Lecture 11:
for loops and the range function
Turtles!?
Happenings

- Slalom Information Session Tuesday, October 22\textsuperscript{nd}
  - Information Booth 10:30-12:00 PM CF First Floor Foyer
  - Information Session 6:00-7:00 PM AW 204
- CS Mentors Program Present “GDB Workshop” Thursday, October 24\textsuperscript{th} 3 PM CF 165
  - Perfect for students in CSCI 247 & 347
- Internship and Volunteer Fair Thursday, October 24\textsuperscript{th} 12:00-4:00 PM VU Multipurpose Room
- Amazon Tuesday, October 29\textsuperscript{th}
  - Info Table 10:30-12:00 PM at VU Lobby
  - Info Session 4:00-5:00 PM at AW 204
  - Resume Prep 5:15-7:00 PM at AW 204
  - Open to all of campus not just CS students
- PACCAR Career Day Wednesday, October 30\textsuperscript{th} 10:00 AM-3:00 PM South Campus
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)
Announcements

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

- Sample programming questions will be released this afternoon.
Goals

• 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.

• Know how to use the turtle module to:
  
  • Create a Turtle `object`

  • Call the turtle object's `methods` (functions) to move it around the screen and draw simple shapes: (forward, left, right, penup, pendown)
Hot take: for some tasks, while loops are annoying.
Hot take: for some tasks, while loops are annoying.

- Often, you want: “Do some_thing() 10 times”
Hot take: for some tasks, while loops are annoying.

• Often, you want: “Do some_thing() 10 times”

• With a while loop you need to:
Hot take: for some tasks, while loops are annoying.

• Often, you want: “Do some\_thing() 10 times”

• With a while loop you need to:

```
i = 0
while i < 10:
    some\_thing()
i += 1
```
Hot take: for some tasks, while loops are annoying.

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

- With a while loop you need to:

```python
i = 0
while i < 10:
    # I don’t even care about i,
    # it’s just bookkeeping!
    some_thing()
    i += 1
```
Hot take: for some tasks, while loops are annoying.

• Often, you want: “Do some_thing() 10 times”

• With a while loop you need to:

```python
i = 0
while i < 10:  # I don’t even care about i, it’s just bookkeeping!
    some_thing()
    i += 1
```

• Wouldn’t it be great if we could:
Hot take: for some tasks, while loops are annoying.

• Often, you want: “Do some\_thing() 10 times”

• With a while loop you need to:

```python
i = 0
while i < 10:
    some\_thing()
    i += 1
```

• Wouldn’t it be great if we could:

```python
do 10 times:
    some\_thing()
```
Hot take: for some tasks, while loops are annoying.

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

- With a while loop you need to:

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

- Wouldn’t it be great if we could:

  ```python
  do 10 times:
      some_thing()
  ```

We (almost) can! Using for loops.
The **for statement**: syntax

```python
for var_name in sequence:
    codeblock
```
The **for statement**: syntax

```python
for var_name in sequence:
    codeblock
```
The **for** statement: syntax

```plaintext
for var_name in sequence:
    codeblock
```
The **for** statement: syntax

```plaintext
for var_name in sequence:
    codeblock
```
The **for statement**: syntax

```
for var_name in sequence:
    codeblock
```
The **for** statement: syntax

- **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

- **for** keyword
- **in** keyword
- **colon**

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

- a variable name
- a sequence

An indented **code block**: one or more statements to be executed **for each** iteration of the loop
The **for** statement: syntax

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

- **for** keyword
- **in** keyword
- **colon**
- **a variable name**
- **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

```python
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

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

This code prints:
red
green
blue
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
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:  

In each iteration, the loop variable

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

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

**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 some_thing() 10 times”? ugh.
Sequences in Python: Ranges

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

“Do some_thing() 10 times”? ugh.

```python
for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    some_thing()
```
Sequences in Python: Ranges

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

“Do some\_thing() 10 times”? ugh.

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

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)

This code prints:
0
1
2
3
4

The `range` function returns a sequence of integers.
Sequences in Python: Ranges

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

This code prints:
```
0
1
2
3
4
```

The `range` function returns a sequence of integers. Not technically a list, but acts like one: more on this later.
Sequences in Python: the `range` function
Sequences in Python: the `range` function

```python
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`

```python
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`

```python
for i in range(5):
    print(i, end=" ")
```
prints:  0  1  2  3  4

```python
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`

```python
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`

```python
for i in range(2, 5):
    print(i, end=" ")  # prints: 2 3 4
```

```python
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

```python
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

```python
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

```python
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

- demo in shell
  - one, two, and three argument versions
- ranges.py
Range function: Demo
Range function: Demo

for posterity: see ranges.py
for x in range(1, 4):
    print (x + x * x, end=str(x))
Size of a range

```
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
```

```
for i in range(1, 8, 3):
    print(i, end=" ")
# prints: 1, 4, 7
```

**Exercise:** How many elements are in `range(n)`?

A. 0  
B. n-1  
C. n  
D. 10
Size of a range

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

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

prints: 1, 4, 7

Exercise: How many elements are in \texttt{range}(a, b)\

A. a-b
B. b-a-1
C. b-a+1
D. b-a
QOTD

When running the code below, what pairs of values could be assigned to the variables \( x \) and \( y \) so that the program prints WWU 43 times? Select all correct choices.

```
x =
y =
for z in range(x, y):
    print("WWU")
```

Equivalent question:
for which of these is \( y - x == 43 \)?

- \( x: 0 \quad y: 44 \)
- \( x: -21 \quad y: 22 \)
- \( x: -21 \quad y: 21 \)
- \( x: -789 \quad y: -746 \)
- \( x: -789 \quad y: 746 \)
- \( x: 1 \quad y: 44 \)
Back to for loops...

- **for keyword**: a variable name
- **in keyword**

```python
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...

- **for keyword**
- **in keyword**
- a variable name
- **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
while loops are annoying.

• Often, you want: “Do some\_thing() 10 times”

• With a while loop you need to:

```python
i = 0
while i < 10:  
    some\_thing()  
    i += 1
```

• Wouldn’t it be great if we could:
while loops are annoying.

- Often, you want: “Do some\_thing() 10 times”

- With a while loop you need to:

\[
\begin{align*}
i & = 0 \\
\textbf{while} & \quad i < 10: \quad \text{I don’t even care about } i, \\
& \quad \textbf{some\_thing}() \quad \text{it’s just bookkeeping!} \\
& \quad i += 1
\end{align*}
\]

- Wouldn’t it be great if we could:

\[
\begin{align*}
\textbf{for} & \quad i \quad \textbf{in range}(10): \\
& \quad \textbf{some\_thing}()
\end{align*}
\]
while loops are annoying.

• Often, you want: “Do _some_thing() 10 times”

• With a while loop you need to:

  ```python
  i = 0
  while i < 10:  # I don’t even care about i,
      some_thing()  # it’s just bookkeeping!
      i += 1
  ```

• Wouldn’t it be great if we could:

  ```python
  for i in range(10):
      some_thing()
  ```

  We can!
Revisiting Repetition

```
for var_name in sequence:
    codeblock
```
Revisiting Repetition

\begin{verbatim}
for var_name in sequence:
    codeblock
\end{verbatim}

- balance3.py - rewrite yearly bank account balance with a for loop
Revisiting Repetition

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

- balance3.py - rewrite yearly bank account balance with a for loop

- Average of 100 random numbers
Revisiting Repetition

for var_name in sequence:
    codeblock

• balance3.py - rewrite yearly bank account balance with a for loop

• Average of 100 random numbers

• New problem: print all possible outcomes of two 6-sided dice.
Task: Print out all possible rolls of two six-sided dice.

Program output:

1 1
1 2
1 3
1 4
1 5
1 6
2 1
2 2
2 3
2 4
(and so on)

6 4
6 5
6 6
Nesting loops!

Task: Print out all possible rolls of two six-sided dice.

Program output:

1 1
1 2
1 3
1 4
1 5
1 6
2 1
2 2
2 3
2 4
(...)
6 4
6 5
6 6
Nesting loops!

**Task:** Print out all possible rolls of two six-sided dice.

Program output:

1 1
1 2
1 3
1 4
1 5
1 6
2 1
2 2
2 3
2 4
...
6 4
6 5
6 6

Break down the problem:
Nesting loops!

**Task:** Print out all possible rolls of two six-sided dice.

Break down the problem:
- print 1 followed by each of 1 to 6

Program output:

1 1
1 2
1 3
1 4
1 5
1 6
2 1
2 2
2 3
2 4
2 5
2 6
3 1
3 2
3 3
3 4
3 5
3 6
4 1
4 2
4 3
4 4
4 5
4 6
5 1
5 2
5 3
5 4
5 5
5 6
6 1
6 2
6 3
6 4
6 5
6 6

(and so on)
Nesting loops!

Task: Print out all possible rolls of two six-sided dice.

Program output:

1 1
1 2
1 3
1 4
1 5
1 6
2 1
2 2
2 3
2 4
(…)
5 1
5 2
5 3
5 4
5 5
5 6
6 1
6 2
6 3
6 4
6 5
6 6

Break down the problem:
• print 1 followed by each of 1 to 6
• print 2 followed by each of 1 to 6
Nesting loops!

Task: Print out all possible rolls of two six-sided dice.

Break down the problem:
• print 1 followed by each of 1 to 6
• print 2 followed by each of 1 to 6
• and so on

Program output:
1 1
1 2
1 3
1 4
1 5
1 6
2 1
2 2
2 3
2 4
(and so on)
...
Nesting loops!

Task: Print out all possible rolls of two six-sided dice.

Program output:
1 1
1 2
1 3
1 4
1 5
1 6
2 1
2 2
2 3
2 4
( and so on)
...
6 4
6 5
6 6

Break down the problem:
• print 1 followed by each of 1 to 6
• print 2 followed by each of 1 to 6
• and so on

Repetitive task
**Nesting loops!**

**Task:** Print out all possible rolls of two six-sided dice.

- Print 1 followed by each of 1 to 6
- Print 2 followed by each of 1 to 6
- And so on

**Program output:**

1 1
1 2
1 3
1 4
1 5
1 6
2 1
2 2
2 3
2 4
...
6 4
6 5
6 6

Repetitive task
Nesting loops! Demo

- dice.py - nested for loops to print all ordered pairs of numbers from 1 to 6 (inclusive)
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.
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:

```python
import module
```

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

Python has Turtles!
turtle module

Python has Turtles!

import turtle
turtle module

Python has Turtles!

import turtle
turtle module

Python has Turtles!

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

What does this do?
turtle module

Python has Turtles!

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

What does this do?
turtle module

Python has Turtles!

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

What does this do? Let’s play with it.
Demo: basic turtle usage

- forward, backward
- left, right
- pendown/down
- penup/up
Creating and Using Objects

```python
import turtle
scott = turtle.Turtle()
```
Creating and Using Objects

```python
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

```python
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

```python
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.:

```python
turtle.forward(10)  # moves the turtle forward 10 units
turtle.left(90)     # turns the turtle left 90 degrees
```
Creating and Using Objects

```python
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.:

```python
turtle.forward(10)  # moves the turtle forward 10 units
turtle.left(90)  # turns the turtle left 90 degrees
```
Creating and Using Objects

```python
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.:

```python
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
turtle module

Python has Turtles!

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

Python has Turtles!

```python
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

```python
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

```python
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

```python
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.:

```python
turtle.forward(10) # moves the turtle forward 10 units
turtle.left(90) # turns the turtle left 90 degrees
```
Creating and Using Objects

```python
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.:

```python
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:
Task: Write pseudocode for an algorithm to draw a square with side length 100:
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
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
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
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)
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)
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?
**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

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

Are for loops always better?
while vs for

Task: Generate and print random integers between 1 and 10 (inclusive) until one of the random numbers exceeds 8.

Would you use a for loop or a while loop?
while vs for

Task: Ask the user for a number (n), then print 100 random numbers between 0 and n.

Would you use a for loop or a while loop?
while vs for

Task: Sum the numbers from 1 to 340, leaving out those divisible by 5.

Would you use a for loop or a while loop?
Generalized Squares
AKA Equilateral Polygons

**Task:** 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:

```python
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 number of degrees.

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