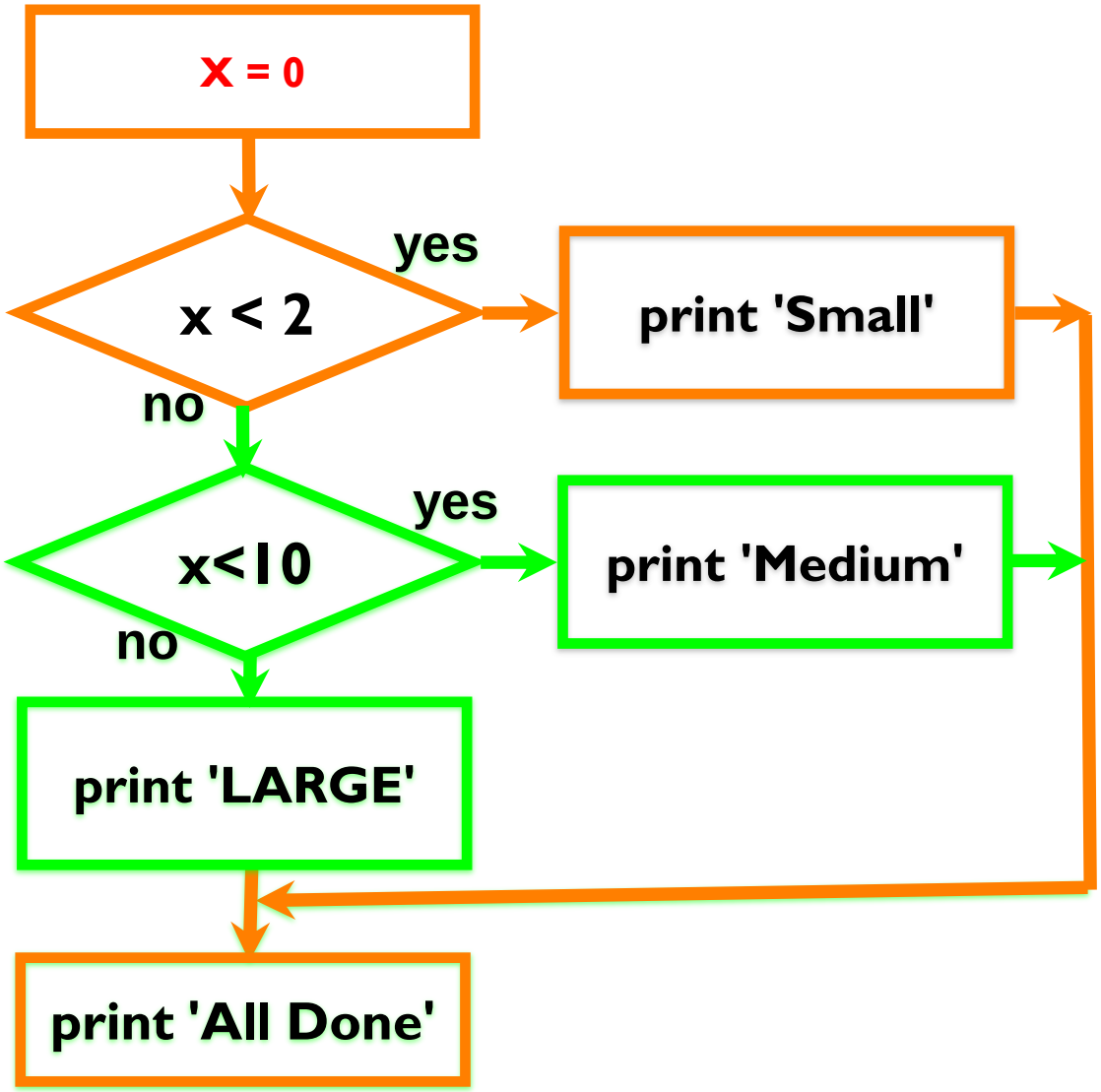
Multi-way

```
if x < 2 :
    print 'Small'
elif x < 10 :
    print 'Medium'
else :
    print 'LARGE'
print 'All done'
```



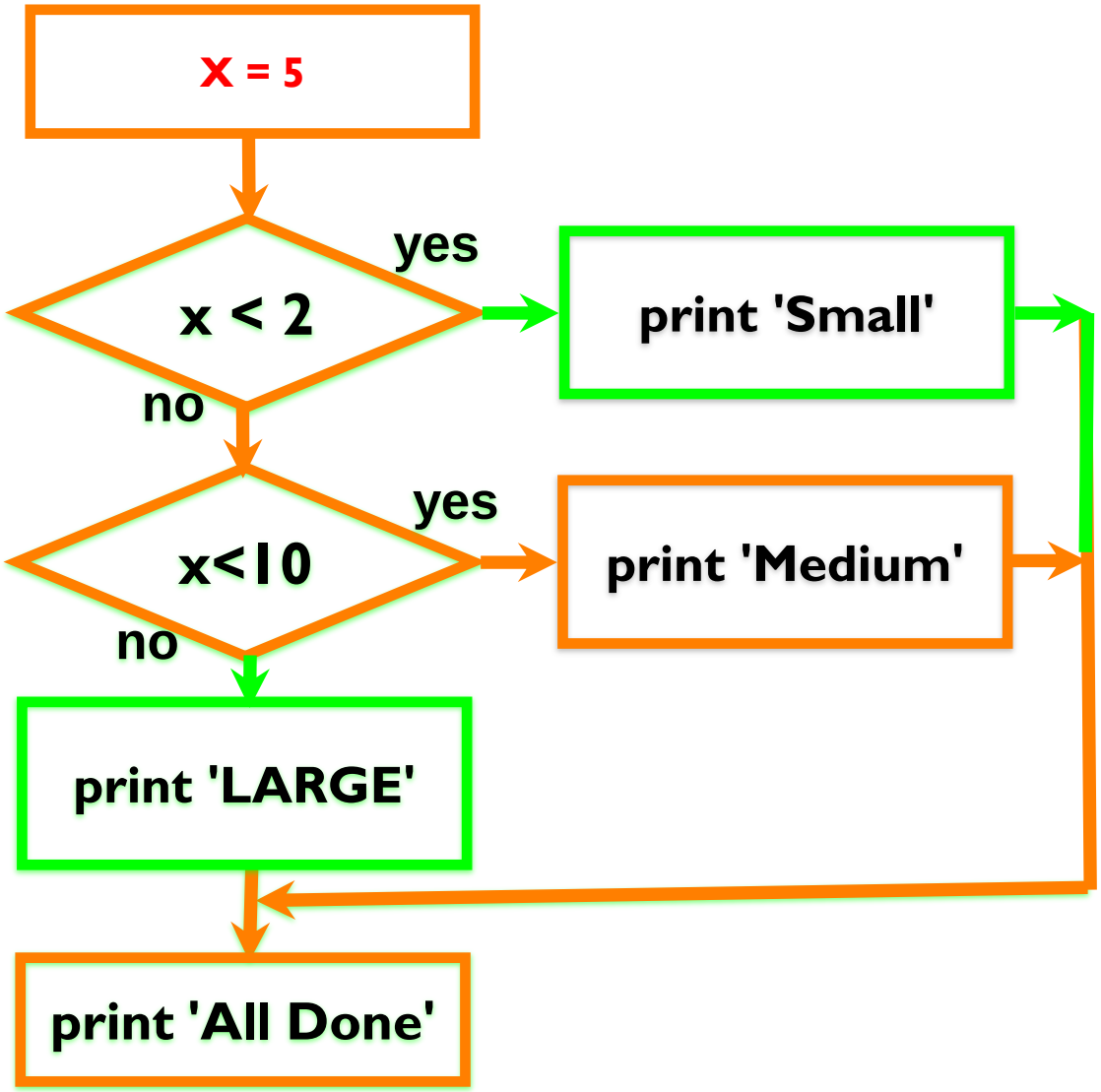
Multi-way

```
x = 0
if x < 2 :
    print 'Small'
elif x < 10 :
    print 'Medium'
else :
    print 'LARGE'
print 'All done'
```



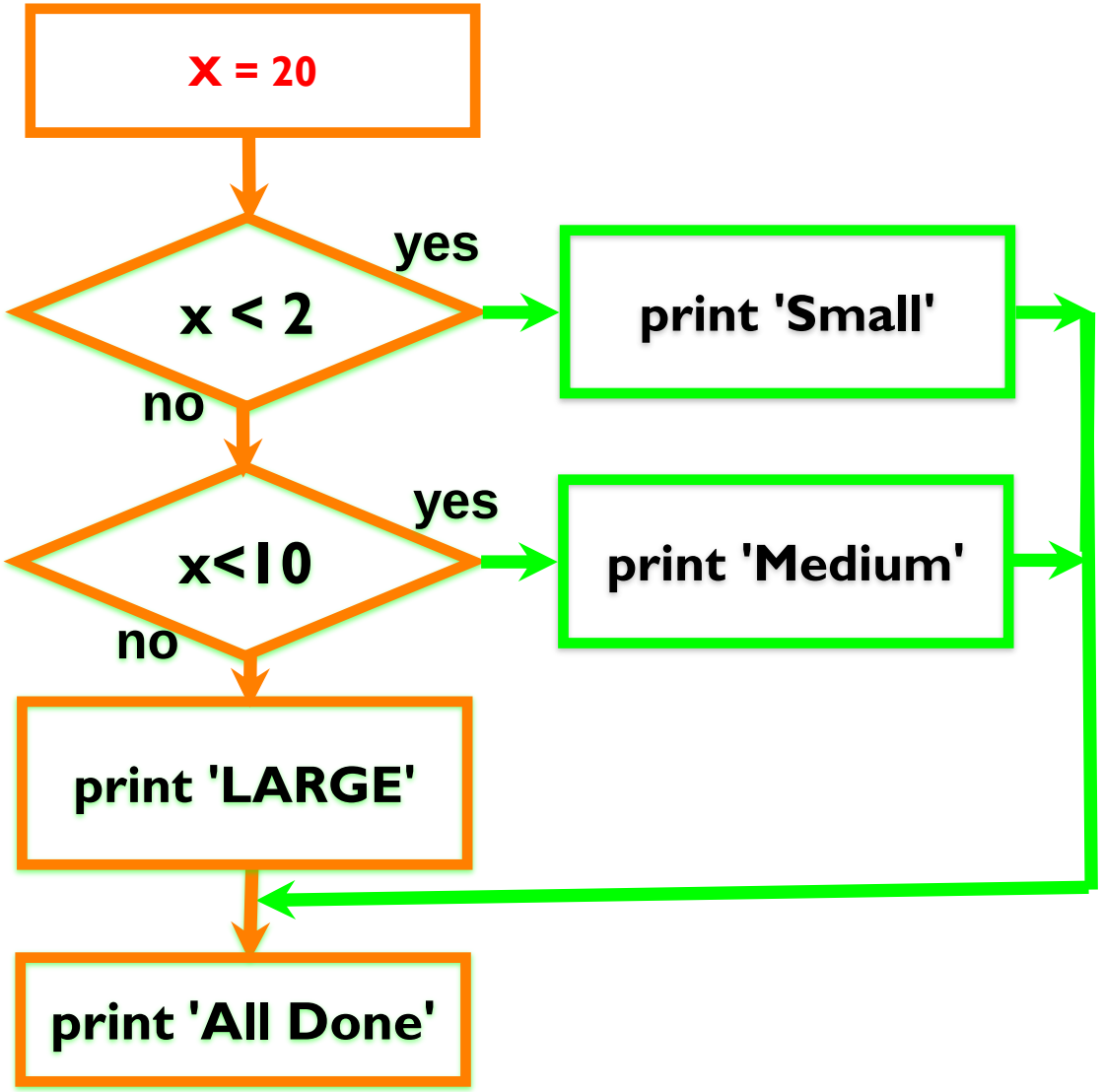
Multi-way

```
x = 5
if x < 2 :
    print 'Small'
elif x < 10 :
    print 'Medium'
else :
    print 'LARGE'
print 'All done'
```



Multi-way

```
x = 20
if x < 2 :
    print 'Small'
elif x < 10 :
    print 'Medium'
else :
    print 'LARGE'
print 'All done'
```



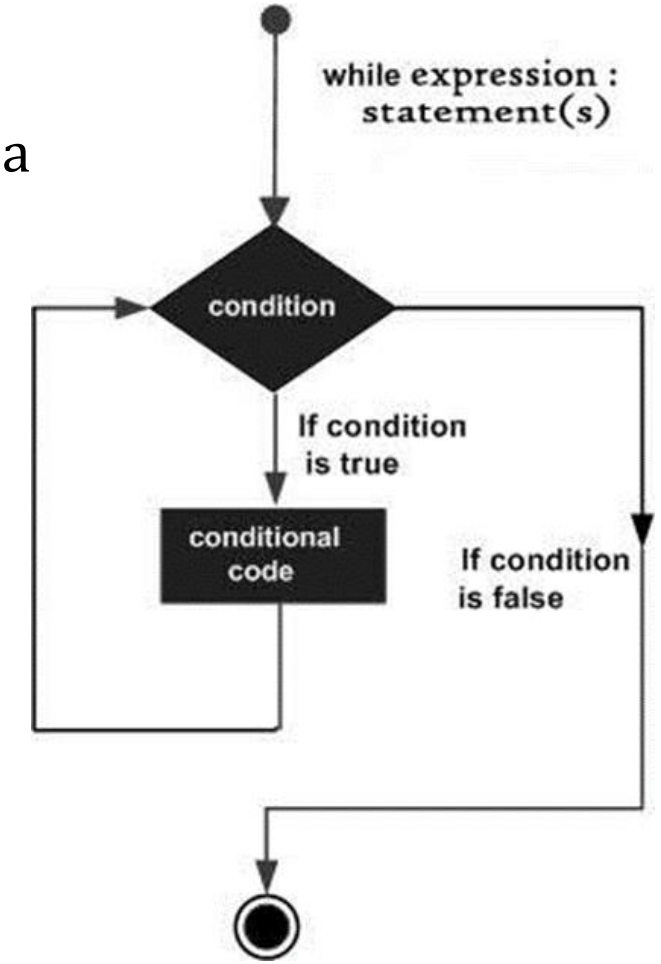
Multi-way

```
# No Else
x = 5
if x < 2 :
    print 'Small'
elif x < 10 :
    print 'Medium'

print 'All done'
```

```
if x < 2 :
    print 'Small'
elif x < 10 :
    print 'Medium'
elif x < 20 :
    print 'Big'
elif x< 40 :
    print 'Large'
elif x < 100:
    print 'Huge'
else :
    print 'WOW'
```

While Loop

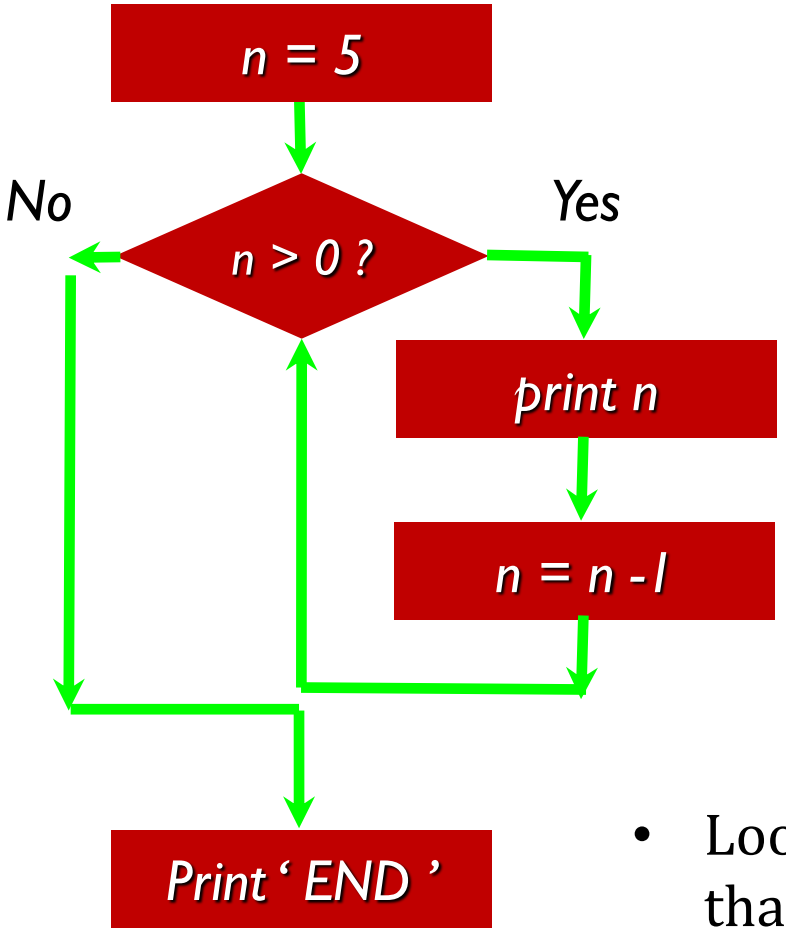
- A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.
- A loop becomes infinite loop if a condition never becomes false
- You would need to use CTRL+C to come out of the program.



While Loop

- Python supports to have an **else** statement associated with a loop statement.
  - If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.
  - If the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has exhausted iterating the list.

While Loop



Program:

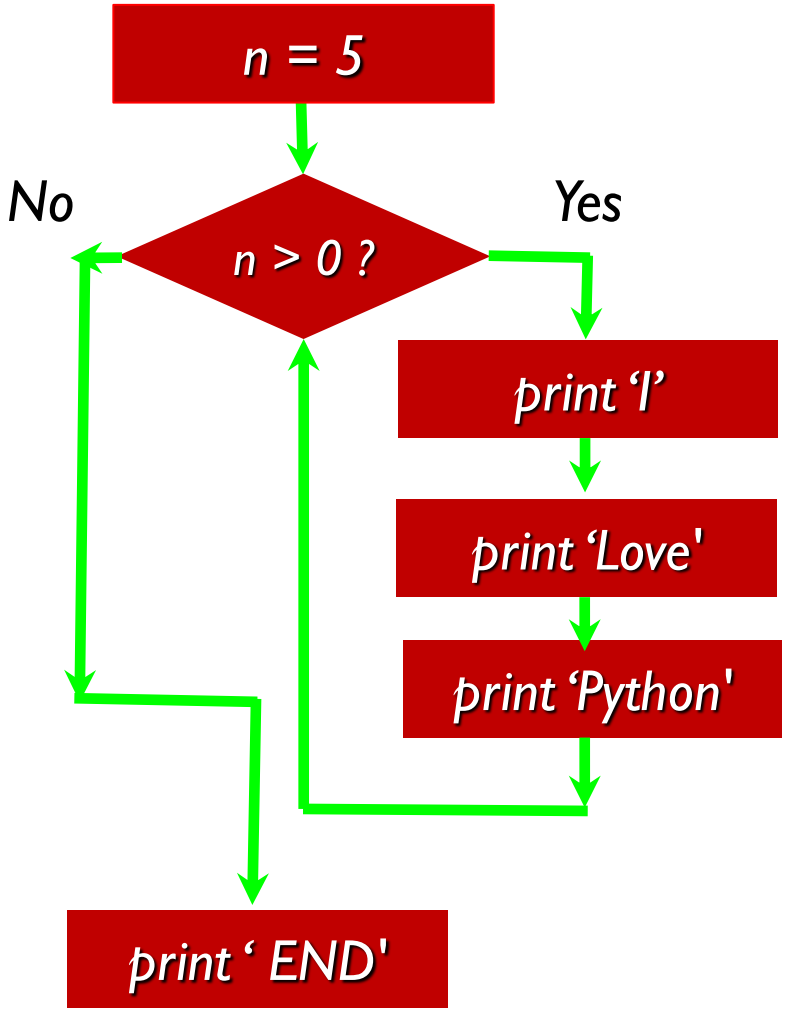
```
n = 5
while n > 0 :
    print n
    n = n – 1
print 'END'
print n
```

Output:

```
5
4
3
2
1
END
0
```

- Loops (repeated steps) have iteration variables that change each time through a loop.

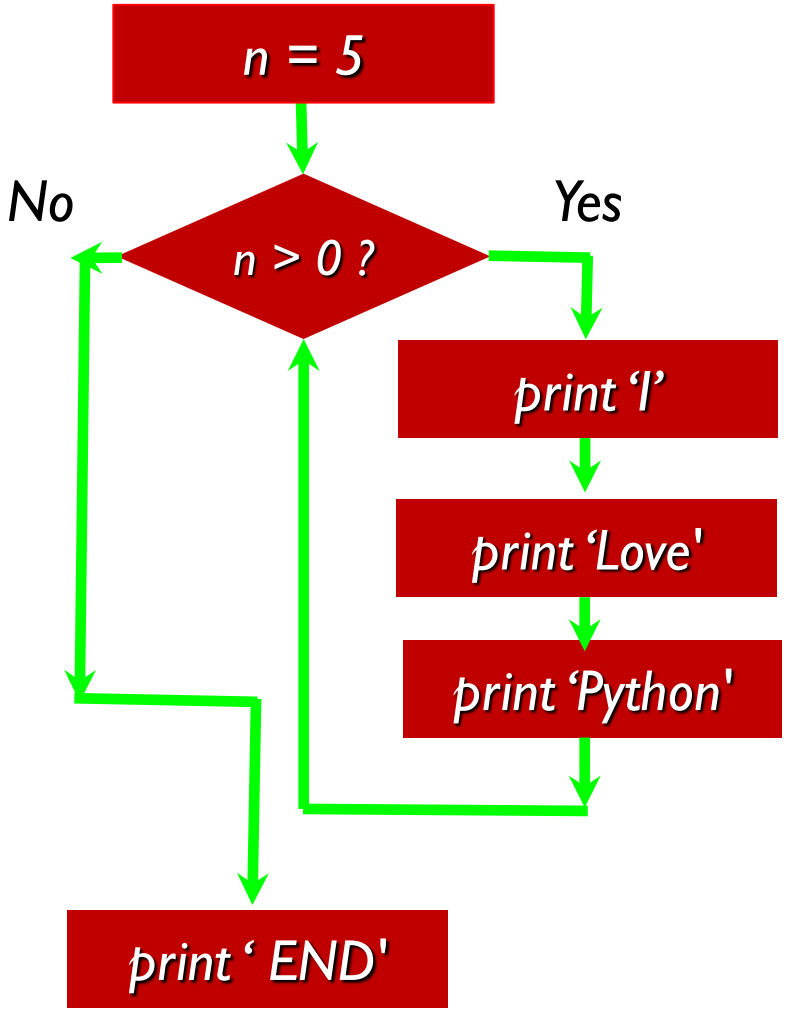
While Loop




```
n = 5
while n > 0 :
    print 'I'
    print 'Love'
    print 'Python'
print 'END'
```

What is wrong with this loop?

Another Loop




```
n = 0
while n > 0 :
    print 'I'
    print 'Love'
    print 'Python'
print ' END'
```

What does this loop do?

Breaking Out of a Loop

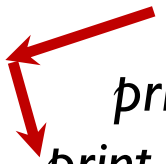
- The break statement ends the current loop and jumps to the statement immediately following the loop

```
while True:
    line = raw_input('> ')
    if line == 'done' :
        break
    print line
print 'Out of loop'
```

```
> hello there
hello there
> finished
finished
> done
Out of loop
```

Breaking Out of a Loop

- The break statement ends the current loop and jumps to the statement immediately following the loop



```
while True:
    line = raw_input('> ')
    if line == 'done' :
        break
    print line
print 'Out of loop'
```

```
> hello there
hello there
> finished
finished
> done
Out of loop
```

Finishing an Iteration with continue

- The continue statement ends the current iteration and jumps to the top of the loop and starts the next iteration

```
while True:
    line = raw_input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print line

print 'Out of loop'
```

```
> hello there
hello there
> # don't print this
>
> done
Out of loop
```

Finishing an Iteration with continue

- The continue statement ends the *current iteration* and jumps to the top of the loop and starts the next iteration

```
while True:
    line = raw_input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print line
print 'Out of loop!'
```

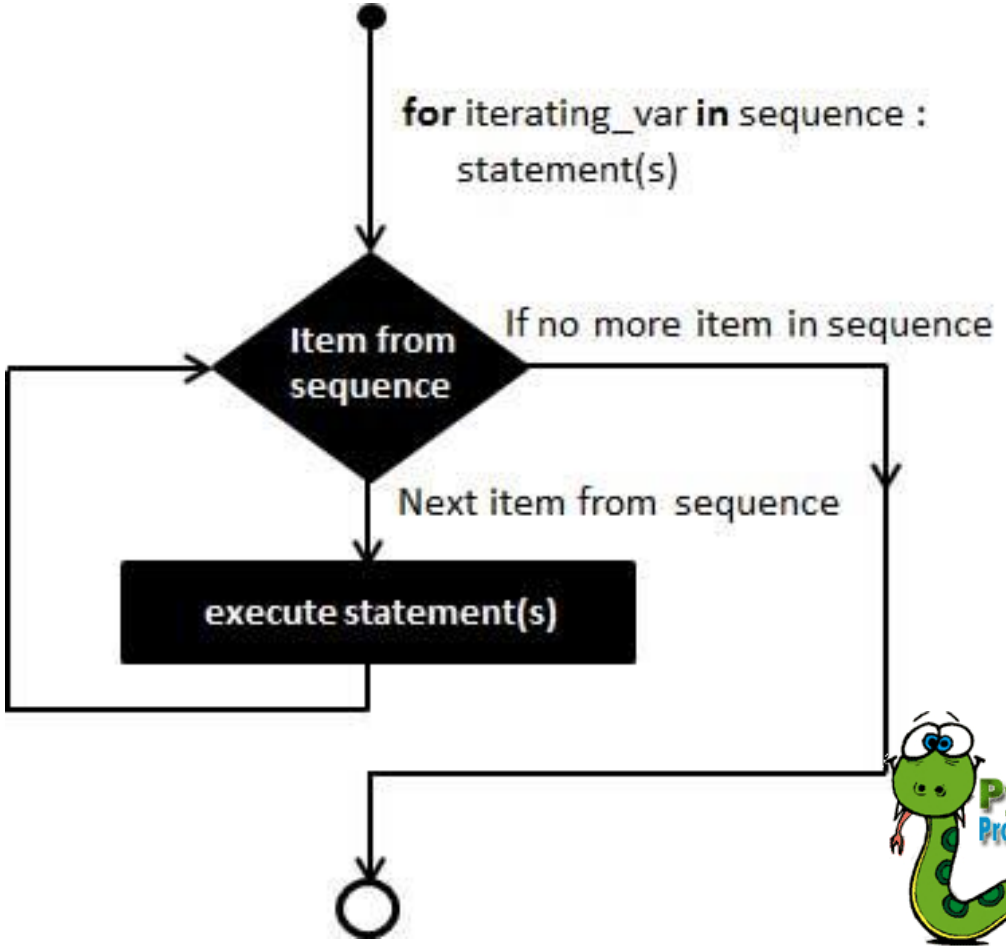
```
> hello there
hello there
> # don't print this
>
> done
Out of loop
```

for loop

- The **for** loop in Python has the ability to iterate over the items of any sequence, such as a list or a string.

Iteration variable

```
for i in [5, 4, 3, 2, 1] :
    print i
print 'END'
```



for loop

- Python supports to have an **else** statement associated with a loop statement.

  - If the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has exhausted iterating the list.

  - If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

A Simple Definite Loop

Five-element sequence

Iteration variable

```
for i in [5, 4, 3, 2, 1] :
    print i
print 'END'
```

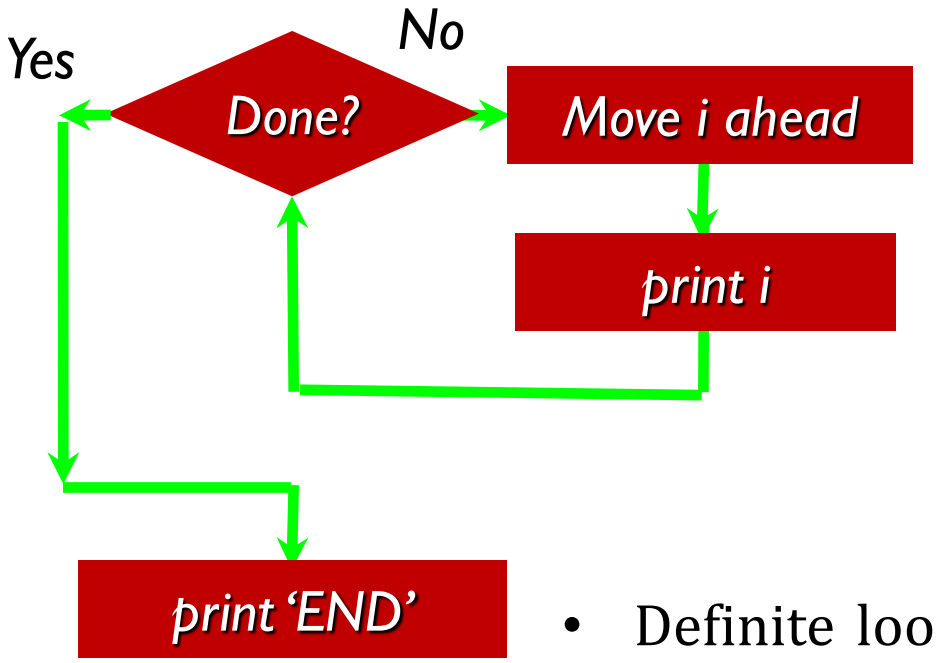
```
5
4
3
2
1
END
```

A Definite Loop with Strings

```
friends = ['Hossein', 'Mauro', 'Moreno']
for friend in friends :
    print 'Happy New Year:', friend
print 'Done!'
```

```
Happy New Year: Hossein
Happy New Year: Mauro
Happy New Year: Moreno
Done!
```

A Simple Definite Loop



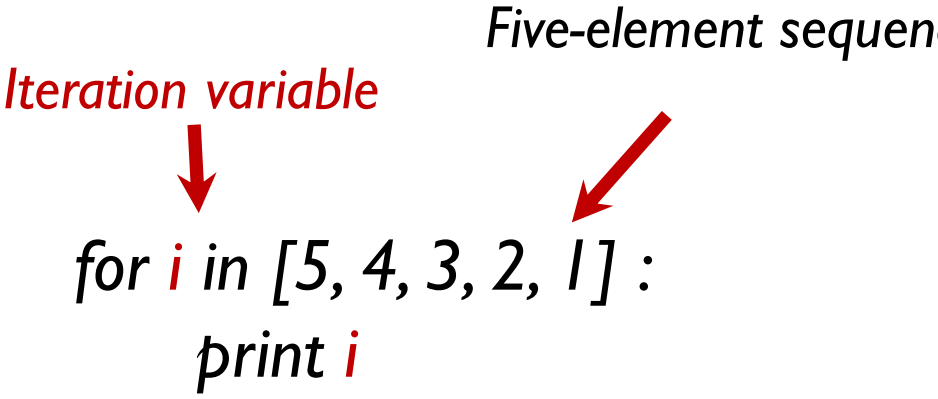

```
for i in [5, 4, 3, 2, 1] :
    print i
print 'END'
```

```
5
4
3
2
1
END
```

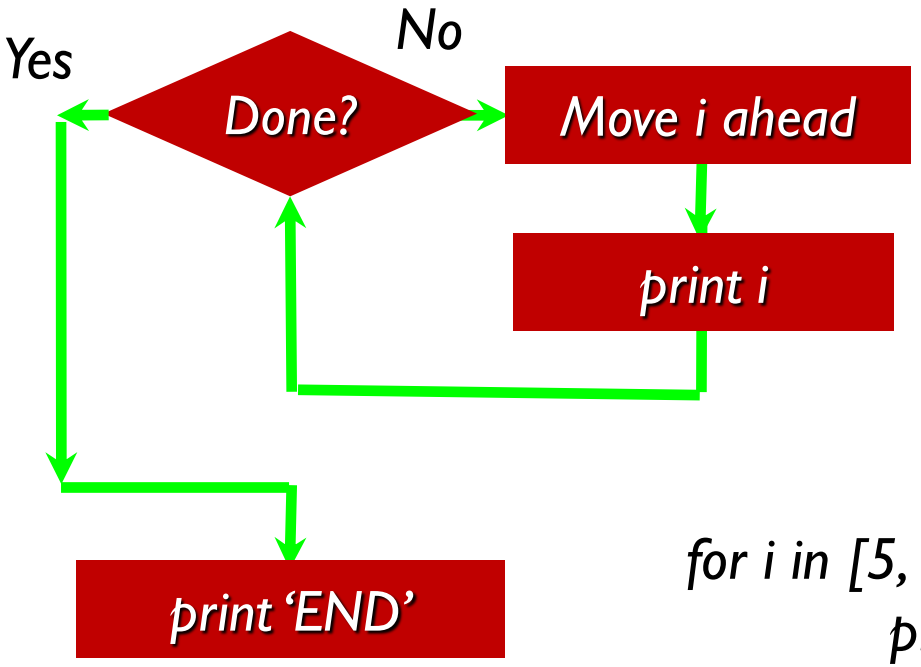
- Definite loops (for loops) have explicit iteration variables that change each time through a loop. These iteration variables move through the sequence or set.

Looking at In...

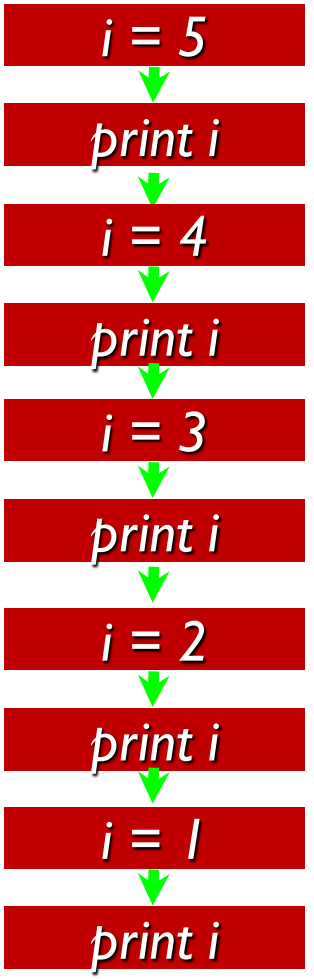
- The block (body) of code is executed once for each value in the sequence
- The iteration variable moves through all of the values in the sequence



Looking at In...




```
for i in [5, 4, 3, 2, 1] :
    print i
```



Looping through a Set

```
print 'Before'
for thing in [9, 41, 12, 3, 74, 15] :
    print thing
print 'After'
```

```
Before
9
41
12
3
74
15
After
```

for thing in data:

Look for something or do something to each entry separately, updating a variable.

Look at the variables.

Counting in a Loop

```
z = 0
print 'Before', z
for thing in [9, 41, 12, 3, 74, 15] :
    z = z + 1
    print z, thing
print 'After', z
```

```
Before 0
1 9
2 41
3 12
4 3
5 74
6 15
After 6
```

- To count how many times we execute a loop we introduce a counter variable that starts at 0 and we add one to it each time through the loop.

Summing in a Loop

```
z = 0
print 'Before', z
for thing in [9, 41, 12, 3, 74, 15] :
    z = z + thing
    print z, thing
print 'After', z
```

```
Before 0
9 9
50 41
62 12
65 3
139 74
154 15
After 154
```

- To add up a value we encounter in a loop, we introduce a sum variable that starts at 0 and we add the value to the sum each time through the loop.

Finding the Average in a Loop

```
count = 0
sum = 0
print 'Before', count, sum
for value in [9, 41, 12, 3, 74, 15] :
    count = count + 1
    sum = sum + value
    print count, sum, value
print 'After', count, sum, sum / count
```

```
Before 0 0
1 9 9
2 50 41
3 62 12
4 65 3
5 139 74
6 154 15
After 6 154 25
```

- An average just combines the counting and sum patterns and divides when the loop is done.

Filtering in a Loop

```
print 'Before'
for value in [9, 41, 12, 3, 74, 15] :
    if value > 20:
        print 'Large number', value
print 'After'
```

```
Before
Large number 41
Large number 74
After
```

- We use an if statement in the loop to catch / filter the values we are looking for.

Search Using a Boolean Variable

```
found = False
print 'Before', found
for value in [9, 41, 12, 3, 74, 15] :
    if value == 3 :
        found = True
    print found, value
print 'After', found
```

```
Before False
False 9
False 41
False 12
True 3
True 74
True 15
After True
```

- If we just want to search and know if a value was found - we use a variable that starts at False and is set to True as soon as we find what we are looking for.

Finding the smallest value

```
smallest = None
print 'Before'
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print smallest, value
print 'After', smallest
```

```
Before
9 9
9 41
9 12
3 3
3 74
3 15
After 3
```

- We still have a variable that is the smallest so far. The first time through the loop smallest is None so we take the first value to be the smallest.

The "is" and "is not" Operators

```
smallest = None
print 'Before'
for value in [3, 41, 12, 9, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print smallest, value
print 'After', smallest
```

- Python has an "is" operator that can be used in logical expressions
  - Similar to ==, but stronger than
  - 'is not' also is a logical operator

REFERENCES

1. http://www.tutorialspoint.com/index.htm
2. http://docs.python.org/lib/string-methods.html
3. http://www.pythonlearn.com/

Contact

- **Website:**
  http://www.math.unipd.it/~hossein/fereidooni.htm
- **E-mail: hossein@math.unipd.it**
- **Skype: fereidooni1983**