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 get comfortable coding up cool stuff from scratch

expectations

- the **lectures** should make sense (you may have to watch more than once)
  - 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 and make a plan for success
  - if it's too easy, let's chat and i'll find you something to do
- the **exams** should be challenging and unsurprising

disclaimer

- homework and exams will be a bit different than in previous years
  - you will sometimes feel like a guinea pig 🐹
  - the TA's will sometimes feel confused
  - it will also be cool 😀 👍

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

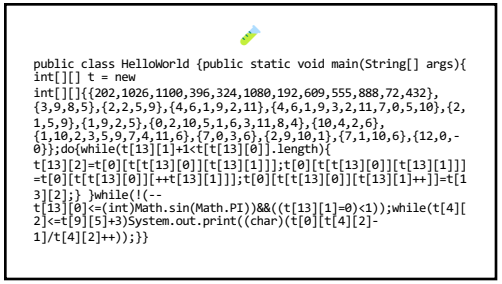
- i use emojis to help you read and study
  - 🐮 info only relevant inside the world of CS136
  - 🌰 fun Java fact! (*i.e,* NOT relevant to C/C++; please forget after CS136)
  - 🔄 comparison to Python
  - ☠️ common misconception or potential source of bugs
  - 👀 spoilers/hints
  - ⏱️ big O runtime
  - ✨ 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[1
3][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]++));}}
```

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?
  - $\mathcal{O}(n)$

primitives

operators

scope

+=, -=, *=, /=

- shorthand assignment operators can make your code easier to read

using shorthand	not using shorthand
`a += b;`	`a = a + b;`
`a -= b;`	`a = a - b;`
`a *= b;`	`a = a * b;`
`a /= b;`	`a = a / b;`

increment

- **increment** means to increase the value of something by one
  - the **pre-increment** `++i` increments `i` and returns the new value of `i`
  - the **post-increment** `i++` increments `i` and returns the old value of `i`

one line	two lines
`j = ++i;`	`i = i + 1;` `j = i;`
`j = i++;`	`j = i;` `i = i + 1;`

```
for (int i = 0; i < n; ++i)
```

```
for (int i = 0; i < n; i++)
```

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

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

NOT equivalent -- doesn't spark joy

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

}
```

array

- an **array** is

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

accessing an array

- accessing an array is O(1)
  - **i.e.,** " access 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

- creating an array is 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];
```