Classes, Types, and Objects

Lecture 6

npm run pull … npm run start … pollev.com/comp110

Announcements

- PS1 – Due Friday
  - Full EC deadline is TODAY 11:59pm
  - Partial EC on Thurs
  - Due Friday at 11:59pm

- Review Session – Today 5-6pm
- Small Group Review Walk-in Hours FB008 – Thursday & Friday 2-6pm

Office Hours

- Located on the 0th floor of Sitterson Hall – Room 008

- 178 students have come to office hours in the past week

- Median wait time has been under 5 minutes – often *immediately*…
  …still better than Uber this semester.

- The UTA team is here to help you learn the material that does not make sense in class!

- **Reminder**: UTAs are the only authorized collaborators on problem sets.

Warm-up #1: What is the output of these two programs?

Left

```
import "introcs";

function main(): void {
   let x: number = 50;
   if (x > 0) {
      print("A");
   } else if (x === 50) {
      print("B");
   }
}

main();
```

Right

```
import "introcs";

function main(): void {
   let x: number = 50;
   if (x > 0) {
      print("A");
   }
   if (x === 50) {
      print("B");
   }
}

main();
```

Warm-up #1: What is the output of these two programs?

Left: A

```
import "introcs";

function main(): void {
    let x: number = 50;
    if (x > 0) {
        print("A");
    } else if (x === 50) {
        print("B");
    }
}

main();
```

Right: A, B

```
import "introcs";

function main(): void {
    let x: number = 50;
    if (x > 0) {
        print("A");
    }
    if (x === 50) {
        print("B");
    }
}

main();
```

Warm-up #2:
Fill in the blank with the name of the function that can be called here.

```
function main(): void {
    let result: string;
    print("result is:");
    result = ___________(1, 2);
    print(result);
}

function a(x: number): number {
    return x + x;
}

function b(x: number, y: number): number {
    return x + y;
}

function c(x: number): string {
    return "x is " + x;
}

function d(x: number, y: number): string {
    return "x is " + x + " y is " + y;
}
```

Warm-up #2:
Fill in the blank with the name of the function that can be called here.

Answer: **d**

```
function main(): void {
    let result: string;
    print("result is:");
    result = ___________(1, 2);
    print(result);
}

function a(x: number): number {
    return x + x;
}

function b(x: number, y: number): number {
    return x + y;
}

function c(x: number): string {
    return "x is " + x;
}

function d(x: number, y: number): string {
    return "x is " + x + " y is " + y;
}
```

Function Definition vs. Calling

Each function definition tells you *how* to call the function (name and parameters) and *where* you can call the function (return type).

Definition

```
function clear(): void

function print(input: string): void

function random(lo: number, hi: number): number
```

Calling

```
clear();

print("Hello");

let x: number;
x = random(1, 2);
```

Warm-up #3 – What is printed to the screen when this program runs?

```
import "introcs";

function main(): void {
    let a: string;
    a = b();
}

function b(): string {
    print("c");
    return "d";
}

main();
```

Warm-up #3 – What is printed to the screen when this program runs?

```
import "introcs";

function main(): void {
    let a: string;
    a = b();
}

function b(): string {
    print("c");
    return "d";
}

main();
```

Answer: **c**

Why are **data types** important?

- Types communicate expectations and capabilities in our programs.

- Take the following variables, for example:

```
let x: number;
let y: number;
```

- What can we *do* with **x** and **y**?
  - Assign number literals to each, i.e. 110 and 3.14
  - Add them together, perform arithmetic
  - Pass them to a function that accepts number parameters
  - Generally: use them anywhere we can use a *number **expression***

- How do we know we can do those things? Because their type is **number**.
  - The operations we can carry out on *string* or *boolean* variables are different.

We can **define our own data types**!

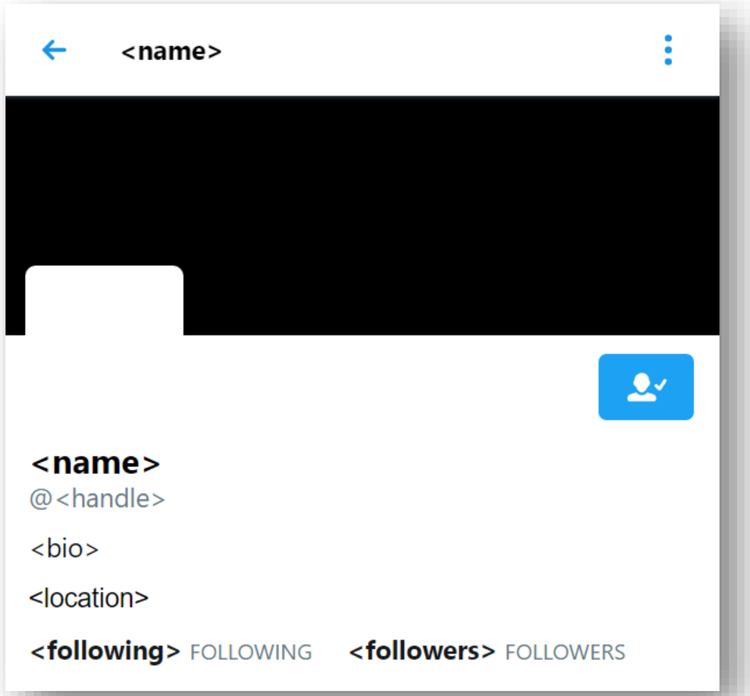
- What if we want to "model" more complex concepts:
  - Twitter Profiles
  - Pizza Order
  - Football Player Stats

- We can invent our own *composite data types* out of other *types*
  - Like in chemistry where *compounds* are made of *atoms*…
  - `number`, `string`, and `boolean` are our atoms

- We define a data type with a **class**

- We call instances of a class **objects**

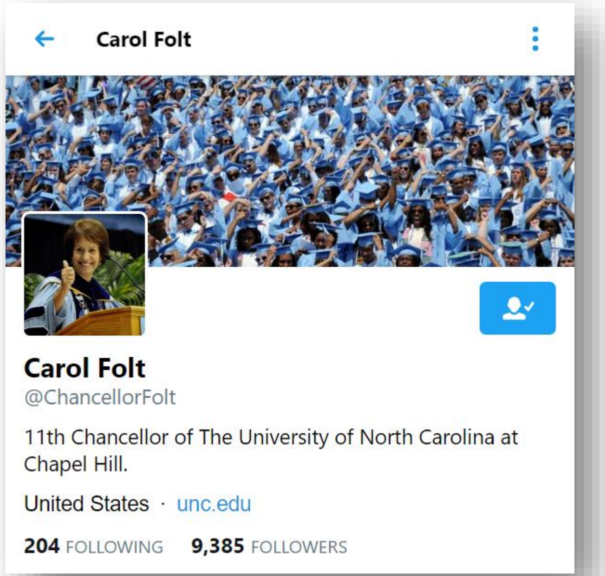
What is this?



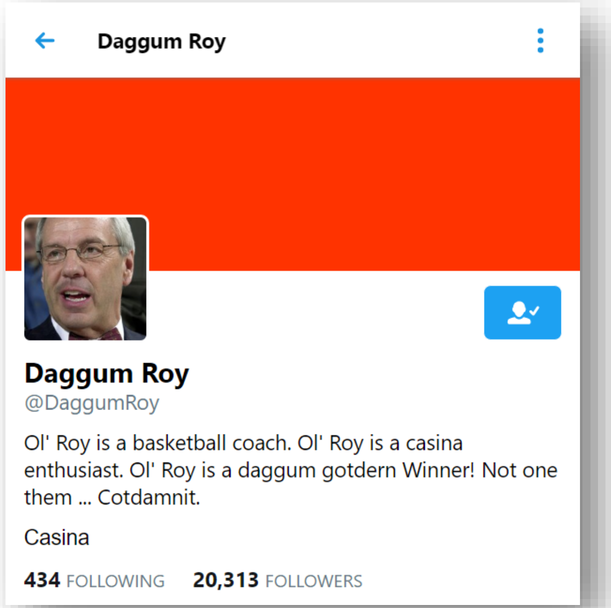
A Class!

- That's a lie. Classes *aren't actually* visual templates.
- They're code specifications of a *type* of object.
- However, this is a useful analogy:

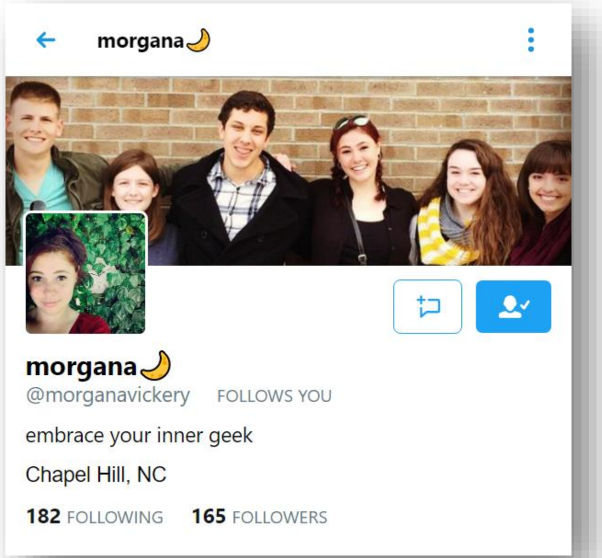
`Class : Object :: A Twitter Profile : @KrisJordan's Profile`


Carol Folt

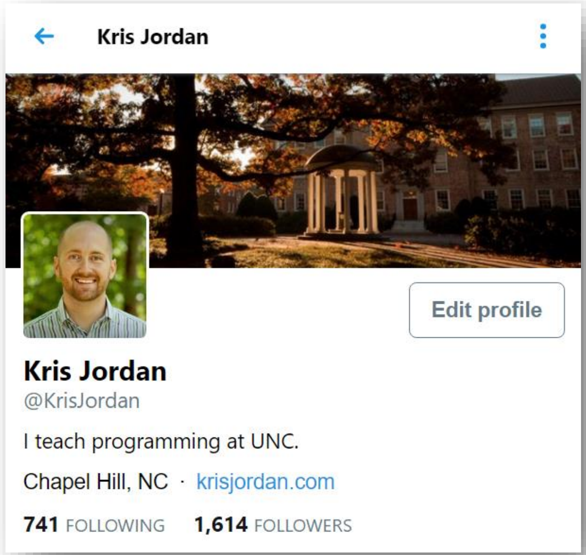
Carol Folt
@ChancellorFolt
11th Chancellor of The University of North Carolina at Chapel Hill.
United States · unc.edu
204 FOLLOWING 9,385 FOLLOWERS


Daggum Roy

Daggum Roy
@DaggumRoy
Ol' Roy is a basketball coach. Ol' Roy is a casina enthusiast. Ol' Roy is a daggum gotdern Winner! Not one them ... Cotdamnit.
Casina
434 FOLLOWING 20,313 FOLLOWERS


morgana🌙

morgana🌙
@morganavickery FOLLOWS YOU
embrace your inner geek
Chapel Hill, NC
182 FOLLOWING 165 FOLLOWERS

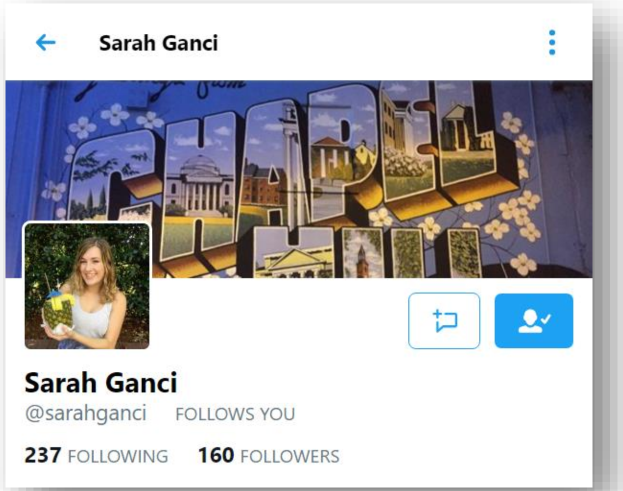

Kris Jordan

Edit profile
Kris Jordan
@KrisJordan
I teach programming at UNC.
Chapel Hill, NC · krisjordan.com
741 FOLLOWING 1,614 FOLLOWERS

What are these?

Objects!

(They're all Twitter profiles.)


Sarah Ganci

Sarah Ganci
@sarahganci FOLLOWS YOU
237 FOLLOWING 160 FOLLOWERS

How would we model a TwitterProfile in code?

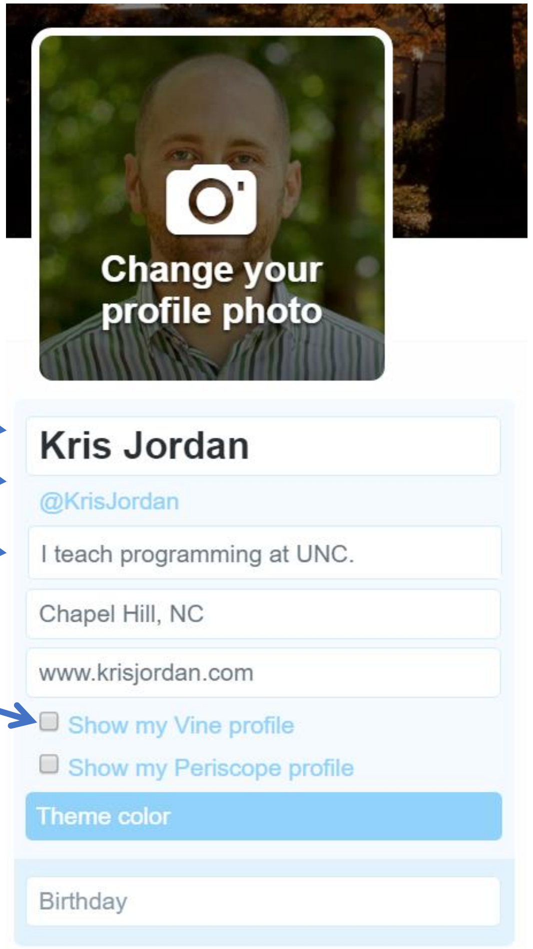
- We have not covered the concepts in the code below, yet, but I'll bet you can connect a few dots.

- These are all *properties* of the TwitterProfile class.

```
class TwitterProfile {

    name: string;
    handle: string;
    bio: string;
    showVine: boolean;
    privateAccount: boolean;
    followers: number;
    following: number;

}
```



Kris Jordan

@KrisJordan

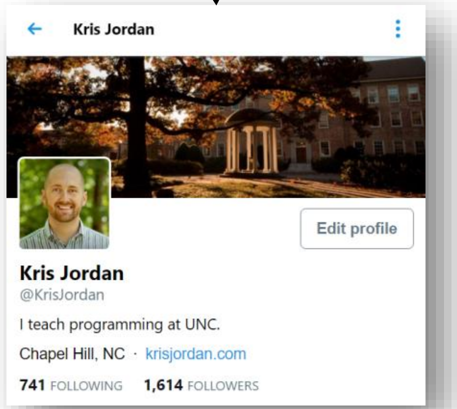
I teach programming at UNC.

Chapel Hill, NC

www.krisjordan.com

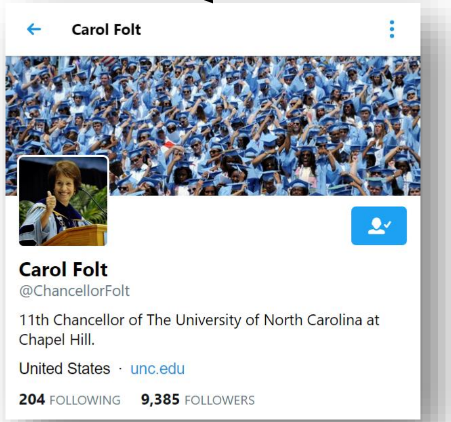
Each **Object's** properties are established by its **Class**

```
class TwitterProfile {
    name: string;
    handle: string;
    bio: string;
    showVine: boolean;
    privateAccount: boolean;
    followers: number;
    following: number;
}
```

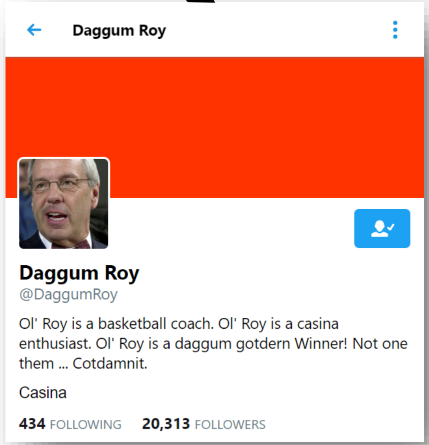

Kris Jordan

Edit profile

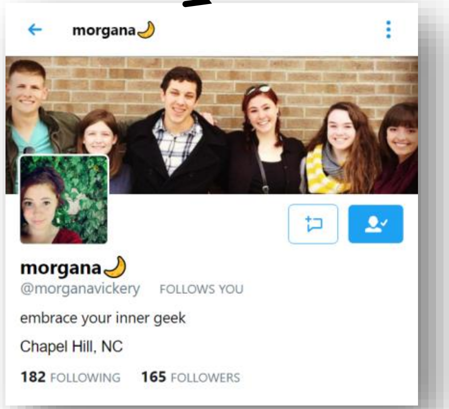
Kris Jordan
@KrisJordan
I teach programming at UNC.
Chapel Hill, NC · krisjordan.com
741 FOLLOWING 1,614 FOLLOWERS


Carol Folt

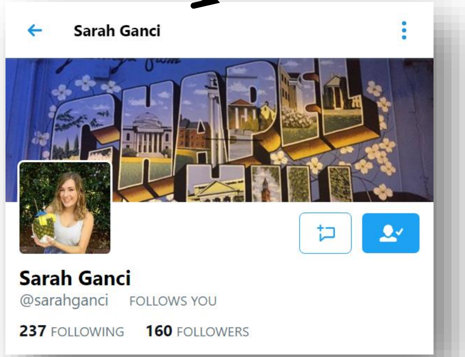
Carol Folt
@ChancellorFolt
11th Chancellor of The University of North Carolina at Chapel Hill.
United States · unc.edu
204 FOLLOWING 9,385 FOLLOWERS


Daggum Roy

Daggum Roy
@DaggumRoy
Ol' Roy is a basketball coach. Ol' Roy is a casina enthusiast. Ol' Roy is a daggum gotdern Winner! Not one them ... Cotdamnit.
Casina
434 FOLLOWING 20,313 FOLLOWERS


morgana🌙

morgana🌙
@morganavickery FOLLOWS YOU
embrace your inner geek
Chapel Hill, NC
182 FOLLOWING 165 FOLLOWERS


Sarah Ganci

Sarah Ganci
@sarahganci FOLLOWS YOU
237 FOLLOWING 160 FOLLOWERS

Defining a Class - *"Inventing a Data Type"*

```
class <ClassName> {
   <propertyName>: <type>;
   <propertyName>: <type> = <default value>;
   …
}
```

- **ClassNames** begin with an uppercase letter
- **Properties** are declared inside of the class body
  - These are *like* variable declarations without the *let* keyword
  - Properties can be assigned default values
- "Every object of <ClassName> will have a <name> property of type <type>".
  - *"Every TwitterProfile will have a followers field of type number"*

Defining a Class - Example

- Here we are defining a class named Rectangle.

- ***Every*** `Rectangle` object will have three properties:
    - color, width, height

- In defining a class, we've invented a new type! We can now use it in our programs anywhere use a type. For example, a variable declaration:

```
class Rectangle {
   color: string = "Black";
   width: number = 1.0;
   height: number = 1.0;
}
```

```
let paper: Rectangle;
```

Hands-on #1: Define a Rectangle class

- Open lec06-objects / 00-object-fundamentals-app.ts
- At TODO #1, before the main function, define the Rectangle class to the right.
- At TODO #2, in the main function, declare a variable of type Rectangle named **origami**.
- Check-in on PollEv.com/comp110 once you've got this code.

```
class Rectangle {
    color: string = "Black";
    width: number = 1.0;
    height: number = 1.0;
}
```

```
let origami: Rectangle;
```

Constructing an Object

```
let paper: Rectangle;
```

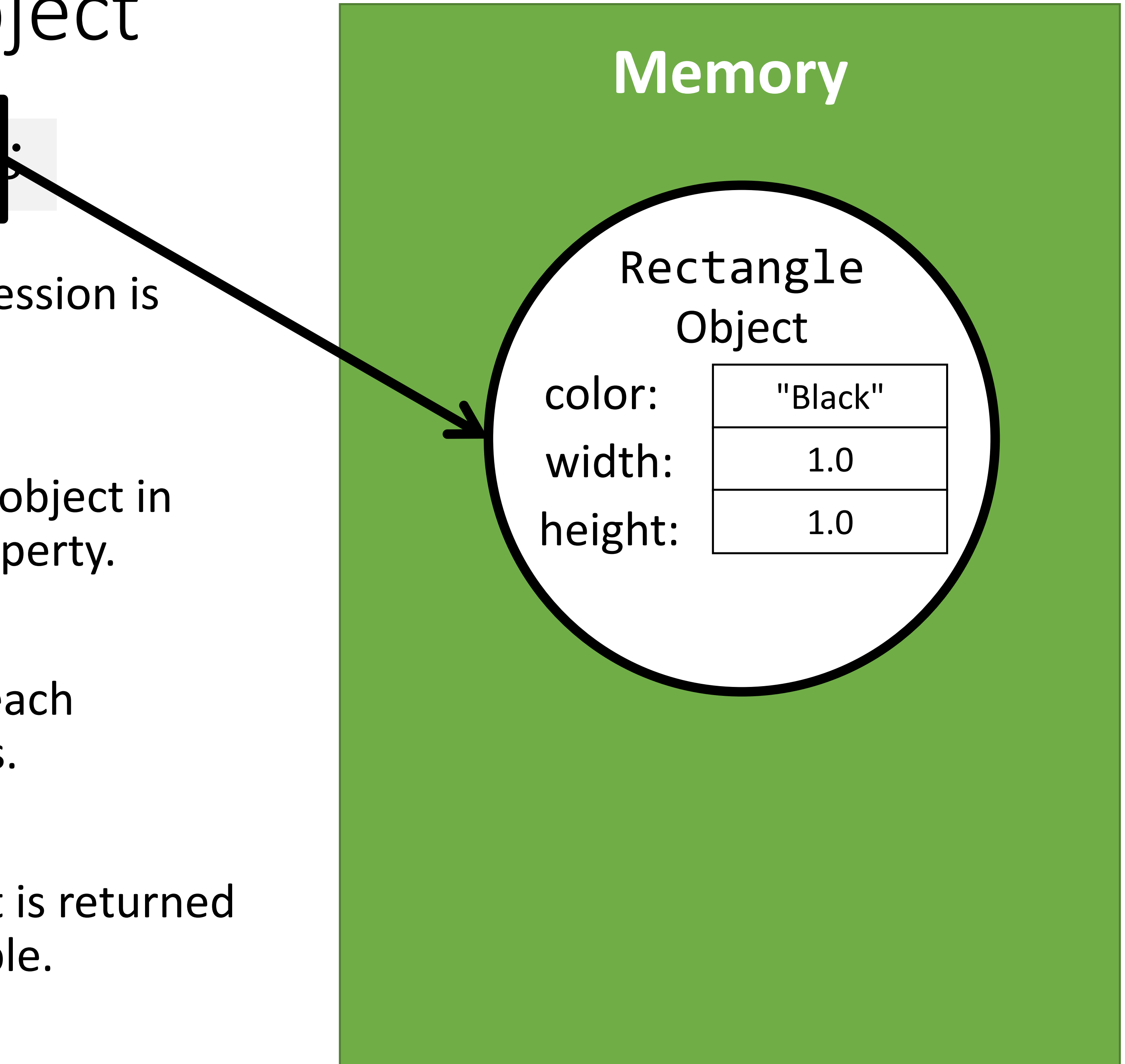
```
paper = new Rectangle();
```

- How do we initialize this variable?
- Unlike simple data types, where declaring a variable reserves memory for it, composite types or "object types" require us to "**construct a new object**" in memory.
- We do this using the new keyword, followed by the class' name, followed by empty parenthesis (for now).

Constructing an Object

```
paper = new Rectangle();
```

- When the **new Rectangle()** expression is reached,
- The processor **constructs** a new object in memory with space for each property.
- It also assigns default values to each property, if specified by the class.
- Finally, **a reference** to this object is returned and assigned to the paper variable.
  - More on *references* soon.



Hands-on #2: Constructing a Rectangle Object

1. Still in lec06-objects / 00-object-fundamentals-app.ts
2. After origami's variable declaration, initialize it by constructing a new Rectangle

```
origami = new Rectangle();
```

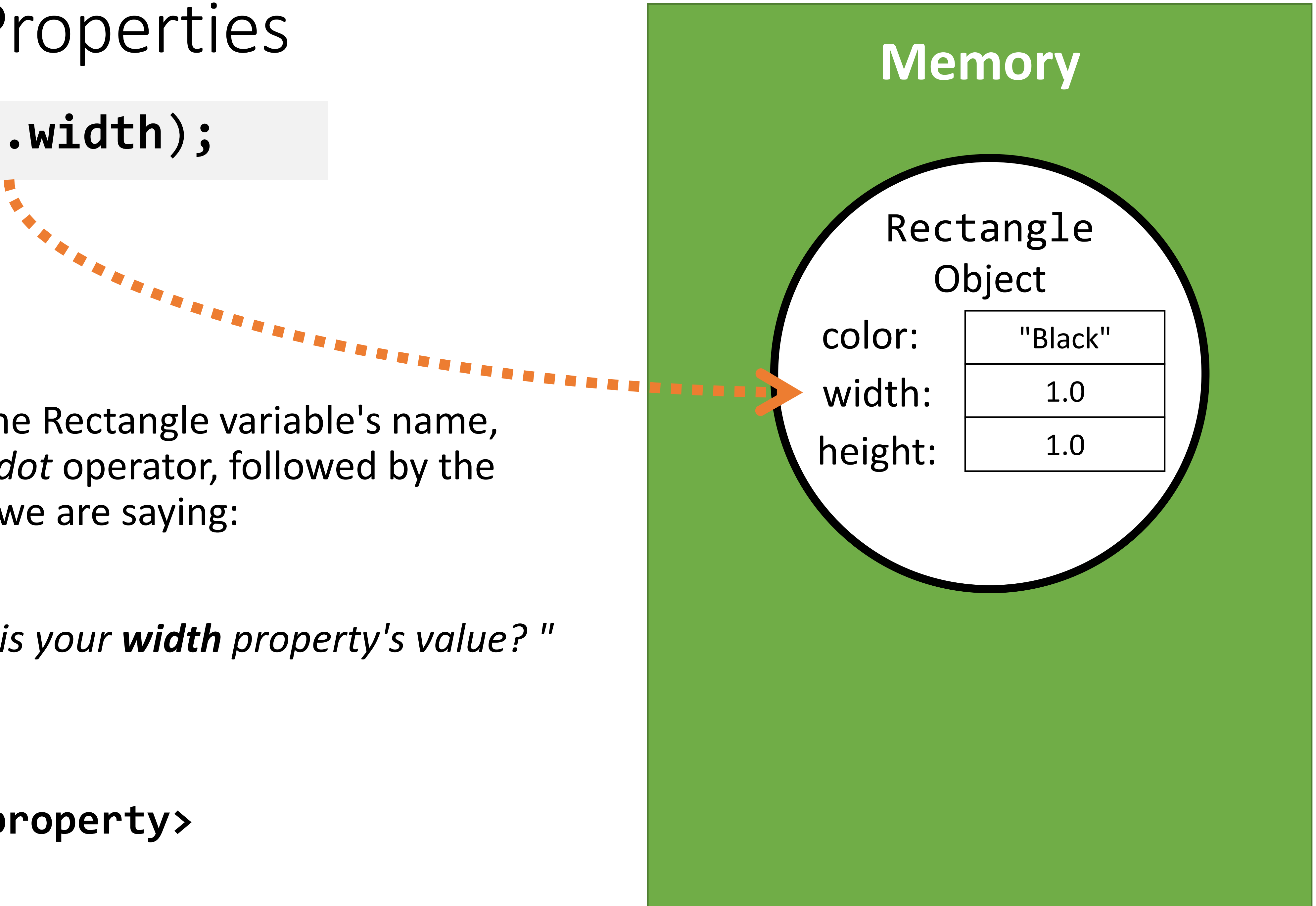
3. Uncomment the **rectToString** function below TODO #3.
4. Then try calling it from inside the main function and printing the return value.

```
print(rectToString(origami));
```

5. Check-in on PollEv.com/comp110 once you've got "Black Rectangle is 1.0x1.0" printing in your browser.

Reading Properties

```
print(paper.width);
```



- By referencing the Rectangle variable's name, followed by the *dot* operator, followed by the property name, we are saying:

*"Hey **paper**, what is your **width** property's value? "*

- General form:

  `<object>.<property>`

Assigning to Properties

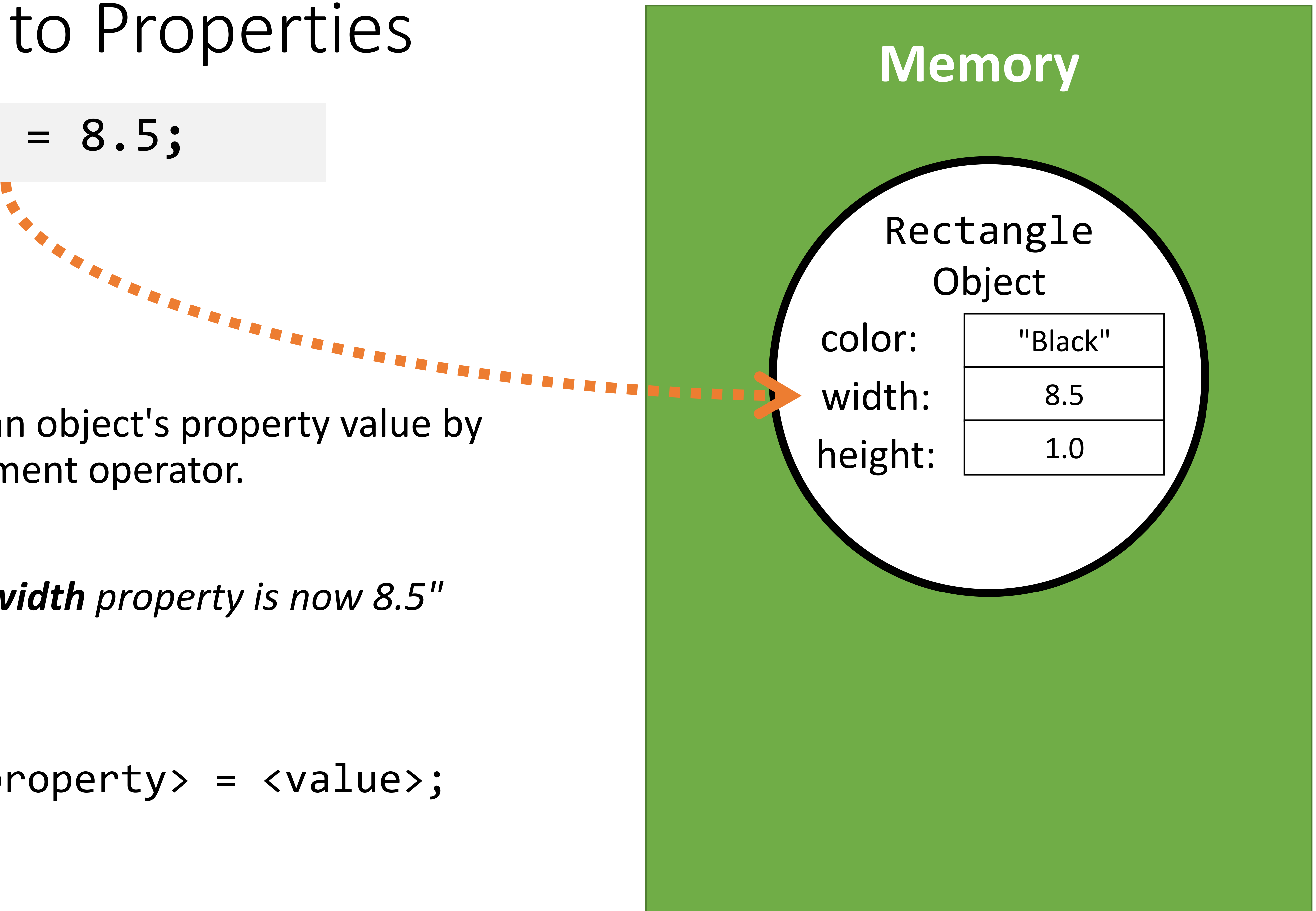
```
paper.width = 8.5;
```

- We can change an object's property value by using the assignment operator.

*"Hey **paper**, your **width** property is now 8.5"*

- General form:

```
<object>.<property> = <value>;
```



Follow-along

- Reading and writing properties.
- Let's change the origami's rectangle dimensions to be 3.0 x 3.0
- Let's also declare another Rectangle variable named standard and change its dimensions to be 8.5 x 11.0

Hands-on #3: Call and Implement the **area** function.

1. From the main function, call the area function using the origami Rectangle as an argument. Print the returned value. (It should print 0 initially.)

2. At TODO #4 - given a Rectangle parameter named input, rather than returning 0, try returning the input Rectangle's height multiplied by its width.

3. Check-in on PollEv.com/comp110 when correctly printing area.

```
print(area(origami));
```

```
function area(input: Rectangle): number {
        return input.width * input.height;
}
```

Hands-on #4: Pizza Pricing Calculator

- Your goal: implement the price functionality underneath the TODO in 01-pizza-price-app.ts

- Pricing logic for you to try implementing with a series of if-then-else statements:
  - Base cost is based on **size** property:
    - small is $7
    - medium is $9
    - any other size is $11
  - If **extraCheese** property is true, it adds $1 to the cost
  - For each # of **toppings,** add $0.75

- You should be increasing the value stored in the cost variable.
  - Hint: cost = cost + 1.0; // This will increment cost by 1

- Test by updating properties of pizza0.

- Check-in on PollEv.com/comp110 when your pricing is printing.

- Done? Try constructing another pizza in your main function.

Be Careful to **Always Initialize your Variables**

- Uncaught TypeError: Cannot set property '<property>' of undefined
- This happens when you try to use an object variable that hasn't been assigned an object.
- For example:
- let pizza1: Pizza;
- pizza1.size = "large"; // ERROR!!! pizza1 has not been initialized