hello

about the class

goals

- help you become the best programmer you can be
  - CS136 is the foundation for all the programming you will do after!
- have fun
  - let’s code up some cool stuff from scratch!

what to expect

- the **lectures** should make sense (especially on Monday)
  - if something doesn't make sense, raise your hand!

- the **homework** should take 10-15+ hours per week
  - if it's too hard, let's chat!--we will make a plan for success
    - *"We don't all have the same floor or ceiling, but we each have a lot more in us than we know , and when it comes to [programming], everyone can achieve feats they once thought impossible. –David Goggins*
  - if it's too easy, do the challenge problem!—if that's too easy, let's chat!
    - *"We're either getting better or we're getting worse." –David Goggins*

- the **exams** should be challenging yet also completely unsurprising

week at a glance

- 👨‍🏫 Monday will be a lecture
  - often, i will explain a data structure
- 🧙‍♀️ Wednesday will be a tutorial
  - often, we will implement a data structure together, step by step (bring your laptop!)
- 👨‍🔬 Thursday is lab
  - you work on the homework (definitely bring your laptop!)
- 🦉✨ Friday is time for Kahoot! and advanced topics
  - Kahoot's are meant to prepare you for the exams' shorter questions
  - advanced topics are meant to challenge and entertain you

are you guinea pigs?

- yes
  - however
    - 1) guinea pigs are awesome, and
    - 2) i am very excited to teach this course 🙂👍

difficulty, inclusivity, and programming and a lifetime of coding

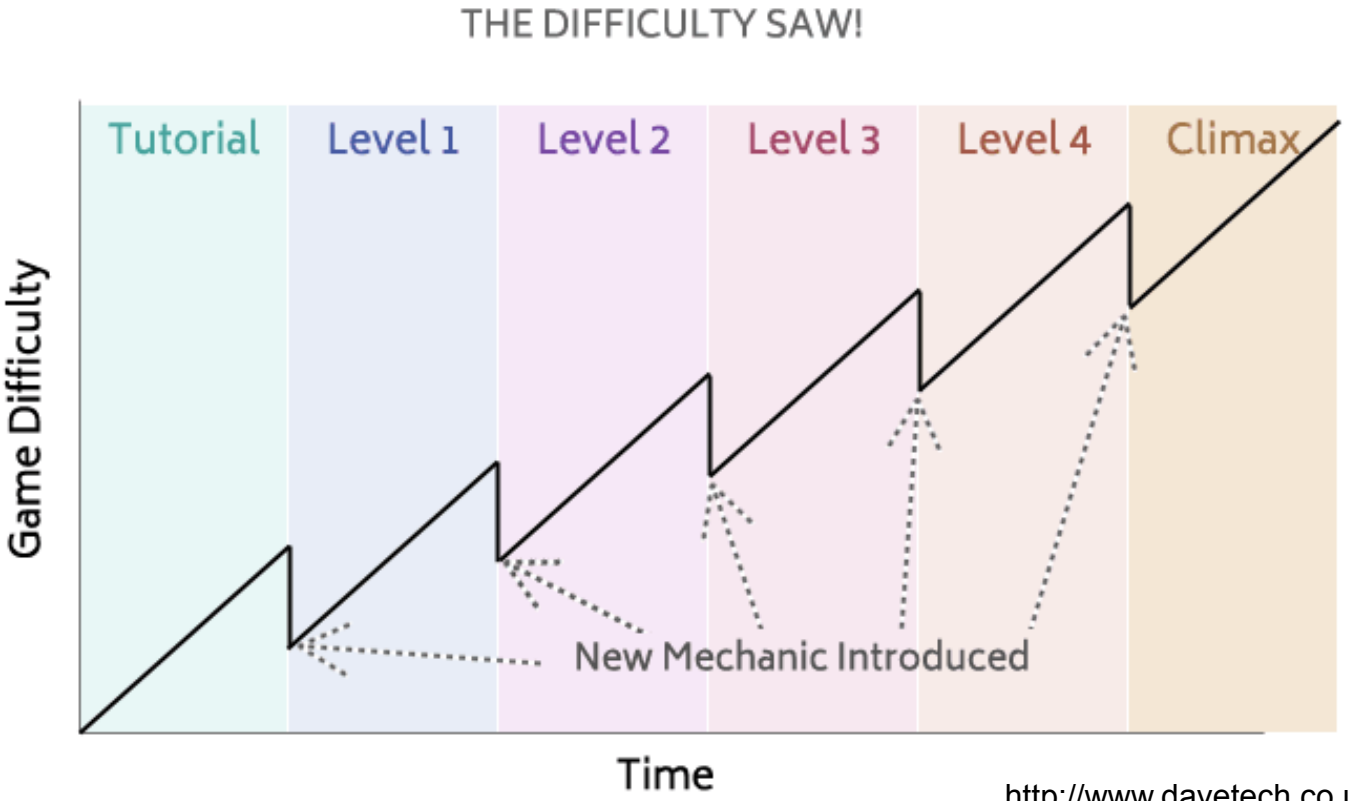
how different are our backgrounds?

- let’s find out!
  - what is an ArrayList?
  - what does *mens rea* mean?
  - what is the integral of x^2 + 5x?
  - what happened in Guatemala in 1954?
  - what does it mean to ”shower” in juggling?
  - what is the Nash Equilibium of the Prisoners’ Dilemna?
  - consider two cups that are just about to overflow with water. one also has some ice floating in it. which weighs more? why?
- 🗣️ is there anyone here who knows the answer to all the questions?
- 🗣️ is there any question that *no one* knows the answer to?

our backgrounds are very different...

- ...but my goal is for this class to be fun and accessible for everyone here
  - each week of lecture will follow an easy-to-hard difficulty ramp
    - Monday is easier
    - Wednesday is harder
    - Friday is hardest
  - each homework will follow an easy-to-hard difficulty ramp
    - B is easier
    - A is harder
    - S is hardest

the course as a whole should follow a saw



http://www.davetech.co.uk/difficultycurves

there is no finish line

- it’s 10+ years since i took the equivalent of CS 136
  - i still code almost every day
  - i still learn something new almost every day
  - i still feel like i have no idea what i’m doing almost every day

big O

how to read/write big O notation

- **big O** describes a function's "limiting behavior"
  - to find a mathematical function's big O notation...
    - 1. throw away the coefficients
    - 2. find the fastest growing term
    - 3. the function is $\mathcal{O}$(FASTEST_GROWING_TERM)

  - **e.g.,** $f(n) = 7n^2 + 100n + 4732$
    - 1. throw away coefficients to get $n^2 + n + 1$
    - 2. fastest growing term is n^2
    - 3. $f(n)$ is $\mathcal{O}(n^2)$

how to read/write big O notation

- **e.g.,** what is $f(n) = 77n^7 + 2^n$ in big O notation?
  - $n^7 + 2^n$
  - 2^n 🧪 is this true?
  - $\mathcal{O}(2^n)$

- **e.g.,** what is $f(n) = 100$ in big O notation?
  - 1
  - 1 🧠 what does this *mean*?
  - $\mathcal{O}(1)$

- **e.g.**, what is $f(n) = n + \log(n)$ in big O notation?
  - $n + \log(n)$
  - n 🧪 is this true?
  - $\mathcal{O}(n)$

how to read/write big O notation

- **e.g.,** Imagine a classroom with n students. I want to figure out if any students are named Carl.
  - I need an ✨Algorithm✨, **e.g.,** `boolean isAnyoneNamedCarl(Student[] students);`
  - In big O, what is longest amount of time each of these algorithms could take to run?
    - Algorithm 1: Ask each student, one at a time, “Are you named Carl?”
    - Algorithm 2: Pass a paper around the room, and have each student write their name on it. Then take the paper, and read through it.
    - Algorithm 3: The students draw straws. The student who draws the short straw must leave. On their way out of the room, ask them whether their name is Carl. Repeat this procedure until the room is empty.
    - Algorithm 4: Play Kahoot. The winner legally changes their name to Carl.

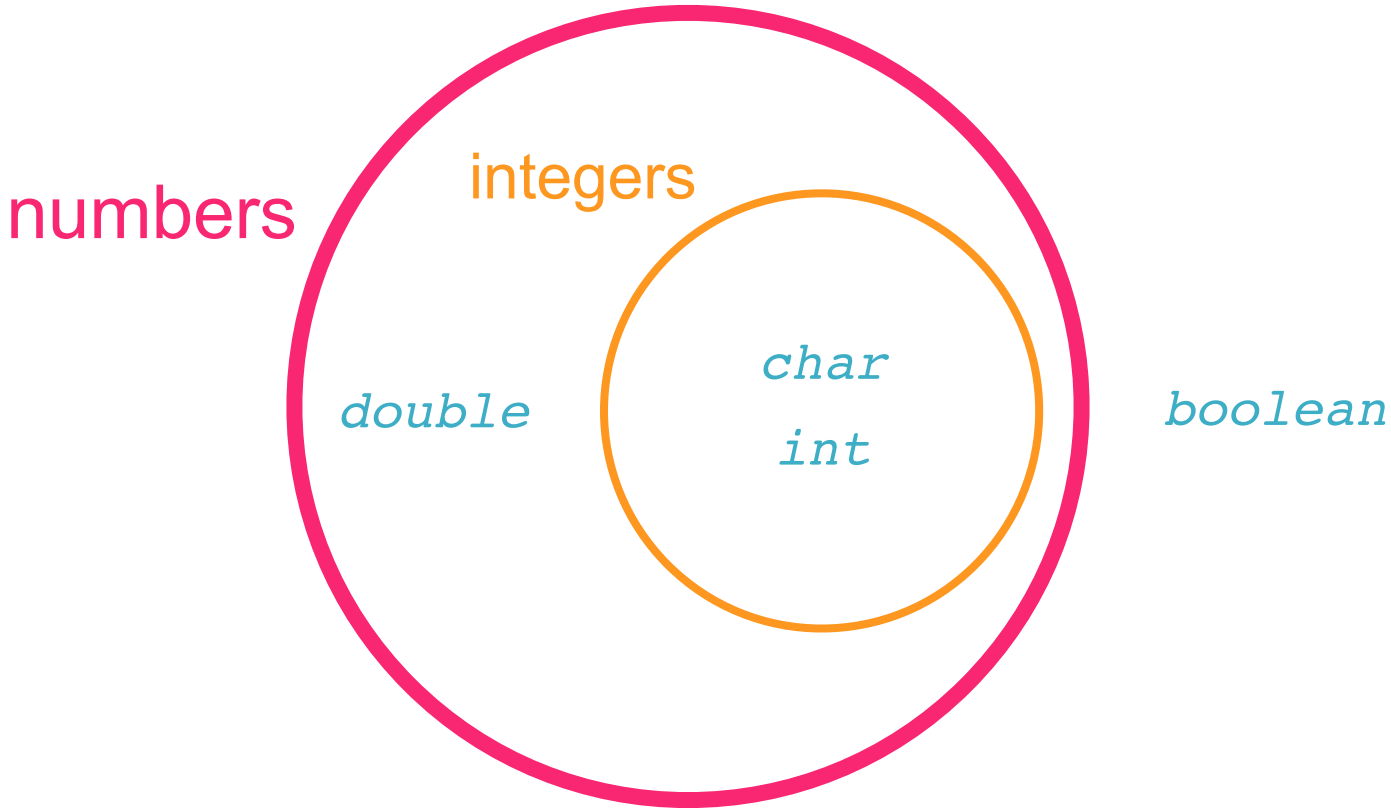
Java primitive

most popular Java primitive data types

boolean, *char*, *double*, *int*

- a ***boolean*** stores a truth value, *e.g.,*
  - `true, false`
- a ***char*** stores a character, *e.g.,*
  - `'\0', 'a', 'Z', '!'`
- a ***double*** stores a floating point number, *e.g.,*
  - `0.0, -0.5, 3.1415926, Double.NEGATIVE_INFINITY`
- an ***int*** stores an integer number, *e.g.,*
  - `0, -1, 4`

primitive data type Venn diagram



`char` is an integer type

- a `char` is an integer type
  - each `char` has a corresponding integer, *e.g.,* `('a' == 97)`
  - the letters are in order, *i.e.,* `('a' == 97)`, `('b' == 98)`, `('c' == 99)`...
  - the numbers are also in order, *i.e.,* `('0' == 48)`, `('1' == 49)`...
  - you can do math with `char`'s, *e.g.,*
    - `char foo = 'a' + 2; // foo is 'c'`
    - `char bar = '0' + 7; // bar is '7'`
    - `int  baz = '6' - '0'; // baz is 6`

zero

- each primitive data type has its own notion of what it means to "be zero"
  - `int     zero =     0;`
  - `double  zero =   0.0;`
  - `boolean zero = false;`
  - `char    zero =  '\0';`

Java operators

(except for bitwise operators, which we'll do later maybe)

assignment operator

assignment operator

- **=** **assigns** the value on the right-hand side to the variable on the left-hand side, *e.g.,*
  - `int i = 0;`
  - `double foo = coolFunction();`

- ☠ **the assignment operator *returns* the value it assigned this is usually pretty confusing, *e.g.,***
  - `int i = 4;`
  - `int j = (i *= 42);`

arithmetic operators

basic arithmetic (number) operators

- `+` **adds** two numbers
- `-` **subtracts** two numbers
- `*` **multiplies** two numbers
- `/` **divides** two numbers
  - ☠ **an *`int`* divided by an *`int`* is an *`int`***, *e.g.,*
    - `int foo = 8 / 2;`
    - `int bar = 2 / 7;`
      - Java "throws away the remainder"

- `-` returns the **negative** of a number, *e.g.,*
  - `int bar = -7;`
  - `int baz = -bar;`

modulo

- `x % N` returns the **remainder** of `x / N`
  and is read "x **modulo** N," *e.g.,*
  - `int foo = 17 % 5;`

- ☠️ `%` **probably doesn't do what you expect for negative numbers**
  - **instead, use** `((x % N + N) % N)`

logical operators

logical operators

- `||` returns whether the left-hand side **or** the right-hand side is true
  - `(true  || true)  // true`
  - `(true  || false) // true`
  - `(false || true)  // true`
  - `(false || false) // false`
- `&&` returns whether the left-hand side **and** the right-hand side are true
  - `(true  && true)  // true`
  - `(true  && false) // false`
  - `(false && true)  // false`
  - `(false && false) // false`

- `!` returns the opposite of a *`boolean`*, and is read as "**not**"
  - `(!true)  // false ("not true")`
  - `(!false) // true`

logical operators

```
// example, step by step
boolean a = (2 + 2 == 5); // false
boolean b = true;         // true
boolean c = (a || b);     // true
boolean d = !c;           // false

// same thing all on one line
boolean d = !((2 + 2 == 5) || true); // false

// equivalent code
boolean d = false;
```

☠ logical operator short-circuiting

- `(false || foo())` "lazily" evaluates to `false` without evaluating `foo()`
- `(true && foo())` "lazily" evaluates to `true` without evaluating `foo()`

comparison operators

☠ equality (is equal to)

- **==** returns whether the left-hand side **is equal to** the right-hand side, *e.g.*,
  - `boolean b = (foo == bar);`
  - `if (foo == bar) { ... }`

- ☠ **this does NOT work for *`String`*'s**
  - **instead, use `(stringA.equals(stringB) )`**

- ☠ **this does NOT work for *`double`*'s**
  - **instead, use `(Math.abs(double1 - double2) < 0.00001)`**

is greater than, is less than

- **>** returns whether the left-hand side **is greater than** the right-hand side
- **<** returns whether the left-hand side **is less than** the right-hand side

convenient operators

(feel free to ignore these for now)

inequality

- `!=` returns whether the left-hand side **is not equal to** the right-hand side
  - `(left != right)` is exactly the same as `(!(left == right))`

greater than or equal to, less than or equal to

- **>=** returns whether the left-hand side **is greater than or equal to** the right-hand side
  - `(left >= right)` is basically the same as
    `((left > right) || (left == right))`
    `greater-than or equal`
- **<=** returns whether the left-hand side **is less than or equal to** the right-hand side

arithmetic assignment operators

- `a += b;`
- `a -= b;`
- `a *= b;`
- `a /= b;`

String concatenation

- + concatenates two *String*'s, *e.g.*,
  -
    ```
    String foo = "Hello" + "World";
    ```
  -
    ```
    String foo = "Hello" + 2; // "Hello2"
    ```

increment operator

- to "**increment**" means to increase the value of a number by one, *e.g.*,
  - `i = i + 1;`
  - `i += 1;`

  - the **pre-increment** `++i` increments `i` and returns the new value of `i`
  - `j = ++i; // i = i + 1;`
  - `           // j = i;`

  - the **post-increment** `i++` increments `i` and returns the old value of `i`
  - `j = i++; // j = i;`
  - `           // i = i + 1;`

decrement operator

- to "**decrement**" means to decrease the value of a number by one, *e.g.,*
  - `i = i - 1;`
  - `i -= 1;`

  - the **pre-decrement** `--i` decrements `i` and returns the new value of `i`
  - `j = --i; // i = i - 1;`
  - `           // j = i;`

  - the **post-decrement** `i--` decrements `i` and returns the old value of `i`
  - `j = i--; // j = i;`
  - `           // i = i - 1;`

examples

// return whether n is prime

```
boolean isPrime(int n) { // TODO
    for (int i = 0; i < n; ++i) {
        if (n % i == 0) {
            return false;
        } // TODO
    }
    return true;
}
```

e.g., when called on `{ 0.0, 4.2, -10.0, 99.0 }`, returns 3
when called on `{}`, returns `-1`

```
int findIndexOfMaxEl
    int result = -1;
    double maxElemen
    for (int i = 0;
        if (array[i]
            result =
            maxEleme
        }
    }
    return result;
}
```

returns the number of digits an integer has (in base-10)
e.g., when called on 427, returns 3

```
int getNumberOfDigits(int n) {
    int result = 0;
    while (n != 0) {
        ++result; // result = result + 1;
        n /= 10;  // n = n / 10;
    }
    return result;
}
```

TODO: modding by length of array

print ☺☻♥♦♣

♫☼►◄↕‼¶§▬↨↑↓→←∟↔▲▼ !"#$
%&'()*+,-./0123456789:;<=>
@ABCDEFGHIJKLMNOPQRSTUVWXYZ
^_`abcdefghijklmnopqrstuvwx
|}~⌂

```
for (char c = 0; c < 128; ++
{
    System.out.print(c);
```


SONIC X
arrays

array

- an **array** is

```
int[] foo;
double[] baz;
String[] bar;
```

accessing an array

- B accessing an array is $\mathcal{O}(1)$
  - **i.e.,** "accessing an array takes a constant number of CPU cycles"

getting the value of array's i-th element

```
int currentValue = array[i];
```

setting the value of array's i-th element

```
array[i] = newValue;
```

creating an array

- B creating an array is $\mathcal{O}(n)$

creating a new integer array with 7 elements

```
int[] array = new int[7];
```

creating a new String array with n elements

```
String[] array = new String[n];
```

`while`

- a **while loop** repeats a block of code

`while`

- `while (true) { ... }` is useful for prototyping

TODO list

- the **list** (*aka* sequence) abstract data type is
  - an array list's *length* is the number of elements stored inside it
  - an array list stores its elements inside of an array
    - i call this array the array list's “internal array”
  - an array list's **capacity** is the length of this internal array

array list

- the **array list** (*aka* dynamic array, stretchy buffer, **vector**) data structure implements the list abstract data type using an array
  - an array list's *length* is the number of elements stored inside it
  - an array list stores its elements inside of an array
    - i call this array the array list's “internal array”
  - an array list's **capacity** is the length of this internal array

array list

- ! an array list's length is NOT the same thing as its capacity
  - *e.g.,* imagine an array list that is currently storing 3 `String`'s in an internal `String[]` of length 8. This array list has length 3 and capacity 8. Its internal array might be `[ "Blango", "Sproot", "Sparket", null, null, null, null, null, ]`, where the `null` "elements" are empty slots in the internal array. There are $8 - 3 = 5$ empty slots
- ! even though another name for an array list is a "vector," the array list is NOT related to the vector from math and physics

`ArrayList() { ... }`

- a new array list should have...
  - `length` equal to 0
  - `internalArray` equal to a new array with length equal to some starting capacity, e.g., 4

`void add(ElementType element) { ... }`

- to **add** (*aka* **append**, **push back**) an element to an array list...
  - if the internal array is full...
    - create a new array two times the length of the current internal array
    - copy the elements of the current internal array over into the new array
    - update the array list's internal array reference to refer to the new array
  - write the element to the first available empty slot in the internal array
  - increment the array list's length

- ⏱ adding an element to an array list has worst-case runtime of $\mathcal{O}(n)$ and amortized worst-case runtime of $\mathcal{O}(1)$
  - if we're out of space, it takes $\mathcal{O}(n)$ time to copy the elements over, however this happens only every $\mathcal{O}(n)$ adds

How to make a Starbucks Java Chip Frappuccino

Watch

primitives

operators

scope

scope

- a **scope** is a region of code in which variables live
  - in Java, a scope is define by a pair of curly braces
    - `OUTSIDE_SCOPE { INSIDE_SCOPE } OUTSIDE_SCOPE`
    - remember, Java doesn't care about whitespace

whitespace

- **whitespace** includes spaces and newlines
- 🐍 Python does care about whitespace (indentation *changes what code does*)
- Java does NOT care about whitespace
  - 🗣️ do *you* care about whitespace?
  - some guidelines:
    - be consistent!
    - carefully indent your scopes (and make sure your curly braces line up)
      - ✨ your text editor can do this for you!

sparks joy	NOT equivalent -- doesn't spark joy

`for (int i = 0; i < 10; ++i) {`
`  if (i % 3 == 0) {`
 	
`System.out.println("fizz");`
`  }`
	`for (int i = 0; i < 10; ++i) {`
	`  if (i % 3 == 0) {`
	`}`
	`  System.out.println("fizz");`

about the lectures

 how to read side by side code

- i often show equivalent code side by side in order to...
  - relate new concepts to old concepts
  - compare and contrast different design decisions, *e.g.,*

a piece of code	equivalent code	equivalent code	equivalent code

`boolean isEven;`
`if (i % 2 == 0) {`
`    isEven = true;`
`} else if (i % 2 != 0) {`
`    isEven = false;`
`}`
 	
`if (isEven) {`
`    ...`
`}`
	`boolean isEven;`
	`if (i % 2 == 0) {`
	`    isEven = true;`
	`} else {`
	`    isEven = false;`
	`}`
	
	`if (isEven) {`
	`    ...`
	`}`
		`boolean isEven = (i % 2 == 0);`
		
		`if (isEven) {`
		`    ...`
		`}`
			`if (i % 2 == 0) {`
			`    ...`
			`}`

🐮 how to read emojis

- i use emojis to help you read and study
  - 🐮 info only relevant inside the world of CS136
  - ☕ fun Java fact! (*i.e.,* NOT relevant to C/C++; please forget after CS136)
  - 🐍 comparison to Python
  - ☠️ **common misconception or potential source of bugs**
  - 👀 spoilers/hints
  - ⏱️ big O runtime
  - ✨ optional (NOT on exams) but sparks joy
  - 🧠 question for you to think about
  - 🗣️ question for your to talk about
  - 🧪 question for you to experiment with
  - 🙂👍

🐮 how to read emojis

- 🧠 what is code?
  - **code** tells a computer how to do something
- 🗣️ what makes code *good*?
  - good code makes your computer do the thing you want it to do, and...
    - runs fast
    - is small
    - is easy to read
- 🧪 make this code *worse*

Java code to make your computer print `Hello World!`

```
class HelloWorld {
    public static void main(String[] args)
{
        System.out.println("Hello,
World!");
    }
```

```
public class HelloWorld {public static void main(String[]
args){ int[][] t = new int[][]
{{202,1026,1100,396,324,1080,192,609,555,888,72,432},
{3,9,8,5},{2,2,5,9},{4,6,1,9,2,11},
{4,6,1,9,3,2,11,7,0,5,10},{2,1,5,9},{1,9,2,5},
{0,2,10,5,1,6,3,11,8,4},{10,4,2,6},
{1,10,2,3,5,9,7,4,11,6},{7,0,3,6},{2,9,10,1},{7,1,10,6},
{12,0,-0}};do{while(t[13][1]+1<t[t[13][0]].length){ t[13]
[2]=t[0][t[t[13][0]][t[13][1]]];t[0][t[t[13][0]][t[13]
[1]]]=t[0][t[t[13][0]][++t[13][1]]];t[0][t[t[13][0]][t[13]
[1]++]]=t[13][2];} }while(!(--t[13]
[0]<=(int)Math.sin(Math.PI))&&((t[13][1]=0)<1));while(t[4]
[2]<=t[9][5]+3)System.out.print((char)(t[0][t[4][2]-1]/t[4]
[2]++));}}
```