Summer Coding School 2023

Aug 19, 2023

Taehee Jeong, Ph.D.

Data type

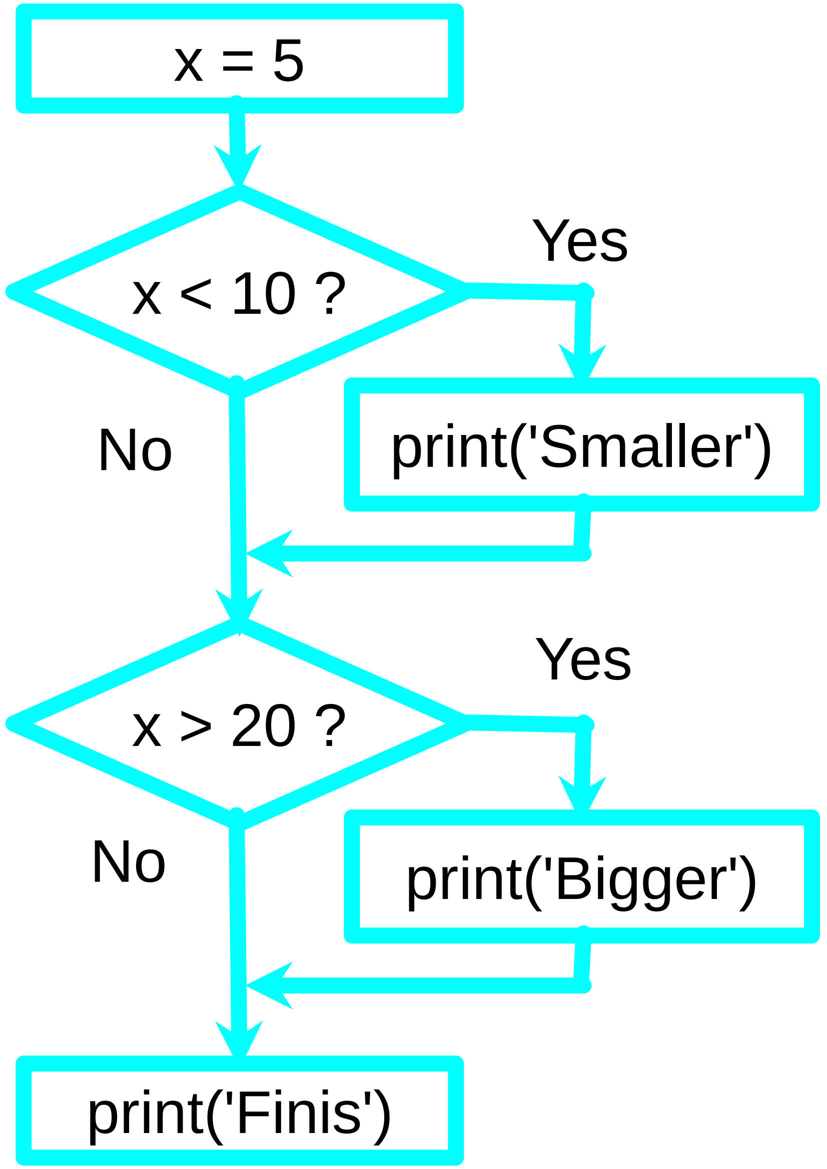
- Integer
- Floating point
- String
- Boolean

Data structure

- List
- Set
- Tuples
- Dictionary

Today’s Agenda

- Conditional and Control flows
- While Loop
- For Loop



Conditional Steps

Program:

```
x = 5
if x < 10:
    print('Smaller')
if x > 20:
    print('Bigger')

print('Finis')
```

Source: Python for Everybody, Charles Severance, University of Michigan School of Information

Comparison Operators

- Boolean expressions ask a question and produce a Yes or No result which we use to control program flow

- Boolean expressions using comparison operators evaluate to True / False or Yes / No

- Comparison operators look at variables but do not change the variables

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

Remember: "=" is used for assignment.

http://en.wikipedia.org/wiki/George_Boole

Comparison Operators

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

Indentation

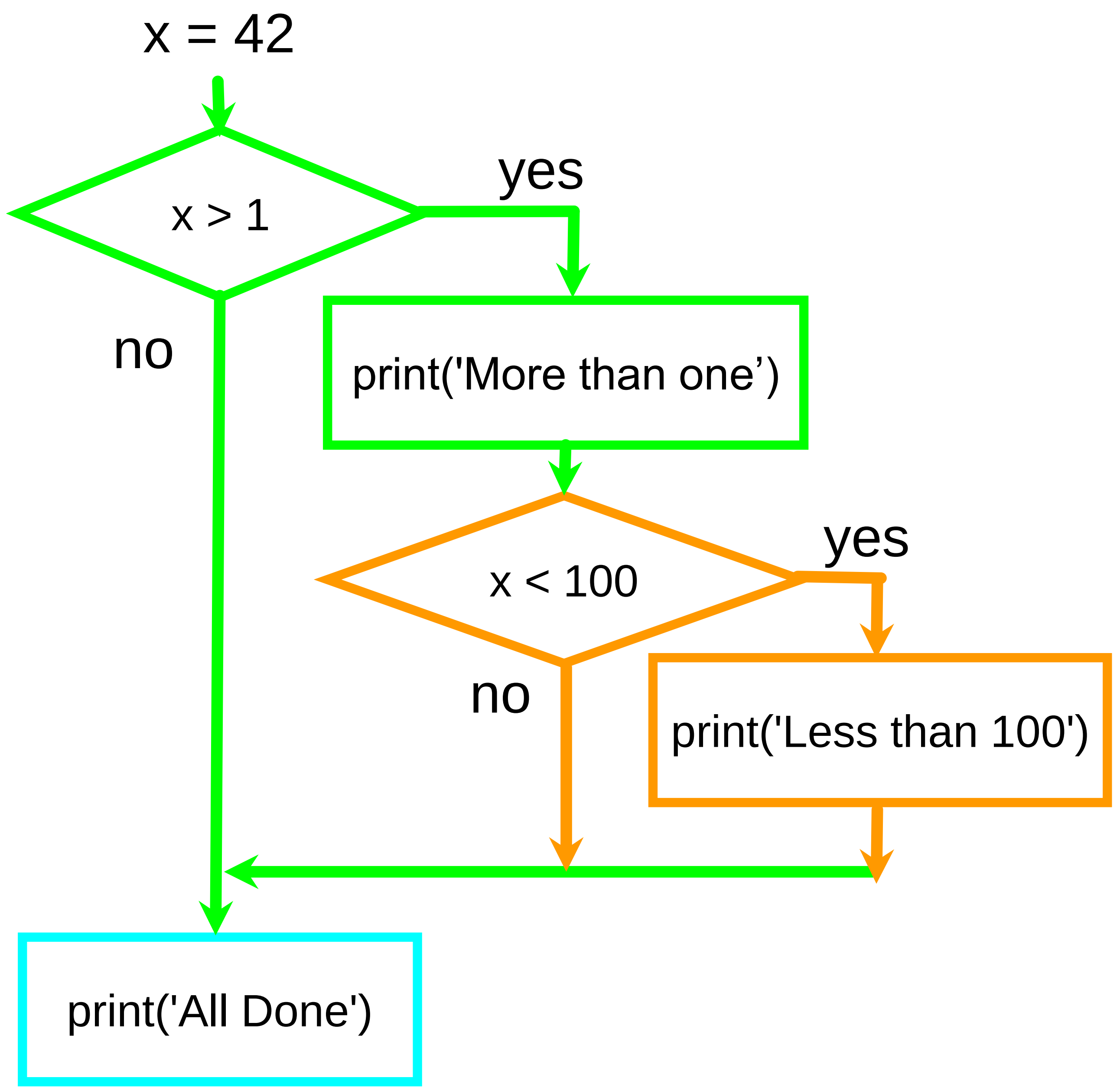
- Increase indent indent after an if statement or for statement (after :)
- Maintain indent to indicate the scope of the block (which lines are affected by the if/for)
- Reduce indent back to the level of the if statement or for statement to indicate the end of the block
- Blank lines are ignored - they do not affect indentation
- Comments on a line by themselves are ignored with regard to indentation

begin/end Blocks

```
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)
print('All Done')
```

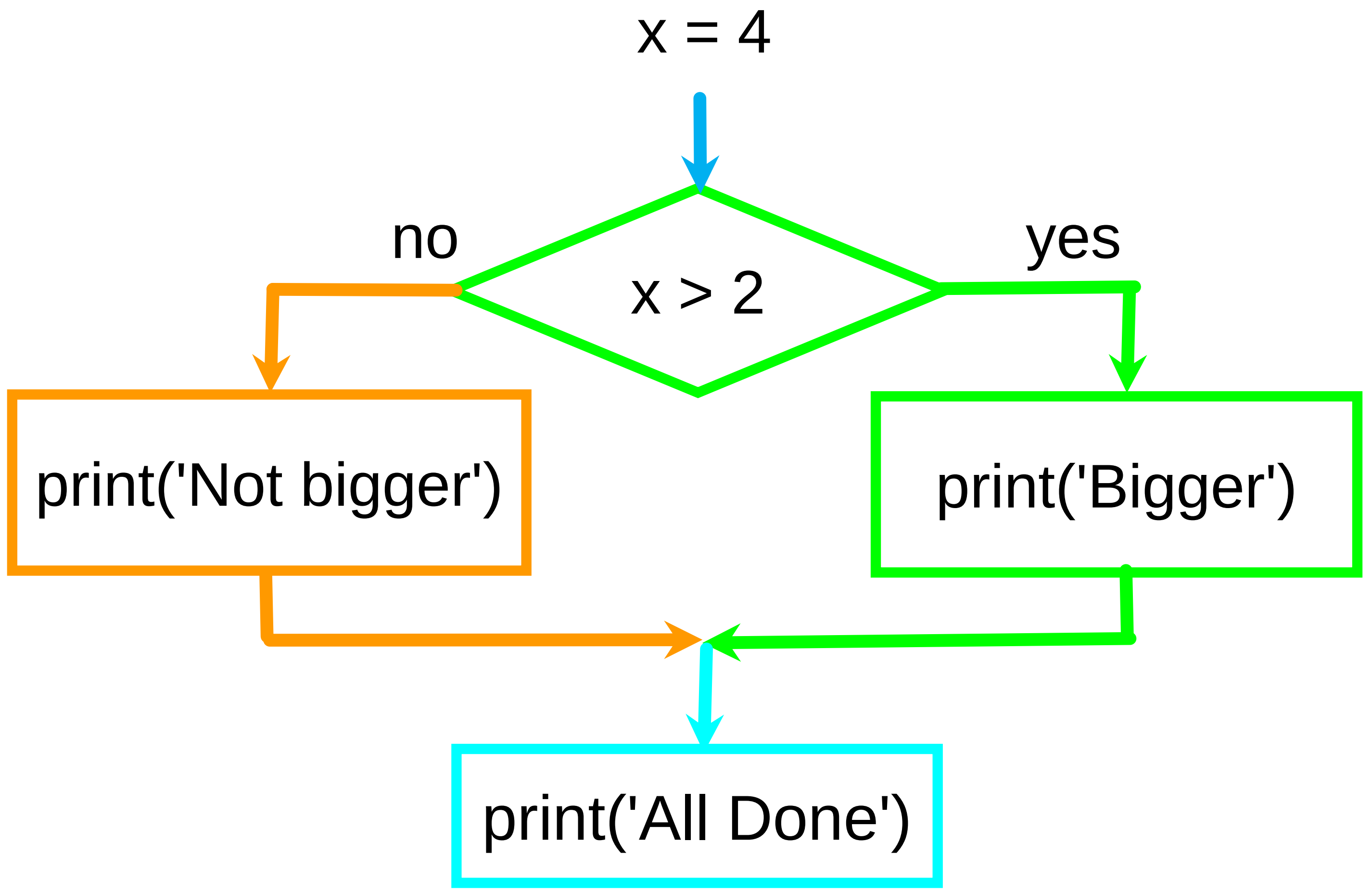
Nested Decisions

```
x = 42
if x > 1 :
    print('More than one')
    if x < 100 :
        print('Less than 100')
print('All done')
```



Source: Python for Everybody, Charles Severance, University of Michigan School of Information

Two-way Decisions

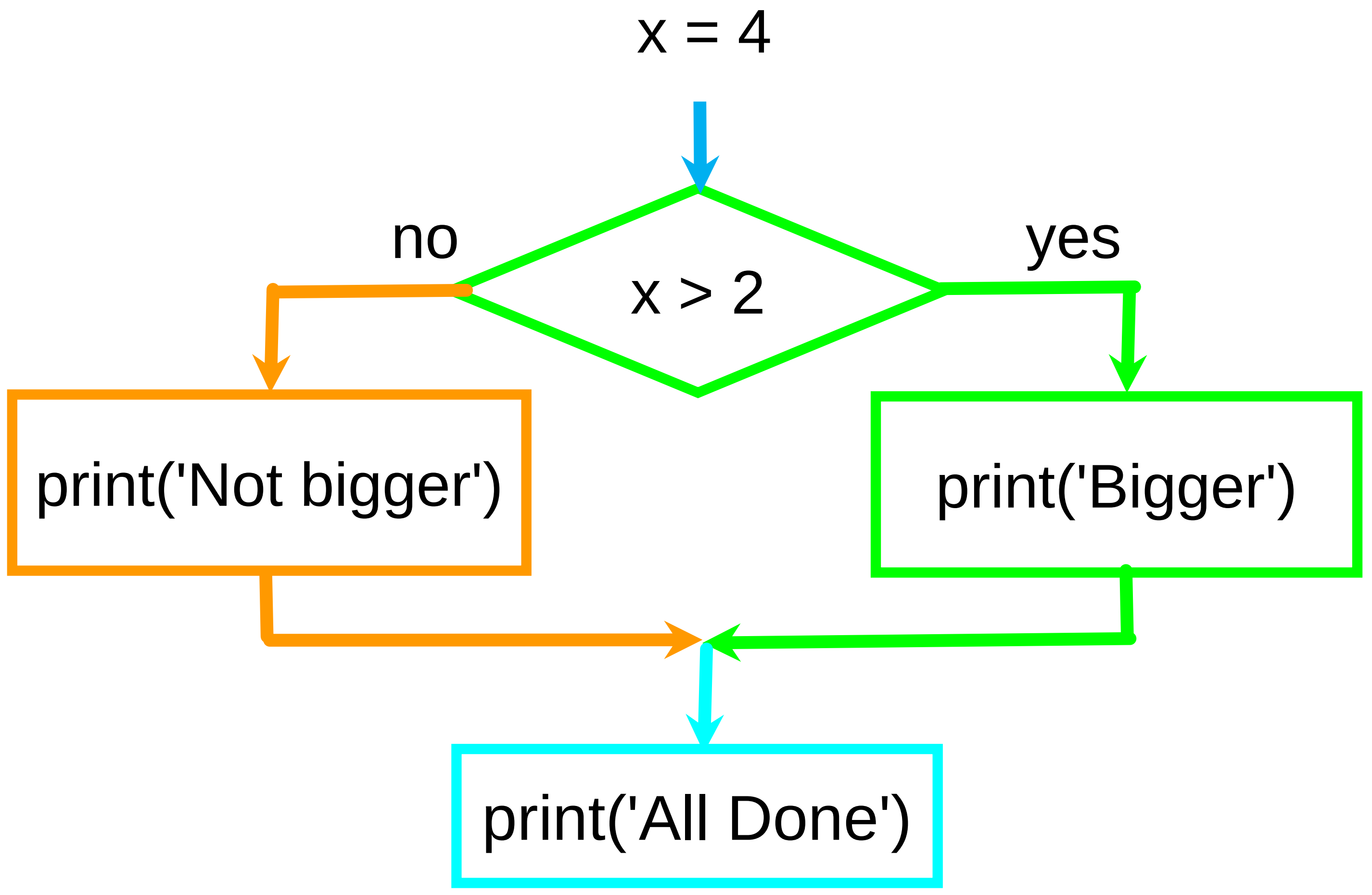


Sometimes we want to do one thing if a logical expression is true and something else if the expression is false

It is like a fork in the road - we must choose one or the other path but not both

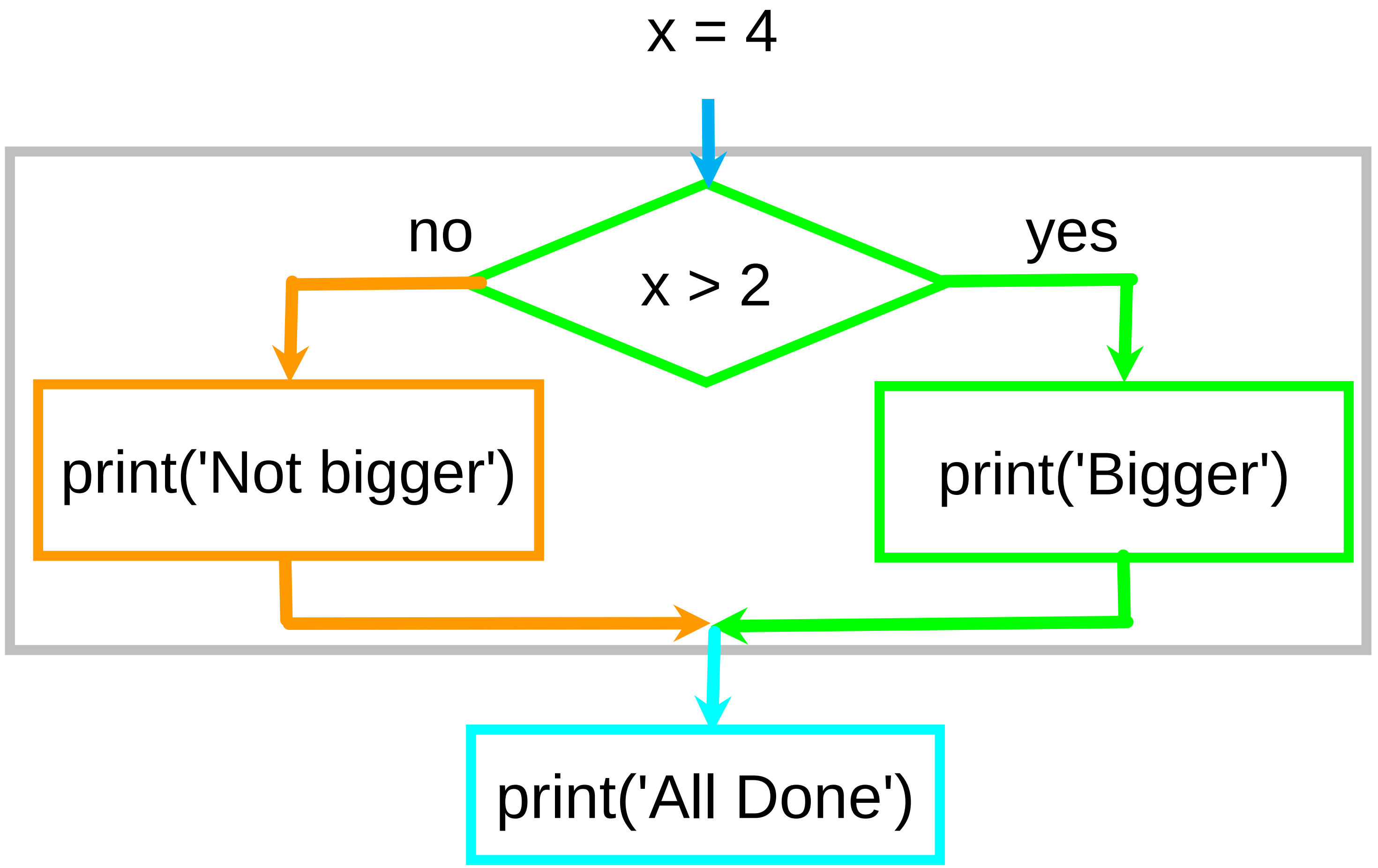
Two-way Decisions with else:

```
x = 4

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

print('All done')
```



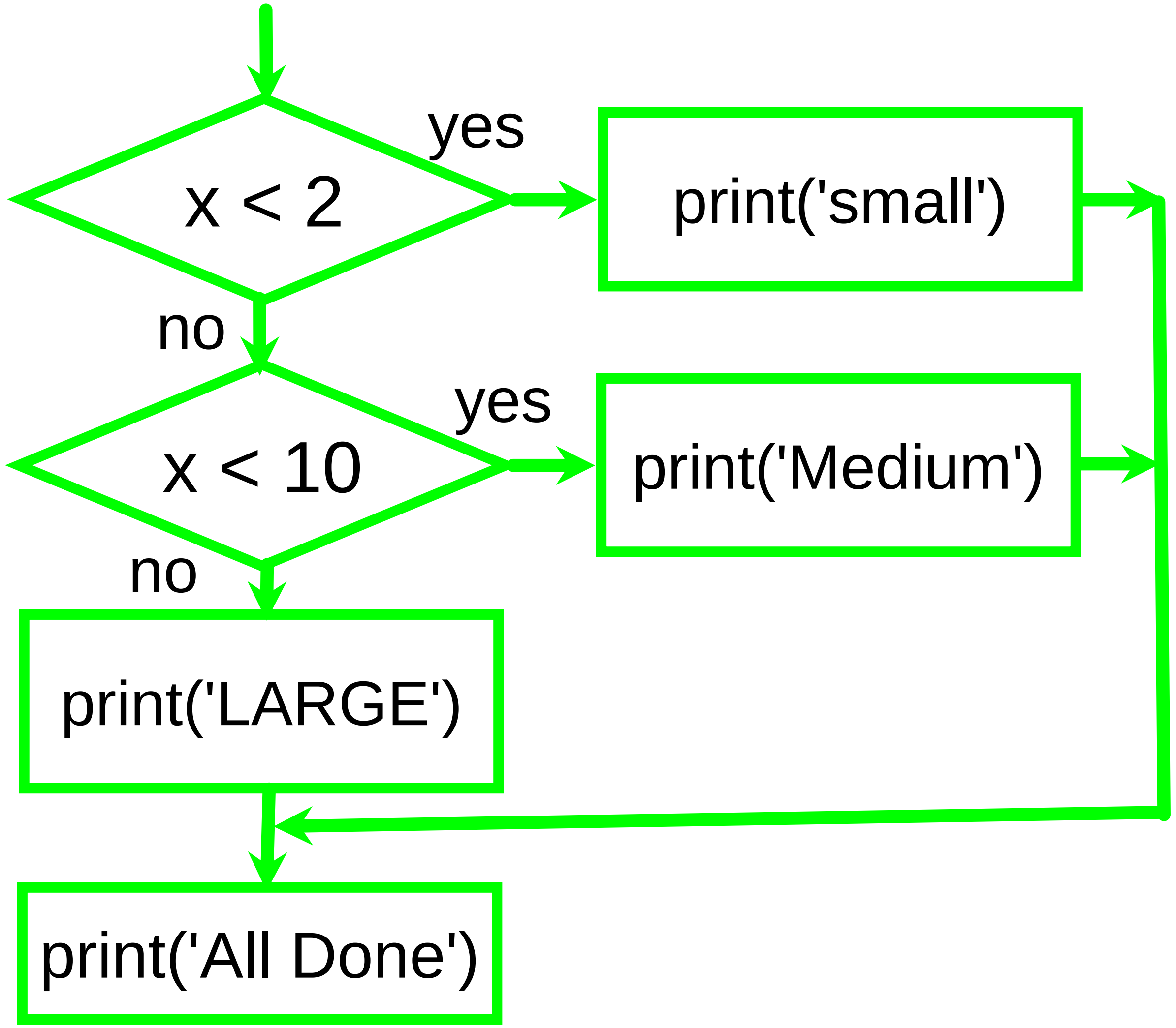
Visualize Blocks

```
x = 4

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

print('All done')
```



Multi-way

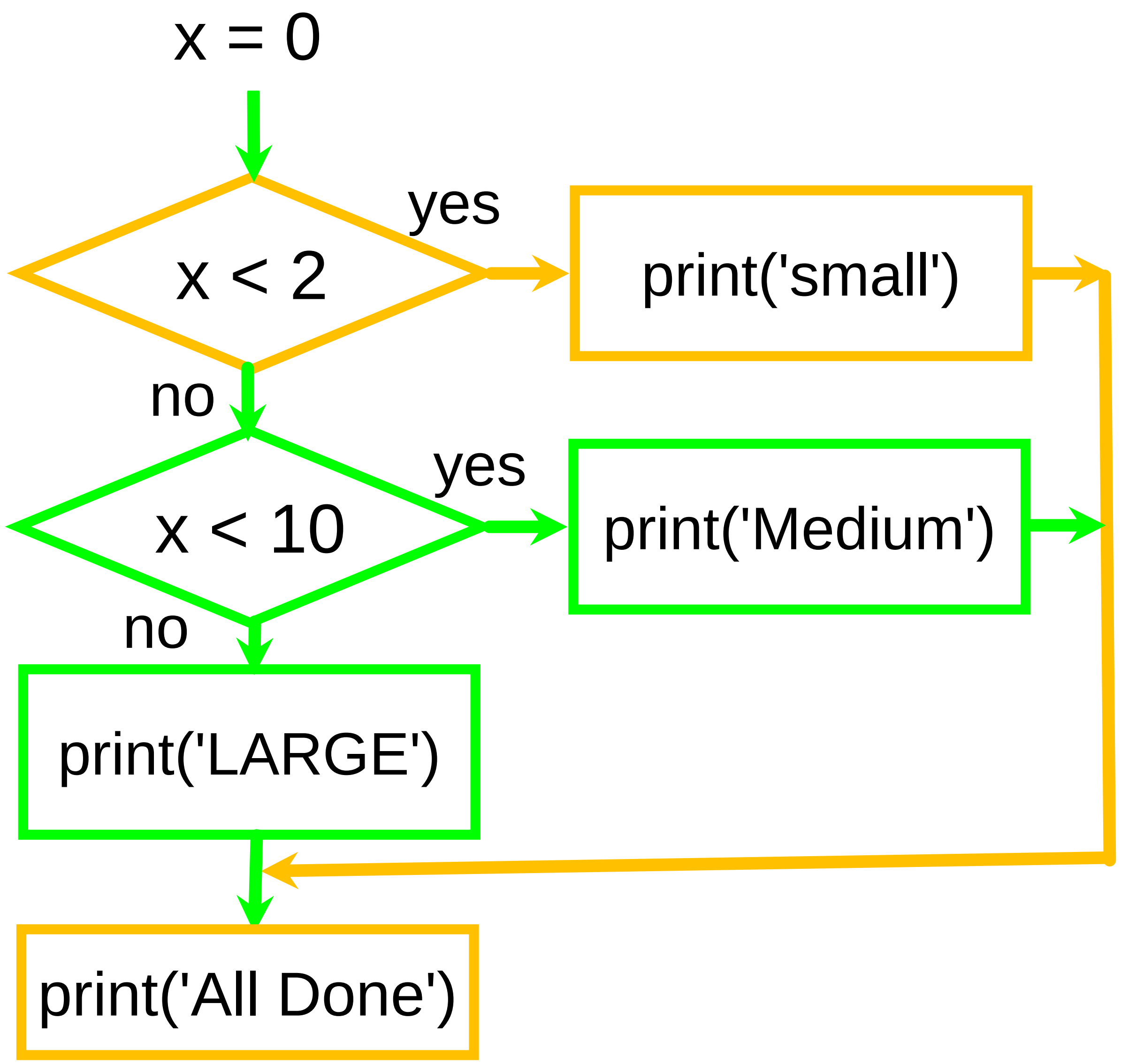
```
if x < 2 :
    print('small')
elif x < 10 :
    print('Medium')
else :
    print('LARGE')
print('All done')
```



Source: Python for Everybody, Charles Severance, University of Michigan School of Information

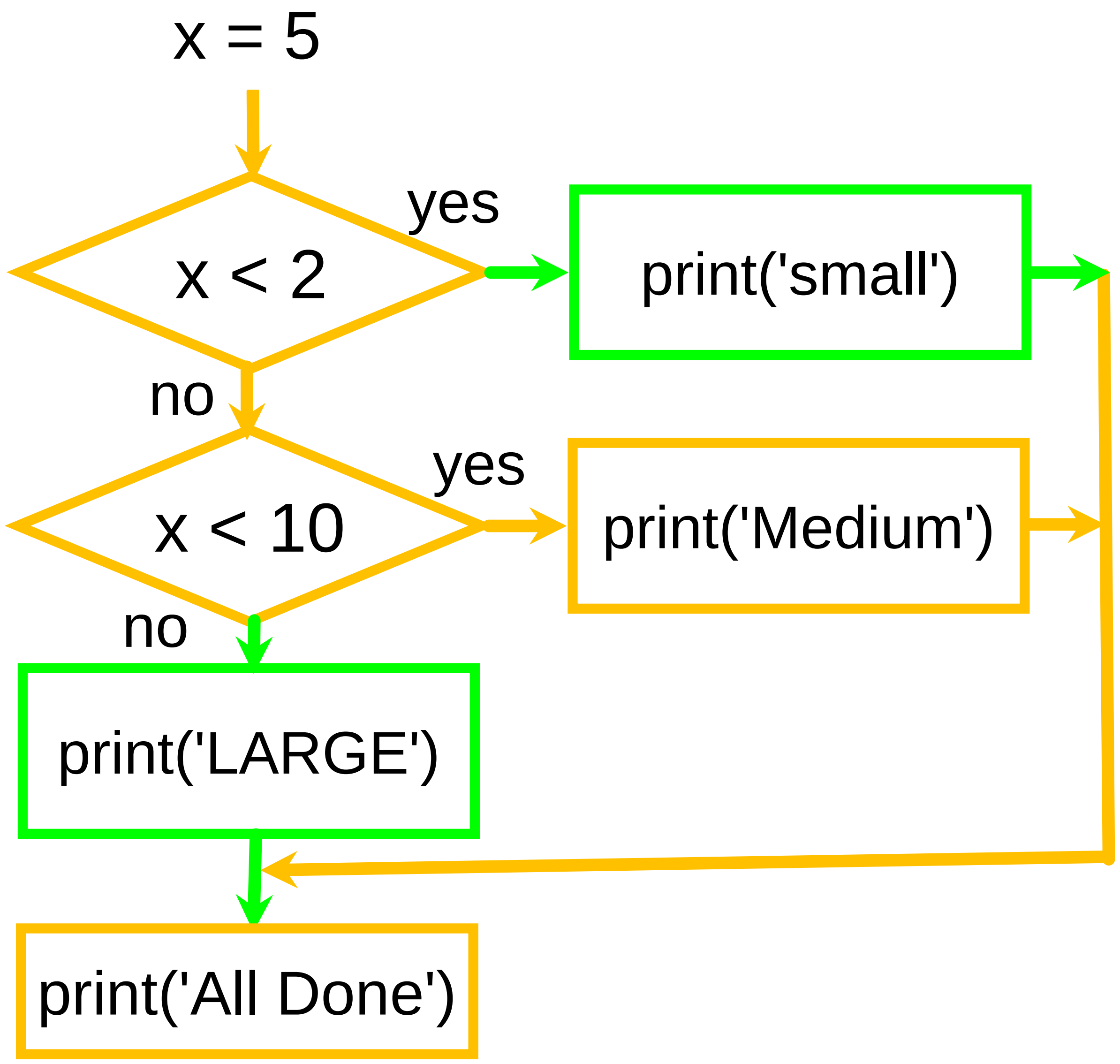
Multi-way

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



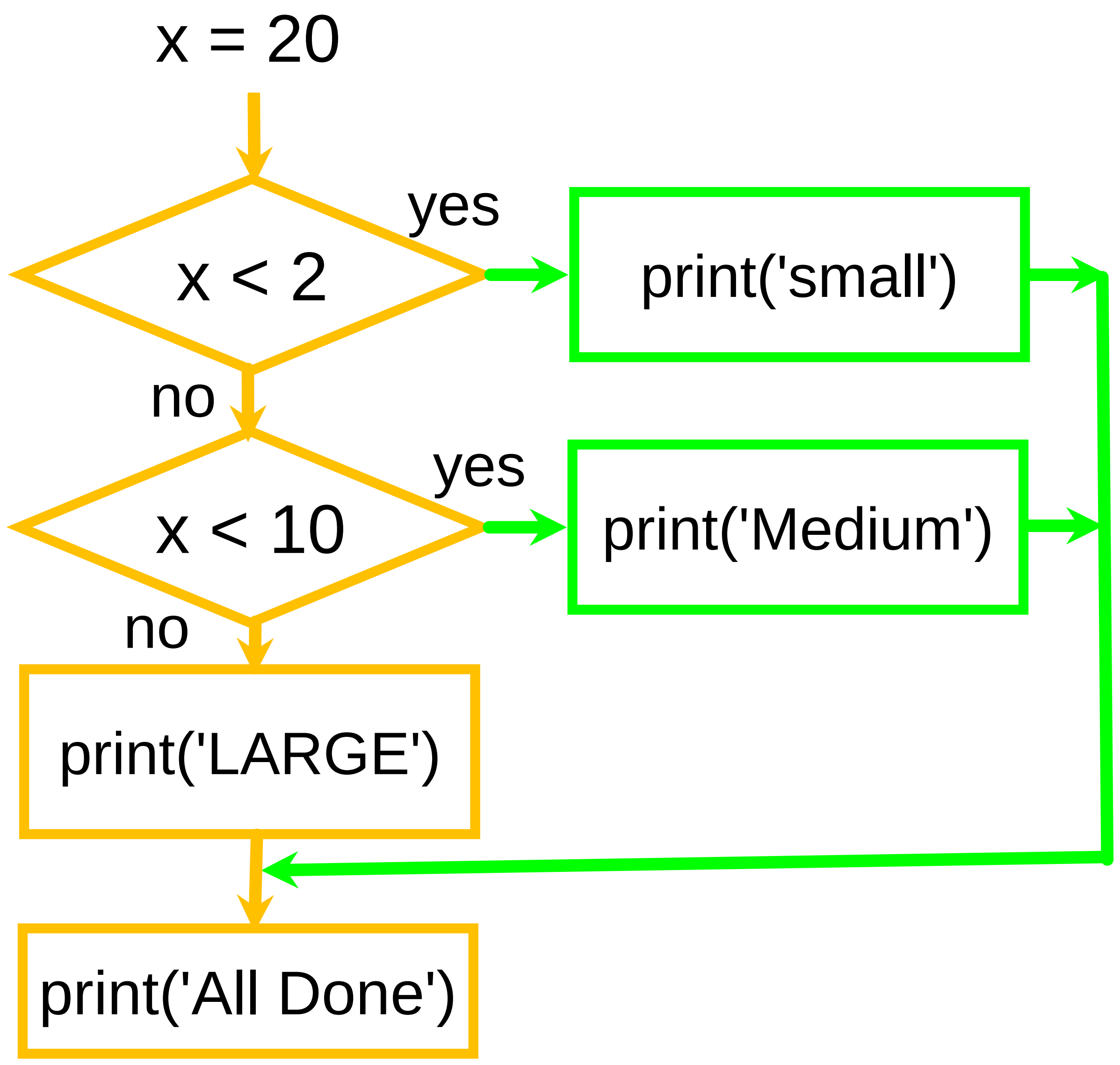
Multi-way

```
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('Ginormous')
```

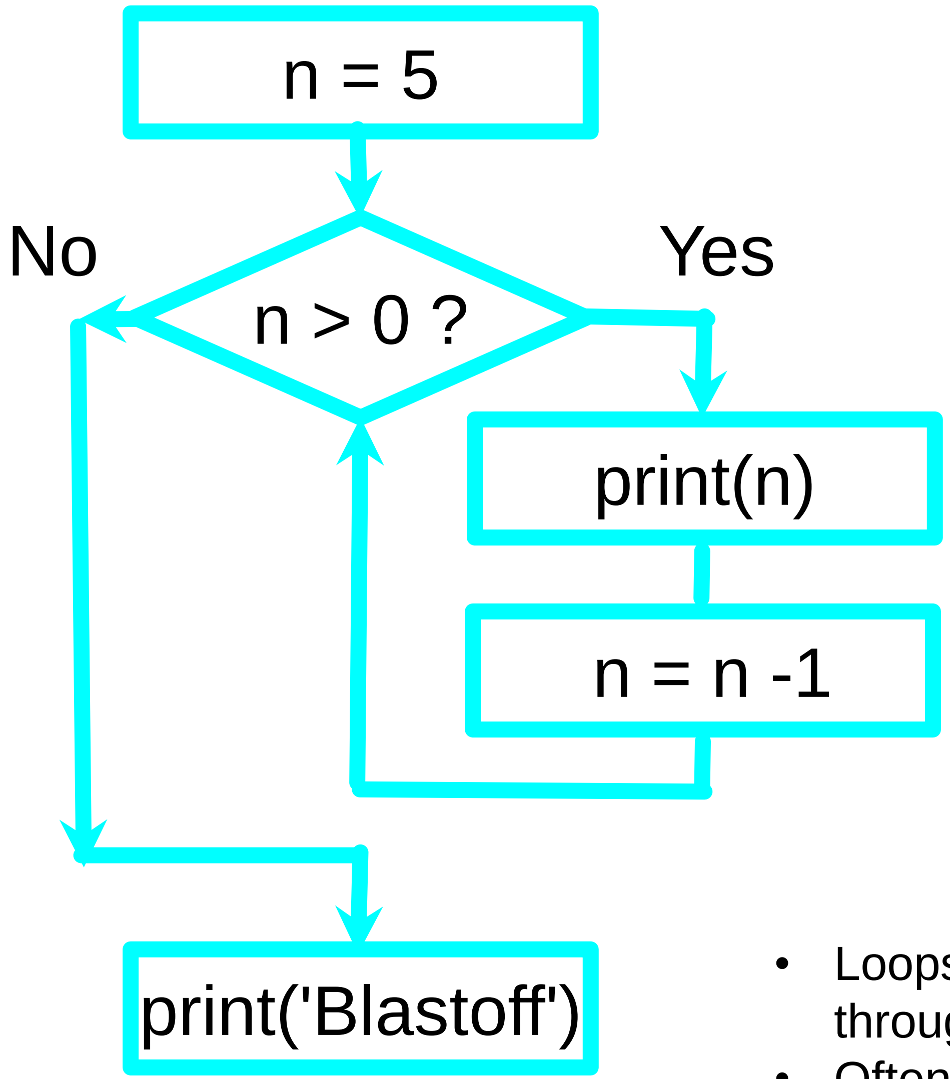
Source: Python for Everybody, Charles Severance, University of Michigan School of Information

Multi-way Puzzles

Which will never print regardless of the value for x?

```
if x < 2 :
    print('Below 2')
elif x >= 2 :
    print('Two or more')
else :
    print('Something else')
```

```
if x < 2 :
    print('Below 2')
elif x < 20 :
    print('Below 20')
elif x < 10 :
    print('Below 10')
else :
    print('Something else')
```



Repeated Steps

Program:

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

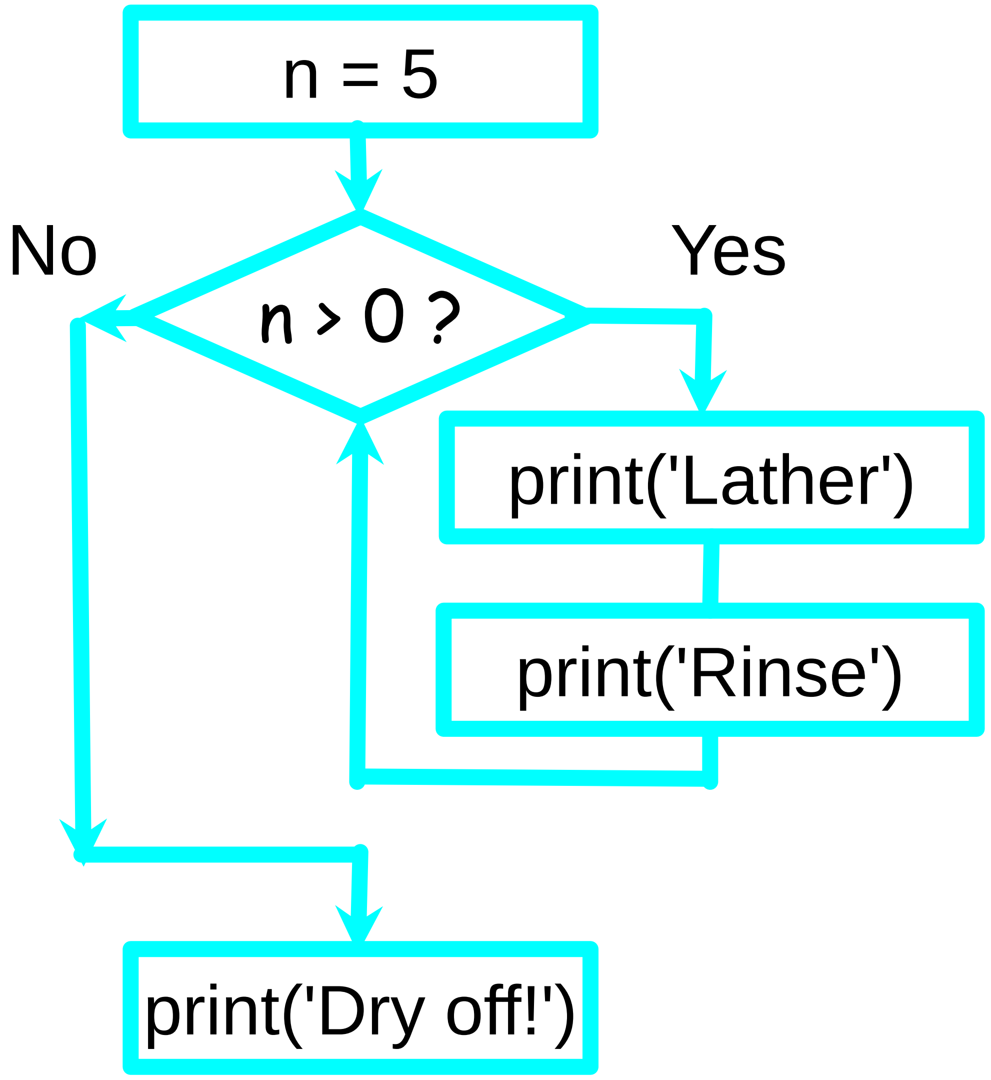
Output:

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

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

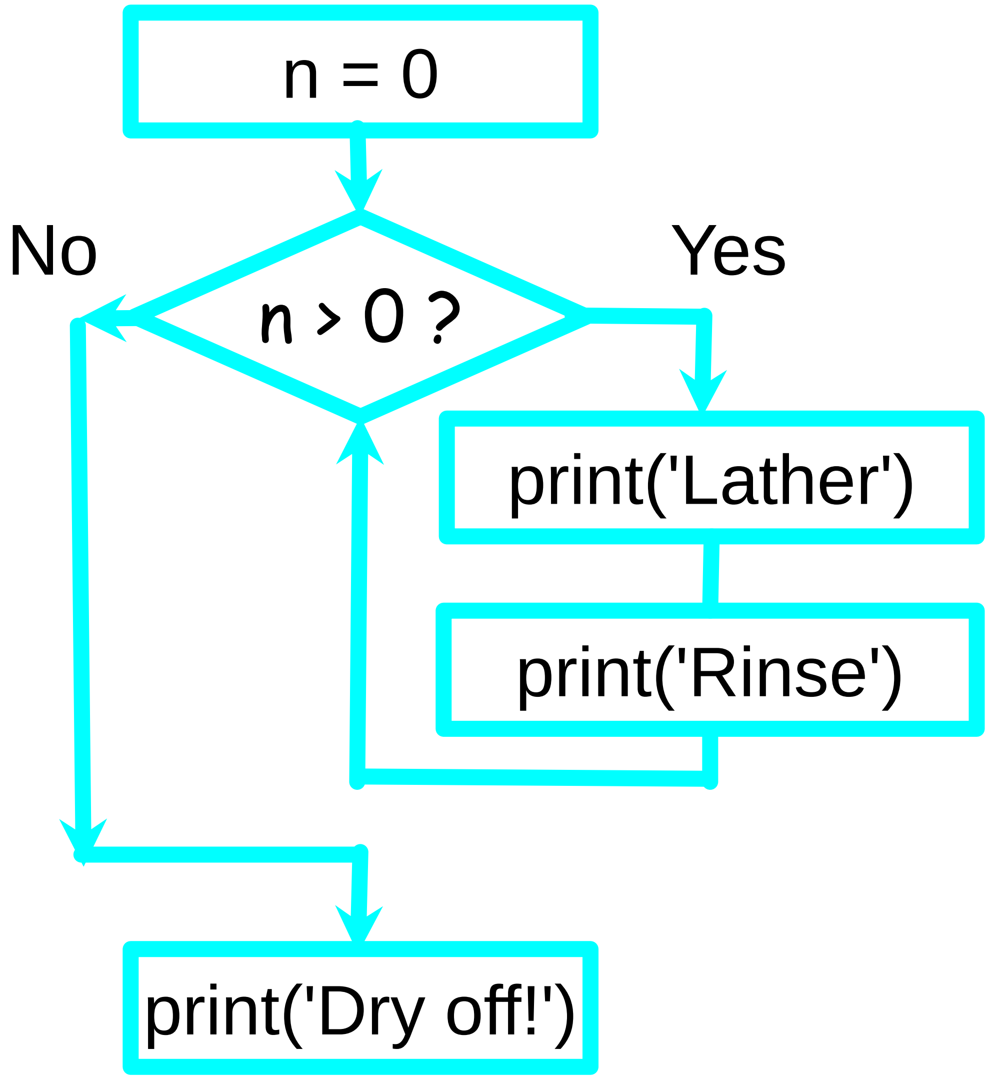
Source: Python for Everybody, Charles Severance, University of Michigan School of Information

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?

Source: Python for Everybody, Charles Severance, University of Michigan School of Information

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!')
```

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!')
```

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

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!')
```

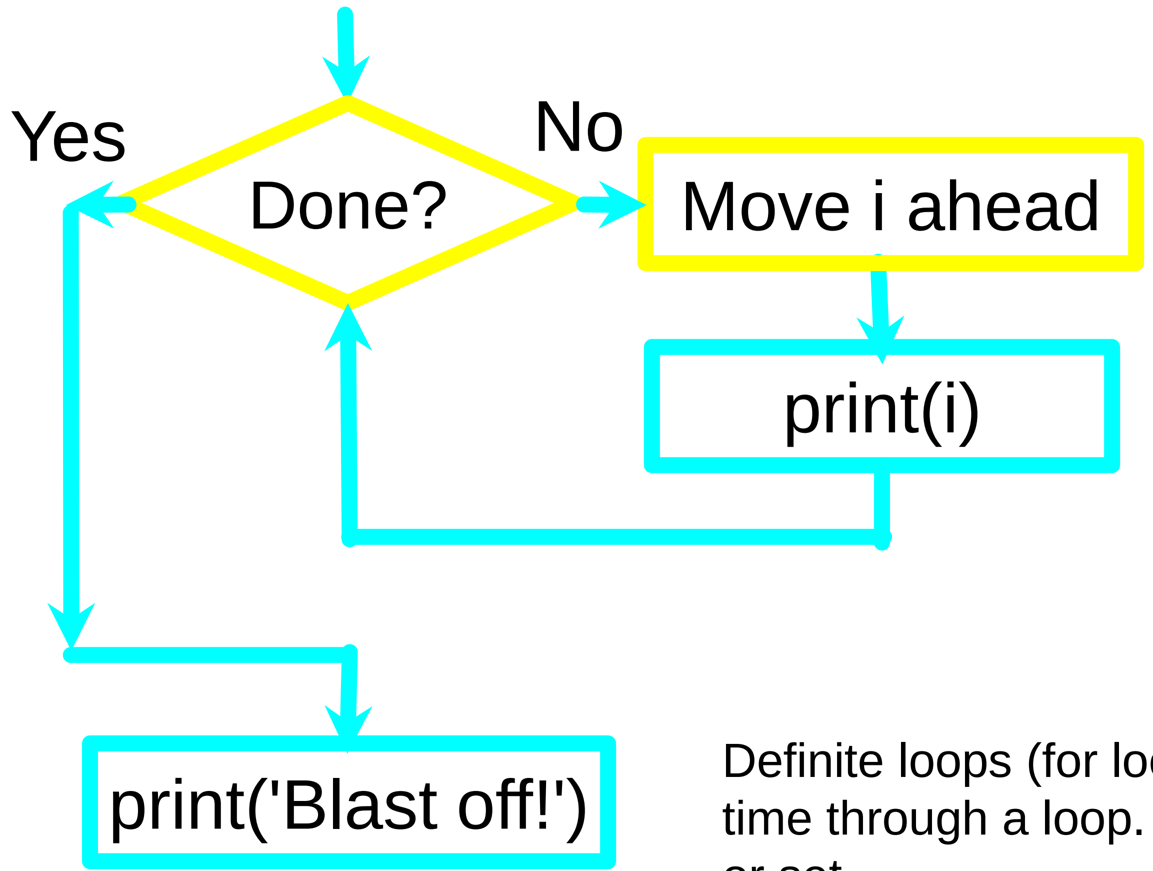
```
5
4
3
2
1
Blastoff!
```

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.

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

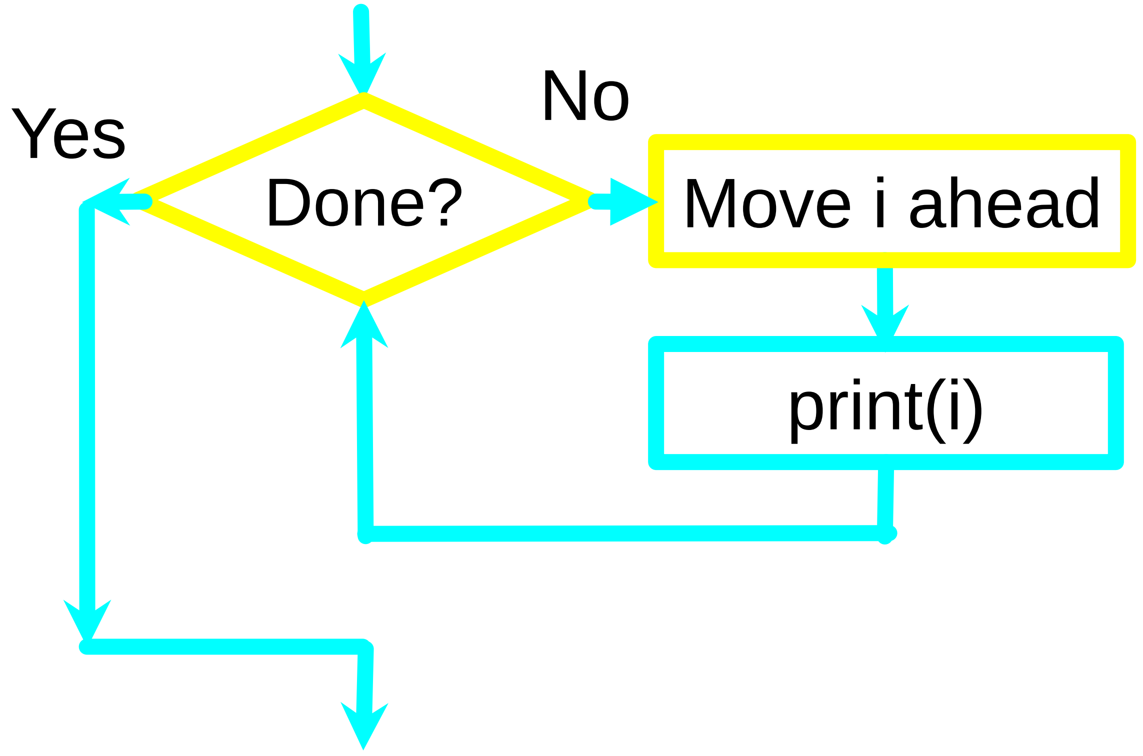
The iteration variable moves through all of the values in the sequence

Five-element sequence

Iteration variable

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

Source: Python for Everybody, Charles Severance, University of Michigan School of Information

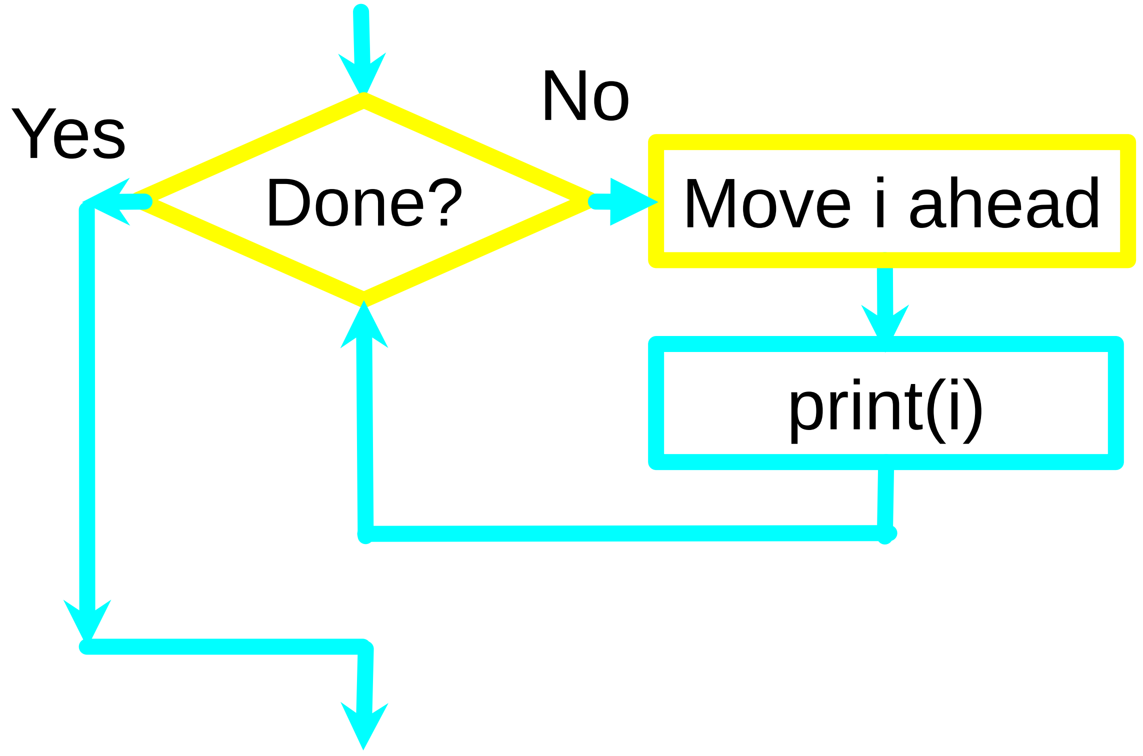


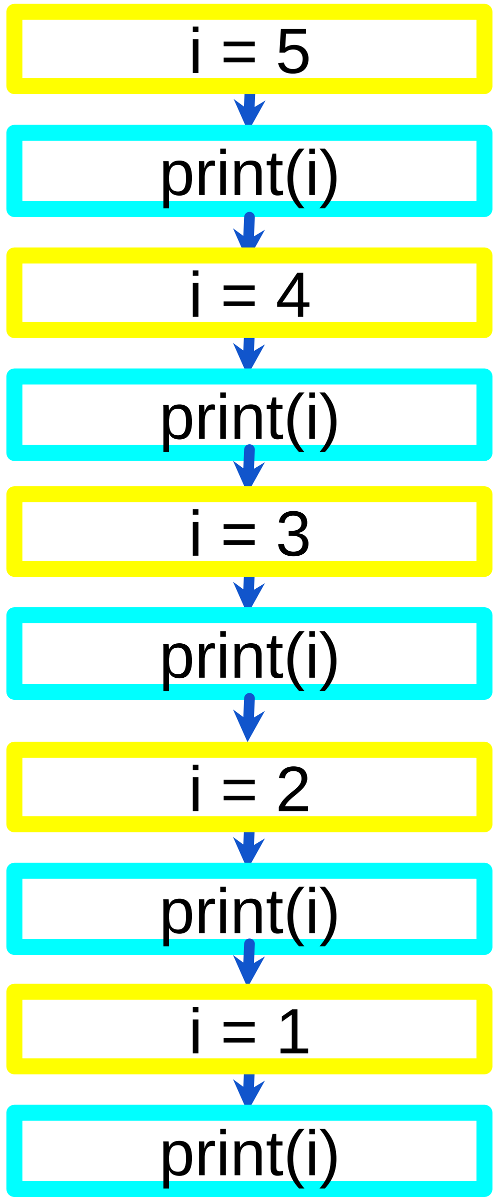
The iteration variable “iterates” through the sequence (ordered set)

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)
```





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

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
```

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.

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

```python
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?

Acknowledgements / Contributions

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

Modification: Taehee Jeong, San Jose State University