CSCI 141

Lecture 11:
More turtles, `for` loops and the `range` function

Special Announcements from Merril Hunt-Paez

Special Announcements from Merril Hunt-Paez

AWC Website: https://wwu-awc.github.io/

Contact AWC: awc.wwu@gmail.com

Contact Merril: huntpam@wwu.edu

Happenings

Tuesday, 4/30 – ACM Hackathon Presentations & Recap

– 5 pm in CF 316

Tuesday, 4/30 – AIA Presents: Intro to SQL and Databases

– 6 pm in PH 228

Wednesday, 5/1 -- Peer Lecture Series: GDB Workshop

– 5 pm in CF 162

Announcements

Announcements

- Exam is next Friday

Announcements

- Exam is next Friday
  - 50 minutes

Announcements

- Exam is next Friday
  - 50 minutes
  - Closed-book; no notes

Announcements

- Exam is next Friday
  - 50 minutes
  - Closed-book; no notes
  - No calculators (there won’t be any hard arithmetic)

Sample Exam Questions

- Submit one sample exam question, along with its solution to Canvas by 1pm Monday
  - Worth 1% extra credit on midterm exam.
  - I will post sample questions and solutions by Monday night.
  - I will choose one question to include on the exam.
  - Canvas assignment with more detailed instructions will go up today.

Study Tips

Reading is not enough: **solve problems**.

- **Goals** slides: can you do these things? Try and see.
- **Terminology**: be able to discuss the meaning of all words that appear in blue in the slides
- **ABCD questions**: solve it before looking at the answer (if provided)
- **Demo code**: solve the same problem without without looking at my code.
- **Homework questions**: understand what you got wrong and why. Understand what you got right and why.
- **Exercises** from the eBook

Goals

- Know how to use `import` statements to get access to modules containing functions that other people have written.
- Understand how to create a Turtle object and call its methods to move it around the screen and draw simple shapes.
    - Methods: `forward, left, right, penup, pendown`
- Know the syntax and behavior of the `for` statement (`for` loop)
- Know how to use the `range` function in the header of a `for` loop.

Last time: Modules

The Python Standard Library is a collection of modules containing many more functions.

To use functions in a module, you need to import the module using an `import` statement:

```
import module
```

By convention, we put all import statements at the **top** of programs.

Last time: Modules

The Python Standard Library is a collection of modules containing many more functions.

To use functions in a module, you need to import the module using an `import` statement:

```
import module
```

(replace the in `this font` with the specific module name)

By convention, we put all import statements at the **top** of programs.

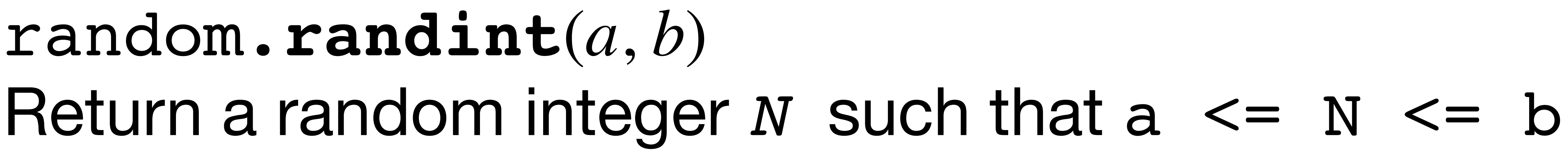
Last time: Modules

Once you've imported a module:

```
import random
```

you can call functions in that module using the following syntax:

```
random.randint(0,10)
```



`random.randint`(a, b)
Return a random integer N such that `a <= N <= b`

Last time: Modules

Once you’ve imported a module:

```
import random
```

you can call functions in that module using the following syntax:

```
random.randint(0,10)
```

Module name

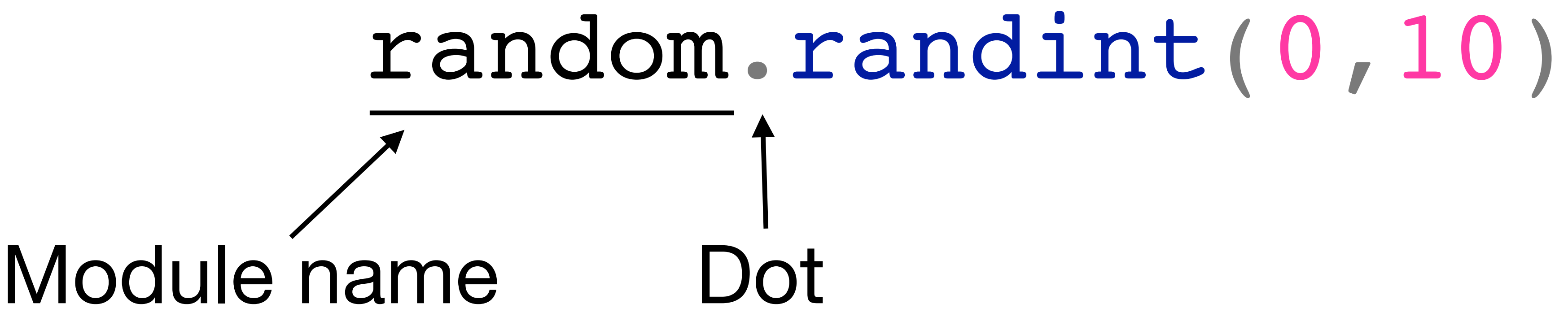
`random.`**`randint`**(a, b)
Return a random integer N such that `a <= N <= b`

Last time: Modules

Once you’ve imported a module:

```
import random
```

you can call functions in that module using the following syntax:



```
random.randint(0,10)
```

Module name

Dot

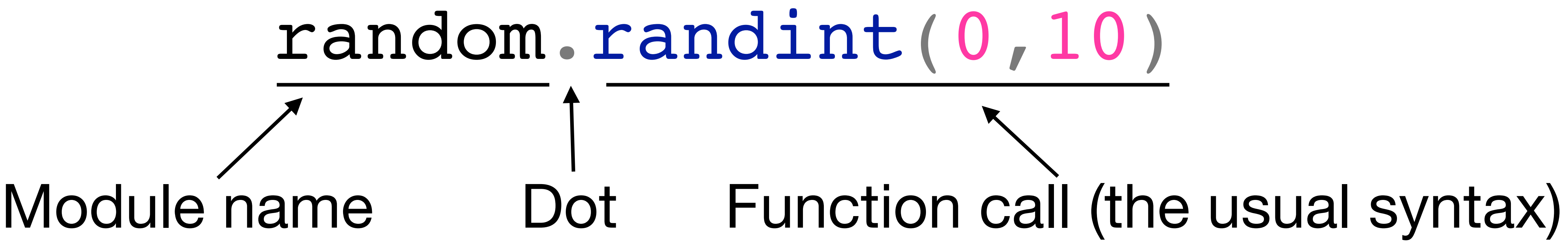
`random.`**`randint`**(a, b)
Return a random integer N such that `a <= N <= b`

Last time: Modules

Once you've imported a module:

```
import random
```

you can call functions in that module using the following syntax:



```
random.randint(0,10)
```

Module name

Dot

Function call (the usual syntax)

`random.`**`randint`**(a, b)

Return a random integer N such that `a <= N <= b`

More on import statements

Import the entire module:

```
import random
num = random.randint(1, 10)
```

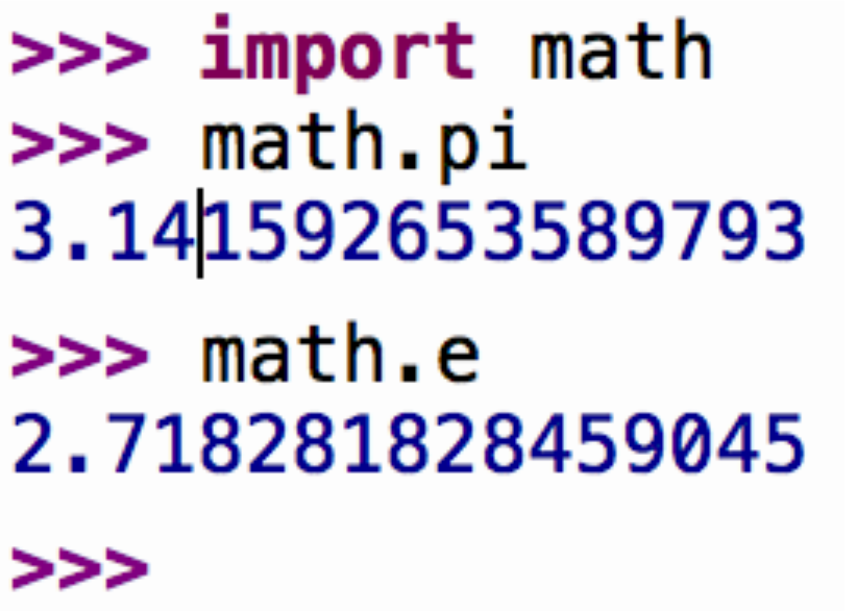
Import a specific function:

```
from math import sin
sin0 = sin(0)
```

- Don’t need module name dot notation
- Other `math` methods are not accessible:
    - `math.sqrt(4)` will throw an error
    - `math.sin(0)` will throw an error

`math` module

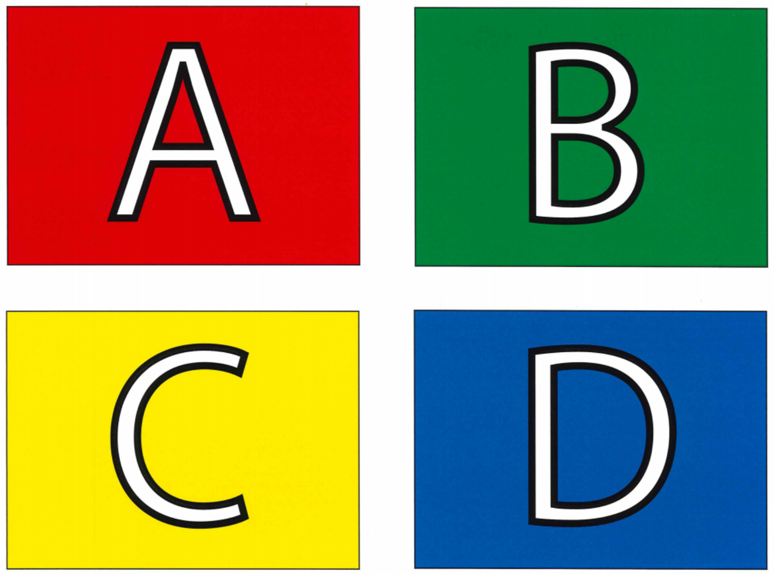
- The math module has useful stuff!
- You can read about it in the documentation.
- logarithms, trigonometry, …
- Modules can also contain values:



```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>>
```

import statements

Which of the following correctly computes the are of a circle with radius 4?



A

```
from math import pi
area = math.pi * 4**2
```

B

```
import math
area = math.pi * 4**2
```

C

```
from math import pi
area = (pi * 4)**2
```

D

```
import pi
area = pi * 4**2
```

import statements

Which of the following correctly computes the are of a circle with radius 4?

Only `pi` is available:
`math` is not imported.

A

```python
from math import pi
area = math.pi * 4**2
```

This works!

B

```python
import math
area = math.pi * 4**2
```

Formula is wrong!

C

```python
from math import pi
area = (pi * 4)**2
```

There is no `pi` module.

D

```python
import pi
area = pi * 4**2
```

`turtle` module

Python has Turtles!

```
import turtle
scott = turtle.Turtle()
```

turtle module

Python has Turtles!

```
import turtle
scott = turtle.Turtle()
```

Basic turtle methods

- forward: moves the turtle forward
- left/right: turns the turtle
- penup/pendown: turns drawing on and off

Creating and Using Objects

```python
import turtle
scott = turtle.Turtle()
```

Creating and Using Objects

```
import turtle
scott = turtle.Turtle()
```

The `Turtle()` function starts with a capital letter.
By convention this indicates that it is a special kind of function called a constructor that creates (and returns) new objects of type `Turtle`.

Creating and Using Objects

```
import turtle
scott = turtle.Turtle()
```

The `Turtle()` function starts with a capital letter.
By convention this indicates that it is a special kind of function called a constructor that creates (and returns) new objects of type `Turtle`.

The Turtle() function returns a Turtle object, and the variable `scott` now refers to it.

Creating and Using Objects

```
import turtle
scott = turtle.Turtle()
```

The `Turtle()` function starts with a capital letter.
By convention this indicates that it is a special kind of function called a constructor that creates (and returns) new objects of type `Turtle`.

The Turtle() function returns a Turtle object, and the variable `scott` now refers to it.

Objects can have functions associated with them, accessed via the dot notation, e.g.:

```
turtle.forward(10) # moves the turtle forward 10 units
turtle.left(90) # turns the turtle left 90 degrees
```

Creating and Using Objects

```
import turtle
scott = turtle.Turtle()
```

The `Turtle()` function starts with a capital letter.
By convention this indicates that it is a special kind of function called a constructor that creates (and returns) new objects of type `Turtle`.

The Turtle() function returns a Turtle object, and the variable `scott` now refers to it.

functions that belong to an object are called its ***methods***

Objects can have functions associated with them, accessed via the dot notation, e.g.:

```
turtle.forward(10) # moves the turtle forward 10 units
turtle.left(90) # turns the turtle left 90 degrees
```

Creating and Using Objects

```
import turtle
scott = turtle.Turtle()
```

The `Turtle()` function starts with a capital letter.
By convention this indicates that it is a special kind of function called a constructor that creates (and returns) new objects of type `Turtle`.

The Turtle() function returns a Turtle object, and the variable `scott` now refers to it.

*functions that belong to an object are called its **methods***

Objects can have functions associated with them, accessed via the dot notation, e.g.:

```
turtle.forward(10) # moves the turtle forward 10 units
turtle.left(90) # turns the turtle left 90 degrees
```

What methods do Turtles have? Lots!
Check the docs: https://docs.python.org/3.3/library/turtle.html?highlight=turtle

Algorithms with Turtles

Task: Write pseudocode for an algorithm to draw a square with side length 100:

Algorithms with Turtles

Task: Write pseudocode for an algorithm to draw a square with side length 100:

Algorithms with Turtles

Task: Write pseudocode for an algorithm to draw a square with side length 100:

1. Move forward 100

Algorithms with Turtles

Task: Write pseudocode for an algorithm to draw a square with side length 100:

```
1. Move forward 100
2. Turn left 90 degrees
```

Algorithms with Turtles

Task: Write pseudocode for an algorithm to draw a square with side length 100:

```
1. Move forward 100
2. Turn left 90 degrees
3. Move forward 100
```

Algorithms with Turtles

Task: Write pseudocode for an algorithm to draw a square with side length 100:

```
1. Move forward 100
2. Turn left 90 degrees
3. Move forward 100
4. Turn left 90 degrees
```

Algorithms with Turtles

Task: Write pseudocode for an algorithm to draw a square with side length 100:

```
1. Move forward 100
2. Turn left 90 degrees
3. Move forward 100
4. Turn left 90 degrees
5. Move forward 100
```

Algorithms with Turtles

Task: Write pseudocode for an algorithm to draw a square with side length 100:

```
1. Move forward 100
2. Turn left 90 degrees
3. Move forward 100
4. Turn left 90 degrees
5. Move forward 100
6. Turn left 90 degrees
```

Algorithms with Turtles

Task: Write pseudocode for an algorithm to draw a square with side length 100:

```
1. Move forward 100
2. Turn left 90 degrees
3. Move forward 100
4. Turn left 90 degrees
5. Move forward 100
6. Turn left 90 degrees
7. Move forward 100
```

Algorithms with Turtles

Task: Write pseudocode for an algorithm to draw a square with side length 100:

```
1. Move forward 100
2. Turn left 90 degrees
3. Move forward 100
4. Turn left 90 degrees
5. Move forward 100
6. Turn left 90 degrees
7. Move forward 100
8. (Turn left 90 degrees)
```

Algorithms with Turtles

Task: Write pseudocode for an algorithm to draw a square with side length 100:

```
1. Move forward 100
2. Turn left 90 degrees
3. Move forward 100
4. Turn left 90 degrees
5. Move forward 100
6. Turn left 90 degrees
7. Move forward 100
8. (Turn left 90 degrees)
```

Algorithms with Turtles

Task: Write pseudocode for an algorithm to draw a square with side length 100:

```
1. Move forward 100
2. Turn left 90 degrees
3. Move forward 100
4. Turn left 90 degrees
5. Move forward 100
6. Turn left 90 degrees
7. Move forward 100
8. (Turn left 90 degrees)
```

Can we do better?

Algorithms with Turtles

Task: Write pseudocode for an algorithm to draw a square with side length 100:

```
Repeat 4 times:
    1. move forward 100
    2. turn left 90
```

Algorithms with Turtles

Task: Write pseudocode for an algorithm to draw a square with side length 100:

```
Repeat 4 times:
    1. move forward 100
    2. turn left 90
```

Demo

Demo

- turtle_square.py: Write a loop-based program that makes a turtle and draws a square with it.

Hot take: for some tasks, while loops are annoying.

Hot take: for some tasks, while loops are annoying.

- Often, you want: “Do `someThing()` 10 times”

Hot take: for some tasks, while loops are annoying.

- Often, you want: “Do `someThing()` 10 times”
- With a while loop you need to:

Hot take: for some tasks, while loops are annoying.

- Often, you want: “Do `someThing()` 10 times”
- With a while loop you need to:

```
i = 0
while i < 10:
    someThing()
    i += 1
```

Hot take: for some tasks, while loops are annoying.

- Often, you want: “Do `someThing()` 10 times”
- With a while loop you need to:

```
i = 0
while i < 10:
    someThing()
    i += 1
```

I don’t even care about `i`, it’s just bookkeeping!

Hot take: for some tasks, while loops are annoying.

- Often, you want: “Do `someThing()` 10 times”
- With a while loop you need to:

```
i = 0
while i < 10:
    someThing()
    i += 1
```

I don’t even care about `i`, it’s just bookkeeping!

- Wouldn’t it be great if we could:

Hot take: for some tasks, while loops are annoying.

- Often, you want: “Do `someThing()` 10 times”
- With a while loop you need to:

```
i = 0
while i < 10:
    someThing()
    i += 1
```

I don’t even care about `i`, it’s just bookkeeping!

- Wouldn’t it be great if we could:

```
do 10 times:
    someThing()
```

Hot take: for some tasks, while loops are annoying.

- Often, you want: “Do `someThing()` 10 times”
- With a while loop you need to:

```
i = 0
while i < 10:
    someThing()
    i += 1
```

I don’t even care about `i`, it’s just bookkeeping!

- Wouldn’t it be great if we could:

```
do 10 times:
    someThing()
```

We (almost) can! Using `for` loops.

The `for` statement: syntax

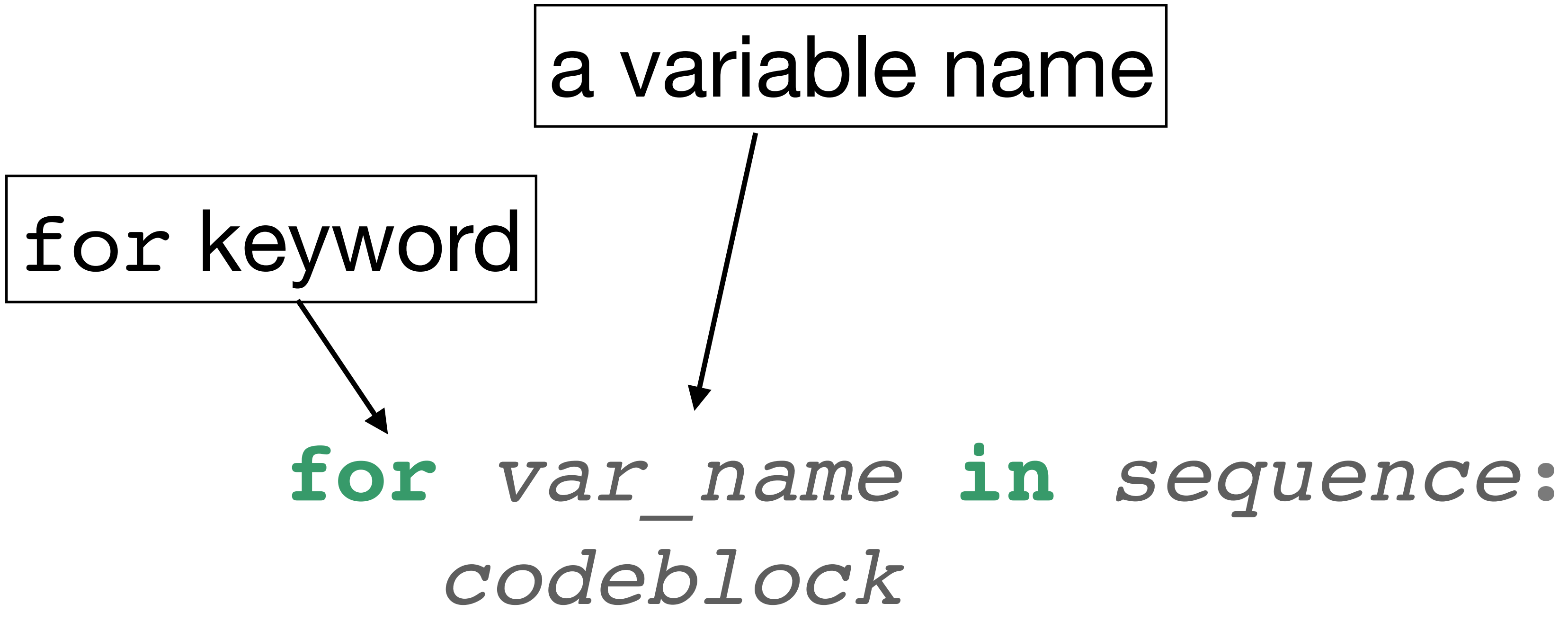
```
for var_name in sequence:
    codeblock
```

The for statement: syntax

for keyword

```
for var_name in sequence:
    codeblock
```

The `for` statement: syntax

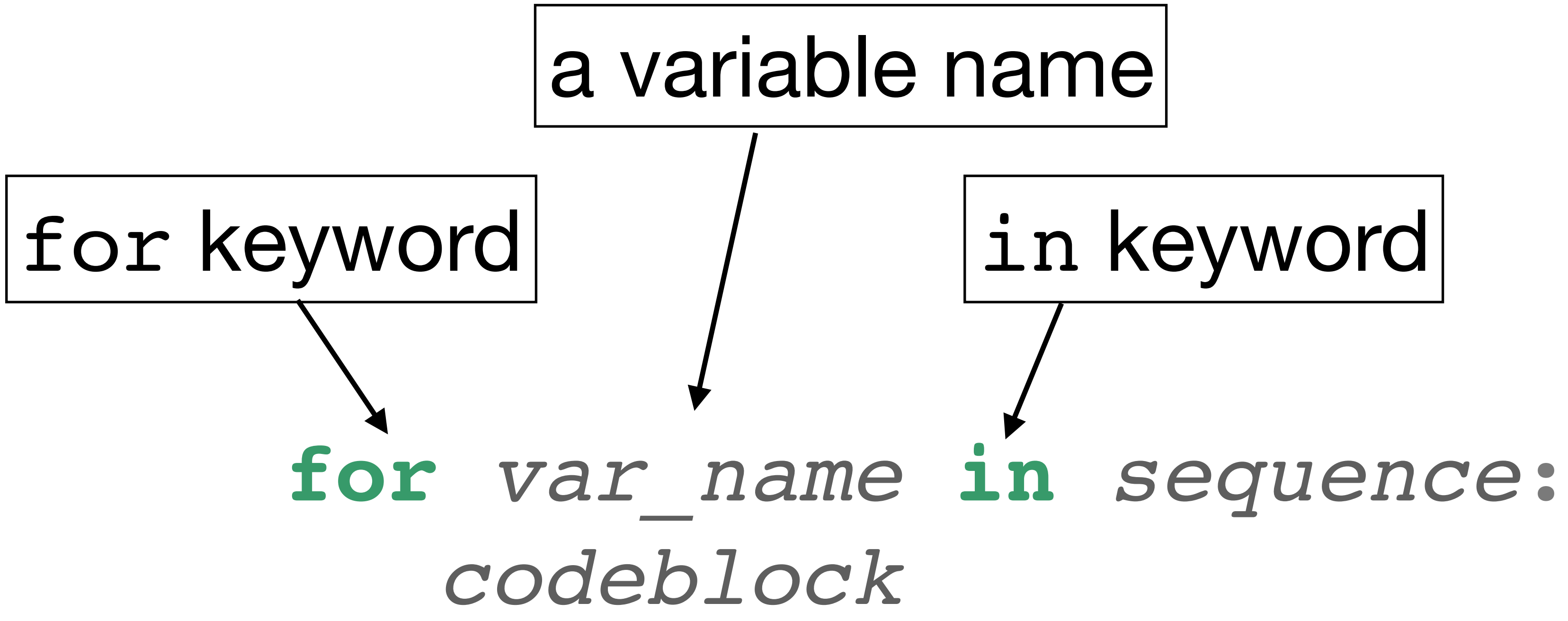


a variable name

`for` keyword

```
for var_name in sequence:
    codeblock
```

The `for` statement: syntax



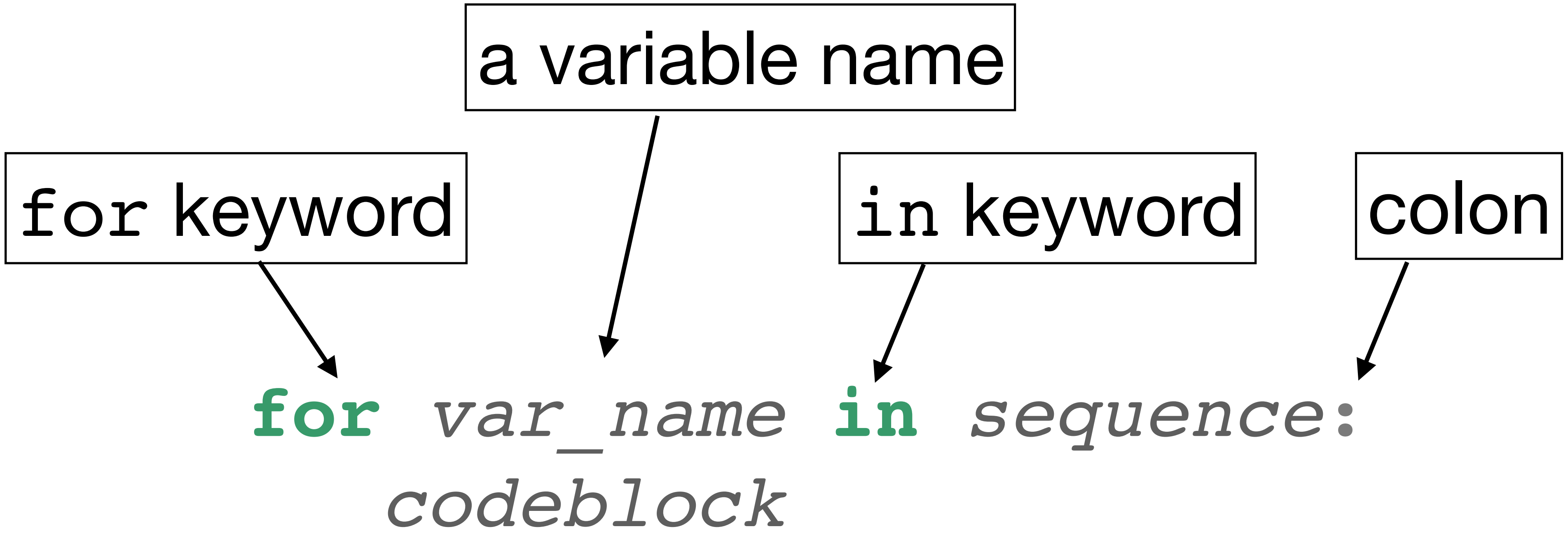
a variable name

for keyword

in keyword

```
for var_name in sequence:
    codeblock
```

The `for` statement: syntax



a variable name

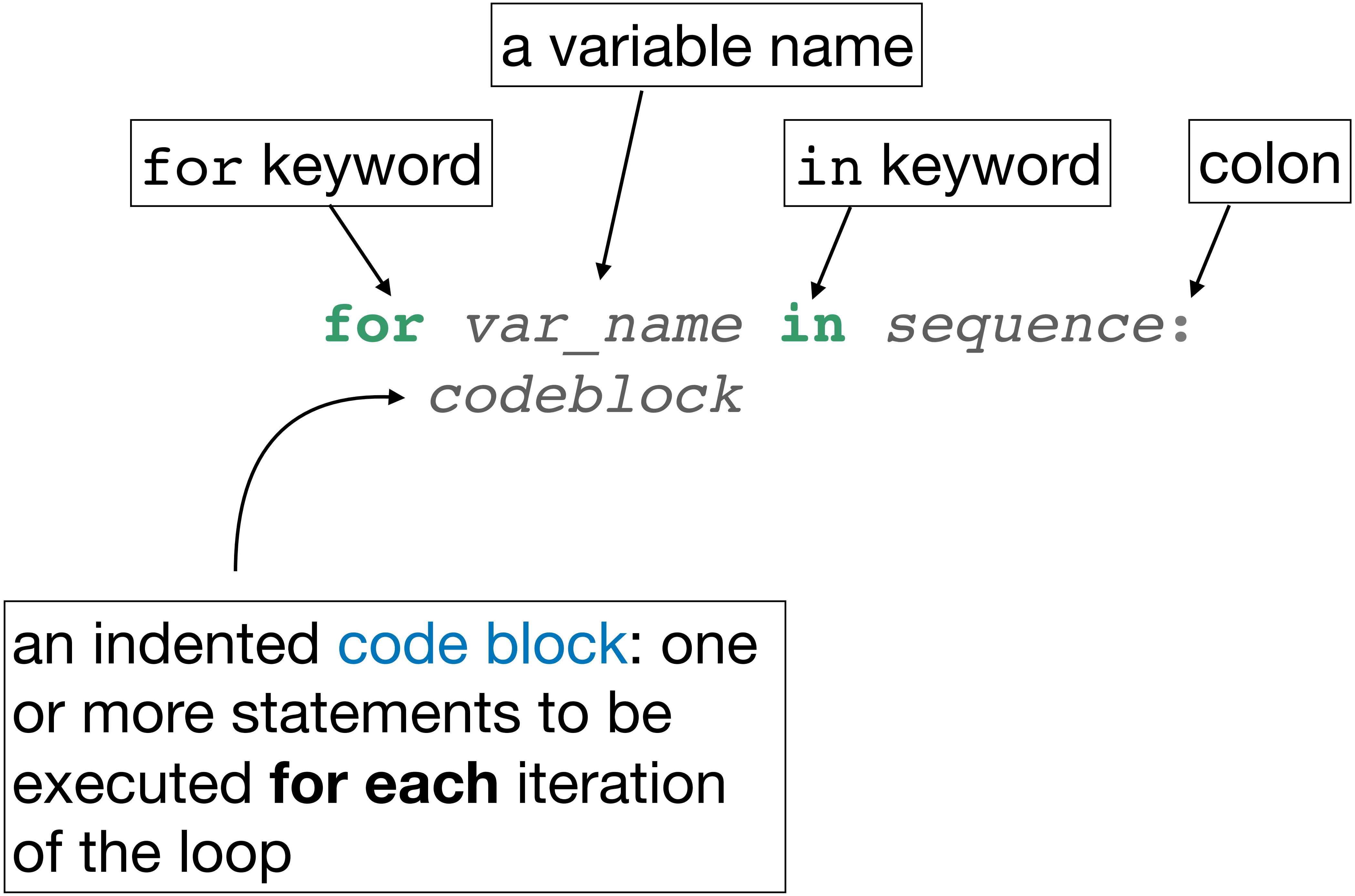
`for` keyword

`in` keyword

colon

```
for var_name in sequence:
    codeblock
```

The `for` statement: syntax



a variable name

`for` keyword

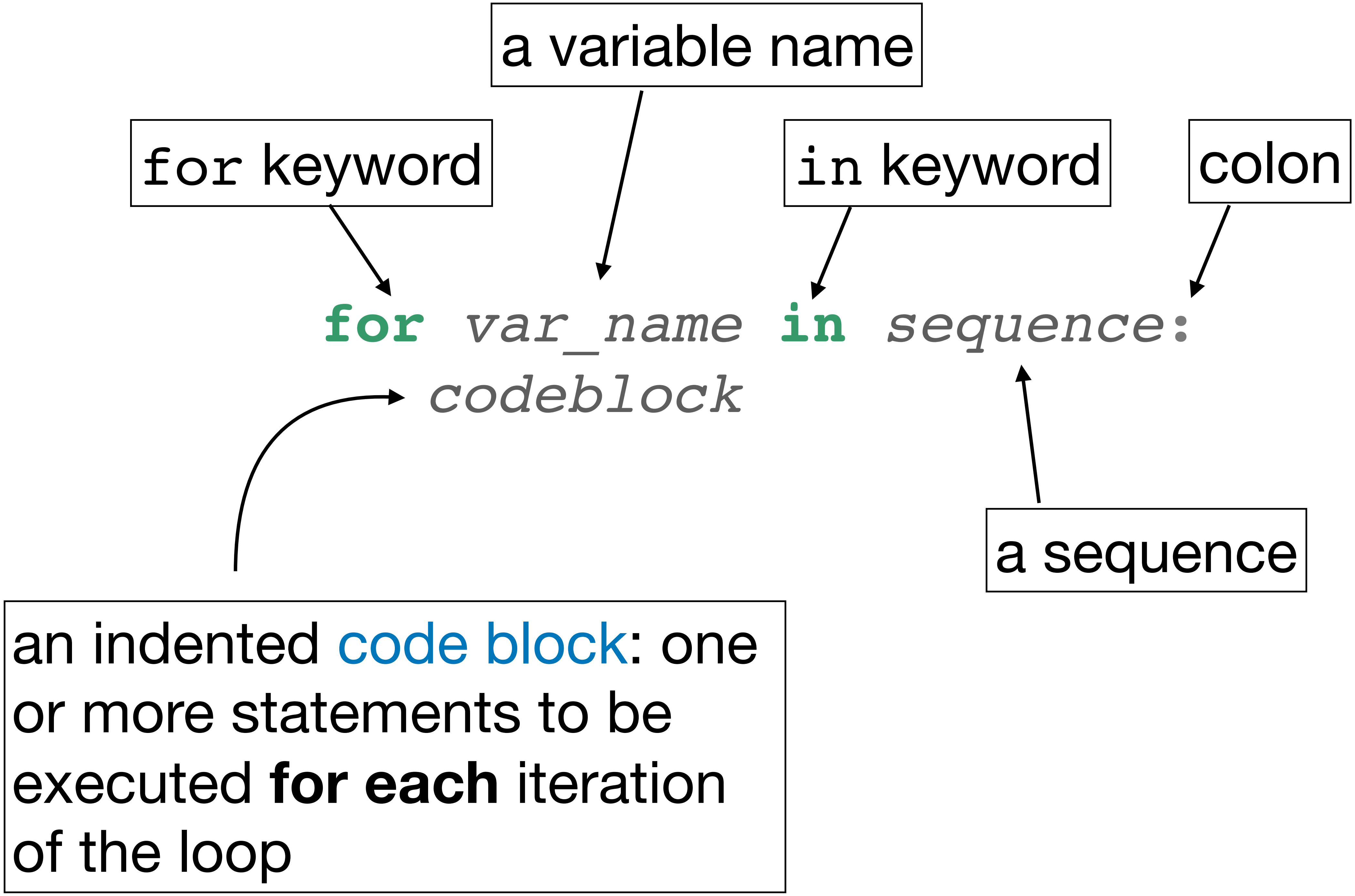
`in` keyword

colon

```
for var_name in sequence:
    codeblock
```

an indented code block: one or more statements to be executed **for each** iteration of the loop

The `for` statement: syntax



a variable name

`for` keyword

`in` keyword

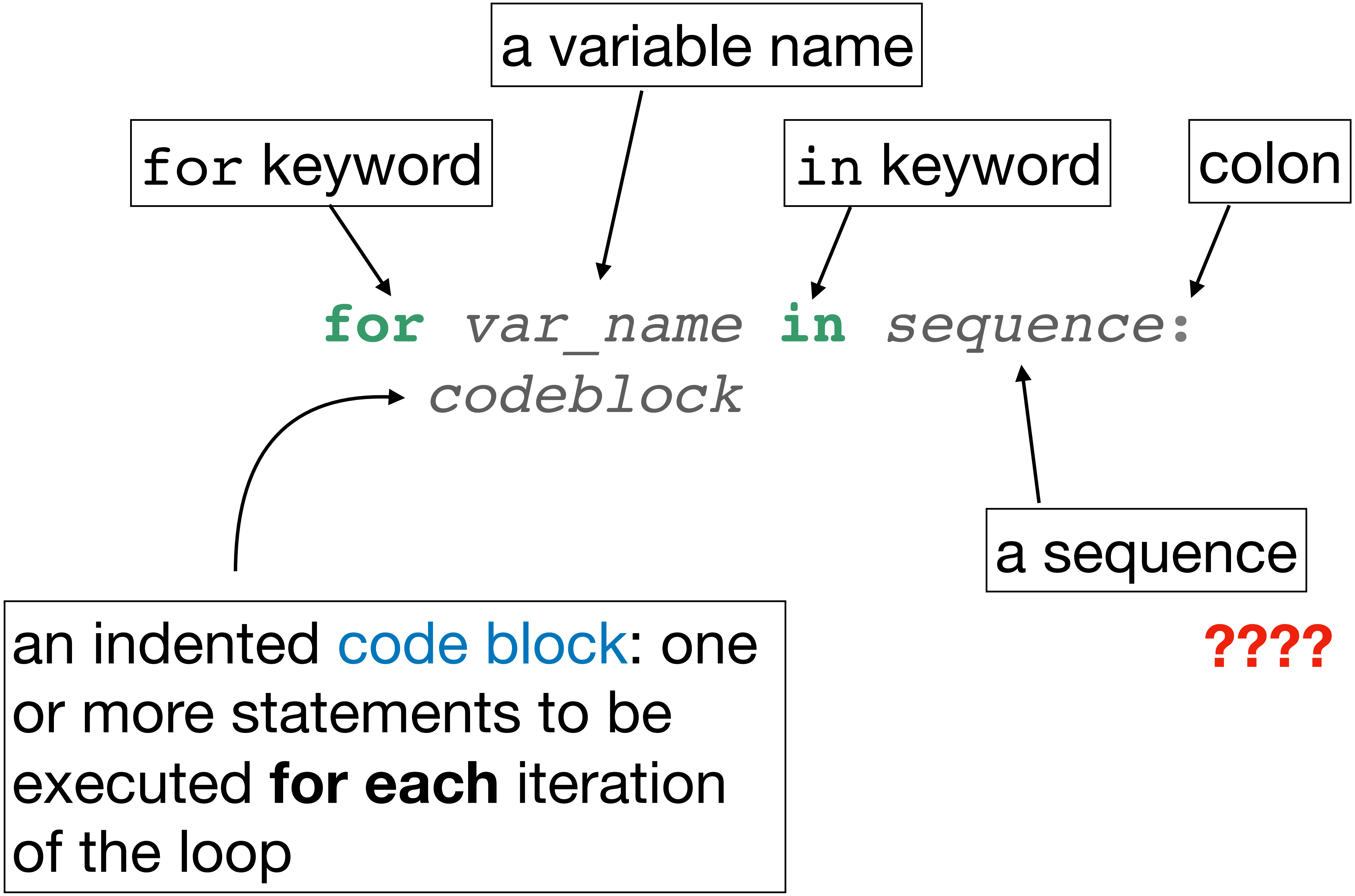
colon

```
for var_name in sequence:
    codeblock
```

a sequence

an indented code block: one or more statements to be executed **for each** iteration of the loop

The `for` statement: syntax



a variable name

`for` keyword

`in` keyword

colon

```
for var_name in sequence:
    codeblock
```

a sequence

an indented code block: one or more statements to be executed **for each** iteration of the loop

????

Sequences in Python: Lists

```python
for color in ["red", "green", "blue"]:
    print(color)
```

This code prints:

```
red
green
blue
```

Sequences in Python: Lists

```
for color in ["red", "green", "blue"]:
    print(color)
```

This is a `list`: an ordered collection of values.
Much more on these later.

This code prints:

```
red
green
blue
```

The `for` statement: behavior

```
for color in ["red", "green", "blue"]:
    print(color)
```

This code prints:

```
red
green
blue
```

The `for` statement: behavior

```
for color in ["red", "green", "blue"]:
    print(color)
```

The loop body is executed once **for each** value in the sequence (list).

This code prints:

```
red
green
blue
```

The `for` statement: behavior

```
for color in ["red", "green", "blue"]:
    print(color)
```

The loop body is executed once **for each** value in the sequence (list).

This code prints:

```
red
green
blue
```

In *each* iteration, the loop variable

The `for` statement: behavior

```
for color in ["red", "green", "blue"]:
    print(color)
```

The loop body is executed once **for each** value in the sequence (list).

This code prints:

```
red
green
blue
```

In *each* iteration, the loop variable (`color`)

The `for` statement: behavior

```
for color in ["red", "green", "blue"]:
    print(color)
```

The loop body is executed once **for each** value in the sequence (list).

This code prints:

```
red
green
blue
```

In *each* iteration, the loop variable (`color`) takes on a *different* value from the sequence:

The `for` statement: behavior

```python
for color in ["red", "green", "blue"]:
    print(color)
```

The loop body is executed once **for each** value in the sequence (list).

This code prints:

```
red
green
blue
```

In *each* iteration, the loop variable (`color`) takes on a *different* value from the sequence:

(`"red"`, then `"green"`, then `"blue"`)

The `for` statement: behavior

```
for color in ["red", "green", "blue"]:
    print(color)
```

The loop body is executed once **for each** value in the sequence (list).

This code prints:

```
red
green
blue
```

In *each* iteration, the loop variable (`color`) takes on a *different* value from the sequence:

(`"red"`, then `"green"`, then `"blue"`)

Notice: the loop variable gets updated **automatically** after each iteration!

Sequences in Python: Ranges

Lists are great if you have a list of things, but what about:

Sequences in Python: Ranges

Lists are great if you have a list of things, but what about:

“Do `someThing()` 10 times”?

Sequences in Python: Ranges

Lists are great if you have a list of things, but what about:

“Do `someThing()` 10 times”? ugh.

Sequences in Python: Ranges

Lists are great if you have a list of things, but what about:

“Do `someThing()` 10 times”? ugh.

```
for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    someThing()
```

Sequences in Python: Ranges

Lists are great if you have a list of things, but what about:

“Do `someThing()` 10 times”? ugh.

```
for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    someThing()
```

New function to the rescue: `range` makes it easy to generate lists like this.

Sequences in Python: Ranges

```python
for i in range(5):
    print(i)
```

This code prints:

```
0
1
2
3
4
```

Sequences in Python: Ranges

```
for i in range(5):
    print(i)
```

The range function returns a sequence of integers.

This code prints:

```
0
1
2
3
4
```

Sequences in Python: Ranges

```
for i in range(5):
    print(i)
```

The `range` function returns a sequence of integers.

Not technically a list, but acts like one: more on this later

This code prints:

```
0
1
2
3
4
```

Sequences in Python:
the `range` function

Sequences in Python: the range function

```
for i in range(5):
    print(i, end=“ “)
```

prints: 0 1 2 3 4

Sequences in Python: the `range` function

`range(a)`: from **0** *up to* but *not including* **a**

```
for i in range(5):
    print(i, end=“ “)
```

prints: 0 1 2 3 4

Sequences in Python: the `range` function

`range(`**`a`**`)`: from **0** *up to* but *not including* **`a`**

```
for i in range(5):
    print(i, end=“ “)
```

prints: 0 1 2 3 4

```
for i in range(2, 5):
    print(i, end=“ “)
```

prints: 2 3 4

Sequences in Python: the `range` function

`range(a)`: from **0** *up to* but *not including* **a**

```
for i in range(5):
    print(i, end=“ “)
```

prints: 0 1 2 3 4

`range(a, b)`: from **a** *up to* but *not including* **b**

```
for i in range(2, 5):
    print(i, end=“ “)
```

prints: 2 3 4

Sequences in Python: the `range` function

`range(`**`a`**`)`: from **0** *up to* but *not including* **`a`**

```
for i in range(5):
    print(i, end=“ “)
```

prints: `0 1 2 3 4`

`range(`**`a, b`**`)`: from **a** *up to* but *not including* **`b`**

```
for i in range(2, 5):
    print(i, end=“ “)
```

prints: `2 3 4`

```
for i in range(1, 8, 3):
    print(i, end=“ “)
```

prints: `1, 4, 7`

Sequences in Python: the `range` function

`range(`**`a`**`)`: from **0** *up to* but *not including* **a**

```
for i in range(5):
    print(i, end=“ “)
```

prints: 0 1 2 3 4

`range(`**`a`**`, `**`b`**`)`: from **a** *up to* but *not including* **b**

```
for i in range(2, 5):
    print(i, end=“ “)
```

prints: 2 3 4

`range(`**`a`**`, `**`b`**`, `**`c`**`)`: sequence from **a** *up to* but *not including* **b** counting in *increments* of **c**

```
for i in range(1, 8, 3):
    print(i, end=“ “)
```

prints: 1, 4, 7

Converting ranges to lists

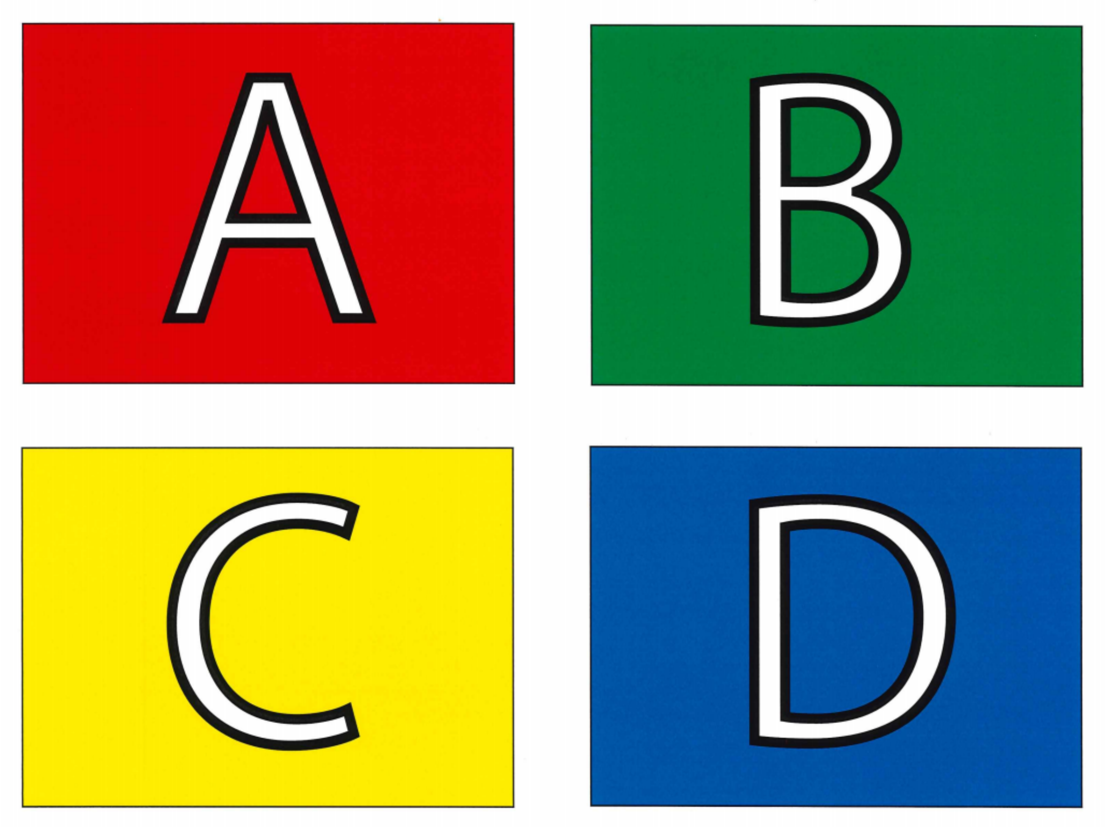
The `range` function returns a sequence of integers.

It’s not technically a list: print(range(4)) does not print `[1, 2, 3]`

To turn the range into a list (e.g., to print it), we can use the list function:

```
list(range(2, 5)) => [2, 3, 4]
```

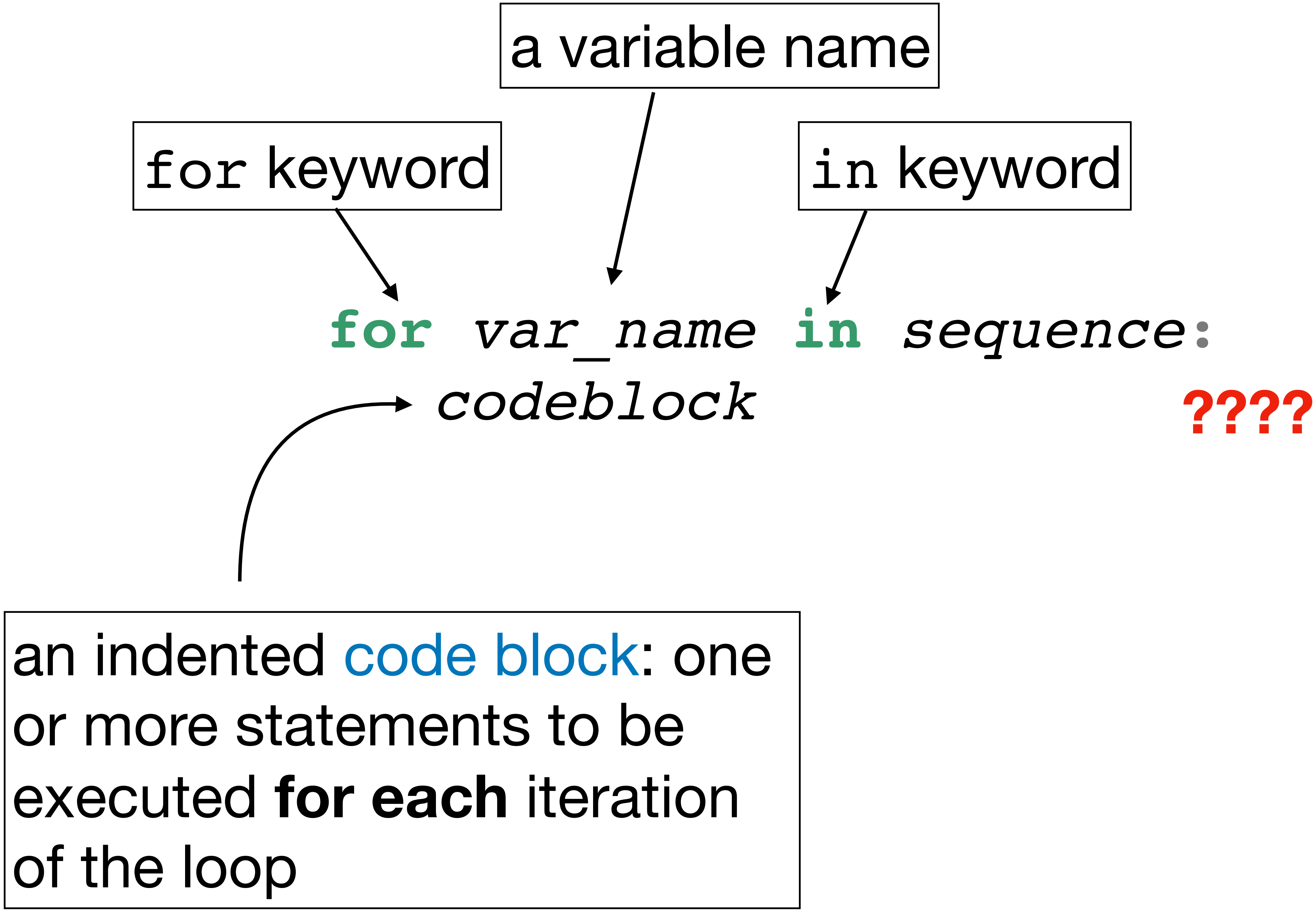
Range function: Demo



Range function: Demo

- range_demo.py

Back to `for` loops...



a variable name

`for` keyword

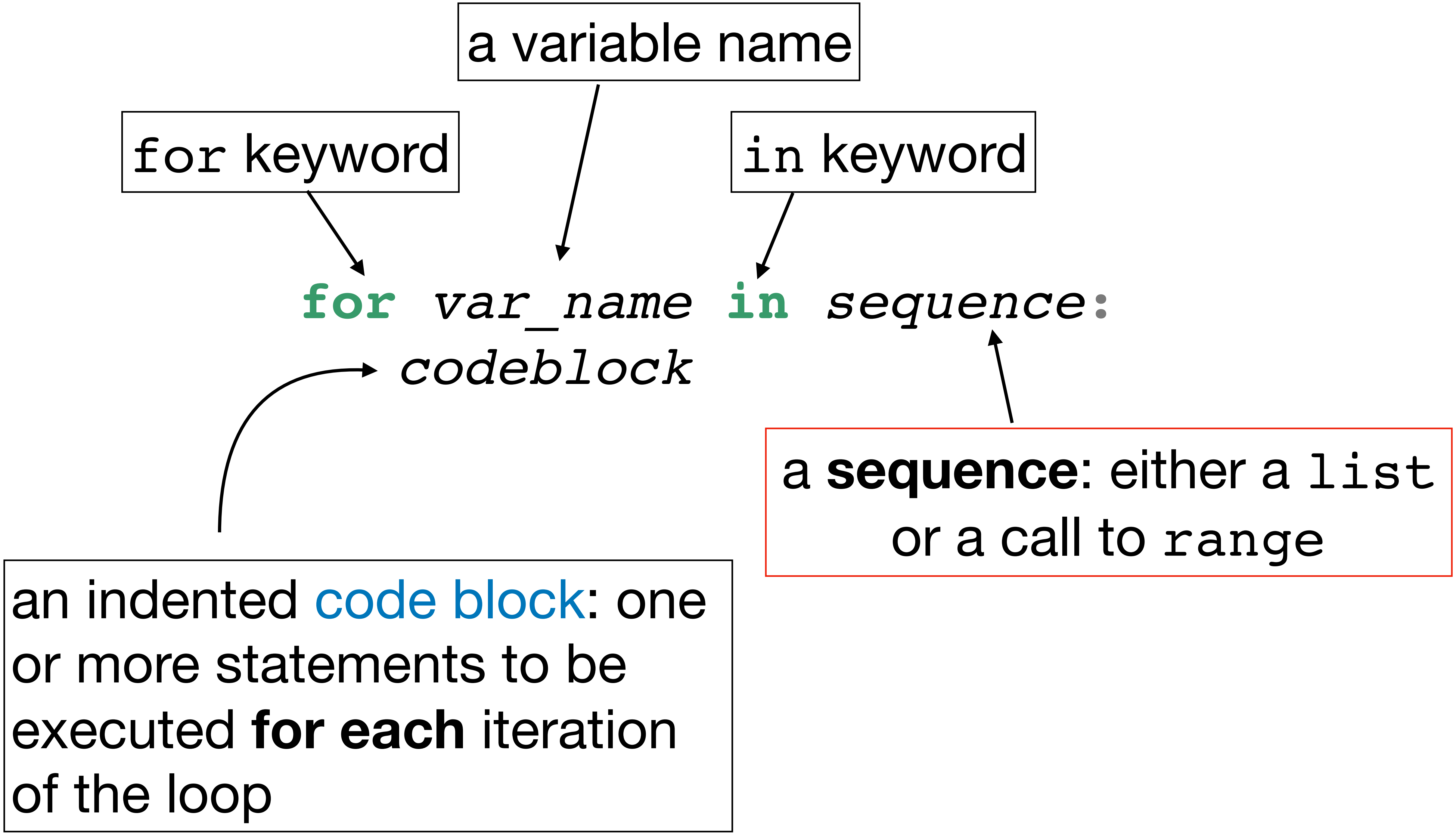
`in` keyword

```
for var_name in sequence:
    codeblock
```

????

an indented code block: one or more statements to be executed **for each** iteration of the loop

Back to `for` loops...



a variable name

`for` keyword

`in` keyword

```
for var_name in sequence:
    codeblock
```

a **sequence**: either a `list` or a call to `range`

an indented code block: one or more statements to be executed **for each** iteration of the loop

Today’s Quiz

- 3 minutes

Today’s Quiz

- 3 minutes
- Working with a neighbor: do your answers agree? (2 minutes)

Demo

- turtle_square.py, revisited: let’s rewrite this with a for loop.

Generalized Squares, AKA Equilateral Polygons

Exercise 4: Write code that makes the Turtle object `scott` draw an n-sided polygon, where n and the length of each side are given by the user.

Hint: the total amount the turtle needs to turn is 360 degrees.
Code from turtle_square:

```
import turtle

scott = turtle.Turtle()
for i in range(4):
    scott.forward(100)
    scott.left(90)
```

Additional Suggested Practice Problems

1. Make a Turtle do a random walk: write a program that repeats the following 100 times:

   - Move the turtle a random distance forward.
   - Turn the turtle a random amount.

2. Re-write the dice exercise from last time using `for` loops (it's simpler this way!)