```python
#!/usr/bin/python3

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf

# Dans cet exercice nous allons générer des échantillons artificiels en utilisant des
# générateurs de nombre aléatoire (dans une loi uniforme ou normale). Au sens strict,
# un ordinateur est incapable de générer des nombres au hasard : il est déterministe
# par construction. Tout ce que nous pouvons faire c'est générer des suites de nombres
# qui ressemblent à des nombres au hasard. C'est pourquoi on parle de générateurs de
# nombres pseudo aléatoires. Ces algorithmes génèrent des suites (déterminsites) de
# nombres qui ont les caractéristiques (il faudrait définir lesquels) de suites de
# nombres aléatoires. Ces algorithmes sont récursifs, dans le sens où pour une
# condition initiale donnée, ils génèreront toujours la même séquence de nombre. Cette
# propriété est intéressante, car quand nous changeons notre code, si nous observons
# un changement dans les résultats alors nous pouvons être certain que c'est la
# conséquence du changement sur le code (car les « aléas » sont les mêmes). La condition
# initiale des générateurs de nombres au hasard s'appelle le « seed », nous le fixons
# au début du programe a une valeur entière arbitraire.

np.random.seed(1938)

#
# Question 1.
# rand pour une loi uniforme entre 0 et 1

m = 500                          # Le nombre d'observations.
x = np.random.rand(m)*10         # La variable explicative.


#
# Question 2.
# randn pour une loi normale de moyenne nulle et d'écart-type=1

u = np.random.randn(m)*6         # Le vecteur des erreurs d'écart-type 6 (variance 36).

print('La moyenne des erreurs est :   '+f'{u.mean():.2f}')
print("L'écart-type des erreurs est : "+f'{u.std():.2f}')

#autre syntaxe pour imprimer
print(f"la moyenne des erreurs est ici de {u.mean():.2f}")


#
# Question 3.
#

y = 1 + 2*x + u                  # La variable expliquée.

plt.plot(x, y, '.b', markersize=.5)
plt.xlabel('x')
plt.ylabel('y')
#plt.show()
# import tikzplotlib
# tikzplotlib.save("exercice-2.4.tex")


#
```

```
# Question 4.
#

# On commence par créer une base de données contenant y et x.
dataset = pd.DataFrame(np.array([y,x]).transpose(),columns=['y','x'])

# Puis on lance l'estimation par les MCO
results = smf.ols('y ~ x', data=dataset).fit()

print(results.summary())

#
# Question 5.
#

# Calculons de ∑ ê[i] et ∑ x[i]ê[i]
s_e = .0
s_xe = .0
for i in range(m):
    s_e += results.resid[i]
    s_xe += x[i]*results.resid[i]

print('La somme des résidus est : '+str(s_e))
print('La somme du produit des résidus et de la variable explicative est : '+str(s_xe))

#autre syntaxe pour imprimer
print(f"la somme des résidus est ici de {str(s_e)}")
#
# Question 6.
#

# Calculons de ∑ u[i] et ∑ x[i]u[i]. Cette fois-ci, plutôt que d'utiliser une boucle,
# nous utilisons des opérations sur les vecteurs. Python n'est pas optimisé pour les
# boucles, et opérer directement sur des boucles est généralement plus efficace (rapide).

s_u = u.sum()
s_xu = (x*u).sum()

print('La somme des « vraies » erreurs est : '+str(s_u))
print('La somme du produit des erreurs et de la variable explicative est : '+str(s_xu))


#
# Question 7.
#

# Pour répondre à la question 7, sachant que la question suivante nous demandera de faire
# la même chose, nous allons définir une fonction. Cette fonction simule des échantillons
# et pour chaque  échantillon calcule l'estimateur des MCO. La fonction retourne un vecteur
# contenant toutes les valeurs obtenues de beta1, il s'agit de la distribution
# empirique de l'estimateur des MCO du paramètre de pente du modèle linéaire.

def mc(n, x):

 #  On définit une fonction qui simule n échantillons conditionellement
 #  à des réalisations de la variable explicative x.

    X = sm.add_constant(x)  # La matrice des régresseurs (on rajoute une colone de 1 pour
                            # la constante).
    b = np.array([1,2])     # Vecteur des « vraies » valeurs des paramètres b0 et b1.
    Z = np.dot(X, b)        # Évaluation de b0 + b1 x
```

```
    beta1 = np.empty([n,1]) # Initialisation du vecteur retourné (pour les estimateurs de b1).
    for i in range(n):
        y = Z + np.random.randn(m)*6   # Simulation de la variable expliquée en rajoutant
                                       # les erreurs toujours tirées dans une loi normale
                                       # de moyenne nulle et d'écart-type 6.
        model = sm.OLS(y, X)           # Déclaration du modèle estimé.
        results = model.fit()          # Estimation
        beta1[i] = results.params[1]   # On enregistre l'estimateur de b1.
#       print (beta1[i])
    return beta1

# Attenion : nous avons m observations par échantillon et n échantillons

b10 = mc(10, x)

print("Pour n = 10, la moyenne de l'estimateur est ", str(b10.mean()))

#
# Question 8.
#

b100 = mc(100, x)
b1000 = mc(1000, x)

print("Pour n = 100, la moyenne de l'estimateur est ", str(b100.mean()))
print("Pour n = 1000, la moyenne de l'estimateur est ", str(b1000.mean()))

print("Écart à la vraie valeur (n=10) ", str(abs(b10.mean()-2)))
print("Écart à la vraie valeur (n=100) ", str(abs(b100.mean()-2)))
print("Écart à la vraie valeur (n=1000) ", str(abs(b1000.mean()-2)))
```