



STEAM CLOWN™ PRODUCTIONS

PYTHON-LOOPS & ITERATIONS

OBJECTIVE, OVERVIEW & INTRODUCTION

- Loops (repeated steps) have iteration variables that change each time through a loop. Often these iteration variables go through a sequence of numbers
- Computers are often used to automate repetitive tasks
- Repeating identical or similar tasks without making errors is something that computers do well and people do poorly
- Because iteration is so common, Python provides several language features to make it easier

WHAT YOU WILL KNOW...

- Prior Knowledge & Certifications
 - You should have a basic understanding of Python language structures
- What You Will Know & Be Able To Do
 - You will be able to implement loop code that can be use to run code over and over again till the loop parameters are met



STEAM CLOWN™ PRODUCTIONS



See Appendix A,B,C, for Licensing & Attribution information

These slides are an adaption, to better target my SVCTE High School Mechatronics Engineering class, primarily from Dr. Charles R. Severance's Python for Everybody class <https://www.py4e.com/> ... but from other sources as well. See Appendix A

CC BY-NC-SA 4.0

<https://creativecommons.org/licenses/by-nc-sa/4.0/>
<https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode>

GNU Public License

Any included Programming Code Is licensed under the [GNU General Public License v3.0](#)

EURL (European Union Public Licence) Code and Content is also licensed under the [EURL 1.2 or later](#)



HOW WILL YOU BE MEASURED

- Individual Students will submit working code
- Students, as individuals or teams will present orally how they solved the coding challenge, and depth of understanding will be graded
- Success will be determined by how well your code runs as checked by the instructor after you have turned in your **Lastname-Firstname-ProgramName.py** text files

NEW WORDS...

- Iteration
- Loop
- While
- For

WHERE CAN I RUN MY PYTHON CODE?

- The main way we will implement Python code will be by running it on a Raspberry Pi, using the Linux command terminal shell, or the Idle3 Python interpreter
- If you don't have a Raspberry Pi, or if you don't have Python installed, there are a few Python interpreters online. This lets you try code with out having to install Python on your own PC or physically have a Raspberry Pi or other hardware. Here are a few. If you find a better one, please let me know
 - [Python 3 On-Line Python Interpreter - Tutorials Point](#)
 - [Python 2.7 On-Line Python Interpreter - Tutorials Point](#)
 - [Python Interpreter - Online GDB](#)
 - [Python Shell - Python.org](#)



STEAM CLOWN™
& **Squeaky Hinge**
PRODUCTIONS

© Copyright 2018 STEAM Clown™

I GOT THIS... CAN I JUMP AHEAD?

- Jump Ahead and do the labs, save them and turn them in (show me and turn in later)
- Still need something to do? Try writing your own program or try this Extra Credit <linktolab> (show me and turn in later)



STEAM CLOWN™
& **Squeaky Hinge**
PRODUCTIONS

© Copyright 2018 STEAM Clown™

RESOURCES & MATERIALS NEEDED

- PY4E Chapter 5 – Loops & Iterations
- PY4E Video Lectures – Loops & Iterations



STEAM CLOWN™ PRODUCTIONS

**MOSTLY DR. CHARLES R.
SEVERANCE'S SLIDES**



**STEAM CLOWN™
& Squeaky Hinge
PRODUCTIONS**

© Copyright 2018 STEAM Clown™

Loops and Iteration

Chapter 5

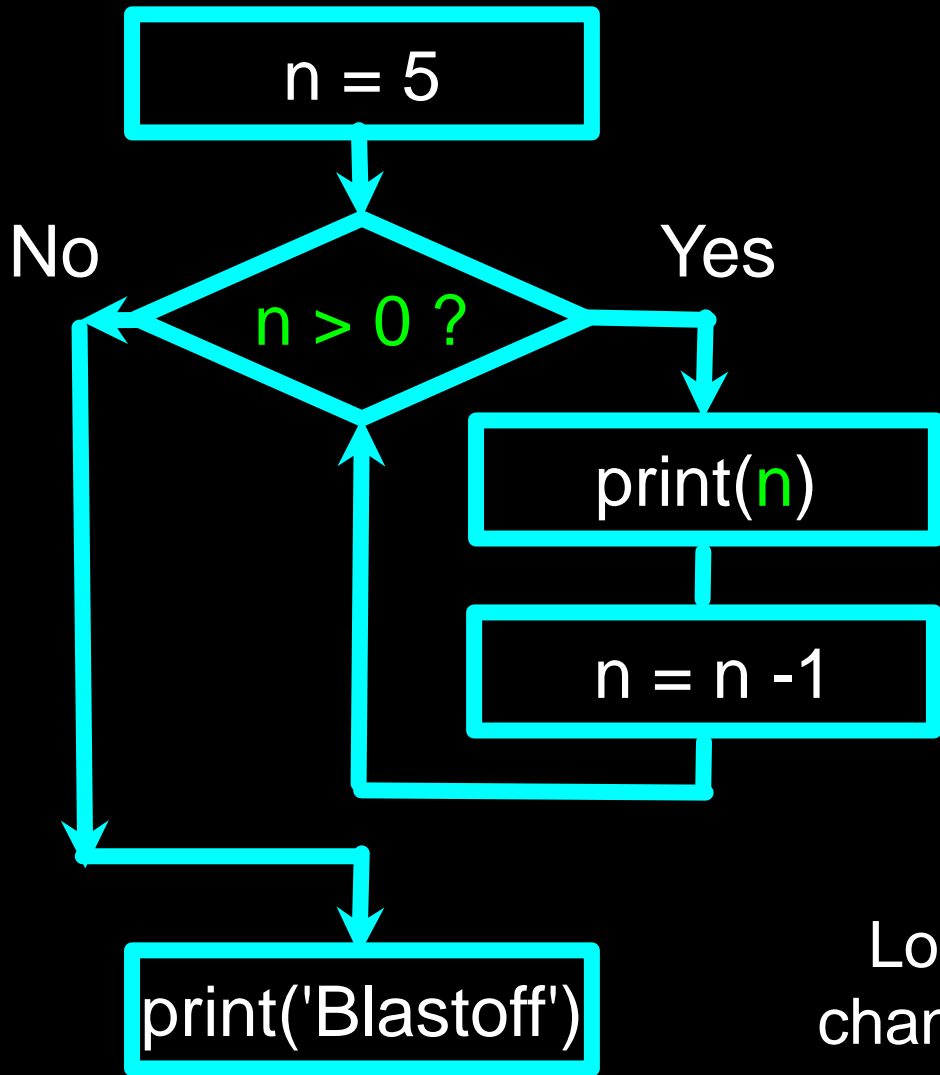


Python for Everybody

www.py4e.com



Repeated Steps



```
n = 5
while n > 0 :
    print(n)
    n = n - 1
print('Blastoff!')
print(n)
```

Output:

5
4
3
2
1
Blastoff!
0

Loops (repeated steps) have **iteration variables** that change each time through a loop. Often these **iteration variables** go through a sequence of numbers.

LAB #1 - SIMPLE WHILE LOOP

- Enter this code
- Try it with different numbers
- Can you make 'n' count by more than +1

```
n = 5
while n > 0 :
    print(n)
    n = n - 1
print('Blastoff!')
print(n)
```

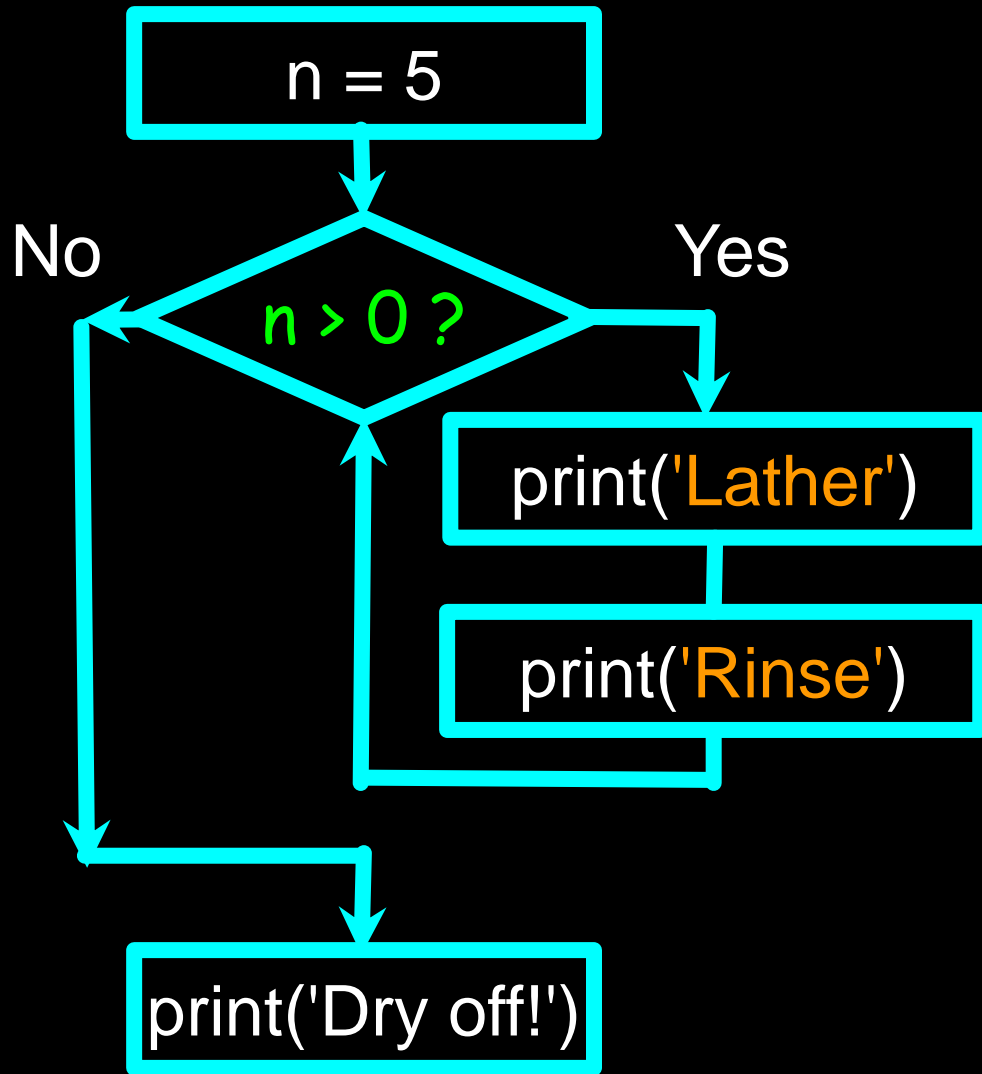
Type this code, then save, and when finished turn in...
then save as, and use for next lab



STEAM CLOWN™
& **Squeaky Hinge**
PRODUCTIONS

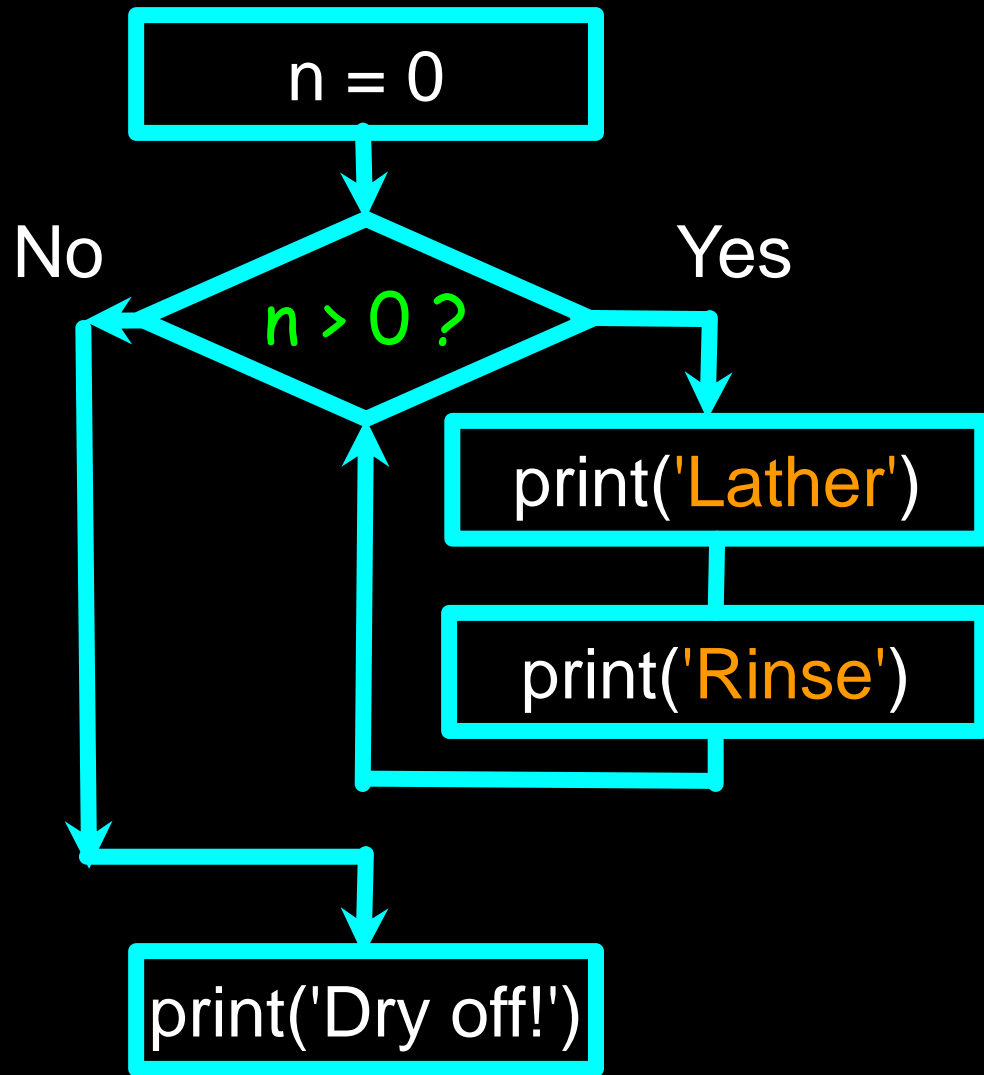
© Copyright 2018 STEAM Clown™

An Infinite Loop



```
n = 5
while n > 0 :
    print('Lather')
    print('Rinse')
print('Dry off!')
```

What is wrong with this loop?



Another Loop

```
n = 0
while n > 0 :
    print('Lather')
    print('Rinse')
    print('Dry off!')
```


What is this loop doing?

Breaking Out of a Loop

- The **break** statement ends the current loop and jumps to the statement immediately following the loop
- It is like a loop test that can happen anywhere in the body of the loop

```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```


> hello there
hello there
> finished
finished
> done
Done!



LAB #2 - "BREAK" LOOP

- Enter this code
- How do you end this loop?

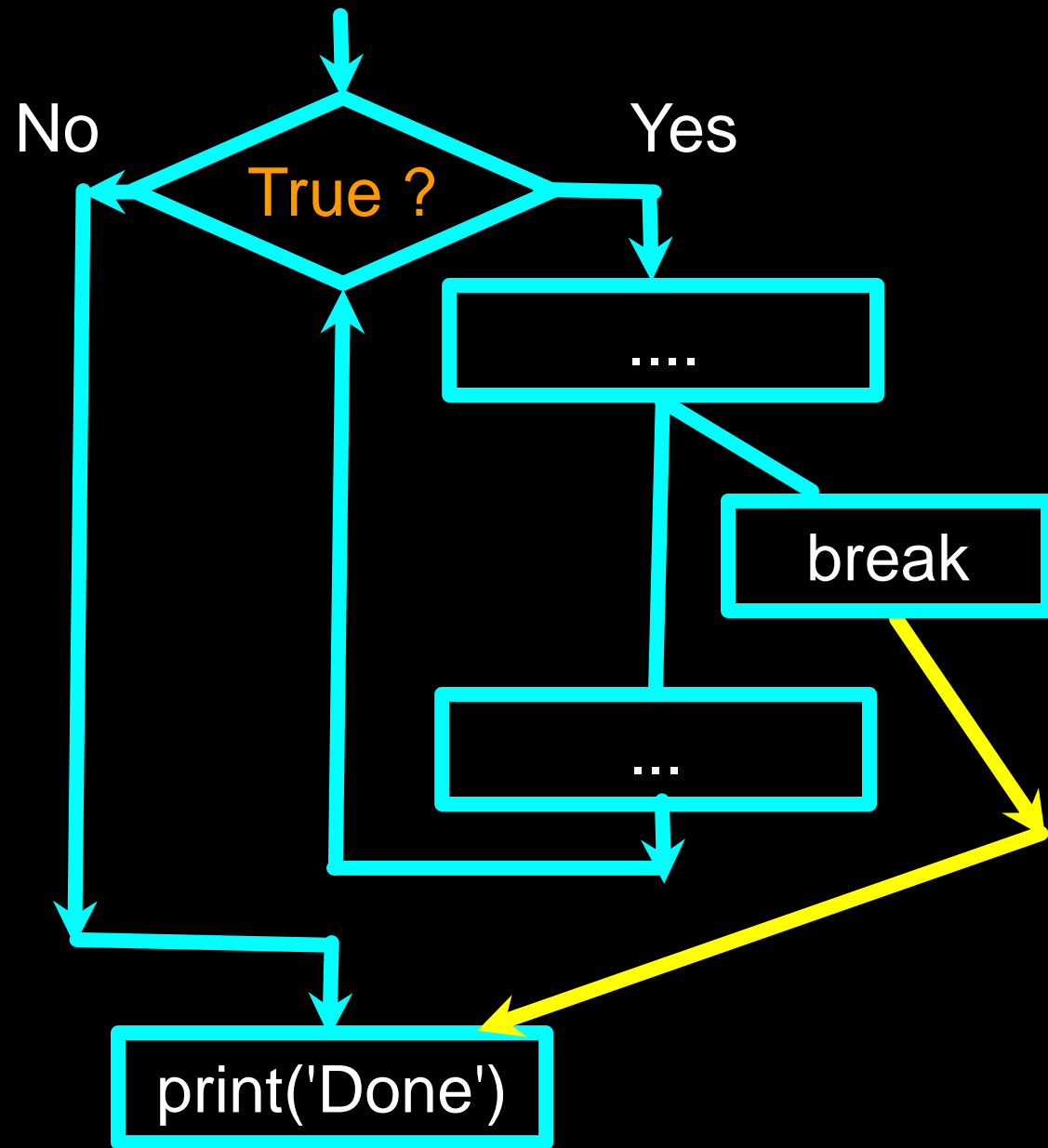
```
while True:  
    line = input('> ')  
    if line == 'done' :  
        break  
    print(line)  
print('Done!')
```



```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```



[http://en.wikipedia.org/wiki/Transporter \(Star Trek\)](http://en.wikipedia.org/wiki/Transporter_(Star_Trek))



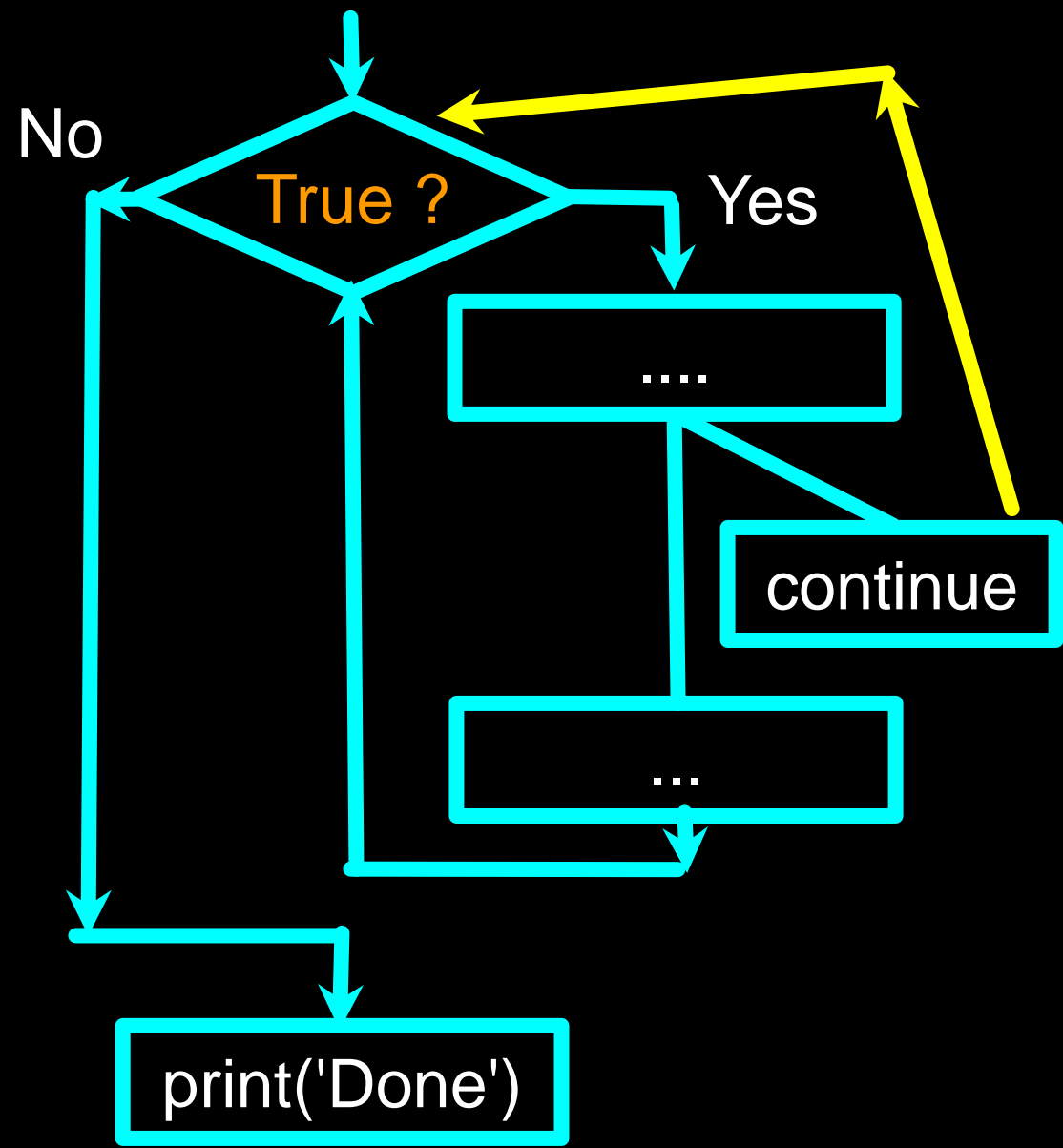
Finishing an Iteration with `continue`

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

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

> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!

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



INDEFINITE LOOPS

- While loops are called “indefinite loops” because they keep going until a logical condition becomes **False**
- The loops we have seen so far are pretty easy to examine to see if they will terminate or if they will be “infinite loops”
- Sometimes it is a little harder to be sure if a loop will terminate

DEFINITE LOOPS

- Iterating over a set of items...

Definite Loops

- Quite often we have a **list** of items of the **lines in a file** - effectively a **finite set** of things
- We can write a loop to run the loop once for each of the items in a set using the Python **for** construct
- These loops are called “**definite loops**” because they execute an exact number of times
- We say that “**definite loops iterate through the members of a set**”

A Simple Definite Loop

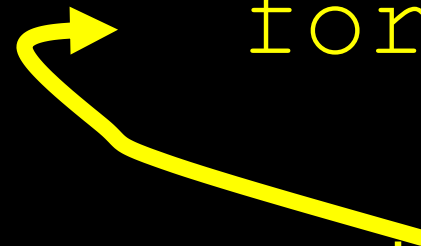
```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Blastoff!')
```

5
4
3
2
1
Blastoff!

LAB #3 - "FOR" LOOP

- Enter this code
- How does this loop end?

```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Blastoff!')
```



Looking at in...

- The **iteration variable** “iterates” through the **sequence** (ordered set)
- 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**

Iteration variable



Five-element
sequence



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

A Definite Loop with Strings

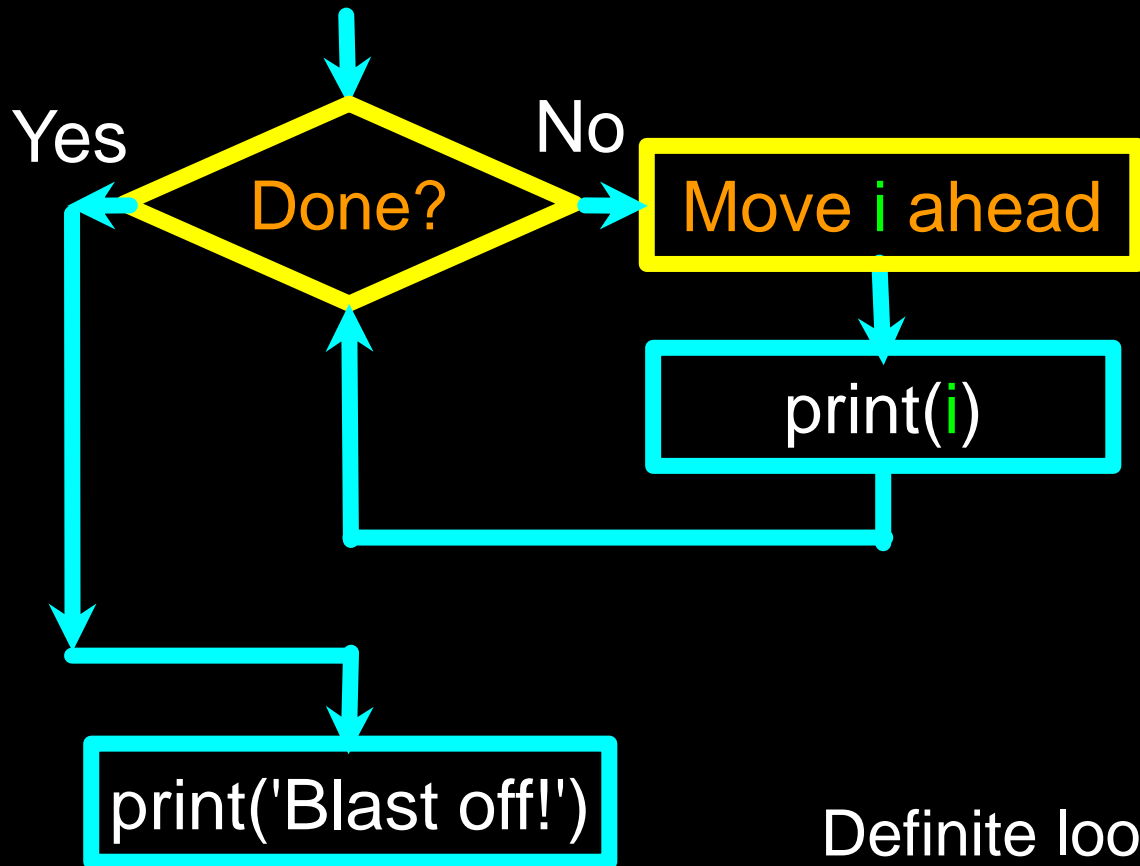
```
friends = ['Joseph', 'Glenn',  
'Sally']  
for friend in friends :  
    print('Happy New Year:', friend)  
print('Done!')
```



Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally

Done!

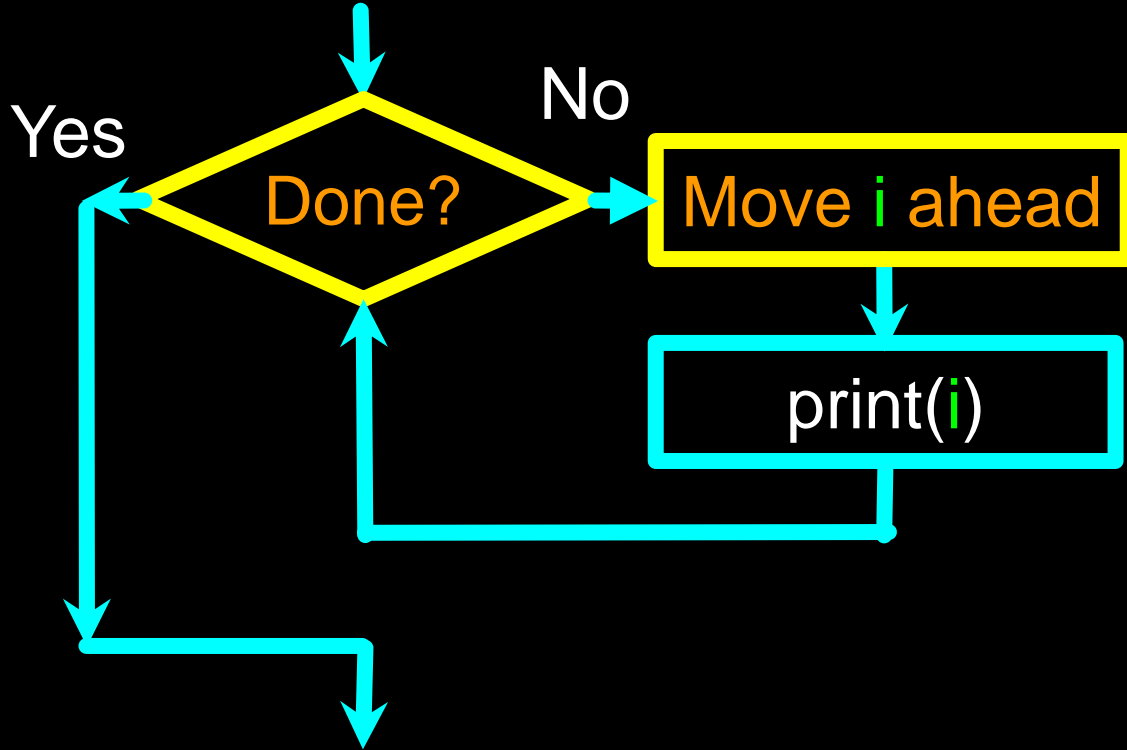
A Simple Definite Loop



```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Blastoff!')
```

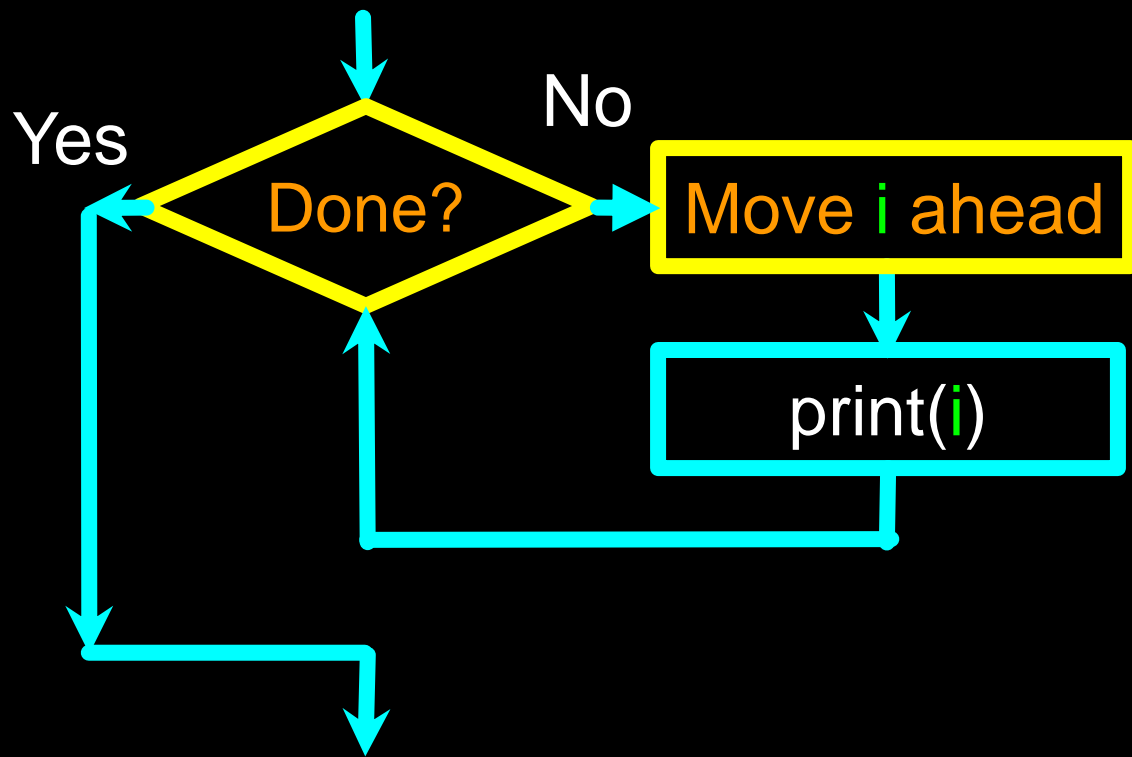
5
4
3
2
1
Blastoff!

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

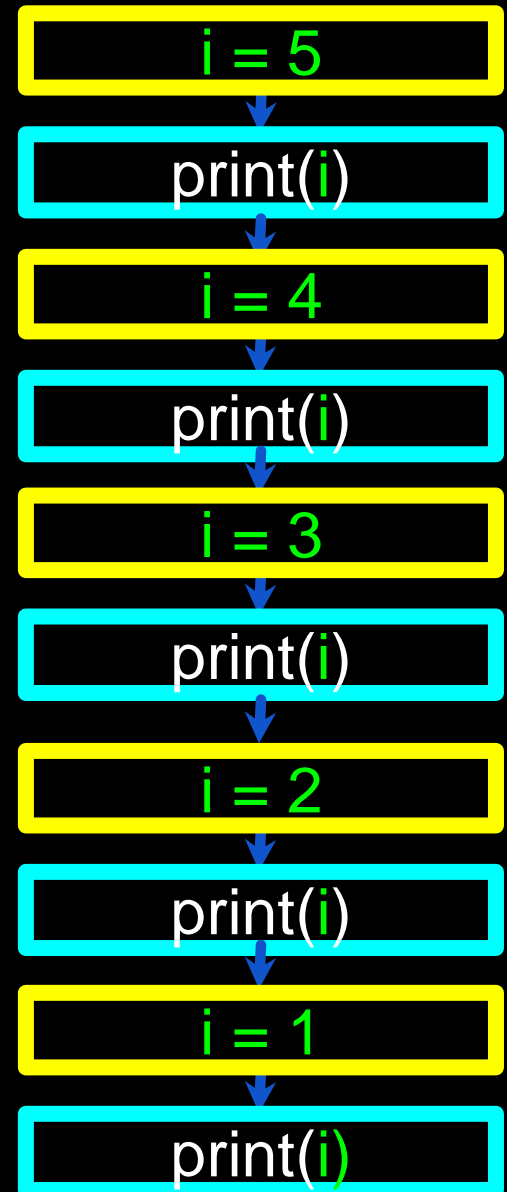


```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```

- The **iteration variable** “iterates” through the **sequence** (ordered set)
- 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**



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



Loop Idioms: What We Do in Loops

Note: Even though these examples are simple,
the patterns apply to all kinds of loops

Making “smart” loops

The trick is “knowing” something about the whole loop when you are stuck writing code that only sees one entry at a time

Set some variables to initial values

for thing in data:

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

Look at the variables

Looping Through a Set

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

```
$ python basicloop.py
```

```
Before
```

```
9
```

```
41
```

```
12
```

```
3
```

```
74
```

```
15
```

```
After
```

LAB #4 - DOING SOMETHING IN A "FOR" LOOP

- Enter this code
- What should we do?

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



What is the Largest Number?

What is the Largest Number?

3

What is the Largest Number?

41

What is the Largest Number?

12

What is the Largest Number?

9

What is the Largest Number?

74

What is the Largest Number?

15

What is the Largest Number?

What is the Largest Number?

3 41 12 9 74 15

What is the Largest Number?

largest_so_far

-1

What is the Largest Number?

3

largest_so_far

3

What is the Largest Number?

41

largest_so_far

41

What is the Largest Number?

12

largest_so_far

41

What is the Largest Number?

9

largest_so_far

41

What is the Largest Number?

74

largest_so_far

74

What is the Largest Number?

15

74

What is the Largest Number?

3 41 12 9 74 15

74

LAB #5 - FIND THE LARGEST NUMBER

- Enter this code
- Find the Largest Number

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



STEAM CLOWN™
& **Squeaky Hinge**
PRODUCTIONS

© Copyright 2018 STEAM Clown™

MORE LOOP PATTERNS...

Counting in a Loop

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

```
$ python countloop.py
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

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

\$ python countloop.py
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)
```

```
$ python averageloop.py
Before 0 0
1 9 9
2 50 41
3 62 12
4 65 3
5 139 74
6 154 15
After 6 154 25.666
```

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')
```

```
$ python search1.py
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)
```

```
$ python search1.py
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.

How to Find the Smallest Value

```
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
        print(largest_so_far, the_num)

print('After', largest_so_far)
```

```
$ python largest.py
```

```
Before -1
```

```
9 9
```

```
41 41
```

```
41 12
```

```
41 3
```

```
74 74
```

```
74 15
```

```
After 74
```

How would we change this to make it find the smallest value in the list?

Finding the Smallest Value

```
smallest_so_far = -1
print('Before', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('After', smallest_so_far)
```

We switched the variable name to `smallest_so_far` and switched the `>` to `<`

Finding the Smallest Value

```
smallest_so_far = -1
print('Before', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('After', smallest_so_far)
```

\$ python smallbad.py
Before -1
-1 9
-1 41
-1 12
-1 3
-1 74
-1 15
After -1

We switched the variable name to `smallest_so_far` and switched the `>` to `<`

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)
```

```
$ python smallest.py
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
- Implies “is the same as”
- Similar to, but stronger than `==`
- `is not` also is a logical operator

LAB #6 - ADD A LAB

- Enter this code

Summary

- While loops (indefinite)
- Infinite loops
- Using break
- Using continue
- None constants and variables
- For loops (definite)
- Iteration variables
- Loop idioms
- Largest or smallest



ACKNOWLEDGEMENTS / CONTRIBUTIONS



These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here

ASSESSMENT

- Assessment Type(s):
 - ✓ Demonstrations
 - ✓ Interviews
 - ✓ Journals
 - ✓ Observations
 - ✓ Labs
 - ✓ Projects
 - ✓ Portfolios
 - ✓ Rubrics
 - ✓ Surveys
 - ✓ Teacher-Made Test
 - ✓ Writing Samples



STEAM CLOWN™
& **Squeaky Hinge**
PRODUCTIONS

© Copyright 2018 STEAM Clown™



STEAM CLOWN™ PRODUCTIONS

REFERENCE SLIDES



STEAM CLOWN™
& **Squeaky Hinge**
PRODUCTIONS

© Copyright 2018 STEAM Clown™

Finding the Largest Value

```
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
        print(largest_so_far, the_num)

print('After', largest_so_far)
```

```
$ python largest.py
```

```
Before -1
```

```
9 9
```

```
41 41
```

```
41 12
```

```
41 3
```

```
74 74
```

```
74 15
```

```
After 74
```

We make a variable that contains the largest value we have seen so far. If the current number we are looking at is larger, it is the new largest value we have seen so far.



STEAM CLOWN™ PRODUCTIONS

APPENDIX



STEAM CLOWN™ PRODUCTIONS

CAN I GET A COPY OF THESE SLIDES? YES, PROBABLY...

Most presentation lecture slides can be found indexed on www.steamclown.org and maybe blogged about here on [Jim The STEAM Clown's Blog](#), and on [STEAM Clown's Mechatronics Engineering Google site](#), where you can search for the presentation title. While you are there, sign up for email updates

APPENDIX A: LICENSE & ATTRIBUTION

- This interpretation is primarily the Intellectual Property of Jim Burnham, Top STEAM Clown, at STEAMClown.org
- This presentation and content is distributed under the Creative Commons License CC-BY-NC-SA 4.0
- My best attempt to properly attribute, or reference any other sources or work I have used are listed in Appendix C



Under the following terms:



Attribution — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.



NonCommercial — You may not use the material for [commercial purposes](#).



ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the [same license](#) as the original.

No additional restrictions — You may not apply legal terms or [technological measures](#) that legally restrict others from doing anything the license permits.

Please maintain this slide with any modifications you make

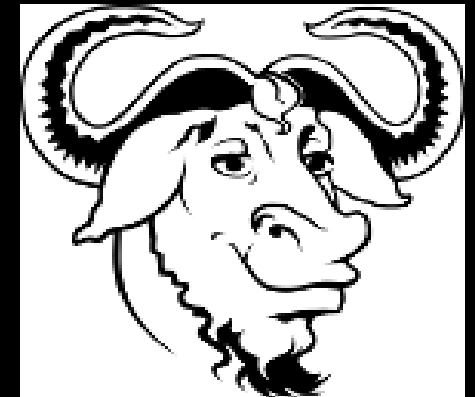


STEAM CLOWN™
& **Squeaky Hinge**
PRODUCTIONS

© Copyright 2018 STEAM Clown™

APPENDIX B: CODE LICENSE & ATTRIBUTION

- This interpretation is primarily the Intellectual Property of Jim Burnham, Top STEAM Clown, at STEAMClown.org
- The programming code found in this presentation or linked to on my Github site is distributed under the:
 - GNU General Public License v3.0
 - European Union Public Licence EUPL 1.2 or later
- My best attempt to properly attribute, or reference any other sources or work I have used are listed in Appendix C



Please maintain this slide with any modifications you make



STEAM CLOWN™
& **Squeaky Hinge**
PRODUCTIONS

© Copyright 2018 STEAM Clown™

APPENDIX C: PRIMARY SOURCES & ATTRIBUTION FOR MATERIAL USED

- Charles R. Severance slides can be found on the <https://www.py4e.com/> site are Copyright 2010 - Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.
 - Initial Development: Charles Severance, University of Michigan School of Information
 - Modifications and Adaptions by Jim Burnham, Top Clown @ www.steamclown.org



Please maintain this slide with any modifications you make



STEAM CLOWN™
& **Squeaky Hinge**
PRODUCTIONS

© Copyright 2018 STEAM Clown™