Review Session 0

1/27/20

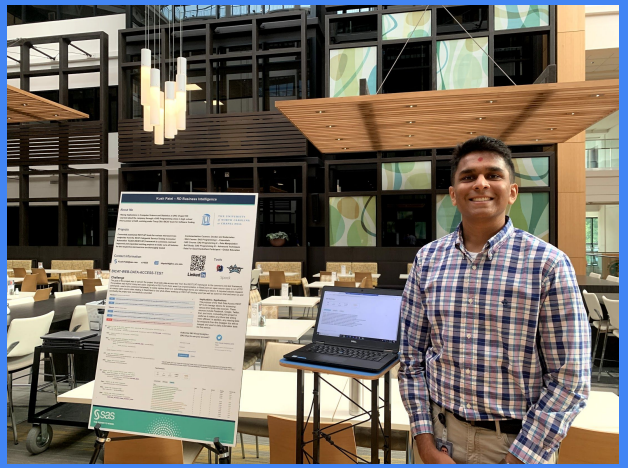
Taking it ~slow~

Hi! I’m Anna

- ★ Sophomore from Cary NC
- ★ CS (BA) + Busi
- ★ Interning @ Cisco this summer in San Jose, CA
- ★ Fun fact: I gave a tour to Ethan Wacker, disney channel star, ex-bf of Olivia Rodrigo from HSM the series

Hi I’m Kush :)

- From Morrisville, NC
- I’m a sophomore!
- Majoring in CS and Stat
- Interned at SAS this past summer and year-round now
- I love basketball and Friends :)



Hi I’m Chelsea!!

- ★ Sophomore
- ★ Computer Science major, Math minor
- ★ From Milwaukee, Wisconsin #gobucks
- ★ Interning at Fidelity in Durham in the upcoming summer
- ★ Fun Fact: I’ve seen Giannis Antetokounmpo at my local movie theater TWICE

Fun Fact #2: I did goat yoga last week

Overview

- ★ Basics: Data types & Variables
- ★ Expressions & statements
- ★ If-then-else & while loops
- ★ Hot date & check in on course.care
- ★ Functions
- ★ Control flow & scope
- ★ Environment diagrams

Data types!

Typescript in Java has 3 ***primitive*** data types:

1. Number → 2,000,000 (# of female turtles in the world)
2. String → “I like turtles”
3. Boolean → A turtle can weigh 1,500 lbs → TRUE

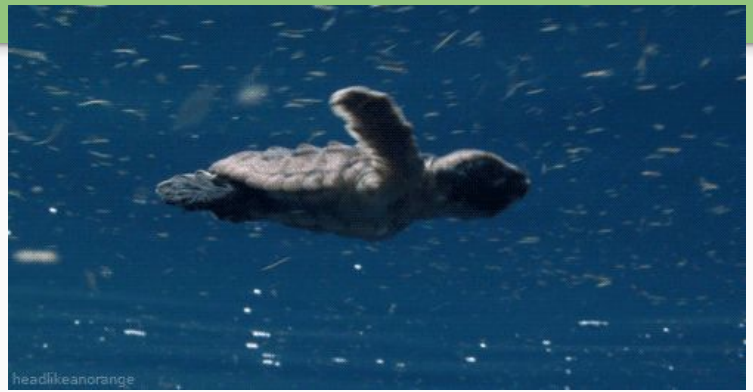
Numbers & Operations

- The number type is relatively straight forward
- These are operations that we can do on numbers =>
- These follow order of operations PEMDAS
- If all else fails, use parentheses to be sure your order is correct!

Name	Operator Symbol	Example
Exponentiation	**	`2 ** 8` (is the same as 2^8)
Multiplication	*	`10 * 3`
Division	/	`100 / 5`
Remainder	%	`18 % 5` (remainder of 18 divided by 5)
Addition	+	`1 + 1`
Subtraction	-	`111 - 1`

Strings

- Strings are textual data
- Usually just words and characters BUT numbers can be strings too
- Tip: if it’s in double quotes “” it’s a string



“He”

“swims”

“2”

“fast”

Concatenation

- Using the + operator to put together strings with other strings OR strings with other numbers

“He” + “ swims” ---------> “He swims”

2 + “fast” ----------> “2fast”

- You need to include spaces if you want them!

Boolean Operations

TRUE or FALSE

- AND && operator

AND truth table

	true	false
true	true	false
false	false	false

- OR || operator

OR truth table

	true	false
true	true	true
false	true	false

Comparing Numbers and Strings

- We use relational and equality operators to compare number and string values
- These are EXPRESSIONS that simplify to boolean values
- (we'll do more on expressions later)

Test	Math	TypeScript Operator
"is greater than?"	>	>
"is at least?"	≥	>=
"is less than?"	<	<
"is at most?"	≤	<=
"is equal to?"	=	===
"is not equal to?"	≠	!==

Variables!!

- Allow us to *store*, *load*, and *change* values in memory
- Every variable has:
  - A **name**
  - Holds a value of a specific **data type**

4

ninjaTurtles

Type: number

Declaration and Initialization

Declaration:

let <name>: <type>

let bigTurtle: string;

Initialization:

<name> = <value>

bigTurtle = “Crush”;

Declaration and Initialization together!

And Type Inference!

Both:

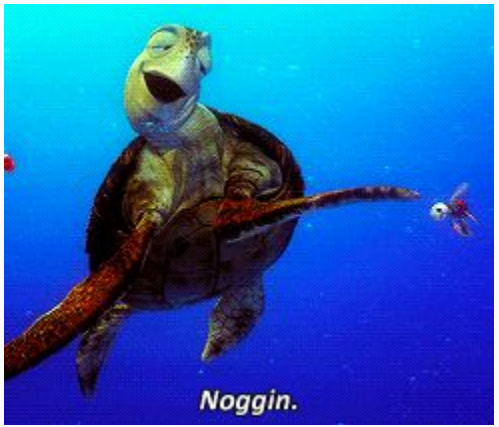
let <name>: <type> = <value>

let littleTurtle: string = “Squirt”;

Type Inference

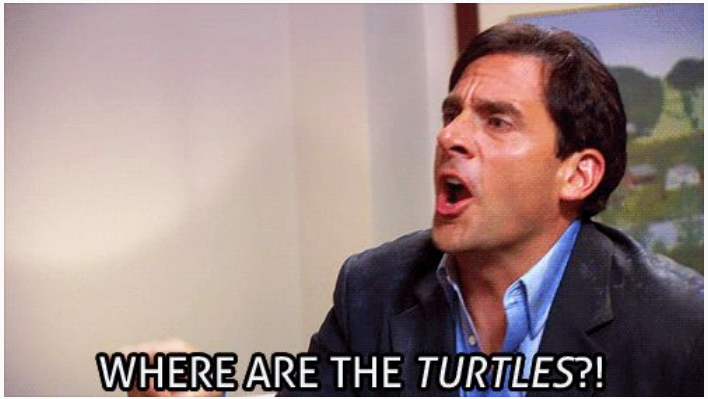
let <name> = <value>

let littleTurtle = “Squirt”;



Expressions vs. Statements

- Statements: instructions to the computer
- Expressions: complicated statements whose values are determined by evaluation
  - Arithmetic operations (i.e. 9 + 10)
  - Concatenation (i.e. “hello “ + “class”)
  - Boolean expressions (i.e. the table from before)



If-then-else

- You only want something to happen **IF** a condition is true
- No such thing as a “then” statement

```
export let main = async () => {

    let response: number = await promptNumber("What is 9 + 10?");

    if (response === 19) {
        print("Correct!");
    } else {
        print("Incorrect!");
    }

};
```

else if vs else

- **else if**: If it doesn't pass the first if statement, AND you want the others option to be conditional
- **else**: No condition specified; the "none of the above" option

```
export let main = async () => {

    let response: number = await promptNumber("What is 9 + 10?");

    if (response === 19) {
        print("Correct!");
    } else if (response === 21) {
        print("You Stupid!");
    } else {
        print("Incorrect!");
    }

};
```

Is this the same?

```
export let main = async () => {

    let response: number = await promptNumber("What is 9 + 10?");

    if (response === 19) {
        print("Correct!");
    }
    if (response === 21) {
        print("You Stupid!");
    } else {
        print("Incorrect!");
    }

};
```

Answer: NO! What prints if response is 19?

```
export let main = async () => {

    let response: number = await promptNumber("What is 9 + 10?");

    if (response === 19) {
        print("Correct!");
    }
    if (response === 21) {
        print("You Stupid!");
    } else {
        print("Incorrect!");
    }

};
```

Answer: NO! What prints if response is 19?

```
export let main = async () => {

    let response: number = await promptNumber("What is 9 + 10?");

    if (response === 19) {
        print("Correct!");
    }
    if (response === 21) {
        print("You Stupid!");
    } else {
        print("Incorrect!");
    }

};
```

Correct!

Incorrect!

Is this the same?

```
export let main = async () => {

    let response: number = await promptNumber("What is 9 + 10?");

    if (response === 19) {
        print("Correct!");
    } else {
        if (response === 21) {
            print("You stupid!");
        } else {
            print("Incorrect!");
        }
    }

};
```

Answer: YAS!!!

```
export let main = async () => {

    let response: number = await promptNumber("What is 9 + 10?");

    if (response === 19) {
        print("Correct!");
    } else {
        if (response === 21) {
            print("You stupid!");
        } else {
            print("Incorrect!");
        }
    }

};
```

While Loop!!

- Anytime you want to repeat a process **WHILE** a certain condition is true
- Saves SO much code!

```
let wantToPlay = true;

while (wantToPlay) {
    let rps = await promptString("Rock, Paper, Scissors?");
    let cpu = "Scissors";
    if (rps === "Rock") {
        print("Rock beats Scissors");
    } else if (rps === "Paper") {
        print("Scissors cuts Paper");
    } else if (rps === "Scissors") {
        print("Draw!");
    } else {
        print("welp");
    }

    let response = await promptString("Want to play again?");
    if (response === "No") {
        wantToPlay = false;
    }
}
```

What prints?

```
let x = 10;
let s = "yo";

while (x < 15) {
    print(x);
    print(s);
    x = x + 1;
    s = s + s;
}
```

What prints?

```
let x = 10;
let s = "yo";

while (x < 15) {
    print(x);
    print(s);
    x = x + 1;
    s = s + s;
}
```

10
yo
11
yoyo
12
yoyoyoyo
13
yoyoyoyoyoyoyoyo
14
yoyoyoyoyoyoyoyoyoyoyoyoyoyoyo

If-then vs. while

If-then

- After the code within the if/else block is complete, the program moves on to the line after (condition is **NOT** re-evaluated)

```
if (win) {
    print("Yay!");
} else {
    print("Oh no!");
}

print("Good game");
```

While

- After the code within the while completes, the program **JUMPS** back to the first line to re-evaluate the condition

```
let win = true;
let numwins = 0;

while (win) {
    numwins++;
    if (numwins === 9) {
        win = false;
    }
}
```

Use cases for If-then vs. while

If-then

- Whenever you want code to be run ONLY if a condition is true
- Multiple choices with if-else if-else

While

- Whenever you want code to be run REPEATEDLY ONLY if a condition is true
- ERROR PRONE

Infinite Loop

```
while (wantToPlay) {
    let rps = await promptString("Rock, Paper, Scissors?");
    let cpu = "Scissors";
    if (rps === "Rock") {
        print("Rock beats Scissors");
    } else if (rps === "Paper") {
        print("Scissors cuts Paper");
    } else if (rps === "Scissors") {
        print("Draw!");
    } else {
        print("welp");
    }

    let response = await promptString("Want to play again?");
    if (response === "No") {
        print("Sike");
    }
}
```

Hot date and check-in!

Go to course.care and check-in using:

35A48

And talk to your neighbor about how your semester is going!

Functions

- Declared outside of the main **function**
- We can call these functions whenever we need it, WITHOUT having to rewrite code

```
let sum = (x: number, y: number): number => {
    return x + y;
};
```

Argument vs. Parameter

- **Argument -** literal values passed into the function upon calling it
- **Parameters -** declared in the function definition
  - takes on the values of the arguments passed into it

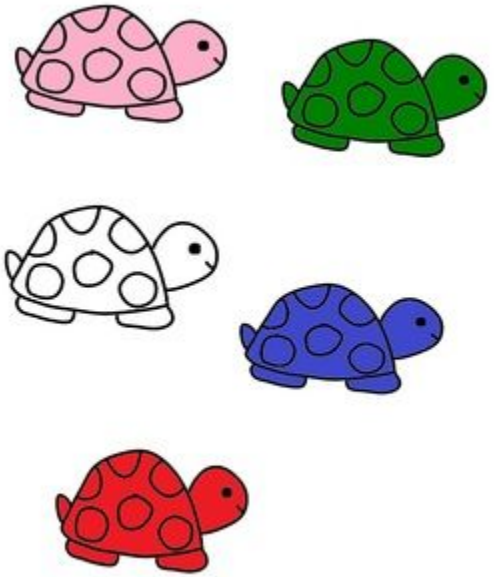
Arguments: Function Call :: Parameters : Function Definition

Argument and Parameter Matching

1. In number
2. In variable type
3. In order

```
let str = f(9, false, "cloud");
```

```
let f = (x: number, y: boolean, z: string): string => {
    if (y) {
        return x + z;
    } else {
        return z + x;
    }
};
```



Return Statement

- As soon as it is encountered, the function STOPS running and the program jumps back to the call
  - Returned value will **replace** the function call

```
let f = (v: boolean, x: number, y: boolean, z: string): string => {
    if (y) {
        return x + z;
    }
    if (v) {
        return z + x;
    }

    return z;
};
```

Other Key Tips

1. Practice how to write the definition:

   let fname = (param1: <type>, param2: <type>): <return type> => {

   }; ← DO NOT FORGET THE SEMICOLON

2. Don’t forget your return statement!
3. Functions don’t need to have parameters!

Control Flow

- How the computer goes through your program!
  - Function calls drop a “bookmark” and jump to evaluate the function
  - Return to that “bookmark” (RA) after hitting a return statement

```
1   import { print } from "introcs";
2
3   export let main = async () => {
4       let x = 23;
5       print(bringUmbrellaToday(x));
6   };
7
8   let bringUmbrellaToday = (percentChanceOfRain: number): boolean => {
9       return (percentChanceOfRain > 50);
10  };
11
12  main();
```

Scope

- Space within which a variable exists and can be accessed
- You can always look out, but not in

```
let i = 0;
while (i < n) {
    let a = 8;
    print(i);
    i = i + 1;
}
print(a);
```

Variables declared in outer blocks are accessible!

```
let i = 0;
while (i < n) {
    le
    pr
    i
}
print(a);
```

any

Cannot find name 'a'. ts(2304)

Peek Problem No quick fixes available

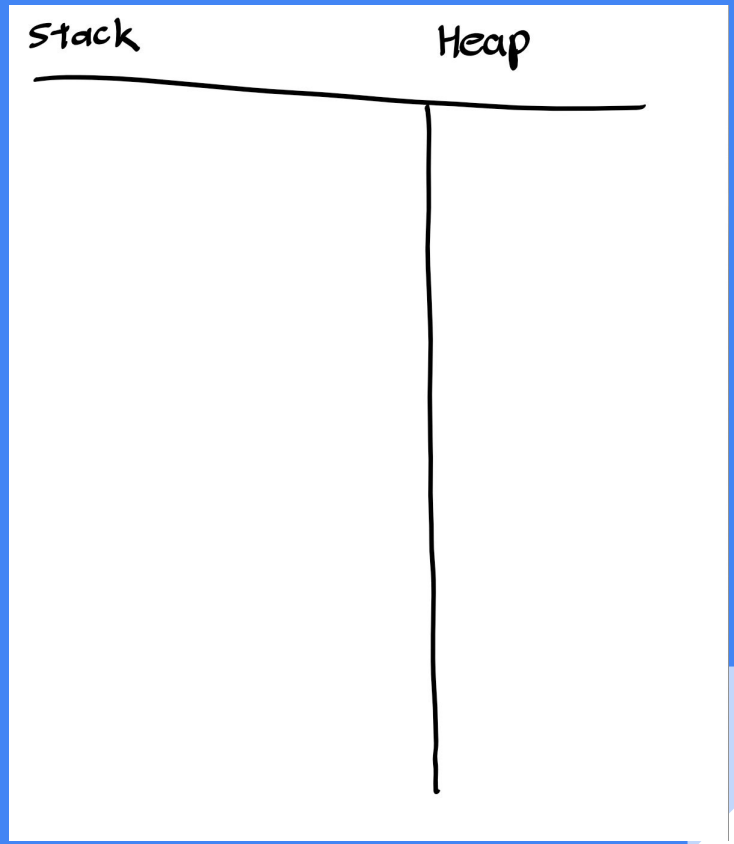
But variables declared in inner blocks are not

Environment Diagrams: Rules of Thumb

1. Start from the top and work your way to the bottom
2. Function call => new frame on the call stack!
    a. Return address
    b. Parameter values
3. Jump to function definition and run through code
    a. “Let” keyword => new variable and value within frame
4. Return statement => fill in RV => Return to RA

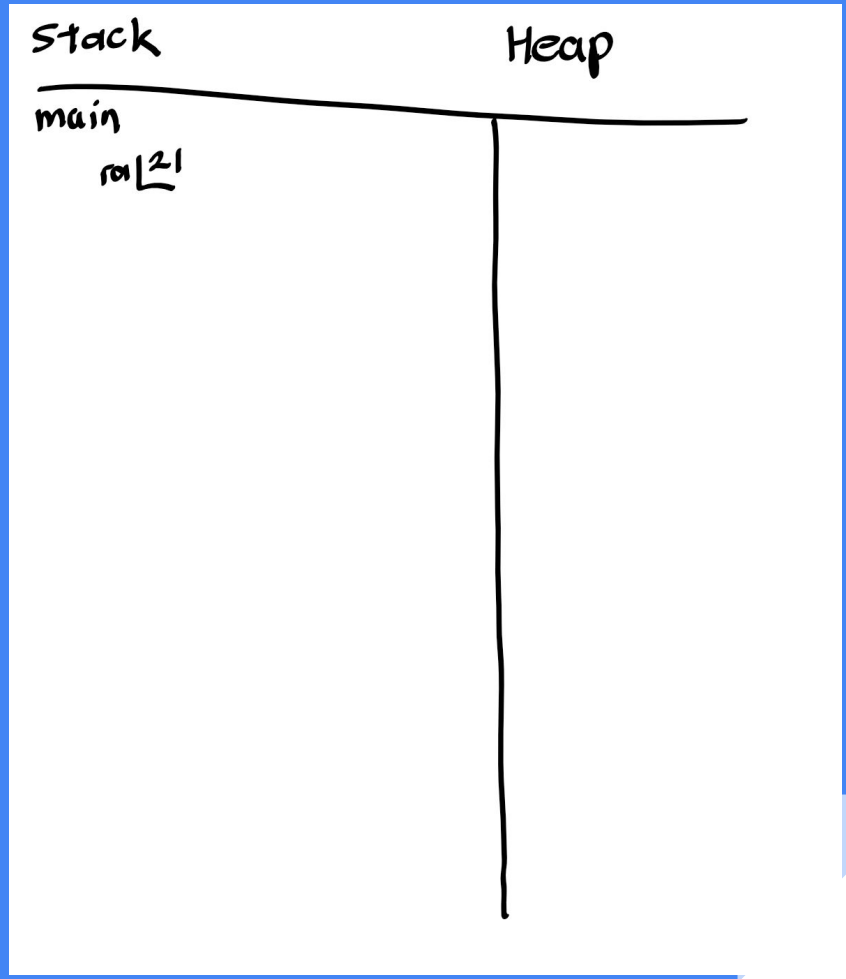
Environment Diagram Example

```
3   export let main = async () => {
4       let x: number = square(3);
5       print(x);
6   };
7
8   let square = (x: number): number => {
9       return pow(x, 2);
10  };
11
12  let pow = (a: number, b: number): number => {
13      let temp = a;
14      while (b > 1) {
15          a *= temp;
16          b--;
17      }
18      return a;
19  };
20
21  main();
22
```



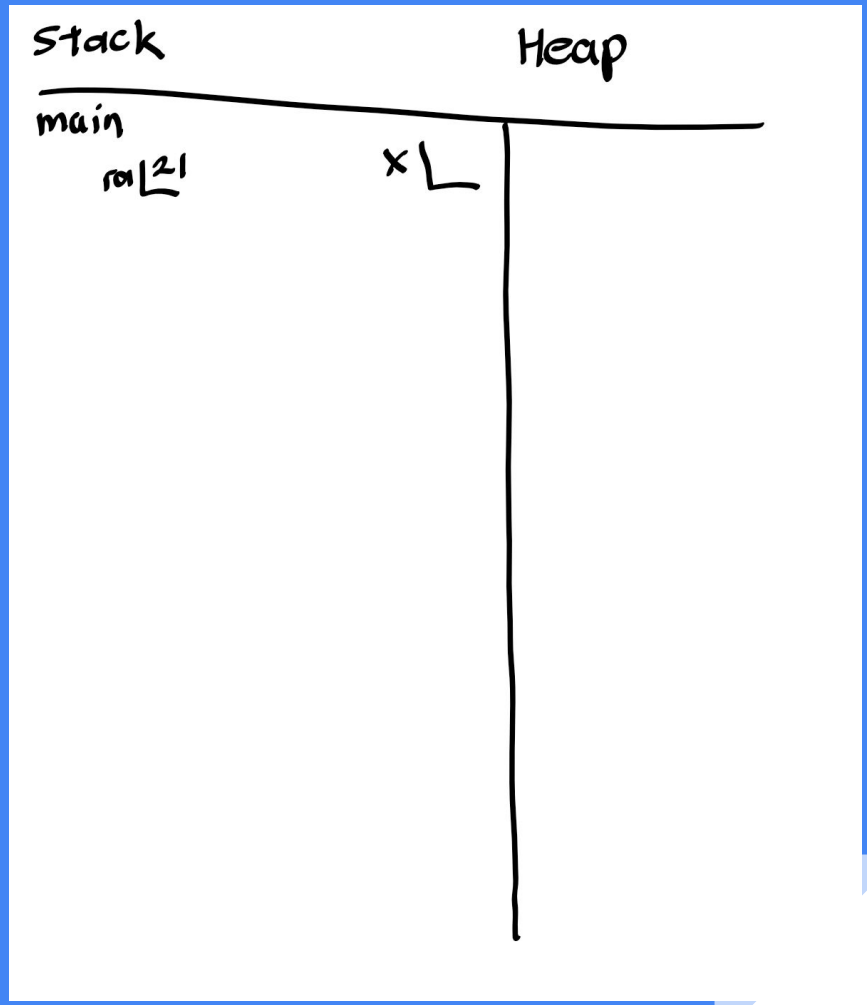
```
3   export let main = async () => {
4       let x: number = square(3);
5       print(x);
6   };
7
8   let square = (x: number): number => {
9       return pow(x, 2);
10  };
11
12  let pow = (a: number, b: number): number => {
13      let temp = a;
14      while (b > 1) {
15          a *= temp;
16          b--;
17      }
18      return a;
19  };
20
21  main();
22
```

Function call!



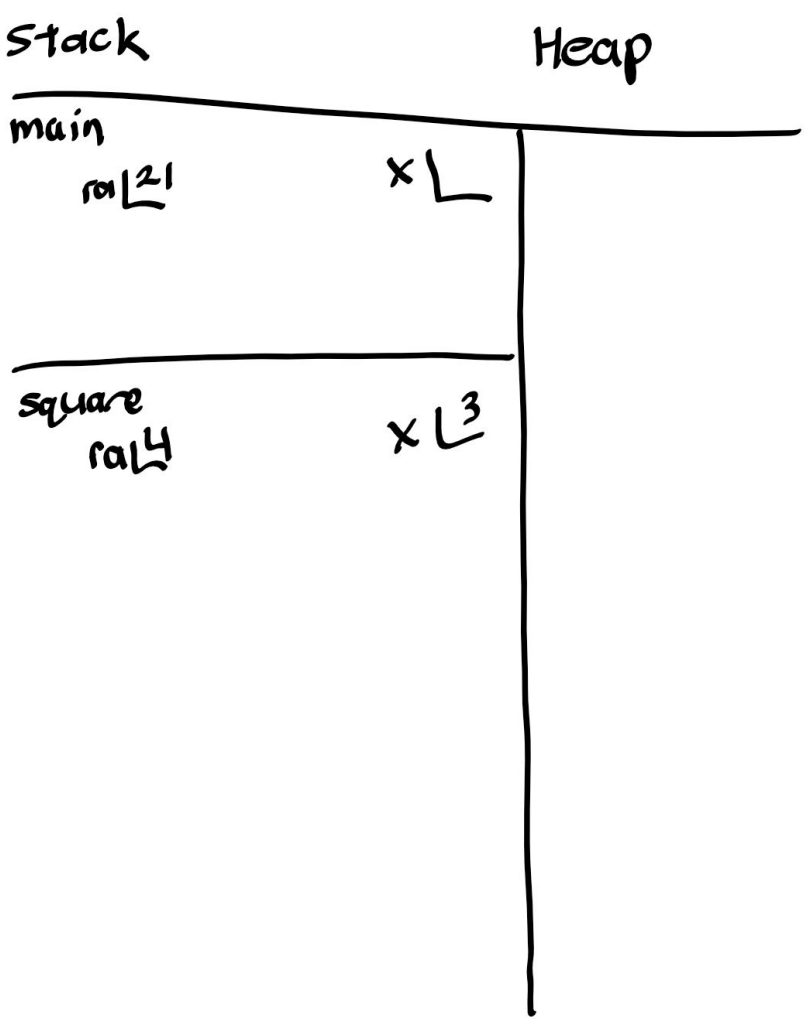
```
3   export let main = async () => {
4       let x: number = square(3);
5       print(x);
6   };
7
8   let square = (x: number): number => {
9       return pow(x, 2);
10  };
11
12  let pow = (a: number, b: number): number => {
13      let temp = a;
14      while (b > 1) {
15          a *= temp;
16          b--;
17      }
18      return a;
19  };
20
21  main();
22
```

Jump to main



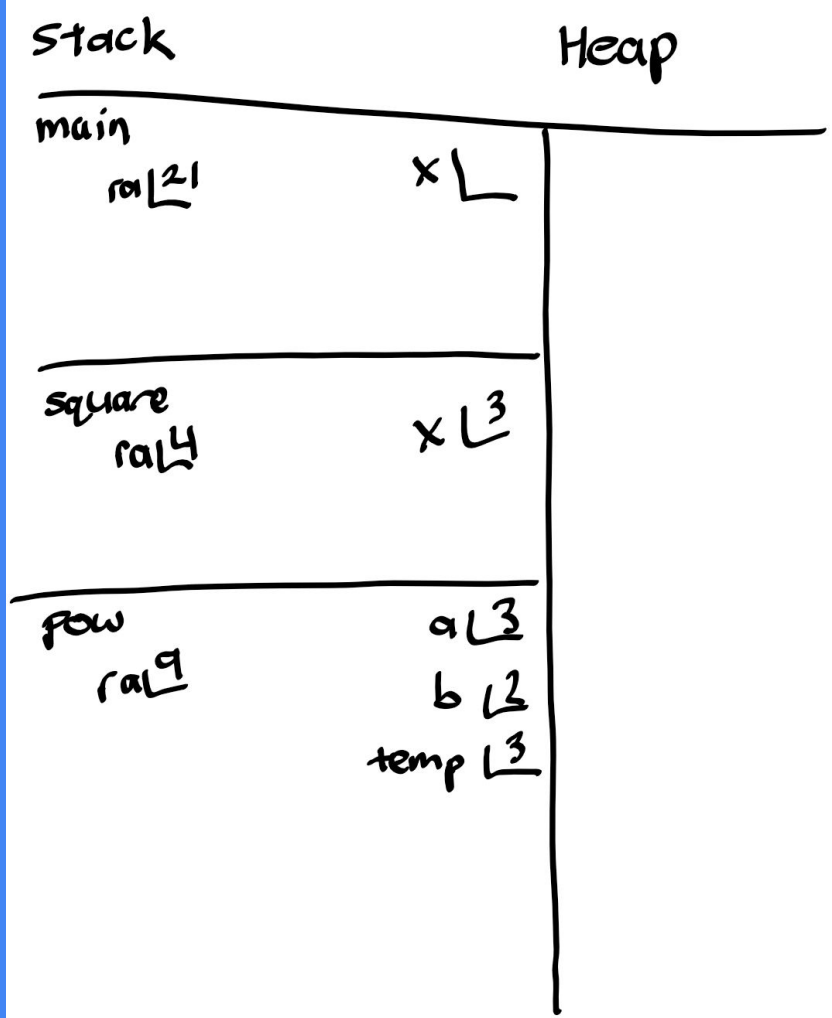
```
3   export let main = async () => {
4       let x: number = square(3);
5       print(x);
6   };
7
8   let square = (x: number): number => {
9       return pow(x, 2);
10  };
11
12  let pow = (a: number, b: number): number => {
13      let temp = a;
14      while (b > 1) {
15          a *= temp;
16          b--;
17      }
18      return a;
19  };
20
21  main();
22
```

Execute square



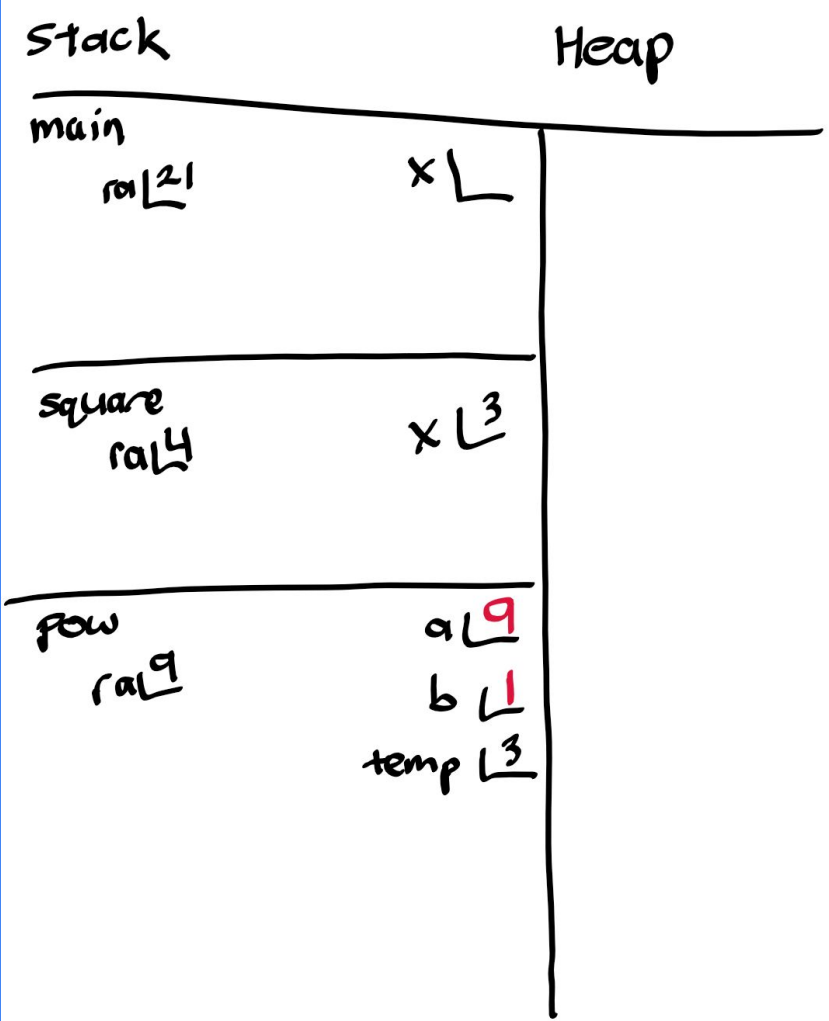
```
3   export let main = async () => {
4       let x: number = square(3);
5       print(x);
6   };
7
8   let square = (x: number): number => {
9       return pow(x, 2);
10  };
11
12  let pow = (a: number, b: number): number => {
13      let temp = a;
14      while (b > 1) {
15          a *= temp;
16          b--;
17      }
18      return a;
19  };
20
21  main();
22
```

Execute pow



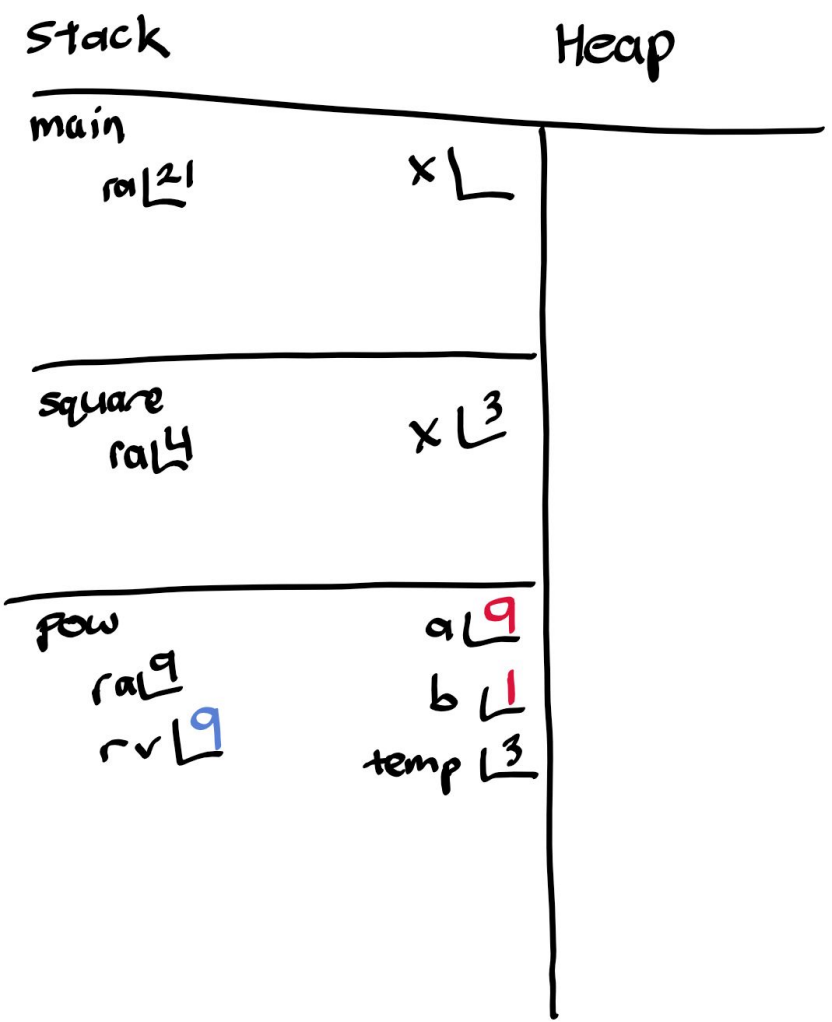
```
3   export let main = async () => {
4       let x: number = square(3);
5       print(x);
6   };
7
8   let square = (x: number): number => {
9       return pow(x, 2);
10  };
11
12  let pow = (a: number, b: number): number => {
13      let temp = a;
14      while (b > 1) {
15          a *= temp;
16          b--;
17      }
18      return a;
19  };
20
21  main();
22
```

handle the while loop



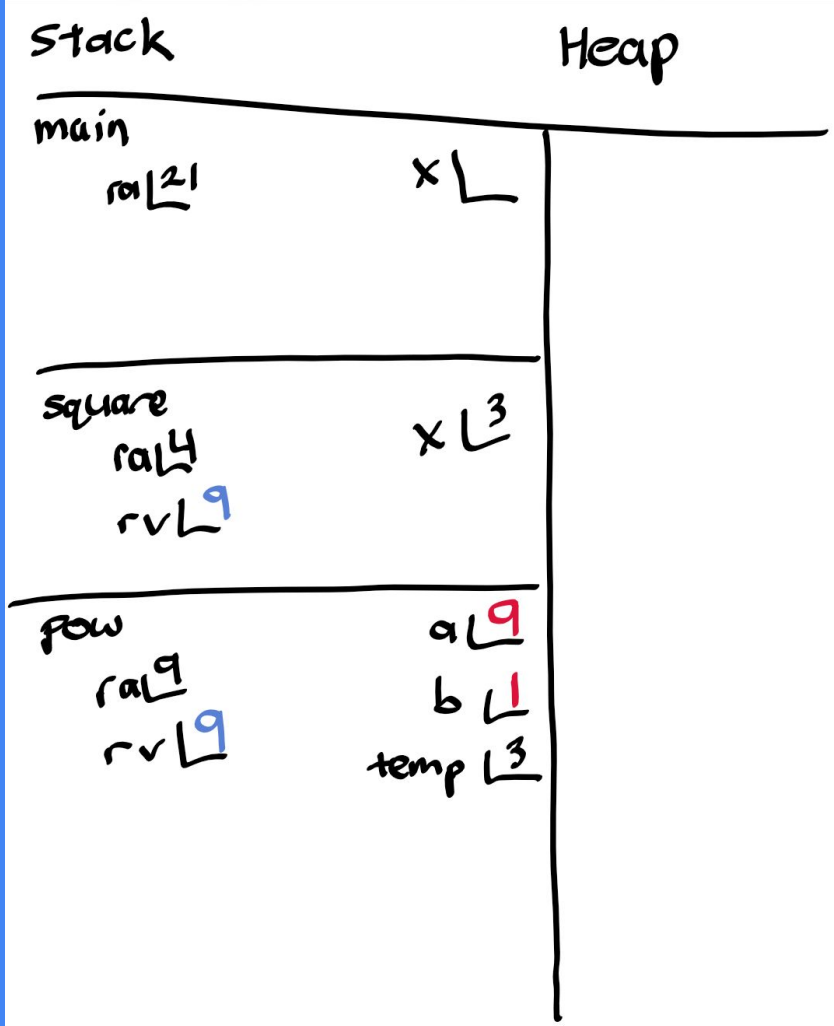
```
3   export let main = async () => {
4       let x: number = square(3);
5       print(x);
6   };
7
8   let square = (x: number): number => {
9       return pow(x, 2);
10  };
11
12  let pow = (a: number, b: number): number => {
13      let temp = a;
14      while (b > 1) {
15          a *= temp;
16          b--;
17      }
18      return a;
19  };
20
21  main();
22
```

Update RV, Return to pow's RA



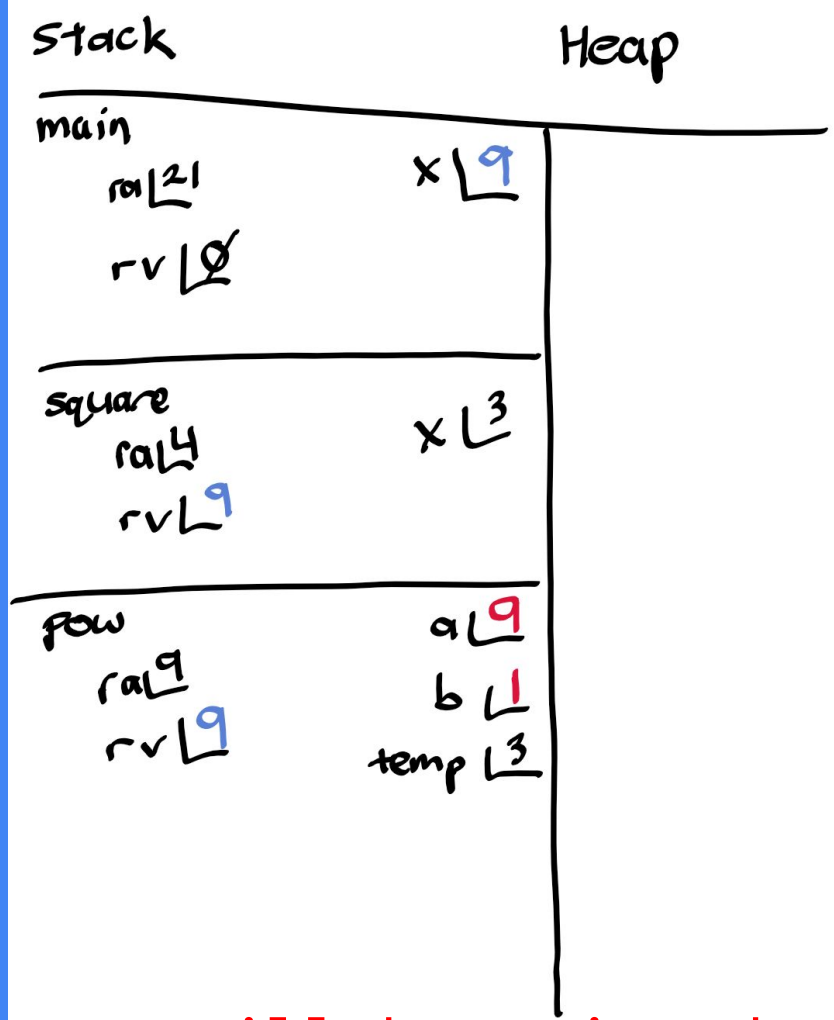
```
3   export let main = async () => {
4       let x: number = square(3);
5       print(x);
6   };
7
8   let square = (x: number): number => {
9       return pow(x, 2);
10  };
11
12  let pow = (a: number, b: number): number => {
13      let temp = a;
14      while (b > 1) {
15          a *= temp;
16          b--;
17      }
18      return a;
19  };
20
21  main();
22
```

Update RV, Return to square's RA

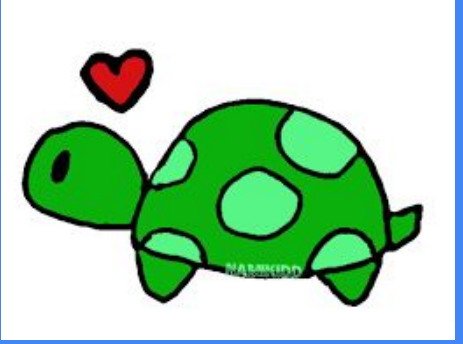


```
3   export let main = async () => {
4       let x: number = square(3);
5       print(x);
6   };
7
8   let square = (x: number): number => {
9       return pow(x, 2);
10  };
11
12  let pow = (a: number, b: number): number => {
13      let temp = a;
14      while (b > 1) {
15          a *= temp;
16          b--;
17      }
18      return a;
19  };
20
21  main();
22
```

9



9 will be printed



Questions???