

# Correction TP 02 AWA

## API fetch

# Question 1.7 – récupérer la température en °C

The screenshot shows the Postman application interface. A GET request is being made to `https://api.openweathermap.org/data/2.5/weather?q=Grenoble&appid=87c1b851`. The 'Params' tab is selected, showing query parameters `q=Grenoble` and `appid=87c1b851`. The 'Body' tab displays the JSON response from the API. A red arrow points to the `temp` field in the JSON, which has a value of `269.02`. A callout bubble with the text "Par défaut la température est en ° Kelvin" is positioned over this field.

```
9   "main": "Clear",
10  "description": "clear sky",
11  "icon": "01d"
12 ]
13 "base": "stations",
14 "main": {
15   "temp": 269.02,
16   "feels_like": 269.02,
17   "temp_min": 269.
18   "temp_max": 269.
19   "pressure": 1012.5
20 }
```

Par défaut la température est en ° Kelvin

# Question 1.7 – récupérer la température en °C

<https://openweathermap.org/current>

The screenshot shows the 'Call current weather data' section of the OpenWeatherMap API documentation. It includes a heading 'How to make an API call' with a code example:

```
https://api.openweathermap.org/data/2.5/weather?lat=[lat]&lon=[lon]&appid=[API key]
```

Below this, there's a 'Parameters' table:

Parameter	Type	Description
lat	required	Latitude. If you need the geocoder to automatically convert city names and zip-codes to geo coordinates and the other way around, please use our <a href="#">Geocoding API</a>
lon	required	Longitude. If you need the geocoder to automatically convert city names and zip-codes to geo coordinates and the other way around, please use our <a href="#">Geocoding API</a>
appid	required	Your unique API key (you can always find it on your account page under the "API key" tab)
mode	optional	Response format. Possible values are <code>xml</code> and <code>html</code> . If you don't use the <code>mode</code> parameter format is JSON by default. <a href="#">Learn more</a>
units	optional	Units of measurement. <code>standard</code> , <code>metric</code> and <code>imperial</code> units are available. If you do not use the <code>units</code> parameter, <code>standard</code> units will be applied by default. <a href="#">Learn more</a>
lang	optional	You can use this parameter to get the output in your language. <a href="#">Learn more</a>

On the right side, there's a sidebar with links like 'Product concept', 'Call current weather data', 'How to make an API call', etc. A red circle labeled '1' highlights the 'How to make an API call' link.

A red box labeled '2' highlights the 'units' parameter description in the 'Parameters' table. A red arrow labeled '3' points from this box to a red box labeled '4' on the right side of the page.

Consulter la documentation de l'API

This screenshot shows the 'Units of measurement' section of the API documentation. It includes a 'Parameters' table and a box containing information about temperature units:

Parameter	Type	Description
units	optional	<code>standard</code> , <code>metric</code> , <code>imperial</code> . When you do not use the <code>units</code> parameter, format is <code>standard</code> by default.

**Temperature is available in Fahrenheit, Celsius and Kelvin units.**

- For temperature in Fahrenheit use `units=imperial`
- For temperature in Celsius use `units=metric`
- Temperature in Kelvin is used by default, no need to use `units` parameter in API call

[List of all API parameters with units openweathermap.org/weather-data](#)

**Examples of API calls:**

Standard

```
https://api.openweathermap.org/data/2.5/weather?lat=[lat]&lon=[lon]&appid=[API key]
```

Il faut rajouter à la requête le paramètre **units** avec comme valeur **metric**

# Question 1.7 – récupérer la température en °C

The screenshot shows the Postman application interface. A GET request is being made to `https://api.openweathermap.org/data/2.5/weather?q=Grenoble&appid=87c1b...&units=metric`. The 'Params' tab is selected, showing three parameters: `q` (Grenoble), `appid` (87c1b...), and `units` (metric). A tooltip explains that for a GET request, parameters are added to the URL as `nomDuParamètre=valeur`, separated by &. The 'Body' tab shows the JSON response from the API, which includes temperature data like `temp: -2.46` and `feels_like: -2.46`. A red box highlights the `units` parameter in the Params table, with a callout '1' pointing to it. Another red box highlights the 'Send' button, with a callout '2' pointing to it. A red arrow points from the 'units' parameter in the Params table to the `units` value in the JSON response body, with a callout '3' pointing to it.

Rajout du paramètre **units** avec la valeur **metric**

1

2

3

La requête HTTP étant un GET le paramètre unit et sa valeur sont insérés dans l'URL sous la forme **nomDuParamètre=valeur** et séparé du paramètre précédent par &

Key	Value
<input checked="" type="checkbox"/> q	Grenoble
<input checked="" type="checkbox"/> appid	87c1b...
<input checked="" type="checkbox"/> units	metric

Send

Code Cookies

Status: 200 OK Time: 1297 ms Size: 866 B

```
9     "main": "Clear",
10    "description": "clear sky",
11    "icon": "01d"
12  }
13  ],
14  "base": "stations",
15  "main": [
16    {
17      "temp": -2.46,
18      "feels_like": -2.46,
19      "temp_min": -2.46,
20      "temp_max": -2.46,
21      "pressure": 1027,
22      "humidity": 54
23    }
24  ]
}
```

Renvoyer à nouveau la requête

# Question 1.8 – récupérer la description en français

The screenshot shows the Postman application interface. The top navigation bar includes File, Edit, Selection, View, Go, and a search bar labeled "FetchAPI". Below the navigation is a tab bar with "POSTMAN", "tp02\_ApplisWeb.html", "testOpenWeather.js", and "Current Weather". The main workspace displays a "New HTTP Request" with a "GET" method and the URL "https://api.openweathermap.org/data/2.5/weather?q=Grenoble&lang=fr". The "Query Params" table contains four entries: "q" with value "Grenoble", "appid" with value "87c1b851d6f616e66b84d35c85fe43de", "units" with value "metric", and "lang" with value "fr". The "Body" section is collapsed. At the bottom, a status message indicates "Status: 200 OK Time: 256 ms Size: 861 B". To the right, a "Code snippet" panel titled "JavaScript - Fetch" shows the following code:

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/  
    data/2.5/weather?q=Grenoble&  
    appid=87c1b851d6f616e66b84d35c85fe4  
    3de&units=metric&lang=fr",  
    requestOptions)  
  .then(response => response.text())  
  .then(result => console.log(result))  
  .catch(error => console.log('error',  
    error));
```

# Question 1.8 – récupérer la description en français

<https://openweathermap.org/current>

Call current weather data

### How to make an API call

**API call**

```
https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}
```

**Parameters**

- lat** required Latitude. If you need the geocoder to automatically convert city names and zip-codes to geo coordinates and the other way around, please use our [Geocoding API](#)
- lon** required Longitude. If you need the geocoder to automatically convert city names and zip-codes to geo coordinates and the other way around, please use our [Geocoding API](#)
- appid** required Your unique API key (you can always find it on your account page under the "API key" tab)
- mode** optional Response format. Possible values are `xml` and `html`. If you don't use the `mode` parameter format is JSON by default.  
[Learn more](#)
- units** optional Units of measurement. `standard`, `metric` and `imperial` units are available. If you do not use the `units` parameter, `standard` units will be applied by default.  
[Learn more](#)
- lang** optional You can use this parameter to get the output in your language.  
[Learn more](#)

Please use [Geocoder](#) if you need automatic conversion of cities and zip-codes

**Product concept**  
Call current weather data **1**  
How to make an API call

**API response**

- JSON format API response example
- JSON format API response fields
- XML format API response example
- XML format API response fields
- List of weather condition codes
- Min/max temperature in current weather
- API and forecast API

**Bulk downloading**

**Other features**

- Geocoding API
- Built-in geocoding
  - Built-in API request by city name
  - Built-in API request by city ID
  - Built-in API request by ZIP code

**Format**

- Units of measurement
- Multilingual support
- Call back function for JavaScript code

Chat with Ulla - OpenWeather AI assistant

Consulter la documentation de l'API

OpenWeather

### Multilingual support

You can use the `lang` parameter to get the output in your language.

Translation is applied for the `city name` and `description` fields.

### API call

```
https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}&lang={lang}
```

**Parameters**

<code>lang</code>	optional	Language code
-------------------	----------	---------------

### Examples of API calls

```
https://api.openweathermap.org/data/2.5/weather?id=524901&lang=fr&appid={API key}
```

We support the following languages that you can use with the corresponding lang values:

- sq Albanian
- af Afrikaans
- ar Arabic
- de German
- es Spanish
- fi Finnish
- fr French **4**
- gl Galician
- it Italian

Il faut rajouter à la requête le paramètre **lang** avec comme valeur **fr**

# Question 1.8 – récupérer la description en français

The screenshot shows the Postman application interface. A GET request is being made to `https://api.openweathermap.org/data/2.5/weather?q=Grenoble&appid=87c1...&units=metric&lang=fr`. The 'lang' parameter is highlighted with a red box and circled with a yellow number 1. The 'Send' button is circled with a yellow number 2. A callout box with the text "Rajout du paramètre **lang** avec la valeur **fr**" points to the 'lang' entry in the query parameters table. Another callout box with the text "Constater que la description des conditions météorologiques est maintenant reçue en français" points to the JSON response body, specifically to the "description" field of a weather forecast entry. The response status is 200 OK.

Rajout du paramètre **lang** avec la valeur **fr**

1

2

3

Constater que la description des conditions météorologiques est maintenant reçue en français

FetchAPI

File Edit Selection View Go Run ... ↺ ↻

POSTMAN My Workspace New HTTP Request

OpenWeather API / Current Weather

HTTP Current Weather

Save View Documentation No Environment

GET https://api.openweathermap.org/data/2.5/weather?q=Grenoble&appid=87c1...&units=metric&lang=fr Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

Key	Value
q	Grenoble
appid	87c1...
units	metric
lang	fr

Code Cookies

Status: 200 OK Time: 256 ms Size: 861 B

Body Cookies Headers (9) Test Results

Pretty Raw Preview JSON

```
1 {  
2   "coord": {  
3     "lon": 5.8333,  
4     "lat": 45.0833  
5   },  
6   "weather": [  
7     {  
8       "id": 800,  
9       "main": "Clear",  
10      "description": "ciel dégagé",  
11      "icon": "01d"  
12    },  
13  ],  
14  "base": "stat",  
15  "main": "Clouds",  
16  "clouds": {  
17    "all": 100  
18  },  
19  "visibility": 10000,  
20  "wind": {  
21    "speed": 2.2,  
22    "deg": 180  
23  },  
24  "atmosphere": {  
25    "pressure": 1012,  
26    "humidity": 81  
27  },  
28  "dt": 1575530200,  
29  "sys": {  
30    "pod": "d",  
31    "country": "FR",  
32    "sunrise": 1575513000,  
33    "sunset": 1575520400  
34  },  
35  "timezone": 3600,  
36  "id": 2000100  
37}
```

Philippe GENOUD - UGA - 2025

Renvoyer à nouveau la requête

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

The screenshot shows a development environment with two main windows:

- Postman:** A dark-themed interface for making HTTP requests. A GET request is being prepared to <https://api.openweathermap.org/data/2.5/weather?q=Grenoble&appid=87c1b851d6f616e66b84d35c85fe43de&units=metric&lang=fr>. The "Code snippet" button is highlighted with a red circle labeled 1.
- VS Code:** An integrated code editor showing a file named `testOpenWeather.js`. It contains a JavaScript code snippet for making a `fetch` request to the same URL as in Postman, using the options provided in the Postman request. A tooltip with a red circle labeled 2 points to the code in the editor, instructing to copy it into the file.

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
  requestOptions)  
  .then(response => response.text())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

Que fait ce programme et quel affichage produit-il ?

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
requestOptions)  
  .then(response => response.text())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

Que fait ce programme et quel affichage produit-il ?

effectue la requête HTTP pour connaître les conditions météorologiques courantes à Grenoble et affiche la réponse brute (*raw data*) retournée par le serveur  
[api.openweathermap.org](https://api.openweathermap.org)

```
> node . testOpenWeather.js  
Terminé !  
{"coord":{"lon":5.8333,"lat":45.0833}, "weather":[{"id":801,"main":"Clouds",  
"description":"peu  
nuageux","icon":"02d"}], "base":"stations", "main":{"temp":7.9,"feels_like":7  
.9,"temp_min":7.9,"temp_max":7.9,"pressure":1025,"humidity":62,"sea_level":  
1025,"grnd_level":902}, "visibility":10000, "wind":{"speed":0.86,"deg":358,"g  
ust":1.78}, "clouds":{"all":14}, "dt":1738679554, "sys":{"country":"FR", "sunri  
se":1738651987, "sunset":1738687683}, "timezone":3600, "id":3014727, "name": "Gr  
enoble", "cod":200}
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
requestOptions)  
  .then(response => response.text())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

Pourquoi Terminé ! est-il affiché avant la réponse du serveur ?

```
> node . testOpenWeather.js  
Terminé !  
{"coord":{"lon":5.8333,"lat":45.0833}, "weather":[{"id":801,"main":"Clouds",  
"description":"peu  
nuageux","icon":"02d"}], "base":"stations", "main":{"temp":7.9,"feels_like":7  
.9,"temp_min":7.9,"temp_max":7.9,"pressure":1025,"humidity":62,"sea_level":  
1025,"grnd_level":902}, "visibility":10000, "wind":{"speed":0.86,"deg":358,"g  
ust":1.78}, "clouds":{"all":14}, "dt":1738679554, "sys":{"country":"FR", "sunri  
se":1738651987, "sunset":1738687683}, "timezone":3600, "id":3014727, "name": "Gr  
enoble", "cod":200}
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
requestOptions)  
  .then(response => response.text())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

Pourquoi Terminé ! est-il affiché avant la réponse du serveur ?

Parce que **fetch** est une fonction **asynchrone**, elle **lance le traitement** qui consiste à envoyer la requête HTTP et à attendre la réponse du serveur **et rend immédiatement la main**. De même les fonctions **then** et **catch** qui définissent les traitements à effectuer après que **fetch** ait reçu une réponse du serveur sont elles aussi **asynchrones**.

```
> node . testOpenWeather.js  
Terminé !  
{"coord":{"lon":5.8333,"lat":45.0833}, "weather":[{"id":801,"main":"Clouds",  
"description":"peu  
nuageux","icon":"02d"}], "base":"stations", "main":{"temp":7.9,"feels_like":7  
.9,"temp_min":7.9,"temp_max":7.9,"pressure":1025,"humidity":62,"sea_level":  
1025,"grnd_level":902}, "visibility":10000, "wind":{"speed":0.86,"deg":358,"g  
ust":1.78}, "clouds":{"all":14}, "dt":1738679554, "sys":{"country":"FR", "sunri  
se":1738651987, "sunset":1738687683}, "timezone":3600, "id":3014727, "name": "Gr  
enoble", "cod":200}  
>
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
requestOptions)  
  .then(response => response.text())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

Préciser l'ordre dans lequel les instructions du programme sont exécutées et ce qu'elles font.

```
> node . testOpenWeather.js
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
  .then(response => response.text())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

- 1 Définit un objet d'options référencé par la variable `requestOptions` qui servira à configurer la requête effectué par l'appel `fetch`

**method** spécifie le type de requête HTTP à effectuer. Les méthodes couramment utilisées sont :

GET : Récupérer des données depuis le serveur.  
POST : Envoyer des données au serveur.  
PUT : Mettre à jour des données existantes sur le serveur.  
DELETE : Supprimer des données sur le serveur.

**redirect** spécifie comment la requête `fetch` doit gérer les redirections HTTP (codes réponses 3xx) .

La valeur '`follow`' indique que la requête `fetch` suivra automatiquement les redirections vers la nouvelle location envoyée dans la réponse HTTP

```
> node . testOpenWeather.js
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
7       &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
8       requestOptions)  
9     .then(response => response.text())  
10    .then(result => console.log(result))  
11    .catch(error => console.log('error', error));  
12  
13 console.log('Terminé !');
```

- ② exécute la fonction **fetch** qui lance de manière asynchrone la requête HTTP pour interroger le serveur **openweathermap**.

Cette fonction retourne un objet promesse (**Promise**) qui sera résolue par la réponse du serveur

promesse initiale  
créée par **fetch**

status: pending  
value : undefined

**url** identifiant la ressource (le service web) à interroger

La requête étant un **GET** les paramètres peuvent être écrits directement dans l'url. Ils sont introduits par ? et définis par des couples **nomparamètre=valeur** séparés par &

> node . testOpenWeather.js

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7       requestOptions)  
8     .then(response => response.text())  
9     .then(result => console.log(result))  
10    .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

③

Envoie un message **then** à l'objet promesse retourné par **fetch**

**then** est asynchrone, elle renvoie une nouvelle promesse chaînée avec la précédente.

promesse initiale  
créeée par **fetch**

status: pending  
value : undefined

promesse créée  
par 1<sup>er</sup> **then**

status: pending  
value : undefined

Fonction fléchée (*arrow function*)

```
function(response) {  
  return response.text();  
}
```

cette fonction sera exécutée quand la promesse  
créée par **fetch** sera tenue

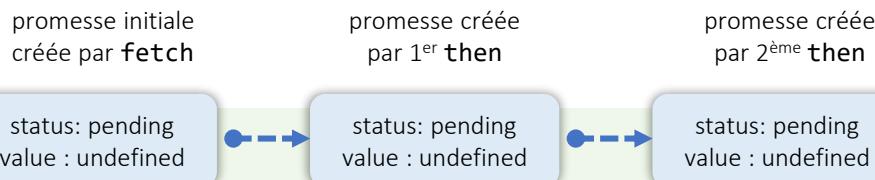
> node . testOpenWeather.js

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
8   .then(response => response.text())  
9   .then(result => console.log(result))  
10  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

4



Fonction fléchée (*arrow function*)

```
function(result) {  
  return console.log(result);  
}
```

cette fonction sera exécutée quand la promesse  
créée par le 1<sup>er</sup> then sera tenue

> node . testOpenWeather.js

4 Envoie le message **then** à l'objet promesse retourné  
précédemment (promesse retournée par le 1<sup>er</sup> **then**)

**then** est asynchrone, elle renvoie une nouvelle promesse  
chainée avec la précédente.

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

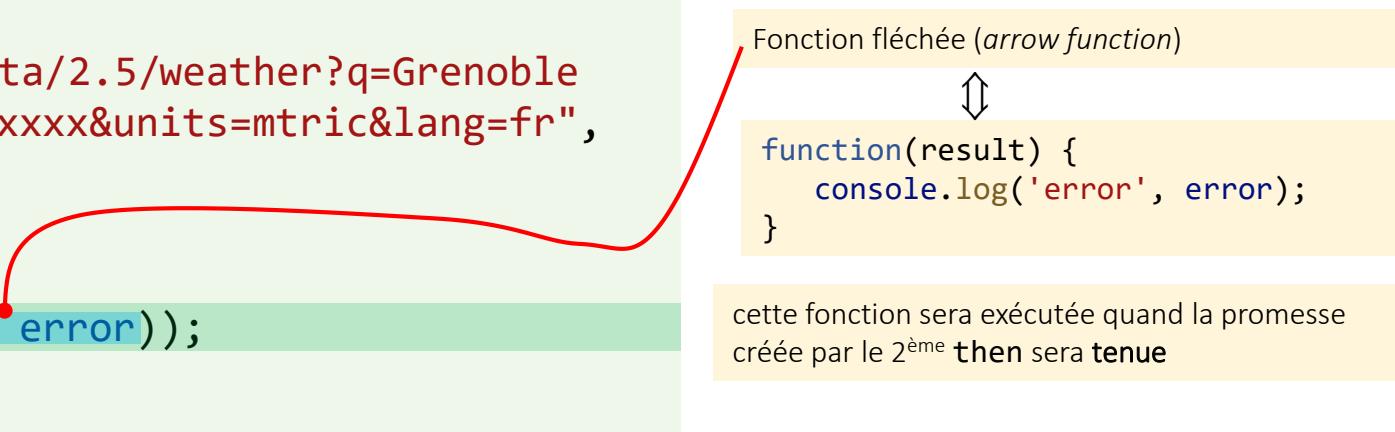
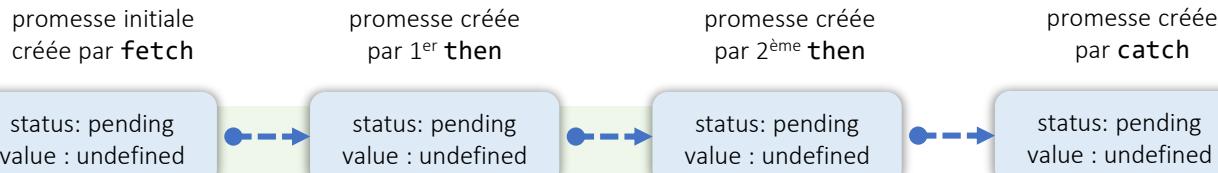
Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
  .then(response => response.text())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

5

Envoie le message **catch** à l'objet promesse retourné précédemment (promesse retournée par le 2<sup>ème</sup> **then**)

**catch** est asynchrone, elle renvoie une nouvelle promesse chainée avec la précédente.

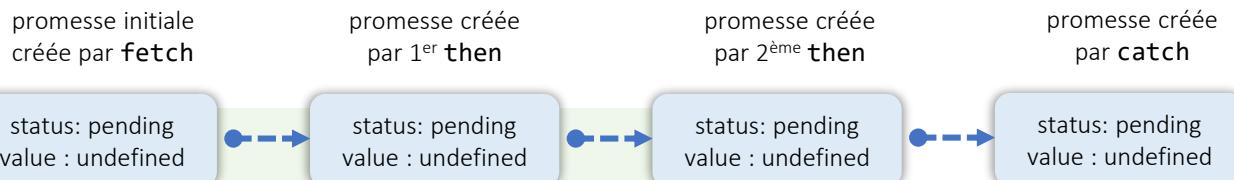


```
> node . testOpenWeather.js
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
requestOptions)  
  .then(response => response.text())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```



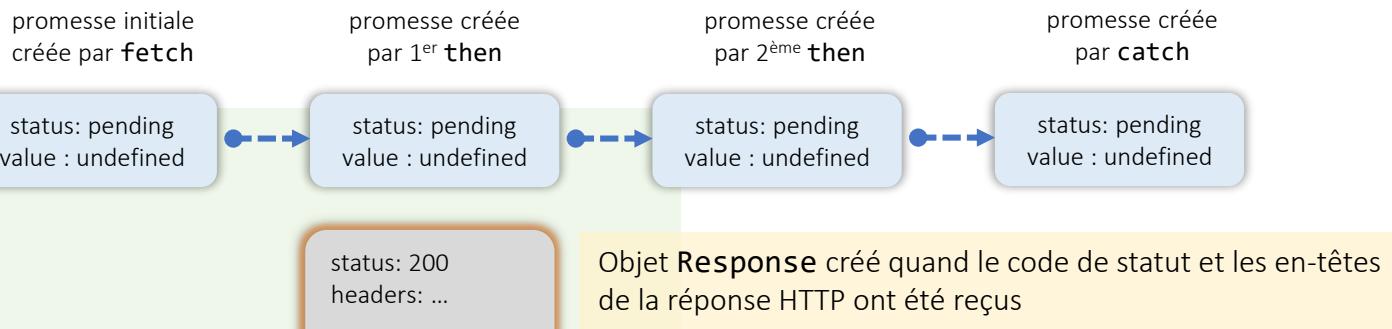
6 Affiche le message Terminé ! .

```
> node . testOpenWeather.js  
Terminé !
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
requestOptions)  
  .then(response => response.text())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

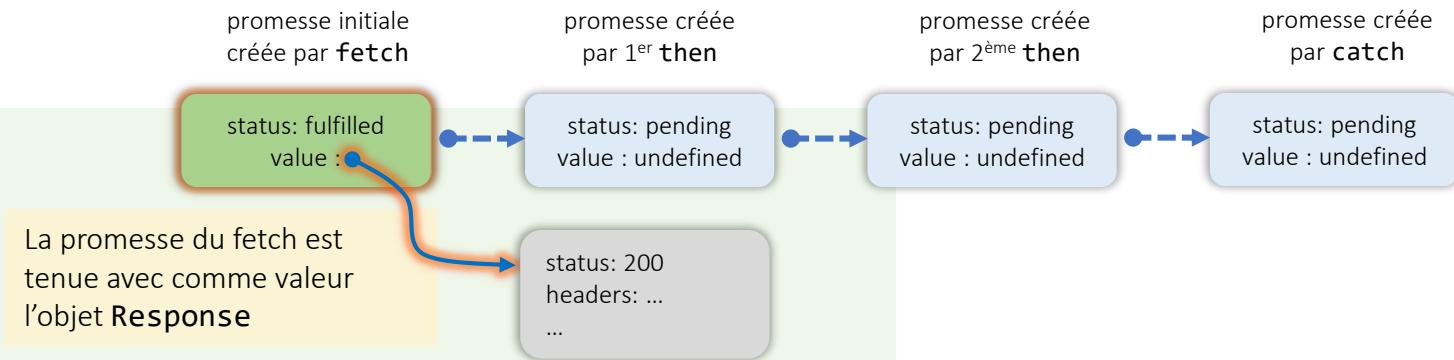


```
> node . testOpenWeather.js  
Terminé !
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
8     .then(response => response.text())  
9     .then(result => console.log(result))  
10    .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```



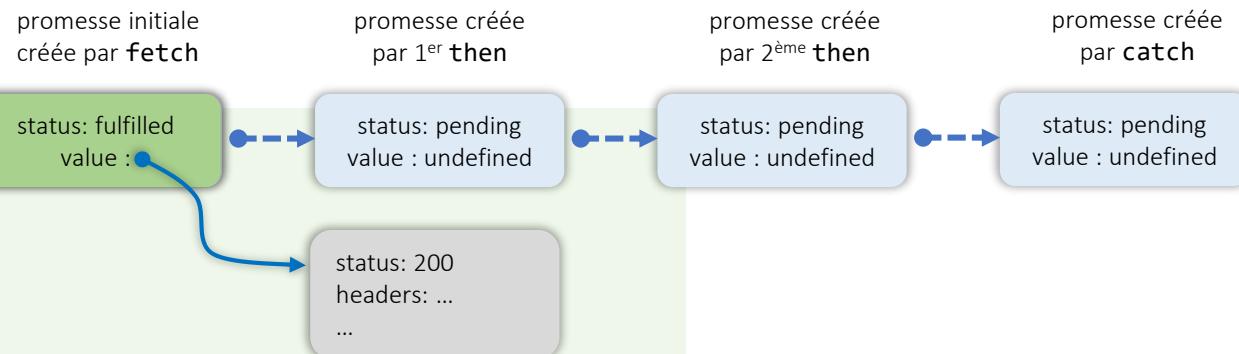
```
> node . testOpenWeather.js  
Terminé !
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
  .then(response => response.text())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

7 la fonction callback du 1<sup>er</sup> **then** est exécutée avec comme argument la valeur (l'objet Response) attachée à la promesse du **fetch**



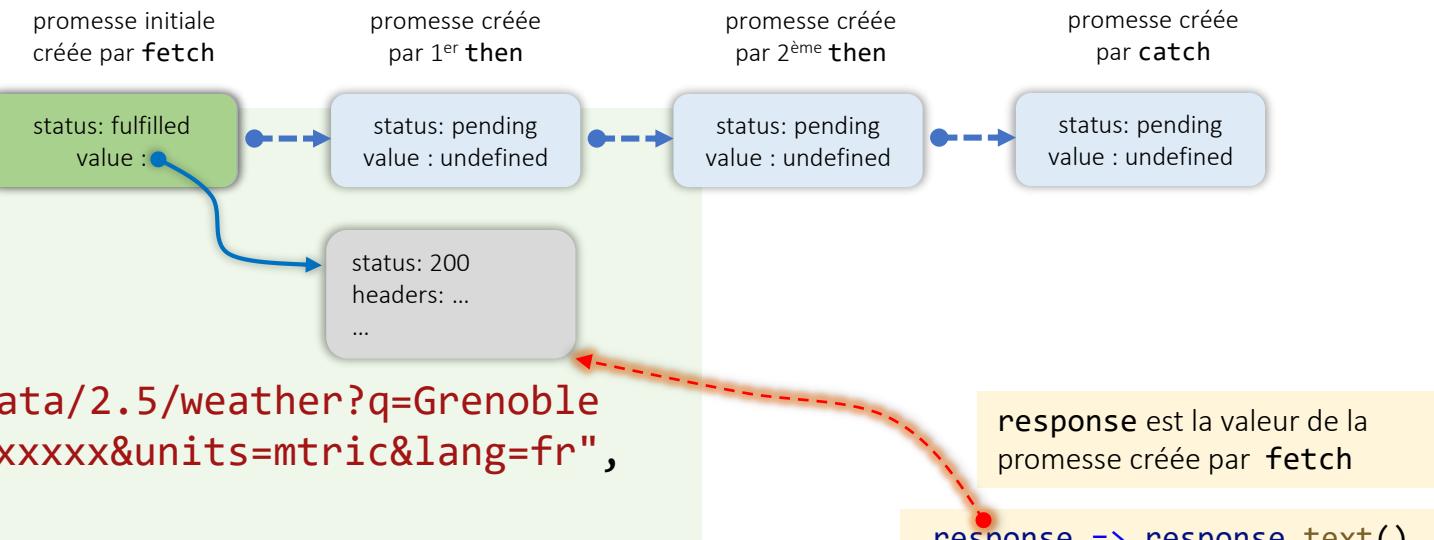
```
> node . testOpenWeather.js  
Terminé !
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
  .then(response => response.text())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

7 la fonction callback du 1<sup>er</sup> **then** est exécutée avec comme argument la valeur (l'objet Response) attachée à la promesse du **fetch**



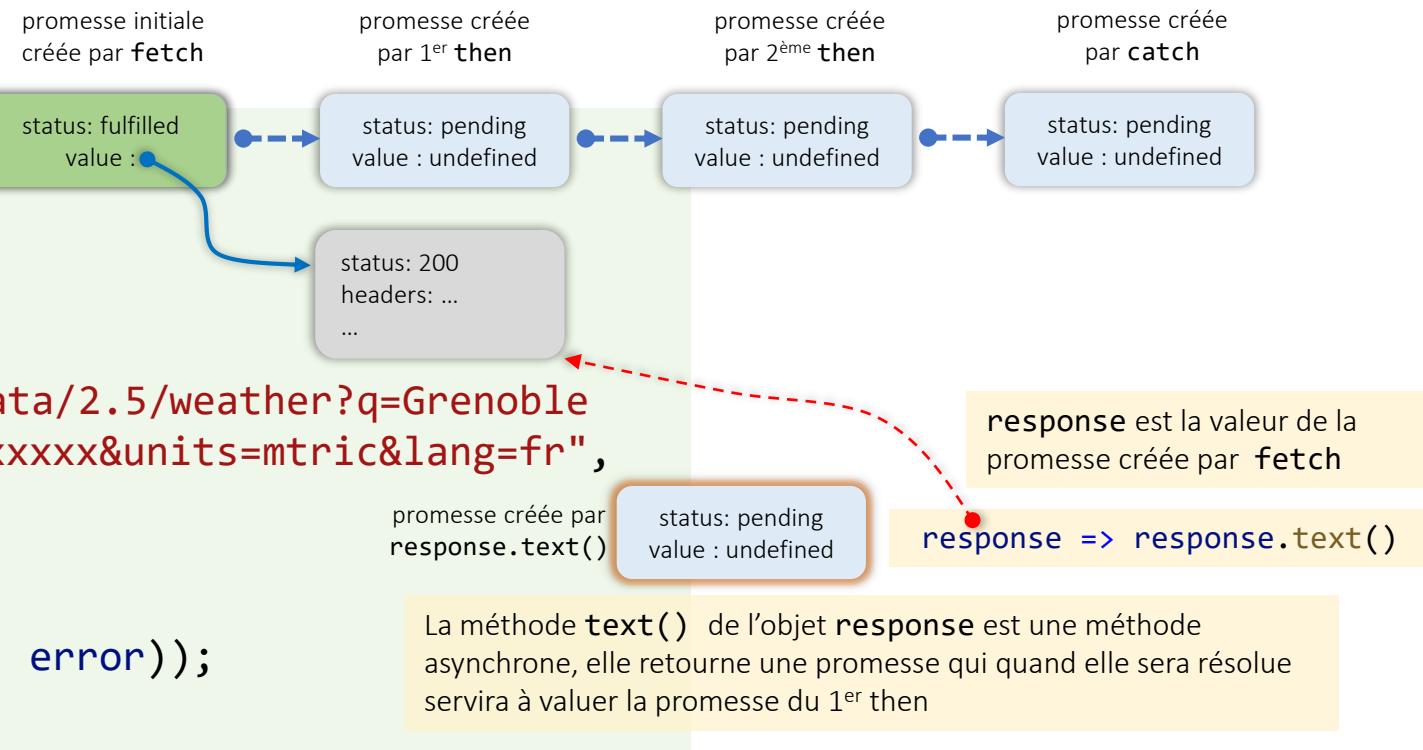
```
> node . testOpenWeather.js  
Terminé !
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
7       &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
8       requestOptions)  
9       .then(response => response.text())  
10      .then(result => console.log(result))  
11      .catch(error => console.log('error', error));  
12  
13 console.log('Terminé !');
```

7 la fonction callback du 1<sup>er</sup> **then** est exécutée avec comme argument la valeur (l'objet Response) attachée à la promesse du **fetch**

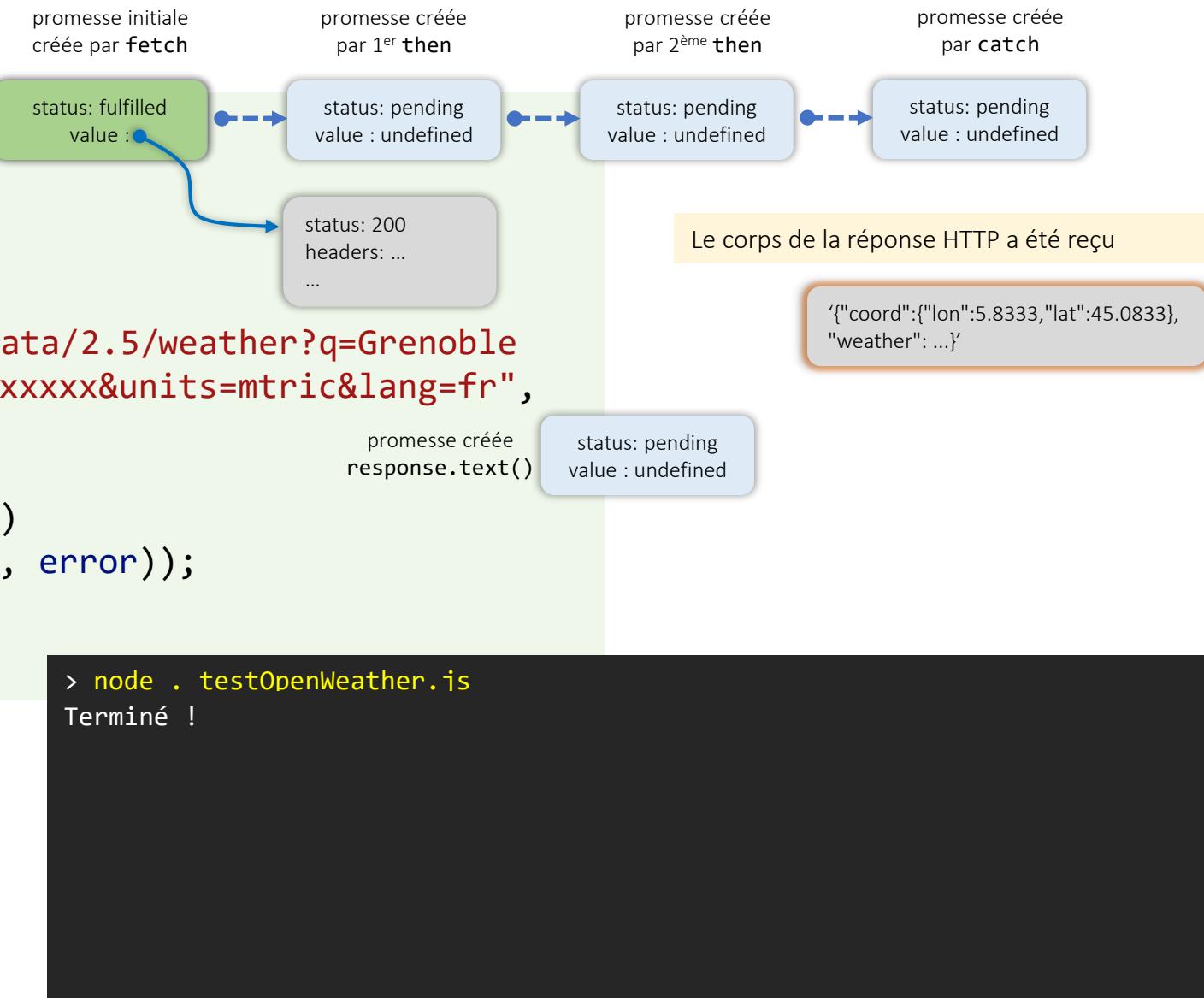


```
> node . testOpenWeather.js  
Terminé !
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

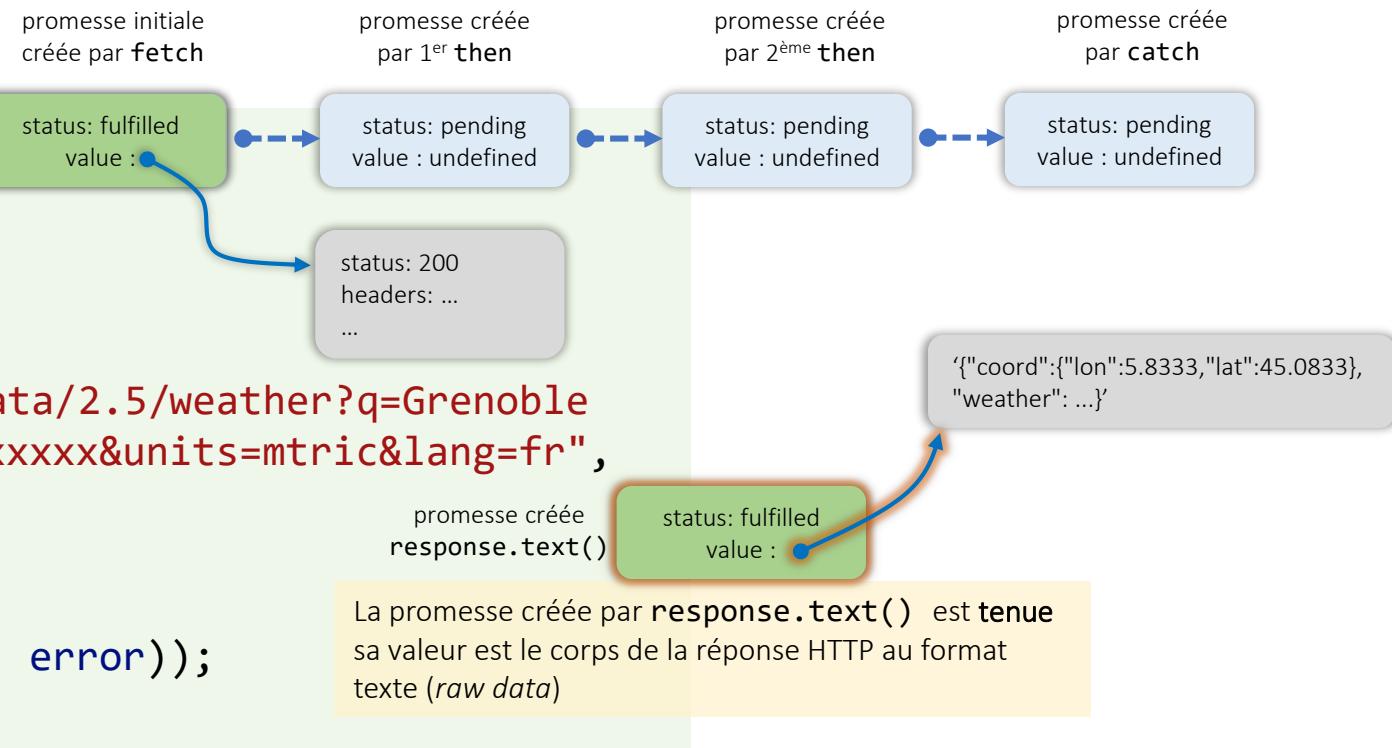
```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
8     .then(response => response.text())  
9     .then(result => console.log(result))  
10    .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```



# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
8   .then(response => response.text())  
9   .then(result => console.log(result))  
10  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

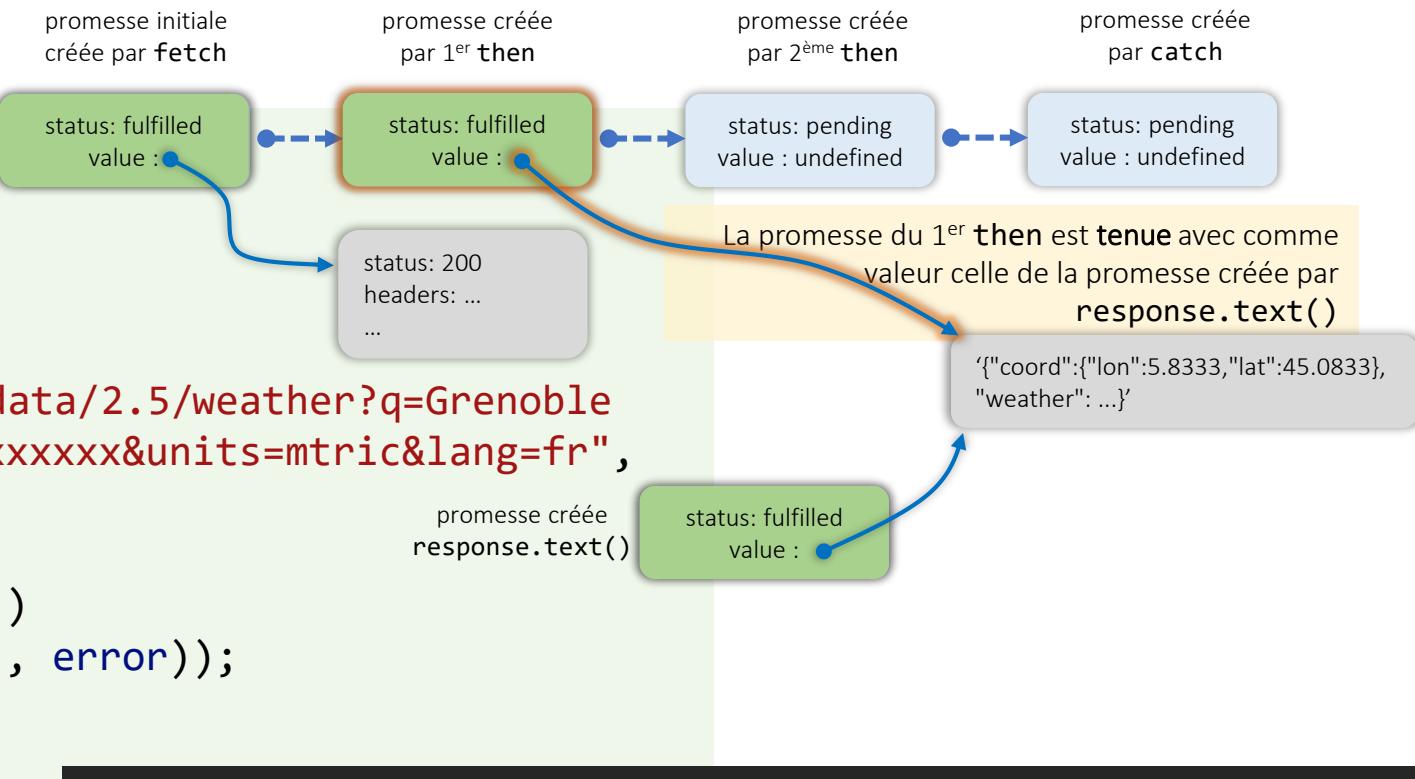


```
> node . testOpenWeather.js  
Terminé !
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
8   .then(response => response.text())  
9   .then(result => console.log(result))  
10  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

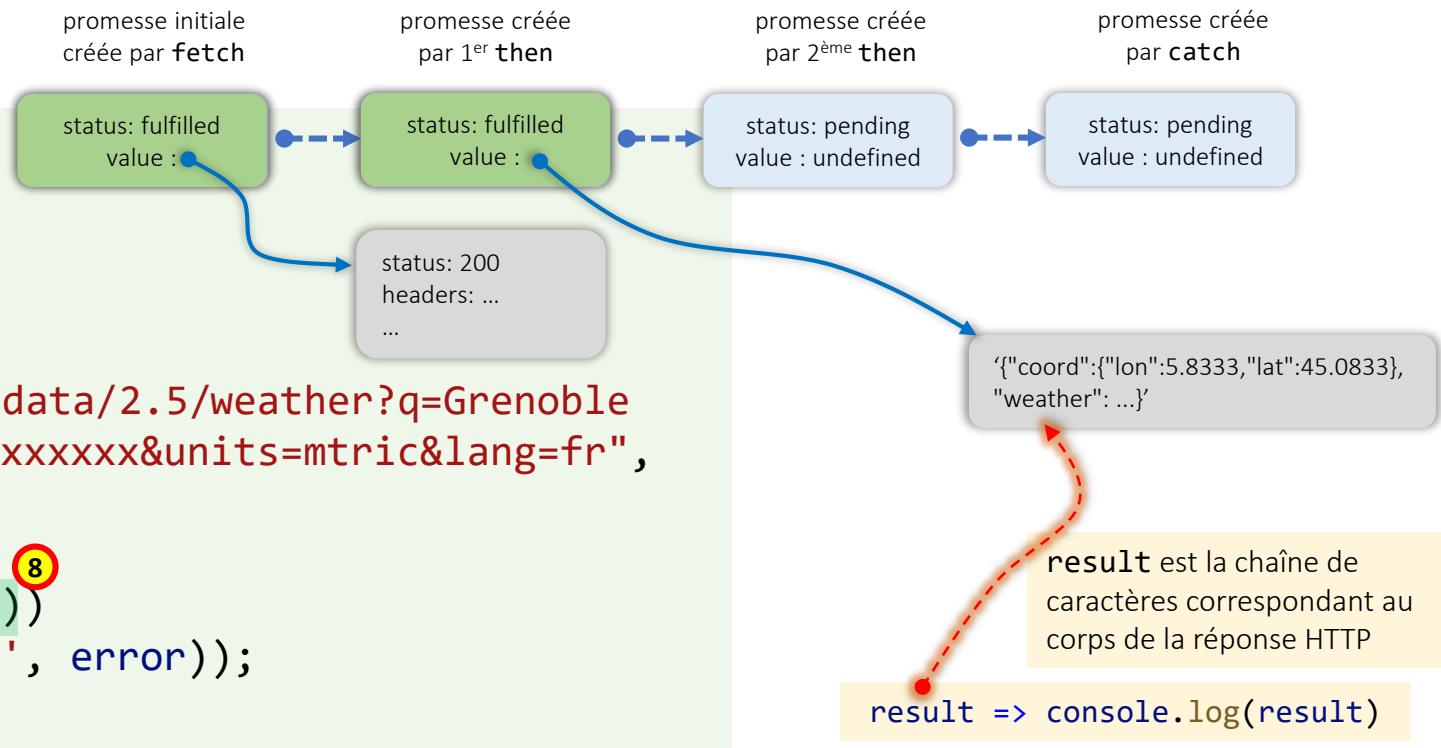


# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
  .then(response => response.text())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

8 la fonction callback du 2<sup>ème</sup> `then` est exécutée avec comme argument la valeur de la promesse précédente



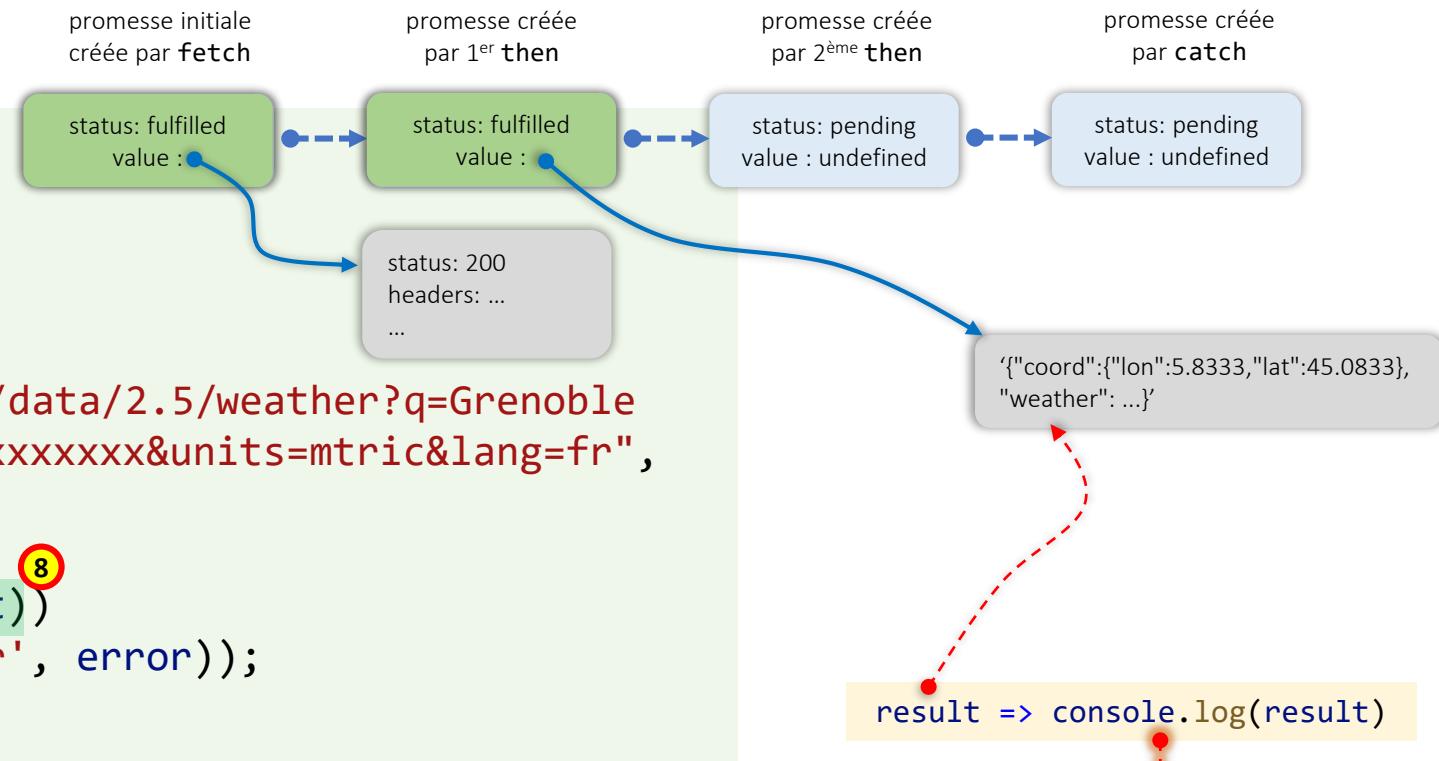
```
> node . testOpenWeather.js  
Terminé !
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
8   .then(response => response.text())  
9   .then(result => console.log(result))  
10  .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```

8 la fonction callback du 2<sup>ème</sup> then est exécutée avec comme argument la valeur de la promesse précédente

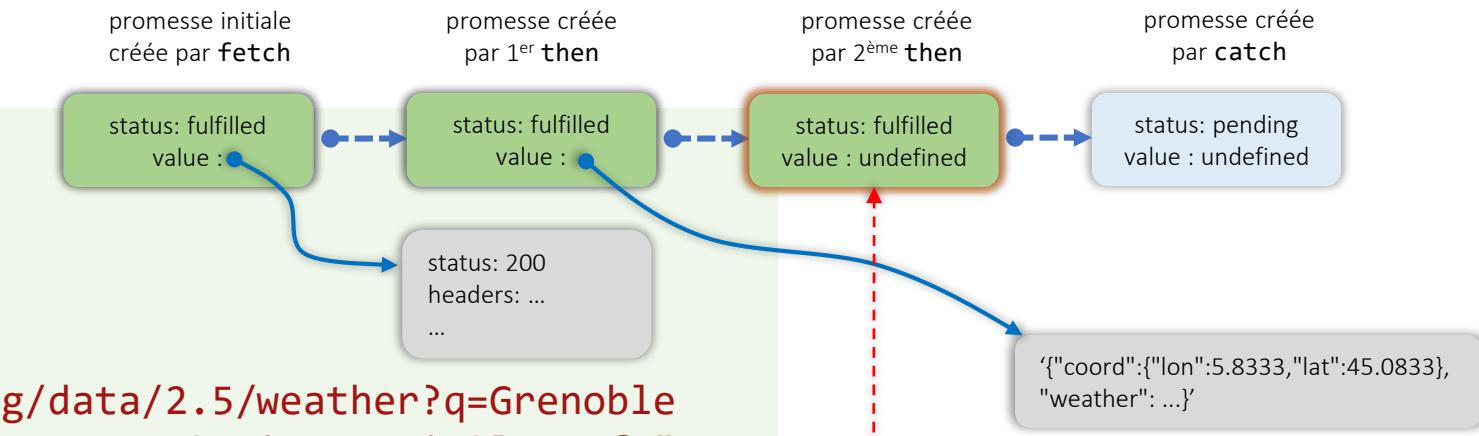


```
> node . testOpenWeather.js  
Terminé !  
{"coord": {"lon": 5.8333, "lat": 45.0833}, "weather": [{"id": 801, "main": "Clouds", "description": "peu nuageux", "icon": "02d"}], "base": "stations", "main": {"temp": 7.9, "feels_like": 7.9, "temp_min": 7.9, "temp_max": 7.9, "pressure": 1025, "humidity": 62, "sea_level": 1025, "grnd_level": 902}, "visibility": 10000, "wind": {"speed": 0.86, "deg": 358, "gust": 1.78}, "clouds": {"all": 14}, "dt": 1738679554, "sys": {"country": "FR", "sunrise": 1738651987, "sunset": 1738687683}, "timezone": 3600, "id": 3014727, "name": "Grenoble", "cod": 200}
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
8     .then(response => response.text())  
9     .then(result => console.log(result))  
10    .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```



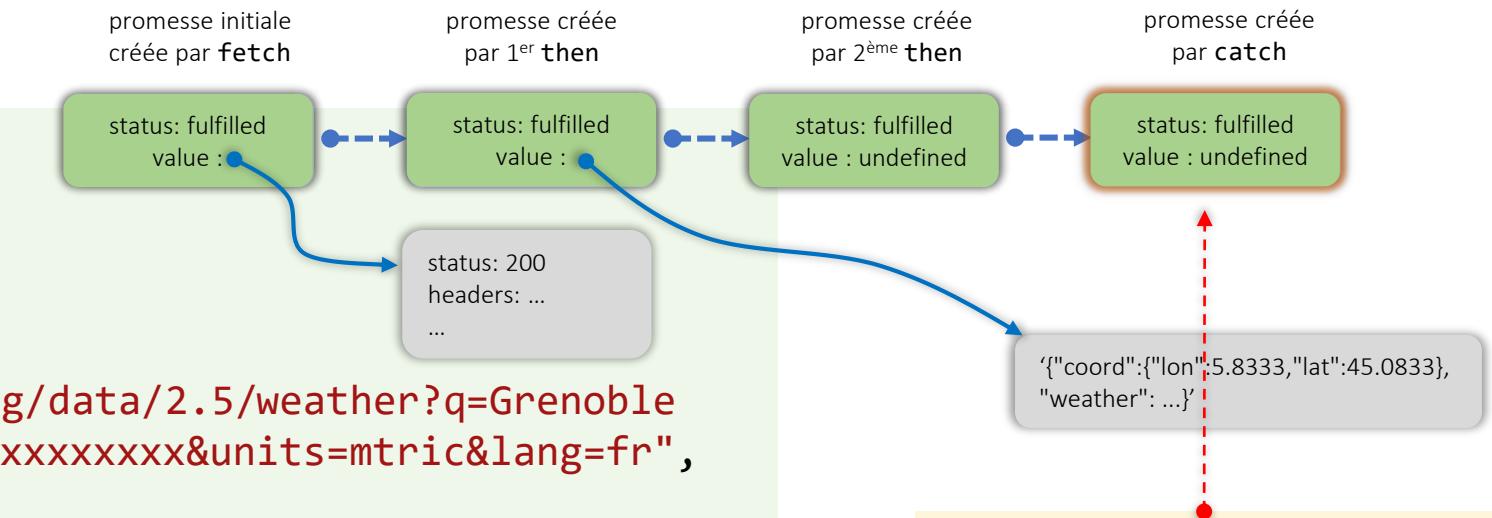
La promesse est résolue (**tenue**) sa valeur  
est la valeur de retour de la fonction  
`result => console.log(result)`

```
> node . testOpenWeather.js  
Terminé !  
{"coord": {"lon": 5.8333, "lat": 45.0833}, "weather": [{"id": 801, "main": "Clouds", "description": "peu  
nuageux", "icon": "02d"}], "base": "stations", "main": {"temp": 7.9, "feels_like": 7.9, "temp_min": 7.9, "temp_max": 7.9, "pressure": 1025, "humidity": 62, "sea_level": 1025, "grnd_level": 902}, "visibility": 10000, "wind": {"speed": 0.86, "deg": 358, "gust": 1.78}, "clouds": {"all": 14}, "dt": 1738679554, "sys": {"country": "FR", "sunrise": 1738651987, "sunset": 1738687683}, "timezone": 3600, "id": 3014727, "name": "Grenoble", "cod": 200}
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
8     .then(response => response.text())  
9     .then(result => console.log(result))  
10    .catch(error => console.log('error', error));  
11  
12 console.log('Terminé !');
```



La promesse est résolue. Comme il n'y a pas eu d'erreur dans les traitements mis en place par les promesses précédentes, elle est tenue et sa valeur est la valeur de la promesse qui la précède (`undefined`)

```
> node . testOpenWeather.js  
Terminé !  
{"coord": {"lon": 5.8333, "lat": 45.0833}, "weather": [{"id": 801, "main": "Clouds", "description": "peu nuageux", "icon": "02d"}], "base": "stations", "main": {"temp": 7.9, "feels_like": 7.9, "temp_min": 7.9, "temp_max": 7.9, "pressure": 1025, "humidity": 62, "sea_level": 1025, "grnd_level": 902}, "visibility": 10000, "wind": {"speed": 0.86, "deg": 358, "gust": 1.78}, "clouds": {"all": 14}, "dt": 1738679554, "sys": {"country": "FR", "sunrise": 1738651987, "sunset": 1738687683}, "timezone": 3600, "id": 3014727, "name": "Grenoble", "cod": 200}
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
requestOptions)  
  .then(response => response.text())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
10
```

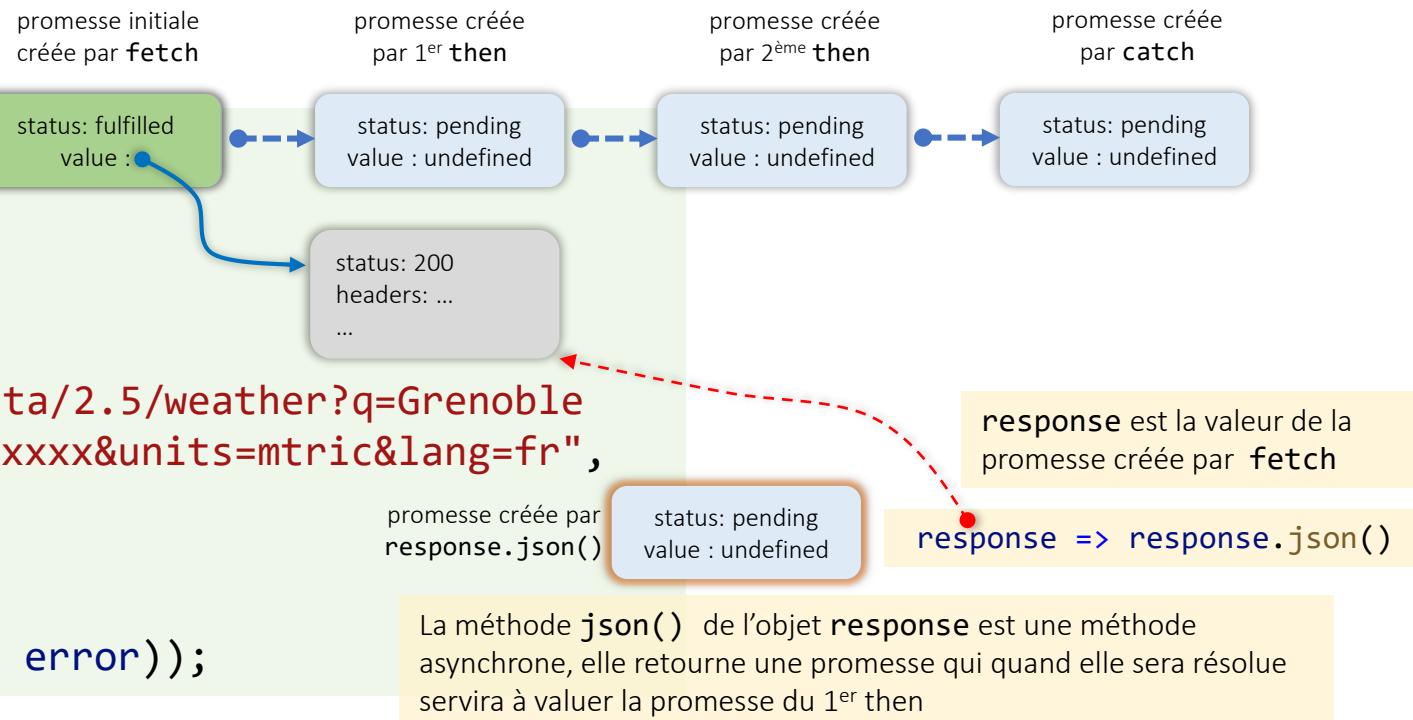
Comment faire pour afficher la description des conditions météo, la température et la date ?

```
> node .\testOpenWeather1.js  
Météo à Grenoble  
Conditions météo : ciel dégagé  
Température : -4.69 + °C  
date : 5 févr. 2025 à 01:05:19  
>
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 Fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
8   .then(response => response.json())  
9   .then(result => console.log(result))  
10  .catch(error => console.log('error', error));
```

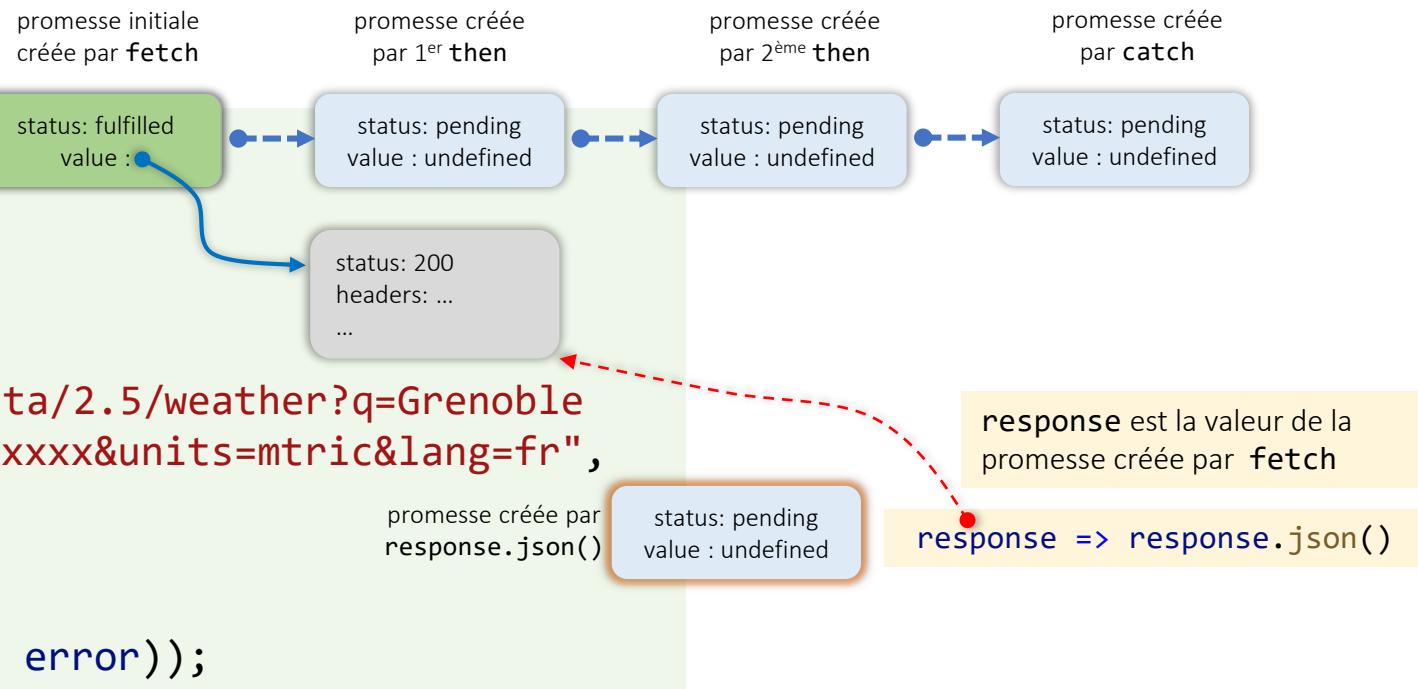


① Remplacer `response.text()` par `response.json()`

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 Fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7   requestOptions)  
8     .then(response => response.json())  
9     .then(result => console.log(result))  
10    .catch(error => console.log('error', error));
```



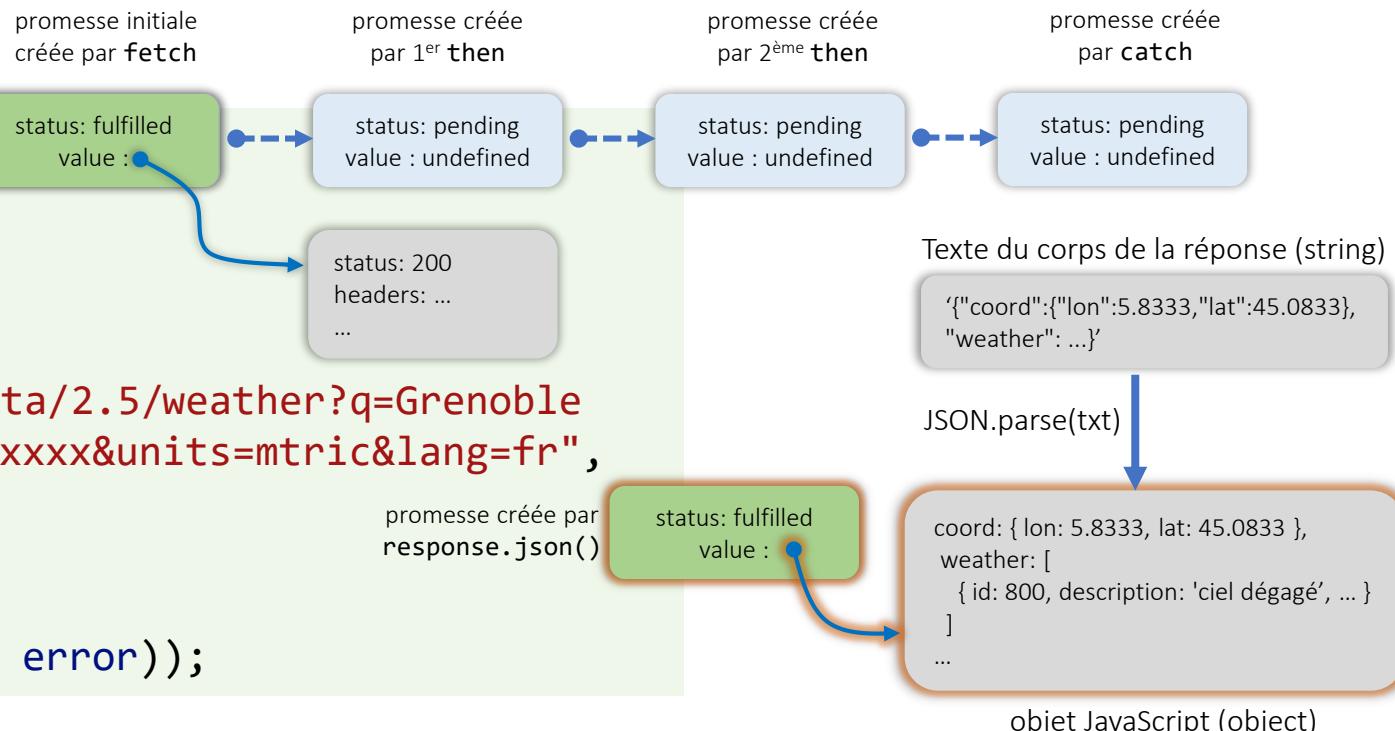
De manière identique à `text()`, la méthode `json()` de l'objet `response` est une méthode asynchrone qui quand elle s'exécute retourne une promesse

```
> node .\testOpenWeather1.js
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7 requestOptions)  
  .then(response => response.json())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
8  
9  
10
```



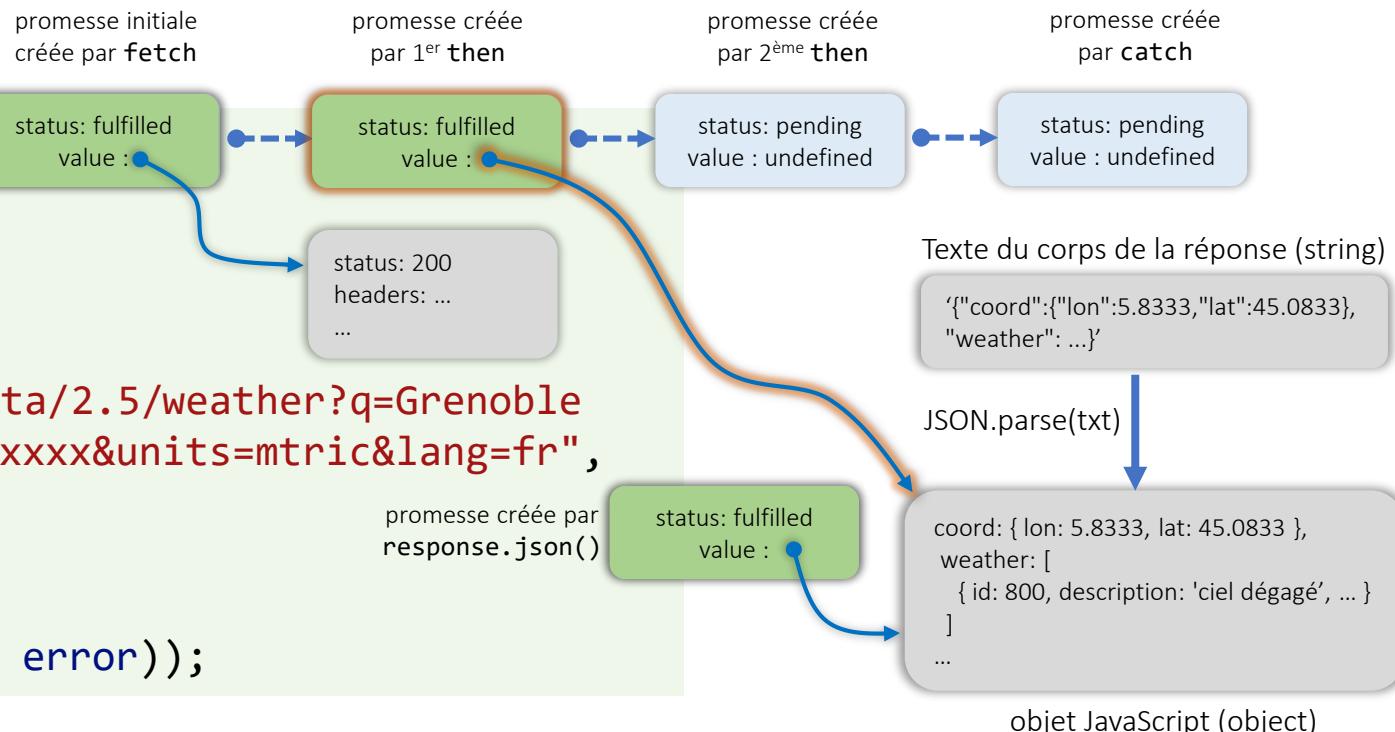
Quand le corps de la réponse HTPP est obtenu, son texte (du JSON) est transformé en un objet JavaScript et la promesse créée par `response.json()` est tenue avec cet objet comme valeur

```
> node .\testOpenWeather1.js
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7 requestOptions)  
  .then(response => response.json())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));  
8  
9  
10
```



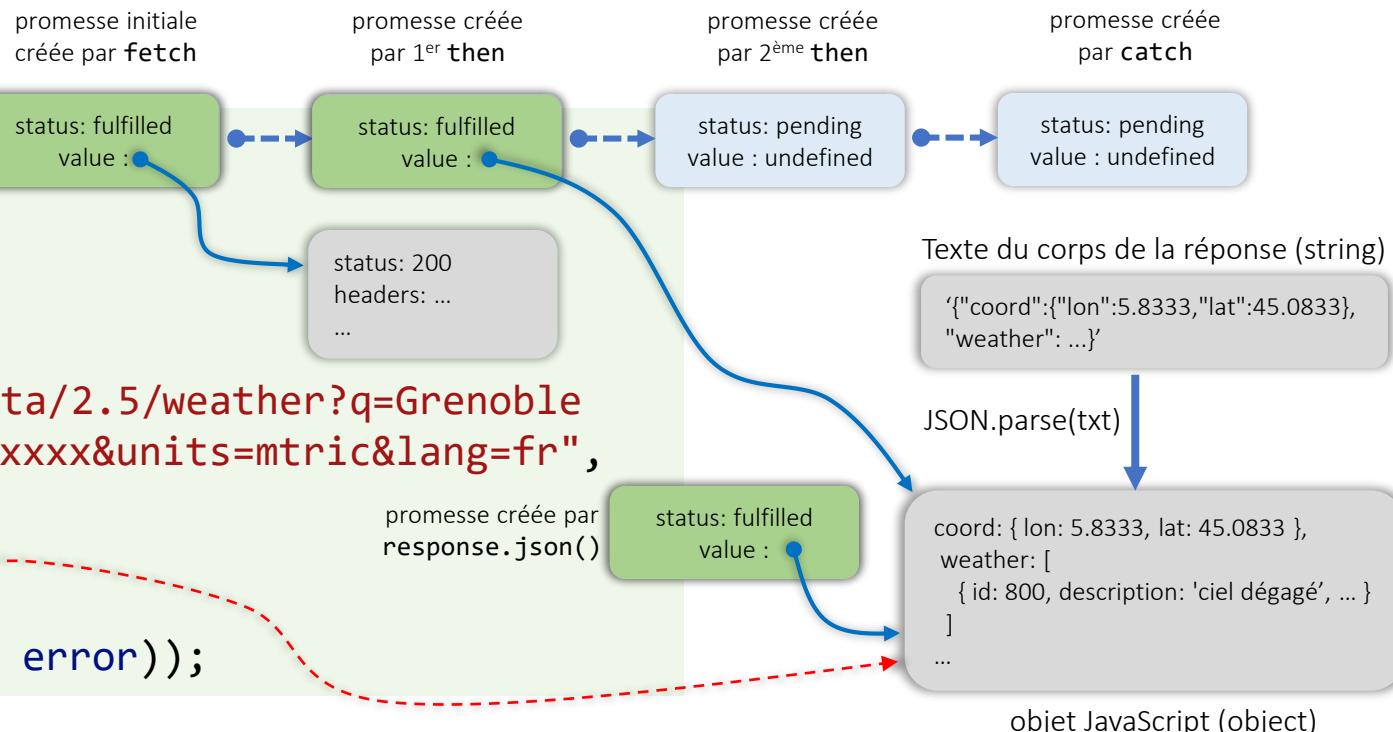
La valeur de cette promesse est transmise à la promesse créée par la méthode then qui devient elle aussi tenue avec comme résultat cet objet

```
> node .\testOpenWeather1.js
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?q=Grenoble  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx&units=metric&lang=fr",  
7 requestOptions)  
  .then(response => response.json())  
  .then(result => console.log(result))  
  .catch(error => console.log('error', error));
```



la fonction associée au 2<sup>ème</sup> then est exécutée avec l'objet JavaScript comme paramètre result.

Affichage de l'objet result

```
> node .\testOpenWeather1.js  
{  
  coord: { lon: 5.8333, lat: 45.0833 },  
  weather: [  
    { id: 800, main: 'Clear', description: 'ciel dégagé', icon: '01n' }  
  ],  
  ...  
}
```

# Question 2.3 : utiliser fetch, récupérer la réponse en JSON

Le code généré par Postman

```
1 var requestOptions = {  
2   method: 'GET',  
3   redirect: 'follow'  
4 };  
5  
6 fetch("https://api.openweathermap.org/data/2.5/weather?  
    &appid=87c1xxxxxxxxxxxxxxxxxxxxxx")  
    .then(response => response.json())  
    .then(result => console.log(result))  
    .catch(error => console.log('error', error))
```

Pour afficher la description des conditions météo, la température et la date, il suffit de remplacer l'instruction `console.log(result)` par un bloc de code accédant directement aux propriétés de l'objet `result`

```
status  
var  
{  
  promesse initiale  
  créée par fetch  
  promesse créée  
  par 1er then  
  promesse créée  
  par 2ème then  
  promesse créée  
  par catch  
  console.log("Météo à Grenoble");  
  console.log(`Conditions météo : ${result.weather[0].description}`);  
  console.log(`Température : ${result.main.temp} + °C`);  
  const timestamp = result.dt * 1000;  
  const date = new Date(timestamp);  
  const dateString = date.toLocaleDateString("fr-FR", {  
    day: "numeric",  
    month: "short",  
    year: "numeric",  
  });  
  const timeString = date.toLocaleTimeString(undefined, {  
    hour: "2-digit",  
    minute: "2-digit",  
    second: "2-digit",  
  });  
  console.log(`date : ${dateString} à ${timeString}`);  
}
```

```
> node .\testOpenWeather1.js  
Météo à Grenoble  
Conditions météo : ciel dégagé  
Température : -4.69 + °C  
date : 5 févr. 2025 à 01:05:19  
>
```

