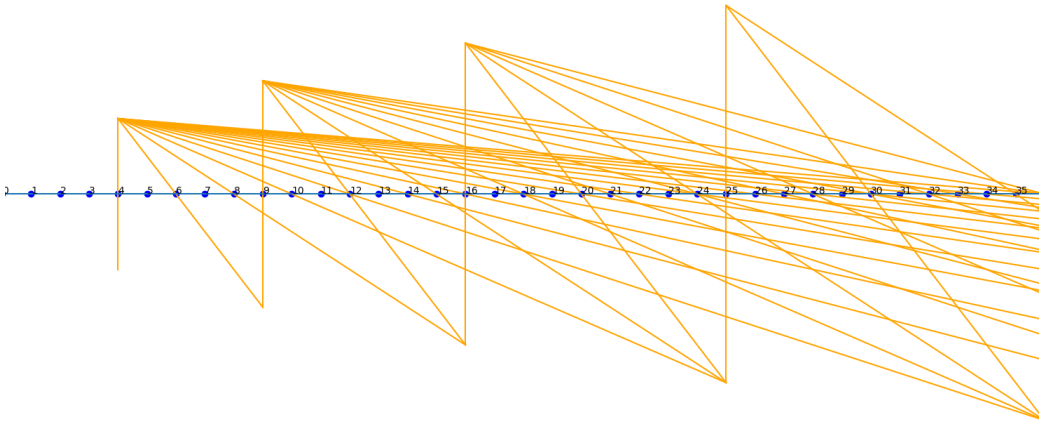
Travaux Pratiques n°13
Révisions

▷ Exercice 1 : Recherche de nombres premiers.

1. Écrire une fonction `estpremier(n: int) -> bool` qui renvoie `True` si n est premier, `False` sinon.
2. L'algorithme du *crible d'Ératosthène* permet, étant donné un entier N, de déterminer les entiers premiers de $[\![0,N]\!]$ en obtenant une liste `L` de $N+1$ booléens telle que `L[i]` vaille `True` si i est premier, `False` sinon.
   Pour cela, la liste `L` ne contenant initialement que des `True`, on assigne `False` à `L[0]` et `L[1]` ; puis on parcourt L de $i=2$ à $i=\left\lfloor\sqrt{N}\right\rfloor$; si `L[i]` vaut `True`, on assigne `False` à `L[k]` pour tout k multiple strict de i.
   Implémenter cet algorithme.
3. Le *crible de Matiyasevich-Stechkin* est une représentation graphique du crible d'Ératosthène : pour tout $m \in \left[\!\left[2,\left\lfloor\sqrt{N}\right\rfloor\right]\!\right]$ et tout $n \in \left[\!\left[m,\left\lfloor\frac{N}{m}\right\rfloor\right]\!\right]$, on relie les points $M(m^2,m)$ et $N(n^2,-n)$. Le segment $[MN]$ coupe alors l'axe horizontal (Ox) au point d'abscisse $m\times n$; ainsi, les abscisses entières de l'axe (Ox) en lesquelles aucun segment ne passe sont les nombres premiers.
   Implémenter ce crible afin d'obtenir une figure semblable à la figure ci-dessous. On utilisera les commandes suivantes pour représenter l'axe (Ox) :

```
import matplotlib.pyplot as plt
for k in range(M):
    plt.scatter(k,0,color="blue")
    plt.annotate(str(k),(k,0))
plt.axhline(0)      # affiche l'axe des abscisses
# plt.axvline(0)    # affiche l'axe des ordonnées
```



▷ **Exercice 2 : Théorème des nombres premiers.**

1. Pour tout réel x, on note $\pi(x)$ le nombre de nombres premiers inférieurs ou égaux à x. À l'aide des données de l'exercice précédent, représenter le graphe de la fonction π.
2. Représenter, sur le même dessin, le graphe de la fonction $x \mapsto \frac{x}{\ln(x)}$, à l'aide de la fonction `log` du module `numpy`. Que constate-t-on ?
3. Toujours sur le même dessin, représenter le graphe de la fonction $x \mapsto \int_2^x \frac{dt}{\ln(t)}$. Pour cela, on utilisera le module `sympy` :

```
import sympy as sp
t = sp.Symbol('t')
Li = lambda x:sp.integrate(1/sp.log(t),(t,2,x))
```

Commenter.

▷ **Exercice 3 : Spirale d'Ulam.** On souhaite représenter les nombres premiers en *spirale*. Pour cela, on associe à chaque entier naturel n un point à coordonnées entières (x,y) du plan. L'entier 1 est associé à l'origine (0,0), puis on suit le parcours suivant :

```
37—36—35—34—33—32—31
|                   |
38  17—16—15—14—13  30
|   |           |   |
39  18   5—4—3   12  29
|   |   |   |   |   |
40  19   6   1—2   11  28
|   |   |           |   |
41  20   7—8—9—10  27
|   |                   |
42  21—22—23—24—25—26
|
43—44—45—46—47—48—49...
```

On pourra remarquer qu'en partant de 1, on se déplace 1 fois vers la droite puis 1 fois vers le haut ; puis 2 fois vers la gauche puis 2 fois vers le bas ; puis 3 fois vers la droite puis 3 fois vers le haut, etc.

1. Lors de ce parcours, stocker les coordonnées des nombres premiers, puis les représenter à l'aide de la fonction `scatter(X,Y)` du module `matplotlib.pyplot`, où `X` est la liste des abscisses et `Y` celle des ordonnées. Que constate-t-on ?
2. Écrire une fonction `nbdiviseurs(n: int) -> int` qui renvoie le nombre des diviseurs de n.
3. Reprendre le parcours précédent en représentant chaque entier par un point de taille proportionnelle au nombre de ses diviseurs, à l'aide de `scatter(X,Y,S)` où `S` est la liste des tailles des points. Commenter.

▷ **Exercice 4.**

Le chiffrement RSA est un procédé de chiffrement de données couramment utilisé pour les transmissions par internet. Son nom est composé des initiales des trois informaticiens qui l'ont défini en 1977 : Ronald Rivest, Adi Shamir et Leonard Adleman.

Le principe est le suivant :

- Les données sont numérisées, c'est-à-dire converties en entiers naturels. Tous ces entiers sont supposés inférieurs à un entier n fixé, appelé *module de chiffrement* ou *nombre RSA*.
- Un entier est chiffré en appliquant la fonction $F_{n,c}$ qui à un entier a associe le reste de la division euclidienne de a^c par n, où c est un entier naturel, appelé *clef publique*.
- On convertit ensuite les entiers au format de départ, par exemple du texte.
- Pour déchiffrer, il suffit d'appliquer la fonction $F_{n,d}$ c'est-à-dire la même fonction mais en remplaçant c par d, entier naturel appelé *clef privée*.
  La clef privée d est l'*inverse modulaire* de c modulo $\varphi(n)$, c'est-à-dire que $cd \equiv 1[\varphi(n)]$, où φ est la fonction indicatrice d'Euler. On a alors $(a^c)^d \equiv a[n]$ (c'est le *théorème d'Euler*).
  En pratique, on prend pour n un produit de deux grands nombres premiers p et q, auquel cas $\varphi(n) = (p-1)(q-1)$.

L'utilisateur qui souhaite recevoir des données chiffrées communique le couple (n,c) aux autres utilisateurs. Il ne communique pas la clef privée d.

La sécurité du chiffrement RSA vient du fait qu'il est très difficile de déterminer d à partir de n et c sans connaître p et q. Un prix de 200 000 dollars a ainsi été offert pour la factorisation de RSA2048, un nombre de 2048 bits (soit 617 chiffres décimaux).

1. Chaque caractère est codé par un entier, c'est le codage ASCII étendu. Par exemple les lettres minuscules `a` à `z` sont codées de 97 à 122.

   La fonction `ord` donne le code ASCII d'un caractère, tandis que la fonction `chr` donne le caractère codé par l'argument.

   Écrire une fonction `Numerisation(S: str) -> [int]` qui renvoie la liste des codes ASCII des lettres de S.
2. Écrire une fonction `ConversionStr(L: [int]) -> S` qui renvoie la chaîne de caractères que code la liste d'entiers L.
3. Écrire une fonction `F(a: int,n: int,c: int) -> int` qui renvoie le reste de la division euclidienne de a^c par n.
4. Écrire une fonction `RSA(L: [int],n: int,c: int)` qui renvoie la liste des éléments $F(a,n,c)$ où a parcourt la liste d'entiers L.
5. Écrire une fonction `Chiffre(S: str,n: int,c: int)` qui renvoie le chiffrement RSA de `S` avec n pour module de chiffrement et c pour clef.
6. On prend $n = 2021 = 43 \times 47$. Trouver une clef publique et une clef privée convenables, et tester la fonction `Chiffre`!

▷ **Exercice 5.** Un joueur joue à pile ou face : s'il gagne, il emporte g euros, et s'il perd, il doit donner p euros. Il joue n parties. Représenter sur un graphe l'évolution de ses gains au cours du temps pour différentes valeurs de g, p et n.
On utilisera la fonction `randint` du module `random`.

▷ **Exercice 6.**

1. Écrire une fonction `bernoulli(p)` qui simule une variable aléatoire de Bernoulli de paramètre $p \in [0,1[$: elle renvoie 1 avec une probabilité p, et 0 avec une probabilité $1-p$. On utilisera la fonction `random` du module `random`.
2. Écrire un programme qui choisit un nombre $p \in [0,1[$, puis qui effectue N tirages de Bernoulli de paramètre p. À chaque étape, on calcule la moyenne des tirages obtenus ; puis on représente sur un graphique l'évolution de cette moyenne. Qu'observe-t-on ?

▷ **Exercice 7.**

On souhaite représenter le *diagramme de factorisation* d'un entier n.

1. Représenter n points en cercle pour différentes valeurs de n. Pour représenter des points, on utilisera la fonction `scatter`. L'option `s=100` permet de contrôler la taille des points.
2. Écrire une fonction `pgdp(n)` renvoyant le plus grand diviseur premier d de n.
3. Écrire une fonction `diagramme(n)` renvoyant les listes X_n et Y_n, respectivement, des abscisses et des ordonnées des points du diagramme de factorisation de n :
   - si n est premier, les points de ce diagramme sont en cercle,
   - sinon, en notant d le plus grand diviseur premier de n : X_n est la liste des $x_d + \frac{x}{d}$ où x_d parcourt X_d et x parcourt $X_{\frac{n}{d}}$; et de même pour Y_n.
4. Ajouter comme titre l'écriture de la décomposition de n, sous la forme `15 = 3*5`.
5. Afficher le diagramme de 2024.