

## Nombre premier : Conjecture de Goldbach

### Compétences visées

- **chercher**, expérimenter – en particulier à l'aide d'outils logiciels ;
- **modéliser**, faire une simulation, valider ou invalider un modèle ;
- **représenter**, choisir un cadre (numérique, algébrique, géométrique...), changer de registre ;
- **calculer**, appliquer des techniques et mettre en œuvre des algorithmes.

Ces compétences sont mises en œuvre dans le cadre de l'extrait du programme de 2<sup>nde</sup> GT ci-dessous :

« Déterminer si un entier naturel est premier. » ou dans le cadre de l'extrait du programme de spécialité mathématiques en terminale S : « questionnement sur les nombres premiers »

### Situation déclenchante

La conjecture de Goldbach affirme que tout nombre pair supérieur ou égal à 4 est la somme de 2 nombres premiers. Peut-on la tester sur les premiers entiers naturels ?

### Problématique

Ecrire un script qui permet de vérifier cette conjecture pour un entier naturel  $n$  donné. Le script devra fournir tous les couples d'entiers premiers dont la somme fait  $n$ .

## Fiche méthode

### Proposition de résolution

A partir d'une liste de nombres premiers, on teste toutes les sommes possibles de manière à obtenir le nombre souhaité.

Ainsi, on crée trois fonctions dans ce script :

- Une fonction **premier** qui prend comme paramètre un entier naturel  $n$  et qui renvoie vrai si  $n$  est premier et faux sinon.
- Une fonction **listepremiers** qui prend comme paramètre un entier naturel  $n$  et qui renvoie la liste des nombres premiers inférieurs ou égaux à  $n$ .
- Une fonction **goldbach** qui prend comme argument un entier naturel  $n$  et qui renvoie une liste de couples d'entiers premiers dont la somme fait  $n$ .

```
PYTHON SHELL
>>> from GOLDBACH import *
>>> premier(15)
False
>>> premier(17)
True
>>> listepremiers(21)
[2, 3, 5, 7, 11, 13, 17, 19]
>>> listepremiers(31)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
>>> |
Fns... | a A # | Outils | Éditer | Script
```

```
PYTHON SHELL
Configuration terminée.

>>> # L'exécution de GOLDBACH
>>> from GOLDBACH import *
>>> goldbach(24)
[[5, 19], [7, 17], [11, 13]]
>>> goldbach(34)
[[3, 31], [5, 29], [11, 23], [17, 17]]
>>> |
Fns... | a A # | Outils | Éditer | Script
```

### Etapes de résolution

#### Fonction **premier** :

L'instruction `for i in range(2,n)` permet de créer une boucle avec  $i$  variant de 2 à  $n-1$ .  
Pour avoir un script plus rapide on aurait pu s'arrêter à  $\text{floor}(\sqrt{n}) + 1$  dans la boucle.  
L'instruction `n%i` permet de renvoyer le reste dans la division euclidienne de  $n$  par  $i$ .

```
EDITEUR : GOLDBACH
LIGNE DU SCRIPT 0001
def premier(n):_
  if n==1:
  return False
  for i in range(2,n):
  if n % i == 0:
  return False
  return True
Fns... | a A # | Outils | Exéc | Script
```

**C'est un principe à retenir :** On peut appeler une fonction (ici : la fonction **premier**) à l'intérieur d'une autre fonction (ici : **listepremier**). L'utilisation successive de fonctions en python rend le script dans son ensemble plus lisible.

#### La fonction **listepremiers** :

L'instruction `listepremier.append(i)` permet de rajouter le nombre  $i$  à la liste `listepremier`.

```
EDITEUR : GOLDBACH
LIGNE DU SCRIPT 0022
def listepremiers(n):
  listepremier = []
  for i in range(2,n+1):
  if premier(i) == True:
  listepremier.append(i)
  return listepremier
Fns... | a A # | Outils | Exéc | Script
```

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus



## Fiche méthode

### Etapas de résolution

La fonction **goldbach** :

On fait appel à la fonction **listepremiers** pour obtenir une liste de nombres premiers.

On utilise une double boucle pour générer toutes les sommes possibles de nombres premiers à partir de la liste obtenue grâce à la fonction **listepremiers**.

La structure conditionnelle permet de tester si la somme est bien égale au nombre recherché et si la paire n'a pas déjà été sélectionnée.

```
ÉDITEUR : GOLDBACH
LIGNE DU SCRIPT 0029
def goldbach(n):
    **paires = []
    **liste = listepremiers(n)
    **for premier1 in liste:
    **for premier2 in liste:
    **if premier1 + premier2 ==
    n and [premier1,premier2] n
    ot in paires and [premier2,
    premier1] not in paires:
    **paires.append([premier1,
    premier2])_
    **return (paires)
```

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

