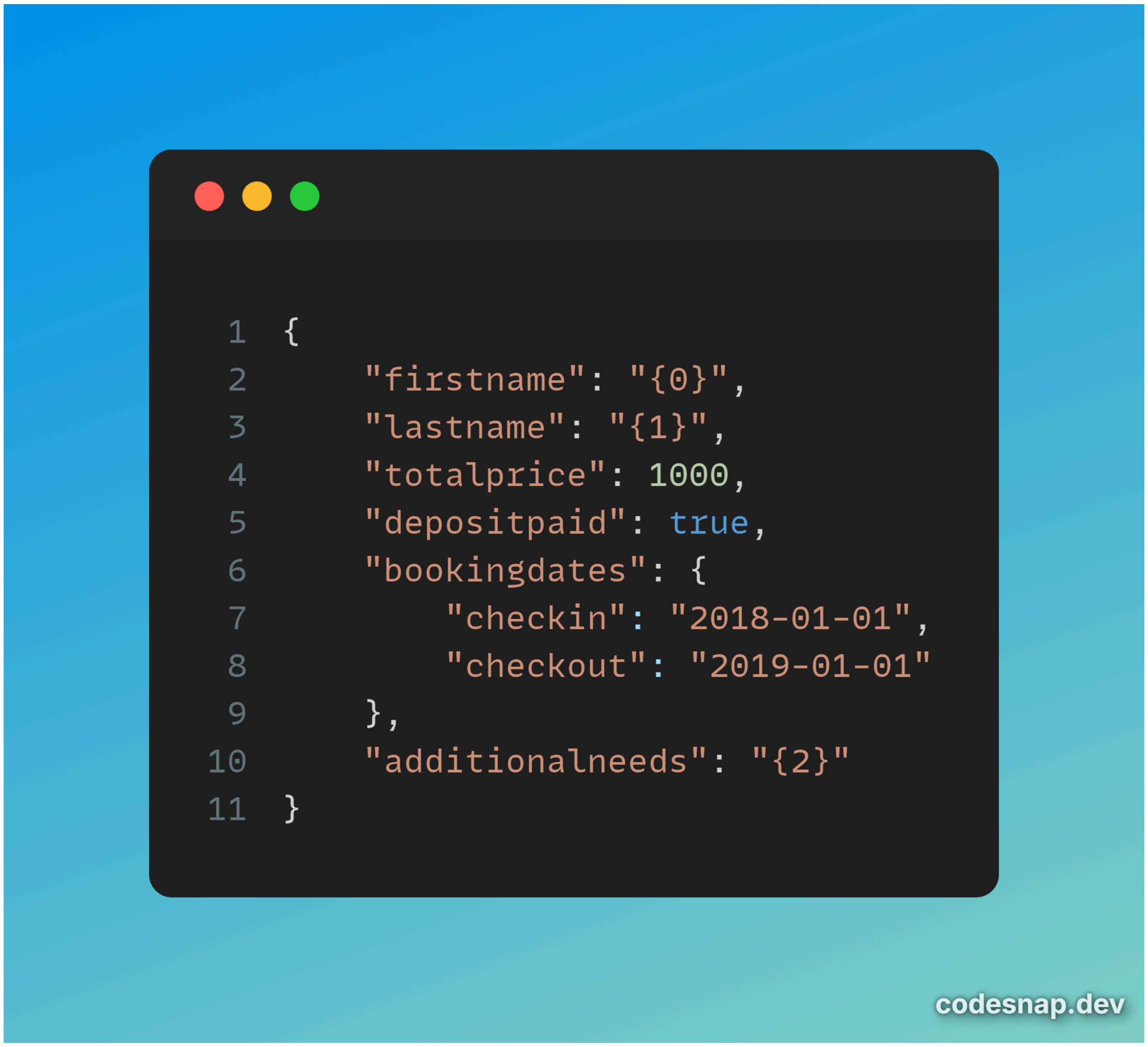
baka-pwapi-5

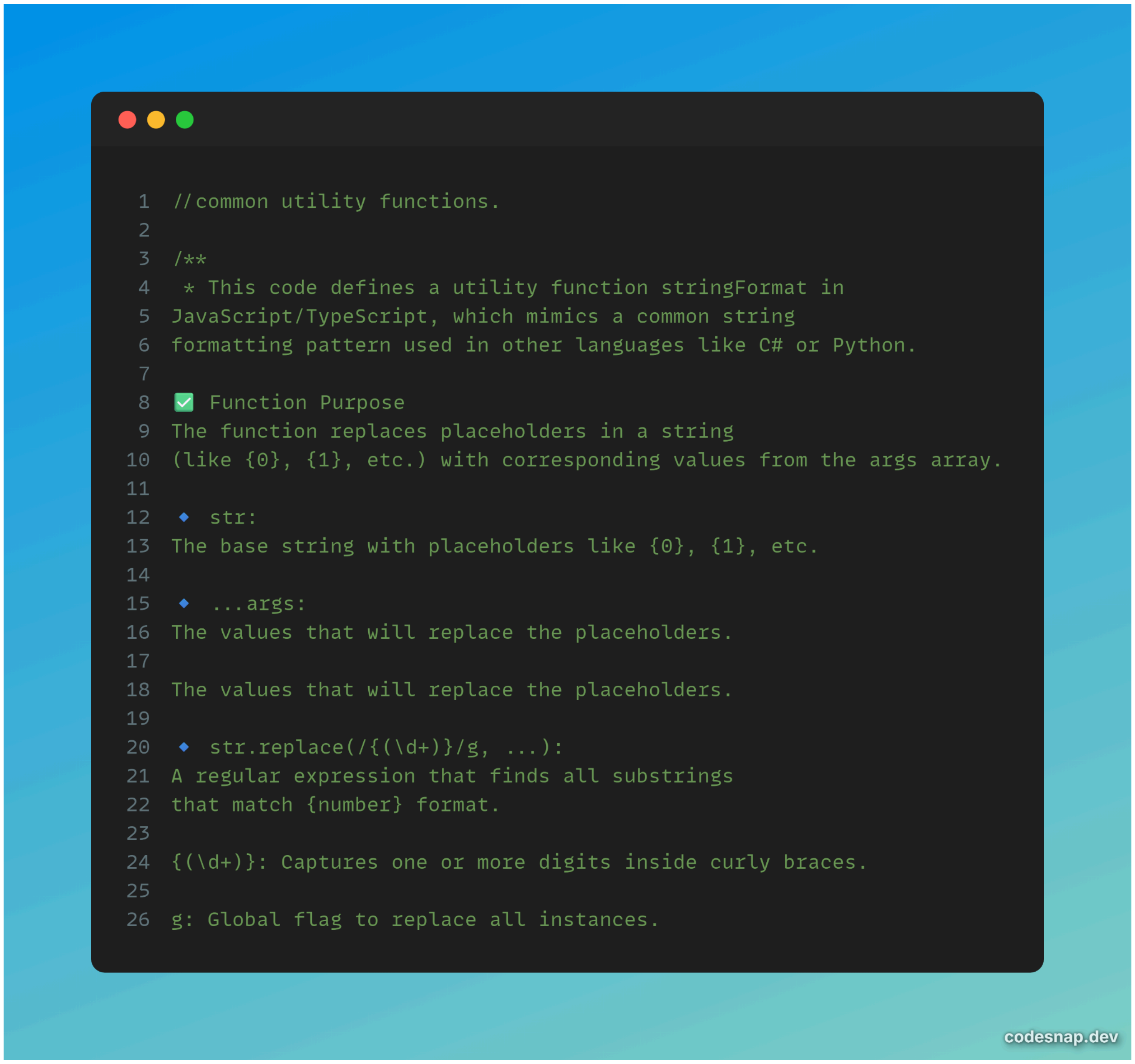
POST API Request using Dynamic JSON File

we need to give {1}. {2} as index numbers inside json file.

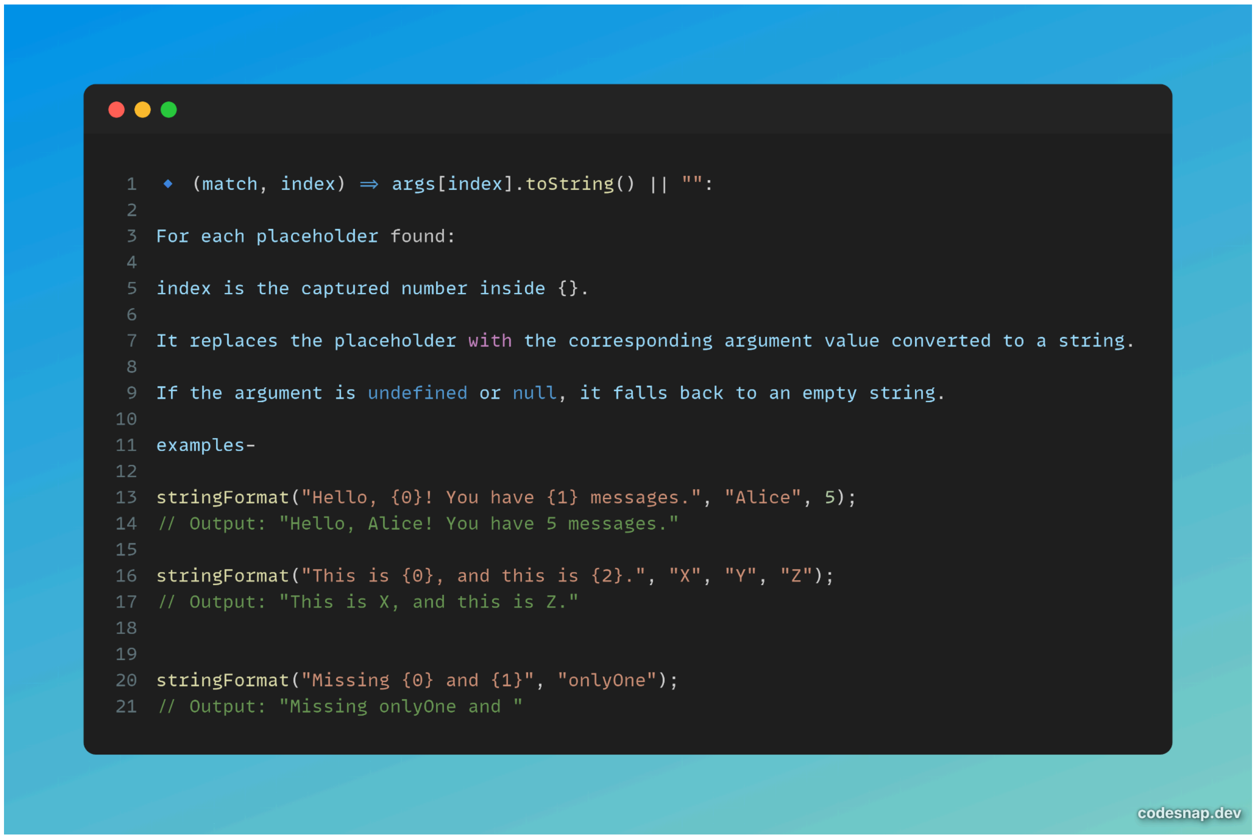
json data-



```json
{
    "firstname": "{0}",
    "lastname": "{1}",
    "totalprice": 1000,
    "depositpaid": true,
    "bookingdates": {
        "checkin": "2018-01-01",
        "checkout": "2019-01-01"
    },
    "additionalneeds": "{2}"
}
```

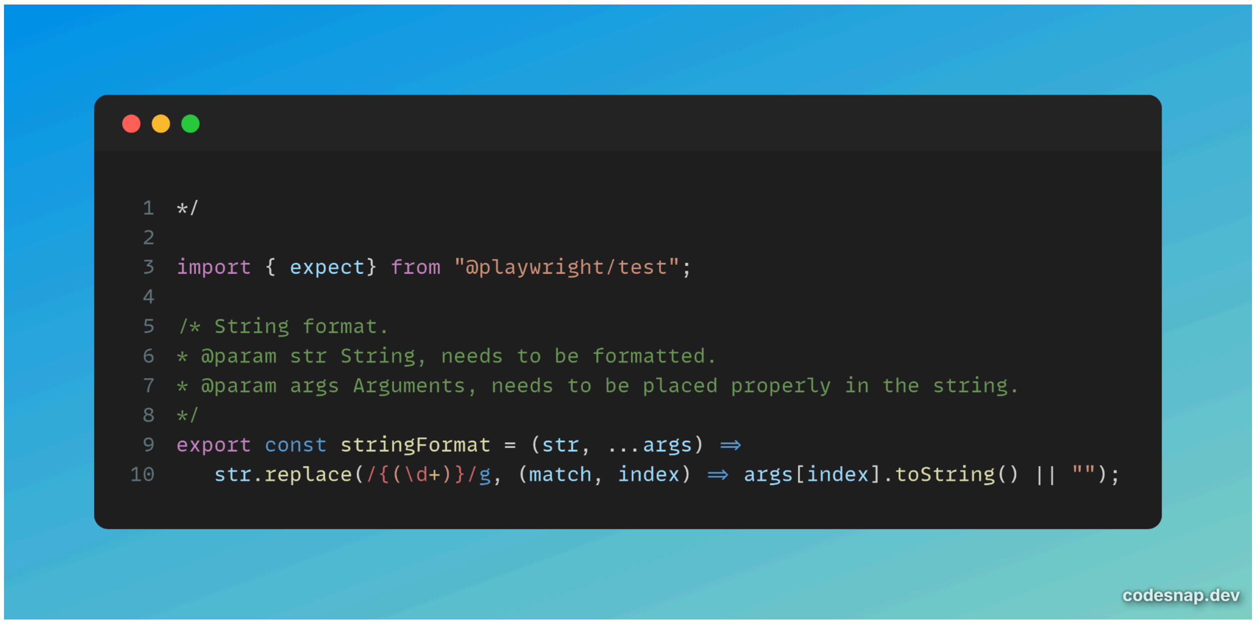


```
//common utility functions.

/**
 * This code defines a utility function stringFormat in
JavaScript/TypeScript, which mimics a common string
formatting pattern used in other languages like C# or Python.

✅ Function Purpose
The function replaces placeholders in a string
(like {0}, {1}, etc.) with corresponding values from the args array.

 🔹 str:
The base string with placeholders like {0}, {1}, etc.

 🔹 ...args:
The values that will replace the placeholders.

The values that will replace the placeholders.

 🔹 str.replace(/{(\d+)}/g, ...):
A regular expression that finds all substrings
that match {number} format.

{(\d+)}: Captures one or more digits inside curly braces.

g: Global flag to replace all instances.
```



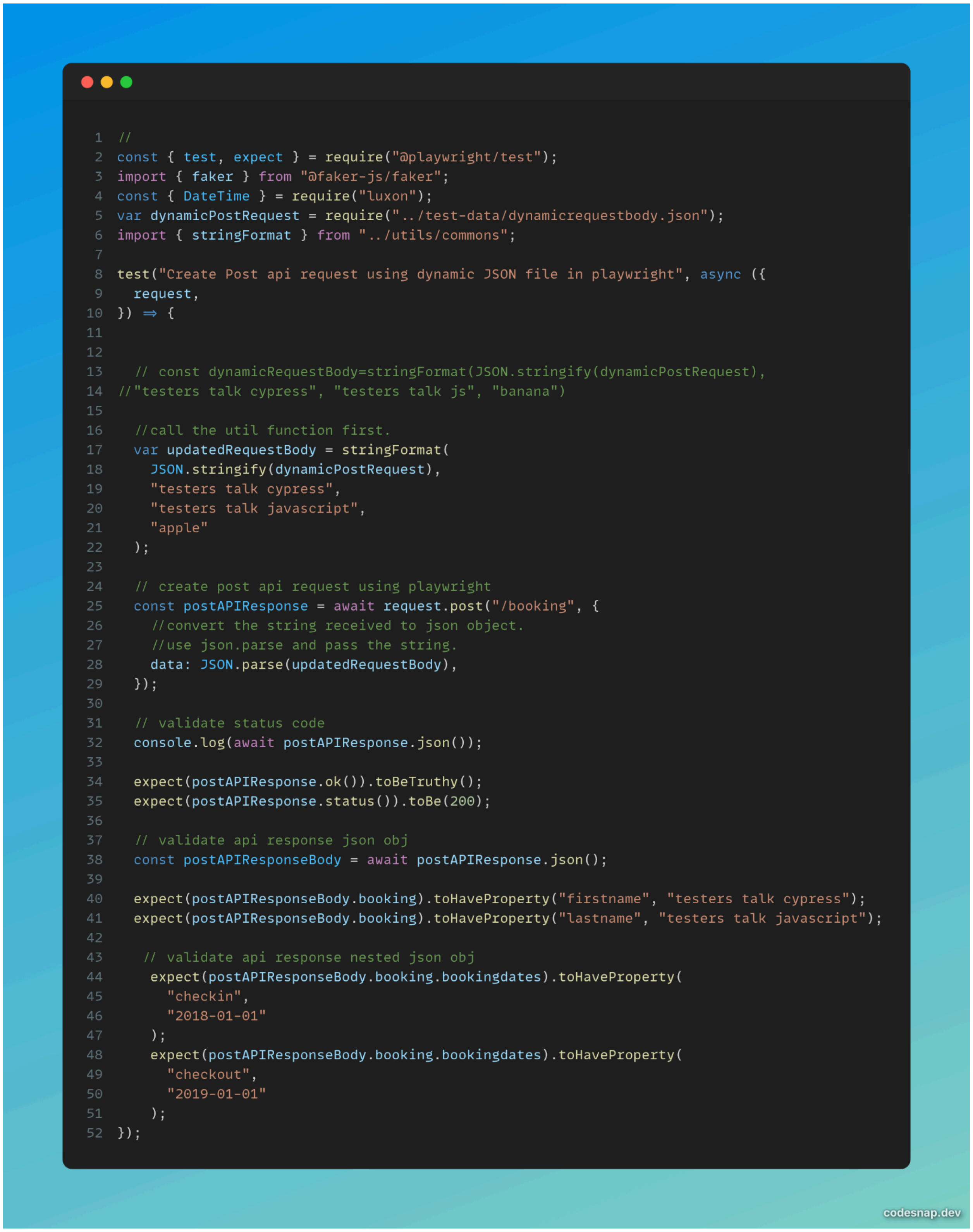
```
 ◆ (match, index) ⇒ args[index].toString() || "":

For each placeholder found:

index is the captured number inside {}.

It replaces the placeholder with the corresponding argument value converted to a string.

If the argument is undefined or null, it falls back to an empty string.

examples-

stringFormat("Hello, {0}! You have {1} messages.", "Alice", 5);
// Output: "Hello, Alice! You have 5 messages."

stringFormat("This is {0}, and this is {2}.", "X", "Y", "Z");
// Output: "This is X, and this is Z."


stringFormat("Missing {0} and {1}", "onlyOne");
// Output: "Missing onlyOne and "
```

codesnap.dev



```
*/

import { expect} from "@playwright/test";

/* String format.
* @param str String, needs to be formatted.
* @param args Arguments, needs to be placed properly in the string.
*/
export const stringFormat = (str, ...args) ⇒
   str.replace(/{(\d+)}/g, (match, index) ⇒ args[index].toString() || "");
```

codesnap.dev



```javascript
//
const { test, expect } = require("@playwright/test");
import { faker } from "@faker-js/faker";
const { DateTime } = require("luxon");
var dynamicPostRequest = require("../test-data/dynamicrequestbody.json");
import { stringFormat } from "../utils/commons";

test("Create Post api request using dynamic JSON file in playwright", async ({
  request,
}) => {


  // const dynamicRequestBody=stringFormat(JSON.stringify(dynamicPostRequest),
//"testers talk cypress", "testers talk js", "banana")

  //call the util function first.
  var updatedRequestBody = stringFormat(
    JSON.stringify(dynamicPostRequest),
    "testers talk cypress",
    "testers talk javascript",
    "apple"
  );

  // create post api request using playwright
  const postAPIResponse = await request.post("/booking", {
    //convert the string received to json object.
    //use json.parse and pass the string.
    data: JSON.parse(updatedRequestBody),
  });

  // validate status code
  console.log(await postAPIResponse.json());

  expect(postAPIResponse.ok()).toBeTruthy();
  expect(postAPIResponse.status()).toBe(200);

  // validate api response json obj
  const postAPIResponseBody = await postAPIResponse.json();

  expect(postAPIResponseBody.booking).toHaveProperty("firstname", "testers talk cypress");
  expect(postAPIResponseBody.booking).toHaveProperty("lastname", "testers talk javascript");

   // validate api response nested json obj
    expect(postAPIResponseBody.booking.bookingdates).toHaveProperty(
      "checkin",
      "2018-01-01"
    );
    expect(postAPIResponseBody.booking.bookingdates).toHaveProperty(
      "checkout",
      "2019-01-01"
    );
});
```