0. What is the output? pollev.com/compunc

```
let a: number = 10;
let b: number = a;
a = 20;
print("a");
print(b);
```

1. What is the output when this program runs? pollev.com/compunc

```
import { print } from "introcs";

let foo = (a: number, b: number): string => {
   if (a >= b) {
      return "Carol";
   } else {
      return "Folt";
   }
};

export let main = async () => {
   print(foo(10, 300));
};

main();
```

2. What is the output when the main function runs? pollev.com/compunc

```
let bar = (a: number, b: number): number => {
   if (a === b) {
      return a + b;
   }
   return b;
};

export let main = async () => {
   let result: number;
   result = bar(10, 10);
   result = result + bar(20, 10);
   print(result);
};
```

3. Which identifiers are used as arguments? Parameters?

pollev.com/compunc

```
import { print } from "introcs";

let f = (a: number): number => {
   return 3 * a;
};

export let main = async () => {
   let b: number = 10;
   let result: number = f(b);
   print(result);
};

main();
```

Expressions

- **Expressions** are a fundamental building block in programs

- Expressions are analogous to the idea of clauses in English
  - Single clause sentence:
    "I am a student."
  - Multiple clause sentence:
    "I am a student and I am currently sitting in COMP110."
  - In English, *Sentences* are *more expressive* through the creative use of *clauses*

- In code, ***Statements*** are *more expressive* through creative uses of ***expressions!***

How can we compute the volume of a cube using different expressions?

```
let answer: number;
answer = 3 * 3 * 3;
```

1. We can "hard-code" the expression with exact numbers.

How can we compute the volume of a cube using different expressions?

```
let answer: number;
let length: number = 3;
answer = length * length * length;
```

2. We can use a variable to hold the length of a side of the cube.

Notice, in doing so, our *expression* has more meaning:
`length * length * length` is more expressive than `3 * 3 * 3`

How can we compute the volume of a cube using different expressions?

```
let answer: number;
let length: number = await promptNumber("Length:");
answer = length * length * length;
```

3. We can use the **`promptNumber`** function to allow *any number*!

Our program is more generally useful.

How can we compute the volume of a cube using different expressions?

```
let cubeVolume = (side: number): number => {
   return side * side * side;
};
```

```
let answer: number;
let length: number = await promptNumber("Length:");
answer = cubeVolume(length);
```

4. We can write a *function* to compute the volume and *call the function*.

This has two benefits:

1. It reads more naturally: "answer is assigned the result of calculating `cubeVolume` using the given `length`"
2. We can *reuse* the `cubeVolume` function without rewriting the equation!

How can we compute the volume of 2 cubes?

```
let cubeVolume = (side: number): number => {
   return side * side * side;
};
```

```
let answer: number;
let lengthA: number = await promptNumber("Length A:");
let lengthB: number = await promptNumber("Length B:");
answer = cubeVolume(lengthA) + cubeVolume(lengthB);
```

Because the `cubeVolume` function returns a number, we can use its result as parts of larger number expressions.

Consider how much more *expressive* (and less repetitive!) the answer assignment statement is than:

```
answer = (lengthA * lengthA * lengthA) + (lengthB * lengthB * lengthB);
```

Expressions

There are two **big ideas** behind expressions:

1. *Every* expression *simplifies to a single value at runtime*
   - Thus, every expression has a *single type*.
   - This occurs *only* when the program runs (runtime) and when the computer reaches the expression in the program.

2. Anywhere you can write an expression you can substitute any other expression *of the same type*

Expressions – Some examples we've seen…

Expression	Resulting Type	Resulting Value	Expression Name
`3`	`number`	3	number Literal
`length`	`number`	The variable's last assigned value.	Variable Reference
`"Length: " + length`	`string`	The string "Length: " concatenated with the last assigned value of length.	String Concatenation, Variable Reference
`length > 3`	`boolean`	`true` if length's last assigned value is greater than **3** - false otherwise	Variable Reference, Inequality Operator

Where have we *used* expressions?

- Assignment operator:

```
let <name>: <type> = <expression of same type>;
```

- We are able to assign *any* of the expressions below because each results in a single *number* value:

```
let x: number = 1;
let y: number = x + 1;
let cubeY: number = y * y * y;
```

- Notice that we are combining *multiple* expressions in the same line.
- After each line completes, the declared variable has a *single* value.

Where else have we *used* expressions?

- if-then statement

```
if (<boolean expression>) {
   // ... elided ...
}
```

- ***Any boolean expression*** can be used as the test expression in an if-then statement
  - `if (age >= 21) { // ...`
  - `let is21: boolean = (await promptNumber("Age")) >= 21;`

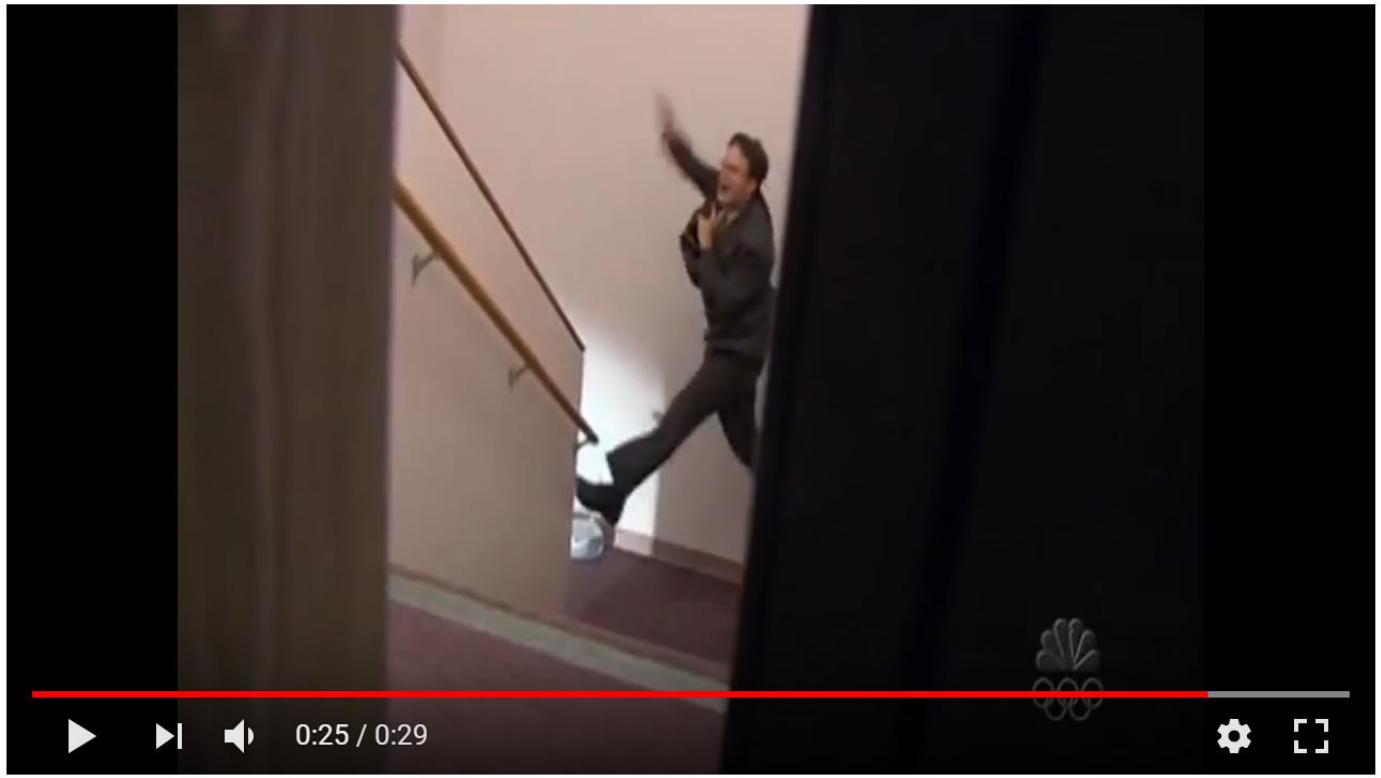
    `if (is21) { // ...`

- When the computer reaches the boolean expression of an if-then statement, it evaluates the expression down to the single value of either **`true`** or **`false`**.

Expressions of Various Kinds

- Literal Values
  - `3.14`
  - `true`
  - `"hi"`
- Variable Access
  - `x`
  - `compCourseNumber`
- "Unary" operators
  - `-x` (number *negation)*
  - `!is21` (boolean negation)
- Function and Method Calls
  - `cubeVolume(x)`
  - `"hello".toUpperCase()`
- "Binary" Operators
  - Arithmetic
    - `1 + 2`
  - Concatenation
    - `"Hello " + name`
  - Equality
    - `x === 1`
    - `x !== 1`
  - Relational
    - `age >= 21`
    - `age < 13`

Dwight's Hype Machine



Dwight is "Awesome"

https://www.youtube.com/watch?v=hkRjL2A41QQ

Hype Machine - Part 1

1. Open lec04 / 01-hype-machine-app.ts

2. At the first TODO comment, declare a function with the following properties:
   - Name: **hypeUp**
   - Parameters:
     1. **name: string**
   - Return Type: **string**

```
let <name> = (<parameters>): <returnType> => {
    <function body statements>
};
```

3. The function should return a string value that concatenates the name parameter to a hype sentence. For example:

```
return name + " IS AWESOME";
```

4. At the second TODO comment, print the result of calling hypeUp with the given name:

```
print(hypeUp(name));
```

5. Save, test, and check-in on PollEv.com/compunc

```
import { print, promptString } from "introcs";

// TODO: define the hypeUp function here
let hypeUp = (name: string): string => {
    return name + " IS AWESOME";
};

export let main = async () => {
    print("Welcome to the Hype Machine");
    let name: string = await promptString("What is your name?");
    // TODO: call the hypeUp function and pass name as an argument
    print(hypeUp(name));
};

main();
```

What if we want it to respond with a random hype up message?

- Let's import a function that generates random numbers for us!

```
import { print, promptString, random } from "introcs";
```

- The random function's definition looks like this…

```
let random = (floor: number, ceiling: number): number => {
    // Magic
};
```

- What do we know just by looking at the definition?

Using the **`random`** Function

```
let random = (floor: number, ceiling: number): number => {
    // …
};
```

- Looking at this definition, we know:
  1. To call `random` we must provide 2 arguments, both `number`s
  2. When `random` returns, it will give us back a `number`

- Thus, we can *call* the random function like so: **`random(1, 6)`**

- The *floor* and *ceiling* parameters represent the smallest and largest numbers **`random`** will choose a number between, inclusive.

How do we generate a random string when all we have is a random number?

- Using **`if-then-else`** statements based on the random number!

```
let choice: number = random(1, 6);
if (choice === 1) {
    return name + " IS AWESOME";
} else {
    if (choice === 2) {
        return name + " IS THE BEST";
    } else {
        return name + " IS GREAT";
    }
}
```

- Let's try this in the next hands-on!

Hands-on: Random Messages

1. Still working in 01-hype-machine-app.ts

2. Add **`random`** to the list of functions being imported from "introcs"

3. In the **`hypeUp`** function, before returning, declare a variable of type number named **`choice`**. Initialize **`choice`** to the result of calling
   `random(1, 3)`

4. Write **`if-then-else`** statements that test if choice is === 1, otherwise if it's === 2, and so on (see previous slide), and return a different message in each case.

5. Save. Check-in on PollEv.com/compunc when your program generates messages at random. Refresh your browser to try multiple times.

```
let hypeUp = (name: string): string => {
    let choice: number = random(1, 3);
    if (choice === 1) {
        return name + " IS AWESOME";
    } else {
        if (choice === 2) {
            return name + " IS GOAT";
        } else {
            return "THE REAL MVP IS " + name + "!!!";
        }
    }
};
```

String Methods - Things We can Ask of **`string`**s

- Expressions of type **`string`** have special capabilities, called methods, that we can utilize to ask questions about the **`string`** data:

  - True or False: Does this string start with the letter "A"?
  - True or False: Does this string end with the string "ily"?
  - True or False: Does this string include the string "ana"?
  - What is the `substring` starting from position 0 and ending at position 3?
    Convert the string to all uppercase characters.
  - Convert the string to all lowercase characters.
  - and more…

Calling a **`string`** Method

- Suppose we have a **`string`** variable named **`s`**:

```
let s: string = "Hello, World";
```

- Calling a **`string`** method is *a lot like* calling a function:

```
let result: string;
result = s.toUpperCase();
```

- The above code says, *"on string **`s`**, call the **`toUpperCase`** method, and assign the returned value to the **`result`** variable."*

- The general syntax is:

```
<string expression>.<method name>(<arguments>);
```

A Few Useful `string` Method Examples

Method	string s' Value	Example	Return Value
startsWith	"jello"	s.startsWith("jell") s.startsWith("jump")	true false
endsWith	"jello"	s.endsWith("ello") s.endsWith("elp")	true false
includes	"jello"	s.includes("el") s.includes("lol")	true false
substr	"jello"	s.substr(0, 3) s.substr(1, 4)	"jel" "ello"
toUpperCase	"JeLlO"	s.toUpperCase()	"JELLO"
toLowerCase	"JeLlO"	s.toLowerCase()	"jello"

Follow-along: Let's Tinker with String Methods

- Open 02-string-methods-app.ts

```
let s: string = "Jello";

print("toUpperCase examples:");
// TODO: Call toUpperCase method
print(s.toUpperCase());

print("toLowerCase examples:");
// TODO: Call toLowerCase method
print(s.toLowerCase());

print("startsWith examples:");
// TODO: Call startsWith method
print(s.startsWith("Jel"));
print(s.startsWith("hel"));

print("endsWith examples:");
// TODO: Call endsWith method
print(s.endsWith("lo"));
print(s.endsWith("je"));

print("includes examples:");
// TODO: Call includes method
print(s.includes("ello"));
print(s.includes("ole"));

print("substr examples:");
// TODO: Call substr
print(s.substr(0, 3));
```