Variables

Intuition

- You're running a fund raiser and want to add up donations received. A friend calls out each donation individually…

  "$11!"
  "$9!"
  "$20!"
  "$30!"
  "Where are we at?"
  "$30!"
  "$4!"
  "$6!"

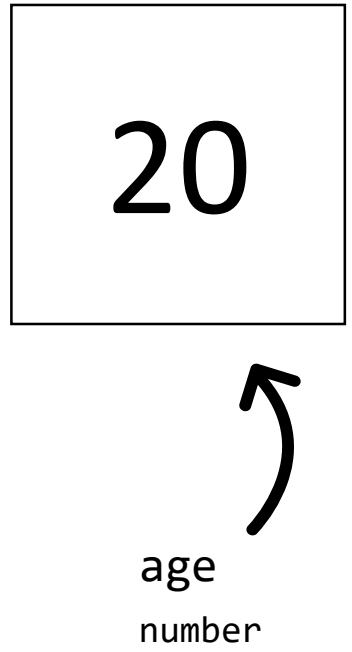
- How did you sum these donations together?

Variables

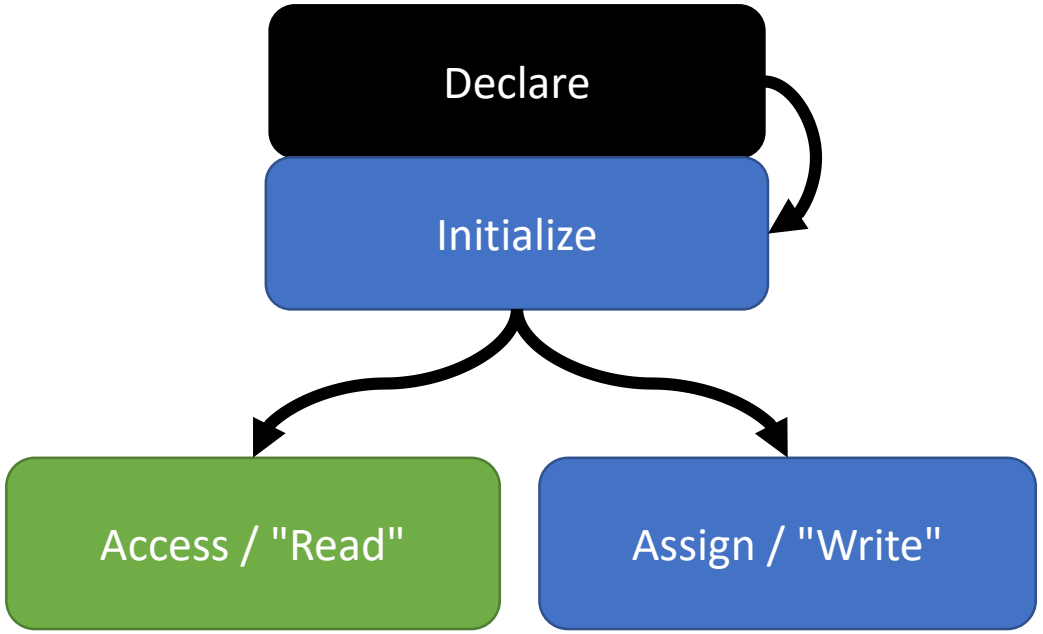
- **Variables** allow your programs to *store, load*, and *change* values in memory.

- *Every* variable:

    1. has a **name** and
    2. holds a value of a specific **data type**



Overview of **how to use a variable**



1. **Declare** the variable by giving it a name
2. **Initialize** the variable with its first value

(Steps 1 and 2 can be combined!)

Once 1 and 2 are done, then you can*:

- **Access** the value stored in a variable, or,
- **Assign** new values to the variable

* There are additional rules governing where you can access and assign a variable from.

Variable Declaration Syntax (1/3)

- When you **declare** a variable, you are proclaiming...
  "henceforth, the identifier <some name> shall refer to a(n) <some type> value stored in memory"

```
let age: number;
```

  - "the identifier **age** shall refer to a **number** value stored in memory."

- General form:

```
let <name>: <type>;
```

- The type can be: **`number, string, boolean`**
  (and more types to come)

Variable Declaration Semantics (2/3)

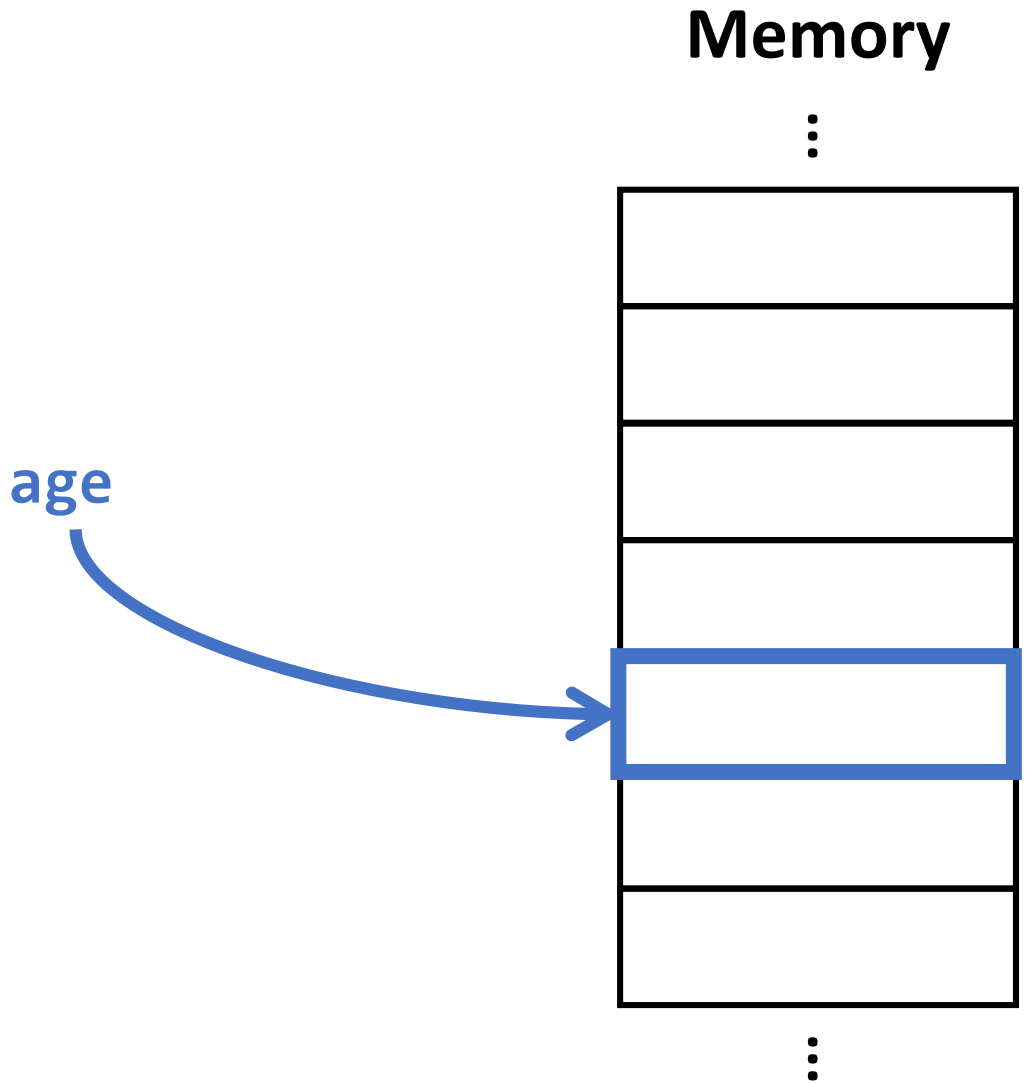
When this statement runs:

```
let age: number;
```

You are reserving a space in your computer's memory that can store a `number` value.

When you later refer to the word **age**, the program knows which exact location in memory it reserved.

Using this name, you can **store, access, and reassign** data in memory!



Variable Naming Rules (3/3)

Variable names cannot contain spaces, must begin with a letter, and contain only letters, numbers, and underscores.

In this course you should use "camelCasing" for multiword names.

For example, if you wanted a variable to store a "year of birth", it would be written as:

```
yearOfBirth
```

To Camel Case remove spaces between words and uppercase *every* subsequent word.

Variable Assignment Syntax (1/3)

- The assignment statement **stores** a value in a variable

```
age = 21;
```

  - "age is assigned a value of 21"
  - "age takes the value of 21"
  - "age is now 21"
  - *Notice:* ***None of these readings uses the word "equals"!***

- General form:

```
<name> = <value>;
```

- The single equal symbol's name is the **assignment operator**.

Variable Assignment Semantics (2/3)

When this line of code runs:

```
age = 20;
```

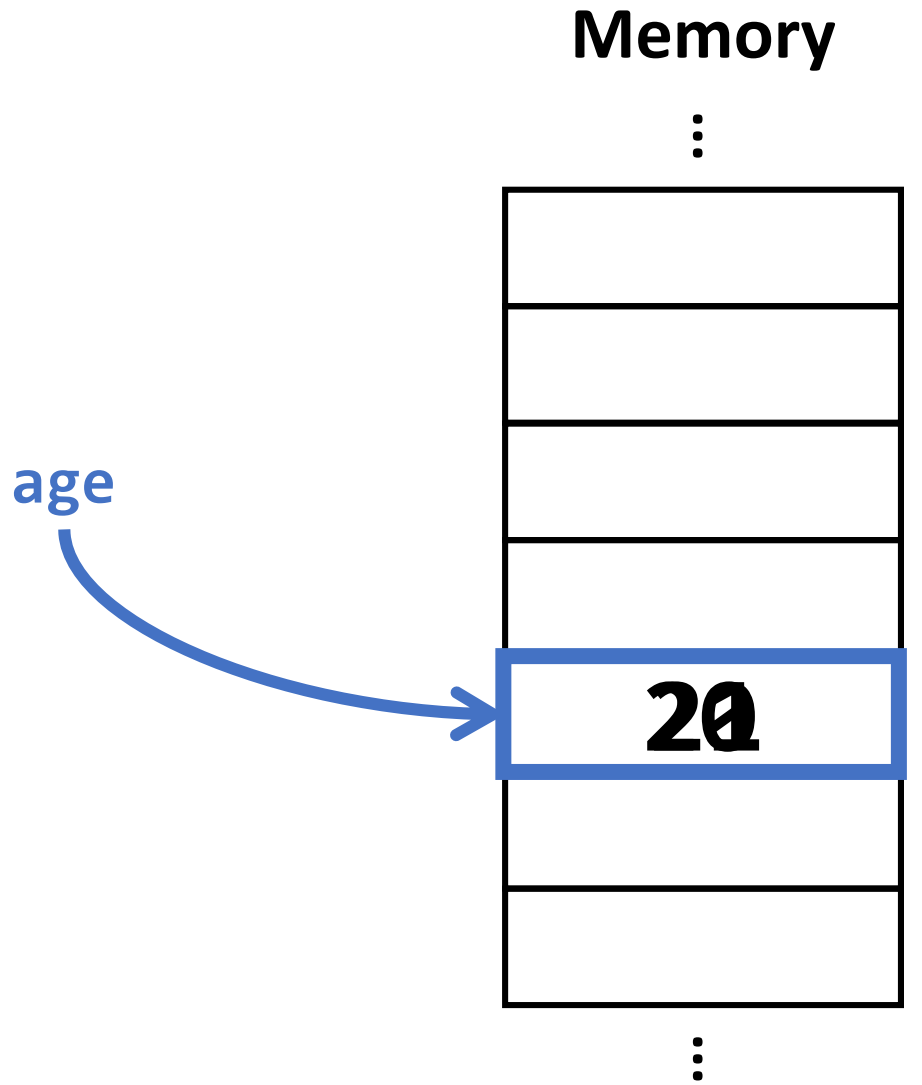
The value 20 is stored in the memory space reserved for the age variable.

Later, if the following line ran:

```
age = 21;
```

The value 21 is stored in the memory space it reserved for the age variable.

Assignment is *not* equality!



Variable Assignment Rules (3/3)

- **A variable's value can change** as the program runs
  - Just assign a another value to the same variable!
  - After an assignment statement runs, when subsequent lines of code run the variable will have the most recently assigned value.

- **The assignment operator <u>is not</u> commutative!**

```
<name> = <value>; // OK
<value> = <name>; // NOT OK
```

  The *variable's name must be on the left* of the assignment operator (=) and *the value being assigned must be on the right*.

- **You cannot refer to a variable until after its declaration.**

- **The value's type must match the variable's declared type**

Variable **Initialization** (1 / 2)

- The *first* time you assign a value to a variable has a special name: **initialization**

- **Always, always, always initialize your variables!**

- You can **declare** *and* **initialize** it in two steps:

```
let lucky: number;
lucky = 13;
```

- Or, you can combine these steps into a single statement:

```
let lucky: number = 13;
```

Variable **Initialization – Type Inference** (2 / 2)

- Notice there is some redundancy in this statement:

```
let lucky: number = 13;
```

- "Let lucky be a ***number*** variable that is initially assigned the ***number*** 13."

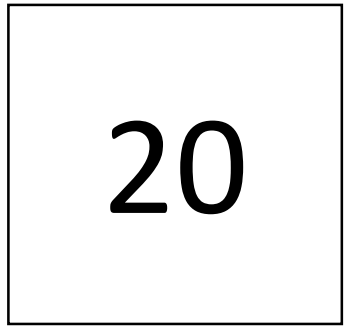
- If you combine declaration and initialization, a modern programming language will ***infer*** the variable's type for you. So you can write:

```
let lucky = 13;
```

- You are encouraged to use type inference when you know a variable's initial value at declaration.

Variable Access – "Read" - Syntax (1/2)

- *After* you have *declared* a variable and initialized it…



- You can **access** ("read", "look up") a variable's value in memory **by name**

```
print(age);
```

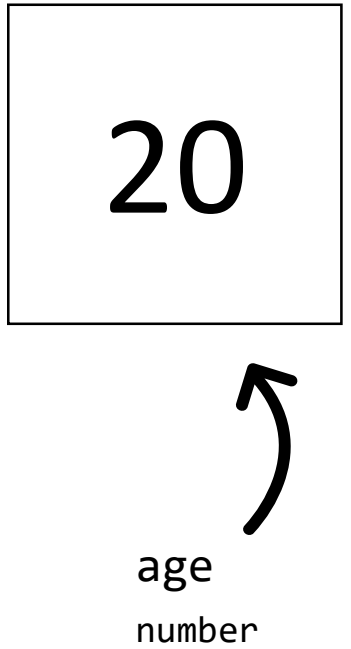
  - "Read *the last value assigned* to **age** and print (output) it to the screen."

- Caution! This is *very different* than: print("age");
  - This would output the word "age" to the screen!

Variable Access in an Assignment Statement (2/2)

- Consider the following assignment statement:


```
age = age + 1;
```



"assign to age the current value of age plus 1."

Steps:

1. current value of **age** is read
2. The number 1 is added to it
3. Result is assigned to **age**'s location in memory

Variable Assignment *is not Equality*

Imagine the following code:

```
1.  print(“Donations”);

2.  let total = 0;

3.  total = total + 11;

4.  total = total + 9;

5.  total = total + 50;

6.  print(total);

7.  total = total + 40;

8.  print("total is " + total);
```

`total`'s value in memory:

```
1.  Undeclared

2.  0

3.  11

4.  20

5.  70

6.  70

7.  110

8.  110
```