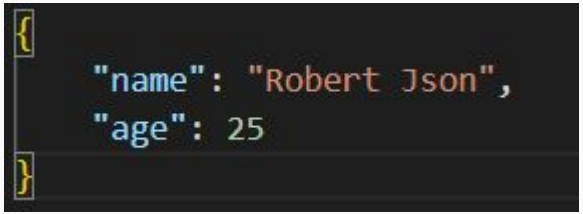
JSONってなに？？

JSONとは

- JSONは「JavaScript Object Notation」の略称
  - JavaScript：プログラミング言語の1つ
  - Object Notation：データを簡単に扱えるようにフォーマットとして構造化した記述方法
    - JavaScriptのオブジェクト表記法に由来するデータの記述方式



- データ通信が必要な場面で言語を問わず活用されている、共通のデータ記述形式
  - JavaScriptに限らず、各種プログラミング言語で利用できる
  - 特にREST APIなどで使われる

JSONのメリット

- JSONが登場する前まで主に使われていたXMLと比較してテキスト量が少ないため、
  高速な通信が求められる場合に有用
- JSONは項目と値のペアで記載するため視覚的に確認しやすく、
  階層が深くなっても構造を追いやすい

  - XMLの例

```xml
<?xml version="1.0" encoding="utf-8"?>
<data>
  <item>
    <id>1</id>
    <name>Edward</name>
  </item>
  <item>
    <id>2</id>
    <name>Bob</name>
  </item>
</data>
```

  - JSONの例

```json
[
    {"id": "1", "name": "Edward"},
    {"id": "2", "name": "Bob"}
]
```

JSONの型

- JSONで使用できる型は6つある
  - 文字列（”...”）
    - ダブルクォーテーションで囲んだ文字列を指定
  - 数値（123, 12.3など）
    - 123, 12.3などの数値を指定
  - Null値（null）
    - nullはすべて小文字で指定
  - Bool値
    - trueかfalseの2値のいずれかを指定
  - オブジェクト（{...}）
    - {...}で指定
    - 階層構造を持つことが可能
  - 配列（[...]）
    - 配列要素には、文字列、数値、Null値、真偽値、オブジェクト、配列すべてを使用可能

JSONの書き方①

- 基本的な形式：データのkeyとvalueを{ }の中にコロンで区切って記載
  ※keyは文字列、valueは様々なデータ型で記述可能
  ※文字列は必ずダブルクォーテーション(")で囲む→シングルクォーテーション(')は不可
  ※データが2つ以上ある場合は、カンマで区切る

```
o {"key1": "value1", "key2": "value2"}
x {'key1': "value1", 'key2': "value2"}
x {key1: "value1", key2: "value2"}
```

- 読みやすさを重視し、データを改行したりインデントを使って見た目を変えても問題なし

```
{
    "key1": "value1",
    "key2": "value2"
}
```

JSONの書き方②

- 配列や値のみの表記もJSONに従ったデータとして認められる
  ※文字列は必ずダブルクォーテーションで囲う

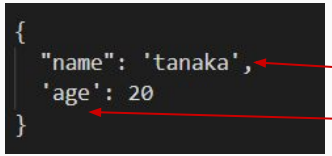
```
["ABC", "DEF"]
```

```
"ABC"
```

```
123
```

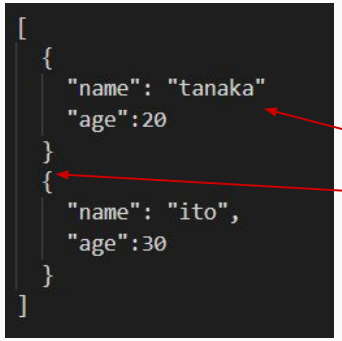
JSONでよくある記述ミス

- シングルクォーテーションを使用している



2行目：valueである「tanaka」に「’」を使用しているので構文エラー
3行目：keyである「age」に「’」を使用しているので構文エラー

- 適切でないカンマ



3行目：複数のオブジェクトがあるにも関わらず「,」で区切られていない
6行目：配列として複数の要素があるにも関わらず「,」で区切られていない

JSONの活用例

- フロント側（例：JavaScript）とバックエンド（例：PHP (Laravel) ）でのデータのやり取り

フロント（JavaScript）での JSONの取り扱い方

- fetchメソッドなどを用いてAPIを叩くことで、JSONデータを取得する
  - 非同期のネットワーク通信を簡単にわかりやすく記述できるメソッド
  - フロント側のJavaScriptから、HTTPリクエストを送信し、
    指定した URL からリソース（データ）を取得することができるようになる

```
const id = {
    id: 10000
};
const url = // 任意のアドレス

fetch(url,
    {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(id)
    })
    .then(response => response.json())
    .then(data => {
        console.log(data);
    });
```

出力結果
（11ページ目に記載の処理がバックエンド側で実行され、
フロント側で受け取った後の出力結果）

```
// オブジェクト
{ id: "001", name: "AAA", age: 20 }
```

バックエンド（PHP(Laravel)）でのJSONの取り扱い方①

- フロント側からのリクエストにてJSONデータを受け取る場合
  - リクエストのContent-Typeヘッダが適切にapplication/jsonへ設定されている限り、inputメソッドを介してJSONデータにアクセスできる

```php
public function example(Request $request)
{
    $id = $request->input('id');
    // $idを用いた処理が続く
```

リクエストボディ：JSONデータ

```json
{
  "id": 10000
}
```

バックエンド（PHP(Laravel)）でのJSONの取り扱い方②

- レスポンスとしてフロント側にJSONデータを送る場合
  - コントローラーから配列を返すことで、Laravelが自動的にJSONレスポンスへ変換する

```php
public function example(Request $request)
{
    $id = $request->input('id');
    // $idを用いた処理が続く

    $data = [
        'id'   => '001',
        'name' => 'AAA',
        'age'  => 20
    ];

    // クライアント側にデータを返す
    return $data;
}
```

RDBMS（MySQL）での JSONの取り扱い方

- JSON型はMySQL5.7以上で利用できる
- JSONデータを扱うための便利な関数が用意されている（下記一部）
  - JSONデータ生成（JSONオブジェクトを生成）→JSON_OBJECT

```
mysql> SELECT JSON_OBJECT('name', 'tanaka', 'gender', 1);
+-----------------------------------------+
| JSON_OBJECT('name', 'tanaka', 'gender', 1) |
+-----------------------------------------+
| {"name": "tanaka", "gender": 1}          |
+-----------------------------------------+
```

  - JSONデータ検索（指定パスのデータ抽出）→JSON_EXTRACT

```
mysql> SELECT `col`, JSON_EXTRACT(`col`, '$.name') FROM `json_users`;
+-----------------------------------------+-----------------------------+
| col                                     | JSON_EXTRACT(`col`, '$.name') |
+-----------------------------------------+-----------------------------+
| {"name": "tanaka", "gender": 1}         | "tanaka"                    |
+-----------------------------------------+-----------------------------+
```

まとめ

- JSONとはデータ通信が必要な場面で言語を問わず活用されている、
  共通のデータ記述形式

- JSONを用いることで、フロント側とバックエンド側での通信時に
  情報を簡単にやり取りすることができる

- RDBMSでもバージョンによってはJSON型を利用でき、
  JSONデータを扱うための便利な関数が用意されている