

# IFT 1015 – Programmation I

## TP 2

- À faire en groupe de **deux** étudiants.
- Remise : Le 27 avril 2025 à **23:59** au plus tard.

### 1. Objectifs du TP2

Dans ce projet, vous aurez l'occasion de pratiquer les concepts suivants:

- Boucles
- Tableaux
- Fonctions
- Décomposition fonctionnelle
- Traitement d'événements
- Programmation web

Le code que vous devez écrire (fichier « tp2.py ») implante un jeu qui s'exécute dans l'environnement du navigateur web. Une bonne partie du jeu peut se développer avec codeBoot. Cependant, il faut comprendre qu'on vise à déployer le programme comme une application web standard (les détails sont expliquées ci-dessous).

Vous pouvez utiliser le code qui a été montré dans le cours, mais vous ne devez pas utiliser de code provenant d'ailleurs, que ce soit du web ou d'une autre personne.

### 2. Introduction

Ce travail pratique consiste à développer un programme web pour jouer au jeu « **La somme des symboles** ». C'est un jeu d'un seul joueur. Le jeu consiste à trouver les nombres associés à cinq symboles remplissant une grille 5 par 5 qui permettent de donner comme sommes les valeurs affichées au bout de chacune des rangées et de chacune des colonnes.


























Le joueur gagne lorsque la bonne combinaison de 5 nombres est retrouvée. Si les assignations partielles aux symboles ne peuvent pas former les sommes demandées, le joueur perd.

### 3. Déroulement du jeu

Il y a 5 symboles utilisés dans le jeu : le cercle, la pyramide, le pentagone, le cube et l'étoile. Chaque symbole représente un nombre différent. Ces symboles sont placés sur une grille 6 x 6 occupant les 5 premières lignes et 5 premières colonnes. Les valeurs, représentant les sommes des nombres associés aux symboles, sont placés sur la dernière colonne et sur la dernière rangée.

Donc, l'interface graphique du jeu contient une grille de 5 rangées et 5 colonnes avec les images des symboles. Au-dessus de la grille, il y a un bouton de démarrage d'une nouvelle partie et une ligne affichant un message.

Voici à quoi ressemble la fenêtre au début d'une partie :

Nouvelle partie					
Jouer!					
					16
					20
					17
					18
					17
12	22	17	14	23	

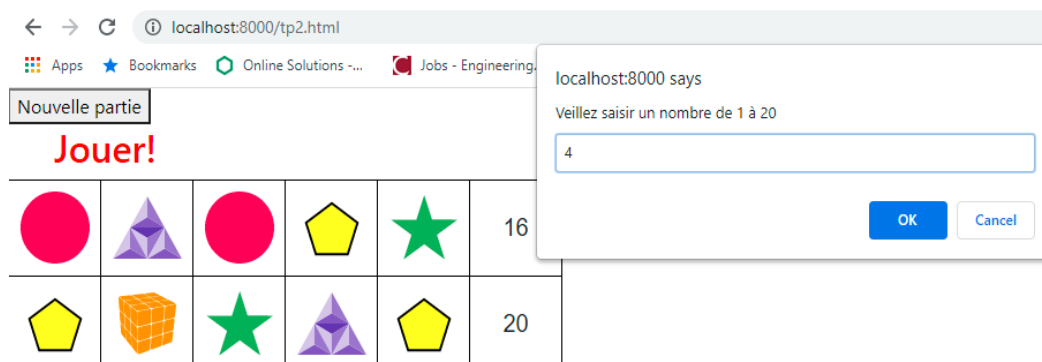
### Étape Initialisation :

1. Générer 5 entiers positifs aléatoires de 1 à 20 et les associer un par un à chacun des 5 symboles utilisés dans le jeu.
2. Sur chaque rangée (colonne), générer (de manière aléatoire) les types de symboles à placer. Le même symbole peut être placé plus qu'une fois sur une rangée (colonne), par exemple sur la ligne 4, la pyramide est présentée 3 fois.
3. Après avoir rempli la grille avec les symboles, il faut calculer les sommes des nombres et les afficher au bout de chaque rangée et de chaque colonne. Par exemple, pour la ligne 4, en supposant que le cube représente 2, la pyramide – 3 et l'étoile – 7, la somme affichée sur cette ligne doit être 18.

### Étape Jeu

1. En cliquant sur un symbole, le joueur doit deviner le nombre représenté par ce symbole en l'écrivant dans une boîte de dialogue (un nombre de 1 à 20, fonction Python `input`). Après avoir choisi le nombre, ce nombre doit apparaître sur toutes les images associées et les sommes sur les rangées et les colonnes doivent être ajustées pour représenter les sommes restantes à atteindre lors de prochaines devinettes. Si un choix implique l'impossibilité de créer l'une des sommes spécifiées, le joueur perd. Le message correspondant doit être affiché.

Voici à quoi pourrait ressembler la fenêtre lorsque le joueur a cliqué sur la première image (cercle, case (0,0)) :



Avant la saisie de 4 :

Nouvelle partie

Jouer!

					16
					20
					17
					18
					17
12	22	17	14	23	

Après avoir saisi 4 :

Nouvelle partie

Jouer!

					8
					20
					13
					18
					13
4	22	13	10	23	

Après avoir saisi 20 :

Nouvelle partie

Vous avez perdu.

					-24
					20
					-3
					18
					-3
-28	22	-3	-6	23	

Après avoir saisi toutes les valeurs correctement :

Nouvelle partie

Vous avez gagné!

					0
					0
					0
					0
					0
0	0	0	0	0	

#### 4. Détails

Pour vous démarrer dans la bonne direction, vous trouverez sur Studium le **fichier serveur-web.py** et le fichier **documents.zip**. Le fichier **documents.zip** contient le répertoire avec les documents qui peuvent être utilisés dans votre tp. Le sous répertoire **symboles** contient les images des symboles en format **SVG**, « Scalable Vector Graphics ».

Pour exécuter le code, il faudra démarrer le serveur avec la commande **python3 serveur-web.py** et lancer la requête **http://localhost:8000/tp2.html** dans votre navigateur. Les fichiers **tp2.html**, **tp2.py**, **codeboot.bundle.js** et **codeboot.bundle.css** sont également présents dans le répertoire **documents**. Vous ne devez pas changer cette organisation de fichiers ni changer le contenu des fichiers sauf **tp2.py** que vous devez compléter avec votre code. Le fichier **tp2.html** contient des directives pour inclure les fichiers **codeboot.bundle.js**, **codeboot.bundle.css** et le code Python **tp2.py**. Le corps du document est le suivant :

```
<body onload="init()">
```

```
<div id="main"></div>
```

```
</body>
```

Le traiteur d'événement **onload** du corps fait l'appel à la fonction **init** définie dans **tp2.py** pour démarrer l'exécution du code Python au moment du chargement du fichier **tp2.html**. La fonction **init** doit créer le contenu HTML qui sera mis dans l'élément **<div id="main"></div>**. On pourrait, par exemple, définir la fonction **init** comme suit pour afficher une grille 2x6 de symboles et sommes :

```
def init():
    main = document.querySelector("#main")
    main.innerHTML = """
    <style>
        #jeu table { float: none; }
        #jeu table td {
            border: 1px solid black;
            padding: 1px 2px;
            width: 80px;
            height: 80px;
            font-family: Helvetica;
            font-size: 20px;
            text-align: center;
        }
        #jeu table td img {
            display: block;
            margin-left: auto;
            margin-right: auto;
            object-fit: contain;
            width: 80%;
            height: 80%;
        }
    </style>
    <div id="jeu">
        <table>
            <tr>
                <td id="case0"></td>
                <td id="case1"></td>
                <td id="case2"></td>
                <td id="case3"></td>
                <td id="case4"></td>
                <td id="case5"></td>
            </tr>
            <tr>
                <td id="case6"></td>
                <td id="case7"></td>
                <td id="case8"></td>
                <td id="case9"></td>
                <td id="case10"></td>
                <td id="case11"></td>
            </tr>
        </table>
    </div>"""
```

Évidemment, pour une grille de grande taille, il serait mieux de créer le HTML avec des boucles pour éviter des répétitions de code. Pour changer les attributs de style de l'élément, on peut utiliser les méthodes **setAttribute/removeAttribute**.

Pour représenter les symboles dans votre programme, il est suggéré d'utiliser un nombre de 0 à 4 pour identifier les symboles. Par exemple, 0 – cercle, 1 – pentagone, etc. Donc, 5 symboles peuvent être encodés par les nombres de 0 à 4. Par la suite, on peut générer aléatoirement une de ces 5 valeurs pour décider quel symbole mettre dans une case de tableau modélisant le jeu.

Le développement du programme peut se faire de cette manière :

Éditer directement le fichier **tp2.py** avec un éditeur de code et utiliser le navigateur pour rafraichir la page **http://localhost:8000/tp2.html** (ce qui va exécuter votre programme **tp2.py** à nouveau). Un clic avec le bouton de droite vous permet d'afficher dans une fenêtre flottante l'état du programme, ce qui est utile s'il y a des bogues ou des appels à **breakpoint()** dans votre code.

#### Note :

1. Pour le système d'exploitation Windows, il se peut qu'il y ait un problème avec l'utilisation du répertoire avec le nom **documents**. Utilisez le nom **documents\_tp2** ou autre variante de ce nom en faisant l'ajustement du nom de répertoire dans le fichier **serveur-web.py** :  
**def obtenirDocument(path) :**  
    **#print(path)**  
    **return readFile('documents\_tp2' + path)**
2. Pour pouvoir superposer les éléments HTML, on doit créer un conteneur ou utiliser un élément déjà existant comme conteneur. Spécifier la position du conteneur comme relative et la position de l'élément superposé comme absolu. Exemple : l'élément avec le contenu « Centered » est placé par-dessus et au centre de l'image du conteneur « container » :

```
<div class="container">
  
  <div class="centered">Centered</div>
</div>

.centered {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

## 5. Évaluation

Ce travail compte pour 15 points dans la note finale du cours. Vous devez le faire par groupes de 2 personnes. Indiquez vos noms clairement dans les commentaires au début de votre code.

**Vous devez remettre votre fichier tp2.py uniquement.** Chaque fonction devrait avoir un bref commentaire pour dire ce qu'elle fait, il devrait y avoir des lignes blanches pour que le code ne soit pas trop dense, les identificateurs doivent être bien choisis pour être compréhensibles et respecter le standard CamelCase.