TP MODULES EN PYTHON

Compétences travaillées :

APP	Mobiliser les concepts et les technologies adaptés au problème
ANA	Modéliser un problème en utilisant les objets conceptuels de l'informatique pertinents (fonctions et modules)
REA	Décrire une démarche, un algorithme ou une structure de données permettant de résoudre le problème

1. Approche de la modularité avec les fonctions

La *programmation fonctionnelle* permet de simplifier l'écriture d'un programme.
Voyons ce que l'on peut faire pour améliorer encore l'aspect *modulaire* d'un programme.

1) Ecrire un programme energie.py qui demande à l'utilisateur la masse d'un objet **m**, sa vitesse **V** et son altitude **H** et qui calcule l'énergie mécanique associée.

L'énergie mécanique Em = Ec + Epp

avec l'énergie cinétique Ec = $\frac{1}{2}$ .m.V^2 et l'énergie potentielle Epp = m.g.H

Les trois énergies sont calculées à l'aide de 3 fonctions distinctes nommées Emeca, Ec et Epp.

2) Créer un nouveau programme fonctions.py dans lequel on écrira les 3 fonctions de calcul des énergies. Dans le programme energie.py, ajouter en première ligne l'instruction :

```
from fonctions import Emeca, Ec, Epp
```

Tester le programme et commenter l'intérêt de cette formulation.

Remarques :

- On parle du *module* fonctions.py et du programme energie.py.
- L'appel des fonctions du module se fait de façon nominative et explicite (« fonction par fonction »), mais on peut :

  → Importer l'ensemble : `from fonctions import *`

  → Importer et donner un alias : `from fonctions import Emeca as Em`
- Dans le cas où l'on a beaucoup de module, on les organisera en packages : concrètement, un dossier package qui contient les modules en question. L'instruction d'appel du module devient :

```
from packages.fonctions import Em, Ec, Epp
```

2. Application : création d'un QCM

Nous allons créer un module permettant la création et la correction de QCM en python.

Le QCM est stocké dans un tableau de tableaux du type :

```
QCMNSI = [["L'amie écureuil de Bob l'éponge s'appelle", "En dormant, Patrick bave plus que Gary", "Patrick est une étoile"],\
          ["Cindy", "*Vrai", "à 3 branches"],\
          ["Mindy", "Faux", "*à 5 branches"],\
          ["*Sandy", "match nul", "à 6 branches"]]
```

Il contient les questions sous forme de **chaînes de caractères** et la proposition correcte est précédée du caractère ***** .

On dispose d'un module qcm.py qui contient les fonctions utiles : elles sont nommées, les *docstrings* sont écrites, des ***assert*** également, mais elles restent à coder...

Il y a les 4 fonctions suivantes :

passageQCM(QCM)	*affiche_correction(QCM, i)*
affichageQCM(QCM, i)	*score_reponse(QCM, rep, i)*

Le résultat attendu de l'interface pour chaque question est comme ci-contre :

```
 L'amie écureuil de Bob l'éponge s'appelle
1  :  Cindy
2  :  Mindy
3  :  Sandy
Entrez votre N° de réponse (1, 2 ou 3) : 3

++++++++++++++++++++++++++++++++++++++++
La réponse correcte était :  3  avec ' Sandy '
Score actuel  1  sur  1
++++++++++++++++++++++++++++++++++++++++
```

Le fichier passage_qcm.py contient le programme principal.

Travail demandé :

1) **Coder** les fonctions du module.
2) **Tester** le programme.
3) **Décrire** précisément l'ensemble des fonctions : les entrées, les sorties et les traitements.

Les formulaires sont délimités par la balise **<form>**. Elle permet de regrouper plusieurs éléments de saisie utilisateur (boutons, champs, ...) et qui possède les attributs suivants :

- Attribut ***method*** indique sous quelle forme seront envoyées les réponses « POST » est la valeur qui correspond à un envoi de données stockées dans le corps de la requête, tandis que « GET » correspond à un envoi des données codées dans l'URL, et séparées de l'adresse du script par un point d'interrogation.
- Attribut ***action*** indique l'adresse d'envoi des données une fois le formulaire validé : URL, script CGI, dresse email (**mailto:adresse.email@machine**), etc...

Exemples :

- Une recherche web peut s'écrire **https://duckduckgo.com/?t=canonical&q=cours+d%27informatique+pour+les+profs+nuls&ia=web** .

Les données sont « en clair » dans l'URL, donc avec la methode GET.

- Exemple de syntaxe : `<form action="page.php" method="get">`

Ce formulaire va envoyer les données en GET vers le fichier **page.php** sur le serveur.

Il est possible d'insérer n'importe quel élément HTML de base dans une balise *form* (textes,boutons,tableaux,liens,...) mais il est surtout intéressant d'insérer des éléments interactifs.
La balise **<input *type*=" " *name*=" " >** permet d'accéder à un ensemble de boutons et de champs de saisie en modifiant la valeur de l'attribut ***type***. L'attribut ***name*** est la variable dans laquelle est stockée la valeur du champ de saisie.
Les deux grandes familles sont :

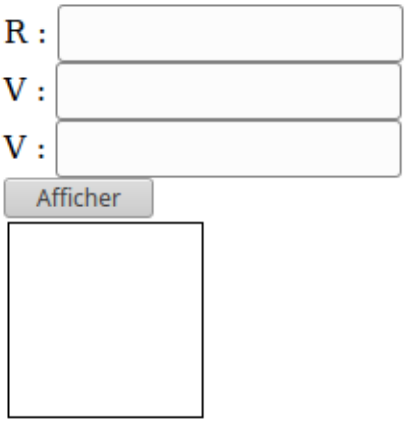
- Le champ de texte **<input type="**`text`**">** : une zone de saisie de texte
- Le bouton radio **<input type="**`radio`**">** : une valeur parmi plusieurs

La validation se fait en général par un bouton final **<button type=**"submit"**>… </button>**.

Pour plus d'éléments : **https://www.w3schools.com/html/html_forms.asp**

Exercice 1 :
Créer un formulaire de 3 champs et un bouton qui donne le visuel ci-contre :

Exercice 2 :
On souhaite créer un formulaire qui affiche la couleur correspondante aux valeurs de RVB saisies.
Il faut donc 3 champs de saisie étiquetés « R », « V » et « B », un bouton « Affichage ».
Pour l'affichage de la couleur demandée, on prévoira un cadre de 100 px (balise ** ** avec ***id*** ="couleur").



ACT 1 Etude d'un formulaire (partie 2)

1.1. Interactions sur le formulaire

Dans les formulaires, l'envoie de paramètres se fait uniquement par une requête HTTP, et donc implique une communication en réseau avec le site distant et le serveur doit recalculer entièrement la page de destination.
L'utilisateur voit donc une nouvelle page se charger, ce qui peut être long !
L'interpréteur JavaScript d'un navigateur offre la possibilité de modifier dynamiquement et en temps réel le rendu d'une page Web, sans passer par le serveur.

Le langage JavaScript est inséré directement dans le code HTML à l'aide de la balise **<script>** :

- soit directement comme dans l'exemple ci-dessus
- soit en intégrant un lien vers un fichier .js

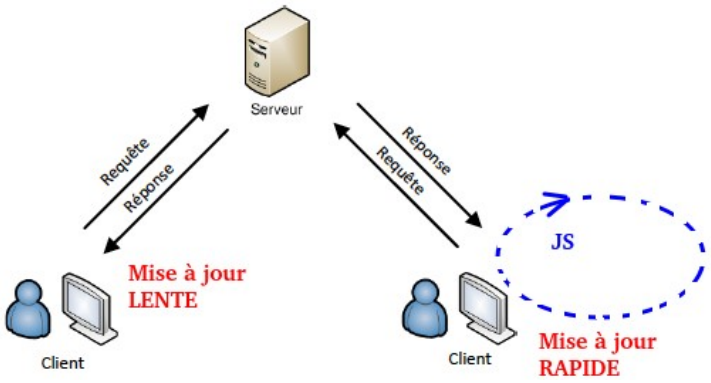
```
<script type="text/javascript" src="js/monscript.js" async></script>
```

Exemple : A partir du fichier ***exemple_js1.html*** donner quelques caractéristiques du langage JavaScript

```
<!DOCTYPE html>
<html>
<head><title>Age</title>
<script type="text/javascript">
let compteur = 0 ;
function suivant() {
compteur = compteur + 1 ;
document.getElementById("valeur").innerHTML = compteur;
}
</script>
</head>
<body>
<button onclick="suivant();">Clic!</button>
<span id="valeur">0</span>
</body>
</html>
```

Des instructions se comportent comme des print en python, il s'agit de **alert("message")** qui ouvre un popup avec le message et **console.log("message")** qui écrit dans la console du navigateur.

Ces instructions sont très pratiques pour voir ce que renvoie le code JavaScript dans une phase de conception.



A l'affichage d'une page, les scripts JavaScript présents sont exécutés. L'une de leur activité principale est la définition de *gestionnaires d'événements.* Ces **gestionnaires d'événements** peuvent modifier la page en ajoutant, supprimant ou modifiant les attributs des balises.

La gestion des évènements par JavaScript permet donc une mise à jour plus rapide de la page (local).

Exemple : A partir du fichier **exemple_js2.html**, ouvrir le navigateur (Firefox) et appuyer sur F12 pour faire apparaître la console.

- Écrire les instructions suivantes dans la console et observer les effets :

Instruction 1	document.body.style.background = "red";	
Instruction 2	document.body.style.background = "";	
Instructions 3	let d = document.getElementById("mondiv"); d.style.color = "blue"; d.style.fontWeight = "bold"; d.style.border = "1pt dashed green";	
Instruction 4	d.innerHTML = "Salut, ça va bien ?";	

- Le fichier html est-il modifié ?

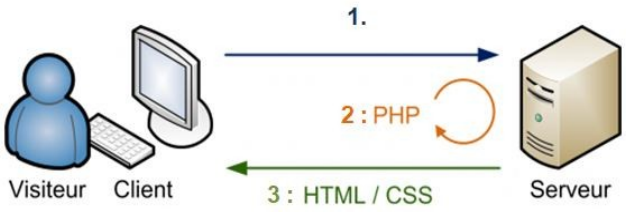
ACT 2 Côté Client en JS

3. Transmettre les données

Nous avons vu que le protocole HTTP permet à des pages web d'envoyer des paramètres de requêtes. Ces derniers sont utilisés pour calculer le contenu de la page renvoyée au client. Le langage HTML permet de décrire un formulaire permettant à l'utilisateur de saisir des valeurs et de les soumettre à un serveur web.
Sur ce serveur le traitement de l'information va se faire en **langage PHP**.
PHP (pour **P**HP **H**ypertext **P**reprocessor) est un langage **libre** et **gratuit**, spécialement conçu pour le développement d'applications web. Il peut être intégré facilement au HTML.

Lorsque le serveur envoie la page web au client, il envoie en fait du code en langage HTML et CSS généré par PHP.

PHP est un langage que seuls les serveurs comprennent et qui **permet de « générer » des pages web.**



Exemple :

- Certains sites parviennent à afficher le pseudo sur toutes les pages. Étant donné que le serveur génère une page à chaque fois qu'on lui en demande une, il peut la personnaliser en fonction des données fournies par le visiteur.

Pour que son ordinateur puisse lire du PHP, il faut qu'il se comporte comme un **serveur**.

Il suffit simplement d'installer les mêmes programmes que ceux que l'on trouve sur les serveurs qui délivrent les sites web aux internautes.

Exemple :

- **Apache** (= serveur web) Il s'agit du plus important (en nombre) des serveurs utiliser sur le web
- **Python CGI** Il s'agit de la bibliothèque python qui permet de créer un serveur, simple et efficace.
  (voir ACT serveur python)
- Les serveur dépendent du protocole utilisé et sont spécialisés :
  HTTP pour le web, SMTP pour les mails, FTP ou SSH pour les fichiers, etc...

ACT 3 Côté Serveur en PHP