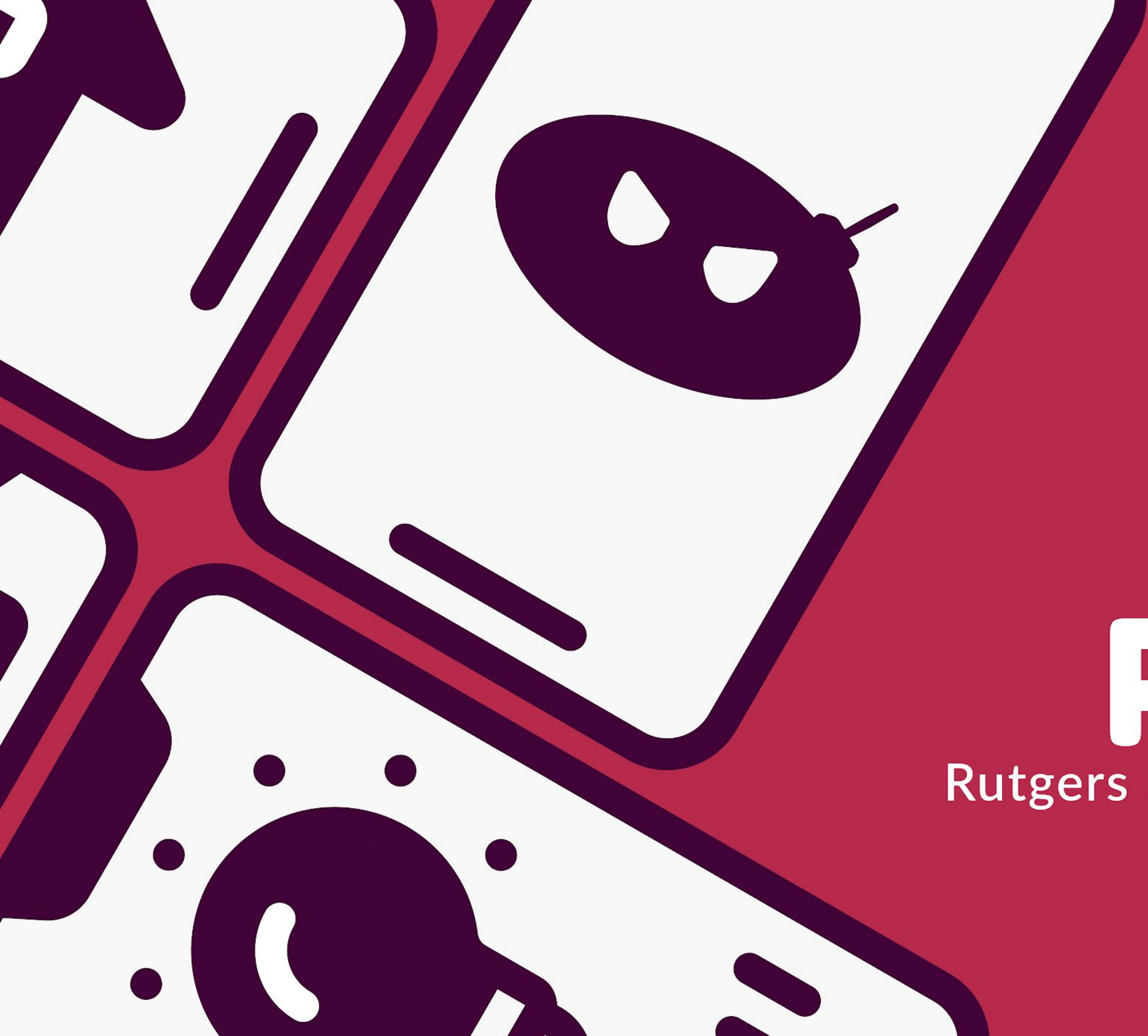


RUMAD

Rutgers Mobile App Development

Meet your mentors (us)

Your Backend Mentors

Hiya Mavani

(Junior, CS & DS)

- Backend Mentor
- Loves playing Ping-Pong, anime/cartoons, and video games

Ayush Mishra

(Junior, CS & Math & DS)

- Backend Mentor
- Loves hiking, biking, guitar, and Ping-Pong

Liam Ta

(Sophomore, CS)

- Backend Mentor
- Loves skiing, bouldering, and playing the guitar with friends

Accelerator Overview

What's in the future?

In this program, you'll learn how to...

- Work with JavaScript and Node
- Set up a web server / API with Express
- Interact with a Supabase database
- Learn different uses of databases and web servers
- Understand and use a simple Auth
- Create your own barebones application

"Spicy" Workshops

- Regular Expressions
- Postman and OpenAPI
- Database Security
- CI/CD

Links & Stuff

GitHub

Learning JavaScript

What is JavaScript?

- “a scripting or programming language that allows you to implement complex features on web pages”
- Syntactically similar to Java (very much so)
- Provides functionality to 98.8% of websites

How do we use it?

Node.js is a back-end JavaScript runtime environment, runs on the V8 JavaScript engine, and executes JavaScript code outside a web browser. (wikipedia)

Node.js provides an environment for us to deploy and run our code!

What is it used for?

- Node.js and JavaScript can be implemented in many applications
  - Command-line
  - High internet traffic
  - Websockets
- In general, it is used in **real-time applications**
  - services many users in <u>real-time</u>
  - communication between <u>client and server</u> and <u>server to client</u>
  - interfaces with external APIs or databases

Let’s get started!

Variables and types

- Variables are “containers” for storing data values
- Variables can have different types
- Types:

string	“Hello World”
number	1
bool	True, False
object	[1,2,3,4,5]

Variables and types

- Behavior of a variable is defined by its type (*strongly typed)*
  - *Adding #s, appending strings*
- Type of a variable is decided by its value (*dynamically typed)*
  - Type can be reassigned

```
// This is a comment

variable1 = "Hello World"   // A string type
print(variable1)

variable2 = 1               // A number type
print(variable2)

variable3 = true            // A boolean type
console.log(variable3)

// Let's change a boolean type to a string type!
variable3 = variable3.toStr()
console.log(variable3)
```

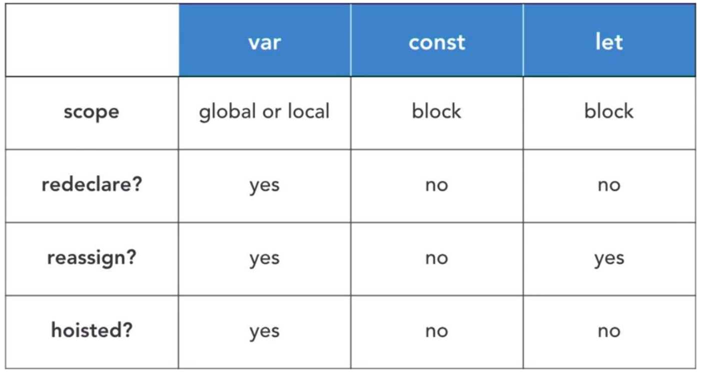
Variable naming rules

- Variables are ***cAsE-sEnSiTiVe***
- May contain any alphanumeric characters
- May contain underscores
- Must start with an underscore or letter

```
_variable1 = "Hello World"
// A variable name can start with an underscore

variable2 = 1234
// A variable name can start with any letter
(case doesn't matter)
Variable3 = 5678

4invalid = "Error!"
// A variable name can't start with a number.

var iable4 = "1"
// A variable name can't contain whitespace.
```

Ways to define variables

- Variables can be defined with any of the following:
  - let, const, var
- Each has a different functionality



	var	const	let
scope	global or local	block	block
redeclare?	yes	no	no
reassign?	yes	no	yes
hoisted?	yes	no	no

```
var random_variable = 123; // This can be updated and

// Both statements are allowed
random_variable = 0;
var random_variable = 246;

const answer_to_the_universe = 42;  // This constant can
not be redeclared or updated!

// Both statements will fail!
answer_to_the_universe = 41;
const answer_to_the_universe = 67;

let declared_by_let = 15;

declared_by_let = 16;       // This is allowed
let declared_by_let = 0;    // Redeclaration is not
allowed
```

Operators

- Used for comparing and performing operations
- Can be used on variables
- Standard operators
  - +, -, *, /, %
  - ++, --
- Comparators
  - ==, >=, <=, !=
- Misc.
  - Exponentiate **

```
sum_1 = 9 + 5
sum_1 += 11
// Same as "sum_1 = sum_1 + 11"
sum1++

product_1 = 5 * 4
power_of = 2 ** 3 // 2 to the power of 3 = 8

modulo_11_5 = 11 % 5 // 1
```

Working with conditions

Conditions

- Keywords used:
  - **if**, **else**, **else if**
- Used to organize code by conditions
- E.g. “run this code if this condition is true, otherwise (else) run this condition”
- Can have many different conditions!

```
// Conditions[0]
username = "admin"
password = "password"

username2 = "rumad"
password2 = "rumad_password"

my_username = "ben"
my_password = "abcdefgh"
// Condition syntax
if (my_username == username && my_password == password){
    console.log("Logged into admin")
} else if (my_username == username && my_password ==
password2){
    console.log("Logged into rumad")
} else {
    console.log("Failed to log in")
}
```

Conditions

- There are many different ways to use conditions
- Conditions can be used as values in variables

```
# Conditions[1]
# You can write this...
if (my_username == username and my_password == password){
    logged_in = True
} else {
    logged_in = False
}

// or simply write this!
logged_in = (my_username == username && my_password ==
password)
```

Conditions

- Has different comparison operators
    - **==**, **===**, **<=**, **>=**, **!=**, **<**, **>**
- Equals to "**==**" (loose), "**===**" (strict, generally preferred)
- Less than or equal to "**<=**"
- Greater than or equal to "**>=**"
- Not equal to "**!=**"
- Less than "**<**"
- Greater than "**>**"

```
# Conditions[2]
prod_1 = 9*7     # 63
prod_2 = 3*15    # 45

if (prod_1 > prod_2){
    console.log("prod_1 greater than or equal to
prod_2")
} else if (prod_2 == prod_2){
    console.log("prod_1 equal to prod_2")
} else {
    console.log("prod_1 less than prod_2")
}

cash_in_wallet = 40
price = 75

can_purchase = cash_in_wallet >= price
print(can_purchase)
```

Loops

For Loops

- For loops are structured similar to Java.
- 2 parts to the loop
  - Loop conditions
    - Declaration of variable
    - Loop end condition
    - Iterator
  - Loop body
- Other variations as well…

```
# Dicts[0]
let array = [0,1,2,3,4,5]

for(let i = 0; i < array.length; i++)
console.log(array[i])
// Prints out all the numbers in the
array
```

How would I only print out every OTHER number?

For-Each Loops

- For each loops perform some action to each element of a list or set of data without indexing
- `numbers.forEach((e) => {})`
  - **numbers** is the list that the loop will iterate through

```
let numbers = [65, 44, 12, 4];
numbers.forEach((element) => {
    // Run code on each element of the
array
    console.log(element)
})
```

While Loops

- While loops are loops that run as long as a conditional is true
- Do-While loops vary from regular while loops
    - Checks conditional at the end instead of at the start

```
// Both do the same thing… kind of

while (i < 10) {
    console.log(i)
    i++
}
do {
    console.log(i)
    i++
} while (i < 10)
```

Objects in-depth

What's an object?

An object is a collection of properties, and a property is an association between a name (or key) and a value. A property's value can be a function, in which case the property is known as a method. (mdn web docs)

- What is considered an object in JavaScript?
    - Lists / Arrays
    - Dictionaries / HashMaps
- Can be used similarly to *classes* in Java

Use cases of objects

- Objects in JavaScript are flexible
- The most simple use cases are:
  - Lists
  - Dictionaries

```
variable4 = {}       // This is an **object**.
// It can be used similarly to a HashMap or dictionary
variable4['a'] = 1
variable4['b'] = "A"
// Object { a: 1, b: A}

// We can also instantiate it like this.
variable4 = {'a': 1, 'b': "A"}
variable5 = [5,1,3,4]    // This is also an **object**.
// We can use it just like a list.
variable5[0] = 2         // Array(4) [ 2, 1, 3, 4 ]
```

Objects as lists

- JavaScript is 0-indexed (lists start at 0!)
- Can contain any types and are not limited to one type per list
- Get length with **my_list.length**
- Get the **n**th item in a list with **my_list[n]**

```
# Lists[0]
let my_bools = [true,false,false]
if (my_bools[0] == true){
    console.log("Hello World!")
}
// output: Hello World!
```

Objects as lists (cont.)

- Is a class, so it has operations you can perform on it
- Items can be added, removed, and accessed
- Easily check if something exists in a list

```
// Lists[1]
my_ints = [1,2,3,4,5]
my_ints.push(6)
// "appends" 6 to end of the list
// my_ints: [1, 2, 3, 4, 5, 6]
```

```
my_any = [15,true,"apple"]
console.log(my_any.includes("apple")) // true
my_any = my_any.splice(0,1)
// my_any: [15, true]
console.log(my_any.includes("apple")) // False
// deletes the item at index=0
// my_any: [true]
```

Objects as dictionaries

- JavaScript is 0-indexed (lists start at 0!)
- Can contain any types and are not limited to one type per list
- Get the value for key with **my_dict[key]**

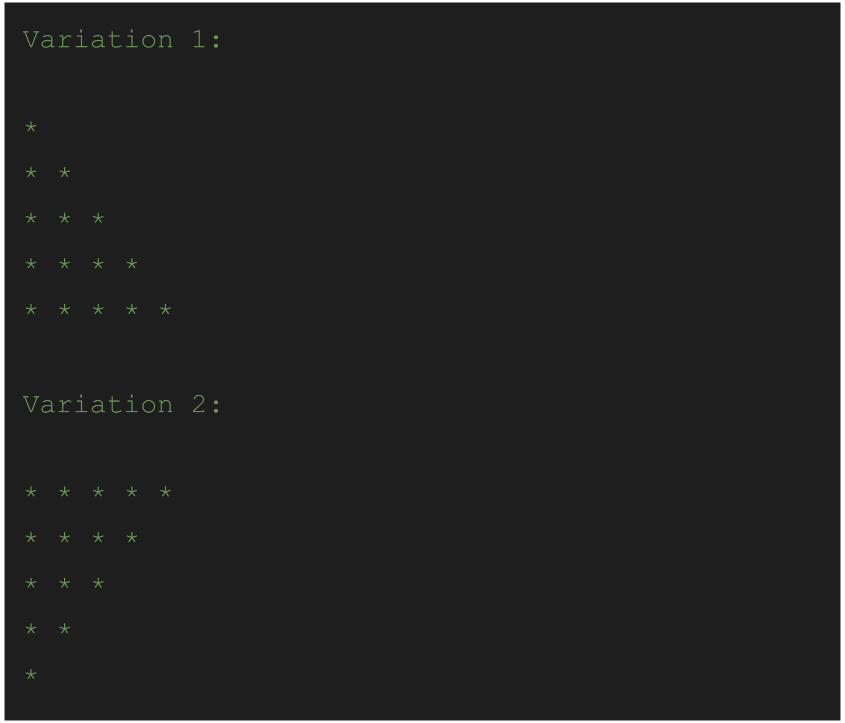
```
# Dicts[0]
let my_dict = {
    "key1": "abc",
    "key2": "def",
    "key3": "ghi"
}

my_dict["key1"] // abc
```

Hands-On Practice

Print a Pyramid

- Using what you learned, print out this pyramid front and back.
- Breakout rooms.

Hint: x.repeat(i) will repeat x i times



```
Variation 1:

*
* *
* * *
* * * *
* * * * *

Variation 2:

* * * * *
* * * *
* * *
* *
*
```

Questions?

Please fill out the feedback form when you have a chance!

Next week...

- Writing a function
- Using npm
- Experimenting with objects and data types

Feedback Form

Please let us know where we can improve the format of the lessons!