CSCI 141

Lecture 12:
More turtles
Opening the black box: introduction to functions
Announcements
Announcements

• Bring your questions to Wednesday's lecture - QOTDs, coding questions, etc.
Announcements

• Bring your questions to Wednesday's lecture - QOTDs, coding questions, etc.

• Exam material: range(functions)
Announcements

• Bring your questions to Wednesday’s lecture - QOTDs, coding questions, etc.

• Exam material: range(functions)

  • that is, 0 up to but not including writing your own functions (this should be partway through today's lecture)
Goals

• Be able to write programs that make turtles draw simple shapes

• Be able to choose which type of loop (while or for) is best for a given problem.

• Know the syntax for defining your own functions

• Know how to define and use functions that take no arguments and return no values

• Know how to define use parameters to refer to the input arguments of a function
QOTD

for i in range(4):
    for j in range(3, 6):
        print("*", end = " ")
    print()
QOTD

\[ v = 1 \]
\[ \text{for } i \text{ in range}(1, 6): \]
\[ \quad v = v \times i \]

Which of the following programs end with \( v \) having the same value as the program above?

**Program A:**

\[ v = 1 \]
\[ i = 0 \]
\[ \text{while } i < 5: \]
\[ \quad i += 1 \]
\[ \quad v = v \times i \]
QOTD

\[ v = 1 \]
\[ \text{for } i \text{ in range}(1, 6): \]
\[ v = v \times i \]

Which of the following programs end with \( v \) having the same value as the program above?

Program B:

\[ v = 2 \]
\[ \text{for } i \text{ in range}(1, 5): \]
\[ v = v \times i \]
QOTD

\[ v = 1 \]
\[ \text{for } i \text{ in range}(1, 6): \]
\[ v = v \times i \]

Which of the following programs end with \( v \) having the same value as the program above?

Program C:

\[ v = 1 \]
\[ \text{for } i \text{ in range}(5): \]
\[ v = v \times (i + 1) \]
QOTD

```python
v = 1
for i in range(1, 6):
    v = v * i
```

Which of the following programs end with `v` having the same value as the program above?

**Program D:**

```python
v = 1
i = 1
while i <= 5:
    v = v * i
    i += 1
```
A question about *for* loops

```python
for value in [1, 16, 4]:
    print(value)
    value = value * 10
```

(for_quirk.py)
Last time: Turtles!
Last time: Turtles!
Creating and Using Objects

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

What is this about?

No new syntax here:
We import a module called `turtle` that has a function called `Turtle`
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 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 a `constructor` that creates (and returns) new `objects` of type `Turtle`.

The variable `scott` now refers to a newly created `Turtle` object.
Creating and Using Objects

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

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

The variable `scott` now refers to a newly created `Turtle` object.

what **is** an object? what can it **do**?
Creating and Using Objects

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

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

The variable `scott` now refers to a newly created `Turtle` object.

what **is** an object? what can it **do**?

(whiteboard: diagram of assignment statement)
Creating and Using Objects

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

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

Objects can have functions associated with them, accessed via the dot notation:
Creating and Using Objects

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

# move the turtle forward 10 units:
scott.forward(10)
# turn the turtle left 90 degrees:
scott.left(90)
```

Objects can have functions associated with them, accessed via the dot notation:
Creating and Using Objects

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

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

```python
# move the turtle forward 10 units:
scott.forward(10)
# turn the turtle left 90 degrees:
scott.left(90)
```

*functions that belong to an object are called its methods*
Creating and Using Objects

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

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

```python
# move the turtle forward 10 units:
scott.forward(10)
# turn the turtle left 90 degrees:
scott.left(90)
```

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

What methods do Turtles have? Lots!
Check the docs:
https://docs.python.org/3.3/library/turtle.html?highlight=turtle
import random
num = random.randint(0,9)

import turtle
scott = turtle.Turtle()
scott.forward(100)
import a module

```
import random
num = random.randint(0, 9)
```

```
import turtle
scott = turtle.Turtle()
scott.forward(100)
```
Modules vs Objects

import a module

import random
num = random.randint(0, 9)

call one of its functions

import turtle
scott = turtle.Turtle()
scott.forward(100)
Modules vs Objects

import a module → import random

num = random.randint(0, 9)

call one of its functions → num = random.randint(0, 9)

import a module

import turtle

scott = turtle.Turtle()

scott.forward(100)
Modules vs Objects

import a module → import random

call one of its functions → num = random.randint(0, 9)

import a module

import turtle

call one of its functions

which creates an object

scott = turtle.Turtle()

scott.forward(100)
import a module
call one of its functions

import random
num = random.randint(0, 9)

import a module
import turtle
which creates an object
call one of that object's methods

scott = turtle.Turtle()
scott.forward(100)
import a module

call one of its functions

**import** random

num = random.randint(0,9)

call one of its functions

**import** turtle

scott = turtle.Turtle()

which creates an object

call one of that

object's methods

call one of its functions

Demo: make more than one 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:
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
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
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)
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?
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_for.py`: Create a turtle and draw a square with a for loop
- `turtle_while.py`: Create a turtle and draw a square with a while loop
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?
Functions, Revisited
Functions, Revisited

- We’ve been using functions since day 1:
Functions, Revisited

• We’ve been using functions since day 1:

print("Hello, World!")
Functions, Revisited

• We’ve been using functions since day 1:
  
  \texttt{print("Hello, World!")}

• Built-in functions so far: 
  \texttt{print, input, type, len, int, str, ...}
Functions, Revisited

• We’ve been using functions since day 1:
  ```python
  print("Hello, World!")
  ```

• Built-in functions so far:
  ```python
  print, input, type, len, int, str, ...
  ```

• We can import more functions:
  ```python
  import math
  import turtle
  math.sqrt(4)
  turtle.Turtle()
  ```
Functions, Revisited

What is a function, anyway?

print("Hello world")
Functions, Revisited

What is a function, anyway?
It’s a chunk of code with a name.

print("Hello world")
Functions, Revisited

What **is** a function, anyway?

It’s a chunk of code with a name.
- It *may* take **arguments** as input

```python
print("Hello world")
```
Functions, Revisited

What is a function, anyway?
It’s a chunk of code with a name.
• It may take arguments as input
• It may do something that has an effect

print("Hello world")
Functions, Revisited

What is a function, anyway?

It’s a chunk of code with a name.

• It may take arguments as input
• It may do something that has an effect
• It may return a value

\texttt{print("Hello world")}
Functions, Revisited

What is a function, anyway?

It’s a chunk of code with a name.

- It *may* take arguments as input
- It *may* do something that has an effect
- It *may* return a value

print("Hello world")

→ print →
Functions, Revisited

What is a function, anyway?
It’s a chunk of code with a name.
• It may take arguments as input
• It may do something that has an effect
• It may return a value

print("Hello world")

Input(s):
• 0 or more values
• (optional) sep and end keywords
Functions, Revisited

What is a function, anyway?

It’s a chunk of code with a name.
• It may take arguments as input
• It may do something that has an effect
• It may return a value

```python
print("Hello world")
```

Input(s):
• 0 or more values
• (optional) sep and end keywords

Return value:
• none
Functions, Revisited

What is a function, anyway?
It’s a chunk of code with a name.
• It may take arguments as input
• It may do something that has an effect
• It may return a value

```python
print("Hello world")
```

Input(s):
• 0 or more values
• (optional) sep and end keywords

Return value:
• none

Effects: prints arguments to the screen, with given separator and end
Functions, Revisited

What **is** a function, anyway?

It’s a chunk of code with a name.
- It *may* take **arguments** as input
- It *may* do something that has an effect
- It *may return* a value

```python
input("Enter a number:")
```

**Input(s):**
- none, or
- a string to print as a prompt

**Return value:**
- the input from the user

**Effects:** prompts for user input and reads it from the keyboard
Functions, Revisited

What is a function, anyway?

It’s a chunk of code with a name.
• It may take arguments as input
• It may do something that has an effect
• It may return a value

\[ \text{type}(6/2) \]

Input(s):
• a value

Return value:
• the type of the value

Effects: none
Functions, Revisited

What is a function, anyway?

It’s a chunk of code with a name.
• It may take arguments as input
• It may do something that has an effect
• It may return a value

```
math.sin(math.pi/2)
```

Input(s):
• a number

Return value:
• the sine of the value

Effects: none
Functions, Revisited

What is a function, anyway?

It’s a chunk of code with a name.
• It may take arguments as input
• It may do something that has an effect
• It may return a value

Input(s):
• a number

Return value:
• none

Effects: moves the turtle forward by the given number of units
Functions, Revisited

What *is* a function, anyway?

Input(s) → (Effects) → Return value
Functions, Revisited

What **is** a function, anyway?

- So far we’ve treated functions as “**black boxes**”, code someone else wrote that does stuff for us.

```
Input(s) ➔ □ □ □ □ □ ➔ Return value
                (Effects)
```
Functions, Revisited

What **is** a function, anyway?
- So far we’ve treated functions as “**black boxes**”, code someone else wrote that does stuff for us.
- All we know are the inputs, effects, and return value.

\[
\text{Input(s)} \rightarrow \boxed{\text{black box}} \rightarrow \text{Return value}
\]

\[
\text{(Effects)}
\]
Functions, Revisited

What is a function, anyway?

- So far we’ve treated functions as “black boxes”, code someone else wrote that does stuff for us.
- All we know are the inputs, effects, and return value.
- We don’t know how it’s done.

\[\text{Input(s)} \rightarrow \text{(Effects)} \rightarrow \text{Return value}\]
Functions, Revisited

What is a function, anyway?

- So far we’ve treated functions as “black boxes”, code someone else wrote that does stuff for us.
- All we know are the inputs, effects, and return value.
- We don’t know how it’s done.

\[
\text{Input(s)} \rightarrow \text{Return value}
\]

\[
\text{(Effects)}
\]

This is a great situation to be in!
Functions, Revisited

What is a function, anyway?
• So far we’ve treated functions as “black boxes”, code someone else wrote that does stuff for us.
• All we know are the inputs, effects, and return value.
• We don’t know how it’s done.

Input(s) → [Black Box] → Return value
(Effects)

This is a great situation to be in!

A bunch of (complicated), powerful stuff is wrapped up in a nice, easy-to-use package.
What if

You want a nice easy-to-use function that does something complicated, but *nobody else* has written it for you…
What if

You want a nice easy-to-use function that does something complicated, but nobody else has written it for you…
What if

You want a nice easy-to-use function that does something complicated, but nobody else has written it for you…

Soon, you will have the power to write your own functions.
What if

You want a nice easy-to-use function that does something complicated, but nobody else has written it for you…

Soon, you will have the power to write your own functions.
Writing Functions: Syntax

```python
def name(parameters):
    statements
```
Writing Functions: Syntax

```python
def name(parameters):
    statements
```

Two important questions:
Writing Functions: Syntax

def name(parameters):
    statements

Two important questions:
1. How does the function use the arguments (inputs) passed to it?
Writing Functions: Syntax

```python
def name(parameters):
    statements
```

Two important questions:
1. How does the function use the arguments (inputs) passed to it?
2. How does the function return a value?
Writing Functions: Syntax

```python
def name(parameters):
    statements
```

Two important questions:
1. How does the function use the arguments (inputs) passed to it?
2. How does the function return a value?

Let's **dodge** these questions for a moment…
Functions: the simplest kind

No arguments, no return value:

```python
def name():
    statements
```

Example:

```python
def print_hello():
    print("Hello, world!")
```
Demo

- hello_fn.py
Demo

- print_hello
- definition does nothing except make the function exist
- call it
- can call it whenever/however many times
- can’t call it before it’s defined
Demo: Function to print a rectangle of # symbols

Input(s):
• none

Return value:
• none

Effects: prints a 2x50 rectangle of #s to the screen

print_rectangle
Demo: Function to print a rectangle of # symbols

- executing a def statement (function definition) has no effect except defining that function.

- after it is defined, a function can be used whenever and wherever in the program.

- modify to ask user what character to print
Writing Functions: Syntax

```python
def name(parameters):
    statements
```

Two important questions:
1. How does the function use the arguments (inputs) passed to it?
2. How does the function return a value?

Let’s dodge these questions for a moment…
Writing Functions: Syntax

1. How does the function use the arguments (inputs) passed to it?

```python
def name(parameters):
    statements
```
Writing Functions: Syntax

1. How does the function use the arguments (inputs) passed to it?

```python
def keyword function name:
  def name(parameters):
    statements
```

- def keyword
- function name
- parameters: comma-separated list of variable names that will refer to the input arguments
Demo: Function to print a rectangle of a symbol passed in as an argument.

**Input(s):**
- character to make a rectangle out of

**Return value:**
- none

**Effects:** prints a 2x50 rectangle of the given character to the screen
Writing Functions: Syntax

1. How does the function use the arguments (inputs) passed to it?

```python
def keyword function name

def name(parameters):
    statements
```
Writing Functions: Syntax

1. How does the function use the arguments (inputs) passed to it?

```
def name(parameters):
    statements
```

- `def` keyword
- Function name
- Comma-separated list of parameters: variable names that will refer to the input arguments
- Inputs
Writing Functions: Syntax

1. How does the function use the arguments (inputs) passed to it?

```
def keyword function name:
    statements
```

Inside the function, the parameters act as **local variables** that refer to the arguments passed into the function.

Inputs:
- Comma-separated list of parameters: variable names that will refer to the input arguments
Demo: Function to draw a square using a turtle