$\mathcal{M}$ $\mathcal{P}$ $\mathcal{S}$ $\mathcal{I}$ 2

LYCEE CHARLEMAGNE
2020/2021

VENDREDI 13 NOVEMBRE

I.P.T.

$\mathcal{M}$ $\mathcal{P}$ $\mathcal{S}$ $\mathcal{I}$ 2 TEX I.P.T.

Pour discuter avec les colleurs ou même entre vous sous Discord, vous pouvez utiliser le compilateur TEX pour écrire des formules mathématiques.

Délimiteurs : toute formule mathématique est encadrée par deux délimiteurs $. C'est ainsi que même les chiffres seront différents à l'affichage entre 54 et 54 (subtil à voir quand même).

54, bonjour	54, bonjour
54, $bonjour$, $comment allez vous ?$	54, $bonjour$, $commentallezvous$
$bonjour\ comment\ allez\ vous$	$bonjour comment allez vous$

En mode mathématique, il n'y a pas d'espace entre les symboles. Ou alors il faut les forcer.

Indices et exposants : l'underscore pour les indices et le circonflexe pour les exposants.

x^n, u_k,$\sum_0 ^na_k$	x^n, u_k, $\sum_0 ^na_k$
$\sum_k=0^na^k.b^n-k$ $\sum_{k=0}^n a^k .b^{n-k}$	$\sum_k = 0^n a^k.b^n - k$ $\sum_{k=0}^n a^k.b^{n-k}$
e^x^n, e^{x^n}, $a_k.x^{\alpha_k}$	e^{xn}, $e^{(x^n)}$ $a_k.x^{\alpha_k}$

Quand il faut mettre plusieurs termes dans l'indice ou l'exposant, on les encadre avec des accolades pour en faire un bloc.
On peut aussi créer des indices d'indices et des exposants d'exposants. La fonte sera de plus en plus petite.

Mots clefs : commencent avec un \, et après il y en a quelques uns à connaître.

$\forall$, $\exists$,, $\in$, $\ldots$	$\forall$, $\exists$, $\in$, $\ldots$
$\sum$, $\prod_{0\leqslant k<n}$, $\int_a^b$	$\sum$, $\prod_{0\leqslant k<n}$, $\int_a^b$
$\Rightarrow$, $\Leftrightarrow$ $\rightarrow$, $\mapsto$, $\cap$	$\Rightarrow$, $\Leftrightarrow$ $\rightarrow$, $\mapsto$, $\cap$
$\exp$, $\sin$, $\cos$, $\tan$ $\log$, $\Re$, $\Im$, π	$\exp$, $\sin$, $\cos$ $\tan$, $\log$, $\Re$, $\Im$, π
$\leqslant$, $\geqslant$, $\neq$, $\notin$, $\sim$, $\subset$	$\leqslant$, $\geqslant$, $\neq$, $\notin$, $\sim$, $\subset$

Le \not vient surcharger certains symboles, avec plus ou moins de bonheur.$\not\Rightarrow$, $\not\exists$

Dispositions particulières : pour les fractions, deux syntaxes possibles avec \frac et \over.

${a\over b}$	$\frac{a}{b}$
$\frac{a}{b}$, $1+\frac{1}{\frac{2^3}{41}}$	$\frac{a}{b}$, $1+\frac{1}{\frac{2^3}{41}}$
$\binom{12}{3}=\frac{12\times 11\times 10}{1\times 2\times 3}$	$\binom{12}{3}=\frac{12\times 11\times 10}{1\times 2\times 3}$

On retrouve cette syntaxe avec des fonctions à deux paramètres comme \binom.

Polices particulières : mettez un bloc entre accolades, et imposez une taille plus grande, plus petite, ou une fonte particulière

$a\in\mathbb R$, $x+i.y\in\mathbb C$	$a\in\mathbb R$, $x+i.y\in\mathbb C$
$bonjour\ \mathbf{les\ petits}\ de\ MPSI2$	$bonjour\ \mathbf{les\ petits}\ de\ MPSI2$
$(a<b)\mathbf{\ et\ c>0}\Rightarrow (a.c<b.c)$	$(a<b)\mathbf{\ et\ c>0}\Rightarrow (a.c<b.c)$

Matrices :

Une prochaine fois.

Travaux pratiques : Écrivez la définition de « $\prec$ est une relation d'ordre sur l'ensemble E ».
Ecrivez et complétez la formule $\int_0^{\pi} \sin^n(t).dt = \frac{\pi...}{2^{2.n+1}}$

Vous savez utiliser le module `pyplot` pour tracer des graphes de fonctions.

Rappel : vous devez
- importer le module
- créer une liste d'abscisses (et encore...)
- une liste d'ordonnées
- créer le graphe
- forcer `Python` à l'afficher

```
import matplotlib.pyplot
import matplotlib.pyplot as pl
from matplotlib.pyplot import *
```

Pour l'importation du module, vous avez le choix entre
La première solution est lourde, elle vous forcera à solliciter des `matplotlib.pyplot.plot()` et `matplotlib.pyplot.show()` assez lourds à écrire.

La dernière ne vous permettra pas de vous souvenir en écrivant `plot(Abscisses,Ordonnees)` et `show()` si les procédures sollicitées viennent du module plot ou d'un autre module (avec le risque aussi si par exemple vous avez importe le module `Turtle` dans le même temps qu'il y ait deux procédures s'appelant `show()`).
On optera donc pour la seconde qui importe le module mais lui donne un petit nom (un alias) raccourci. [1].

Pour créer une liste d'abscisses, on est matheux, on sait que quand on découpe $[a,\ b]$ en n segments, les extrémités sont les $a + k.\frac{b-a}{n}$ (s'écrivant aussi comme des moyennes $\frac{(n-k).a+k.b}{n}$).Je ne vois pas l'intérêt de solliciter une fonction préparée par des individus compatissants qui vous prennent pour une brèle en maths.

```
Abscisses=[2*k*pi/n for k in range(n)]
```

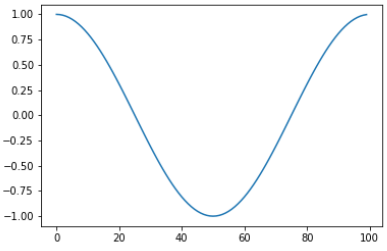
me semble facile à écrire soi même, et même plus facile que de retenir un nom de procédure. [2]

Pour les ordonnées vous créez une nouvelle liste en prenant les abscisses une à une (for x in Abscisses) et en calculant pour chacune l'ordonnée qui vous intéresse.

```
Ordonnees = [sin(x) for x in Abscisses]
```

On a alors deux listes de même longueur, on peut les représenter, puis demander à Python de montrer ce qu'il a fait.

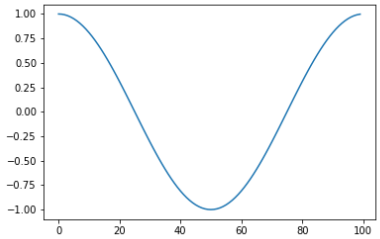
```
import mathplotlib.pyplot as pl
abscisses = [2*k*pi/100 for k in range(100)]
ycos=[cos(x) for x in abscisses]
ysin=[sin(x) for x in abscisses]
pl.plot(abscisses, ycos)
pl.show()
```



Si on ne donne que la liste des ordonnées (de longueur n), Python complète par une liste d'abscisses faite de n réels régulièrement espacés.

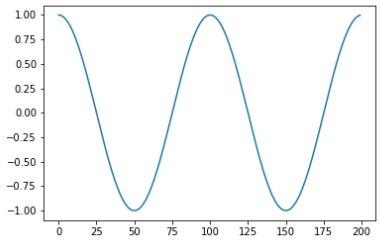
1. les plus idiots taperont `import matplotlib.pyplot as PourTracerDesGraphes` et devront chaque fois taper des phrases bien longues
2. au fait, avec vous importé `math` pour que `pi` soit un nom connu de `Python` ?

```
import mathplotlib.pyplot as pl
abscisses = [2*k*pi/100 for k in range(100)]
ycos=[cos(x) for x in abscisses]
ysin=[sin(x) for x in abscisses]
pl.plot(ycos)
pl.show()
```



Attention, que signifie *2 sur une liste ?

```
import mathplotlib.pyplot as pl
abscisses = [2*k*pi/100 for k in range(100)]
ycos=[cos(x) for x in abscisses]
ysin=[sin(x) for x in abscisses]
pl.plot(ycos*2)
pl.show()
```



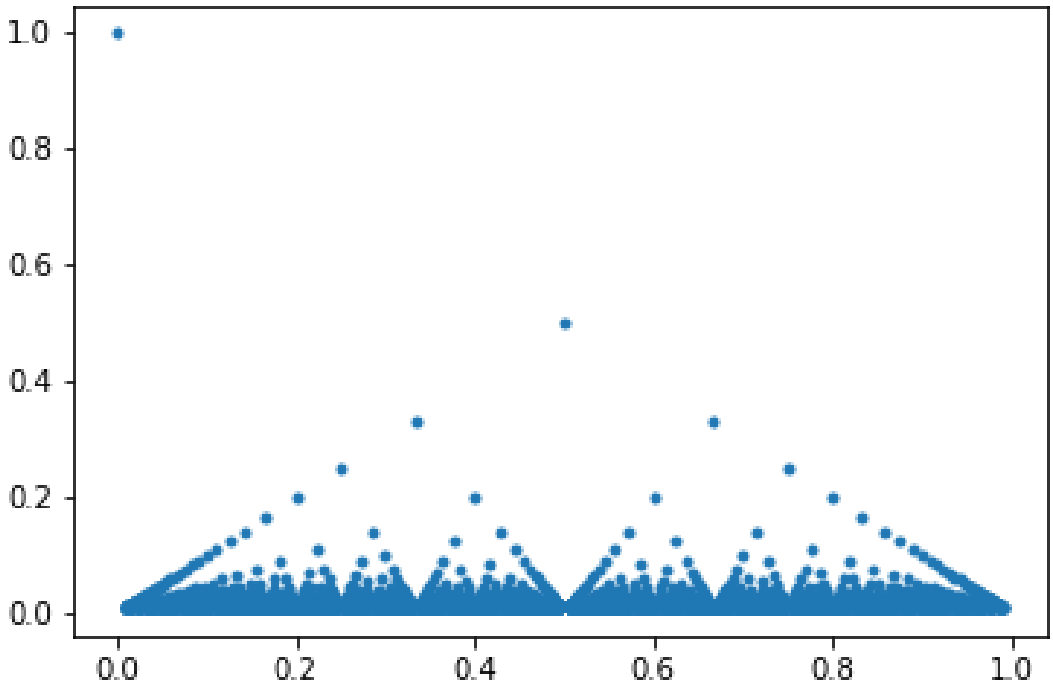
Si vous donnez trois listes, la première sera celle des abscisses, les deux suivantes les ordonnées.
Mais si vous en oubliez une ?

```
import mathplotlib.pyplot as pl
abscisses = [2*k*pi/100 for k in range(100)]
ycos=[cos(x) for x in abscisses]
ysin=[sin(x) for x in abscisses]
pl.plot(abscisses,ycos,ysin)
pl.show()
```

```
import mathplotlib.pyplot as pl
abscisses = [2*k*pi/100 for k in range(100)]
ycos=[cos(x) for x in abscisses]
ysin=[sin(x) for x in abscisses]
pl.plot(ysin, ycos)
pl.show()
```

Expliquez le troisième graphe obtenu (celui sans « abscisses).

Notre objectif est non pas de tracer des fonctions gentilles, comme des polynômes, lignes trigonométriques, mais des fonctions discontinues,c comme celle appelée dégoulinante de Baire dans notre cours (ou pop-corn de Thomae dans les autres cours) : $\left| x \longmapsto \begin{cases} \frac{1}{q} & si \quad x = \frac{p}{q} \quad irreductible \\ 0 & si \quad x \quad irrationnel \end{cases} \right|$



On ne peut évidemment pas définir une fonction qui fait ça
```
def f(x) :
....if ....
........return(....)
```
.
Pour Python, tout nombre est décimal, et jamais irrationnel.

On va en fait tracer des points. Comment `plot()` fait il ?[3]
Si on lui donne deux listes de nombres de même longueur [a, b, c] et [A, B, C], il trace les points de coordonnées (a, A), (b, B) et (c, C) et les relie.

```
import mathplotlib.pyplot as pl
abscisses = [1, 1, 2, 3, 3, 1, 3, 1, 3]
ordonnees = [0, 2, 3, 2, 0, 2, 2, 0, 0]
pl.plot(abscisses, ordonnees)
pl.show()
```

tracez le

Pour voir si vous avez compris : tracez vos initiales.

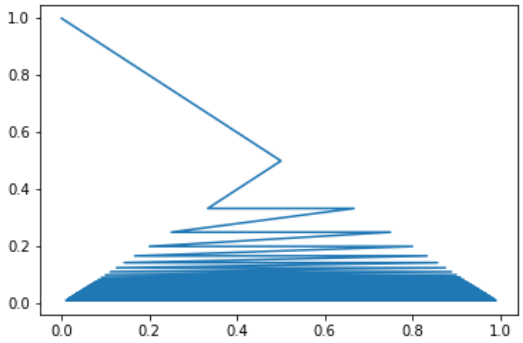
Testez ensuite les options `linesytle` et `marker`.

```
pl.plot(abscisses, ordonnees, marker='O')
pl.plot(abscisses, ordonnees, linestyle='none',
marker='*')
```

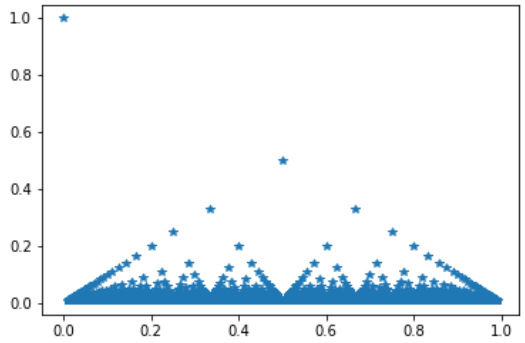
Pour la fonction de Baire, que devrez vous alors faire ?
Vous vous fixez un entier N qui fixera la précision de votre graphe (entre 0 et 1 par périodicité).

3. et en fait, tout graphe est une suite de points qu'il relie, pour lesquels en fait les abscisses sont en progression arithmétique gentille

Créez alors une liste d'abscisses (rationnelles) : les p/q avec $0 \leqslant p \leqslant q \leqslant N$.
une liste d'ordonnées (rationnelles) : les images 1/q avec $0 \leqslant p \leqslant q \leqslant N$.
Par exemple $\begin{matrix} 0 & 1 & 0.5 & 0.333 & 0.666 & 0.25 & 0.75 & 0.2 & 0.4 & 0.6 & 0.8 \\ 1 & 1 & 0.5 & 0.333 & 0.666 & 0.25 & 0.25 & 0.2 & 0.2 & 0.2 & 0.2 \end{matrix}$.[4]
Tracez.

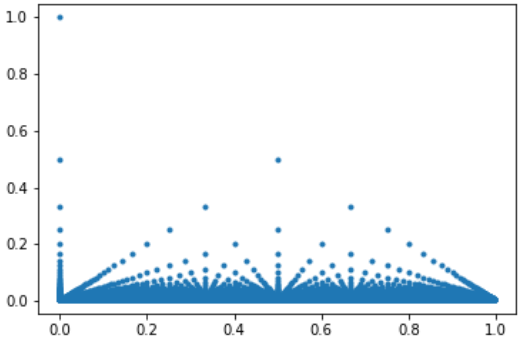


A

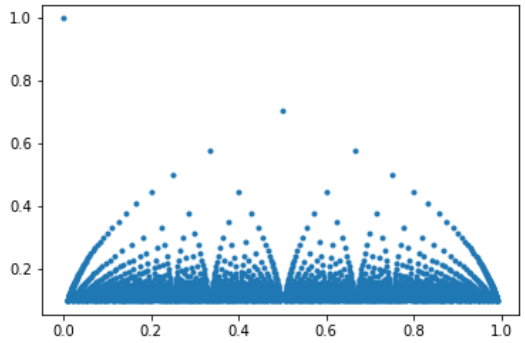


B

Quelle est l'« erreur » du premier graphe A ? Que pensez vous du second B ?



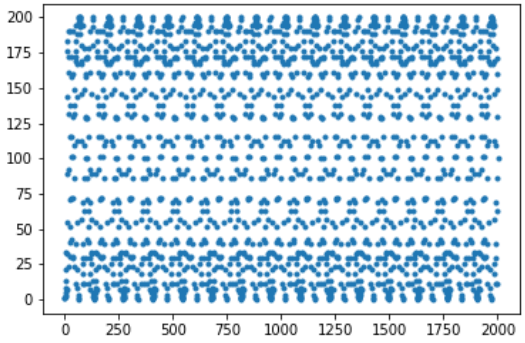
C



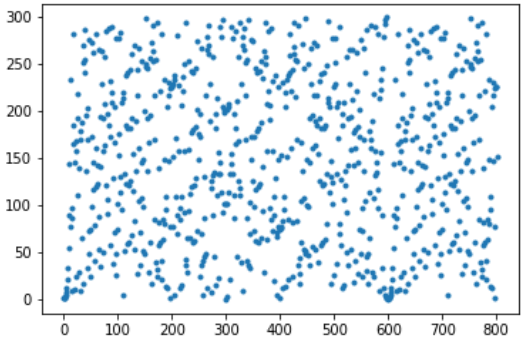
D

L'erreur du graphe C est qu'il y a trop de points. Avez vous commis la même erreur ? Comment pouvez vous l'éviter.
Le graphe D représente une variante qui dégouline moins vite. Saurez vous l'identifier ?

Question pour les experts :
avec un graphe point par points : représentez la suite de Fiboacci.
C'est quoi ? C'est comme la suite de Fibonacci, mais modulo n (n est un entier fixé à l'avance).
Les termes sont définis par $F_0 = F_1 = 1$ et $F_{p+2} = P_{p+1} + F_p \ mod \ n$.



E



F

4. évidemment, ne tapez pas tout à la main, faites des boucles imbriquées `for q in range(...) : for p in range(...)`

Indiquez moi modulo combien pour E et pour F.

$\mathcal{M}$	$\mathcal{P}$	$\mathcal{S}$	$\mathcal{I}$	2	Défi arithmétique.	I.P.T.

Qu'on de particulier les entiers comme 3114, 173, 15, 1000 ?
Regardez leurs carrés : 9696996, 29929, 225 et 1000000.
L'écriture de leurs carrés ne fait intervenir que deux chiffres.
Contrairement à 3115 par exemple dont le carré 9703225 consomme six chiffres différents.
On qualifiera ces entiers de PetitCarre (il n'y a pas encore de nom pour eux).

Votre mission : trouver des entiers ayant cette propriété (même sans ordinateur, on sait en trouver des aussi grands qu'on veut).
Écrire une procédure qui teste si un nombre n est PetitCarre.

Pour voir si vous avez bien programmé, envoyez moi le nombre de PetitCarre plus petits que 10^{10} et le plus grand que vous ayez trouvé qui ne soit pas de la forme $k.10^p$ avec k dans $\{1,2,3\}$.

$\mathcal{M}$	$\mathcal{P}$	$\mathcal{S}$	$\mathcal{I}$	2	0 points	I.P.T.

Solution pour le défi arithmétique.

```
def NbChiffres(N) :
....#prend un entier N et donne le nombre de chiffres distincts qui le composent
....L = [] #la liste des chiffres
....NS = N #copie pour ne pas abimer
....while NS > 0 : #on va le faire fondre
........unite = NS%10 #on lit son chiffre des unités
........if unité not in L : #si c'est un nouveau chiffre
............L.append(unite) #on l'ajoute à L
........NS = NS//10 #on efface le dernier chiffre en divisant par 10
....return(len(L)) #L contient les chiffres de carre, mais cités une seule fois chacun
```

Rappel : Un élément de $\mathbb{N}$ s'appelle un nombre et les dix symboles utilisés pour l'écrire sont ses chiffres.
C'est comme mot et lettre. C'est pourtant facile. Alors pourquoi confondez vous ?

Le test proprement dit prend `n`, calcule `n*n`[5], calcule combien de chiffres différents a ce carré et regarde si ce nombre vaut 2.

```
def Test(n) :
....carre=n*n
....return(NbChiffres(carre)==2) #on retourne un test tout de suite
```

On veut les nombres PetitCarre plus petits que 10**6 (dans une liste) :

```
L = [ ]
for n in range(10**6) :
....if Test(n) :
........L.append(n)
print(L)
```

On veut les 25 premiers PetitCarre :

5. plus rapide que `n**2`

```
n, L = 0, [ ] #affectation simultanée
while len(L)<25 : #condition d'arrêt
....if Test(n) :
........L.append(n)
....n +=1 #on passe au suivant, ne pas oublier
print(L)
```

[4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 20, 21, 22, 26, 30, 38, 88, 100, 109, 173, 200, 212, 235, 264, 300, 1000, 2000, 3000, 3114, 10000, 20000, 30000, 81619, 100000, 200000]

Il y a d'autres teste plus rapides, en s'arrêtant dès qu'on a trouvé plus de deux chiffres différents.

Pour la fonction de Baire, on peut prendre les rationnels entre 0 et 1 de dénominateur plus petit que N.

```
Abscisses=[ ]
for q in range(1, N) : #un denominateur ne peut s'annuler
....for p in range(q) : #pour avoir p/q ⩽ 1, on demande p in range(q)
........Abscisses.append(p/q)
```

Mais cette liste va contenir des termes en double. En effet, 1/2 et 2/4 vont s'y retrouver.
En fait, on veut des fractions irréductibles. [6]

```
Abscisses=[ ]
for q in range(1, N) :
....for p in range(q) :
........if gcd(p, q) == 1 : #ils sont premiers entre eux
............Abscisses.append(p/q)
```

Le module `math` contient une fonction `gcd` qui calcule le p.g.c.d. de deux entiers.

Mais sinon

```
def gcd(a, b) :
....while b !=0 :
........a, b = b, a%b
....return(a)
```

On a la liste des abscisses, pas dans l'ordre mais qu'importe.
De plus on les a sous forme flottante et pas sous forme « rationnelle ». mais c'est pour un graphe.

Il nous aurait fallu aussi la liste des images $\frac{1}{q}$. On la construit en même temps, c'est ça la clef.

```
Abscisses, Ordonnees =[ ], [ ]
for q in range(1, N) :
....for p in range(q) :
........if gcd(p, q) == 1 : #ils sont premiers entre eux
............Abscisses.append(p/q) #le rationnel
............Ordonnees.append(1/q) #son image
```

Les deux listes ont la même longueur, et chaque x de l'une correspond à son y dans l'autre.

On plotte :

```
pl.plot(Abscisses, Ordonnees)
pl.show()
```

La fonction un peu renflée a remplacé la ligne `............Ordonnees.append(1/q)`
par `............Ordonnees.append(1/sqrt(q))`

Pour la suite de Fibonacci.
On fabrique la suite en utilisant sa définition.
Chaque terme est la somme des deux précédents, et les précédents sont là :

6. c'est l'oubli avec le graphe où au dessus de chaque rationnel il semble y avoir plusieurs images

```
Fibo = [0, 1] #car il faut bien commencer
for k in range(N) :
....Nouveau=Fibo[-1]+Fibo[-2] #la somme des deux derniers
....Fibo.append(Nouveau)
```

Plus simple que ça tu meurs...

Il ne reste qu'à faire un

```
pl.plot(Fibo)
pl.show()
```

Et pour la suite modulo n :

```
FiboMod = [0, 1] #car il faut bien commencer
for k in range(N) :
....Nouveau=(FiboMod[-1]+FiboMod[-2])%n #la somme des deux derniers
....FiboMod.append(Nouveau)
pl.plot(FiboMod)
pl.show()
```

D'après les graphes, il suffit de voir si en prenant n égal à 198, 199, 200, 201, 202 ou un nombre un peu comme ça.