Repetition with while

while = ifs that repeat

if, while & for

```
count = 0
while count < 10:
    print(count)
    count += 1
```

while output:

& for output

```
0
1
2
3
4
5
6
7
8
9
```

```
count = 0
if count < 10:
    print(count)
    count += 1
```

if output:

```
0
```

```
for count in range(10):
    print(count)
```

Why Loops? Who Needs Repetition?

Q: Which ones can we use a for loop for?

Repetitive, frequent tasks:

- Shoveling snow or raking leaves
- Putting a case of beverages in the fridge one at a time
- Waiting for an internet release time (tickets, books, iPhones)
- Checking texts or status updates
- code.org: moving a zombie painter
- Folding socks

A: Only the ones where we know before the loop executes how many times they will need

Review: Definite Loops

for

`for` Loop Syntax & Semantics

- When we know how many times a loop will execute
  - Repeat N times

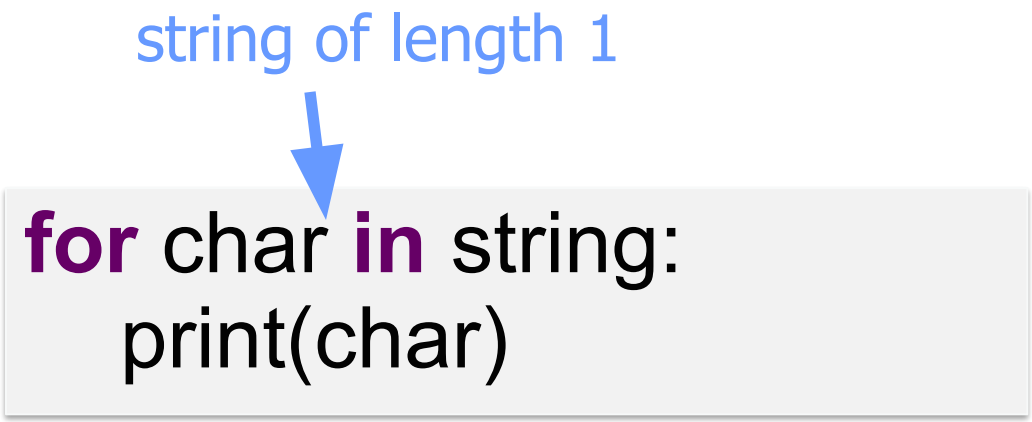
Number of times to repeat (N)

```
for x in range(10):
   statement_1
statement_2
   …
   statement_n
```

Body of `for` loop
- Gets repeated
- Note indentation

Iterating Through a String

- Use a **for** loop to iterate through *characters* in a string



  - Read as “for each character in the string”

Python shell

New: Indefinite Loops

while

Indefinite While Loops

- For loops are definite: we know in advance exactly how many times they should execute
- What if we don't know how many times a loop should execute?
- Real-life examples:
  - Folding socks
  - Checking status (Are we there yet? Any new texts?)
- Programming:
  - Requesting user input
  - Reading in data from a file

`while` vs `for`

- Any for loop can be written as a while loop
  - i.e., any definite loop can be written as a while **OR** a for
- What are the differences between these loops?
- What are the advantages/disadvantages of each?

```
i = 0
while i < 10 :
    print("i equals", i)
    i += 1
print("Done", i)
```

```
for i in range(10):
    print("i equals", i)

print("Done", i+1)
```

whilevsfor.py

What will this loop do?

```
count = 1
while count > 0:
    print(count)
    count += 1
```

loop.py

Infinite Loop

- Condition will never be False so keeps executing!

```
count = 1
while count > 0:
   print(count)
   count += 1
```

- To stop an executing program use
  - Control-C on command line
  - Stop button in pycharm

Infinite Loop Discussion

- Is there ever a time that an infinite loop is wanted?
  - Yes! For example in web servers:

```
while True:
    listenForRequest()
    handleRequest()
```

- Can a computer automatically detect infinite loops?
  - No that is an **undecidable** problem
  - Best to **prevent** infinite loops (more later)
    - Benefit of Python's **for** loops: definite loops

Practicing `while` Loops

- Write the Python code to print the following using while loops:

A)

```
1
2
3
4
5
```

B)

```
2
5
8
1
1
```

C)

```
***
*
***
*
***
*
```

Control Flow

What order are the statements executed?

7, 1, 2, 4, 5, 7, 8

```
1   def maximum(x, y):
2     if x > y:
3       return x
4     else:
5       return y
6
7   print(maximum(3, 30))
8   print(x)
```

Functions, conditions, & loops change program control flow from sequential to out of order

Self-check

- How can we make something repeat when some condition is true?
- True or False: Every for loop can be converted into a while loop
- True or False: A while loop is more powerful than a for loop

Design Pattern: Sentinel Loop

- Sentinel: when to stop
  - “guard” to the loop

```
value = initialize
while value != sentinel :
    process value
    value = updated value
```

While

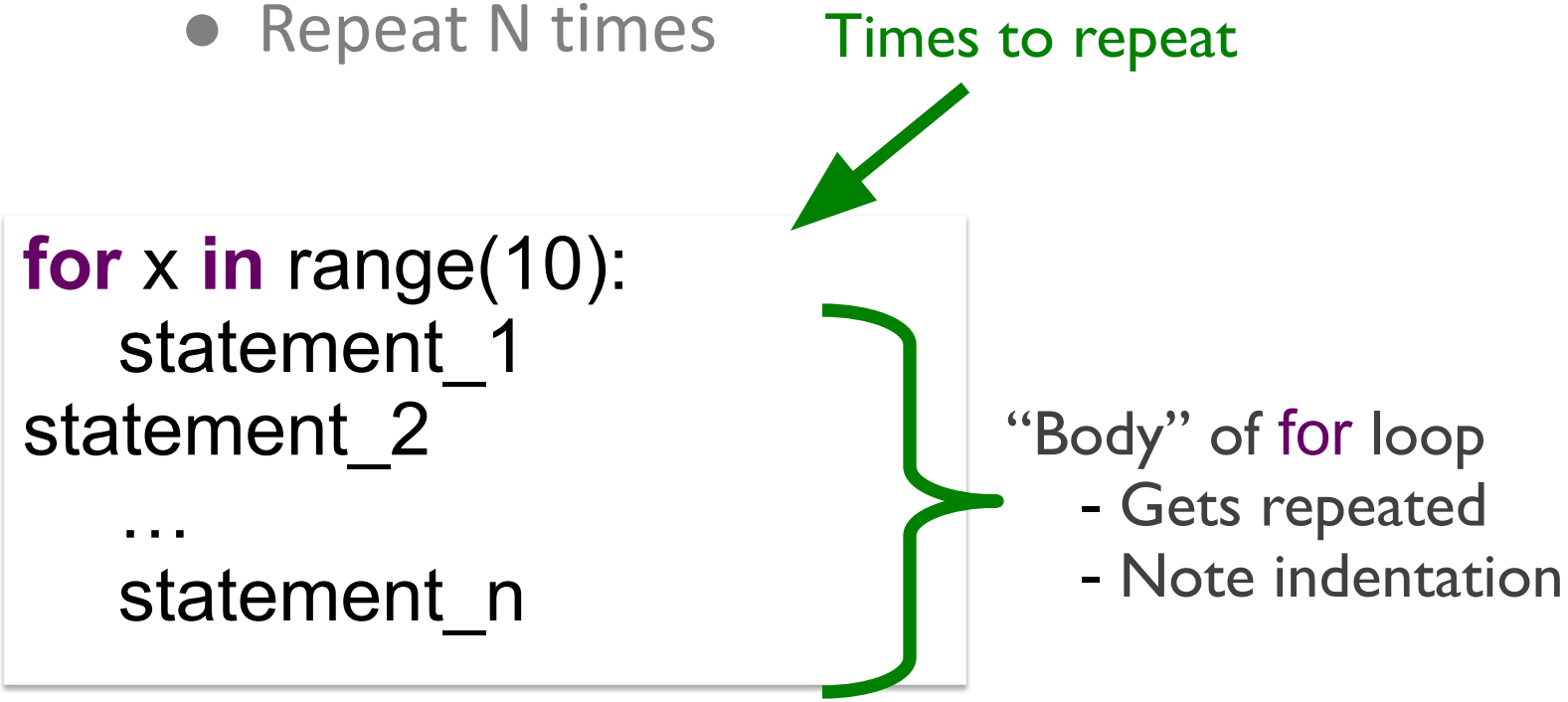
More details

The 7 Programming Basics

	Concept	Example from Math
1.	Variables	$x = 5$ *hellothere* = "howdy"
2.	Math & Logic	$5 * 7 + a - 3 / b \% 4$ a is 5 AND $x < 7$ OR *degree* ≥ 98
3.	Input/Output (IO)	print "Hello World"
4.	Conditionals	**if** $(x == f(x))$ **then** print "x is 0 or 1" **else** print "x is not 0 or 1"
5.	Loops	**foreach** x **in** $(array)$ print x
6.	Functions	$f(x) = x^2$
7.	Lists	*array* = 1:5 *array* = 1, 4, 7, 8, a, b, c, d

for Loop Definition

- When we know how many times a loop will execute
  - Repeat N times



Indefinite While Loops

- For loops are definite: we know in advance exactly how many times they should execute
- What if we don't know how many times a loop should execute?
- Real-life examples:
  - Folding socks
  - Checking status (Are we there yet? Any new texts?)
- Programming:
  - Requesting user input
  - Reading in data from a file

Indefinite While Loops

keyword

```
while condition :
    statement_1
    statement_2
    …
    statement_n
```

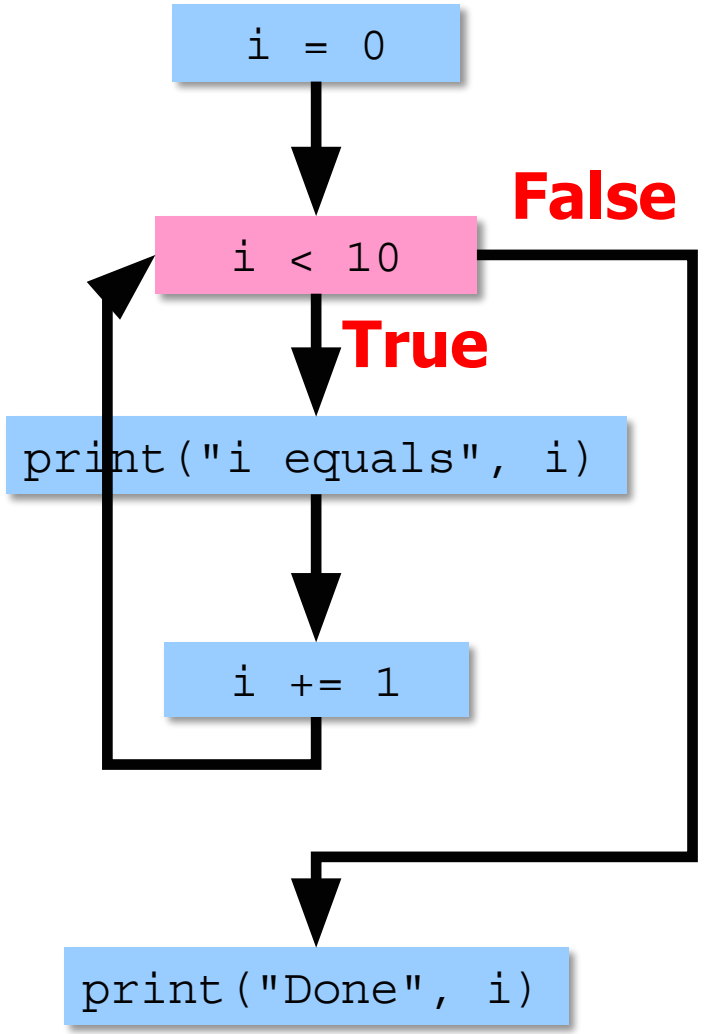
loop stops when condition is False

body of while loop

- Like a looped if: execute loop body only while condition is true

While Loop Example

```
i = 0
while i < 10 :
    print("i equals", i)
    i += 1
print("Done", i)
```



- What will be printed?
- How many times is the loop body executed?
- How many times is the loop condition evaluated?

while.py

`while` vs `for`

- Any for loop can be written as a while loop
  - i.e., any definite loop can be written as a while **OR** a for
- What are the differences between these loops?
- What are the advantages/disadvantages of each?

```
i = 0
while i < 10 :
    print("i equals", i)
    i += 1
print("Done", i)
```

```
for i in range(10):
    print("i equals", i)

print("Done", i+1)
```

whilevsfor.py

Shorthand Arithmetic

- Make loop increment/decrement easier
- The following are equivalent:

```
i = i + 1
i += 1
```

```
i = i - 1
i -= 1
```

```
i = i * 10
i *= 10
```

```
i = i / 10
i /= 10
```

```
i = i ** 2
i **= 2
```

```
i = i % 10
i %= 10
```

What will this loop do?

```
count = 1
while count > 0:
    print(count)
    count += 1
```

loop.py

Infinite Loop

- Condition will never be False so keeps executing!

```
count = 1
while count > 0:
   print(count)
   count += 1
```

- To stop an executing program use
  - Control-C

Infinite Loop Discussion

- Is there ever a time that an infinite loop is wanted?
  - Yes! For example in web servers:

```
while True:
    listenForRequest()
    handleRequest()
```

- Can a computer automatically detect infinite loops?
  - No that is an **undecidable** problem
  - Best to **prevent** infinite loops (more later)
    - Benefit of Python's **for** loops: definite loops

Try it!

Practicing `while` Loops

- Write the Python code to print the following using while loops:

A)

```
1
2
3
4
5
```

B)

```
2
5
8
1
1
```

C)

```
***
*
***
*
***
*
```

Control Flow Review

A matter of ***scope***…

function formal parameters create ***local*** variables:

```
def maximum(x, y):
    if x > y:
        return x
    else:
        return y

print(maximum(3, 30))
print(x)
```

scope of x & y

NameErr or

scope.py

Control Flow

What order are the statements executed?

7, 1, 2, 4, 5, 7, 8

```
1   def maximum(x, y):
2     if x > y:
3       return x
4     else:
5       return y
6
7   print(maximum(3, 30))
8   print(x)
```

Functions, conditions, & loops change program control flow from sequential to out of order

scope.py