

Génération du tableau par python

Afin de faire les cases de manière plus efficace (trop la flemme de faire les 118 à la main), j'ai décidé d'adapter le programme que j'ai réalisé pour la page sur les [cubes périodiques](#). L'idée étant alors de pouvoir cliquer sur une case du tableau et décider de réaliser le cube pour cet élément comme précédemment ou de sélectionner le groupe (*i.e.* gaz noble, métal de transition, ...) et de produire le .svg pour le tableau.

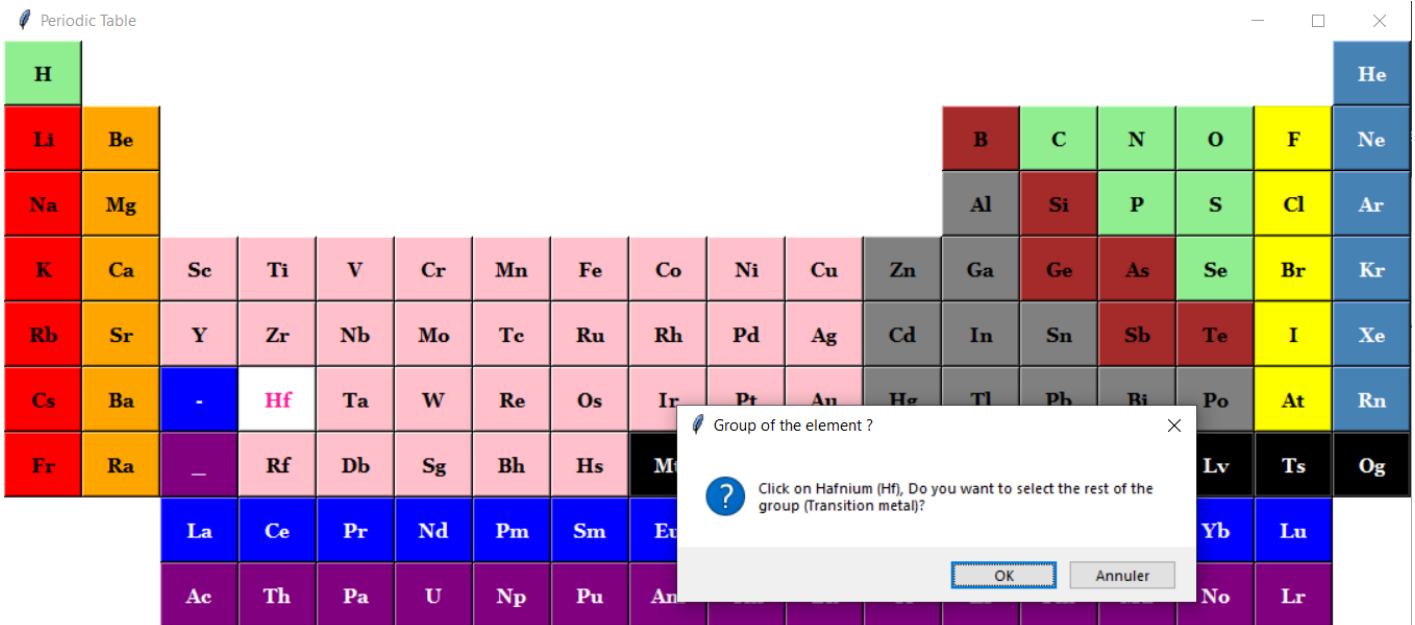
Description

Améliorations générales

Pour une utilisation "plus agréable" du tableau, j'ai apporté quelques améliorations au programme :

- La police des symboles à été changée. Les cases sont désormais colorées selon leur groupe, afin d'améliorer la lisibilité du tableau (pas forcément daltonien-friendly malheureusement)
- Il est désormais possible de sélectionner un groupe entier dans le tableau
- Lors de la sélection la ou les case(s) concernée(s) change(nt) de couleur de fond et de texte
- Deux cases "*placeholders*" ont été ajoutées pour indiquer la position des lanthanides et actinides dans le tableau.

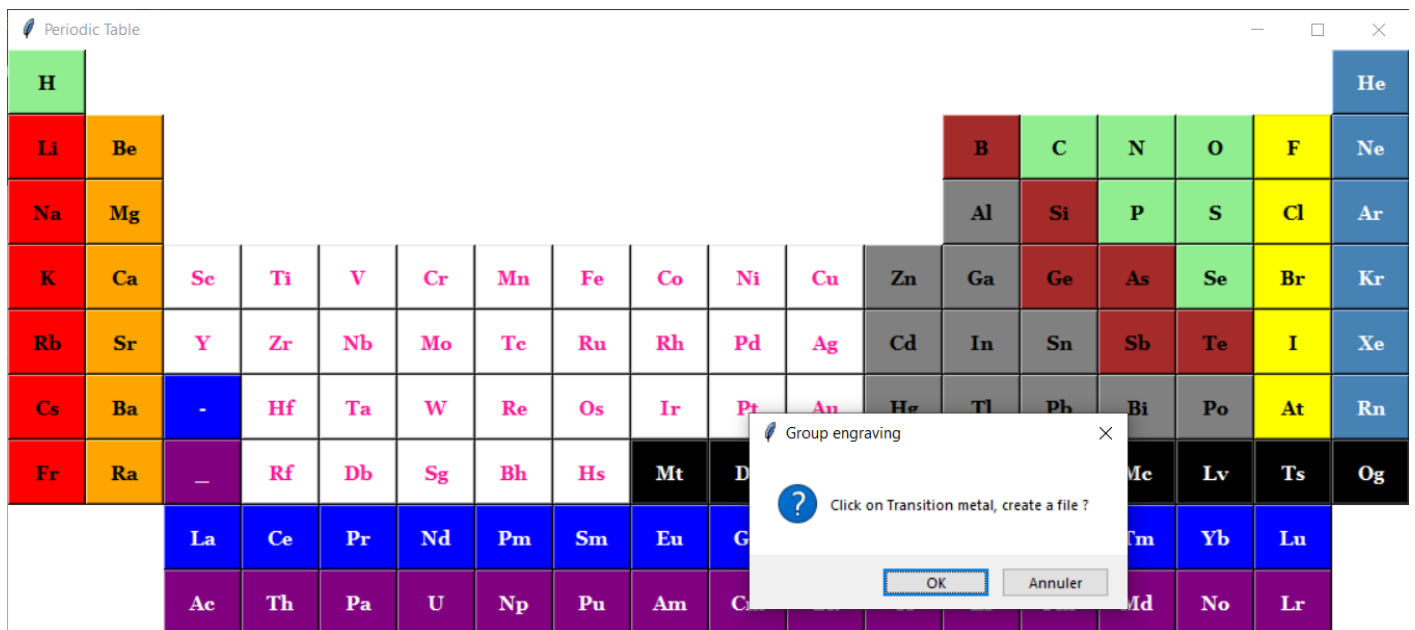
Le clic sur n'importe quelle case invoque un boîte de dialogue demandant si l'utilisateur.ice veut sélectionner l'ensemble du groupe. Si oui, le groupe entier est sélectionné et il est possible de demander la réalisation du .svg pour le tableau ; si non, il est possible de réaliser le .svg pour le [cube](#) de l'élément sélectionné.



Sélection de la case correspondant au Hafnium

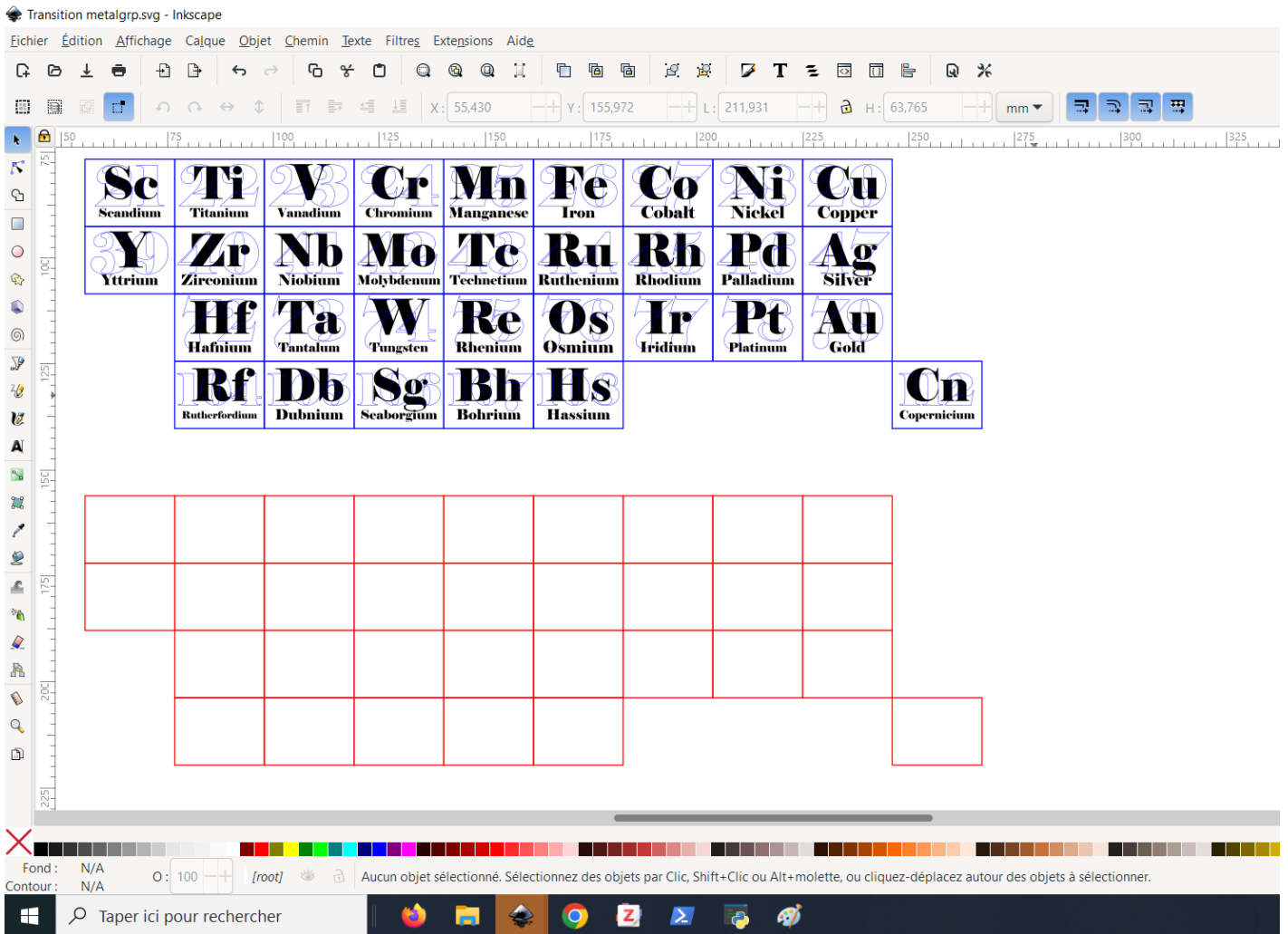
Sélection du groupe

Invoque une deuxième boîte de dialogue pour demander la réalisation du .svg pour le tableau.



Sélection du groupe des métaux de transition

Si oui, le fichier .svg est crée puis ouvert, et les cases gardent le fond blanc et le texte en couleur.



Fichier .svg résultant.

Les cases en rouges sont le pattern de coupe pour le passage au laser. Il faut toutes les sélectionner et faire "union" dans le menu "chemin", puis remodifier les paramètres de contours et de fond. Il est aussi nécessaire de transformer les cases du tableau en chemin pour éviter de potentielles pertes d'information en passant vers le logiciel de la machine. Par ailleurs, j'ai constaté que les numéros atomiques se finissant par "7" ne sont pas bien centrés dans les cases. Etant maniaque notoire, j'ai utilisé l'outil d'alignement de Inkscape pour corriger chaque occurrence.

Une autre amélioration du programme serai d'enlever les lignes bleues qui vont être recouvertes par les rouge (donc garder celles qui séparent deux cases uniquement), plutôt que de laisser les cases. En effet, la superposition cause un deuxième passage du laser. Celui-ci n'a pas d'incidence sur la forme du produit de gravure, mais participe à un échauffement évitable du matériau (mais ne risque pas non plus de faire exploser le fablab).

Sélection d'un seul élément

Si l'utilisateur.ice ne souhaite pas sélectionner le groupe il peut faire le cube de l'élément sur lequel iel à cliqué. Ce programme fait ouvre un autre terminal et fait appel à une version adaptée du programme utilisé pour les [cubes périodiques](#). La case sélectionnée apparait rouge sur le nouveau tableau créé. Il est ensuite possible d'interagir avec ce tableau comme celui des cubes périodiques.

C:\windows\System32\WindowsPowerShell\v1.0\powershell.exe

```

Hf
Clic sur : Hf
Hafnium
Configuration électronique : [['1s', 2], ['2s', 2], ['2p', 6], ['3s', 2], ['3p', 6], ['4s', 2], ['3d', 10], ['4p', 6], ['5s', 2], ['4d', 10], ['5p', 6], ['6s', 2], ['4f', 14], ['5d', 2]]
Population : [2, 8, 18, 32, 10, 2, 0]
Configuration électronique : [['1s', 2], ['2s', 2], ['2p', 6], ['3s', 2], ['3p', 6], ['4s', 2], ['3d', 10], ['4p', 6], ['5s', 2], ['4d', 10], ['5p', 6], ['6s', 2], ['4f', 14], ['5d', 2]]

```

Tableau Périodique

H																	He
Li	Be											B	C	N	O	F	Ne
Na	Mg											Al	Si	P	S	Cl	Ar
K	Ca	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Cu	Zn	Ga	Ge	As	Se	Br	Kr
Rb	Sr	Y	Zr	Nb	Mo	Tc	Ru	Rh	Pd	Ag	Cd	In	Sn	Sb	Te	I	Xe
Cs	Ba		Hf	Ta	W	Re	Os	Ir	Pt	Au	Hg	Tl	Pb	Bi	Po	At	Rn
Fr	Ra		Rf	Db	Sg	Bh	Hs	Mt	Ds	Rg	Cn	Nh	Fl	Mc	Lv	Ts	Og
		La	Ce	Pr	Nd	Pm	Sm	Eu	Gd	Tb	Dy	Ho	Er	Tm	Yb	Lu	
		Ac	Th	Pa	U	Np	Pu	Am	Cm	Bk	Cf	Es	Fm	Md	No	Lr	

Hafnium (Hf)

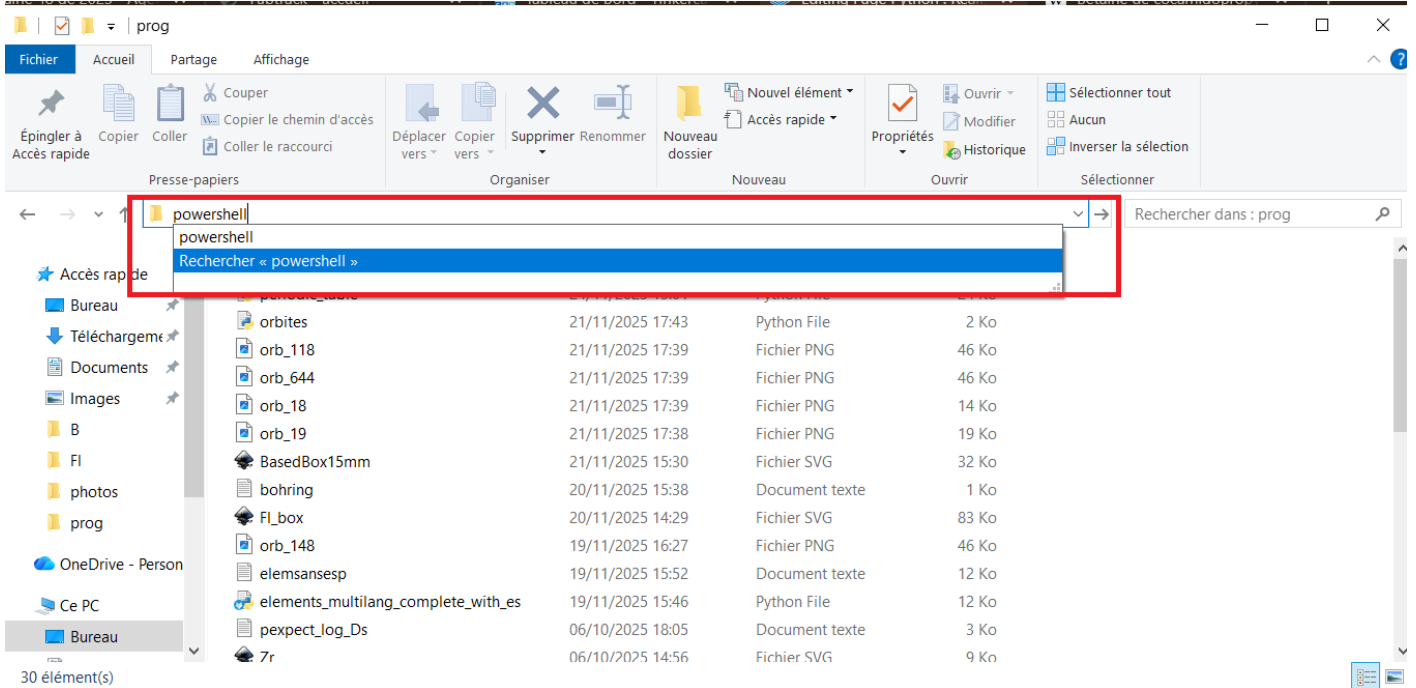
Hf Hafnium	
	Hafnium Hafnium hafnium hafnium Гафний Hafnio
Number 72	

Fait avec Python

Utilisation

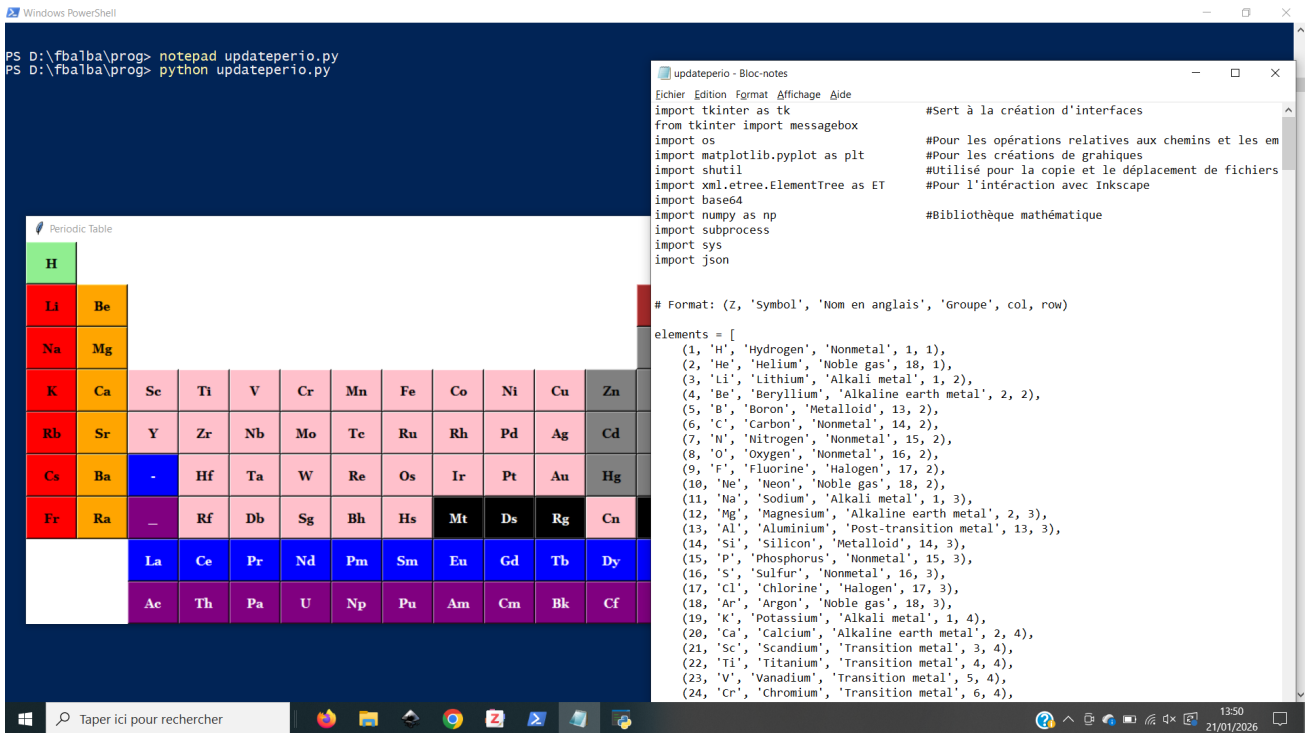
L'utilisation est identique à celle du programme des [cubes périodiques](#).

Sur windows, dans le dossier contenant le programme taper "powershell" dans la barre au dessus des fichiers.



Une fois dans le powershell, :

- "notepad" + [nomdufichier].py permet d'ouvrir le programme avec le bloc note
- "python" + [nomdufichier].py permet d'exécuter le programme (j'ai travaillé avec python 3.13.5)



Si une bibliothèque utilisée dans le programme n'est pas installée, le programme renverra une erreur. Pour l'installer, il suffit de taper "pip install" + [nom de la bibliothèque] dans le powershell pour lancer l'installation. Les bibliothèques utilisées ici sont très connues donc pas de risques d'installer des malwares.

Programmes

(Dispo en pièces jointes)

Updateperio.py : programme principal

Ligne 347 à adapter pour les destinations de fichiers

(Regardez pas trop le prog fait un peu mal aux yeux)

```
import tkinter as tk    ##Sert à la création d'interfaces
from tkinter import messagebox
import os###Pour les opérations relatives aux chemins et les emplacements des fichiers
import matplotlib.pyplot as plt##Pour les créations de graphiques
import shutil    ##Utilisé pour la copie et le déplacement de fichiers
import xml.etree.ElementTree as ET##Pour l'interaction avec Inkscape
import base64
import numpy as np###Bibliothèque mathématique
import subprocess
import sys
import json

# Format: (Z, 'Symbol', 'Nom en anglais', 'Groupe', col, row)

elements = [
    (1, 'H', 'Hydrogen', 'Nonmetal', 1, 1),
    (2, 'He', 'Helium', 'Noble gas', 18, 1),
    (3, 'Li', 'Lithium', 'Alkali metal', 1, 2),
    (4, 'Be', 'Beryllium', 'Alkaline earth metal', 2, 2),
    (5, 'B', 'Boron', 'Metalloid', 13, 2),
    (6, 'C', 'Carbon', 'Nonmetal', 14, 2),
    (7, 'N', 'Nitrogen', 'Nonmetal', 15, 2),
    (8, 'O', 'Oxygen', 'Nonmetal', 16, 2),
    (9, 'F', 'Fluorine', 'Halogen', 17, 2),
    (10, 'Ne', 'Neon', 'Noble gas', 18, 2),
```

(11, 'Na', 'Sodium', 'Alkali metal', 1, 3),
(12, 'Mg', 'Magnesium', 'Alkaline earth metal', 2, 3),
(13, 'Al', 'Aluminium', 'Post-transition metal', 13, 3),
(14, 'Si', 'Silicon', 'Metalloid', 14, 3),
(15, 'P', 'Phosphorus', 'Nonmetal', 15, 3),
(16, 'S', 'Sulfur', 'Nonmetal', 16, 3),
(17, 'Cl', 'Chlorine', 'Halogen', 17, 3),
(18, 'Ar', 'Argon', 'Noble gas', 18, 3),
(19, 'K', 'Potassium', 'Alkali metal', 1, 4),
(20, 'Ca', 'Calcium', 'Alkaline earth metal', 2, 4),
(21, 'Sc', 'Scandium', 'Transition metal', 3, 4),
(22, 'Ti', 'Titanium', 'Transition metal', 4, 4),
(23, 'V', 'Vanadium', 'Transition metal', 5, 4),
(24, 'Cr', 'Chromium', 'Transition metal', 6, 4),
(25, 'Mn', 'Manganese', 'Transition metal', 7, 4),
(26, 'Fe', 'Iron', 'Transition metal', 8, 4),
(27, 'Co', 'Cobalt', 'Transition metal', 9, 4),
(28, 'Ni', 'Nickel', 'Transition metal', 10, 4),
(29, 'Cu', 'Copper', 'Transition metal', 11, 4),
(30, 'Zn', 'Zinc', 'Post-transition metal', 12, 4),
(31, 'Ga', 'Gallium', 'Post-transition metal', 13, 4),
(32, 'Ge', 'Germanium', 'Metalloid', 14, 4),
(33, 'As', 'Arsenic', 'Metalloid', 15, 4),
(34, 'Se', 'Selenium', 'Nonmetal', 16, 4),
(35, 'Br', 'Bromine', 'Halogen', 17, 4),
(36, 'Kr', 'Krypton', 'Noble gas', 18, 4),
(37, 'Rb', 'Rubidium', 'Alkali metal', 1, 5),
(38, 'Sr', 'Strontium', 'Alkaline earth metal', 2, 5),
(39, 'Y', 'Yttrium', 'Transition metal', 3, 5),
(40, 'Zr', 'Zirconium', 'Transition metal', 4, 5),
(41, 'Nb', 'Niobium', 'Transition metal', 5, 5),
(42, 'Mo', 'Molybdenum', 'Transition metal', 6, 5),
(43, 'Tc', 'Technetium', 'Transition metal', 7, 5),
(44, 'Ru', 'Ruthenium', 'Transition metal', 8, 5),
(45, 'Rh', 'Rhodium', 'Transition metal', 9, 5),
(46, 'Pd', 'Palladium', 'Transition metal', 10, 5),
(47, 'Ag', 'Silver', 'Transition metal', 11, 5),
(48, 'Cd', 'Cadmium', 'Post-transition metal', 12, 5),
(49, 'In', 'Indium', 'Post-transition metal', 13, 5),
(50, 'Sn', 'Tin', 'Post-transition metal', 14, 5),


```

def __init__(self, master):
    #Fonction qui se lance automatiquement lors de la création de l'objet
    self.master = master
    #Permet de combiner la fenêtre tkinter à l'objet
    self.last_clicked = None
    #Création d'une variable pour stocker le dernier click effectué
    for (num, sym, name, fam, col, row) in elements:
        #Parcourt la liste des éléments
        btn = tk.Button(master, text=sym, width=5, height=2, font=("Georgia", 11, "bold"), command=lambda s=sym:
self.on_click(s))
        #Paramètres d'affichage du bouton
        btn.grid(row=row, column=col)
        #Grille des boutons avec les coordonnées
        buttons[sym] = btn
    for (num, sym, name, fam, *_ ) in elements :
        #Donne une couleur au bouton selon le groupe de l'élément
        if fam == "Halogen" :
            buttons[sym].config(bg="yellow")
            buttons[sym].config(fg="black")
        elif fam == "Nonmetal" :
            buttons[sym].config(bg="light green")
            buttons[sym].config(fg="black")
        elif fam == "Transition metal" :
            buttons[sym].config(bg="pink")
            buttons[sym].config(fg="black")
        elif fam == "Noble gas" :
            buttons[sym].config(bg="steelblue")
            buttons[sym].config(fg="white")
        elif fam == "Post-transition metal" :
            buttons[sym].config(bg="gray")
            buttons[sym].config(fg="black")
        elif fam == "Metalloid" :
            buttons[sym].config(bg="brown")
            buttons[sym].config(fg="black")
        elif fam == "Alkali metal" :
            buttons[sym].config(bg="red")
            buttons[sym].config(fg="black")
        elif fam == "Alkaline earth metal" :
            buttons[sym].config(bg="orange")
            buttons[sym].config(fg="black")
        elif fam == "Lanthanide" :
            buttons[sym].config(bg="blue")
            buttons[sym].config(fg="white")
        elif fam == "Actinide" :
            buttons[sym].config(bg="purple")
            buttons[sym].config(fg="white")
        elif fam == "none":

```

```
        buttons[sym].config(fg="white")
```

```
        buttons[sym].config(bg="black")
```

```
    
```

```
def on_click(self, symbol):  
    #Fonction qui s'active lors d'un click sur un bouton
```

```
    famille=""
```

```
    for i in range(len(elements)) :  
        #Parcourt la liste d'éléments
```

```
        if symbol in elements[i] and len(symbol)==len(elements[i][1]) :  
            
```

```
            famille=elements[i][3]
```

```
            name=elements[i][2]
```

```
            num=elements[i][0]
```

```
        buttons[symbol].config(bg="white")  
        #Le bouton sélectionné devient blanc
```

```
    if famille == "Halogen" :
```

```
        buttons[symbol].config(fg="gold")
```

```
    elif famille == "Nonmetal" :
```

```
        buttons[symbol].config(fg="green")
```

```
    elif famille == "Transition metal" :
```

```
        buttons[symbol].config(fg="deeppink")
```

```
    elif famille == "Noble gas" :
```

```
        buttons[symbol].config(fg="steelblue")
```

```
    elif famille == "Post-transition metal" :
```

```
        buttons[symbol].config(fg="dimgray")
```

```
    elif famille == "Metalloid" :
```

```
        buttons[symbol].config(fg="brown")
```

```
    elif famille == "Alkali metal" :
```

```
        buttons[symbol].config(fg="red")
```

```
    elif famille == "Alkaline earth metal" :
```

```
        buttons[symbol].config(fg="darkorange")
```

```
    elif famille == "Lanthanide" :
```

```
        buttons[symbol].config(fg="blue")
```

```
    elif famille == "Actinide" :
```

```
        buttons[symbol].config(fg="purple")
```

```
    elif famille == "none":
```

```
        buttons[symbol].config(fg="black")
```

```

# buttons[symbol].config(fg="black")           #Le texte du bouton sélectionné devient noir
self.last_clicked = symbol #Stocke le symbole dans la variable
print("Clic sur :", name, "-", symbol, "-", num, f"({famille})") #Affiche dans le terminal l'élément cliqué (symbol

reponse = messagebox.askokcancel("Group of the element ?", f"Click on {name} ({symbol}), Do you want to
select the rest of the group ({famille})?")
if reponse is True :
    for (num, sym, name, fam, *_ ) in elements :
        if fam == famille : #Sélection du groupe entier
            buttons[sym].config(bg="white")
            if famille == "Halogen" :
                buttons[sym].config(fg="gold")
            elif famille == "Nonmetal" :
                buttons[sym].config(fg="green")
            elif famille == "Transition metal" :
                buttons[sym].config(fg="deeppink")
            elif famille == "Noble gas" :
                buttons[sym].config(fg="steelblue")
            elif famille == "Post-transition metal" :
                buttons[sym].config(fg="dimgray")
            elif famille == "Metalloid" :
                buttons[sym].config(fg="brown")
            elif famille == "Alkali metal" :
                buttons[sym].config(fg="red")
            elif famille == "Alkaline earth metal" :
                buttons[sym].config(fg="darkorange")
            elif famille == "Lanthanide" :
                buttons[sym].config(fg="blue")
            elif famille == "Actinide" :
                buttons[sym].config(fg="purple")
            elif famille == "none":
                buttons[sym].config(fg="black")

# buttons[sym].config(fg="black")

```

```

print(f"Selecting the {famille} group")
reponse2 = messagebox.askokcancel("Group engraving", f"Click on {famille}, create a file ?")
if reponse2 is True :

    list_col=[] for i in range(18)]
    for (num, sym, name, fam, col, row) in elements :
        if fam == famille :#Boucle pour la création d'une liste regroupant les infos du groupe sélectionné
            if sym == "-" or sym == "_":
                continue #Exclut les placholders s'il y en a
                list_data=[]
                list_data.append(sym)
                list_data.append(num)
                list_data.append(name)
                list_data.append(fam)
                list_data.append(col)
                list_data.append(row)
                list_col[col-1].append(list_data)
            print(list_col)

#Création du .svg
width_mm = 600
height_mm = 300
square_size=80
square_sizeh=60
stroke_width=1
svg_content = f'<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" width="600mm" height="300mm">

'''

for i in range(len(list_col)):
    for j in range(len(list_col[i])):
        x = 50 + i * square_size # Each square starts exactly where previous ends
        y = 50 + list_col[i][j][5] * square_sizeh
        svg_content += f' <rect x="{x}" y="{y}" width="{square_size}" height="{square_sizeh}" fill="none"
stroke="blue" stroke-width="{stroke_width}"/>'
        svg_content += f' <rect x="{x}" y="{y+700}" width="{square_size}" height="{square_sizeh}" fill="none"
stroke="red" stroke-width="{stroke_width}"/>'
        text_x=x+square_size/2

```

```
text_y=y+square_size/2
```

```
text_content_sym = str(list_col[i][j][0])
```

```
if len(str(list_col[i][j][0]))==1:
```

```
font_size = square_size * 0.55
```

```
if len(str(list_col[i][j][0]))==2:
```

```
font_size = square_size * 0.50
```

```
svg_content += f' <text x="{text_x-0.1}" y="{text_y-4}"
```

```
text-anchor="middle" dominant-baseline="middle"
```

```
font-size="{font_size}" font-family="Elephant" fill="black"> {text_content_sym}</text>
```

```
...
```

```
text_content_num = str(list_col[i][j][1])
```

```
if len(str(list_col[i][j][1]))==1:
```

```
font_size = square_size * 0.68
```

```
if len(str(list_col[i][j][1]))==2:
```

```
font_size = square_size * 0.63
```

```
if len(str(list_col[i][j][1]))==3:
```

```
font_size = square_size * 0.48
```

```
svg_content += f' <text x="{text_x}" y="{text_y}"
```

```
text-anchor="middle" dominant-baseline="middle"
```

```
font-size="{font_size}" font-family="Elephant" fill="none" stroke="blue" stroke-width="{stroke_width * 0.3}"> {text_content_num}</text>
```

```
text_content_name = str(list_col[i][j][2])
```

```
...
```

```
text_content_name = str(list_col[i][j][2])
```

```
if len(str(list_col[i][j][2]))>4 and len(str(list_col[i][j][2]))<9:
```

```
font_size = (square_size * 0.2)/(len(str(list_col[i][j][2]))*0.18)
```

```
elif len(str(list_col[i][j][2]))<5 :
```

```
font_size = (square_size * 0.2)/(len(str(list_col[i][j][2]))*0.3)
```

```
elif len(str(list_col[i][j][2]))>5 :
```

```
font_size = (square_size * 0.2)/(len(str(list_col[i][j][2]))*0.14)
```

```
svg_content += f' <text x="{text_x-0.1}" y="{text_y+19}"
```

```
text-anchor="middle" dominant-baseline="middle"
```

```
font-size="{font_size}" font-family="Elephant" fill="black"> {text_content_name}</text>
```

```
'''
```

```
svg_content += '</svg>'
```

```
with open(f'{famille}grp.svg', 'w') as f:
```

```
f.write(svg_content)
```

```
print(f"SVG created: {famille}grp")
```

```
os.startfile(f"D:/fbalba/prog/{famille}grp.svg") #Ouvre le .svg
```

```
if reponse2 is False : #Pour que les boutons récupèrent leur couleur originale s'ils ne sont plus sélectionné
```

```
for (num, sym, name, fam, *_ ) in elements :
```

```
if fam == famille :
```

```
if famille == "Halogen" :
```

```
buttons[sym].config(bg="yellow")
```

```
buttons[sym].config(fg="black")
```

```
elif famille == "Nonmetal" :
```

```
buttons[sym].config(bg="light green")
```

```
buttons[sym].config(fg="black")
```

```
elif famille == "Transition metal" :
```

```
buttons[sym].config(bg="pink")
```

```
buttons[sym].config(fg="black")
```

```
elif famille == "Noble gas" :
```

```
buttons[sym].config(bg="steelblue")
```

```
buttons[sym].config(fg="white")
```

```
elif famille == "Post-transition metal" :
```

```
buttons[sym].config(bg="gray")
```

```
buttons[sym].config(fg="black")
```

```
elif famille == "Metalloid" :
```

```
buttons[sym].config(bg="brown")
```

```
buttons[sym].config(fg="black")
```

```
elif famille == "Alkali metal" :
```

```
buttons[sym].config(bg="red")
```

```
buttons[sym].config(fg="black")
```

```
elif famille == "Alkaline earth metal" :
```

```
buttons[sym].config(bg="orange")
```

```
buttons[sym].config(fg="black")
```

```
elif famille == "Lanthanide" :
```

```
buttons[sym].config(bg="blue")
```

```

buttons[sym].config(fg="white")
elif famille == "Actinide" :
buttons[sym].config(bg="purple")
buttons[sym].config(fg="white")
elif famille == "none":
buttons[sym].config(fg="white")
buttons[sym].config(bg="black")

if reponse is False :
reponse3 = messagebox.askokcancel("Element engraving", f"Click on {name} ({symbol}), create a file ?")

if reponse3 is True :
JSON_PATH = f"{symbol}.json"#Crée un fichier .json
with open(JSON_PATH, "w", encoding="utf-8") as f:#Rédige le .json avec la case à cliquer
json.dump({"symbol": symbol}, f, ensure_ascii=False, indent=2)
subprocess.Popen(["start", "powershell", "-NoExit", "-Command", "python updaterunna.py"], shell=True)
# la ligne précédente ouvre un autre powershell et lance updaterunna.py dans celui-ci. Ce programme lit
ensuite le .json le plus récent du dossier pour cliquer la case voulue.

if reponse3 is False :#Pour que les boutons récupèrent leur couleur originale s'ils ne sont plus sélectionnés
if famille == "Halogen" :
buttons[symbol].config(bg="yellow")
buttons[symbol].config(fg="black")
elif famille == "Nonmetal" :
buttons[symbol].config(bg="light green")
buttons[symbol].config(fg="black")
elif famille == "Transition metal" :
buttons[symbol].config(bg="pink")
buttons[symbol].config(fg="black")
elif famille == "Noble gas" :
buttons[symbol].config(bg="steelblue")
buttons[symbol].config(fg="white")
elif famille == "Post-transition metal" :
buttons[symbol].config(bg="gray")
buttons[symbol].config(fg="black")
elif famille == "Metalloid" :
buttons[symbol].config(bg="brown")
buttons[symbol].config(fg="black")

```

```

elif famille == "Alkali metal" :
    buttons[symbol].config(bg="red")
    buttons[symbol].config(fg="black")
elif famille == "Alkaline earth metal" :
    buttons[symbol].config(bg="orange")
    buttons[symbol].config(fg="black")
elif famille == "Lanthanide" :
    buttons[symbol].config(bg="blue")
    buttons[symbol].config(fg="white")
elif famille == "Actinide" :
    buttons[symbol].config(bg="purple")
    buttons[symbol].config(fg="white")
elif famille == "none":
    buttons[symbol].config(fg="white")
    buttons[symbol].config(bg="black")

root = tk.Tk()#Permet de créer la fenêtre
root.configure(bg="white")
root.title("Periodic Table")#Nom de la fenêtre
app = PeriodicTableApp(root)#Permet à l'objet PeriodicTableApp de s'afficher dans la fenêtre
root.mainloop()#Fait fonctionner la fenêtre jusqu'à sa fermeture

```

Updaterunna.py : Pour les cubes

lignes 298, 300, 302, 413 et 415 à adapter

```

import tkinter as tk #Sert à la création d'interfaces
from tkinter import messagebox
import os#Pour les opérations relatives aux chemins et les emplacements des fichiers
import matplotlib.pyplot as plt#Pour les créations de graphiques
import shutil #Utilisé pour la copie et le déplacement de fichiers
import xml.etree.ElementTree as ET#Pour l'interaction avec Inkscape
import base64
import numpy as np#Bibliothèque mathématique
import sys

```

```
import json
import glob
import time
```

liste des 118 éléments avec Numéro, Symbole, Noms en anglais, français, chinois, russe et espagnol. Les deux nombres ensuite sont les coordonnées dans le tableau.

```
elements = [
    (1, "H", ("Hydrogen", "Hydrogène", "☐", "ن ه ي ج و ر د ي ه ل", "Водород", "Hidrógeno"), 1, 1),
    (2, "He", ("Helium", "Hélium", "☐", "م و ي ل ي ه ل", "Гелий", "Helio"), 18, 1),
    (3, "Li", ("Lithium", "Lithium", "☐", "م و ي ث ي ل ل", "Литий", "Litio"), 1, 2),
    (4, "Be", ("Beryllium", "Béryllium", "☐", "م و ي ل ي ر ب ل", "Бериллий", "Berilio"), 2, 2),
    (5, "B", ("Boron", "Bore", "☐", "ن و ر و ب ل", "Бор", "Boro"), 13, 2),
    (6, "C", ("Carbon", "Carbone", "☐", "ن و ب ر ك ل", "Углерод", "Carbono"), 14, 2),
    (7, "N", ("Nitrogen", "Azote", "☐", "ن ه ي ج و ر ت ي ل", "Азот", "Nitrógeno"), 15, 2),
    (8, "O", ("Oxygen", "Oxygène", "☐", "ن ه ي ج س ك أ ل", "Кислород", "Oxígeno"), 16, 2),
    (9, "F", ("Fluorine", "Fluor", "☐", "ر و ل ف ل", "Фтор", "Flúor"), 17, 2),
    (10, "Ne", ("Neon", "Néon", "☐", "ن و ي ن ل", "Неон", "Neón"), 18, 2),
    (11, "Na", ("Sodium", "Sodium", "☐", "م و ي د و ص ل", "Натрий", "Sodio"), 1, 3),
    (12, "Mg", ("Magnesium", "Magnésium", "☐", "م و ي س ي ن غ م ل", "Магний", "Magnesio"), 2, 3),
    (13, "Al", ("Aluminum", "Aluminium", "☐", "م و ي ن م و ل أ ل", "Алюминий", "Aluminio"), 13, 3),
    (14, "Si", ("Silicon", "Silicium", "☐", "ن و ك ي ل ي س ل", "Кремний", "Silicio"), 14, 3),
    (15, "P", ("Phosphorus", "Phosphore", "☐", "ر و ف س و ف ل", "Фосфор", "Fósforo"), 15, 3),
    (16, "S", ("Sulfur", "Soufre", "☐", "ت ي ر ب ك ل", "Серa", "Azufre"), 16, 3),
    (17, "Cl", ("Chlorine", "Chlore", "☐", "ر و ل ك ل", "Хлор", "Cloro"), 17, 3),
    (18, "Ar", ("Argon", "Argon", "☐", "ن و ج ر أ ل", "Аргон", "Argón"), 18, 3),
    (19, "K", ("Potassium", "Potassium", "☐", "م و ي س ا ت و ب ل", "Калий", "Potasio"), 1, 4),
    (20, "Ca", ("Calcium", "Calcium", "☐", "م و ي س ل ك ل", "Кальций", "Calcio"), 2, 4),
    (21, "Sc", ("Scandium", "Scandium", "☐", "م و ي د ن ك س إ ل", "Скандий", "Escandio"), 3, 4),
    (22, "Ti", ("Titanium", "Titane", "☐", "م و ي ن ا ت ي ل", "Титан", "Titanio"), 4, 4),
    (23, "V", ("Vanadium", "Vanadium", "☐", "م و ي د ا ن أ ل", "Ванадий", "Vanadio"), 5, 4),
    (24, "Cr", ("Chromium", "Chrome", "☐", "م و ر ك ل", "Хром", "Cromo"), 6, 4),
    (25, "Mn", ("Manganese", "Manganèse", "☐", "ز ي ن غ ن م ل", "Марганец", "Manganeso"), 7, 4),
    (26, "Fe", ("Iron", "Fer", "☐", "د ي ح ل", "Железо", "Hierro"), 8, 4),
    (27, "Co", ("Cobalt", "Cobalt", "☐", "ت ل ا ب و ك ل", "Кобальт", "Cobalto"), 9, 4),
    (28, "Ni", ("Nickel", "Nickel", "☐", "ن ي ك ل", "Никель", "Níquel"), 10, 4),
    (29, "Cu", ("Copper", "Cuivre", "☐", "س ا ح ن ل", "Медь", "Cobre"), 11, 4),
    (30, "Zn", ("Zinc", "Zinc", "☐", "ن ك ز ل", "Цинк", "Zinc"), 12, 4),
    (31, "Ga", ("Gallium", "Gallium", "☐", "م و ي ل ا غ ل", "Галлий", "Galio"), 13, 4),
```

- (32, "Ge", ("Germanium", "Germanium", "□", "موي ن ا م ر ج ل", "Германий", "Germanio"), 14, 4),
- (33, "As", ("Arsenic", "Arsenic", "□", "مخي ن ر ز ل ا", "Мышьяк", "Arsénico"), 15, 4),
- (34, "Se", ("Selenium", "Sélénium", "□", "موي ن ي ل ي س ل", "Селен", "Selenio"), 16, 4),
- (35, "Br", ("Bromine", "Brome", "□", "مور ب ل ا", "Бром", "Bromo"), 17, 4),
- (36, "Kr", ("Krypton", "Krypton", "□", "نوت ب ي ر ك ل ا", "Криптон", "Kriptón"), 18, 4),
- (37, "Rb", ("Rubidium", "Rubidium", "□", "موي دي ب و ر ل ا", "Рубидий", "Rubidio"), 1, 5),
- (38, "Sr", ("Strontium", "Strontium", "□", "موي ش ن و رت س ل ا", "Стронций", "Estroncio"), 2, 5),
- (39, "Y", ("Yttrium", "Yttrium", "□", "موي ر ت ي ل ا", "Иттрий", "Itrio"), 3, 5),
- (40, "Zr", ("Zirconium", "Zirconium", "□", "موي ن و ك ر ز ل ا", "Цирконий", "Circonio"), 4, 5),
- (41, "Nb", ("Niobium", "Niobium", "□", "موي ب و ي ن ل ا", "Ниобий", "Niobio"), 5, 5),
- (42, "Mo", ("Molybdenum", "Molybdène", "□", "مون ي د ب ي ل و م ل ا", "Молибден", "Molibdeno"), 6, 5),
- (43, "Tc", ("Technetium", "Technétium", "□", "موي ت ي ن ك ت ل ا", "Технеций", "Tecneccio"), 7, 5),
- (44, "Ru", ("Ruthenium", "Ruthénium", "□", "موي ن ي ث و ر ل ا", "Рутений", "Rutenio"), 8, 5),
- (45, "Rh", ("Rhodium", "Rhodium", "□", "موي دور ل ا", "Родий", "Rodio"), 9, 5),
- (46, "Pd", ("Palladium", "Palladium", "□", "موي د ا ل ب ل ا", "Палладий", "Paladio"), 10, 5),
- (47, "Ag", ("Silver", "Argent", "□", "موي ص ف ل ا", "Серебро", "Plata"), 11, 5),
- (48, "Cd", ("Cadmium", "Cadmium", "□", "موي م د ا ك ل ا", "Кадмий", "Cadmio"), 12, 5),
- (49, "In", ("Indium", "Indium", "□", "موي دن ا ل ا", "Индий", "Indio"), 13, 5),
- (50, "Sn", ("Tin", "Étain", "□", "موي ص ق ل ا", "Олово", "Estaño"), 14, 5),
- (51, "Sb", ("Antimony", "Antimoine", "□", "ن و م ي ن ا ل ا", "Сурьма", "Antimonio"), 15, 5),
- (52, "Te", ("Tellurium", "Tellure", "□", "موي ر و ل ي ل ا", "Теллур", "Telurio"), 16, 5),
- (53, "I", ("Iodine", "Iode", "□", "موي ل ا", "Йод", "Yodo"), 17, 5),
- (54, "Xe", ("Xenon", "Xénon", "□", "ن و ن ي ز ل ا", "Ксенон", "Xenón"), 18, 5),
- (55, "Cs", ("Cesium", "Césium", "□", "موي ز ي س ل ا", "Цезий", "Cesio"), 1, 6),
- (56, "Ba", ("Barium", "Baryum", "□", "موي ر ا ب ل ا", "Барий", "Bario"), 2, 6),
- # Lanthanides
- (57, "La", ("Lanthanum", "Lanthane", "□", "مون ا ث ن ا ل ل ا", "Лантан", "Lantano"), 3, 9),
- (58, "Ce", ("Cerium", "Cérium", "□", "موي ر ي س ل ا", "Церий", "Cerio"), 4, 9),
- (59, "Pr", ("Praseodymium", "Praséodyme", "□", "موي م ي د و ي س ا ر ب ل ا", "Празеодим", "Praseodimio"), 5, 9),
- (60, "Nd", ("Neodymium", "Néodyme", "□", "موي م ي د و ي ن ل ا", "Неодим", "Neodimio"), 6, 9),
- (61, "Pm", ("Promethium", "Prométhium", "□", "موي ث ي مور ب ل ا", "Прометий", "Prometio"), 7, 9),
- (62, "Sm", ("Samarium", "Samarium", "□", "موي ر ا م ا س ل ا", "Самарий", "Samario"), 8, 9),
- (63, "Eu", ("Europium", "Europium", "□", "موي ب و ر و ي ل ا", "Европий", "Europio"), 9, 9),
- (64, "Gd", ("Gadolinium", "Gadolinium", "□", "موي ن ي ل و د ا غ ل ا", "Гадолиний", "Gadolinio"), 10, 9),
- (65, "Tb", ("Terbium", "Terbium", "□", "موي ب ر ي ت ل ا", "Тербий", "Terbio"), 11, 9),
- (66, "Dy", ("Dysprosium", "Dysprosium", "□", "موي س و ر ب س ي د ل ا", "Диспрозий", "Dispro시오"), 12, 9),
- (67, "Ho", ("Holmium", "Holmium", "□", "موي م ل و ه ل ا", "Гольмий", "Holmio"), 13, 9),
- (68, "Er", ("Erbium", "Erbium", "□", "موي ب ر ل ل ا", "Эрбий", "Erbio"), 14, 9),
- (69, "Tm", ("Thulium", "Thulium", "□", "موي ل و ث ل ا", "Туллий", "Tulio"), 15, 9),
- (70, "Yb", ("Ytterbium", "Ytterbium", "□", "موي ب ر ت ي ل ا", "Иттербий", "Iterbio"), 16, 9),

(71, "Lu", ("Lutetium", "Lutécium", "☐", "موي تي ولول", "Лютеций", "Lutecio"), 17, 9),

(72, "Hf", ("Hafnium", "Hafnium", "☐", "موي ن فاهال", "Гафний", "Hafnio"), 4, 6),

(73, "Ta", ("Tantalum", "Tantale", "☐", "مولات ننتال", "Тантал", "Tantalio"), 5, 6),

(74, "W", ("Tungsten", "Tungstène", "☐", "ننتس غنتال", "Вольфрам", "Wolframio"), 6, 6),

(75, "Re", ("Rhenium", "Rhénium", "☐", "موي ني رل", "Рений", "Renio"), 7, 6),

(76, "Os", ("Osmium", "Osmium", "☐", "موي مزوأل", "Осмий", "Osmio"), 8, 6),

(77, "Ir", ("Iridium", "Iridium", "☐", "موي دي ري إل", "Иридий", "Iridio"), 9, 6),

(78, "Pt", ("Platinum", "Platine", "☐", "ني تال بل", "Платина", "Platino"), 10, 6),

(79, "Au", ("Gold", "Or", "☐", "به ذل", "Золото", "Oro"), 11, 6),

(80, "Hg", ("Mercury", "Mercure", "☐", "قب ئزل", "Ртуть", "Mercurio"), 12, 6),

(81, "Tl", ("Thallium", "Thallium", "☐", "موي لاثل", "Таллий", "Talió"), 13, 6),

(82, "Pb", ("Lead", "Plomb", "☐", "ص اص رل", "Свинец", "Plomo"), 14, 6),

(83, "Bi", ("Bismuth", "Bismuth", "☐", "ت وم زبل", "Висмут", "Bismuto"), 15, 6),

(84, "Po", ("Polonium", "Polonium", "☐", "موي نول وبل", "Полоний", "Polonio"), 16, 6),

(85, "At", ("Astatine", "Astate", "☐", "ني تات سأل", "Астат", "Astato"), 17, 6),

(86, "Rn", ("Radon", "Radon", "☐", "نودارل", "Радон", "Radón"), 18, 6),

(87, "Fr", ("Francium", "Francium", "☐", "موي س نارفال", "Франций", "Francio"), 1, 7),

(88, "Ra", ("Radium", "Radium", "☐", "موي دارل", "Радий", "Radio"), 2, 7),

Actinides

(89, "Ac", ("Actinium", "Actinium", "☐", "موي ني ت كأل", "Актиний", "Actinio"), 3, 10),

(90, "Th", ("Thorium", "Thorium", "☐", "موي روثال", "Торий", "Torio"), 4, 10),

(91, "Pa", ("Protactinium", "Protactinium", "☐", "موي ني ت كات وربال", "Протактиний", "Protactinio"), 5, 10),

(92, "U", ("Uranium", "Uranium", "☐", "موي نارويال", "Уран", "Uranio"), 6, 10),

(93, "Np", ("Neptunium", "Neptunium", "☐", "موي نوت ببي لال", "Нептуний", "Neptunio"), 7, 10),

(94, "Pu", ("Plutonium", "Plutonium", "☐", "موي نوت ولبلال", "Плутоний", "Plutonio"), 8, 10),

(95, "Am", ("Americium", "Américium", "☐", "موي كي ري مأل", "Америций", "Americio"), 9, 10),

(96, "Cm", ("Curium", "Curium", "☐", "موي روكال", "Кюрий", "Curio"), 10, 10),

(97, "Bk", ("Berkelium", "Berkelium", "☐", "موي لي ك ربال", "Берклий", "Berkelio"), 11, 10),

(98, "Cf", ("Californium", "Californium", "☐", "موي نروف لكال", "Калифорний", "Californio"), 12, 10),

(99, "Es", ("Einsteinium", "Einsteinium", "☐", "موي ني تشن ش آل", "Эйнштейний", "Einsteinio"), 13, 10),

(100, "Fm", ("Fermium", "Fermium", "☐", "موي مرفال", "Фермий", "Fermio"), 14, 10),

(101, "Md", ("Mendelevium", "Mendélévium", "☐", "موي في لدن مال", "Менделевий", "Mendelevio"), 15, 10),

(102, "No", ("Nobelium", "Nobélium", "☐", "موي ل بلونال", "Нобелий", "Nobelio"), 16, 10),

(103, "Lr", ("Lawrencium", "Lawrencium", "☐", "موي سن رولال", "Лоуренсий", "Laurencio"), 17, 10),

(104, "Rf", ("Rutherfordium", "Rutherfordium", "☐☐", "موي دروف رذرال", "Резерфордий", "Rutherfordio"), 4, 7),

(105, "Db", ("Dubnium", "Dubnium", "☐☐", "موي ن ب ودل", "Дубний", "Dubnio"), 5, 7),

(106, "Sg", ("Seaborgium", "Seaborgium", "𐤎𐤍", "موي غروب سي ل ا", "Сиборгий", "Seaborgio"), 6, 7),
 (107, "Bh", ("Bohrium", "Bohrium", "𐤁𐤇", "موي ره و ب ل ا", "Борий", "Bohrio"), 7, 7),
 (108, "Hs", ("Hassium", "Hassium", "𐤇𐤎", "موي سه ل ا", "Хассий", "Hassio"), 8, 7),
 (109, "Mt", ("Meitnerium", "Meitnerium", "𐤍𐤕", "موي ريت ن ت م ل ا", "Мейтнерий", "Meitnerio"), 9, 7),
 (110, "Ds", ("Darmstadtium", "Darmstadtium", "𐤃𐤎", "موي ت ا ت ش م ر ا د", "Дармштадтий", "Darmstadtio"), 10, 7),
 (111, "Rg", ("Roentgenium", "Roentgenium", "𐤓𐤁", "موي ن ي ج ت ن و ر ل ا", "Рентгений", "Roentgenio"), 11, 7),
 (112, "Cn", ("Copernicium", "Copernicium", "𐤁𐤍", "موي سي ن ر ب و ك ل ل ا", "Коперниций", "Copernicio"), 12, 7),
 (113, "Nh", ("Nihonium", "Nihonium", "𐤍𐤇", "موي ن و ه ي ن ل ا", "Нихоний", "Nihonio"), 13, 7),
 (114, "Fl", ("Flerovium", "Flérovium", "𐤃𐤌", "موي ف و ر ي ل ف ل ا", "Флеровий", "Flerovio"), 14, 7),
 (115, "Mc", ("Moscovium", "Moscovium", "𐤍𐤁", "موي ف و ك س و م ل ا", "Московский", "Moscovio"), 15, 7),
 (116, "Lv", ("Livermorium", "Livermorium", "𐤌𐤅", "موي ر و م ر ف ي ل ل ل ا", "Ливерморий", "Livermorio"), 16, 7),
 (117, "Ts", ("Tennessee", "Tennessee", "𐤕𐤎", "ن ي س ي ن ي ت ل ا", "Теннессин", "Tenesino"), 17, 7),
 (118, "Og", ("Oganesson", "Oganesson", "𐤓𐤁", "ن و س ي ن ا ج و أ ل ا", "Оганессон", "Oganesón"), 18, 7)

]

```
def get_last_json(folder="."): # récupère le dernier .json crée dans le dossier
```

```
    # récupère tous les fichiers .json dans le dossier
```

```
    files = glob.glob(os.path.join(folder, "*.json"))
```

```
    if not files:
```

```
        return None # aucun fichier trouvé
```

```
    # trie par date de modification (du plus récent au plus ancien)
```

```
    files.sort(key=os.path.getmtime, reverse=True)
```

```
    return files[0] # le plus récent
```

```
## Donne la configuration électronique en fonction du nombre d'électron Z donné.
```

```
def
```

```
def Config(Z):
```

```
    [
```

```
    orbitales = ["1s", "2s", "2p", "3s", "3p", "4s", "3d", "4p", "5s", "4d", "5p", "6s", "4f", "5d", "6p", "7s", "5f", "6d", "7p"]
```

```

max_elec = {"s": 2,"p": 6,"d": 10,"f": 14} #nombre max d'électrons par couche

config = []
reste = Z

for orb in orbitales:
    #Parcourt la liste des orbitales
    l = orb[-1]
    #Identifie le type de sous-couche: s, p, d, f
    n = min(max_elec[l], reste)
    #Compte le nombre d'électrons dans la sous-couche
    config.append([f"{orb}",n])
    #Sauvegarde dans la liste config de l'orbitale et de son remplissage
    reste -= n
    #Retire le nombre d'électrons ajoutés à la sous-couche
    if reste <= 0:
        #Fin du parcourt des orbitales quand il n'y a plus d'électrons à répartir
        break

print('Configuration électronique :',config) #Affiche les résultats de configuration
return config
#Renvoie les résultats de configuration sous forme de liste

## Trace le modèle de Borh d'un atome, avec son nom et son symbole, à but illustratif

def traceur1(Z,name,symbol) :
    orbits = Config(Z)
    #Fait appel à la fonction Config() et stocke le résultat dans la variable orbits
    population = [0 for i in range(7)]
    #Crée une liste de 0 (Longueur de 7) (les couches de 1 à 7)
    for orb in orbits :
        #Parcourt la liste orbits
        n = int(orb[0][0])
        #Lit la couche orbitale "1" dans la notation "1s" par exemple
        population[n-1]+=orb[1]
        #Ajoute le nombre d'électrons dans la couche correspondante
    print('Population :',population)
    #Affiche la répartition de population par couche

    fig, ax = plt.subplots()
    #Création du graphique
    rayons=np.linspace(1,7,7,dtype=int)
    #Création d'une liste de 7 valeurs (pour les rayons des cercles)

    for r, n_e in zip(rayons, population):
        #Parcourt les couches de 1 à 7 et la liste des rayons en même temps
        if n_e!=0 :
            #Ne trace le cercle que si le nombre d'électrons est supérieur à 0

            cercle = plt.Circle((0, 0), r, fill=False)
            #Crée un cercle de rayon r et de centre 0,0
            ax.add_patch(cercle)
            #Ajoute le cercle sur le graphe

    angles = np.linspace(0, 2*np.pi, n_e, endpoint=False)
    #Calcule les angles pour répartir les n électrons de la
    couche sur le cercle

```

```

[]x = r * np.cos(angles)[]#Ces deux lignes placent les électrons sur les cercles
[]y = r * np.sin(angles)

[]ax.scatter(x, y, s=50) # s=taille du point

[]nucleus = plt.Circle((0, 0), 0.2, color=(0,0,1))[]#Crée le noyau
[]ax.add_patch(nucleus)[]#Ajoute le noyau

[]ax.set_aspect('equal', 'box')[]#Fait une boîte carrée pour l'affichage du graphique
[]ax.set_xlim(-max(rayons)-1, max(rayons)+1)[]#Fixe les limites de l'axe x
[]ax.set_ylim(-max(rayons)-1, max(rayons)+1)[]#Fixe les limites de l'axe y
[]ax.set_xticks([])[]#Enlève les graduations de l'axe x
[]ax.set_yticks([])[]#Enlève les graduations de l'axe y
[]plt.title(f"Modèle de Bohr - {name} ({symbol})")[]#Titre

[]plt.show(block=False)[]#Affiche le graphique dans une nouvelle fenêtre et ne bloque pas la suite de l'exécution
su programme

## Trace le modèle de Bohr d'un atome, et l'enregistre en png dans le dossier actif

def traceur2(Z,name) :
[]orbits = Config(Z)
[]population = [0 for i in range(7)]
[]for orb in orbits :
[]n = int(orb[0][0])
[]population[n-1]+=orb[1]
[]fig, ax = plt.subplots()
[]rayons=np.linspace(1,7,7,dtype=int)

[]for r, n_e in zip(rayons, population):
[]if n_e!=0 :
[]
[]cercle = plt.Circle((0, 0), r, fill=False)
[]ax.add_patch(cercle)

[]angles = np.linspace(0, 2*np.pi, n_e, endpoint=False)
[]x = r * np.cos(angles)

```

```

y = r * np.sin(angles)

ax.scatter(x, y, s=50) # s=taille du point

nucleus = plt.Circle((0, 0), 0.2, color=(0,0,1))
ax.add_patch(nucleus)

ax.set_aspect('equal', 'box')
plt.axis('off')#Enlève les axes
ax.set_xlim(-max(rayons)-1, max(rayons)+1)
ax.set_ylim(-max(rayons)-1, max(rayons)+1)
ax.set_xticks([])
ax.set_yticks([])

plt.savefig(f'orb_{name}')#Sauvegarde la figure sous le nom 'orb_[nom de l'atome]'
plt.close()#Ferme la figure après l'enregistrement
[]
[]

buttons = {}
class PeriodicTableApp:#Création d'un objet
def __init__(self, master):#Fonction qui se lance automatiquement lors de la création de l'objet
self.master = master#Permet de combiner la fenêtre tkinter à l'objet
self.last_clicked = None #Création d'une variable pour stocker le dernier click effectué
[]
[]
for (num, sym, name, col, row) in elements: #Parcourt la liste des éléments
btn = tk.Button(master, text=sym, width=5, height=2, font=("Georgia", 11, "bold"),
command=lambda s=sym: self.on_click(s))#Paramètres d'affichage du bouton
btn.grid(row=row, column=col)#Grille des boutons avec les coordonnées
buttons[sym] = btn

#sym_clicked = sys.argv[1] if len(sys.argv) > 1 else None

def on_click(self, symbol):#Fonction qui s'active lors d'un click sur un bouton
buttons[symbol].config(bg="red")
name=""#Crée une variable string vide

```

```

self.last_clicked = symbol#Stocke le symbole dans la variable
print("Clic sur :", symbol)#Affiche dans le terminal l'élément clické (symbol)
for i in range(len(elements)) :#Parcourt la liste d'éléments
    if symbol in elements[i] and len(symbol)==len(elements[i][1]) :#Cherche le symbole clické dans la liste
        name = elements[i][2][1]#Récupère le nom
        print(name)#Affiche le nom
    os.makedirs(f"{symbol}_box", exist_ok=True)#Créé un dossier dans le dossier actif en fonction du symbole

#Les lignes suivantes servent à copier un template .svg dans le dossier récemment créé. Il convient de modifier
les chemins comme nécessaire.

os.chdir(f"C:/Users/utilisateur/Desktop/SC_HM/prog/{symbol}_box")#Change le dossier actif à celui designé
(celui de l'élément)
os.chdir(f"D:/fbalba/prog/{symbol}_box")
#src=f"C:/Users/utilisateur/Desktop/SC_HM/prog/BasedBox15mm.svg"#Création de stings avec des
emplacements de fichiers
src=f"D:/fbalba/prog/BasedBox15mm.svg"
#dst=f"C:/Users/utilisateur/Desktop/SC_HM/prog/{symbol}_box/{symbol}_box.svg"
dst=f"D:/fbalba/prog/{symbol}_box/{symbol}_box.svg"

shutil.copy(src,dst)#Commande pour copier le fichier choisi dans le nouvel emplacement

for i in range(len(elements)):#Parcourt la liste des éléments
    if symbol in elements[i] and len(symbol)==len(elements[i][1]) :#Cherche le symbole clické dans la liste
        traceur1(elements[i][0],elements[i][2][1],elements[i][1])#Execute les fonctions traceur avec les bons arguments
        traceur2(elements[i][0],elements[i][2][1])

tree = ET.parse(f"{symbol}_box.svg")#Charge un fichier .svg et permet de travailler avec en python
root = tree.getroot()#Nécessaire pour modifier le fichier
ns = {"svg": "http://www.w3.org/2000/svg"}#ns=Namespace : Dictionnaire qui contient des spécifications
concernant les .svg
ET.register_namespace("", ns["svg"])#Permet au code d'utiliser le namespace

for i in range(len(elements)):#
    if symbol in elements[i] and len(symbol)==len(elements[i][1]) :
        if len(elements[i][1])==1 :#Change la taille de police selon la longueur du symbole
            symb = ET.Element("{http://www.w3.org/2000/svg}text", {#Mise en forme du texte voulu, ici le symbole
                "x": "25.1", # X position
                "y": "33", # Y position

```

```

    "font-size": "17",
    "font-family": "Elephant",
    "fill": "black",
    "text-anchor": "middle"
    })
    symb.text = f"{elements[i][1]}"
    root.append(symb)
else :
    symb = ET.Element("{http://www.w3.org/2000/svg}text", {
        "x": "25.1",      # X position
        "y": "33",      # Y position
        "font-size": "14",
        "font-family": "Elephant",
        "fill": "black",
        "text-anchor": "middle"
    })
    symb.text = f"{elements[i][1]}"
    root.append(symb)

for j in range(len(elements[i][2])): #Ajoute les noms dans les différentes langues
    ypos=54+4*j
    text = ET.Element("{http://www.w3.org/2000/svg}text", {
        "x": "56.74",    # X position
        "y": str(ypos),  # Y position
        "font-size": "4",
        "font-family": "Elephant",
        "font-weight": "bold",
        "fill": "black",
        "text-anchor": "middle"
    })
    text.text = f"{elements[i][2][j]}"
    root.append(text)

eng=ET.Element("{http://www.w3.org/2000/svg}text", { #Ajoute le numéro atomique
    "x": "25.1",
    "y": "38",
    "font-size": "4.5",
    "font-family": "Elephant",
    "fill": "black",
    "text-anchor": "middle"

```

```

    })
    eng.text = f"{elements[i][2][0]}"
    root.append(eng)
    num=ET.Element("{http://www.w3.org/2000/svg}text", {#Ajoute le numéro atomique
        "x": "25.1",
        "y": "103",
        "font-size": "17",
        "font-family": "Elephant",
        "fill": "black",
        "text-anchor": "middle"
    })
    num.text = f"{elements[i][0]}"
    root.append(num)
    number=ET.Element("{http://www.w3.org/2000/svg}text", {#Ajoute le mot "Number"
        "x": "25.1",
        "y": "87",
        "font-size": "6",
        "font-family": "Elephant",
        "fill": "black",
        "text-anchor": "middle"
    })
    number.text = "Number"
    root.append(number)

with open(f"orb_{name}.png", "rb") as f:#Ajoute le modèle de Bohr enregistré dans le dossier actif
orb_data = f.read()
encoded_orb = base64.b64encode(orb_data).decode("utf-8")
orb = ET.Element("{http://www.w3.org/2000/svg}image", {
    "x": "80",
    "y": "80",
    "width": "200",
    "height": "200",
    "href": f"data:image/png;base64,{encoded_orb}"
})
root.append(orb)

watermark=ET.Element("{http://www.w3.org/2000/svg}text", {#Ajoute le texte "Fait avec Python"
    "x": "130",
    "y": "125",
    "font-size": "10",

```

```

    "font-family": "Elephant",
    "fill": "black",
    "text-anchor": "middle"
})
watermark.text = "Fait avec Python"
root.append(watermark)

tree.write(f"{symbol}_box.svg", encoding="utf-8", xml_declaration=True)

#os.chdir(f"C:/Users/utilisateur/Desktop/SC_HM/prog")#Remet le dossier parent comme actif
os.chdir(f"D:/fbalba/prog")
#os.startfile(f"C:/Users/utilisateur/Desktop/SC_HM/prog/{symbol}_box/{symbol}_box.svg")#Ouvre le fichier .sv
qui vient d'être modifié
os.startfile(f"D:/fbalba/prog/{symbol}_box/{symbol}_box.svg")

JSON_PATH = None
while JSON_PATH is None:
    JSON_PATH = get_last_json()
    time.sleep(0.1)

with open(JSON_PATH, "r", encoding="utf-8") as f:
    data = json.load(f) #lit le fichier .json

root = tk.Tk()#Permet de créer la fenêtre
root.configure(bg="white")
root.title("Tableau Périodique")#Nom de la fenêtre
app = PeriodicTableApp(root)#Permet à l'objet PeriodicTableApp de s'afficher dans la fenêtre

sym_clicked = data.get("symbol", None).strip()
print(sym_clicked)
#print(buttons)

```

if sym_clicked in buttons:

```
root.after(1000, lambda: buttons[sym_clicked].invoke()) #simule un clic sur le bouton correspondant
```

```
root.mainloop()#Fait fonctionner la fenêtre jusqu'à sa fermeture
```

```
print("Valeur finale :", app.last_clicked) #Affiche la dernière case cliquée lors de la fermeture
```

Revision #9

Created 19 January 2026 15:55:20 by Hugo Maman

Updated 30 January 2026 17:27:11 by Hugo Maman