# DSCI 510
# PRINCIPLES OF PROGRAMMING FOR DATA SCIENCE

Itay Hen

*Information Sciences Institute*

USC Viterbi
School of Engineering

# CONDITIONAL EXECUTION

# Conditional Execution

```
if x == 5:
    print('x is definitely 5')
print('x may or may not be 5')
```

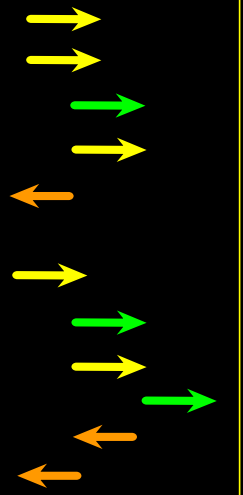
- Note the colon!
- Note the indentation
- The last line will be executed regardless of the value of $x$.

# 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

increase / maintain after if or for

decrease to indicate end of block



```
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')
```

4 spaces 4 spaces

# Think About 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')
```

# Else

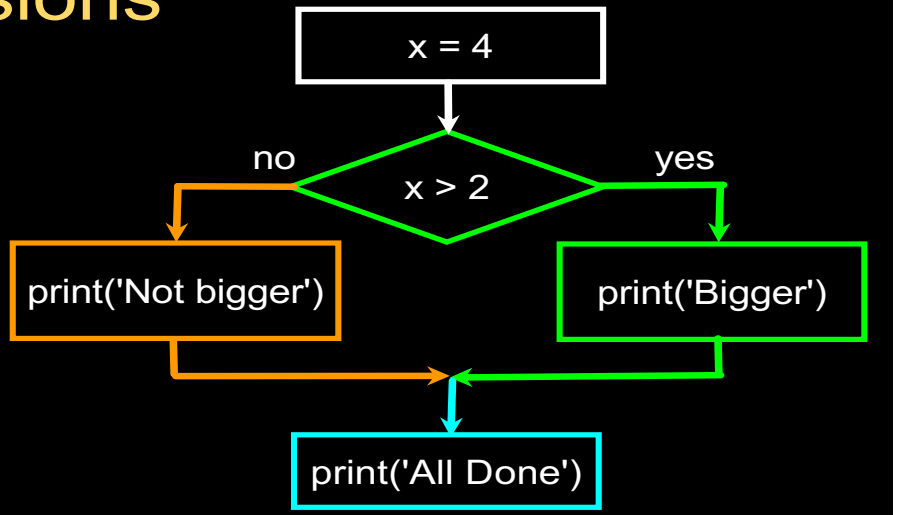
- With an *else* statement, only one block will be executed, either the *if* block or the *else* block

```
if age >= 18:
    print('you can vote!')
else:
    print("you're too young to vote!")
```

- (Note the use of double quotes. Why?)
- Same indentation rules apply to *else* as to *if*
- And note the colon again !

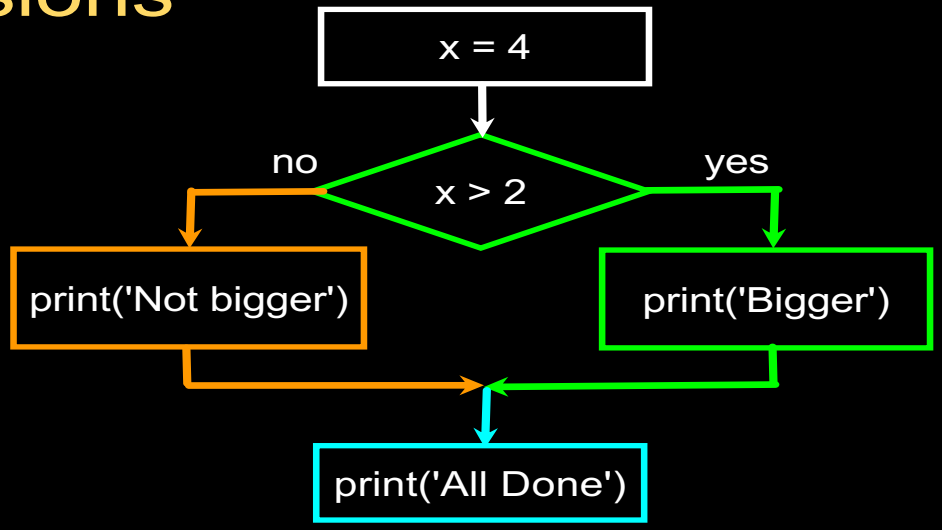
# Two-way Decisions

- 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



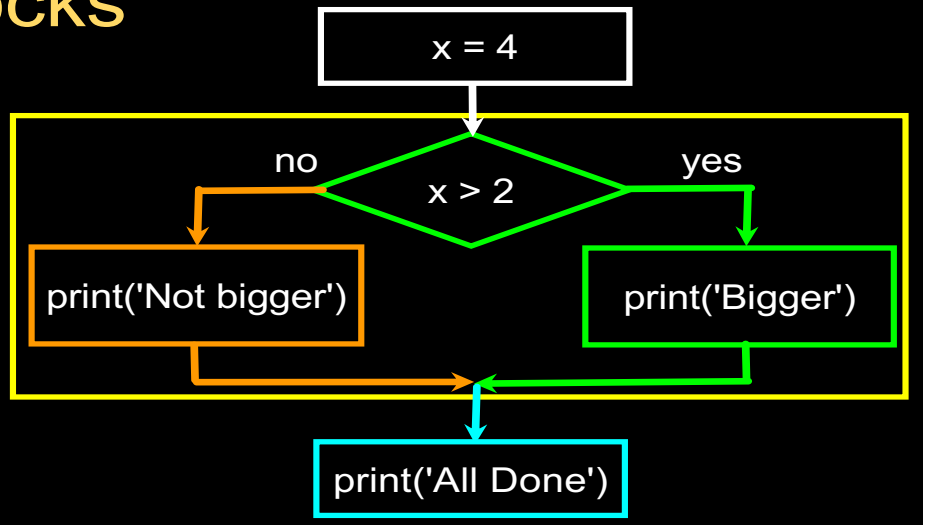
# Two-way Decisions with else:

```
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')
```

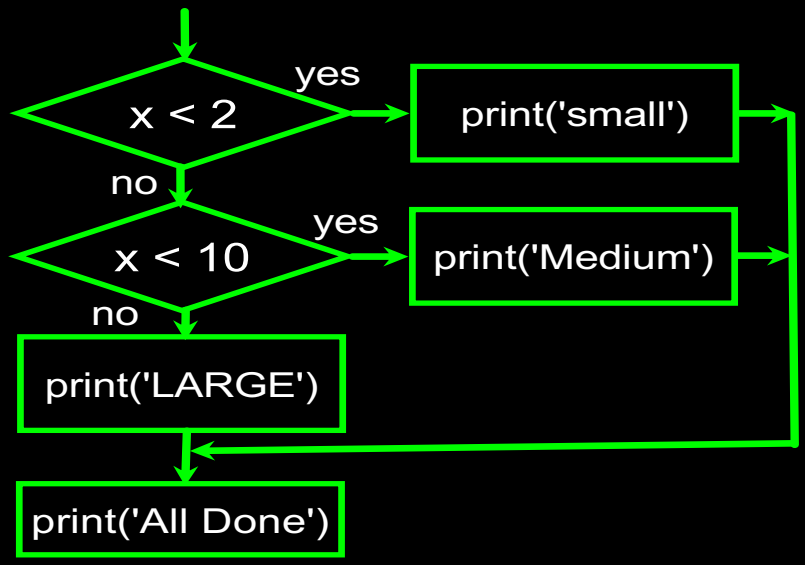


# Chained Conditionals - *elif*

- *elif* means “else if”
- If this is true <do something>,
  otherwise, if this is true <do something else>
  otherwise, if this is true <do something else>
  . . .
- An *else* at the end will catch the case where all the other ones above were false

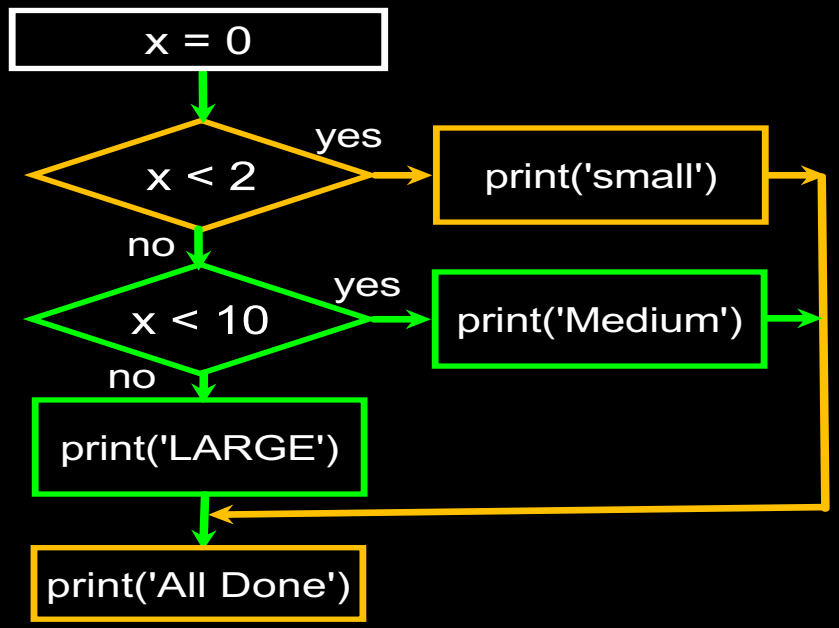
# Multi-way

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



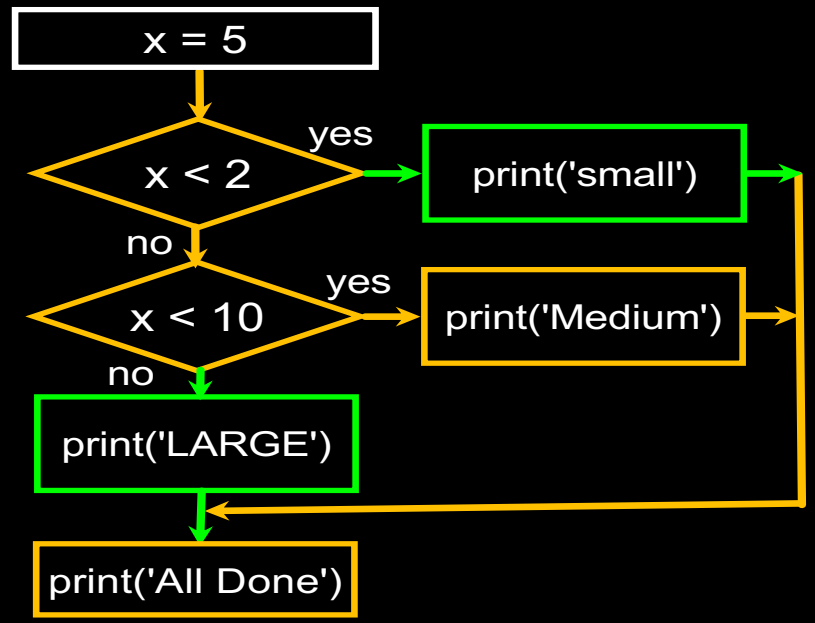
# Multi-way

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



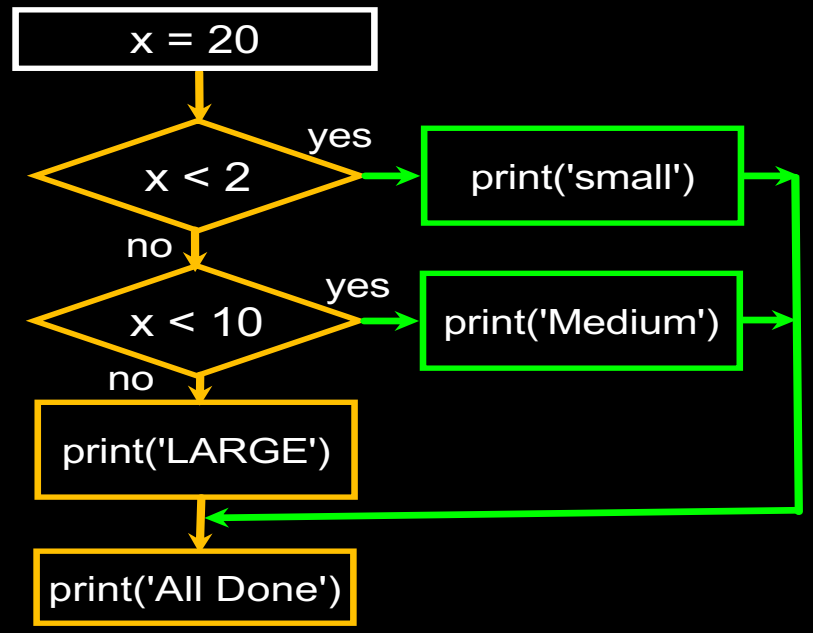
# Multi-way

```
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')
```



# Can end with Else, or Not
# Good practice to end with else

## 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')
```

# 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')
```

'Something else' never prints!

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

'Below 10' never prints!


# What will be printed?

**Consider this Python program:**

```
n = 15
if n % 2 == 0:
    print(n, 'is a multiple of 2')
elif n % 3 == 0:
    print(n, 'is a multiple of 3')
elif n % 5 == 0:
    print(n, 'is a multiple of 5')
```

**What will be printed?**


# What will be printed?

**Consider this Python program:**

```
n = 15
if n % 2 == 0:
    print(n, 'is a multiple of 2')
elif n % 3 == 0:
    print(n, 'is a multiple of 3')
elif n % 5 == 0:
    print(n, 'is a multiple of 5')
```

**What will be printed?**

```
15 is a multiple of 3
```

only this one


# What will be printed?

**But maybe this was really intended:**

```
n = 15
if n % 2 == 0:
    print(n, 'is a multiple of 2')
if n % 3 == 0:
    print(n, 'is a multiple of 3')
if n % 5 == 0:
    print(n, 'is a multiple of 5')
```

*elif* replaced by *if*

*elif* replaced by *if*

**What will be printed?**

```
15 is a multiple of 3
15 is a multiple of 5
```


*What could possibly go wrong?*

## What will be printed?

**Consider this Python program:**

```
points = 0
score = 100
if score >= 80:
    points = points + 5
elif score >= 90:
    points = points + 10
    print(points, 'points')
```

**What will be printed?**


*What could possibly go wrong?*

## What will be printed?

**Consider this Python program:**

```
points = 0
score = 100
if score >= 80:
    points = points + 5
elif score >= 90:
    points = points + 10
    print(points, 'points')
```

**What will be printed?**

**Nothing** will ever be printed for any score,
because the ***elif*** block will never be executed.
At the end, there will be 5 points, in silence.


*What could possibly go wrong?*

**What will be printed?**

**But maybe this was really intended:**

```
points = 0
score = 100
if score >= 90:
    points = points + 10
elif score >= 80:
    points = points + 5
print(points, 'points')
```

Stricter condition comes first.

*print(...)* no longer indented.

**What will be printed?**

```
10 points
```

# EXCEPTIONS

# try/except

- Sometimes things do not go according to plan
- For example, you may get input you don't expect

```
# Convert Fahrenheit to Celsius
f_temp = input('Temp in Fahrenheit ')
f_temp = float(f_temp)
c_temp = (f_temp - 32.0) * 5.0 / 9.0
print (f_temp, 'Fahrenheit is', c_temp, 'Celsius')
```

- What if someone types something other than a number?
- Let's check it out

# try/except

```
try:
    <a block of code>
except:
    <a block of code that will be
     executed only if there was an
     error in the block above>
```

- *try/except* blocks are in some sense similar to *if/else* blocks
- A statement in the “try” block is said to *throw* the exception, and the “except” block *catches* it
- **KEEP *try* BLOCKS SMALL!**

# LISTS

# What is a List?

- A list is a built-in data structure in Python
- Used to store multiple items in a single variable
- Lists are ordered, mutable, and allow duplicates
- Defined using square brackets: [ ]

```
# 1. Empty list
empty_list = []

# 2. List of integers
numbers = [1, 2, 3, 4, 5]

# 3. List of strings
fruits = ["apple", "banana", "cherry"]
```

```
# 4. Mixed data types
mixed = [1, "hello", 3.14, True]

# 5. Nested lists (list inside a list)
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# 6. List of booleans
flags = [True, False, True, False]
```

# Looking Inside Lists

(Like with strings), we can get at any single element in a list using an index specified in **square brackets**

- Lists are indexed starting at 0
- len(my_list) → number of elements
- sum(numbers) → adds up numeric elements
- 'apple' in my_list → checks membership

| Joseph | Glenn | Sally |
|---|---|---|
| [0] | [1] | [2] |

```
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(friends[1])
Glenn
>>>
```

# List Slicing

- *Slicing* allows extracting a portion of a list
- Syntax: list[start:end:step] → returns a (sub)list
    - start: index where slice begins (default = 0)
    - end: index where slice stops (exclusive)
    - step: interval between elements (default = 1)

# List Slicing

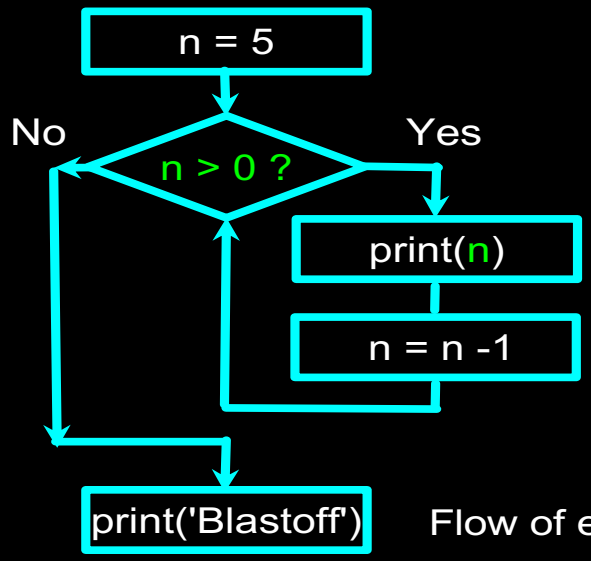
```
# Example list
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# 1. Basic slicing
print(numbers[2:6])    # [2, 3, 4, 5] → from index 2 up to (not including) 6

# 2. Omitting start or end
print(numbers[:5])     # [0, 1, 2, 3, 4] → from start to index 4
print(numbers[5:])     # [5, 6, 7, 8, 9] → from index 5 to end
```

# List Slicing

```
# 3. Using step
print(numbers[::2])    # [0, 2, 4, 6, 8] → every 2nd element
print(numbers[1::3])   # [1, 4, 7] → start at index 1, step 3

# 4. Negative indices
print(numbers[-4:])    # [6, 7, 8, 9] → last 4 elements
print(numbers[:-3])    # [0, 1, 2, 3, 4, 5, 6] → everything except last 3

# 5. Reversing with step
print(numbers[::-1])   # [9, 8, 7, 6, 5, 4, 3, 2, 1, 0] → reverse the list

# 6. Slice assignment (modifying part of a list)
numbers[2:5] = [20, 30, 40]
print(numbers)         # [0, 1, 20, 30, 40, 5, 6, 7, 8, 9]
```

# ITERATION, LOOPS

# Iteration: While loop




Program:

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

Output:

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

Flow of execution for a *while statement*:

1. Evaluate the condition, yielding True or False.
2. If the condition is *False*, exit the while statement and continue execution at the next statement.
3. If the condition is *True*, execute the body and then go back to step 1.

# A Simple while Loop

pythontutor.com

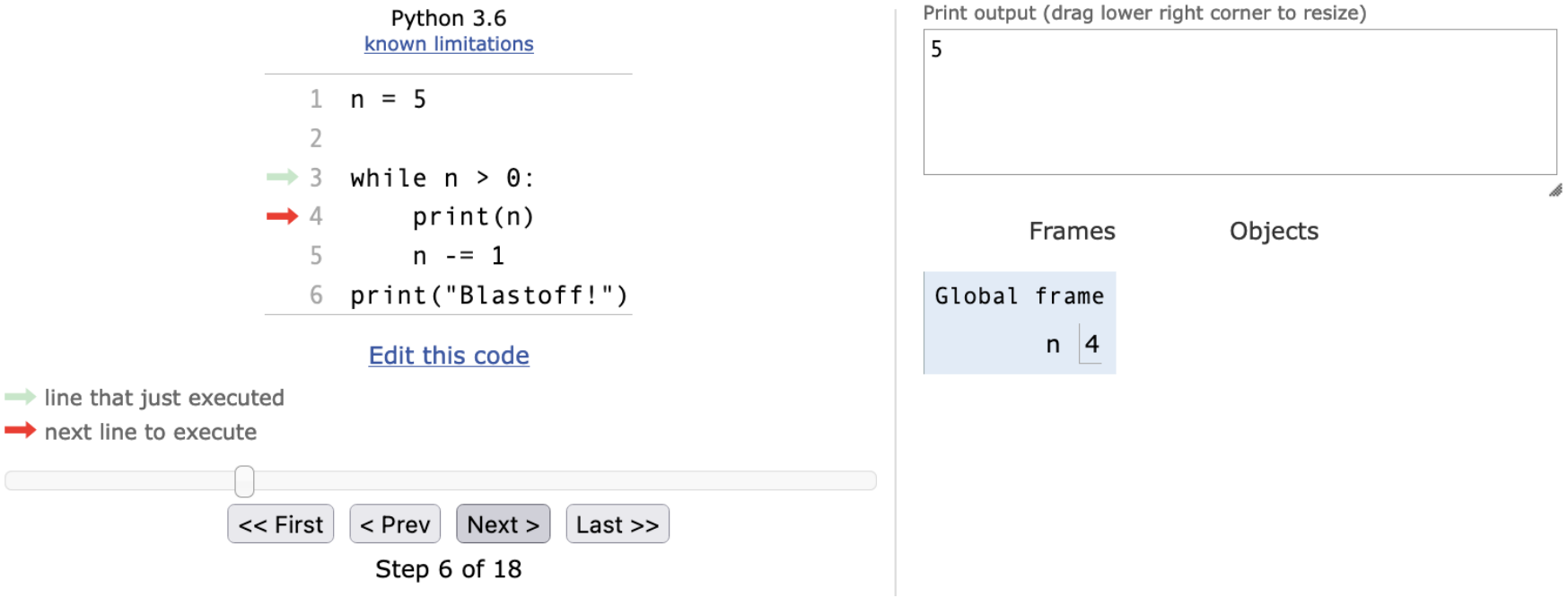
# A Simple while Loop

# A Simple while Loop

# A Simple while Loop

# A Simple while Loop

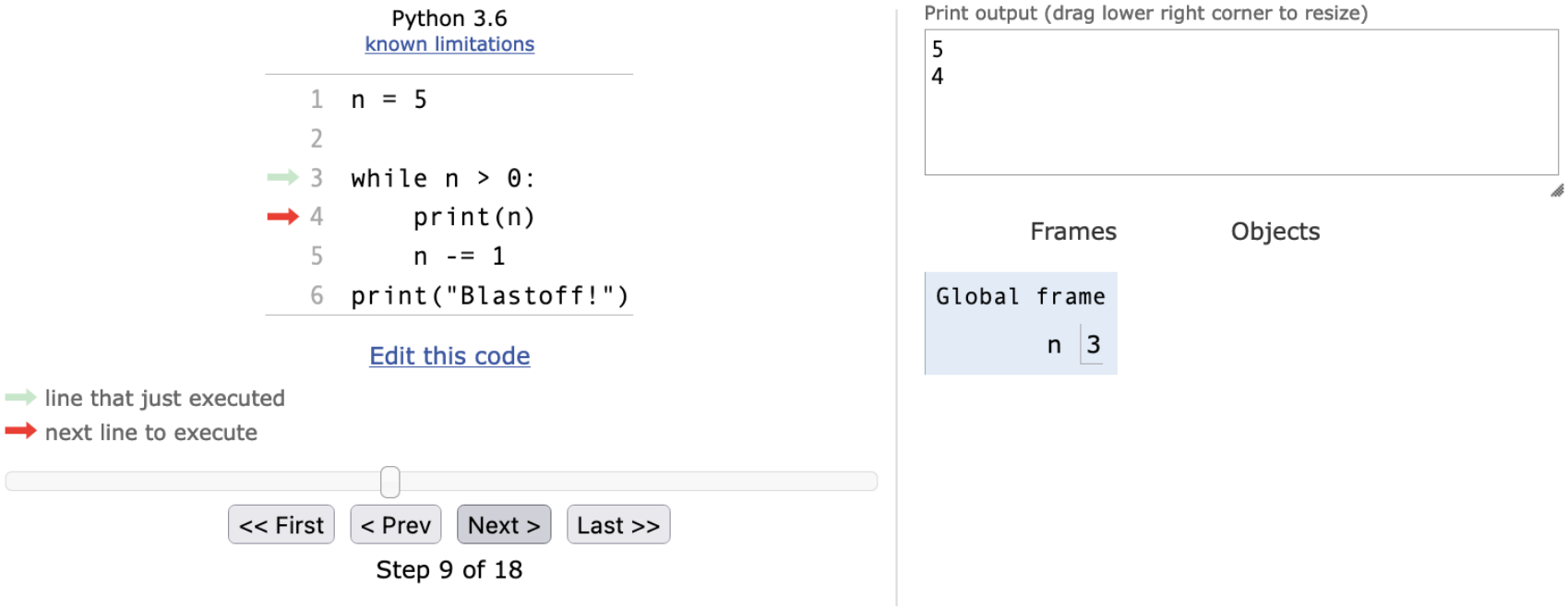
# A Simple while Loop



# A Simple while Loop

# A Simple while Loop

# A Simple while Loop

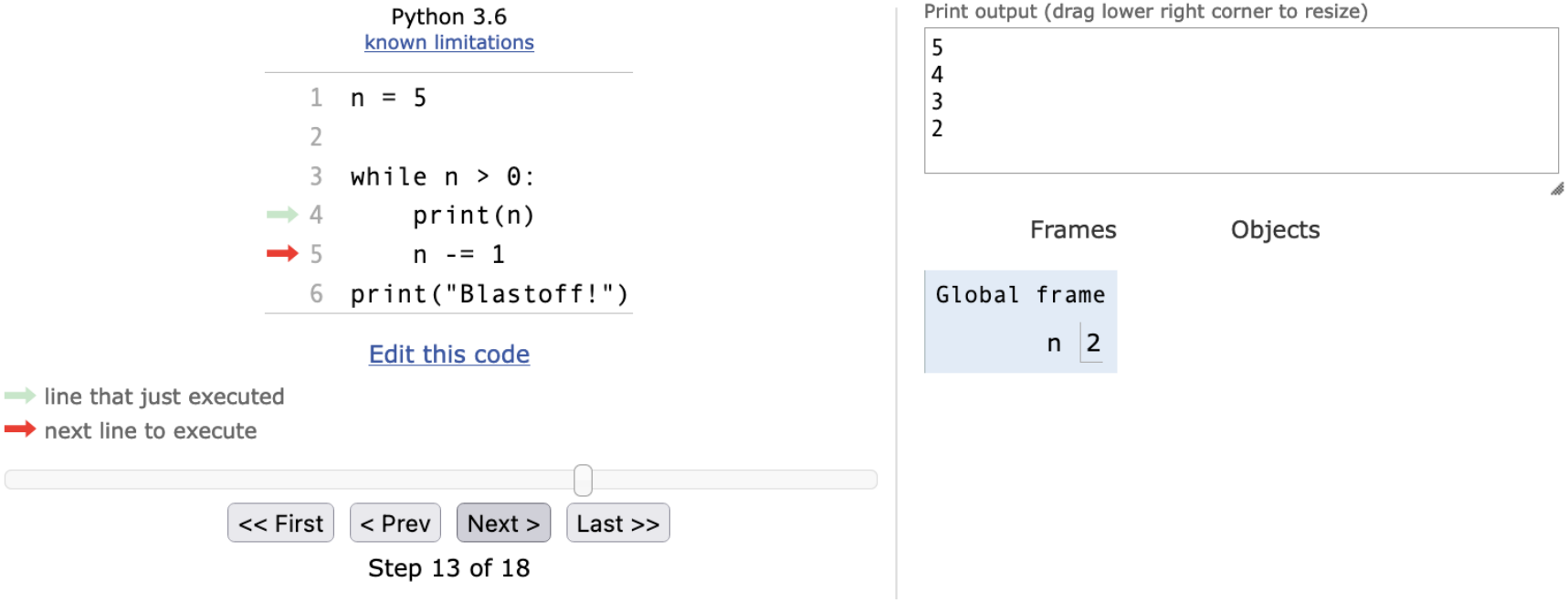


# A Simple while Loop

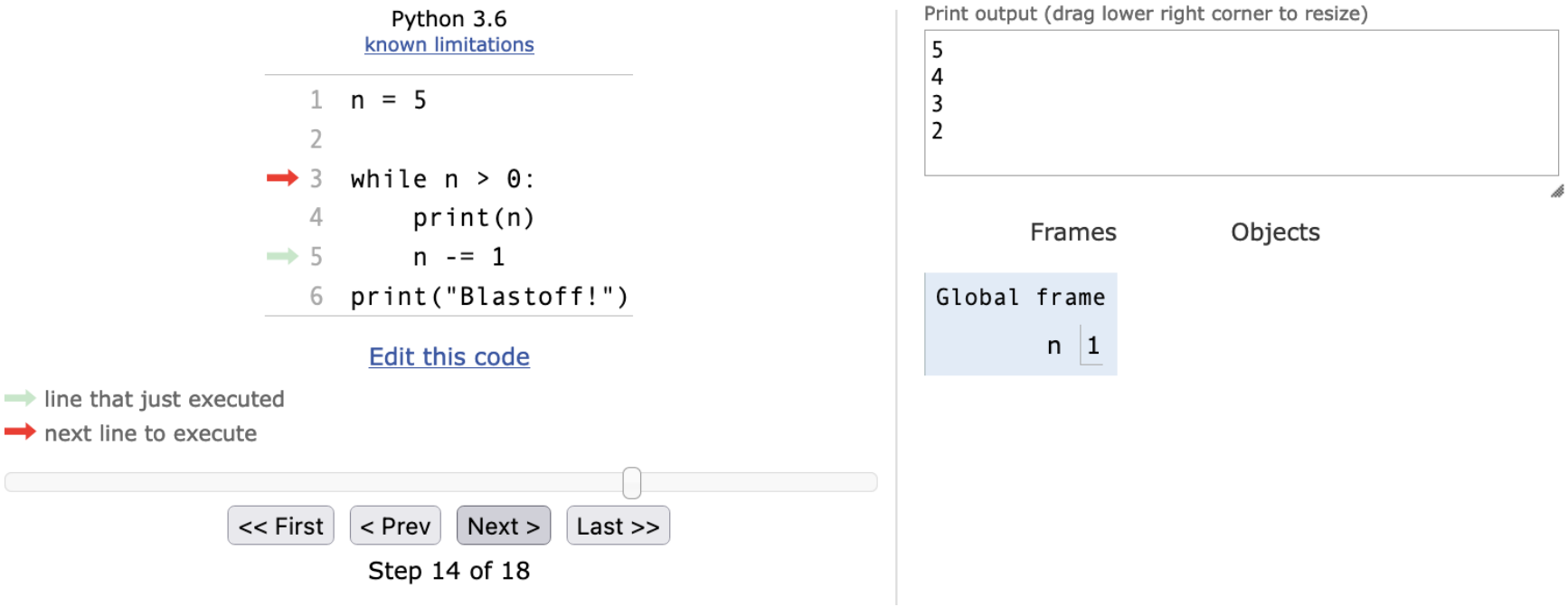
# A Simple while Loop

# A Simple while Loop

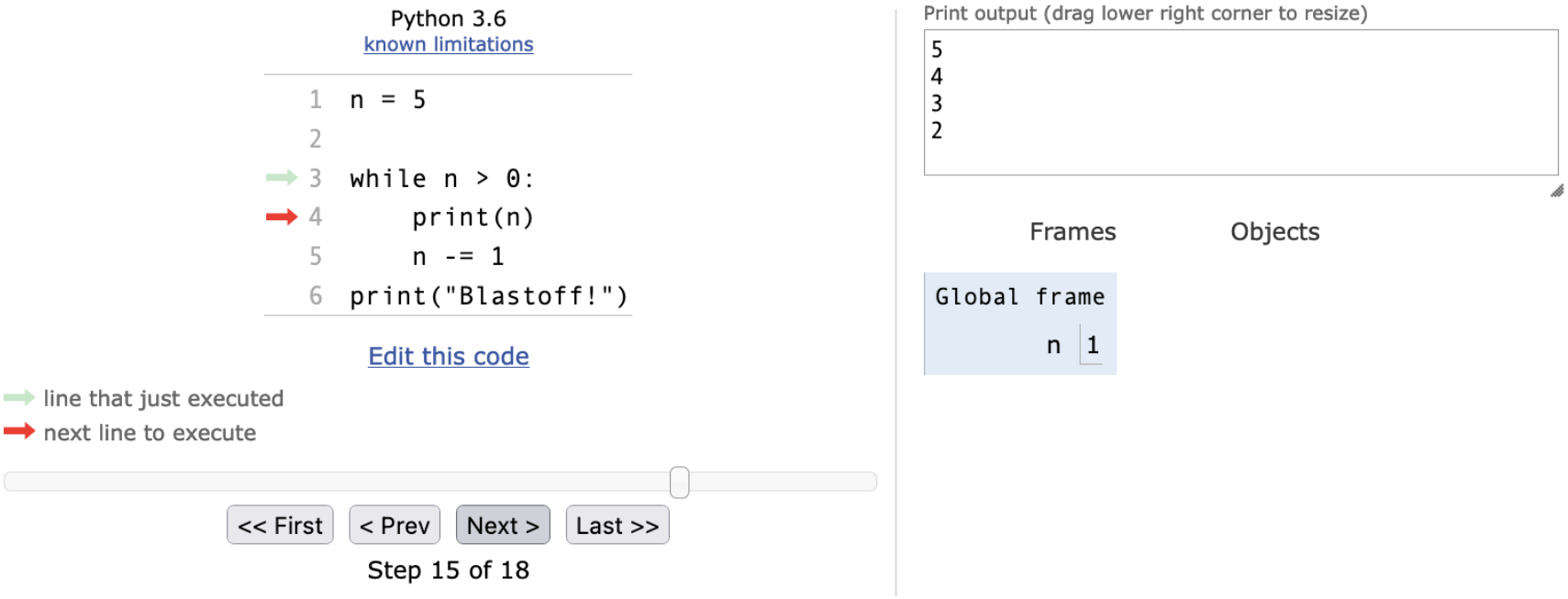
# A Simple while Loop



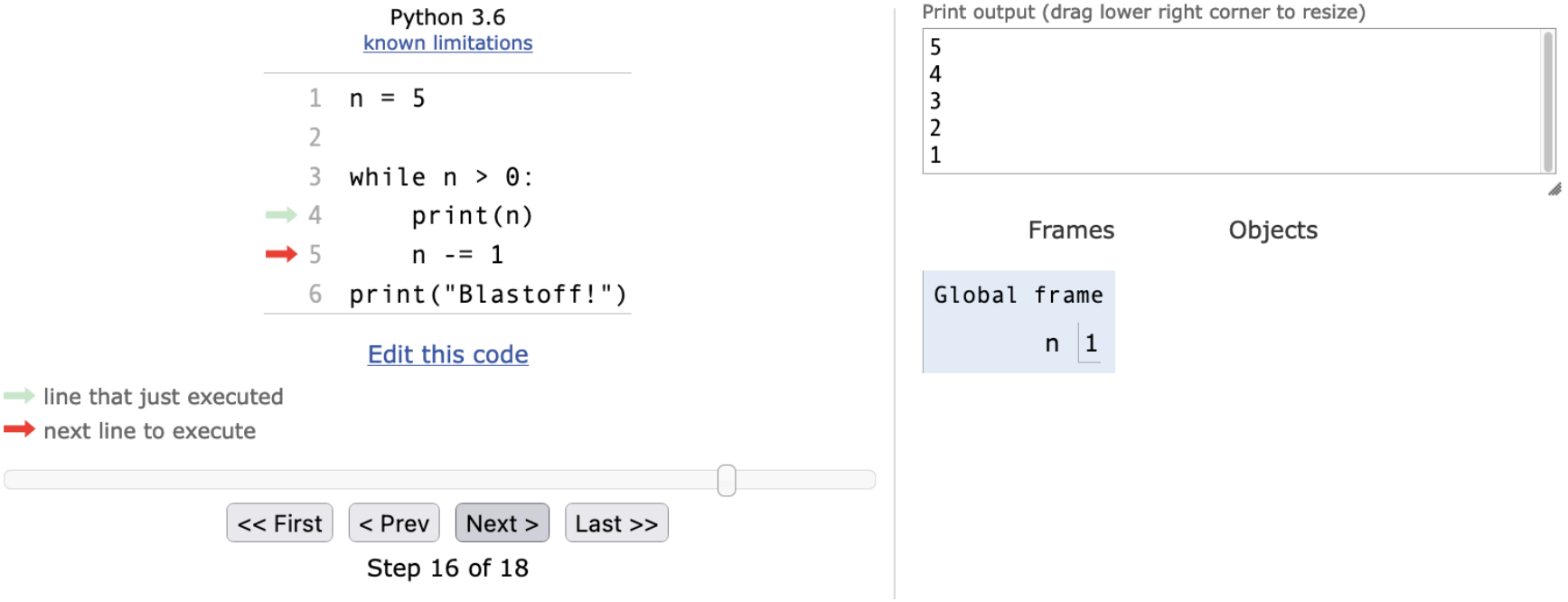
# A Simple while Loop



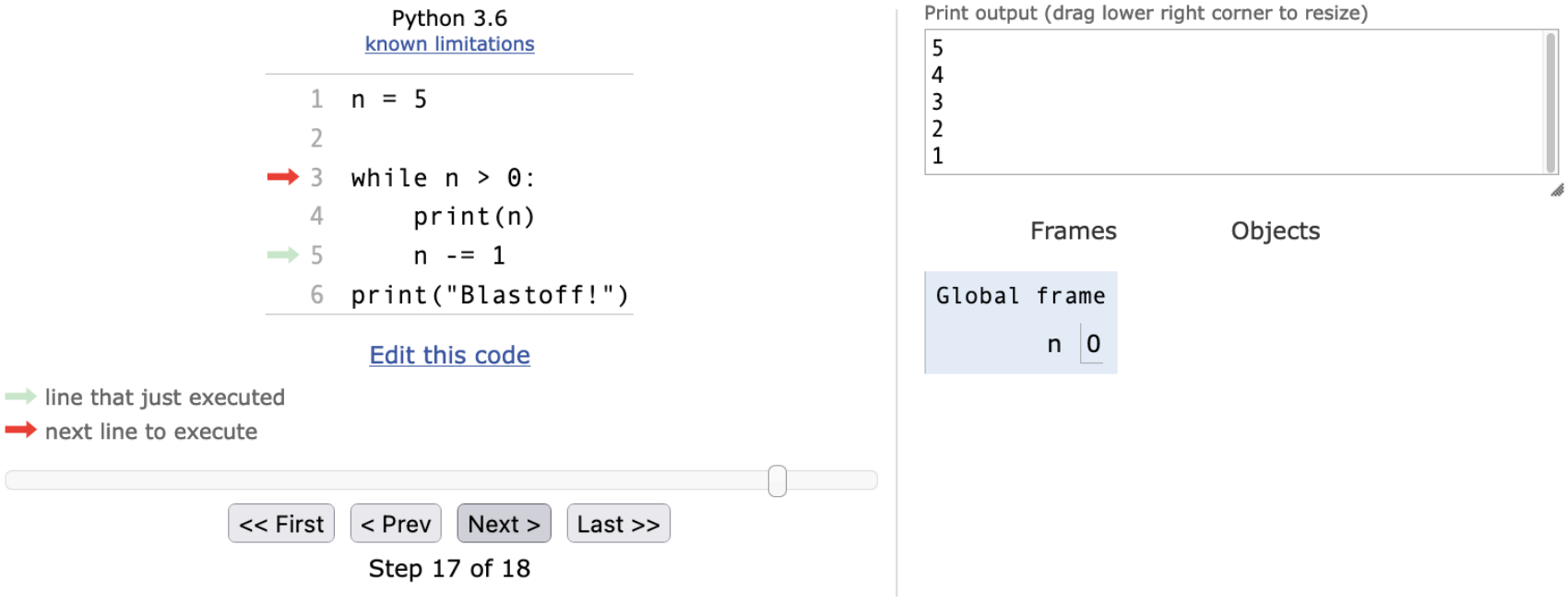
# A Simple while Loop



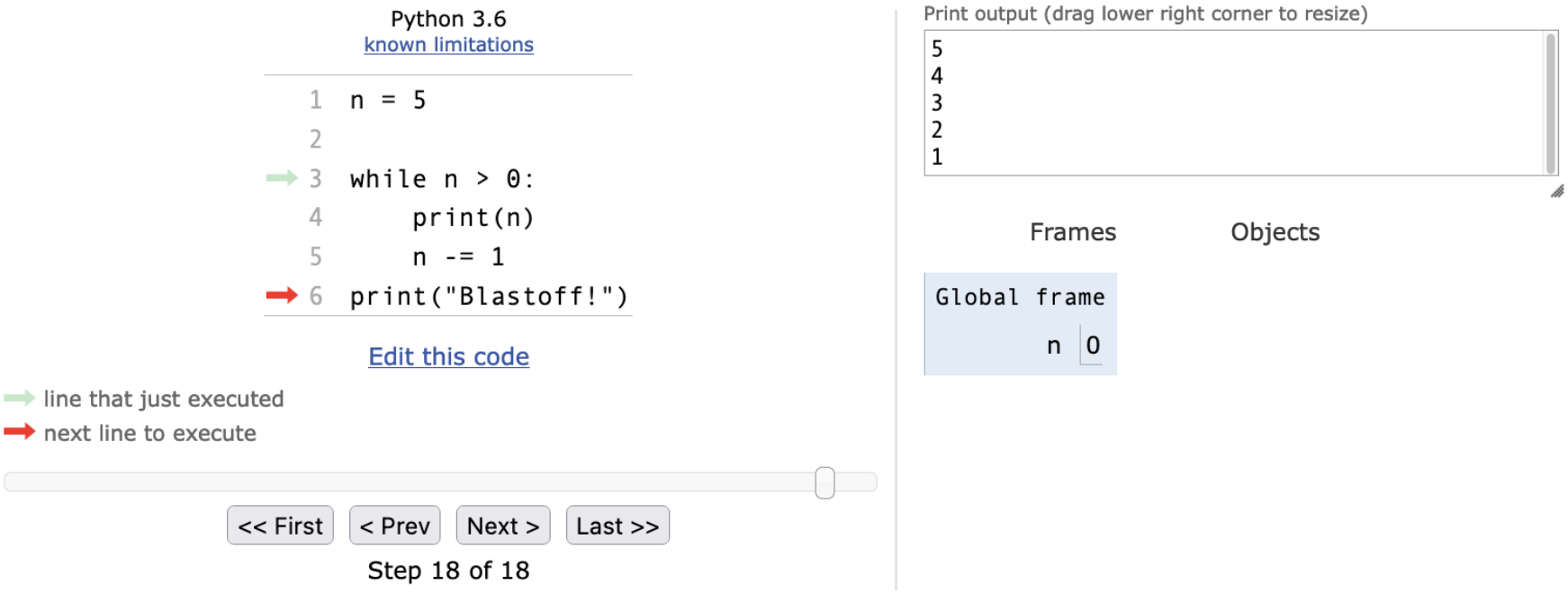
# A Simple while Loop



# A Simple while Loop

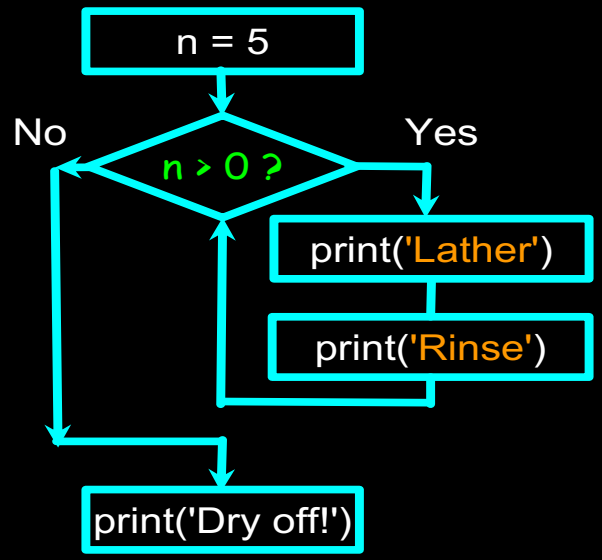


# A Simple while Loop



# A Simple while Loop

# Infinite Loop




# An Infinite Loop

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

What is wrong with this loop?

# break

- break is used within loops to terminate the loop prematurely when a certain condition is met.
- It allows you to exit the loop even if loop's condition is still true.
- It is often used when you want to stop the loop as soon as a specific condition is satisfied.

# Breaking out of a Loop

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

- 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

# Infinite loop with **break**

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

```
> hi
hi
> how
how
> do
do
> i
i
> exit
exit
> done
Done!
```

- Captures user input
- Checks if user typed ‘done’
- If not, print the user input
- If user type ‘done’ break out of loop

# continue

- continue is used within loops to skip the current iteration and move to the next iteration of the loop.
- It is often used when you want to skip specific items or conditions within a loop and continue processing the remaining items.

# Finishing an iteration with **continue**

https://pythontutor.com

```
num = 0
while num < 12:
    num += 1
    if num % 2 == 0:
        continue
    print(f"Processing {num}")
```

- Iterate through numbers 1 to 11
- If the number is an even number, continue to the next iteration
- If the number is an odd number, print "Processing {num}"

```
Processing 1
Processing 3
Processing 5
Processing 7
Processing 9
Processing 11
```

# What will be printed ?


```
num = 0
while num < 12:
    if num % 2 == 0:
        continue
    print(f"Processing {num}")
    num += 1
```

This is an infinite loop because
num is initialized to 0 and
the condition num % 2 == 0 will cause the loop
to go to next iteration without incrementing
num.

# while: indefinite loop

- A while loop is an *indefinite* loop, because it keeps on iterating an indefinite number of times, until its logical condition becomes False.
- Sometimes we want a loop to iterate a specific number of times