

# Drupal 初心者の練習帳 - 1

## ～食堂兼仕出し屋の食材購入計算入門～

2021.6.15 蛙野 卦呂

### 1. はじめに

“Drupal”（ドゥルーパル）は、インターネット上のホームページやブログを簡単に作るツールのひとつです。

本書は、Drupal による初歩的なアプリ開発事例の紹介です。

具体的には、「X 人前の Y 料理を作るための食材購入計算アプリ」を開発します。

本書では Drupal の基本的な使い方の一部しか扱っていません。

本書の中には、Drupal だけでなく HTML や JavaScript や CSS といったツール（言語等）も登場しますが、出てきた都度、入門程度で結構ですのでネットか何かで調べて勉強してください。

本書は、これまでの日本語版 Drupal 入門解説書には出てこない（2021 年 5 月 28 日現在）ことも非常に実践的に書いていますので、途中で Drupal に挫折した人にも参考になると思います。

なお、このアプリはスマホではなく PC（MacBook Air early 2015）で開発しますが、Windows PC でも動くと思います。

### 2. Drupal のインストール

まず、Drupal の最新バージョン “Drupal 9” か、一つ前のバージョンの “Drupal 8” が使えるようにします。

お手持ちのブラウザで「Drupal インストール」と検索すると、無数の検索結果が表示されます。

お手持ちの PC（Mac とか Windows）に適合し、具体的で分かりやすい解説を見つけたら、それに従ってインストールしてください。

また、Drupal をインストールする場所は、「ローカルホスト」（自分の PC）と「サーバ」（海外の Drupal 支援会社）の 2 種類があり、解説によって異なります。サーバに Drupal をインストールしてアプリを開発すると、自分の PC だけでなく、自分のスマホや他者の PC・スマホでも、そのアプリを動かすことができるようになります。

自分の PC 上にインストールしたら、その PC だけで自分の Drupal サイトを扱うことになります。

web 検索してみると、Pantheon（パンテオン）とか Aquia（アクイア）などの海外サービス会社版や、Studio-umi や ANNAI といった国内の会社もインストールの手引をしています。その他にも多様なインストール解説があるので、気に入ったやり方で実際にチャレンジしてみてください。

本書ではこれから頻繁に、画面上のメニュータグ、フレーズ、ボタンなどをクリック

する動作の説明が出てきます。それらを、「[なにになに] →」と略記することにしします。これは、「なにになに」と書いてあるフレーズやボタンをクリックすると「→」で次の画面に移る、という意味です。

なお、「→」で少々時間がかかる場合がありますが、我慢して待ってください。

本書ではサーバタイプの「Pantheon」で Drupal 9 を導入しましたが、以下の特殊な操作が必要でした。「Drupal のインストール完了」後、[環境設定] → 「開発」の [メンテナンスモード] → 「☐サイトをメンテナンスモードにする」に✓し [構成を保存] → [機能拡張] → [アンインストール] → 「☐Update Manager」に✓し [アンインストール] → [オンラインにする] → 「✓ サイトをメンテナンスモードにする」の✓を外し [構成を保存] → [機能拡張] → 「☐Update Manager」に✓し [インストール]。

これで、新しいテーマやモジュールがインストールできるようになります。

いずれにせよ、インストールが成功したら、図 1 のような画面が出ます。

「✓ おめでとうございます。Drupal のインストールが完了しました。」と表示されます。

なお、本アプリのサイト名は、インストール手続きの中で、「etude-1」としました。

【図 1：初期画面】



### 3. アプリ作成計画

アプリ作成の手順を概説します。

#### (1) テーマの変更

Drupal には基本的なデザインの「ヘッダー」（最上部の青色の帯、ロゴマーク、メニュータグなどで構成）や「フッター」（謝辞やリンク先を表示）があり、それらを「テーマ」と呼んでいます。初期画面では「Baltic 8.9.13」（「バルチック」）というテーマが設定されていますが、これを自分の好みに応じて変更できます。

本書では、「Bootstrap（ブートストラップ）」という一般的なテーマを使います。

## (2) フロントページの作成

図 1 の初期画面の「etude-1 へようこそ」表示の下には、「フロントページのコンテンツはまだ作成されていません。」という文が小さく付記されています。そこでまずフロントページ (= 正式な初期画面) を作成します。

## (3) 仕入先ノート作成

いつも食材を購入している馴染みの仕入先に識別記号を付け、店名、TEL、FAX、担当者を登録します。

## (4) 食材ノート作成

食材に個別の識別記号を付け、仕入先及び単価等を登録します。

## (5) 料理メニュー&レシピ作成

料理に個別の識別記号を付け、料理名、イメージ、価格、使われる食材、及びレシピを登録します。

## (6) 注文ノート作成

注文があったら、その内容を記録し、調理や配膳の進捗を店主や従業員に分かりやすく表示します。

## (7) 食材量計算式作成

仕出しの予約があった場合、無駄なく調理するためには、「勘と経験」ではなく、レシピに基づいて必要な食材とその数量を計算し、それらをどの仕入先に発注すればよいか計算する必要があります。

## (8) 食材購入発注書作成

これが本書の目的です。

(7) で計算した結果をプリントアウトし、仕入先に FAX すれば発注できるような「発注書」画面を作ります。

本書は、実際に PC に向かって手を動かし、試行錯誤しながら読み進めることをオススメします。

Drupal 初心者の私には本書のレベルに達するまで約 2 年間かかりましたが、まだ Drupal のごく一部しか分かっていません。本書では、これまでうまく行ったことのみを集大成していますので、初心者でも 1 週間足らずで「食材購入発注書」が完成するでしょう。その成功体験をバネにして、Drupal を更に勉強されても良いと思います。

## 4. アプリ作成

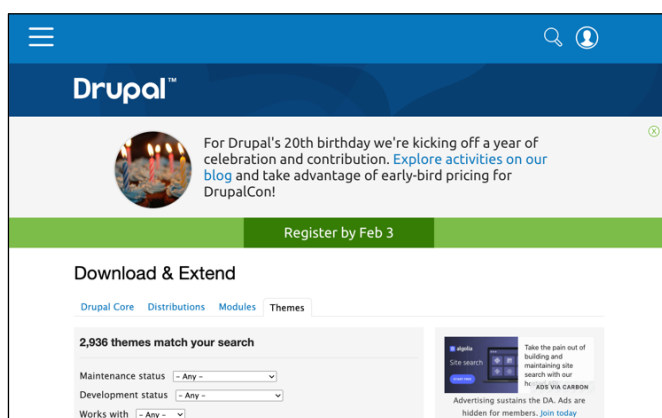
### (1) テーマの変更

”Bootstrap 3”（「ブートストラップ・スリー」）というテーマに変更します。

サイト管理ツールバー（図1のヘッダの上段に表示されている「コンテンツ、サイト構築、テーマ、機能拡張、環境設定、ユーザー、レポート、ヘルプ」）の「テーマ」をクリックすると「+新しいテーマをインストール」という青ボタンが現れます。

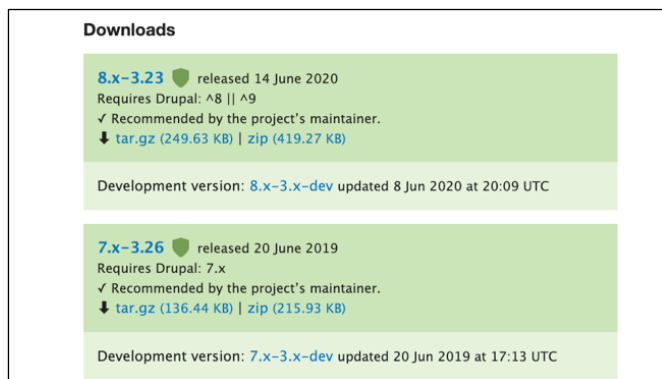
「あなたのウェブサイトのデフォルトテーマを設定してください。別のテーマが利用可能です。」の「テーマ」という文字をクリックすると以下の画面（図2）が出てきます。英語ですが恐れないでください。

【図2： Drupal のテーマ選択画面】



スクロールして「Bootstrap」を選びます。Bootstrapに関する同様な画面に変わり、下の方に「Downloads」という項があります（図3）。

【図3： Bootstrap のダウンロード】



最新版 ”8.x-3.23 released 14 June 2020” の、「tar.gz(249.63KB)」を右クリックし、「リンク(のアドレス)をコピー」を左クリックします。

ブラウザの「←」（または「<」）で Drupal の初期画面に戻って、サイト管理ツールバーの「テーマ」を選び、「+新しいテーマをインストール」→「次の URL からイン

ストールする」欄に先程のリンクのアドレスをペーストし、最下段の「インストール」をクリック。

インストールが無事終了したら、「更新マネージャー」という画面が出てきます。「次のステップ」の項で、「新しく追加されたテーマをインストールする」をクリックします。

「テーマ」画面に戻ります。

「アンインストール済みテーマ」というところに、先程の「Bootstrap 3」が入っています。これを「インストールしてデフォルトに設定」します。すると、これが「インストール済のテーマ」の先頭に移動します。テーマ図の右の「設定」をクリックします。

「Bootstrap」という画面が出ます。

最下段の「構成を保存」ボタンを押します。

上の方に、「**✓設定オプションが保存されました。**」と表示されます。

最上位メニューの「サイトへ戻る」をクリックすると、Baltic に比べて割とあっさりした画面が表示されます。これが「Bootstrap 3」の初期画面です（図4）。

【図4：Bootstrap テーマの初期画面】



黒キャベツのようなロゴマークは Bootstrap3 for Drupal のシンボルマークです。

これを自分用のロゴマークに変更してみましょう。

ツールバーの「テーマ」を選び、「インストール済みテーマ」である「Bootstrap 8.x-3.23」の「設定」をクリックします。

「Bootstrap」の設定画面が出てきます。

下の方の「ロゴ画像」をクリックします。

「テーマが提供するロゴを使用」のチェックを外すと、「ロゴ画像のアップロード」という欄が出てきますので、「ファイルを選択」で、ご自分のロゴ・イメージを入れる

ことができます。jpeg や png 形式のファイルが使えます。ファイルサイズでロゴの大きさが変わります。jpeg 形式なら 1 ～2KB 程度のサイズで良いかと思います。

最下段の「構成を保存」→（しばらくたって）→「サイトへ戻る」をクリック。

次に「サイト名」を変更します。

ツールバーの「環境設定」→システム欄の「サイトの基本設定」をクリック。

「サイト名」欄に「etude-1」と書かれているのを、適当な屋号（例えば「丸丸食堂」）に変更し、最下段の「構成を保存」をクリックし、「サイトへ戻る」ります。

今のままでしたらヘッダーの背景色が白で、その下の本文にあたる領域との区別がつきにくいので、ヘッダーの背景色を変更することにします。

### (1)-1 テーマの背景色の変更 (CSS を使う)

Drupal に「Asset Injector」というモジュールを追加し、Asset Injector 内の「CSS Injector」というツールで、CSS を記述します。「CSS」とは「Cascading Style Sheet」（ウェブページのスタイルを指定するための言語）です。ネットで少し勉強しておいてください。

#### (1)-1-1 モジュール「Asset Injector」の導入

ブラウザで新しいタブを追加し、「Drupal asset injector module download」という語句で検索すると、検索結果の中に「Asset Injector | Drupal.org Nov 24, 2015 -- ...」というサイトがあると思います。

それをクリックし、現れた画面をずっと下までスクロールすると、「Downloads」欄の黄緑色の四角の中に↓に続き、「tar.gz(24.67KB)」という青色の記載があります。これを右クリックして「リンクのアドレスをコピー」（または「リンクをコピー」）します。

Drupal の管理画面に戻り、サイト管理ツールバーの「機能拡張」→「+新しいモジュールをインストール」をクリック。

「新しいモジュールをインストール」画面の「次の URL からインストールする」欄に先程のアドレスを貼り付け、「インストール」をクリック。

「✓ インストールは正常に終了しました。」という表示が出たら、「次のステップ」項で「[新しく追加されたモジュールを有効にします](#)」をクリックします。「拡張」画面に戻るので、一番下の方に加わった「☐ Asset Injector」にチェックを入れ、最下段の「インストール」をクリック。

インストールが完了したらサイト管理ツールバーの「環境設定」をクリックしてみましょう。「開発」欄に「Asset Injector」という項目が追加されています。それをクリックすると、「CSS インジェクター」と「JS Injector」の2つの機能が使えることが分かります。「JS」とは「JavaScript」のことです。

#### (1)-1-2 「Asset Injector」内の「CSS Injector」を使う

「CSS インジェクター」→「+ Add Css Injector」をクリック。

「ラベル」欄に、例えば「myheader」と入力。

「コード」欄には、以下のコード（図5）を記述。

なお、「/\* … \*/」はコメントなので、書かなくても良いです。

【図 5：ヘッダーを CSS で装飾する。(myheader.css の作成)】

```
#navbar { /* ヘッダー(“navigation bar”)について */
  background: #b1f9D0; /* 背景色を #b1f9D0(薄緑)に */
}
.navbar-brand { /* ヘッダーのブランド名(丸丸食堂)について */
  font-size: 230%; /* フォントサイズを 2.3 倍に */
  font-style: italic; /* フォントをイタリックに */
}
.menu--main li { /* メニュータグについて */
  margin: 3px; /* 外枠の太さを 3px に */
  border: 1px solid #000000; /* 輪郭線の太さを 1px、実線、色は黒で */
  background: #ffffff; /* 背景色を #ffffff(薄ベージュ)に */
}
```

そして [保存]。

[サイトへ戻る] と、ヘッダーは図 6 のようになっています。

【図 6：ヘッダー変更結果】



## (2) フロントページの作成

しかし、図 6 の初期画面にはまだ「フロントページのコンテンツはまだ作成されていません。」と表示されています。

そこで、フロントページ（初期画面）を作成します。

ツールバー [コンテンツ] → [+コンテンツを追加] → [基本ページ] → 「基本ページの作成」画面が表示されます。

「タイトル」欄は、初期画面で「・・・へようこそ」と表示される語句の変更になります。例えば「山珍海味！ 美酒佳肴！ 暗中模索！ 加持祈祷！」と入れます。

「本文」欄は、語句や文章を書き込むことも、絵や写真のイメージファイルを貼り付けることもできます。

イメージファイルは「ソース」と書いてある行の「山と太陽」のアイコンをクリックすると、「ファイル選択」ボタンが表示されますので、PCの中から選んで入れます。イメージのレイアウトは左・右・中央等選択できますので、試してみてください。

画面右の「▶ URL エイリアス」をクリックすると、「URL エイリアス」欄が表示されます。ここに「/node」と入力し、「保存」。

これで初期画面（フロントページ）ができましたが、画面右の「検索欄」や「Tools」は、今回は不要ですので消します。

### (2)-1 画面右の「検索欄」や「Tools」を消去する

[サイト構築] → [ブロックレイアウト]。

ブロック「Secondary」の中の「Search」と「Tools」について、行の右にある[設定 ▼]の[▼]をクリックして[無効]を選択。→ [ブロックの保存] → [サイトへ戻る]。これで、初期画面が完成しました（図7）。

【図7：初期画面（例）の完成版】



### (3) 仕入先ノートの作成

これから「ノート（台帳）」をいくつか作ります。

Drupal としては、いずれも基本的には、「コンテンツタイプ」で入力項目の書式を決め、「コンテンツの追加」で個別データを入力し、「ビュー」で一覧表示します。

本書の目標は「X 人前の Y 料理を作るための食材購入計算アプリ」でした。



想像してみてください。

あなたは「丸丸食堂」のシェフです。

ある宴会場から、「○月×日□時まで、これこれの料理を 20 人前持って来てほしい。」との注文を受けたとします。

さあ、食材を仕入れる必要があります。「何をどこから、どれだけ買うか？」という問題です。

いつもの仕入先のリストを予めノートに作表しておきます。これを「仕入先ノート」と呼ぶことにします。

必要項目は、「仕入先名」、「TEL」、「FAX」、「担当者」くらいでしょうか？

これらの先頭に適当な識別子（「店 ID」）を付けて表示することにします。

「書式」は表 1 のヘッダー部分に、「個別データ」（例示）は 2 行目以降に示します。

【表 1：仕入先ノート】

店 ID (識別番号)	仕入先名 (単純文字列)	TEL (単純文字列)	FAX (単純文字列)	担当者 (単純文字列)
s01	八百屋お七	03-3502-8111	03-3502-8112	吉三郎
s02	たこ八鮮魚店	04-4613-9000	04-4613-9001	鮫島
s03	山ネコ精肉店	05-6789-0123	05-6789-0124	猪熊

### (3)-1 仕入先ノートの「コンテンツタイプ」の作成

まず、「コンテンツタイプ」（書式設定）を作ります。

ツールバー [サイト構築] → [コンテンツタイプ] → [+ コンテンツタイプの追加]。

「名前」欄に「仕入先」と入力すると、右の方に「システム内部名称: shiruxian [\[編集\]](#)」と表示されます。「shiruxian」では理解不能なので、[\[編集\]](#)をクリックして「システム内部名称」欄の名前を「supplier」に変更します。

また、「タイトルフィールドのラベル」欄に「店 ID」と入力しておきます。

[保存してフィールドを管理] で「フィールドの管理」画面になります。

★ [+ フィールドの追加] → 「新しいフィールドの追加」欄で「テキスト (プレーン)」を選択。(★は繰り返し起点の印です。)

「ラベル」欄に「仕入先名」と入力すると、右の方に「システム内部名称: field\_shiruxian ming [\[編集\]](#)」と表示されるので、[\[編集\]](#)をクリックして「システム内部名称」欄の名前を「field\_supplier\_name」に変更します。アンダーバーもちゃんと入力してください。

→ [保存して次へ] → 「フィールドの設定」画面ですが何もせず → [フィールド設定を保存] → [設定の保存]。

同様にして (上記★以降ご参照)、「TEL」、「FAX」、「担当者」も入力します。それぞれの「システム内部名称」は、「field\_supplier\_tel」、「field\_supplier\_fax」、及び「field\_supplier\_person」としてください。

以上で「仕入先ノート」のコンテンツタイプの作成は終わりです。

### (3)-2 仕入先ノートの「個別データ」の入力

ツールバー [コンテンツ] → [+ コンテンツを追加] → [仕入先] を選択すると、「仕入先の作成」画面になります。

表 1 の例示のようにデータを入力します。

例えば「店 ID」欄には「s01」、「仕入先名」欄には「八百屋お七」、「TEL」欄には「03-3502-8111」、「FAX」欄には「03-3502-8112」、「担当者」欄には「吉三郎」と入れ、→ [保存] → 先程入力した個別データが表示されます。

同様に、s02 や s03 の個別データも入力しておきましょう。

### (3)-3 仕入先ノートの「ビュー」表示

コンテンツタイプ「仕入先」に従って仕入先の情報を入力したので、今度はそれらを一覧表示させてみましょう。

「ビュー」という機能を使います。

ツールバー [サイト構築] → [ビュー] → [+ビューを追加] → 「ビューを追加」画面。

「ビューの名前」欄に、「仕入先ノート」と入力すると、システム内部名称が表示されます。[編集] で理解できる語句（英字）に変更します。例えば「supplier\_note」。

「ビューの設定」欄は、表示は「コンテンツ」のまま、タイプ指定は「仕入先」を、並び順は「タイトル (=ID)」を選びます。

「ページの設定」欄は、「☐ ページを作成する」に ✓ を入れます。

「Page title」欄は、「仕入先ノート」のまま。

「パス」欄は、ビューの名前と同じく「supplier\_note」としておきます。

「ページの表示設定」欄では、ディスプレイフォーマットを「テーブル」にします。

「表示件数」欄は空欄にし、「☒ ページャーを使用する」のチェックをはずします。

「☐ メニューリンクを作成」に ✓ を入れます。

「メニュー」欄も「リンクテキスト」欄もそのまま、[保存して編集] → ビューの編集画面に移ります。

スクロールして下の方を見ると、「プレビュー」で出来栄が確認できます。

現在は「タイトル」のカラムに、「s01」、「s02」、「s03」と縦に表示されているのみです。

これらは「店 ID」なので、「タイトル」を「店 ID」に変えます。

上の「ディスプレイ」欄の「フィールド」項に、青字で「[コンテンツ：タイトル \(タイトル\)](#)」と書いてあります。このカッコ内の「タイトル」をクリックします。

小画面が出てきて、「ラベル」欄に「タイトル」と書いてあるので、これを「店 ID」に変更し、[適用] します。ビューの編集画面に戻りプレビューを見ると、項目名が「店 ID」になっています。

コンテンツタイプ「supplier」に入力したその他の情報も表示させましょう。

先程の「フィールド」項の [追加] をクリックすると、「フィールドを追加」の小画面が現れます。

「Search」欄に「supplier」と入力すると、「supplier」に関係した行だけが表示されます。

タイトルを見ると、「本文」以外は全て入力していますので、これらの項目の左側の ☐ にチェックを入れ、[フィールドを追加して設定] をクリックします。

各フィールドの設定画面が現れますが、すべて何もしないで「適用して続ける」をクリックします。最後は「適用」ボタンになります。→ 編集画面に戻ります。

プレビューを見ると、確かに仕入先について入力したデータがすべて表示されています。ただし、並び順は変えた方が良さそうです。

「フィールド」項の「追加 | ▼」の「▼」→「並べ替え」で、小画面「フィールドの並べ替え」が現れます。

各行の左側に「十字矢印」が表示されています。これらをドラッグすることにより、並べ替えができます。並び順を、「店 ID」、「仕入先名」、「TEL」、「FAX」、「担当者」の順にし、「適用」で編集画面に戻ります。

プレビューを見ると、望み通りの一覧表ができています。

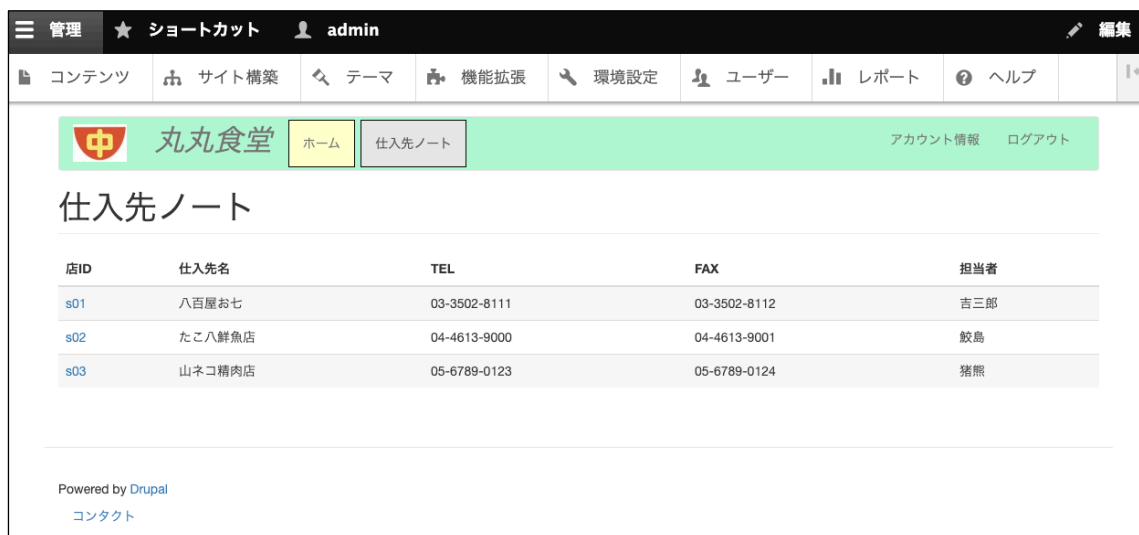
「保存」します。

「サイトへ戻」ってみましょう。

ヘッダーに「仕入先ノート」というメニュータグができています。

これをクリックすると、先程のプレビューで見たとおりの仕入先一覧表が表示されます(図8)。

【図8：「仕入先ノート」の画面】



店ID	仕入先名	TEL	FAX	担当者
s01	八百屋お七	03-3502-8111	03-3502-8112	吉三郎
s02	たこ八鮮魚店	04-4613-9000	04-4613-9001	鮫島
s03	山ネコ精肉店	05-6789-0123	05-6789-0124	猪熊

#### (4) 食材ノートの作成

次に「食材ノート」を作ります。

要領は「仕入先ノート」とほぼ同じですが、新しいことも含みます。

必要項目は、「食材名」、「食材 100g の単価 (円) (「百 g 単価」とします)」、「仕入先」とします。

これらの先頭に適当な識別子(「食材 ID」)を付けて表示することにします。

なお、百 g 単価は当面変動のない一定価格とします。

「書式」は表2のヘッダー部分に、「個別データ」(例示)は2行目以降に示します。

【表 2：食材ノート】

食材 ID (タイトル)	食材名 (プレーンテキスト)	百 g 単価 (数値(整数型))	仕入先 (エンティティ参照)
f001	玉葱	36	s01
f002	馬鈴薯	42	s01
f003	人参	48	s01
f004	イカゲソ	128	s02
f005	タコ足	256	s02
f006	カンパチ	512	s02
f007	鶏モモ	77	s03
f008	豚バラ	222	s03
f009	牛ロース	444	s03

表 2 で「仕入先」のデータとして「店 ID」を入れています、「エンティティ参照」という種類の入力方法で入れると、この店 ID をたどって、「仕入先ノート」に入力したデータ（仕入先名など）にアクセスできるようになります。

#### (4)-1 食材ノートの「コンテンツタイプ」の作成

まず、「コンテンツタイプ」（書式設定）を作ります。

ツールバー [サイト構築] → [コンテンツタイプ] → [+ コンテンツタイプの追加]。

「名前」欄に「食材」と入力すると、右の方に「システム内部名称: shikai[編集]」と表示されます。「shikai」では理解不能なので、[編集]をクリックして「システム内部名称」欄の名前を「foodstuff」に変更します。

また、「タイトルフィールドのラベル」欄に「食材 ID」と入力しておきます。

[保存してフィールドを管理] で「フィールドの管理」画面になります。

[+ フィールドの追加] → 「新しいフィールドの追加」欄で「テキスト (プレーン)」を選択。

「ラベル」欄に「食材名」と入力すると、右の方に「システム内部名称: field\_??????[編集]」と表示されるので、[編集]をクリックして「システム内部名称」欄の名前を「field\_foodstuff\_name」に変更します。

→ [保存して次へ] → 「フィールドの設定」画面ですが何もせず → [フィールド設定を保存] → [設定の保存]。

次は、フィールド「百 g 単価」の設定です。

[+ フィールドの追加] → 「フィールドタイプの選択」では [▼] → [数値 (整数型)] をクリックします。

「ラベル」は「百 g 単価」、システム内部名称は「field\_foodstuff\_unitprice」とします。

→ [保存して次へ] → [フィールド設定を保存] → [設定の保存]。

次に、フィールド「仕入先」を設定します。

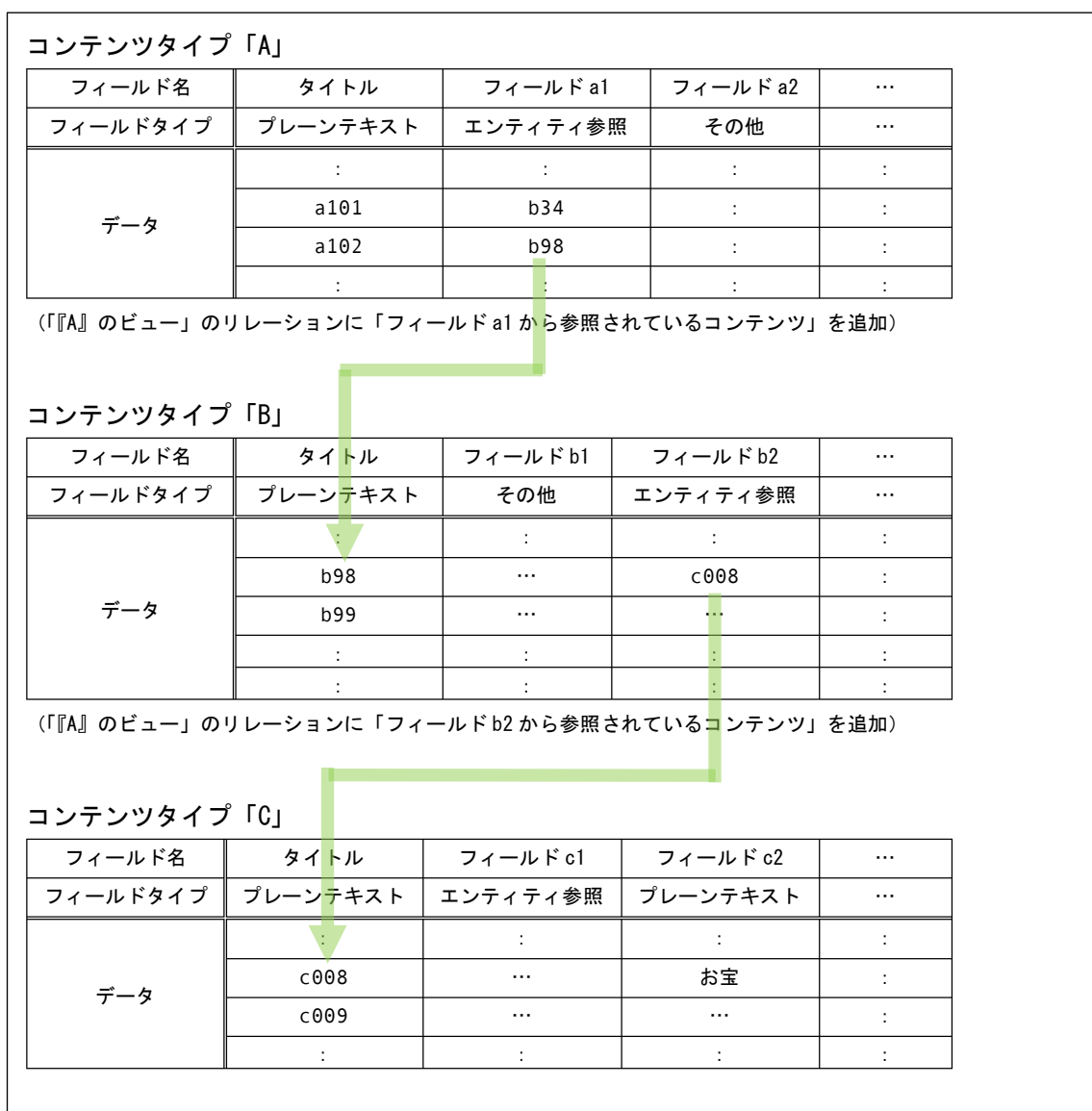
#### (4)-1-1 「エンティティ参照フィールド」の設定

「エンティティ参照フィールド」というのは、そのフィールドに入れるデータをたどって、他のコンテンツタイプの情報を参照（アクセス）することができるフィールドのことです。

これは「ビュー」で機能を発揮します。図9で原理を説明します。

今、「『コンテンツタイプ A』のビュー」を作ろうとします。「コンテンツタイプ A」にエンティティ参照型の「フィールド a1」があり、その中のデータが「コンテンツタイプ B」の「タイトル」のデータと同じで、ビューのリレーションとして「フィールド a1 から参照されているコンテンツ」が追加されていると、コンテンツタイプ B のフィールドデータをすべて参照表示できます。更にこの機能は、図 9 でコンテンツタイプ C が参照できるように、重層が可能です。

【図9: エンティティ参照フィールドを有する『コンテンツタイプ A』のビュー】



この機能により、コンテンツタイプ「食材」の中に、コンテンツタイプ「仕入先」と重複して、仕入先名や FAX や担当者を保持しなくて良くなります。

さて「仕入先」の設定ですが、[＋フィールドの追加] → 「フィールドタイプの選択」では[▼] → [コンテンツ] をクリックします。

「ラベル」欄は、「仕入先」と入力。システム内部名称は「field\_foodstuff\_supplier」にします。

[保存して次へ] → [フィールド設定を保存] → 「食材の仕入先 設定」画面になります。

「▼ 参照タイプ」欄の「コンテンツタイプ」は、「□仕入先」にチェック。→ [設定の保存]。

これで「エンティティ参照タイプのフィールド」が設定できました。

「フィールドの管理」画面になり、これまで設定したフィールドが表示されます。フィールドの並びは入力した順ではないかもしれませんが、気にしないでください。

#### (4)-2 食材ノートの「個別データ」の入力

ツールバー [コンテンツ] → [+ コンテンツを追加] → [食材] を選択すると、「食材の作成」画面になります。

表 2 の例示のようにデータを入力します。

例えば「食材 ID」欄には「f001」、「食材名」欄には「玉葱」、「百 g 単価」欄には「36」、「仕入先」欄には「s01」と入れ、→ [保存] → 先程入力した個別データが表示されます。同様に、表 2 を参照して f002 ～ f009 の個別データも入力しておきましょう。

#### (4)-3 食材ノートの「ビュー」表示

コンテンツタイプ「食材」に従って食材の情報を入力したので、今度は「ビュー」を使ってそれらを一覧表示させます。操作はほとんど「仕入先ノート」の場合と同じです。

ツールバー [サイト構築] → [ビュー] → [+ ビューを追加] → 「ビューを追加」画面。

「ビューの名前」欄に、「食材ノート」と入力すると、システム内部名称が表示されます。[編集] で理解できる語句（英字）に変更します。例えば「foodstuff\_note」。

「ビューの設定」欄は、表示は「コンテンツ」のまま、タイプ指定は「食材」を、並び順は「タイトル (=食材 ID)」を選びます。

「ページの設定」欄は、「□ページを作成する」に✓を入れます。

「Page title」欄は、「食材ノート」のまま。

「パス」欄は、ビューの名前と同じく「foodstuff\_note」としておきます。

「ページの表示設定」欄では、ディスプレイフォーマットを「テーブル」にします。

「表示件数」欄は空欄にし、「✓ ページャーを使用する」のチェックをはずします。

「□メニューリンクを作成」に✓を入れます。

「メニュー」欄も「リンクテキスト」欄もそのまま、[保存して編集] → ビューの編集画面に移ります。

「プレビュー」では、「タイトル」のカラムに「f001」、・・・、「f009」と、表示されます。これらは「食材 ID」なので、フィールドの [タイトル] を「食材 ID」に変えます。

コンテンツタイプ「foodstuff」に入力したその他の情報も表示させます。

先程の「フィールド」項の [追加] をクリックすると、「フィールドを追加」の小画面が現れます。

「Search」欄に「food (foodstuff の部分文字列)」と入力すると、「foodstuff」に関係した行だけが表示されます。

タイトルを見ると、「本文」以外は全て入力していますので、これらの項目の左側の□にチェックを入れ、[フィールドを追加して設定] をクリックします。

各フィールドの設定画面が現れますが、すべて何もしないで[適用して続ける] をクリックします。最後は[適用] ボタンになります。→ 編集画面に戻ります。

プレビューを見ると、食材名が最後に表示されていますので、これを表 2 のようになるようカラムを並び替えます。

プレビューを見ると、一応入力したデータどおりの一覧表ができています。

次に、「仕入先」は ID ではなく、仕入先名を表示させるようにします。

ビューの編集画面の右側の [▶ 高度] から→「リレーションシップ」の [追加] →「☐ field\_foodstuff\_supplier から参照されているコンテンツ」にチェック → [リレーションシップを追加して設定] → [適用]。

ビューの編集画面に戻るので、「フィールド」項の [追加] →「☐ 仕入先名」にチェック → [フィールドを追加して設定] →「リレーションシップ」欄の「リレーションを使用しない」を「field\_foodstuff\_supplier: コンテンツ」に変更 → [適用]。

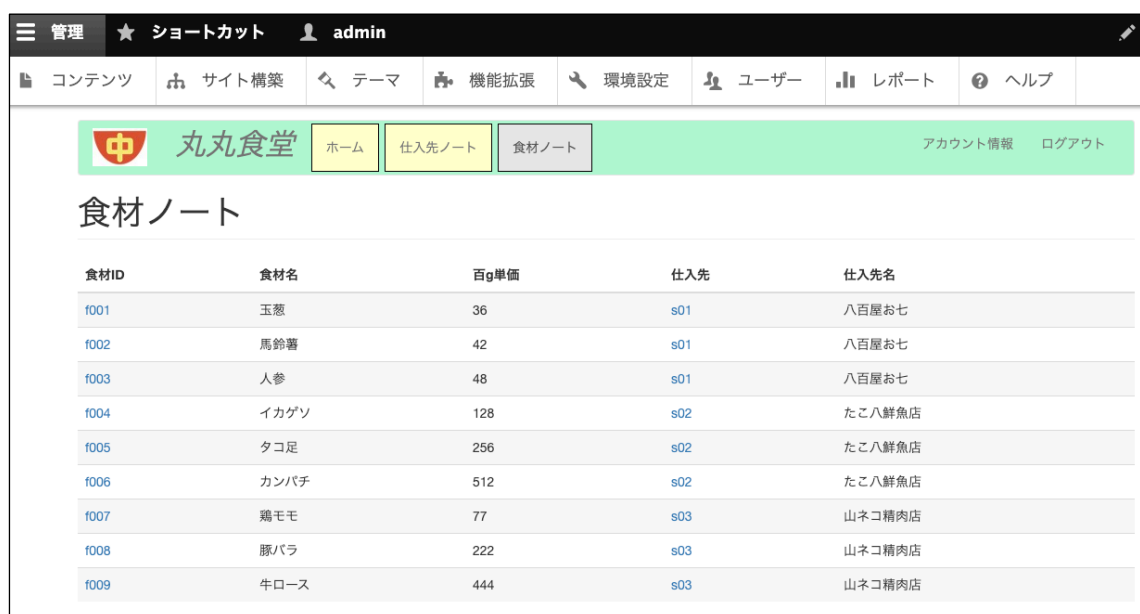
ビュー編集画面に戻るので、プレビューを確認します。ちゃんと仕入先 ID と仕入先名が表示されています。

[保存] します。

[サイトへ戻] ってみましょう。

ヘッダーに「食材ノート」というメニュータグができ、これをクリックすると、先程のプレビューで見たとおりの食材一覧表が表示されます (図 9)。

【図 9 : 「食材ノート」の画面】



食材ID	食材名	百g単価	仕入先	仕入先名
f001	玉葱	36	s01	八百屋お七
f002	馬鈴薯	42	s01	八百屋お七
f003	人参	48	s01	八百屋お七
f004	イカゲソ	128	s02	たこ八鮮魚店
f005	タコ足	256	s02	たこ八鮮魚店
f006	カンパチ	512	s02	たこ八鮮魚店
f007	鶏モモ	77	s03	山ネコ精肉店
f008	豚バラ	222	s03	山ネコ精肉店
f009	牛ロース	444	s03	山ネコ精肉店

## (5) 料理メニュー&レシピの作成

「料理メニュー&レシピ」もコンテンツですが、今度は別の作り方をしてみます。

外部ファイルを用いてデータを一括入力します。イメージファイルも外部から一括して入れます。

### (5)-1 料理メニュー&レシピの「コンテンツタイプ」の作成

これまでの「仕入先」や「食材」と同じように、まずコンテンツタイプを作ります。

コンテンツタイプの名前は「料理」、システム内部名称は「dish」とします。

「タイトルフィールドのラベル」は「料理 ID」とし、例：d01, d02, …と付番します。

追加フィールドは下表の項目とします。項目ごとに、フィールドタイプ、ラベル、システム内部名称、及び留意点を示します。

順番	フィールドタイプ	ラベル	システム内部名称	留意点
1	テキスト（プレーン）	料理名	field_dish_name	
2	画像	料理イメージ	field_dish_image	ファイル種：jpeg, png, gif ファイルディレクトリーは「myimage」
3	数値（整数）	単価（円）	field_dish_unitprice	
4	テキスト（プレーン）	宣伝文句	field_dish_salestalk	
5	コンテンツ	食材 ID	field_dish_foodstuff	許容する値の数→無制限、 参照タイプは「食材」、並び替え基準は「ID」、向き「昇順」
6	数値（整数）	食材量	field_dish_foodamount	許容する値の数→無制限
7	ファイル	調理法	field_dish_recipe	ファイル種：txt, pdf, doc, docx ファイルディレクトリーは「myimage」

### (5)-2 料理メニュー&レシピノートの「個別データ」の入力

念のため、ひとつだけ手入力で個別データを入力してみます。ファイルでの一括入力が入手く行ったら、後で消去します。

ツールバー [コンテンツ] → [+ コンテンツを追加] → [料理] を選択すると、「料理の作成」画面になります。

「ID」欄に「test」、「料理名」に「susi」、「料理イメージ」には適当な jpeg ファイル、「単価（円）」、「宣伝文句」も適当に入力してください。

「食材」のフィールドには複数のボックスに 1 つずつ食材 ID を入力してください。ボックスが足りなければ [別のアイテムを追加] でボックスを追加してください。

入力の途中からポップアップメニューが入力できる情報を青字で表示してくれますので、それから選んでも構いません。

「調理法」は、適当な txt, pdf, doc, docx ファイルを選んで入れてみてください。

[保存] でコンテンツがちゃんとできているか、確認してください。

外部ファイルを用いて一括してデータ入力するためには、Drupal にモジュールを追加する必要があります。画像ファイル等アップロード用モジュール「IMCE」と、CSV ファイル注入用モジュール「CSV importer」です。



### (5)-2-1 モジュール「IMCE」と「CSV importer」の導入

ブラウザで新しいタブを追加し、「Drupal IMCE module download」という語句で検索すると、検索結果の中に「IMCE | Drupal.org」というサイトがあると思います。

それをクリックし、現れた画面をずっと下までスクロールすると、「Downloads」欄の黄緑色の四角の中に↓に続き、「tar.gz(144.49KB)」という青色の記載があります。これを右クリックして「リンクのアドレスをコピー」（または「リンクをコピー」）します。

Drupal の管理画面に戻り、サイト管理ツールバーの「機能拡張」→「+新しいモジュールをインストール」をクリック。

「新しいモジュールをインストール」画面の「次の URL からインストールする」欄に先程のアドレスを貼り付け、「インストール」をクリック。

「✓ インストールは正常に終了しました。」という表示が出たら、「次のステップ」項で「[新しく追加されたモジュールを有効にします](#)」をクリックします。「拡張」画面に戻るので、下の方に加わった「☐ Imce File Manager」にチェックを入れ、最下段の「インストール」をクリック。

同様にして、ブラウザの別タブで、「Drupal CSV importer module download」という語句で検索し、「CSV Importer | Drupal.org」をクリック。「Downloads」欄の「8.x-1.11…」の黄緑色の四角の中の「tar.gz(24.69KB)」という青色の記載を右クリックして「リンクのアドレスをコピー」（または「リンクをコピー」）。Drupal の管理画面に戻り、このモジュールをインストール&有効化してください。

インストールが完了したらサイト管理ツールバーの「環境設定」をクリックしてみましょう。「メディア」欄に「Imce File Manager」が、「開発」欄に「CSV importer」という項目が追加されています。

### (5)-2-2 「IMCE」でファイルのアップロード

画像ファイルは、各「コンテンツを追加」する際に、個別に指定して入力することもできますが、コンテンツが沢山ある場合は、複数の画像ファイルを一括してサーバにアップロードしておき、コンテンツには識別子で紐付けする、という方法で対応することができます。

まず、必要な画像ファイルを準備し、どこかのフォルダに貯めておきます。

次に、「任意のコンテンツタイプのデフォルトの body 欄を出し、編集メニューバーにある『チェーン』の形をしたアイコン（リンク機能）をクリック」します。

「リンクを追加」という小画面が開くので、「[Open File Browser](#)」をクリック。

画面の左カラムに public:// から下流のフォルダ構成が表示されます。表示されない場合は、「public://」をクリックしてみてください。いくつかのフォルダが表示され、その中に「myimage」があるはずです。

「myimage」フォルダをクリックして開き、「アップロード」→「+ ファイルを追加」→ 自分の PC 内のフォルダ・ファイル構成が表示されるので、必要な画像ファイルを選択（複数可）。→「開く」（または「アップロード用に選択」）で、ファイルがアップロードされます。

「調理法」の pdf ファイルも同様にして、必要なファイルを選択し、[開く]（または [アップロード用に選択]）をクリックします。

「File Manager」（または「File Browser」）の小画面を閉じ、「リンクを追加」を閉じます。

#### (5)-2-3 インポートしたファイルや、エンティティ参照フィールド値の ID を調べておく

まずファイル ID（識別子）です。管理メニュー [サイト構築] → [ビュー] で、ファイルという名前のビューを [編集] します。ビュー編集画面で「フィールド」欄の最初の [ファイル: ファイル ID(FID)] をクリックし、「表示を除外」のチェックをはずすと、「プレビュー」欄でアップロードしたファイル名の左に FID (ID) が表示されます。この値をファイルの ID として、CSV データの中に盛り込みます。

次に、エンティティ参照フィールドも CSV データに入れる場合、同様にして、管理メニュー [サイト構築] → [ビュー] で、コンテンツという名前のビューを [編集] します。ビュー編集画面の「フィールド」欄で [追加] します。タイトル「ID」、カテゴリー「コンテンツ」という行の左の□をチェックし、→ [フィールドを追加して設定] → [適用] で、「コンテンツ: ID(ID)」というフィールドが追加されます。プレビューを見ると「ID」というカラムができていて、下の方に数値が並んでいます。これが「コンテンツのシステム内部 ID」です。この値をエンティティ参照フィールドのデータとして、CSV データの中に盛り込みます。

なお、ID の並び順とファイルやデータの並び順はちぐはぐな場合がありますので、ご注意ください。

#### (5)-2-4 CSV 形式のデータファイルを作る

適当なテキストエディタまたは Microsoft の Excel で、データファイルを作ります。

「CSV」というのは、「Comma Separated Value」の略で、「Comma(カンマ)で Separated (区切った) Value (値)」のことです。

CSV 形式のデータは、「表」をイメージすれば良いです。

最初の行は「ヘッダー」であり、項目名をカンマ区切りで横に並べます。Drupal では、この項目名はコンテンツタイプを作ったときのフィールドのシステム内部名称に一致させておく必要があります。その他に付け加える情報もあります。フィールドの並び順は自由です。カンマの後に半角スペースを補って見栄えを良くしてもよさそうです。

例えば以下の通りです。最初の 2 項目 (title と langcode) は必須です。

```
title, langcode, body, field_dish_name, field_dish_image|target_id, field_dish_image|alt,
field_dish_unitprice, field_dish_salestalk, field_dish_foodstuff, field_dish_foodstuff,
field_dish_foodstuff, field_dish_foodamount, field_dish_foodamount, field_dish_foodam
ount, field_dish_recipe <LF>
```

ここで、

「title」は、料理 ID です。

「langcode」は、「言語コード」と呼ばれるもので、日本語なら「ja」を設定します。

「body」は、コンテンツの中の「body」です。コンテンツタイプ「料理」では、body は

空（カラ）なので、何も入れないでください。

「field\_dish\_name」は、料理名です。

「field\_dish\_image|target\_id」には、画像ファイルの ID(FID)を入れます。

「field\_dish\_image|alt」には、画像ファイルの適当な英語代替名を入れます。

「field\_dish\_unitprice」は、料理の単価です。半角数字の整数です。

「field\_dish\_salestalk」は、宣伝文句です。256 文字以内で記載します。

「field\_dish\_foodstuff」と「field\_dish\_foodamount」はそれぞれ3つずつ並べていますが、コンテンツタイプ作成では複数個（無制限）データを入れるようにしていました。今回はとりあえず3つまで入れられるようにします。3つ未満の場合は空欄にしておきます(フィールドとしては必要です)。

「field\_dish\_foodstuff」はエンティティ参照フィールドですので、「食材 ID」ではなく、前項で調べた「コンテンツのシステム内部 ID」を記載します。

なお、foodstuff は若い順に、foodmount は foodstuff と対応させた順に記載します。

「field\_dish\_recipe」には、先程調べたファイルの ID (FID) を入れます。

〈LF〉は、ラインフィードのことです。Microsoft の Excel ファイルを CSV ファイルに変換した場合や、エディタの設定によっては、「改行」が「CR+LF」になったりしますが、CSV ファイルでは「改行」は LF のみです。ご注意ください。

ヘッダーとともに、表 3 のようにデータを入れていきます。

#### 【表 3 : 「料理メニュー&レシピ」の CSV ファイル】

title, langcode, body, field\_dish\_name, field\_dish\_image|target\_id, field\_dish\_image|alt, field\_dish\_unitprice, field\_dish\_salestalk, field\_dish\_foodstuff, field\_dish\_foodstuff, field\_dish\_foodstuff, field\_dish\_foodamount, field\_dish\_foodamount, field\_dish\_foodamount, field\_dish\_recipe (ここまでがヘッダー。改行は以下も含め LF のみ。)

d01, ja,,	ギョーザ,	9,	gyoza,	200,	美味一番,	5,	7,	9,	2,	4,	6,	14
d02, ja,,	ラーメン,	7,	ramen,	500,	美味二番,	7,	9,	11,	3,	6,	9,	13
d03, ja,,	チャーハン,	8,	chahan,	400,	美味三番,	11,	12,		7,	8,		12
d04, ja,,	麻婆豆腐,	5,	mabotofu,	600,	美味四番,	6,	7,	10,	2,	5,	8,	11
d05, ja,,	野菜炒め,	6,	yasai,	300,	美味五番,	8,	10,	13,	1,	4,	7,	10

これを「menurecipe.csv」というファイル名で適当な所に保存します。

#### (5)-2-4 CSV ファイルのインポート

モジュール「CSV importer」を使って、Drupal 内に「menurecipe.csv」をインポートします。

[環境設定] → [CSV importer] → 「Import CSV」画面になります。

「Select entity type」欄は、「コンテンツ」。「Select entity bundle」欄は、「料理」。

「Select delimiter」欄は、「,(カンマ)」のまま。「Select CSV file」欄は、「menurecipe.csv」を選択して [開く] (または [アップロード用に選択]) → [インポート]。

インポートが完了したら、管理メニューの [コンテンツ] → 例えば d03 の [編集] を

クリックし、「料理イメージ」や「食材」が正しく入っているかどうか確認してください。

うまくできていなかったら、先程入れたコンテンツ d01 から d05 をすべて削除し、エディタを変えてみるとか、空白を削除するとか、「field\_dish\_foodamount」になっていないか、改行が CR+LF になっていないか、コピペで「(ここまでがヘッダー…」も貼り付けていないか、など調べてみてください。なお、文字コードは「UTF-8」です。

### (5)-3 料理メニュー&レシピノートの「ビュー」表示

料理メニュー&レシピノートの「ビュー」を作ります。例によって、操作はほとんど「仕入先ノート」の場合と同じです。

ツールバー [サイト構築] → [ビュー] → [+ビューを追加] → 「ビューを追加」画面。

「ビューの名前」欄に、「料理ノート」と入力すると、システム内部名称が表示されます。

[編集] で理解できる語句 (英字) に変更します。例えば「dish\_note」とします。

「ビューの設定」欄は、表示は「コンテンツ」のまま、タイプ指定は「料理」を、並び順は「タイトル (=ID)」を選びます。

「ページの設定」欄は、「☐ ページを作成する」に ☒ を入れます。

「Page title」欄は、「料理ノート」のまま。

「パス」欄は、ビューの名前と同じく「dish\_note」としておきます。

「ページの表示設定」欄では、ディスプレイフォーマットを「テーブル」にします。

「表示件数」欄は空欄にし、「☒ ページャーを使用する」のチェックをはずします。

「☐ メニューリンクを作成」に ☒ を入れます。

「メニュー」欄も「リンクテキスト」欄もそのまま、[保存して編集] → ビューの編集画面に移ります。

「プレビュー」では、「タイトル」のカラムに「d01」、「d02」、・・・、「d05」と、縦に表示されているのみです。これらは「ID」なので、編集画面の「フィールド」項の「(タイトル)」をクリックして「料理 ID」に変えます。

コンテンツタイプ「dish」にインポートしたその他の情報も表示させます。

先程の「フィールド」項の [追加] をクリックすると、「フィールドを追加」の小画面が現れます。

「Search」欄に「dish」と入力すると、「dish」に関係した行だけが表示されます。

これらの項目の左側の ☐ にチェックを入れ、[フィールドを追加して設定] をクリックします。

各フィールドの設定画面が現れますが、ほとんど何もしないで [適用して続ける] をクリックします。ただ、「料理イメージ」の「画像のスタイル」欄は「サムネイル(100×100)」にしておきます。最後は [適用] ボタンになります。→ 編集画面に戻ります。

プレビューを見ると、「BODY」のカラムには何も入っていないので、削除しましょう。また、「食材 (FIELD\_DISH\_FOODSTUFF: デルタ)」と「食材量 (FIELD\_DISH\_FOODAMOUNT: デルタ)」という項にわけのわからない数字が入っているので、これらのフィールドも削除しましょう。

編集画面の「フィールド」項の [コンテンツ: Body (Body)]、[コンテンツ: 食材: デルタ (食材 (field\_dish\_foodstuff: デルタ)) ]、[コンテンツ: 食材量: デルタ (食材量

(field\_dish\_foodmount:デルタ))」のそれぞれについて、クリックし、最下段の「削除」をクリックします。

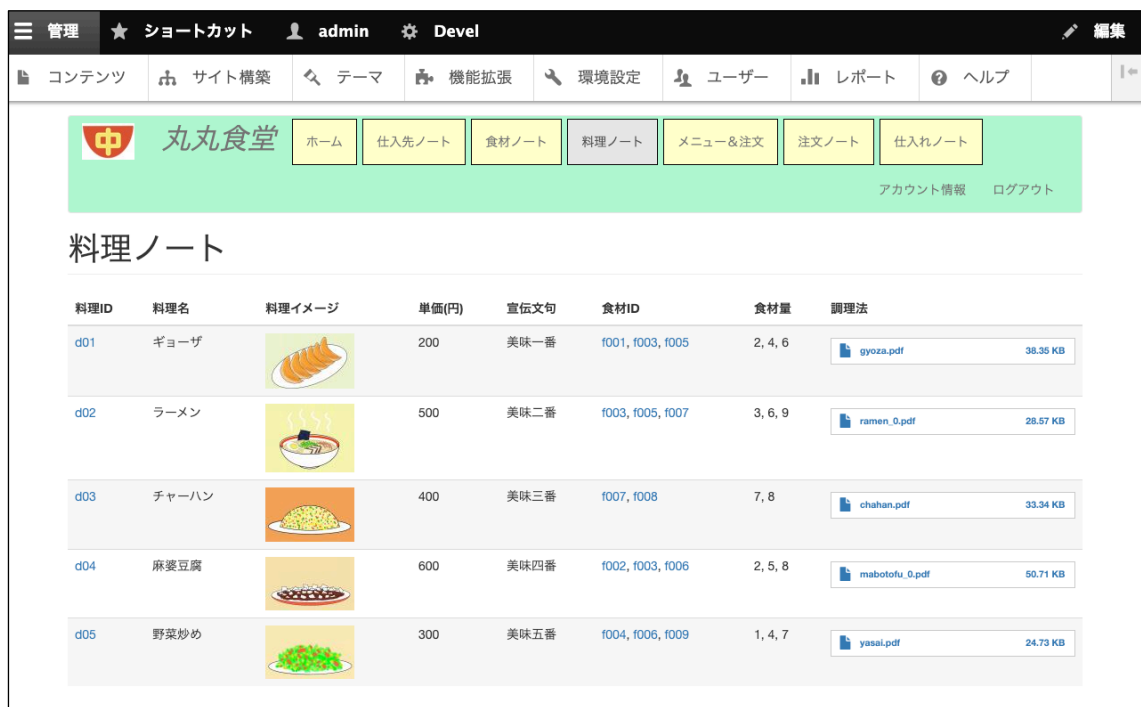
プレビューを見ると、一応、「1 コンテンツ 1 行」の表示になっていますが、項目の並び順がめちゃめちゃですので、並び順を、ID、料理名、料理イメージ、単価、宣伝文句、食材、食材量、調理法の順にしましょう。フィールド項の「▼」で「並べ替え」をします。






「保存」→「サイトへ戻る」で、メニュータグ「料理ノート」ができていることと、それをクリックすると「料理ノート」が一覧表示されることを確認しましょう。

メニュータグ「料理ノート」は、必ずしも「仕入先ノート」と「食材ノート」の右に表示されるとは限りません。メニュータグの順番を変えるには、[サイト構築] → [メニュー] → タイトル「Main navigation」の操作 [メニューの編集] をクリックし、メニューリンクのカラムの十字マークをドラッグ → [保存] でできます。

ここまでうまく行ったら、コンテンツ「test」を削除しておきましょう (図 10)。

【図 10 : 「料理ノート」の画面】



料理ID	料理名	料理イメージ	単価(円)	宣伝文句	食材ID	食材量	調理法
d01	ギョーザ		200	美味一番	f001, f003, f005	2, 4, 6	<a href="#">gyoza.pdf</a> 38.36 KB
d02	ラーメン		500	美味二番	f003, f005, f007	3, 6, 9	<a href="#">ramen_0.pdf</a> 28.57 KB
d03	チャーハン		400	美味三番	f007, f008	7, 8	<a href="#">chahan.pdf</a> 33.34 KB
d04	麻婆豆腐		600	美味四番	f002, f003, f006	2, 5, 8	<a href="#">mabotofu_0.pdf</a> 50.71 KB
d05	野菜炒め		300	美味五番	f004, f006, f009	1, 4, 7	<a href="#">yasai.pdf</a> 24.73 KB

## (6) 注文ノートの作成

### (6)-1 注文ノートの「コンテンツタイプ」の作成

以下のようなコンテンツタイプを作ります。

コンテンツタイプ名：注文。

システム内部名称：order。

タイトルフィールドのラベル：「注文 ID」に変更。

以下、追加するフィールドを表示します。

順番	フィールドタイプ	ラベル	システム内部名称	留意点
1	テキスト（プレーン）	座席 No.	field_orderer	
2	日付	注文日時	field_order_time	「デフォルトの日付」は「現在の日時」を選択
3	コンテンツ	注文料理	field_order_dish	コンテンツタイプ「料理」にチェック
4	数値（整数型）	注文数	field_order_quantity	

## (6)-2 注文ノートの「個別データ」の入力

これまで作成した「仕入先」、「食材」、及び「料理」は固定的であり、減多に増減・変更するものではありません（現実的には食材の値段は変動しますが、ここでは暫くは安定しているものとします。）。

しかし、「注文」は随時発生し、増えていきます。しかも、その入力は、不慣れな接客係（複数かも）がしたり、お客様にしてもらったりする可能性があります。

そこで、誰でもミスすることなく簡単に注文入力できる画面を作ります。

ここでは、図 11 のような「メニュー&注文」画面を考えます。まず座席番号を入力し、料理メニューを見ながら、料理番号と皿数を入力すると、料理名、単価、小計を自動表示し、合計金額を更新してくれます。

【注文ボタン】をクリックすると注文が確定します。

【図 11 : 「メニュー&注文」画面アウトライン】

ヘッダー

座席No.

品番	料理名	単価	数量	小計
<span style="background-color: yellow; border: 1px solid black; display: inline-block; width: 40px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 100px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 40px; height: 20px;"></span>	<span style="background-color: yellow; border: 1px solid black; display: inline-block; width: 30px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 60px; height: 20px;"></span>
<span style="background-color: yellow; border: 1px solid black; display: inline-block; width: 40px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 100px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 40px; height: 20px;"></span>	<span style="background-color: yellow; border: 1px solid black; display: inline-block; width: 30px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 60px; height: 20px;"></span>
<span style="background-color: yellow; border: 1px solid black; display: inline-block; width: 40px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 100px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 40px; height: 20px;"></span>	<span style="background-color: yellow; border: 1px solid black; display: inline-block; width: 30px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 60px; height: 20px;"></span>
<span style="background-color: yellow; border: 1px solid black; display: inline-block; width: 40px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 100px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 40px; height: 20px;"></span>	<span style="background-color: yellow; border: 1px solid black; display: inline-block; width: 30px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 60px; height: 20px;"></span>
<span style="background-color: yellow; border: 1px solid black; display: inline-block; width: 40px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 100px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 40px; height: 20px;"></span>	<span style="background-color: yellow; border: 1px solid black; display: inline-block; width: 30px; height: 20px;"></span>	<span style="background-color: green; border: 1px solid black; display: inline-block; width: 60px; height: 20px;"></span>

料理メニュー

注文ボタン 合計

: 入力欄       : 自動表示

### (6)-2-1 モジュール「webform」の導入

まず、入力画面を作ってくれるモジュール「webform」を追加します。

「IMCE」や「CSV importer」を導入した方法と同様に、ブラウザの新しい画面を開き、「Drupal webform download」という語句で検索し、適合したバージョンの webform を見つけます。

「Download」欄の“tar.gz”を右クリックして [リンク (のアドレス) をコピー] → Drupal の管理画面に戻り、→ [機能拡張] → [+新しいモジュールをインストール] → 「次の URL からインストール」欄にアドレスを貼り付け、→ [インストール] → [・新しく追加されたモジュールを有効にします] → 「拡張」画面に戻ります。

薄く表示された「名前と説明でフィルター」欄に「webform」と入力して表示されるモジュール名で、左のチェックボックスにチェックを入れられる項目すべてにチェックを入れ、最下段の [インストール] をクリック。→ 数分後、「拡張」画面に戻ります。

私の場合、なかなか戻らなかったのが、[インストール] ボタンを再度押してみると、「webform」以外の項目のチェックをはずすとかの操作をしていると、いきなり「拡張」画面に戻り、先程チェックを入れた webform 関連項目すべてにチェックが入っていました。

項目の中で、チェックの入っていない「Webform Custom Form Example」が見たいので、右横の [▶] をクリックします。「Webform Devel(無効), devel(missing)が必要」との表示があるので、「devel」を追加でインストールし、「Webform Devel」を有効にすることになります。

モジュール「devel」を、これまでのモジュール追加方法と同様に探してインストールして有効化し [インストール] します。次に、「Webform Devel」を有効化し [インストール] します。

それから「Webform Custom Form Example」にチェックを入れ、[インストール] で有効化します。

いちいちチェックボックスに✓を入れ、一番下の [インストール] ボタンをクリックしなければ「有効化」にならないので、ご注意ください。

### (6)-2-2 モジュール「Token」の導入

新しく「Token」というモジュールを追加します。

これは、Drupal の正規のデータアクセス方法とは異なり、小手先でデータにちょこっとアクセスするために使う「オマジナイ」のようなツールと私は考えています。

これまでのモジュールと同様に探して導入してください。

### (6)-2-3 「注文入力」画面の作成

まず、「メニュー&注文」画面の右側の「注文入力」画面を作ります。

[サイト構築] → [Webforms] → [+ Add webform ]。

「タイトル」欄には「注文入力」、システム内部名称は「dish\_order」で → [保存]。

「注文入力」画面で、[+ Add element]。

まず、座席 No. の入力欄を作ります。「タイプ」は任意の文字列を許容するため「テキストフィールド」を選んで [Add element]。タイトルは「座席 No.」、対応する「キー」は「sheetno」→ [保存]。

次に、「品番・料理名・単価・数量・小計」のセットを 1 行ずつ設定します。

セットを作るには「flexbox」というタイプを使います。

[+ Add element] → タイプ「Flexbox layout」の [Add element] → キーは「flexbox1」("1" を付番しています) → [保存]。

「注文入力」画面に戻り、タイトルに [Flexbox1] が加わっています。その行の右の方に [+ Add element] というボタンがあります。これをクリックすると、Flexbox1 のセットの中の element として、項目が追加されます。

この [+ Add element] をクリックして、まず料理 ID の入力欄を作ります。タイプは「Entity select」、タイトルは「料理 ID」、キーは「dishid1」(付番 1 を付けています。以下同様です。)、コンテンツタイプは「☐ 料理」にチェック → [▶ FORM DISPLAY] 中の「Title display」を「invisible」にして → [Save + Add element]。

直ちに次の element タイプ入力になります。

「料理名」のタイプは「Computed token」、タイトル「料理名」、キー「dishname1」、Computed value/markup には以下の token を書き込みます。

```
[webform_submission:values:dishid1:entity:field_dish_name]
```

この意味は、「webform で発信する dishid1 のタイトル値で、指定したタイプ (「料理」) にアクセスし、その中の field\_dish\_name (料理名) のデータを引用して表示しなさい。」ということのようです。

さらに、「☐ Automatically update the computed value using Ajax」にチェック、Title display を「invisible」にして [Save + Add elements]。

同様にして「単価 (円)」も、タイプは「Computed token」、タイトル「単価 (円)」、キー「dishprice1」、Computed value/markup には以下の token を書き込みます。

```
[webform_submission:values:dishid1:entity:field_dish_unitprice]
```

また、「☐ Automatically update the computed value using Ajax」にチェックし、Title display を「invisible」にして [Save + Add element] してください。

「数量」は、タイプ「数値」、タイトル「数量」、キー「dishquantity1」、Title display は [Invisible] で → [Save + Add element]。

「小計」は、タイプ「Computed Twig」、タイトル「小計」、キー「dishsum1」とします。Computed value/markup に書き込む Twig は以下の呪文 (これは達人から教わったので、このまま覚えてください。token に比べ少し長いので「呪文」。) を入れてください。

```
{% if data.dishprice1|length and
data.dishquantity1|length %}{{ data.dishprice1 *
data.dishquantity1 }}{% else %}{% endif %}
```

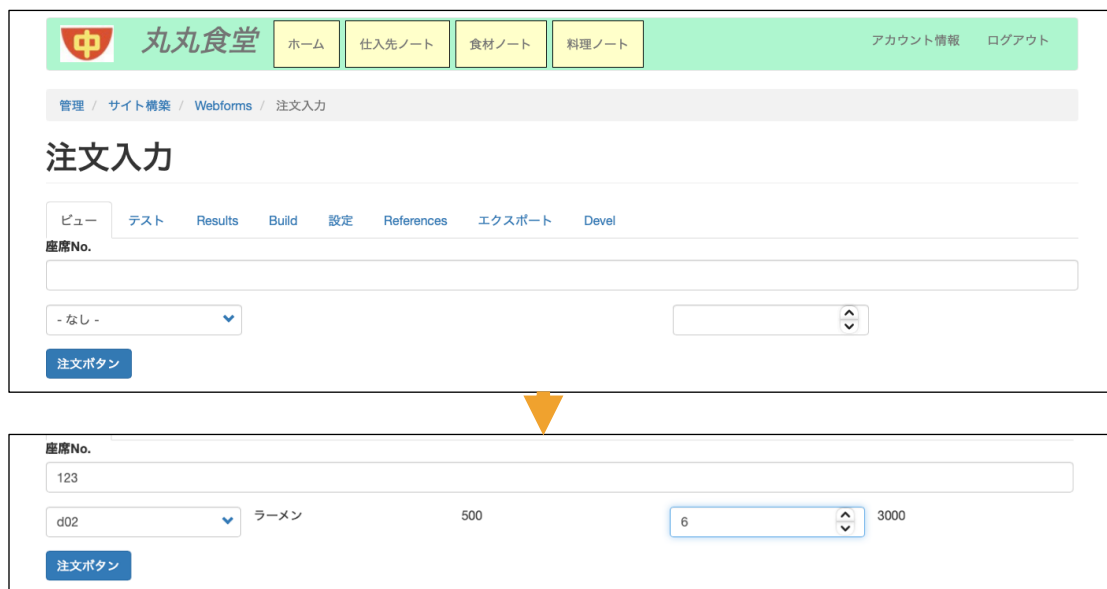
また、「☐ Automatically update the computed value using Ajax」にチェック → Title display は [Invisible] → [保存]。

「注文入力」の「Build」画面でまずこれだけを設定し、[Save elements] して [ビュー] でブラウザ画面を確認してみてください。

図 11 の上図が表示され、ボックス欄 (座席 No.、品番、数量) に適当に入力すると、下図のように料理名、単価、及び小計が自動的に表示されるはずです。



【図 11：注文入力画面】



同様にして、5 品目までの注文ができる様式を作ります。

これまでの付番"1"を順番に "2", "3", "4", "5" と変えて Flexbox2 … 5 を作ってみてください。

最後に、flexbox5 の中ではなく、「注文入力」の「Build」画面の上の方にある [+ Add element] で、「合計」element を設定します。

タイプは「Computed Twig」、タイトル「合計」、キー「dishtotal」とします。

Computed value/markup に書き込む Twig は以下のとおりです。

```
{% if data.dishquantity1|length %}{ { data.dishprice1 *  
data.dishquantity1 + data.dishprice2 * data.dishquantity2 +  
data.dishprice3 * data.dishquantity3 + data.dishprice4 *  
data.dishquantity4 + data.dishprice5 *  
data.dishquantity5 } }{% else %}{% endif %}
```

「data.dishprice1 \* data.dishquantity1」は「data.dishsum1」で良さそうですが、うまく表示してくれません。

なお、[Submit button] は、[編集]（または [カスタマイズ]）で「送信 button label」名称を「注文ボタン」などに変えたら、分かりやすいと思います。Build 画面では変更が反映されませんが、ビュー画面ではちゃんと「注文ボタン」になっています。

[ビュー] で画面を見てみると図 12 のようになります。

この事例は、既に座席 No. 及び二つの料理とその数を入力しています。

[Build] で最下段の [Save elements] をクリックし、構成を確定しておきましょう。

【図 12：注文入力画面】

#### (6)-2-4 「メニュー&注文」画面の作成

「メニュー&注文」画面は、まずビュー「(料理) メニュー」画面を作り、それを「コンテンツ・ブロック」として画面の中央に配置し（自動的に配置されます）、「注文入力」画面を右側（「Secondary ブロック」）に配置する、という作り方をします。

##### (6)-2-4-1 ビュー「(料理)メニュー」の作成

ツールバー [サイト構築] → [ビュー] → [+ ビューを追加]。

「ビューの名前」を「メニュー&注文」、システム内部名称を「menu\_order」、ビューの設定では、タイプ指定を「料理」とします。

「☐ ページを作成する」にチェック → 「Page title」はそのまま、「パス」は「menu\_order」としておきます。

「ページの表示設定」では「ディスプレイフォーマット」を「テーブル」に変えます。

「表示件数」のデフォルト「10」は消去し、「☐ ページャーを使用する」のチェックを外し、「メニューリンクを作成」にチェックを入れます。出てきた「メニュー」欄を「Main navigation」に変更し、「リンクテキスト」欄はそのまま → [保存して編集] → ビュー「メニュー&注文」編集画面になります。

フィールドを編集・追加します。

「タイトル」の表示は「タイトル」から「料理 ID」に変えます。

図 10 の「料理名」、「料理イメージ」、「単価 (円)」及び「宣伝文句」を追加します。

方法は、(5)-3 料理メニュー&レシピノートの「ビュー」表示 を参照してください。

なお、項目名はメニューらしく、それぞれ「品名」、空欄、「円 (税込)」、「おすすめ」と

します。また「並び替え基準」は、「タイトル」を「昇順で並び替え」で追加し、「投稿日時（降順）」をクリックして「削除」します。

「保存」し、「サイトへ戻」って、メニュー「メニュー&注文」をクリックしてみてください。図 13 のようになっていればオッケーです。

【図 13：「メニュー&注文」のうち「メニュー」だけの画面】

中丸丸食堂

ホーム

仕入先ノート

食材ノート

料理ノート

メニュー&注文

アカウント情報

メニュー&注文

料理ID	品名		円 (税込)	おすすめ
d01	ギョーザ		200	美味一番
d02	ラーメン		500	美味二番
d03	チャーハン		400	美味三番
d04	麻婆豆腐		600	美味四番
d05	野菜炒め		300	美味五番

#### (6)-2-4-2 「メニュー&注文」画面の作成

ツールバー「サイト構築」→「ブロックレイアウト」→ Secondary「ブロックを配置」→「Webform」の「ブロックを配置」→「ブロックの設定」。

「タイトル」欄に書かれている「Webform」を「注文入力」に変更します。画面が変わりますが、下の方に「システム内部名称」欄があるので、これを例えば「order\_input」に変更しておきます。

「Webform」欄には、システム内部名称「dish\_order」ではなくて、タイトルどおりの「注文入力」と入力します。（システム内部名称を使わないので、頭が混乱します。）

「閲覧の制限」欄の「ページ」項は、「/menu\_order」と入力します。

→「ブロックの保存」→「ブロックレイアウト」画面の「Secondary」の中に「注文入力」が入っています。

「サイトへ戻る」と、メニュー「メニュー&注文」の画面の右側に「注文入力」画面が表示されていますが、非常に窮屈で、入力値が表示されません。

そこで、CSS を使って左右の画面の幅を変えます。

ツールバー「環境設定」→「Asset Injector」→「CSS インジェクター」→「+ Add Css Injector」→「Add Css Injector」の設定画面になります。

「ラベル」欄に「menuordercss」と入力すると、システム内部名称も同じになります。

「コード」欄には、以下の CSS を記述します。

```
.col-sm-9 {
  width: 650px;
}
.col-sm-3 {
  width: 500px;
}
```

これは、「クラス名『col-sm-9』の画面（メニュー画面）幅を 650 ピクセルに、『col-sm-3』の画面（注文入力画面）幅を 500 ピクセルにしてください。」という意味です。（ピクセル数は適当です。）

「条件」欄の「ページ」項をクリックして、この CSS を使う画面のパスを指定します。それは「メニュー&注文」画面のパス「/menu\_order」です。

「保存」→「サイトへ戻る」で画面が図 14 のようになっていることを確認します。

【図 14：「メニュー&注文」画面】

料理ID	品名	円 (税込)	おすすめ
d01	ギョーザ	200	美味一番
d02	ラーメン	500	美味二番
d03	チャーハン	400	美味三番
d04	麻婆豆腐	600	美味四番
d05	野菜炒め	300	美味五番

**注文入力**

座席No.

d01	ギョーザ	200	<input type="text" value="2"/>	400
d03	チャーハン	400	<input type="text" value="4"/>	1600
d05	野菜炒め	300	<input type="text" value="3"/>	900
- なし			<input type="text"/>	
- なし			<input type="text"/>	

合計 2900

とりあえず、図 11 で考えた内容は、これで達成したと考えて良いでしょう。

更に見栄えを良くするという課題もありますが、別途 CSS を勉強して改良してください。

### (6)-3 webform「dish\_order」のデータを、コンテンツタイプ「order」に移す

「Webform Content Creator」は、webform のデータをコンテンツに移してくれるモジュールです。

「注文入力」では、ひとつの注文に最大 5 つの入力が可能としました。webform 内ではこれらの入力データは「ひとかたまり」になっていて、個々の料理について細かく操作することは困難です。そこで、これらのデータをコンテンツに移す際、バラバラにして「ひとつの注文料理の情報はひとつのコンテンツ」へ入れることにします。

例えば、webform「dish\_order」の 1 番目の料理はコンテンツタイプ「order」へ、同じく 2 番目の料理は次のコンテンツタイプ「order」へ、3 番目の料理は更に次のコンテンツタイプ「order」へ、…、と移します。コンテンツタイプ「order」の各コンテンツにはユニークな ID 番号 (nid) が付番されますので、移されたコンテンツは料理ごとに別々に扱うことができるようになります。

ただ、ここで「非常に重要な課題」を覚えておいてください。それは、ある座席の客が1〜4種類の料理しか注文しなくても、常に5つの注文コンテンツができてしまうことです。1種類の料理しか注文しなかったら、残りの4つのコンテンツは不要な「空（カラ）コンテンツ」です。注文があるたびに空コンテンツが増えるので、管理者が随時これらを自力で削除するか、自動的に削除する機能を構築する、とかしなくてはなりません。今回は手に余るので、将来的な「課題」として留めておきます。

#### (6)-3-1 モジュール「Webform Content Creator」の導入

これまでのモジュールと同じ方法で導入します。

復習すると、ブラウザ検索にて「Drupal webform content creator download」という語句で検索し、適合する最新バージョンのモジュールを探しあて、「tar.gz」のリンク（アドレス）をコピーしてDrupalに戻り、ツールバー [機能拡張] → [+ 新しいモジュールをインストール] → 「次の URL からインストールする」欄にペースト → [インストール] → [有効化] → 「☐ Webform Content Creator」にチェックを入れ → （最下段の）[インストール]、の手順を踏みます。

#### (6)-3-2 モジュール「Webform Content Creator」の利用

このモジュールの使い方は、他のモジュールとは違います。

ツールバー [拡張機能] → 「Webform Content Creator」の説明文の「▶」をクリックして「▼」に変えると、マシン名・バージョン・「…が必要」、に続いて「Help」、「権限」、「構成」と表示されます。この「構成」をクリック → 「Webform Content Creator の一覧表」が表示されます。→ [+ Add configuration] → 「Add Webform Content Creator entity」という画面になります。

「タイトル」欄は例えば「dishorder2order1」（webform のシステム名称+2+コンテンツタイプのシステム名称+1）とします。「2」は「to」のことです。Content Creator のシステム内部名称も、同じです。名前の最後の「1」は、ひとつの注文入力の中の何番目かを示す数字です。「1」から「5」まで Content Creator を作ります。

（上記の [+ Add configuration] から、下の [保存] まで、繰り返して設定します。）

「webform」欄は「注文入力」を選択、「Entity Type」欄は「コンテンツ」のまま、「バンドル」欄は「注文」を選択 → [保存] で、画面が「Content Creator の一覧表」に戻ります。「dishorder2order1」行の「フィールドの管理」をクリックすると、「フィールドの管理」画面になります。

「☐注文料理(field\_order\_dish)」にチェックを入れ、[-選択- ▼] をクリックして「料理 ID (dishid1) - Entity select」を選択。

「☐注文数(field\_order\_quantity)」にチェックを入れ、[-選択- ▼] をクリックして「数量(dishquantity1) - 数値」を選択。

「☐座席 No.(field\_orderer)」にチェックを入れ、[-選択- ▼] をクリックして「座席 No.(sheetno) - テキストフィールド」を選択。

「☐注文 ID (title)」(【注意！】「ID(nid)」ではありません) にチェックを入れ、「カスタム」のカラムの□（「選択」欄の左）をクリックすると、右の灰色ボックスが白くなり

ます。ここに「[webform\_submission:sid] 1」と入力します。「1」の前は半角スペースを入れます。「sid」は「Submission ID」のことで、webform「注文入力」内ではユニークな整数です。「[webform\_submission:sid]」は、Tokenで「『webformで発信するsid』を表示しなさい」という意味です。これに、ひとつの注文のうちの1番目なので、「(半角スペース)+1」を追記します。2～5番目もそれぞれの番号を振ります。【これは、くれぐれも注意してください！】

「☐ 投稿者 (uid)」にチェックを入れ、[-選択- ▼] をクリックして「Submitted by (uid) - entity\_reference」を選択。

→ [保存]。

最終的には、「Webform Content Creator の一覧表」は図 15 のようになります。

【図 15：Webform Content Creator の一覧表】

Webform Content Creator ☆				
<a href="#">ホーム</a> » <a href="#">管理</a> » <a href="#">環境設定</a>				
✓ Saved the dishorder2order5 entity.				
<a href="#">+ Add configuration</a>				
タイトル	WEBFORM	エンティティタイプ	バンドル	操作
dishorder2order1 (dishorder2order1)	注文入力 (dish_order)	コンテンツ (node)	注文 (order)	<a href="#">フィールドの管理</a> ▼
dishorder2order2 (dishorder2order2)	注文入力 (dish_order)	コンテンツ (node)	注文 (order)	<a href="#">フィールドの管理</a> ▼
dishorder2order3 (dishorder2order3)	注文入力 (dish_order)	コンテンツ (node)	注文 (order)	<a href="#">フィールドの管理</a> ▼
dishorder2order4 (dishorder2order4)	注文入力 (dish_order)	コンテンツ (node)	注文 (order)	<a href="#">フィールドの管理</a> ▼
dishorder2order5 (dishorder2order5)	注文入力 (dish_order)	コンテンツ (node)	注文 (order)	<a href="#">フィールドの管理</a> ▼

#### (6)-4 注文ノート「ビュー」表示

注文入力の「ビュー」を作ります。

操作はほとんど「仕入先ノート」の場合と同じです。

「ビューの名前」は「注文ノート」、システム内部名称は「order\_note」、ビューの設定では、タイプ指定を「注文」とします。

「☐ ページを作成する」にチェック → 「Page title」はそのまま、「パス」は「order\_note」とします。

「ページの表示設定」では「ディスプレイフォーマット」を「テーブル」に変えます。

「表示件数」のデフォルト「10」は消去し、「☐ ページャーを使用する」のチェックを外し、「メニューリンクを作成」にチェックを入れます。出てきた「メニュー」欄を「Main navigation」に変更し、「リンクテキスト」欄はそのまま → [保存して編集] → ビュー「注文ノート」編集画面になります。

フィールドを編集・追加します。

「タイトル」の表示は「タイトル」から「注文 ID」に変えます。

追加フィールド「注文日時」は、「タイムゾーンオーバーライド」を「Tokyo」に、「日付の書式」を「デフォルトの短い日付」に設定し、[Apply]（または「適用」）。以下同様）します。

「注文 ID」と「注文日時」を並べ変え、「注文日時」を先に持って来ます。

次のフィールドは「料理名」にしますが、先にすることがあります。

「料理名」は既に「field\_dish\_name」として設定していますが、これはコンテンツタイプ「注文」には入っていないので、「注文料理 (field\_order\_dish)」から参照されているコンテンツ（＝「料理」）に在るので、そちらを参照するように「関係」を設定しておきます。

編集画面右の「▶ 高度」の▶をクリックして「リレーションシップ」の項を表示させます。→ [追加] → 「☐ field\_order\_dish から参照されているコンテンツ」にチェック → [リレーションシップを追加して設定] → [適用] します。すると、編集画面の「リレーションシップ」項に「field\_order\_dish: コンテンツ」が入ります。

それから追加フィールドに「料理名」を選択します。「Relationship」を「field\_order\_dish: コンテンツ」にして → [Apply]。

追加フィールド「注文数」及び「座席 No.」は、リレーションシップを使用せず、素直にそのまま設定します。

「フィルターの条件」を追加します。

「☐ 注文料理 (field\_order\_dish)」にチェック → [フィルター条件を追加して設定] → 「オペレータ」欄は「空でない (NOT NULL)」に変更 → [Apply]。

「☐ 注文数 (field\_order\_quantity)」も同じようにします。

これらのフィルターにより、ビュー「注文ノート」では注文料理が空欄の場合や注文数が空欄の場合には、注文を表示させないことができます。

「検索用のフィルター」を追加します。

特定の注文 ID、料理名、座席 No.で検索できるようにします。

「特定の注文 ID」は、「フィルターの条件」の [追加] → 「☐ タイトル」をチェック → [フィルター条件を追加して設定] → 「☐ このフィルターを訪問者へ表示し、変更できるようにする」にチェック → 「ラベル」欄は「注文 ID」、「オペレータ」欄は「正規表現」を選択 → [Apply]。

「料理名」は、「フィルターの条件」の [追加] → 「☐ 料理名(field\_dish\_name)」をチェック → [フィルター条件を追加して設定] → 「☐ このフィルターを訪問者へ表示し、変更できるようにする」にチェック → 「ラベル」欄は「料理名」、「オペレータ」欄は「含む」、「Relationship」欄は「field\_order\_dish: コンテンツ」とし → [Apply]。

「座席 No.」は、「フィルターの条件」の [追加] → 「☐ 座席 No.(field\_orderer)」をチェック → [フィルター条件を追加して設定] → 「☐ このフィルターを訪問者へ表示し、変更できるようにする」にチェック → 「ラベル」欄は「座席 No.」 → [Apply]。

「並び替え基準」ですが、「コンテンツ: 投稿日時 (降順)」と「コンテンツ: 注文料理 (昇順)」を設定します。

以上でビュー「注文ノート」の設定はとりあえず終わりです。

予め「メニュー&注文」で入力しておいたデータが、プレビューで例えば図 16 のようになることを確認します。



【図 16：ビュー「注文ノート」のプレビュー画面】

タイトル 注文ノート				
外部設置フィルター				
注文ID	料理名	座席No.		
<input type="text"/>	<input type="text"/>	<input type="text"/>		
<input type="button" value="適用"/>				
コンテンツ				
注文日時	注文ID	料理名	注文数	座席NO.
2021/05/06 - 21:30	94 1	ギョーザ	3	95
2021/05/06 - 21:30	94 2	ラーメン	4	95
2021/05/06 - 19:05	90 1	ギョーザ	5	91
2021/05/06 - 19:05	90 2	チャーハン	2	91
2021/05/06 - 19:05	90 3	野菜炒め	3	91

#### (6)-4-1 注文ノートへの「すみ」(調理済 or 配膳済)表示

少しチャレンジングな欲が出てきました。

調理が終わった（または配膳した）料理には、「すみ」を表示するよう機能追加します。

Drupal では、一旦作ったコンテンツを編集画面以外で編集することはなかなかできないようです。例えば、コンテンツタイプ「注文」にプレーンテキストのフィールド「調理済？」を追加しておき、調理終了時にそこに「すみ」とか入力する方法を管理画面以外では見つけることができませんでした。（どこかにモジュールがあるかもしれませんが。）

そこで、「コメント」入力を利用することにしました。

管理メニュー [サイト構築] → [コメントタイプ] → [+ コメントタイプの追加] → ラベル欄に「すみ？」、システム内部名称欄に「sumiqu」、対象エンティティタイプ欄は「コンテンツ」 → [保存]。

コメントタイプ「すみ？」の [フィールドの管理] → ラベル「Comment」の [編集▼] の「▼」をクリックして「Delete」を選択 → [Delete]。

管理メニュー [サイト構築] → [コンテンツタイプ] → 「注文」の [フィールドの管理] → [+フィールドの追加] → 「新しいフィールドの追加」欄は「コメント」を選択、ラベルは「調理済？」、システム内部名称は「(field\_)sumi9」とします (9 は qu の代わりです)。 → [保存して次へ] → 「コメントタイプ」欄は「すみ？」を選択 → [フィールド設定を保存] → 「スレッド形式」のチェックをはずし、「1 ページあたりのコメント数」を 1 に、「コメントへの返信フォームを、コメントと同じページ内に表示する」のチェックをはずし → [設定の保存]。

これでコメントフィールドの追加ができました。

ビューに加えて表示させましょう。

管理メニュー [サイト構築] → [ビュー] → 「注文ノート」が「無効」になっていれば、その行の右端の [有効] をクリック → 「注文ノート」の [編集] をクリック → 「フィール



ド」に「壊れている、もしくは消失しているハンドラー」があれば、削除（その文言をクリックして「削除」）→「フィールド」を「追加」→「調理済？」にチェックを入れ → 「フィールドを追加して設定」 → 「結果の書き換え」 → 「□このフィールドの出力を、指定の文字列で上書き」にチェックを入れ → 「テキスト」欄に「スミ」と入力 → [Apply] → ビューを「保存」。

「サイトへ戻る」→「注文ノート」で、「注文ノート」が表示され、表の右端に「調理済？」というカラムができています。

ここに「スミ」と表示させるには、該当する注文 ID をクリックします。→ その注文 ID の情報内容が表示され、その中に青字で「コメントを追加」と書かれたところがあります。これをクリック → 「コメントを追加」画面の下の方の緑色「✓保存」ボタンをクリック → 「調理済？」や「(件名なし)」とかの表示が出ますが何もせず、メニュー「注文ノート」をクリック。

これで、例えば図 17 のような注文ノートが表示されます。

(注文 ID: 90 2 と 90 3 を「スミ」にしました。)

マウスクリックのみ (4 ポチですが) で「(調理) スミ」が表示できるようになりました。

【図 17: 「調理済？」カラムを加えた「注文ノート」】

丸丸食堂						
ホーム		仕入先ノート	食材ノート	料理ノート	メニュー&注文	注文ノート
					アカウント情報	ログアウト
注文ノート						
注文ID		料理名		座席No.		適用
注文日時	注文ID	料理名	注文数	座席No.	調理済?	
2021/05/06 - 21:30	94 1	ギョーザ	3	95		
2021/05/06 - 21:30	94 2	ラーメン	4	95		
2021/05/06 - 19:05	90 1	ギョーザ	5	91		
2021/05/06 - 19:05	90 2	チャーハン	2	91	スミ	
2021/05/06 - 19:05	90 3	野菜炒め	3	91	スミ	

## (7) 食材仕入れのためのデータ処理

これから、例えば図 17 の座席 No.95 のような注文（注文 ID: 94 1 と 94 2）があったとき、「これらの調理にはどんな食材が合計でどのくらい必要で、それらをどこの仕入先に発注すれば良いか？また、その費用はいくらか？」という計算処理を考えます。

「注文→即調理」とは違い、「注文→食材仕入れ→調理」なので、仕出し屋さんが数日先の宴会の料理予約を受けた場合など、とお考えください。

注文 ID: 94 1 は料理 ID: d01 のギョーザを 3 つ、注文 ID: 94 2 は料理 ID: d02 のラーメン 4 つです。

普通の思考の流れは以下のようでしょう。

- ① 「注文ノート」を見て、対象となる注文 ID が 94 1 と 94 2 であることを知る。
- ② 注文 ID: 94 1 は料理 ID: d01 が 3 人前。

- ③ 「料理ノート」を見て、料理 ID: d01 を 1 人前作るには、食材 f001 が 2、f003 が 4、f005 が 6 必要。
- ④ それらが 3 人前では、食材 f001 が  $2 \times 3 = 6$ 、f003 が 12、f005 が 18 必要。
- ⑤ 同様に、注文 ID: 94 2 は料理 ID: d02 が 4 人前。
- ⑥ 料理 ID: d02 を 1 人前作るには、食材 f003 が 3、f005 が 6、f007 が 9 必要。
- ⑦ それらが 4 人前では、食材 f003 が 12、f005 が 24、f007 が 36 必要。
- ⑧ ④と⑦から食材合計量は、f001 は 6、f003 は  $12 + 12 = 24$ 、f005 は 18、f007 は 36。
- ⑨ 「食材ノート」を見て、各食材費=食材単価×食材合計量だから、f001 は  $36 \times 6 = 216$  円、f003 は 1152 円、f005 は 4608 円、f007 は 2772 円。
- ⑩ 食材 f001 の仕入先は仕入先 ID: s01 の「八百屋お七」。
- ⑪ 「仕入先ノート」を見て、「八百屋お七」宛に以下の注文書を FAX 送信。  
「ニンジン（食材 ID: f001 の食材名）600g(単位量が 100g なので)を売ってください。費用は 216 円。」
- ⑫ 他の食材についても⑧以下同様な計算をして発注書を作成し、FAX 送信。

このように、普通のやり方では、各ノートを逐次的に参照することになります。

でも、Drupal でのやり方はちょっと違います。

Drupal の「ビュー」機能は、コンテンツやその参照先情報をすべて 1 行にして一覧表示できるので、これを利用しない手はありません。

要するに、必要な全ての情報をビューで一括して引き出しておき、それを「JavaScript」や「CSS」といった言語で改造します。

必要なすべての情報を盛り込んだビューを「仕入れノート」と名付けます。

データ処理では一旦「仕入れノート」を表示させ、それから Asset Injector の JavaScript で構築するスクリプトによって「仕入れ計算」をし、JavaScript と CSS によって「発注書発行」を行います。

### (7)-1 ビュー「仕入れノート」の作成

座席 No.95 の注文について必要な情報を「各注文 ID の食材 ID ごと」に表示すると、表 3 のようになります。

料理 ID は「料理ノート」を、食材 ID は「食材ノート」を、仕入先 ID は「仕入先ノート」をそれぞれ参照するために必要なもので、入れています。ここでは、「数量」は 1 食あたりの食材量 (×100g)、「単価」は 100g あたりの食材費 (円) です。

【表 3：図 17 における座席 No.95 の注文の食材仕入れに必要な情報】

注文 ID	料理 ID	注文数	食材 ID	食材名	数量	単価	仕入先 ID	仕入先名	FAX
94 1	d01	3	f001	玉葱	2	36	s01	八百屋お七	03-3502-8111
94 1	d01	3	f003	人参	4	48	s01	八百屋お七	03-3502-8111
94 1	d01	3	f005	タコ足	6	256	s02	蛸八鮮魚店	04-4613-9000
94 2	d02	4	f003	人参	3	48	s01	八百屋お七	03-3502-8111
94 2	d02	4	f005	タコ足	6	256	s02	蛸八鮮魚店	04-4613-9000
94 2	d02	4	f007	鶏モモ	9	77	s03	山猫精肉店	05-6789-0123

まず、必要な情報をすべて盛り込んだビューを作成します。

ベースは「注文ノート」ですが、別のビュー名にして改造します。

管理メニュー [サイト構築] → [ビュー] → 「注文ノート」の [編集 ▼] の「▼」をクリックして「複製」を選択 → 「ビューの名前」は「仕入れノート」とします。システム内部名称を「purchase\_note」にして → [複製]。

ビュー管理画面で、まず「タイトル」を「注文ノート」から「仕入れノート」へ、「パス」を「/order\_note」から「/purchase\_note」へ、「メニュー」名を「注文ノート」から「仕入れノート」へ変更します。

ビューの [保存] → [サイトへ戻る] で、どうなったか確認します。

メニュー「仕入れノート」が「注文ノート」の左にあれば、[サイト構築] → [メニュー] → 「Main navigation」の [メニューの編集] で、「Main Navigation」中のメニュー「仕入れノート」が右端に来るように順番を入れ替えます。

「仕入れノート」の画面で、検索欄の行の右端にカーソルをゆっくり持って行くと [丸にペン] のマークが幽霊のように現れます。これをクリックすると、[ビューを編集] という文言が現れ、これをクリックすると、ビュー編集画面に変わります。

表3の情報を表示するようにフィールドを改変します。

フィールド「コンテンツ: 調理済?」を削除します。

「料理 ID」を追加します。これはコンテンツタイプ「注文」では「注文料理」というタイトル (ラベル) でしたが、このラベルは「料理 ID」に変更します。フィールド順を並び替えて「料理名」の前に持って来ます。

「座席 No.」というラベル名は「予約」に変更し、並びは「注文 ID」の前に持って来ます。「フィルターの条件」のラベル名も「予約」に変更し、「オペレータ」を「正規表現」にして [Apply] し、並びを「タイトル」の前に持って来ます。とりあえずここで [保存] し、[サイトへ戻る] って「メニュー&注文」画面を開きます。

「注文入力」の「座席 No.」欄に、例えば「6/10 12:00 帝国ホテル/鳳の間」とか入力し、適当に料理を注文します。

また、今度は「6/12 14:00 砂防会館/987 号室」とかの「座席 No.」で、料理を注文します。

「仕入れノート」を開くと、図18のような一覧表が表示されます。

【図18: 予約の入った「仕入れノート」事例】

仕入れノート					
予約	<input type="text"/>	注文ID	<input type="text"/>	料理名	<input type="text"/>
					<input type="button" value="適用"/>
注文日時	予約	注文ID	料理ID	料理名	注文数
2021/05/09 - 21:58	6/12 14:00砂防会館/987号室	98 1	d02	ラーメン	16
2021/05/09 - 21:58	6/12 14:00砂防会館/987号室	98 2	d05	野菜炒め	16
2021/05/09 - 21:46	6/10 12:00帝国ホテル/鳳の間	97 1	d01	ギョーザ	50
2021/05/09 - 21:46	6/10 12:00帝国ホテル/鳳の間	97 2	d02	ラーメン	30
2021/05/09 - 21:46	6/10 12:00帝国ホテル/鳳の間	97 3	d03	チャーハン	20
2021/05/09 - 21:46	6/10 12:00帝国ホテル/鳳の間	97 4	d05	野菜炒め	40
2021/05/06 - 21:30	95	94 1	d01	ギョーザ	3
2021/05/06 - 21:30	95	94 2	d02	ラーメン	4
2021/05/06 - 19:05	91	90 1	d01	ギョーザ	5
2021/05/06 - 19:05	91	90 2	d03	チャーハン	2
2021/05/06 - 19:05	91	90 3	d05	野菜炒め	3

ちなみに、「6/10 と 6/12 の予約用の食材をまとめて仕入れたい」と考えたします。  
それらの予約だけの表示をするには、「予約」検索欄（フィルター）に「6/1[0-2]」と入力して「適用」します。これは「6/1」という文字列と、それに続く 1 文字が 0 から 2 の間にあるような 4 文字の文字列を表現していて、それに合致しているものを含んでいれば表示してくれます。なお、「砂防会館」だけの予約なら「砂」などといった部分文字列でも引っ掛けてくれます。

次に「食材 ID」、及び「数量」のフィールドを追加しますが、その前に、煩雑さを避けるために注文 ID が 97～（帝国ホテル分）と 98～（砂防会館分）のコンテンツを消去します。

管理メニュー「コンテンツ」→「タイトル」が 97・・と 98・・の行の左の□にチェックを入れ、「Action」は「コンテンツを削除」で、「選択したアイテムに適用」→「delete」してください。

ビュー「仕入れノート」の編集画面に戻り、フィールドの追加をします。

表 3 の「食材 ID (field\_dish\_foodstuff)」は「注文料理 (field\_order\_dish)」から参照されるコンテンツ（＝「料理」）に入っているなので、Relationship は「field\_order\_dish: コンテンツ」とします。

「▶ 複数フィールドの設定」では「✓ すべての参照先を一行で表示」のチェックを外します。（これで食材 ID ごとに別の行で表示されます）→「Apply」。

念のため、「並び替え基準」に「食材 ID」を昇順で「追加」します（リレーションシップは「field\_order\_dish: コンテンツ」を選択）。

フィールドの追加に戻りますが、「食材量(1 皿分)」も「食材 ID」と同じコンテンツにあるので、Relationship を「field\_order\_dish: コンテンツ」として「Apply」します。なお、ビューでは、同一行に表示されるので、後で JavaScript で「1 行 1 食材量」に修正します。

続けて「食材名」と「百 g 単価（食材 100g あたり単価（円））」のフィールドを追加します。

「食材名」は、コンテンツタイプ「食材 (foodstuff)」にあり、コンテンツタイプ「料理 (dish)」のフィールド「field\_dish\_foodstuff (食材 ID)」から参照されるので、先に「リレーションシップ」を追加します。「▶ 高度」→「リレーションシップ」の「追加」→「□ field\_dish\_foodstuff から参照されているコンテンツ」にチェック → 「リレーションシップを追加して設定」→ Relationship は「field\_order\_dish: コンテンツ」→「Apply」とします。追加されたリレーションシップは「(field\_order\_dish: コンテンツ) field\_dish\_foodstuff: コンテンツ」となります。

「食材名」のフィールド追加は、「フィールド」欄「追加」→「□ 食材名」にチェックを入れ → 「フィールドを追加して設定」→ Relationship は「field\_dish\_foodstuff: コンテンツ」を選択、ラベルは「食材名」→「Apply」とします。

「百 g 単価」も「食材名」同じく、Relationship で「field\_dish\_foodstuff: コンテンツ」を選択し → 「Apply」とします。

次に、「店 ID」、「仕入先名」、及び「FAX」のフィールドを追加します。

「店 ID」は、コンテンツタイプ「仕入先 (supplier)」にあり、コンテンツタイプ「食材 (foodstuff)」のフィールド「foodstuff\_supplier (店 ID)」から参照されるので、先に「リレーションシップ」として「☐ field\_foodstuff\_supplier から参照されるコンテンツ」にチェックを入れ → [リレーションシップを追加して設定] → Relationship は「field\_dish\_foodstuff: コンテンツ」 → [Apply] としておきます。追加されたリレーションシップは「(field\_dish\_foodstuff: コンテンツ) field\_foodstuff\_supplier: コンテンツ」です。

「店 ID」のフィールド追加は、「フィールド」欄 [追加] → 「☐ 仕入先」にチェックを入れ → [フィールドを追加して設定] → Relationship は「field\_dish\_foodstuff: コンテンツ」を選択、ラベルは「店 ID」 → [Apply] とします。

「仕入先名」は、「仕入先名」を選択し → [フィールドを追加して設定] → Relationship は「field\_foodstuff\_supplier: コンテンツ」 → [Apply]。

「FAX」も同様に、「FAX」を選択し → [フィールドを追加して設定] → Relationship は「field\_foodstuff\_supplier: コンテンツ」 → [Apply]。

これで情報収集は終わりですが、表示スペース節約のため、「注文日時」は時間表示をやめ、日付だけにします。フィールド編集で、[コンテンツ: 注文日時 (注文日時)] をクリック → 「日付の書式」で「HTML 年無し日付」を選択し [Apply] してください。

「フィルターの条件」を追加します。

「☐ 食材 ID」をチェック → [適用] → リレーションシップは「field\_order\_dish: コンテンツ」、オペレータは「空でない (NOT NULL)」 → [適用]。この順番を「コンテンツ: 注文数 (空ではない)」の次に上げます。

ここまでで、図 19 の一覧表ができます。[保存] してメニュー「仕入れノート」に戻ってみてください。

この表は、「予約」 and/or 「注文 ID」 and/or 「料理名」で検索して必要情報を絞り、発注書を作成する材料となるものです。

【図 19: 「仕入れノート」のビュー (一部)】

仕入れノート												
予約 <input type="text"/>		注文ID <input type="text"/>		料理名 <input type="text"/>		<input type="button" value="適用"/>						
注文日時	予約	注文ID	料理ID	料理名	注文数	食材ID	食材名	食材量	材単価	店ID	仕入先名	FAX
05-06	95	94 1	d01	ギョーザ	3	f001	玉葱	2, 4, 6	36	s01	八百屋お七	03-3502-8112
05-06	95	94 1	d01	ギョーザ	3	f003	人参	2, 4, 6	48	s01	八百屋お七	03-3502-8112
05-06	95	94 1	d01	ギョーザ	3	f005	タコ足	2, 4, 6	256	s02	タコ八鮮魚店	04-4613-9001
05-06	95	94 2	d02	ラーメン	4	f003	人参	3, 6, 9	48	s01	八百屋お七	03-3502-8112
05-06	95	94 2	d02	ラーメン	4	f005	タコ足	3, 6, 9	256	s02	タコ八鮮魚店	04-4613-9001
05-06	95	94 2	d02	ラーメン	4	f007	鶏モモ	3, 6, 9	77	s03	山ネコ精肉店	05-6789-0124
05-06	91	90 1	d01	ギョーザ	5	f001	玉葱	2, 4, 6	36	s01	八百屋お七	03-3502-8112
05-06	91	90 1	d01	ギョーザ	5	f003	人参	2, 4, 6	48	s01	八百屋お七	03-3502-8112

## (7)-2 ビュー「仕入れノート」を JavaScript で改造する

食材の発注に必要な全ての情報をビューで一括して引き出しました。

今度はそれを「JavaScript」で改造します。JavaScript については別のところで適宜勉強してください。

それにしても、「仕入れノート」ビューのどこをどのように触ればよいのでしょうか？  
ビュー表示画面の「ソースコード」を開いてそれを調べます。

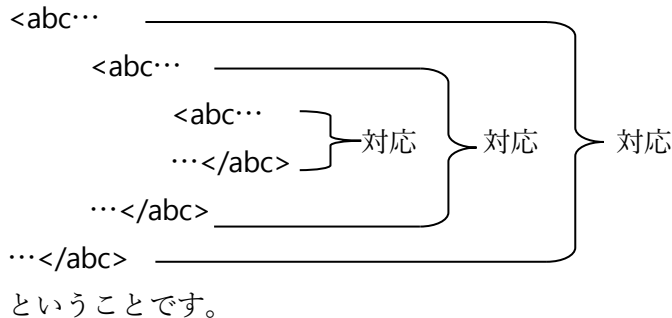
#### (7)-2-1 JavaScript で改造する前に、ビューの「ソースコード(HTML)」を調べる

メインメニューから「仕入れノート」を表示し、ブラウザ・メニューの [開発] → [ページのソースを表示]、または [表示] → [開発/管理] → [ソースを表示] (または [要素の検証]) で、ブラウザの舞台裏 (ソースコード) が表示されます。気持ち悪くなったら、[X] をクリックすると消えますからご安心を。

我慢して 160~200 行目あたりを見ていると、馴染みのある「注文日時」とか「予約」などといった言葉が出てきます (図 20)。ブラウザが safari の場合、▶を手当たり次第クリックして中身を表示させて探してください。

この中で、「<th>」から始まり、対応する「/th>」で終わっている一連の文字列は、「要素 (エレメント)」と呼ぶことになっています。「<th>」は「<abc>」でも「<efg>」でも何でも構いません。それに対応する「</abc>」や「</efg>」までが 1 要素です。

「対応する」というのは、



【図 20: 「仕入れノート」のソースコードの一部 (「注文日時」付近)】

```
155 <div class="view-content">
156 <div class="table-responsive">
157 <table class="table table-hover table-striped">
158 <thead>
159 <tr>
160 <th id="view-field-order-time-table-column" class="views-field views-field-field-order-time" scope="col">注文日時</th>
161 <th id="view-field-orderer-table-column" class="views-field views-field-field-orderer" scope="col">予約</th>
162 <th id="view-title-table-column" class="views-field views-field-title" scope="col">注文ID</th>
163 <th id="view-field-order-dish-table-column" class="views-field views-field-field-order-dish" scope="col">料理ID</th>
164 <th id="view-field-dish-name-table-column" class="views-field views-field-field-dish-name" scope="col">料理名</th>
165 <th id="view-field-order-quantity-table-column" class="views-field views-field-field-order-quantity" scope="col">注文数</th>
166 <th id="view-field-dish-foodstuff-table-column" class="views-field views-field-field-dish-foodstuff" scope="col">食材ID</th>
167 <th id="view-field-foodstuff-name-table-column" class="views-field views-field-field-foodstuff-name" scope="col">食材名</th>
168 <th id="view-field-dish-foodamount-table-column" class="views-field views-field-field-dish-foodamount" scope="col">食材料量</th>
169 <th id="view-field-foodstuff-unitprice-table-column" class="views-field views-field-field-foodstuff-unitprice" scope="col">材単価</th>
170 <th id="view-field-foodstuff-supplier-table-column" class="views-field views-field-field-foodstuff-supplier" scope="col">店ID</th>
171 <th id="view-field-supplier-name-table-column" class="views-field views-field-field-supplier-name" scope="col">仕入先名</th>
172 <th id="view-field-supplier-fax-table-column" class="views-field views-field-field-supplier-fax" scope="col">FAX</th>
173 </tr>
174 </thead>
175
```

図 20 で、「class="..."」の「...」で表示されている、例えば「注文日時」なら「views-field-field-order-time」とか、「予約」なら「views-field-field-orderer」というのは、それぞれの情報が属している「クラス」特有の名前です。これで JavaScript や CSS で改造したことを実行する HTML 上の位置を特定します。なお、「views-field」というのは特有ではないので無視して大丈夫です。

### (7)-2-2 JavaScript を使う準備をする

Drupal で JavaScript を使うには、モジュール「Asset Injector」内の「JS Injector」というツールを用います。Asset Injector は管理メニューの「環境設定」の中にあります。→「Asset Injector」→「JS Injector」→「+ Add Js Injector」→「ラベル」欄は「purchasenotejs」とします。（システム内部名称も同じままです。）

「コード」欄に JavaScript のプログラミングをした後、「条件」は「ページ」を選び → 「ページ」欄にビュー「仕入れノート」のシステム内部名称に「/」を付けて「/purchase\_note」と記述し → 「保存」します。

「コード」欄にとりあえず「abc」とか適当な文字を入れておいて「保存」し、後で「編集」でコード欄に正しく記述しても良いです。

### (7)-2-3 JavaScript でビューのデータを獲得する

「Edit Js Injector」の「コード」欄に記述する内容を以下説明します。

**【要注意】** 以下のスクリプト部分を Js Injector のコード欄にコピーする場合、フォントの都合でクォーテーション・マーク（single, double とともに）が Syntax Error を起こすことがあるので、コピー後、コード欄内でそれぞれのクォーテーション・マークをすべて書き直してください。「書き直す」だけでオッケーです。「左右対称な半角クォーテーション・マーク」なら正しく動きます。

JavaScript では、HTML 文章内のクラス名を指定した以下のコマンドで、そのクラスに属している全ての要素をゴソッと得ることができます。

```
document.getElementsByClassName('クラス名')
```

それらの情報は文字列として JavaScript 文法の中の「配列」に入ります。

例えば以下のようなスクリプトを記述します。「const」は「定数」の意味です。

```
const orderTime = document.getElementsByClassName('views-field-field-order-time')
```

これにより、orderTime[0]には

```
<th id="view-field-order-time-table-column" class="views-field views-field-field-order-time" scope="col">注文日時</th>
```

が、orderTime[1]には、

```
<td headers="view-field-order-time-table-column" class="views-field views-field-field-order-time"><time datetime="2021-05-06T12:30:39Z">05-06</time></td>
```

という文字列が入ります。（黄色マーカーはクラス名。青色は本当に欲しい情報。）

いくつ要素が入ったかは、以下の記述で orderTime に入った要素数から得ます。

```
const orderLength = orderTime.length
```

ビュー「仕入れノート」の一覧表の全ての要素を JavaScript に持って来るには、以下の記述をします。（行末の「// …」は「行内コメント」です。）

```
// ビュー「仕入れノート」から情報を抽出
const orderTime = document.getElementsByClassName('views-field-field-order-time'); // 注文日時
const orderer = document.getElementsByClassName('views-field-field-orderer'); // 予約
const orderID = document.getElementsByClassName('views-field-title'); // 注文 ID
const dishID = document.getElementsByClassName('views-field-field-order-dish'); // 料理 ID
const dishName = document.getElementsByClassName('views-field-field-dish-name'); // 料理名
const dishQuant = document.getElementsByClassName('views-field-field-order-quantity'); // 注文数
```



```

const foodstuffID = document.getElementsByClassName('views-field-field-dish-foodstuff'); // 食材 ID
const foodstuffName = document.getElementsByClassName('views-field-field-foodstuff-name'); // 食材名
const foodamount = document.getElementsByClassName('views-field-field-dish-foodamount'); // 食材量
const foodstuffPrice = document.getElementsByClassName('views-field-field-foodstuff-unitprice'); // 材単価
const supplierID = document.getElementsByClassName('views-field-field-foodstuff-supplier'); // 店 ID
const supplierName = document.getElementsByClassName('views-field-field-supplier-name'); // 仕入先名
const FAX = document.getElementsByClassName('views-field-field-supplier-fax'); // FAX
const orderLength = orderTime.length; // 「注文日時」の数を要素の数とします

```

次に、各要素から、「注文日時」などの項目名や「05-06」といったデータ（値）のみを取り出し、別の配列変数に入れます。

要素からデータのみを切り出すには、要素の入った変数に、以下の波下線部を追記するだけです。

変数 `_.textContent.trim()`

例えば、`orderTime[1].textContent.trim()` と書くと、「05-06」が得られます。

まずデータのみを収納する「配列変数を宣言」します。変数名として、例えば要素を入れた変数名に、全て「array」の「A」を追加した名前を使います。

次に、繰り返し構文によって、データを配列変数に放り込みます。

以上のことを「…GetElementsByClass …」の記述の後に次のように追加します。

「const」は一旦入れたら変更できない定数ですが、「let」は「定数」ではなくコロコロ変わる値の器の宣言です。

// 要素から切り出すデータのみを収容する配列変数を宣言する（A は array の意味）

```

const orderTimeA = []; // 注文日時
const ordererA = []; // 予約
const orderIDA = []; // 注文 ID
const dishIDA = []; // 料理 ID
const dishNameA = []; // 料理名
const dishQuantA = []; // 注文数
const foodstuffIDA = []; // 食材 ID
const foodstuffNameA = []; // 食材名
const foodamountA = []; // 食材量
const foodstuffPriceA = []; // 材単価
const supplierIDA = []; // 店 ID
const supplierNameA = []; // 仕入先名
const FAXA = []; // FAX

```

// 新しい配列変数に、要素からデータを取り出して入れる

```

for(let index = 0; index < orderLength; index++) { // 0 番から「要素数-1」番まで{}内を繰り返し
  orderTimeA[index] = orderTime[index].textContent.trim(); // 注文日時
  ordererA[index] = orderer[index].textContent.trim(); // 予約
  orderIDA[index] = orderID[index].textContent.trim(); // 注文 ID
  dishIDA[index] = dishID[index].textContent.trim(); // 料理 ID
  dishNameA[index] = dishName[index].textContent.trim(); // 料理名
  dishQuantA[index] = dishQuant[index].textContent.trim(); // 注文数
  foodstuffIDA[index] = foodstuffID[index].textContent.trim(); // 食材 ID
  foodstuffNameA[index] = foodstuffName[index].textContent.trim(); // 食材名
  foodamountA[index] = foodamount[index].textContent.trim(); // 食材量
  foodstuffPriceA[index] = foodstuffPrice[index].textContent.trim(); // 材単価
  supplierIDA[index] = supplierID[index].textContent.trim(); // 店 ID
  supplierNameA[index] = supplierName[index].textContent.trim(); // 仕入先名
  FAXA[index] = FAX[index].textContent.trim(); // FAX
} // {}の終わり

```



#### (7)-2-4 ビュー「仕入れノート」の「食材量」を正しく表示させる

図 19 では、「食材量」のカラムに複数のデータがカンマ区切りで表示されています。これは「複数の値を持つフィールド」だからです。

配列変数 `foodamountA[index]` にも、一つの変数に複数のデータが入っています。

例えば最初の 6 個の変数の値は、以下の通りです。

```
foodamountA[0] = 食材量
foodamountA[1] = 2,4,6
foodamountA[2] = 2,4,6
foodamountA[3] = 2,4,6
foodamountA[4] = 3,6,9
foodamountA[5] = 3,6,9
      :           :
```

これらを、

```
foodamountA[0] = 食材量
foodamountA[1] = 2
foodamountA[2] = 4
foodamountA[3] = 6
foodamountA[4] = 3
foodamountA[5] = 6
      :           :
```

のようにデータを個分けすると同時に、表示も同様に一つずつにします。

```
let kokoValue = foodamountA[1].split(",");
```

とすると、`foodamountA[1]` の中のデータが「,」で区切られ、`kokoValue[0]=2`, `kokoValue[1]=4`, `kokoValue[2]=6` の「値入り配列変数」が自動的に生成されます。

`kokoValue` の `[]` の中はゼロ始まりで、0~2 までの 3 つです。なお、発生する配列数は、`kokoValue.length` で分かります。

また、「`.textContent`」の大変ありがたい機能として、「`let foodamount[i].textContent = データ`」とすると、そのデータが `foodamount[i]` の元となるソースコード (HTML) に反映 (画面に表示) されます!! (以下は例です。図 20 参照)

```
<td headers="view-field-dish-foodamount-table-column" class="views-field views-field-field-dish-foodamount">データ</td>
```

結局、「食材量 1 つ表示」のためのスクリプトを以下のように追記します。

```
// カンマ区切りの食材量を一つずつ表示する
for(let index = 1; index < orderLength; index++) { // 1 番から「要素数-1」番まで{…}を繰り返す
  let kokoValue = foodamountA[index].split(","); // カンマ区切りの値を切り出して配列変数に入れる
  for(let ko = 0; ko < kokoValue.length; ko++) { // 切り出した数だけ順に繰り返す
    foodamountA[index] = kokoValue[ko]; // foodamountA[・]に値を 1 個だけ入れる
    foodamount[index].textContent = kokoValue[ko]; // 対応するソースコードに値を 1 個だけ入れる
    index++; // ビューの 1 行分終わったので index を 1 増やす
  } // 内側の for 構文を繰り返す
  index--; // index の値を 1 つ増やしすぎたので 1 つ減らす
} // 外側の for 構文を繰り返す
```

ここまでコードを入力したら、Js Injector を「保存」して、「サイトへ戻」って「仕入れノート」画面を見てみましょう。図 21 のようになっていたら成功です。

【図 21: 「仕入れノート」で「食材量」を各行 1 つずつ表示（一部）】

仕入れノート												
予約		注文ID	料理名		適用							
注文日時	予約	注文ID	料理ID	料理名	注文数	食材ID	食材名	食材量	材単価	店ID	仕入先名	FAX
05-06	95	94 1	d01	ギョーザ	3	f001	玉葱	2	36	s01	八百屋お七	03-3502-8112
05-06	95	94 1	d01	ギョーザ	3	f003	人参	4	48	s01	八百屋お七	03-3502-8112
05-06	95	94 1	d01	ギョーザ	3	f005	タコ足	6	256	s02	タコ八鮮魚店	04-4613-9001
05-06	95	94 2	d02	ラーメン	4	f003	人参	3	48	s01	八百屋お七	03-3502-8112
05-06	95	94 2	d02	ラーメン	4	f005	タコ足	6	256	s02	タコ八鮮魚店	04-4613-9001
05-06	95	94 2	d02	ラーメン	4	f007	鶏モモ	9	77	s03	山ネコ精肉店	05-6789-0124
05-06	91	90 1	d01	ギョーザ	5	f001	玉葱	2	36	s01	八百屋お七	03-3502-8112
05-06	91	90 1	d01	ギョーザ	5	f003	人参	4	48	s01	八百屋お七	03-3502-8112

#### (7)-2-5 JavaScript が上手く動かなかった場合の対処法

JavaScript を一発で動かせる人は「才能アリ」です。私は「才能ナシ」でした。

上記「図 21」がうまく出なかった場合、どうなっているのか 2 つ考えられます。

ひとつは、処理が一応終了し「停止」している場合です。もうひとつは、画面に変化はないのだけれど処理が終了せず無限に何かが「動いている」（ブラウザの上中央の曲がった矢印がぐるぐる回っているとか）場合、またはブラウザ画面が消え、最上段に「修理中なのでまた後で・・・」などの表示が 1 行だけ出る場合です。

後者の場合は、まずブラウザを強制終了し、改めて Drupal の管理画面を立ち上げ、[環境設定] → [パフォーマンス] → [すべてのキャッシュをクリア] してください。その後、[環境設定] → [Asset Injector] → [Js Injector] → で「purchasenotejs」の [編集] をクリックし、コード欄を点検してください。

前者（「停止」している）の場合、ブラウザのメニュー [開発]（safari の場合）または [表示] → [開発/管理]（chrome の場合）を選び、[JavaScript コンソールを表示] します。

JavaScript が原因であれば、コンソールに赤色でエラーメッセージが表示されます。それを読んで対処してください。「Uncaught TypeError:…」と出ていれば、どこかのタイプミスです。その他のエラーの対処法は、別途ネットか何かで「対処療法的に（その場しのぎ的に）」勉強してください。一つのエラーに対処すれば複数のエラーメッセージが消えることがよくあります。なお、Google Chrome では「Unchecked runtime.lastError: The message port closed before a response was received.」というエラーメッセージが出るがありますが、無視して問題ありません。

#### (7)-2-6 食材別に食材量を集計する

JavaScript のコードの中で `orderLength` というのは、ビューの一覧表の行数 + 1（項目名）でした。

集計方法は、最初に「集計欄」をゼロに初期設定しておき、ビューの一覧表の（項目名（ゼロ行目）を除き）1行目から順番に全てのデータ行を処理していき、それが終わると集計欄の値を出す、という流れになります。

まず配列変数 `stuffSum`（食材(`foodstuff`)の合計(`Sum`)という意味）を宣言します。

```
let stuffSum = {};
```

食材は何か出てくるか分からないので、連番にしなくて良い「連想配列」を利用します。連想配列を宣言する場合は、「`= {}`」を付けて宣言しなければなりません。

次に `for` 構文で、一覧表のデータ1行目から「`orderLength - 1`」行目まで、以下を繰り返します。

「店 ID」を読み（`= m` とします）、初出か既出かを判断します。

初出なら、その店 ID の集計欄を新設します。こんな具合です。

```
stuffSum[m] = {};
```

 // 「m」の店について記憶域を新設

「食材 ID」を読み（`= s` とします）、初出か既出かを判断します。

初出なら、その食材の集計欄を `stuffSum[m]` の中に新設します。食材は店が決まっているので、こんな具合になります。

```
stuffSum[m][s] = {};
```

 // 「m」の店の食材「s」について記憶域を新設

また、そこに初期値としてゼロを入れておきます。

```
stuffSum[m][s] = 0;
```

`if` 文のカッコの中の「`!`」は「否定」を示し、「`if(!stuffSum[supID])`」は「`stuffSum[supID]`が存在しなければ」（= 初出なら）という意味になり、その通りであれば続く `{}` の中を実行します。

結局、以下のようなスクリプトになります。（コピペはちょっと後で。）

```
// 食材別に食材量を集計する
let stuffSum = {}; // 連想配列の宣言
for (let index = 1; index < orderLength; index++) { // データ行分だけ繰り返し
  let supID = supplierIDA[index]; // 仕入先 ID を supID へ代入
  let stufID = foodstuffIDA[index]; // 食材 ID を stufID へ代入
  if(!stuffSum[supID]) { // もし、店 ID が初出なら
    stuffSum[supID] = {}; // その店 ID の集計欄を新設
  } // そうでなければ、前行は実施せず
  if(!stuffSum[supID][stufID]) { // もし、食材 ID が初出なら
    stuffSum[supID][stufID] = {}; // その食材 ID の集計欄を新設
    stuffSum[supID][stufID] = 0; // その食材 ID の集計欄の初期値設定
  } // そうでなければ、前 2 行は実施せず
  stuffSum[supID][stufID] += dishQuantA[index] * foodamountA[index];
} // 前行で集計欄に注文数×食材量を加える。
```

これで、使うすべての食材について、各合計量が算出できます。

ただ、食材をしかるべき店に発注する場合、食材合計量だけでなく、その店の店名、FAX 番号、食材名、食材単価、食材ごとの料金の表示が必要です。

これらの情報も簡単に取り出せるようにしておきます。

店関連情報は「`supplier`」、食材関連情報は「`foodstuff`」という連想配列変数名にします。

```
let supplier = {};  
let foodstuff = {};
```

これらの変数は、stuffSum の「コバンザメ」のような扱いで情報を入れると良いです。それらの処理を、先程の「食材別に食材量を集計する」スクリプトブロックに青字で盛り込み、以下に示します。

```
// 食材別に食材量を集計する  
let stuffSum = {};  
let supplier = {};  
let foodstuff = {};  
for (let index = 1; index < orderLength; index++) {  
  let supID = supplierIDA[index];  
  let stufID = foodstuffIDA[index];  
  if(!stuffSum[supID]) {  
    stuffSum[supID] = {};  
    supplier[supID] = {};  
    supplier[supID]['name'] = supplierNameA[index];  
    supplier[supID]['FAX'] = FAXA[index];  
    foodstuff[supID] = {};  
  }  
  if(!stuffSum[supID][stufID]) {  
    stuffSum[supID][stufID] = {};  
    stuffSum[supID][stufID] = 0;  
    foodstuff[supID][stufID] = {};  
    foodstuff[supID][stufID]['name'] = foodstuffNameA[index];  
    foodstuff[supID][stufID]['price'] = foodstuffPriceA[index];  
  }  
  stuffSum[supID][stufID] += dishQuantA[index] * foodamountA[index];  
}  
// 食材量集計の連想配列の宣言  
// 仕入先情報用連想配列の宣言  
// 食材情報用連想配列の宣言  
// データ行分だけ繰り返し  
// 仕入先 ID を supID へ代入  
// 食材 ID を stufID へ代入  
// もし、店 ID が初出なら  
// その店 ID の集計欄を新設  
// 仕入先別情報欄を新設  
// 仕入先名を代入  
// FAX 番号を代入  
// 仕入先別情報欄を新設  
// そうでなければ、前行は実施せず  
// もし、食材 ID が初出なら  
// その食材 ID の集計欄を新設  
// その食材 ID の集計欄の初期値設定  
// その食材の情報欄を新設  
// 食材情報欄に食材名を代入  
// 食材情報欄に単価を代入  
// そうでなければ、前 2 行は実施せず  
// 前行で集計欄に注文数×食材量を加える。
```

最後に、途中経過を JavaScript のコンソールで確認するために、

```
console.log(stuffSum);  
console.log(supplier);  
console.log(foodstuff);
```

を書き加えておきます。

これで Js Injector を [保存] し、[サイトに戻る] → [仕入れノート] → ブラウザの [開発] → [JavaScript コンソールを表示] で、エラーがなければ図 22 のような表示になるはずです。

【図 22：JavaScript コンソール表示/ 仕入先毎に食材量、店名・FAX、食材名・単価】



## (7)-2-7 JavaScript から HTML への書き込み

情報がそろったので、各仕入先へ発注書を発行します。

ローテクで恐縮ですが、先方の ICT 技術にも留意し、今回は発注書をプリントアウトし、別途、人が FAX するという方針やっています。

「仕入先ノート」は、既に必要情報の一覧表（ビュー）を表示しています。

その一覧表に後続して、「発注書」を画面に書き込んで行きます。

「ビューに後続して書き込む」には、ビュー（図 20 に出てくるクラス名="view-content"）の範囲（「<div …」から、対応する「</div>」まで）の、「その後に書き込む」というスクリプトの記述が必要です。

ビューの範囲を取得するには、以下の文を用います。

```
const view = document.querySelector('.view-content'); … (1)
```

「'.view-content'」のピリオドは「クラス」を意味します。

書き込む内容は HTML 要素です。例えばクラス「myAddContent」を設定します。

以下の要素を変数に入れます。

```
let temp = '<div class = "myAddContent"></div>'; …… (2)
```

この要素を、画面を表示する HTML 文章に入れるには、

```
view.insertAdjacentHTML('afterend',temp); …… (3)
```

とします。これは、「temp 内容を、view の終わりの直後に、挿入せよ」という意味です。

表示画面のソースコードは、図 23 のように青字行が追加されます。

【図 23：ソースコード作成途中】

```
画面設定記述・・・
<div class = "view-content">
    ビューの一覧表など・・・
</div>  // "view-content"の終わり
<div class = "myAddContent"></div>
      :
```

'afterend' の代わりに 'beforeend' とすると、「終わり（対応する</div>）の直前に」挿入することになり、JavaScript で HTML 要素を次から次と書き重ねることができるようになります。

例えば、クラス「myAddContent」の中に「発注書発行」の宣言文を入れてみましょう。

まず、クラス名="myAddContent"の範囲を取得しておきます。

```
const myAdd = document.querySelector('.myAddContent');
```

表示したい文字列を HTML 的に書き、temp に代入します。

```
temp = '<div id = "declaration">==== 発注書発行 =====</div>';
```

id の設定は、後で CSS で文字を装飾するためです。

以下のように書くと、temp の内容が画面に表示されます。

```
myAdd.insertAdjacentHTML('beforeend',temp);
```

まとめると、次のような記述になります。

```
// ここから「発注書作成」です
const view = document.querySelector('.view-content');
let temp = '<div class = "myAddContent"></div>';
view.insertAdjacentHTML('afterend',temp);
const myAdd = document.querySelector('.myAddContent');
temp = '<div id = "declaration">===== 発注書発行 =====</div>';
myAdd.insertAdjacentHTML('beforeend',temp);
```

[保存] 後 → [サイトへ戻] ってメニュー [仕入れノート] を見ると、画面の左下に図 24 の表示が出ます。

文字を大きくしたり中央に移動したりといった修飾は、後で CSS インジェクターで行います。

【図 24：「仕入れノート」の左下に「発注書発行」を表示】

05-06	91	90 3	d05	野菜炒め
05-06	91	90 3	d05	野菜炒め
===== 発注書発行 =====				
Powered by <a href="#">Drupal</a>				
<a href="#">Contact</a>				

## (7)-2-8 各仕入先への発注「表」部分の計算

一番重要な「表」部分を計算します。

仕入先ごとの繰り返しになるので、for 構文を用いますが、必要な情報を連想配列に入れてあるので、index を一つずつ増やしながら繰り返し処理をすることができません。

「for-in 構文」という方法を用います。例えば次の式で、

```
for( let A in B ) { ... }
```

とすると、連想配列 B（正しくは「オブジェクト」一般）の中の、すべての A について繰り返し処理をします。実際に用いてみましょう。

すべての仕入先及びその中のすべての食材について、食材合計量と単価を掛けて、食材ごとの料金を計算するスクリプトは以下ようになります。

```
// 発注表の作成
for(let sID in supplier) {                                // 仕入先ごとの繰り返し
  for ( let fID in stuffSum[sID] ) {                      // 仕入先別の食材ごとの繰り返し
    let fPrice = foodstuff[sID][fID]['price'];           // 材単価
    let fAmount = stuffSum[sID][fID];                    // 食材合計量
    let subtotal = fAmount * fPrice;                      // 小計額計算
  }
}
```

これに仕入先名、FAX 番号、及び食材名を加えるため、以下青字のように追記します。

また、食材の「小計額」まで計算しましたが、その仕入先へのすべての食材分を合わせた

「合計金額(単に「合計」とします)を計算しておくことにします。それを赤字で示します。更に、これらの処理が正しくできているか確認のため、console.log()を緑色で追記します。

```
// 発注書の作成
for(let sID in supplier) { // 仕入先ごとの繰り返し
  let supName = supplier[sID]['name']; // 仕入先名
  let supFAX = supplier[sID]['FAX']; // FAX
  let groundTotal = 0; // 仕入先別合計金額初期設定
  for ( let fID in stuffSum[sID]) { // 仕入先別の食材ごとの繰り返し
    let fName = foodstuff[sID][fID]['name']; // 食材名
    let fPrice = foodstuff[sID][fID]['price']; // 材単価
    let fAmount = stuffSum[sID][fID]; // 食材合計量
    let subtotal = fAmount * fPrice; // 小計額計算
    console.log(supName, supFAX, fName, fPrice, fAmount, subtotal);
    groundTotal += subtotal; // 合計金額へ小計を加算
  }
  console.log(supName, groundTotal);
}
```

これでうまく処理ができているか、[サイトに戻] って [仕入れノート] を開き (既に開いている場合は再度メニューをクリックしてください)、「JavaScript コンソール」を表示してみてください。図 25 のような表示があれば成功です。

【図 25：JavaScript コンソール表示/ 発注表に必要な情報】

八百屋お七	-	"03-3502-8112"	-	"玉葱"	-	"36"	-	16	-	576
八百屋お七	-	"03-3502-8112"	-	"人参"	-	"48"	-	44	-	2112
八百屋お七	-							2688		
タコ八鮮魚店	-	"04-4613-9001"	-	"タコ足"	-	"256"	-	72	-	18432
タコ八鮮魚店	-	"04-4613-9001"	-	"イカゲソ"	-	"128"	-	3	-	384
タコ八鮮魚店	-	"04-4613-9001"	-	"カンパチ"	-	"512"	-	12	-	6144
タコ八鮮魚店	-							24960		
山ネコ精肉店	-	"05-6789-0124"	-	"鶏モモ"	-	"77"	-	50	-	3850
山ネコ精肉店	-	"05-6789-0124"	-	"豚バラ"	-	"222"	-	16	-	3552
山ネコ精肉店	-	"05-6789-0124"	-	"牛ロース"	-	"444"	-	21	-	9324
山ネコ精肉店	-							16726		

これらが表示されず、エラーメッセージが出ていたら、JavaScript の記載確認やクォーテーションの修正等検討してみてください。

うまく行っていれば、JS Injector に戻って console.log 文を削除します。

## (8) 食材購入発注書の作成

### (8)-1 発注書に必要な情報を「とにかく」画面に表示する

発注書を書くために必要な情報を一応用意したので、これらの情報をとにかく画面に表示します。画面表示ができれば、CSS で修飾して「発注書」を完成させます。

発注書の大体の様式イメージは、仕入先ごとに改ページして、第 1 行目の中央に大文字で「発注書」、その左下に仕入先名と FAX 番号、それよりちょっと低くして右側に自社名・住所・連絡先を書き、その下に、食材名、単価、数量、小計の表を載せ、最後に合計額を表示する、という感じです。

これらの情報は既にすべて入手しています。



Js Injector のスクリプトを見やすくするために、関数を使います。

「発注書」は関数「sheetHeader」に、仕入先名・FAX は関数「supOnchu」に、自社名他は関数「myCompany」に、「表」は関数「tableStart」・「tableBody」・「tableEnd」に書かせることにします。

それぞれの関数は、スクリプト「発注書の作成」内の以下の赤字の場所に挿入します。

```
// 発注書の作成
for(let sID in supplier) {                // 仕入先ごとの繰り返し
    sheetHeader();                        // 【関数】発注書のアタマ書き出し
    let supName = supplier[sID]['name'];  // 仕入先名
    let supFAX = supplier[sID]['FAX'];    // FAX
    supOnchu(supName, supFAX);           // 【関数】仕入先御中と FAX 番号
    myCompany();                          // 【関数】自社情報
    tableStart();                         // 【関数】食材注文表のヘッダ
    let groundTotal = 0;                  // 仕入先別合計金額初期設定
    for ( let fID in stuffSum[sID]) {     // 仕入先別の食材ごとの繰り返し
        let fName = foodstuff[sID][fID]['name']; // 食材名
        let fPrice = foodstuff[sID][fID]['price']; // 材単価
        let fAmount = stuffSum[sID][fID];      // 食材合計量
        let subtotal = fAmount * fPrice;       // 小計額計算
        tableBody(fName, fPrice, fAmount, subtotal); // 【関数】食材注文表の本体
        groundTotal += subtotal;              // 合計金額へ小計を加算
    }
    tableEnd(groundTotal);                // 【関数】食材注文表のフッタ
}
}
```

JavaScript における関数の作り方ですが、今回必要なことだけを言うと、以下の通りです。例えば「anyterm」という名前の関数を作ります。

宣言 … function anyterm(引数){やらせたいこと…}

使い方 … 「anyterm(引数);」を、引数が使えるところか適当なところに放り込む。

例えば、「sheetHeader」では、ただ「発注書」と書くだけの機能なので、「引数」は無し（カッコは必要）で、以下ようになります。「+=」は「継ぎ足し」を示す式です。

```
function sheetHeader() {
    temp = '<div class="eachSheet">'; // 最後に「</div>」を忘れずに！！
    temp += '<div id="sheetName">発注書</div>'; // 「発注書」って書きます
    myAdd.insertAdjacentHTML('beforeend',temp); // temp を myAdd の最後の直前に書く
}
```

「<div class="eachSheet"」に対応する「</div>」は、ひとつの仕入先への発注書作成が完結したら付記するので、ここでは付けず、次の仕入先の処理に移る直前に入れます。

supOnchu は、以下のスクリプトとなります。

```
function supOnchu(shop, fax) { // 仕入先名・FAX の変数を引数として入れる
    temp = '<div class="supplier">';
    temp += '<div id="supName">' + shop + '&nbsp;&nbsp;&nbsp;御中<br></div>';
    temp += '<div id="supFAX">FAX:&nbsp;&nbsp;&nbsp;' + fax + '<br></div></div>';
    myAdd.insertAdjacentHTML('beforeend',temp);
}
```

ここでは、文字列「<div id="supName">」と引数 shop と文字列「&nbsp;&nbsp;&nbsp;御中<br></div>」を接続して temp へ継ぎ足しています。FAX も同様です。「&nbsp;&nbsp;&nbsp;」は「スペース」1つの意味です。HTML 文法ではスペースは意味を持たず無視されるので、ブラウザで表示するには、このような特殊な表現が必要です。



myCompany は、以下のように決り文句を書くだけです。

```
function myCompany() {
    temp = '<div class="myCompany">';
    temp += '<div id = "myCompName">マイカンパニー株式会社</div>';
    temp += '<div id = "myPcode">〒160-0021</div>';
    temp += '<div id = "myAddress">東京都新宿区歌舞伎町1丁目1-1</div>';
    temp += '<div id = "myTEL">TEL: 03-3204-7166</div>';
    temp += '<div id = "myFAX">FAX: 03-3204-7167</div></div>';
    myAdd.insertAdjacentHTML('beforeend',temp);
}
```

次はいよいよ「食材注文表」部分の表示です。

HTML では、「<table…から</table>」で「表（テーブル）」を表現します。

```
<table border="1" class="stuffTable"> … </table>
```

と書くと、自動的に、罫線が「1」でクラス名「stuffTable」の「表」が設定されます。

表には「ヘッダ部（<thead…から</thead>まで）」と「本体（<tbody…から</tbody>まで）」と「フッタ部（<tfoot…から</tfoot>まで）」があります。

「<table…から</table>」の中に、これらを記載することになります。

ヘッダ部は何行も書けるので、その各行（たった1行でも）を「<tr (table row) …から</tr>」で設定します。

今回ヘッダ部は単なる項目名を1行横並びに表示するだけです。

ヘッダ部の各マス目は「<th…から</th>」でくくります。

従って、ヘッダ部を表示する関数「tableStart」を次のように記載します。

```
function tableStart() {
    temp = '<table border="1" class="stuffTable">';
    temp += '<thead>'; // ヘッダ部はじまり
    temp += '<tr class="myKoumoku">'; // 項目名行にクラス名「myKoumoku」を設定
    temp += '<th>食材名</th>'; // 第1項目は「食材名」（以下同じ）
    temp += '<th>百 g 単価</th>'; // 第2項目「百 g 単価」
    temp += '<th>数量(×百 g)</th>'; // 第3項目「数量(×百 g)」
    temp += '<th>小計(円)</th>'; // 第4項目「小計」
    temp += '</tr>'; // 項目名行おわり
    temp += '</thead>'; // ヘッダ部おわり
    temp += '<tbody>'; // 次の body 部はじまり。繰り返しの外に置く
}
```

次は、食材注文表の本体部分です。以下のように記載します。

```
function tableBody(name, price, amount, subt) {
    temp += '<tr>'; // 1行書くことを宣言
    temp += '<td>' + name + '</td>'; // 食材名です
    temp += '<td>' + price.toLocaleString() + '</td>'; // 単価を3桁表示
    temp += '<td>' + amount.toLocaleString() + '</td>'; // 数量を3桁表示
    temp += '<td>' + subt.toLocaleString() + '</td>'; // 小計を3桁表示
    temp += '</tr>'; // 1行のおわり
}
```

最後に、食材注文表のおわり部分です。総合計の表示と仕入先別「発注書」のおわり処理も含めたので、少しゴチャつきました。「//」以下のコメントをご参照ください。

```
function tableEnd(total) {
    temp += '</tbody>'; // Body 部おわり
    temp += '<tfoot>'; // フッタ部はじまり
    temp += '<tr><th></th><th></th><td id="goukeina">合計</td>';
}
```

```

temp += '<th id="goukeiti">' + total.toLocaleString() + '</th></tr>';
let addTax = Math.floor(total * 1.08); // total を 1.08 倍し、小数部切り捨て
temp += '<tr><th></th><th></th><td id="zeikomina">合計(消費税込)</td>';
temp += '<th id="zeikomiti">' + addTax.toLocaleString() + '</th></tr>';
temp += '</tfoot>'; // フッタ部おわり
temp += '</table>'; // 食材注文表全体のおわり
temp += '</div>'; // クラス名「eachSheet」のおわり
myAdd.insertAdjacentHTML('beforeend',temp); // 仕入先別「発注書」の書き出し
}

```

以上の Js Injector の記載で、[サイトに戻] ってメニュー [仕入れノート] を開き、ビューによる一覧表の下に、図 26 のような表示が出れば、成功です。

なお、発注書ごとに「改ページ」する関数「kaipage()」を作っておきます。

```

function kaipage() {
    temp = '<div id = "pagebreak"></div>';
    myAdd.insertAdjacentHTML('beforeend',temp);
}

```

です。実際は CSS の処理で改ページを行います。

この関数を「====発注書発行====」表示の直後と、関数「tableEnd」の最後に放り込んでおきます。(対応する CSS スクリプトを作るまで、この関数は働きません。)

【図 26：「発注書」に必要な情報を「とにかく」表示する/「仕入れノート」下側の一部】

05-06	91	90 3	d05	野菜炒め	3	f009	牛ロース	7	444	s03	山ネコ精肉店
-------	----	------	-----	------	---	------	------	---	-----	-----	--------

└==== 発注書発行 =====

発注書

八百屋お七 御中

FAX: 03-3502-8112

マイカンパニー株式会社

〒160-0021

東京都新宿区歌舞伎町 1 丁目 1-1

TEL: 03-3204-7166

FAX: 03-3204-7167

食材名	百g単価	数量(×百g)	小計 (円)
玉葱	36	16	576
人参	48	44	2,112
		合計	2,688
		合計(消費税込)	2,903

発注書

タコ八鮮魚店 御中

FAX: 04-4613-9001

マイカンパニー株式会社

## (8)-2 「CSS」で発注書のスタイルを決める

発注書のスタイル設定を CSS で行います。

管理ツールバー [環境設定] → [Asset Injector] → [CSS インジェクター] → [+ Add Css Injector] と進みます。

ラベル欄は「purchasenotecss」とします。

コード欄には、とりあえず以下のように記述します。

```

#declaration { /* id="declaration" の範囲の文字について */
    font-size: 48px; /* 文字サイズを 48 ピクセルに */
    display: flex; /* 中央に表示するためには以下 3 行必要 */

```

```

    justify-content: center;
    align-items: center;
}

#pagebreak {
    page-break-before : always; /* 改ページします */
}

```

最初の行の「#declaration」は、JavaScript で設定した「発注書発行」のところの id 名です。「#」は「id」を表しています。ちなみに「.」は「クラス」です。

次の「font-size: 48px;」は明白ですね。フォントサイズを 48 ピクセルにする、ということです。なお、「px」とは「ピクセル（画素）」という画面の構成単位のことです。ちなみに 13 インチの MacBook は、1280×800 ピクセルです。

以下、いろいろなパラメータの設定がありますが、自分で勝手に多少数値を変えて実際の表示を見て、遊んでみてください。勉強になります。

その次の 3 行は、とにかく「中央に表示」という意味です。

「#pagebreak」は「改ページ」の実行スクリプトです。JavaScript で、「kaipage();」と書き込んだ箇所で改ページが行われます。

コメントは「/\*」と「\*/」ではさみます。

さて、「条件」の中の「ページ」ですが → ページ欄に「/purchase\_note」と記載します。これはビュー「仕入れノート」のパス名です。 → 「保存」。

「[サイトへ戻]」って「仕入れノート」を開いてみると、一覧表のすぐ下中央に「====発注書発行====」と大書されていますが、それ以降は図 26 と変わらないと思います。「印刷」しようとしてみてください。プレビューを見ると、改ページされた各仕入先別発注書の中身が、各ページの左上に小さく表示されているはずです。

もしそのようになっていなかったら、まず以下の操作を試みてください。

「[環境設定] → [パフォーマンス] → [すべてのキャッシュをクリア] → 「✓ キャッシュがクリアされました」との表示が出たら、 → 「[サイトへ戻]」って → 再び「仕入れノート」を見てください。これで治ることがよくあります。

それでもダメでしたら、CSS インジェクター記載内容を確認してください。

さて、「中身」をこれから CSS で修飾します。再び CSS 「purchasenotecss」を開きます。

まず各ページの「発注書」という記載です。

JavaScript 「purchasenotejs」では、関数「sheetHeader」の中で「sheetName」という id が付けられています。そこに対して、例えば以下の記載をします。

```

#sheetName {
    height: 200px;          /* id="sheetName"の範囲の文字は */
    padding-top: 100px;    /* 文字を入れる仮想 box の高さを 200px に */
    font-size: 36px;       /* box の上辺から 100px 下に文字を表示 */
    display: flex;         /* 文字サイズは 36px */
    justify-content: center; /* 以下 3 行で中央に表示 */
    align-items: center;
}

```

「仕入先」と「FAX」は、クラス名「supplier」でまとめています。これは、発注書ページの左側に表示するためです。以下のようにします。

```
.supplier {                /* class="supplier"の範囲の事項は */
    position: absolute; /* 「絶対配置」法で */
    left: 15%;          /* 画面左から 15%のところへ揃えて表示 */
}
```

「仕入先」は以下のように修飾します。

```
#supName {                /* id="supName"の範囲の文字は */
    font-size: 24px;      /* 文字サイズ 24 ピクセルに */
    text-decoration-line: underline; /* 下線を引く */
}
```

「FAX」は修飾なしでそのまま表示してみましょう。

「自社情報」は、クラス名「myCompany」でまとめています。これはページの右側に表示します。以下のように書きます。

```
.myCompany {              /* 自社のことは */
    position: absolute; /* 「絶対配置」法で */
    left: 65%;          /* 左から 65%のところへ揃えて表示 */
}
```

これで「仕入れノート」を表示してみると、「マイカンパニー株式会社」が、「仕入先 御中」よりも位置的に若干上に出ていたので、2行程度下げることになります。

CSS を更に勉強するのは面倒なので、JavaScript を手直しします。関数「myCompany」の中の「マイカンパニー株式会社」という文字列の直前に、改行マーク「<br>」をひとつ書き込みます。なお、これは推奨されない方法のようです。

次は「食材注文表」です。JavaScript では関数「tableStart」の中にクラス名を「stuffTable」と書いているので、表全体に関わることはこれを利用します。

まず縦の位置ですが、自社の FAX 番号よりも下になるようにしなければなりません。また左右の余白を等しく、表自体の幅は 800 ピクセルにします。文字の大きさは FAX 番号などより少し大きくします。以下ご参照。

```
.stuffTable {
    margin-top: 150px;      /* 上は 150px 開ける */
    margin-left: auto;      /* 左右余白を均等に自動調整 */
    margin-right: auto;     /* 同上 */
    width: 800px;          /* 表の幅を 800px に */
    font-size: larger;     /* 文字サイズを 1 段階大きく */
}
```

表の項目名はセルの中央に、数値は右寄せにします。

```
.myKoumoku th {          /* クラス「myKoumoku」中の「th」項目は */
    text-align: center;    /* 文字をセルの中央に */
}

#suji {                  /* id が「suji」の項目は */
    text-align: right;     /* 右寄せで */
}
```

おっと、「数値は右寄せ」は、JS Injector で設定していませんでしたね。数値（変数）を記載している箇所の直前の「<td>」などの中に、「id="suji"」と書き込みします。場所は関数「tableBody」と「tableEnd」の中にあります。例えば、

```
「temp += '<td>' + price.toLocaleString() …」が、  
「temp += '<td id="suji">' + price.toLocaleString() …」となります。
```

表のセル幅を設定します。

```
.stuffTable td:first-child { /* 表の中の第1「td」項目（食材名）は */  
  width: 400px;                /* セル幅を 400px に */  
  text-align: center;          /* 文字をセルの中央に */  
}  
  
.stuffTable td:nth-child(2) { /* 表の中の第2「td」項目（単価）は */  
  width: 100px;                /* セル幅 100px に */  
}
```

以上で CSS を「保存」します。

また少し欲が出てきました。

「発注日」（発注書発行日時）を発注書の中に表示させ、発注書の「ID（識別符）」としましょう。

これで最後です。

### (8)-3 蛇足（「発注日」の表示）

「環境設定」→「Asset Injector」→「JS Injector」→「purchasenotejs」の「編集」。

「コード」欄の最後に、以下の関数を継ぎ足します。

```
function hachubi() {                                /* 発注日  
  let date = new Date();                            /* 現在日時を取得  
  let year = date.getFullYear();                    /* 年を切り取り  
  let month = date.getMonth() + 1;                  /* 月を切り取り（月はゼロ始まり）  
  let day = date.getDate();                          /* 日を切り取り  
  let time = date.toLocaleTimeString();              /* 時刻取得  
  let dfmt = 'YYYY 年 MM 月 DD 日 ';                 /* 日付表示書式設定  
  dfmt = dfmt.replace(/YYYY/g,year);                 /* 表示書式の YYYY を実年に置き換え  
  dfmt = dfmt.replace(/MM/g,month);                  /* 月について同上  
  dfmt = dfmt.replace(/DD/g,day);                    /* 日について同上  
  dfmt += time;                                       /* 日付に時刻を加える  
  return dfmt;                                       /* dfmt の値を返す  
}
```

また、関数「myCompany」の「マイカンパニー株式会社」と書いた行の直前に以下の行を書き込みます。

```
temp += '<div id = "hachubi">発注日: ' + hachubi() + '</div>';
```

以上で、「仕入れノート」の一覧表の後に、仕入先ごとに改ページされて図 27 の「発注書」が出来上がれば、完成です。

【図 27：発注書/ 完成版（仕入先ごとに改ページされてプリントできます。）】

発注書			
<u>八百屋お七 御中</u>		発注日：2021 年 5 月 27 日 22:46:57	
FAX: 03-3502-8112		マイカンパニー株式会社	
		〒160-0021	
		東京都新宿区歌舞伎町 1 丁目 1 - 1	
		TEL: 03-3204-7166	
		FAX: 03-3204-7167	
食材名	百g単価	数量(×百g)	小計 (円)
玉葱	36	16	576
人参	48	44	2,112
		合計	2,688
		合計(消費税込)	2,903

これでやっと本書のゴールに到達しました。  
お疲れさまでした。

本書の成果物は、以下のサイトで公開しています。

<https://dev-maru-1.pantheonsite.io/>

user name: anyone  
pass word: any1

ログインしなくても料理の注文や、仕入れノートの検索はできますが、ログインすると、「調理済」の「コメント追加」が可能となります。試してみてください。

登場するロゴマークやイラストは、すべて筆者の手書きなので、著作権で問題が生じることはありません。

Drupal はホームページやブログサイトを作ることしかできない、とは思えません。

もしかすると、製造業等の企業内で、ネットでの宣伝や e-コマースはもとより、営業管理や生産管理をも統合する、巨大な基幹システムを構築することができる手段となり得るのではないかと考えています。

それにしても私は無能で、不勉強で、好奇心旺盛でも飽きっぽく、しかも人生の時間がもうあまりありません。

どなたか Drupal に関する文化のバトンを受け取ってくださる方を期待しています。

以 上

## 【付録 1 : Js Injector 「purchasenotejs」 のコード】 (コピペしてお使いください。)

```
// ビュー「仕入れノート」から情報を抽出
const orderTime = document.getElementsByClassName('views-field-field-order-time');// 注文日時
const orderer = document.getElementsByClassName('views-field-field-orderer');// 予約
const orderID = document.getElementsByClassName('views-field-title');// 注文 ID
const dishID = document.getElementsByClassName('views-field-field-order-dish');// 料理 ID
const dishName = document.getElementsByClassName('views-field-field-dish-name');// 料理名
const dishQuant = document.getElementsByClassName('views-field-field-order-quantity');//注文数
const foodstuffID = document.getElementsByClassName('views-field-field-dish-foodstuff');//
const foodstuffName = document.getElementsByClassName('views-field-field-foodstuff-name');//
const foodamount = document.getElementsByClassName('views-field-field-dish-foodamount');//
const foodstuffPrice = document.getElementsByClassName('views-field-field-foodstuff-unitprice');//
const supplierID = document.getElementsByClassName('views-field-field-foodstuff-supplier');//
const supplierName = document.getElementsByClassName('views-field-field-supplier-name');//
const FAX = document.getElementsByClassName('views-field-field-supplier-fax');//
const orderLength = orderTime.length; // 「注文日時」の数を要素の数とします

// 要素から切り出すデータのみを収容する配列変数を宣言する (A は array の意味)
const orderTimeA = []; // 注文日時
const ordererA = []; // 予約
const orderIDA = []; // 注文 ID
const dishIDA = []; // 料理 ID
const dishNameA = []; // 料理名
const dishQuantA = []; // 注文数
const foodstuffIDA = []; // 食材 ID
const foodstuffNameA = []; // 食材名
const foodamountA = []; // 食材量
const foodstuffPriceA = []; // 材単価
const supplierIDA = []; // 店 ID
const supplierNameA = []; // 仕入先名
const FAXA = []; // FAX

// 新しい配列変数に、要素からデータを取り出して入れる
for(let index = 0; index < orderLength; index++) { // 0 番から「要素数-1」番まで{}内を繰り返し
  orderTimeA[index] = orderTime[index].textContent.trim(); // 注文日時
  ordererA[index] = orderer[index].textContent.trim(); // 予約
  orderIDA[index] = orderID[index].textContent.trim(); // 注文 ID
  dishIDA[index] = dishID[index].textContent.trim(); // 料理 ID
  dishNameA[index] = dishName[index].textContent.trim(); // 料理名
  dishQuantA[index] = dishQuant[index].textContent.trim(); // 注文数
  foodstuffIDA[index] = foodstuffID[index].textContent.trim(); // 食材 ID
  foodstuffNameA[index] = foodstuffName[index].textContent.trim(); // 食材名
  foodamountA[index] = foodamount[index].textContent.trim(); // 食材量
  foodstuffPriceA[index] = foodstuffPrice[index].textContent.trim();// 材単価
  supplierIDA[index] = supplierID[index].textContent.trim(); // 店 ID
  supplierNameA[index] = supplierName[index].textContent.trim(); // 仕入先名
  FAXA[index] = FAX[index].textContent.trim(); // FAX
} // {}の終わり
// カンマ区切りの食材量を一つずつ表示する
for(let index = 1; index < orderLength; index++) { // 1 番から「要素数-1」番まで{...}を繰り返す
  let kokoValue = foodamountA[index].split(","); // カンマ区切りの値を切り出して配列変数に入れる
  for(let ko = 0; ko < kokoValue.length; ko++) { // 切り出した数だけ順に繰り返す
    foodamountA[index] = kokoValue[ko]; // foodamountA[...]に値を 1 個だけ入れる
    foodamount[index].textContent = kokoValue[ko]; // 対応するソースコードに値を 1 個だけ入れる
    index++; // ビューの 1 行分終わったので index を 1 増やす
  } // 内側の for 構文を繰り返す
  index--; // index の値を 1 つ増やしすぎたので 1 つ減らす
} // 外側の for 構文を繰り返す

// 食材別に食材量を集計する
let stuffSum = {}; // 食材量集計の連想配列の宣言
let supplier = {}; // 仕入先情報用連想配列の宣言
let foodstuff = {}; // 食材情報用連想配列の宣言
```

```
for (let index = 1; index < orderLength; index++) { // データ行だけ繰り返し  
    let supID = supplierIDA[index]; // 仕入先 ID を supID へ代入  
    let stuffID = foodstuffIDA[index]; // 食材 ID を stuffID へ代入  
    if(!stuffSum[supID]) {  
        stuffSum[supID] = {};  
        supplier[supID] = {};  
        supplier[supID]['name'] = supplierNameA[index];  
        supplier[supID]['FAX'] = FAXA[index];  
        foodstuff[supID] = {};  
    }  
    if(!stuffSum[supID][stuffID]) {  
        stuffSum[supID][stuffID] = {};  
        stuffSum[supID][stuffID] = 0;  
        foodstuff[supID][stuffID] = {};  
        foodstuff[supID][stuffID]['name'] = foodstuffNameA[index];  
        foodstuff[supID][stuffID]['price'] = foodstuffPriceA[index];  
    }  
    stuffSum[supID][stuffID] += dishQuantA[index] * foodamountA[index];  
}  
  
// =====ここから「発注書作成」です=====  
const view = document.querySelector('.view-content');  
let temp = '<div class ="myAddContent"></div>';  
view.insertAdjacentHTML('afterend',temp);  
const myAdd = document.querySelector('.myAddContent');  
temp += '<div id ="declaration">=====  
myAdd.insertAdjacentHTML('beforeend',temp);  
kaipage();  
  
// 発注書の作成  
for(let sID in supplier) {  
    sheetHeader();  
    let supName = supplier[sID]['name'];  
    let supFAX = supplier[sID]['FAX'];  
    supOnchu(supName,supFAX);  
    myCompany();  
    tableStart();  
    let groundTotal = 0;  
    for ( let fID in stuffSum[sID]) {  
        let fName = foodstuff[sID][fID]['name'];  
        let fPrice = foodstuff[sID][fID]['price'];  
        let fAmount = stuffSum[sID][fID];  
        let subtotal = fAmount * fPrice;  
        tableBody(fName,fPrice,fAmount,subtotal);  
        groundTotal += subtotal;  
    }  
    tableEnd(groundTotal);  
}  
  
function sheetHeader() {  
    temp = '<div class="eachSheet">';  
    temp += '<div id="sheetName">発注書</div>';  
    myAdd.insertAdjacentHTML('beforeend',temp);  
}  
  
function supOnchu(shop, fax) {  
    temp = '<div class="supplier">';  
    temp += '<div id ="supName">' + shop + '&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~<br></div>';  
    temp += '<div id ="supFAX">FAX:&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~<br></div></div>';  
    myAdd.insertAdjacentHTML('beforeend',temp);  
}  
  
function myCompany() {  
    temp = '<div class="myCompany">';  
    temp += '<div id ="hachubi">発注日:' + hachubi() + '</div>';  
    temp += '<div id ="mvCompName"><br>マイカンパニー株式会社</div>':
```



```

temp += '<div id = "myPcode">〒160-0021</div>'; // 自社郵便番号 (同上)
temp += '<div id = "myAddress">東京都新宿区歌舞伎町1丁目1-1</div>'; // 自社住所 (同上)
temp += '<div id = "myTEL">TEL: 03-3204-7166</div>'; // 自社電話番号 (同上)
temp += '<div id = "myFAX">FAX: 03-3204-7167</div></div>'; // 自社 FAX 番号 (同上)
myAdd.insertAdjacentHTML('beforeend',temp); // temp を myAdd の最後の直前に書く
}

function tableStart() { // 【関数】食材注文表のヘッダ
temp = '<table border="1" class="stuffTable">'; // 「表」のはじまり
temp += '<thead>'; // ヘッダ部のはじまり
temp += '<tr class="myKoumoku">'; // 項目名行にクラス名「myKoumoku」を設定
temp += '<th>食材名</th>'; // 第1項目は「食材名」(以下同様)
temp += '<th>百 g 単価</th>'; // 第2項目「百 g 単価」
temp += '<th>数量(×百 g)</th>'; // 第3項目「数量(×百 g)」
temp += '<th>小計 (円) </th>'; // 第4項目「小計」
temp += '</tr>'; // 項目名行おわり
temp += '</thead>'; // ヘッダ部おわり
temp += '<tbody>'; // 次の body 部のはじまり。繰り返しの外に置く
}

function tableBody(name,price,amount,subt) { // 【関数】食材注文表の本体
temp += '<tr>'; // 1行書くことを宣言
temp += '<td>' + name + '</td>'; // 食材名です
temp += '<td id="suji">' + price.toLocaleString() + '</td>'; // 単価を3桁表示
temp += '<td id="suji">' + amount.toLocaleString() + '</td>'; // 数量を3桁表示
temp += '<td id="suji">' + subt.toLocaleString() + '</td>'; // 小計を3桁表示
temp += '</tr>'; // 1行のおわり
}

function tableEnd(total) { // 【関数】食材注文表全体のおわり処理です
temp += '</tbody>'; // Body 部おわり
temp += '<tfoot>'; // フッタ部のはじまり
temp += '<tr><th></th><th></th><td id="goukeina">合計</td>'; // 「合計」記載
temp += '<th id="suji">' + total.toLocaleString() + '</th></tr>'; // 合計値記載
let addTax = Math.floor(total * 1.08); // total を 1.08 倍し、小数部切り捨て
temp += '<tr><th></th><th></th><td id="zeikomina">合計(消費税込)</td>'; // 「8%消費税込み合計」記載
temp += '<th id="suji">' + addTax.toLocaleString() + '</th></tr>'; // 税込み合計値記載
temp += '</tfoot>'; // フッタ部おわり
temp += '</table>'; // 食材注文表全体のおわり
temp += '</div>'; // クラス名「eachSheet」のおわり
myAdd.insertAdjacentHTML('beforeend',temp); // 仕入先別「発注書」の書き出し
kaipage();
}

function kaipage() { // 【関数】改ページ
temp = '<div id = "pagebreak"></div>';
myAdd.insertAdjacentHTML('beforeend',temp);
}

function hachubi() { // 【関数】発注日計算
let date = new Date(); // 現在日時を取得
let year = date.getFullYear(); // 年を切り取り
let month = date.getMonth() + 1; // 月を切り取り (月はゼロ始まり)
let day = date.getDate(); // 日を切り取り
let time = date.toLocaleTimeString(); // 時刻取得
let dfmt = 'YYYY 年 MM 月 DD 日 '; // 日付表示書式設定
dfmt = dfmt.replace(/YYYY/g,year); // 表示書式の YYYY を実年に置き換え
dfmt = dfmt.replace(/MM/g,month); // 月について同上
dfmt = dfmt.replace(/DD/g,day); // 日について同上
dfmt += time; // 日付に時刻を加える
return dfmt; // dfmt の値を返す
}

```

## 【付録2：CSS Injector「purchasenotecss」のコード】（コピペしてお使いください。）

```
#declaration {                /* id="declaration" の範囲の文字について */
    font-size: 48px;           /* 文字サイズを 48 ピクセルに */
    display: flex;             /* 中央に表示するためには以下 3 行必要 */
    justify-content: center;
    align-items: center;
}

#pagebreak {
    page-break-before : always; /* 改ページします */
}

#sheetName {                   /* id="sheetName" の範囲の文字は */
    height: 200px;             /* 文字を入れる仮想 box の高さを 200px に */
    padding-top: 100px;        /* box の上辺から 100px 下に文字を表示 */
    font-size: 36px;           /* 文字サイズは 36px */
    display: flex;             /* 以下 3 行で中央に表示 */
    justify-content: center;
    align-items: center;
}

.supplier {                    /* class="supplier" の範囲の事項は */
    position: absolute;        /* 「絶対配置」法で */
    left: 15%;                 /* 画面左から 15% のところへ揃えて表示 */
}

#supName {                     /* id="supName" の範囲の文字は */
    font-size: 24px;           /* 文字サイズ 24 ピクセルに */
    text-decoration-line: underline; /* 下線を引く */
}

.myCompany {                   /* 自社のことは */
    position: absolute;        /* 「絶対配置」法で */
    left: 65%;                 /* 左から 65% のところへ揃えて表示 */
}

.stuffTable {
    margin-top: 150px;         /* 上は 150px 開ける */
    margin-left: auto;         /* 左右余白を均等に自動調整 */
    margin-right: auto;        /* 同上 */
    width: 800px;              /* 表の幅を 800px に */
    font-size: larger;         /* 文字サイズを 1 段階大きく */
}

.myKoumoku th {                /* クラス「myKoumoku」中の「th」項目は */
    text-align: center;        /* 文字をセルの中央に */
}

#suji {                        /* id が「suji」の項目は */
    text-align: right;         /* 右寄せで */
}

.stuffTable td:first-child {   /* 表の中の第 1「td」項目（食材名）は */
    width: 400px;              /* セル幅を 400px に */
    text-align: center;        /* 文字をセルの中央に */
}

.stuffTable td:nth-child(2) {  /* 表の中の第 2「td」項目（単価）は */
    width: 100px;              /* セル幅 100px に */
}
```

以 上