PHP - Formulaires

Tout script PHP qui attend des paramètres (GET ou POST) doit scrupuleusement contrôler la présence et la validité de la valeur de chacun.

Analyse Comme d'habitude en développement, la première étape consiste à faire une analyse du problème, avant de coder. Il s'agit ici de spécifier les paramètres qu'attend le script et le comportement à adopter si ces paramètres ne sont pas conformes . Pour chaque paramètre il faut établir :

1. est-il obligatoire ou facultatif ?
   — s'il est facultatif, quelle est la valeur par défaut ?
   — s'il est obligatoire que doit faire le script en cas d'absence ?
2. quelles sont les valeurs autorisées ?
   — que doit faire le script si la valeur est incorrecte ?

Développement

1. On regroupera en **un seul composant** la vérification de l'ensemble des paramètres, qui aura lieu avant de débuter la génération de la page. En présence d'une erreur pour l'un des paramètres, une procédure d'erreur (par exemple génération d'une page d'erreur) sera déclenchée.
2. Chaque paramètre est testé séparément (annoter en commentaire la spécification du paramètre). À l'issue de ce traitement, la valeur retenue pour le paramètre est rangée dans une variable du programme (ou alors une erreur est positionnée)

Voici une solution utilisant le mécanisme des exceptions :

```
<?php
class ParmsException extends Exception{}; // classe exception spécifique

// Bloc de verification/filtrage des paramètres HTTP :
try {
  /* nom : abscisse  ;   valeur autorisée : nombre;   statut : requis.
   */
  if ( ! isset($_GET['abscisse']) )
     throw new  ParmsException("param 'abscisse' : valeur absente");
  else if  ( ! is_numeric($_GET['abscisse']) )
     throw new  ParmsException("param 'abscisse' : valeur incorrecte");
  else
     $abscisse = $_GET['abscisse'];

  /* nom : hauteur;  valeur autorisée : nombre;  statut : optionnel (défaut : 1);
      Erreur en cas de valeur incorrecte
  */
  if ( ! isset($_GET['hauteur']) )
     $hauteur = 1;  // val par défaut
  else if (! is_numeric($_GET['hauteur']) )
     throw new  ParmsException("param 'hauteur' : valeur incorrecte");
  else
     $hauteur = $_GET['auteur'];
  /* traitement normal  */
  require('views/pageNormale.php'); // traitement normal
}  catch (ParmsException $e) { // traitement erreur
    $errorMessage = $e->getMessage();
    require('views/pageErreur.php');
}
?>
```

Utilisation des filtres

La fonction `filter_input` fournit une autre façon d'accéder aux paramètres, sans manipuler directement les tableaux `$_GET` ou `$_POST` :

`filter_input(`*méthode, nom, filtre , options_et_flags*`)`

- méthode : `INPUT_GET` ou `INPUT_POST` (constantes prédéfinies)
- nom : celui du paramètre (chaîne)
- filtre à appliquer. Un filtre est désigné par une constante prédéfinie. Le filtre va vérifier la valeur reçue (et parfois la transformer)
- options_et_flags (facultatif).

Résultat de la fonction (en règle générale) :

- `NULL` : si le paramètre est absent
- `FALSE` : si la valeur reçue est incorrecte (« ne passe pas le filtre »)
- la `valeur reçue` pour le paramètre si elle est correcte (« passe le filtre »)

Exemple : `filter_input(INPUT_GET,'abscisse',FILTER_VALIDATE_FLOAT)` vérifie que le paramètre abscisse est un nombre. La vérification du paramètre `abscisse` devient :

```
$abscisse = filter_input(INPUT_GET,'abscisse',FILTER_VALIDATE_FLOAT);
if ($abscisse === NULL)
  throw new  ParmsException("param 'abscisse' : valeur  absente");
else if  ($abscisse === FALSE)
  throw new  ParmsException("param 'abscisse' : valeur  incorrecte");
```

Voici une sélection de filtres

filtres de validation			
nom	accepte	rés.	options
`FILTER_VALIDATE_INT`	entiers	`int`	`default`, `min_range`, `max_range`
`FILTER_VALIDATE_FLOAT`	nombres	`float`	`default`
`FILTER_VALIDATE_EMAIL`	adresses mail	`string`	`default`
`FILTER_VALIDATE_REGEXP`	selon une exp. rég.	`string`	`regexp`, `default`

filtres de nettoyage			
nom	action	rés.	flags
`FILTER_SANITIZE_STRING`	encode les car. spéciaux supprime les balises	`string`	cf doc PHP
`FILTER_UNSAFE_RAW`	ne fait rien	`string`	cf doc PHP

Un filtre de nettoyage accepte toutes les valeurs (et donc ne renvoie jamais `FALSE`) mais peut modifier cette valeur.

Un filtre de validation accepte une option `default` (qui définit une valeur par défaut) ; dans ce cas il ne renvoie jamais `NULL` ni `FALSE` mais la valeur par défaut, tant en cas d'absence que de valeur incorrecte.

Exemple :

```
$hauteur = filter_input(INPUT_GET,'hauteur',FILTER_VALIDATE_INT,
                ['options'=>['default'=>1,'min_range'=>0]]);
```

`$hauteur` vaudra 1 dans tous les cas où le paramètre n'est pas un entier positif ou nul.

Remarques :

1. en cas d'utilisation d'une valeur par défaut, `filter_input()` l'applique même en cas de paramètre incorrect. Si on veut déclencher une erreur dans ce cas, il faut gérér la valeur par défaut en dehors de `filter_input()` (voir ex ci-dessous)
2. il n'existe pas de filtre prédéfini pour les valeurs énumérées. On peut les traiter comme dans l'exemple ci-dessous. L'apport de `filter_input` pour ce type de valeurs peut paraître faible, mais on le retiendra par cohérence avec le traitement des autres paramètres.

```php
/* nom : hauteur;  valeur autorisée : nombre;  statut : optionnel (défaut : 1);
   Erreur en cas de valeur incorrecte
*/
$hauteur = filter_input(INPUT_GET,'hauteur',FILTER_VALIDATE_INT,
             ['options'=>['min_range'=>0]]);
if ($hauteur === NULL)
  $hauteur = 1; // valeur par défaut
else if ($hauteur === FALSE)
  throw new ParmsException("param 'hauteur' : valeur incorrecte");
/* nom : couleur;  valeurs autorisées : rouge vert bleu;  statut : obligatoire;
*/
const COULEUR_VALUES = ['rouge','vert','bleu']; // valeurs autorisées
$couleur = filter_input(INPUT_GET,'couleur',FILTER_UNSAFE_RAW);
if ($couleur === NULL)
  throw new ParmsException("param 'couleur' : valeur absente");
else if (! in_array($couleur, COULEUR_VALUES))
  throw new ParmsException("param 'couleur' : valeur incorrecte");
```

Exercice 1 :
Le thème de cet exercice est la réalisation du site du club des fans de StarWars. Ce site permet de passer une commande de figurines et/ou d'adhérer au club. Les commandes sont ouvertes à toutes et tous mais les adhérents ont un tarif préférentiel.
Le script `factureStarWars.php` reçoit les commandes et affiche la facture. C'est le script principal du site.
Les différents éléments de la commande sont envoyés via des paramètres HTTP. Le script produit la facture correspondant à la commande reçue.
De nombreux éléments de l'implémentation vous sont fournis (tout ce qui concerne la génération de la facture, par exemple). Il vous reste à compléter et notamment à écrire le traitement et la vérification des arguments reçus par le script.
Voici la liste des paramètres HTTP attendus par le script. On distingue 2 catégories : les informations client (nom, adresse, ...) qui n'influent pas sur le montant de la facture et les informations commande. Dans un premier temps nous ne traiterons que les informations client.

`nom`		requis ?	défaut	valeurs autorisées
Informations client				
`civilite`	scalaire	requis		`'Mr'`, `'Mme'`
`nom`	scalaire	requis		chaîne non vide
`prenom`	scalaire	requis		chaîne non vide
`voie`	scalaire	requis		chaîne non vide
`ville`	scalaire	requis		chaîne non vide
`complAd`	scalaire	facultatif	`''`	chaîne
`cp`	scalaire	requis		code postal
Informations commande				
`adhesion`	scalaire	facultatif	`'non'`	`'oui'`,`'non'`,`'dejaMembre'`
`fig`	tableau	requis		nom parmi la liste des figures proposées

Le dossier qui vous est fourni contient
- `factureStarWars.php` : le script principal
- `formulaireProvisoire.html` : formulaire (ne sera utilisé QUE pour la partie 1)
- un dossier `views` qui contient les pages à afficher. Le client n'accède **jamais** directement au contenu de ce dossier, qui est protégé par un `.htaccess`. Le script principal qui incluera l'une ou l'autre de ces « vues »
- un dossier `lib`, lui aussi protégé par un `.htaccess` et destiné à contenir diverses parties de code utilitaires.

Première partie : vérification des arguments

Cette partie est consacrée uniquement à la vérification et au filtrage des paramètres. La facture n'est pas engendrée. Le script principal inclut, selon le cas l'une ou l'autre des 2 pages

- `views/pageTest.php` quand les paramètres sont corrects
- `views/pageTestErreur.php` dans le cas contraire

Ces fichiers, ainsi que le script principal `factureStarWars.php` (consultez son contenu), ne doivent PAS être modifiés à cette question.
Vous allez développer la prise en compte et le filtrage des paramètres, en utilisant la fonction `filter_input`.
Ces opérartions sont à réaliser dans `lib/verifyParms.php` (qui est inclus par le script principal).
Quand un paramètre reçu est correct, vous positionnerez une variabale globale **de même nom** avec la valeur retenue (valeur reçue, ou par défaut ...).
Quand un paramètre est incorrect ou quand un paramètre requis est manquant vous déclencherez une exception instance de `ParmsException` avec comme argument un message d'erreur approprié (voir exemples ci-dessus)
Q 1 . Complétez le script `lib/verifyParms.php` pour prendre en compte les paramètres de la catégorie « informations client ». Les paramètres de type chaîne seront nettoyés en utilisant `FILTER_SANITIZE_STRING`
Vous testerez scrupuleusement le script en utilisant le formulaire provisoire `et aussi` en envoyant directement des paramètres dans l'URL.
Q 2 . Vous allez maintenant ajouter au script `lib/verifyParms.php` la prise en compte des 2 derniers paramètres.
Le paramètre `fig` est cette fois un tableau. Pour prendre en compte ce genre de paramètre, il faut passer à `filter_input` le flag `FILTER_REQUIRE_ARRAY`. Le résultat de `filter_input` est alors un tableau et nom une valeur simple. Exemple :

```
filter_input(INPUT_GET,'t',FILTER_VALIDATE_INT, ['flags'=>FILTER_REQUIRE_ARRAY]);
```

Comme précédemment, testez le fonctionnement.

Deuxième partie : la facture

Le filtrage des paramètres étant bon, vous allez pouvoir générer la vraie facture. Editez cette fois `factureStarwars.php` pour changer les paramètres des `require` et inclure les pages définitives à la place des pages de test.
Le script `formulaireProvisoire.html` est maintenant inutile (renommez-le pour ne plus l'utiliser par mégarde). Le comportement du script principal est maintenant d'inclure

- `views/pageFactureStarWars.php` quand les paramètres sont corrects
- `views/formulaireStarWars.php` dans le cas contraire. Ce script affiche le message d'erreur et re-propose le formulaire.

Q 3 . Vérifiez le bon fonctionnement de la partie « client » de la facture.
La facture est générée par une classe `Facture` qui vous est fournie (à consulter).
Pour l'instant les paramètres `fig` et `adhesion` ne sont pas intégrés à la facture.

a) Complétez le script `views/formulaireStarWars.php` pour ajouter la saisie de
   - l'information sur l'adhésion (paramètre `adhesion`) : prevoir une liste de choix (`select`) parmi les 3 valeurs possibles.
   - des figurines à commander (paramètre `fig[]`) : vous choisirez d'implémenter soit par un `select` à choix multiple, soit par une série de cases à cocher. Il est préférable d'engendrer cette partie en PHP à partir de la liste des valeurs possibles.

b) Complétez le script `views/pageFactureStarwars.php`. pour qu'il tienne compte des paramètres que vous venez d'ajouter (utilisation des méthodes `setAdherent()`, `ajouterAdhesion()` `ajouterFigurine()` de la classe `Facture` dont vous consulterez la doc)

Vérifiez que la facture obtenue est correcte

Q 4 . Amélioration du formulaire
Le formulaire est pour l'instant identique à celui de la question 1.

1. En l'état, il possible d'envoyer un formulaire incomplet. Complétez le code HTML pour faire en sorte que l'envoi devienne impossible en l'absence d'information dans les champs `nom, prenom, voie, cp, commune`

2. De même, contraignez le champ code postal pour n'y valider qu'une chaîne de 5 caractères Vous ferez en sorte qu'une chaîne invalide s'affiche en rouge.