香港浸會大學
HONG KONG BAPTIST UNIVERSITY

Functions

JOUR7280/COMM7780

Big Data Analytics for Media and Communication

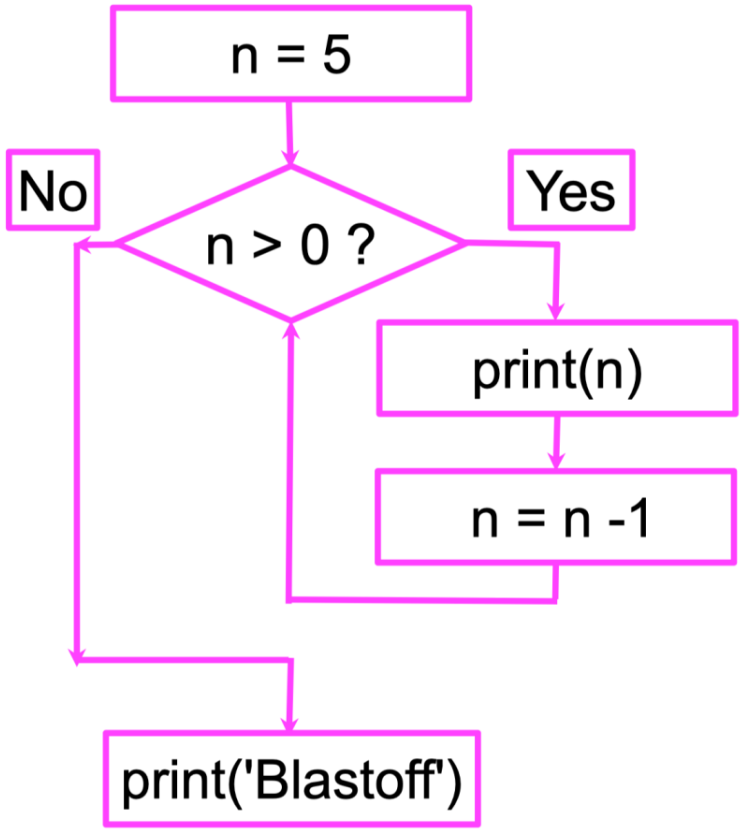
Instructor: Dr. Xiaoyi Fu

Agenda

- Indefinite Loops
  - while loop
- Definite Loops
  - for loop
- Loop Idioms

Indefinite Loops

Repeated steps



Program

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

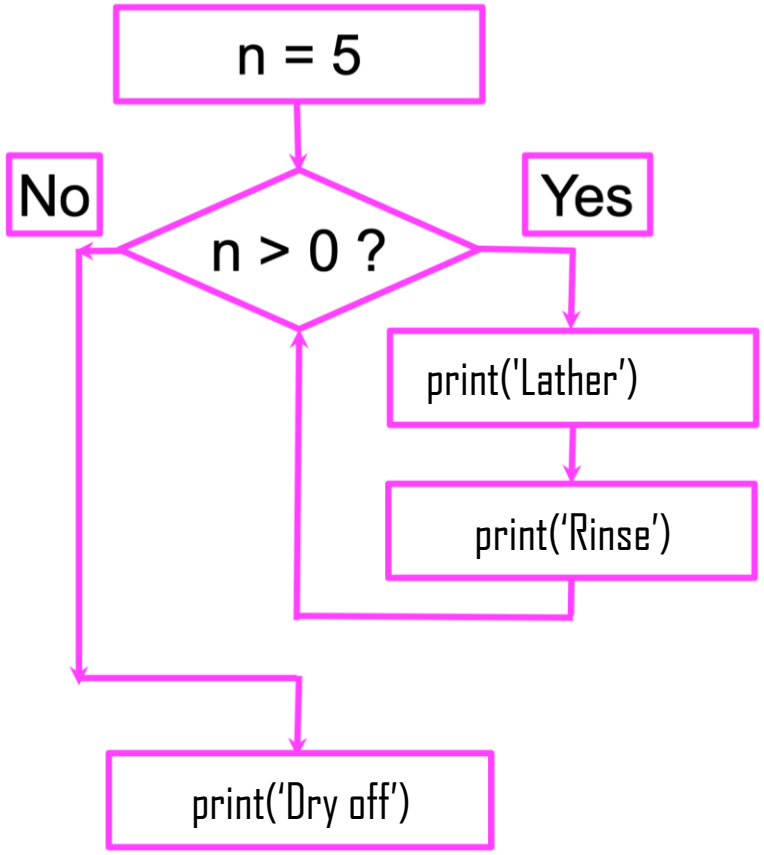
Output

```
5
4
3
2
1
Blastoff!
0
```

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

5 Loops. ipynb

An Infinite Loop



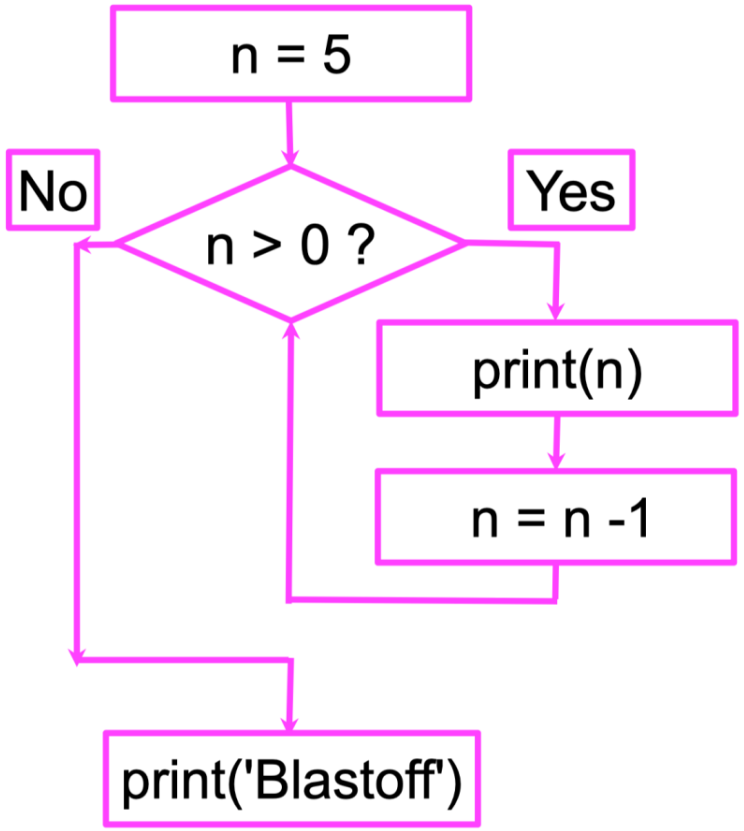
Program

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

What is wrong with this loop?

5 Loops.ipynb

Repeated steps



Program

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

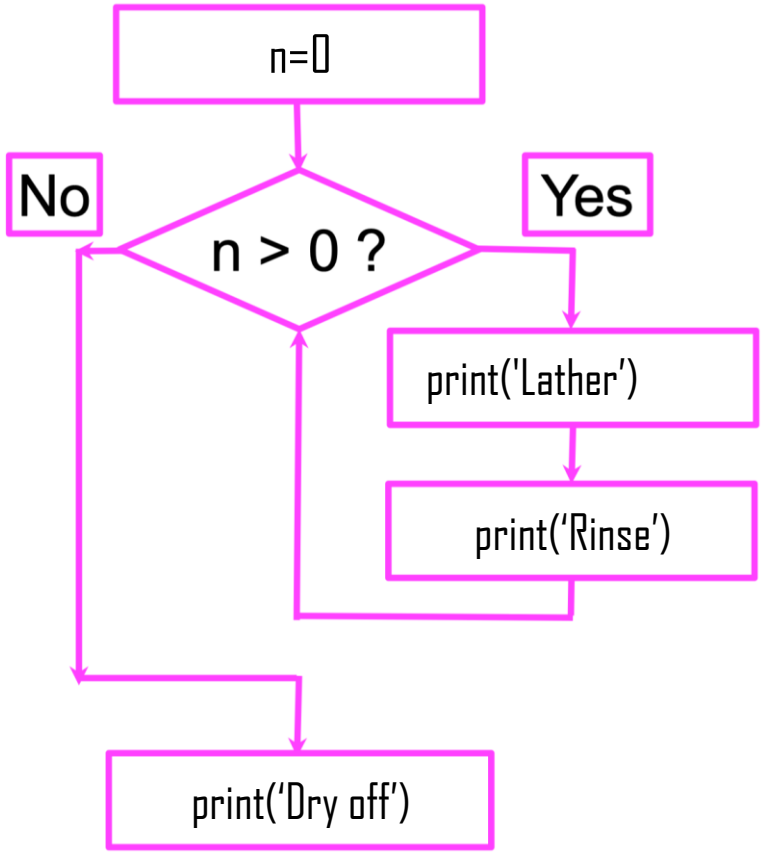
Output

```
5
4
3
2
1
Blastoff!
0
```

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

5 Loops.ipynb

Another Loop



Program

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

What is this loop doing?

zero-trip loop

5 Loops.ipynb

Break 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!')
```

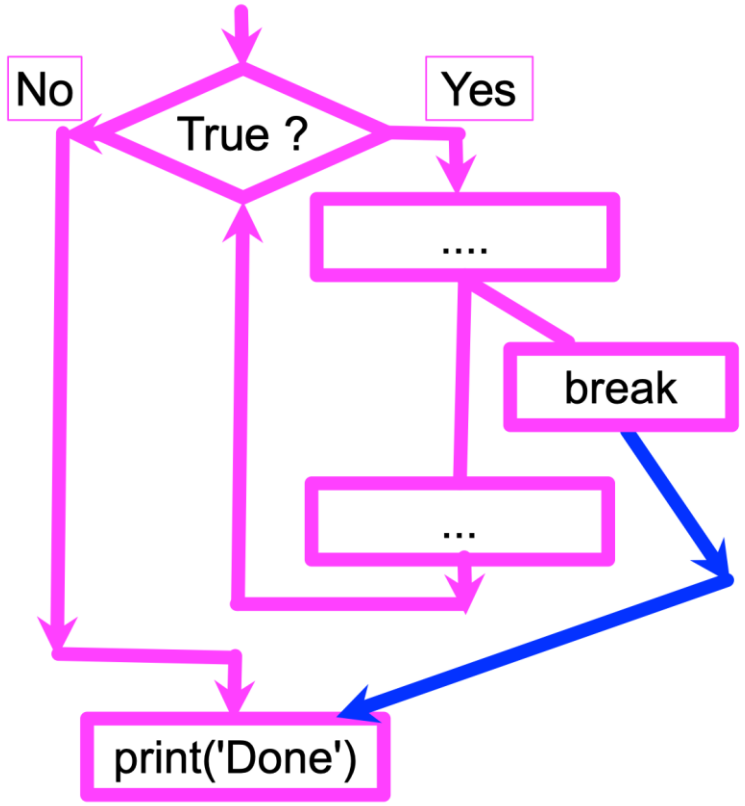
Output

```
> Hello there
Hello there
> finished
finished
> done
Done!
```

5 Loops.ipynb

Break Out of a Loop

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



Finish an Iteration with Continue

- The continue statement ends the current iteration and jumps to the top of the loop and starts the next iteration.
- Sometimes you are in an iteration of a loop and want to finish the current iteration and immediately jump to the next iteration.
- In that case you can use the continue statement to skip to the next iteration without finishing the body of the loop for the current iteration.

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

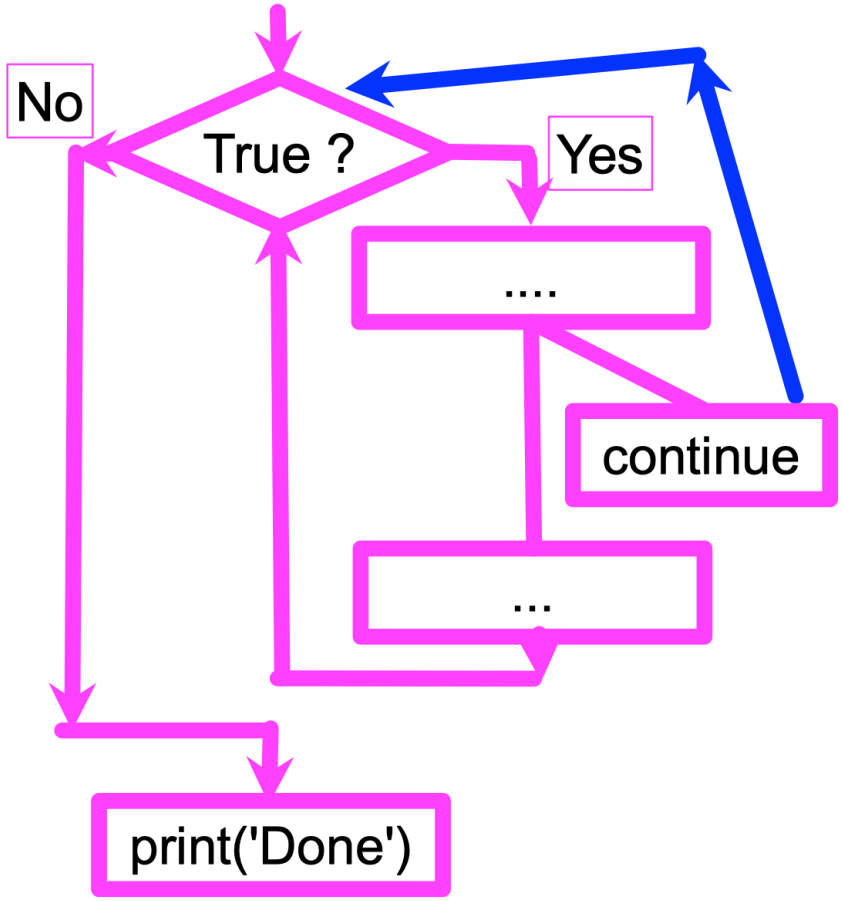
Output

```
> Hello there
Hello there
> # don't print this
> print this
print this
> done
Done!
```

5 Loops.ipynb

Finish an Iteration with Continue

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



Definite Loops

A Simple Definite Loop

- When we have a list of things to loop through, we can construct a definite loop using a for statement.
- The iteration variable is explicitly part of the syntax.

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

Output:

5
4
3
2
1
Blastoff!

5 Loops.ipynb

A Definite Loop with Strings

```
friends = ['Tony', 'Peter', 'Natasha']
for friend in friends:
    print('Happy New Year:', friend)
print('Done!')
```

Output:

Happy New Year: Tony

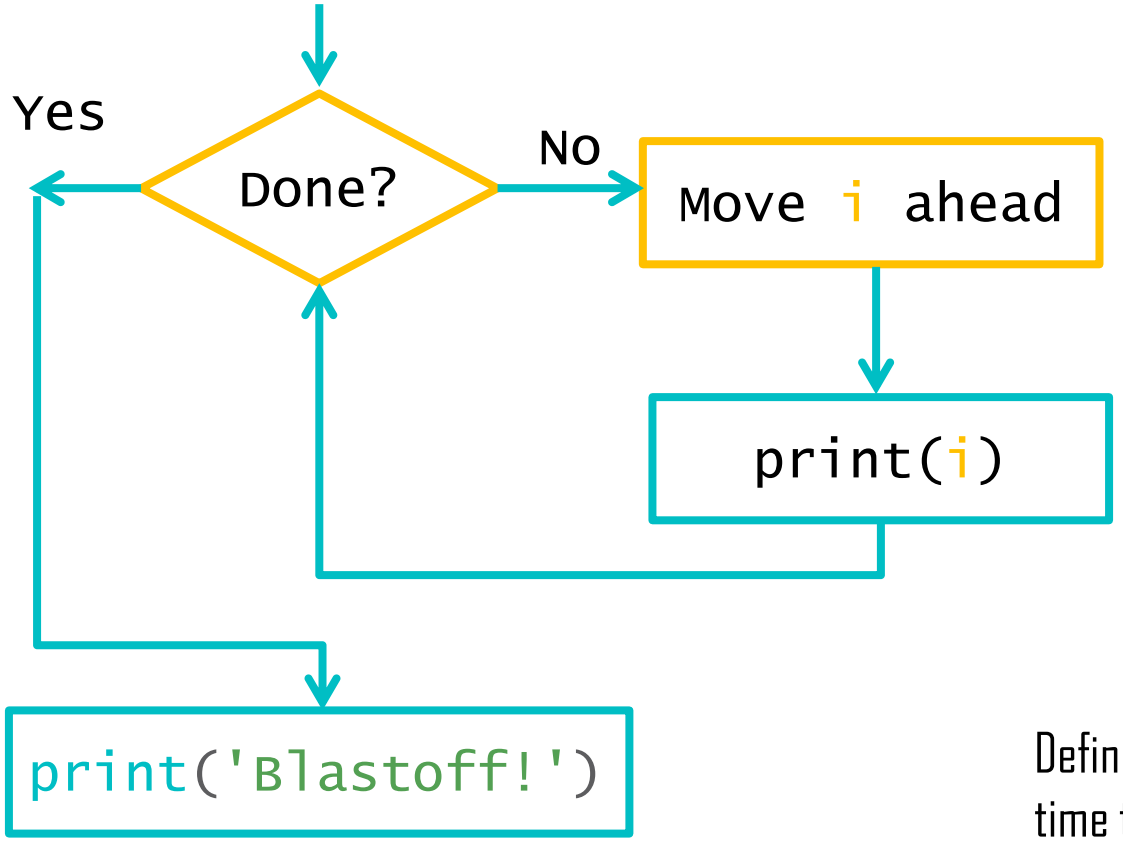
Happy New Year: Peter

Happy New Year: Natasha

Done!

5 Loops.ipynb

A Simple Definite Loop




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

Output:

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.

Look at in

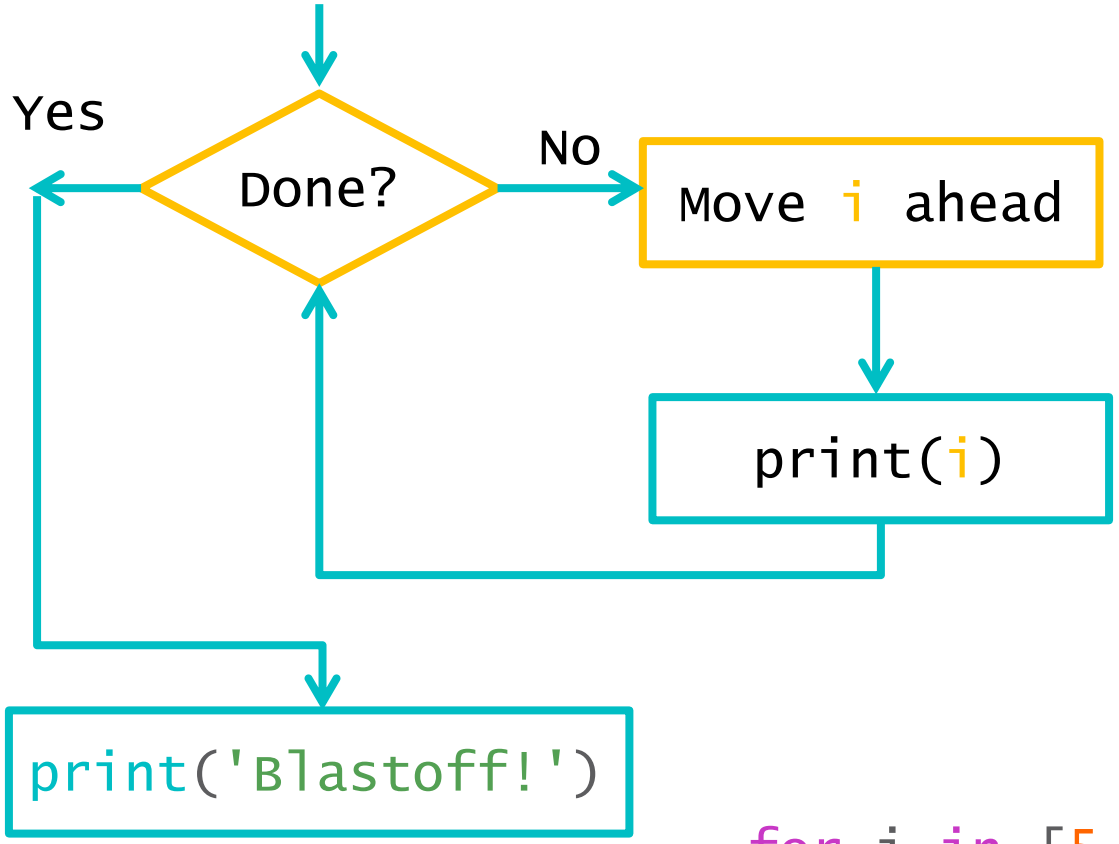
- The iteration variable "iterates" through the sequence (ordered set)
- The block (body) of code is executed once for each value in the sequence
- The iteration variable moves through all of the values in the sequence

Iteration variable

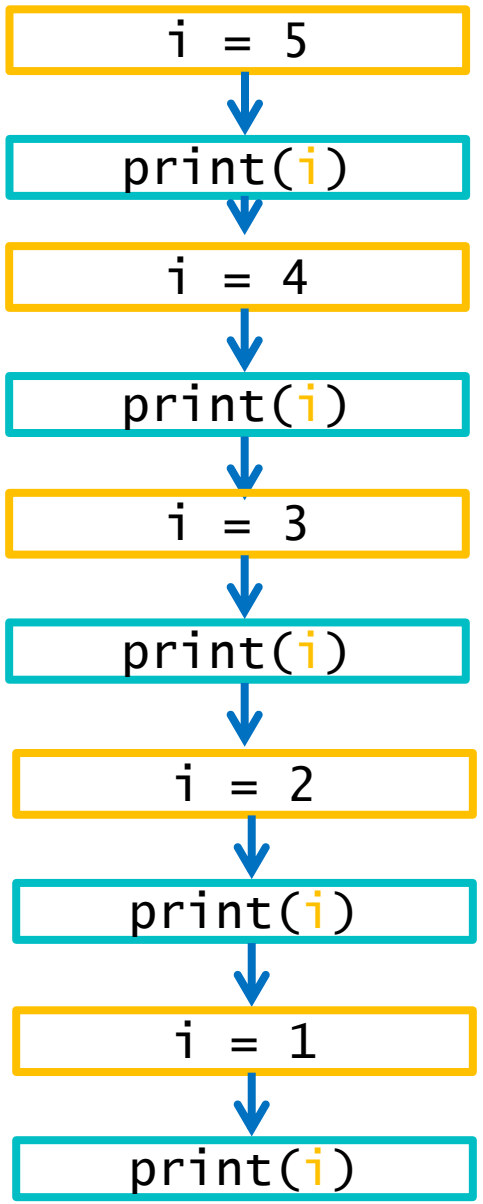
Five elements sequence

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

A Simple Definite Loop




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



Loop Idioms

What is the Largest Number?

9 41 12 3 74 15

- largest_so_far
  - Initialize: -1

Find 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)
```

Output:

```
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.

5 Loops.ipynb

Count 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)
```

Output:

```
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 1 to it each time through the loop.

5 Loops.ipynb

Find the Sum 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)
```

Output:

```
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.

5 Loops.ipynb

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.

5 Loops.ipynb

Search Using a Boolean Variable

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

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

- 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.

5 Loops.ipynb

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)
```

Output:

```
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?

Find 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)
```

Output:

```
Before -1
-1 9
-1 41
-1 12
-1 3
-1 74
-1 15
After -1
```

- We switch the variable name to smallest_so_far and switch the > to <

Find the Smallest Value

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

print('After', smallest_so_far)
```

```
Before None
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

5 Loops.ipynb

The “is” and “is not” Operators

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

print('After', smallest_so_far)
```

- Python has an is operator that can be used in logical expressions
- Implies “is the same as”
- Similar to, but stronger than “==”
- is not is also a logical operator
- Usually use them for a True, False, or None.
  - Don’t overuse is

Acknowledgements / Contributions

- Some of the slides used in this lecture from:
  - Charles R. Severance - University of Michigan School of Information

Thank You