DABERT DARIUS
DAIRAIN LEO
MPSI 1

Informatique

Projet n°1

-

Multiplication Modulaire

Introduction

1) OBJECTIF

L'objectif de ce porjet projet était de produire un code permettant la construction de figures correspondant à la représentation graphique de tables de multiplication modulaire, selon les règles de constructions énoncées dans le sujet. Pour cela, il était nécessaire de définir des fonctions mathématiques relatives aux produits modulaires, ainsi que des fonctions graphiques permettant de construire les figures en elles-mêmes. Pour cela nous avons utilisé les outils de deux modules de Python : le module math pour la partie mathématique, et le module Tkinter pour la partie graphique.

Nous avons pris l'initiative d'incorporer des curseurs permettant à l'utilisateur de faire varier l'entier dont on construit la table, ainsi que l'entier qui sert de modulo, afin de faciliter la visualisation des différentes figures obtenues par combinaisons différentes de ces deux entiers. Nous avons également réalisé un prolongement du sujet aux nombres décimaux, qui sera détaillé dans une partie ultérieure.

2) DEMARCHE DE RESOLUTION

Afin de répondre au problème posé, nous nous sommes tout d'abord intéressés a l'aspect mathématique du sujet, c'est à dire à la manière de déterminer le produit modulaire de deux entiers. Nous avons ensuite effectué des recherches sur les modules graphiques de python afin de déterminer de quelle manière ils nous étaient possible de réaliser les figures souhaitées. Nous avons choisit d'utiliser le module Tkinter car il permet la réalisation des figures de manière efficace, ainsi que la création d'une interface facilitant la visualisation des nombreuses figures disponibles via les différentes combinaisons des nombres a et p. Lors de la réalisation de la partie graphique nous avons tout d'abord taché d'appréhender les fonctionnalités de Tkinter afin de réaliser les figures en elles-mêmes, puis nous nous sommes penchés sur la façon de permettre une observation plus simple et plus complète des représentations graphiques, ce qui nous a mener à construire des curseurs permettant de faire varier les deux paramètres de la table et d'observer les modifications directement sur la figure.

Résolution du problème

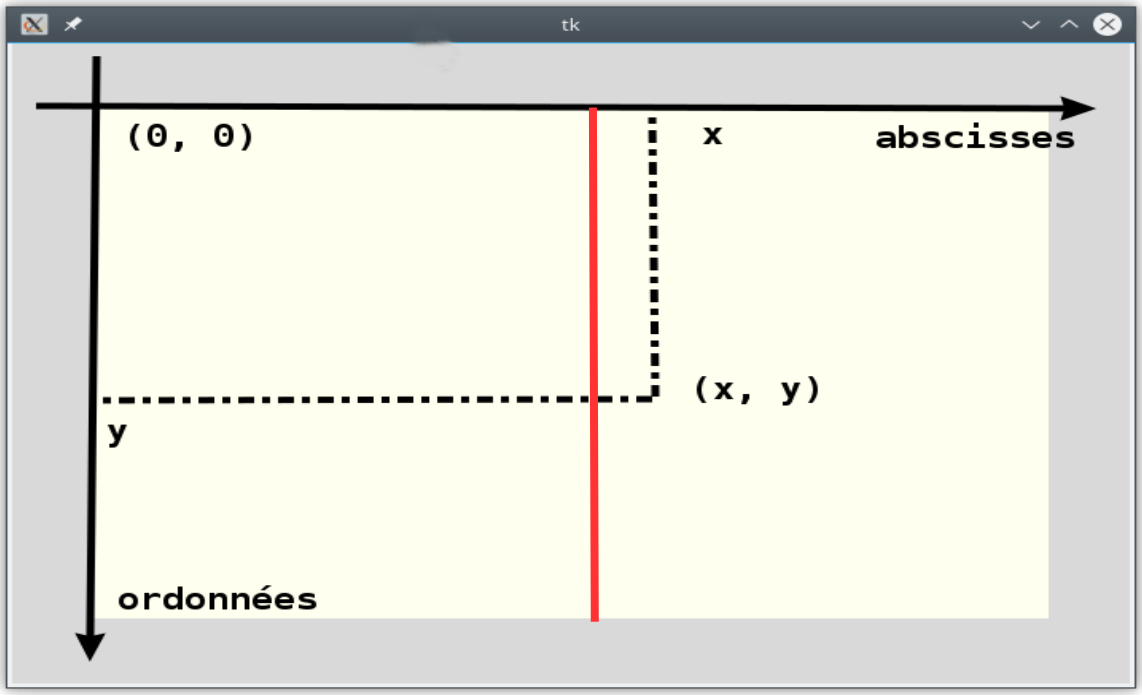
1) RAFFINAGE

Afin de simplifier la résolution du problème posé, nous avons tout d'abord réalisé un découpage du travail en plusieurs parties :

- Création d'une représentation graphique d'une table de multiplication modulaire
  - ➢ Calculer une table de multiplication modulaire
    - Calculer le produit modulaire de deux entiers
    - *(Calculer le produit modulaire de d'un nombre décimal et d'un entier)*
    - Itérer le procédé en faisant varier un des deux entiers pour créer la table
  - ➢ Afficher sa représentation graphique
    - Créer la figure correspondant à la table
      - Créer le cercle qui sert de base à la figure
        - Détermination des coordonnées du cercle et de son rayon
      - Positionner les points sur le cercle
        - Positionner p points équidistants sur le cercle (correspondant aux entiers compris entre 0 et p-1)
          - Détermination des coordonnées de chaque point
      - Relier chaque point au point correspondant à son image par la table
        - Créer un segment reliant des points définis
    - Afficher la figure à l'écran
      - Créer une fenêtre qui s'ouvre lorsque l'on exécute le programme
      - Placer les éléments définissant la figure dans cette fenêtre
    - *(Permettre la modification des paramètres et l'observation des résultats simplement*
      - *Permettre de modifier directement, depuis la fenêtre où se trouve la figure, les paramètres*
        - *Création de curseurs présents sur la même fenêtre que la figure, pour faire varier les paramètres*
      - *Actualiser la figure à chaque modification pour observer les nouvelles figures immédiatement)*

NB : Les parties de raffinage entre parenthèses sont des éléments qui ne figuraient pas expressément dans le sujet, mais qui sont des extensions que nous avons jugées pertinentes lors de notre réflexion sur le projet, et que nous avions donc inclus dan notre découpage du problème assez tôt, et qui font donc partie intégrante de la résolution du problème.

passageRepereTkinter(x,y, a, b) :



Utilité de la fonction :

Tout d'abord, le module Tkinter de Python utilise un système d'axe différents du repère classique qu'on a l'habitude d'utiliser en mathématiques par exemple. En effet, son origine est situé en haut à gauche de la fenêtre graphique, la où on a l'habitude d'un repère dont l'origine est au centre de la fenêtre. Pour simplifier nos calculs, on a donc choisi d'écrire une fonction « passageRepereTkinter(x,y, a, b) » qui permet de passer des coordonnées d'un point dans le repère «classique» dont l'origine est au centre, à ses coordonnées dans le repère de Tkinter.

Explication du code :

Si on note (a,b) les coordonnées de l'origine du repère centré dans le repère de Tkinter et (x,y) les coordonnées d'un point quelconque dans le repère centré, alors il apparaît que ce même point aura pour coordonnées (x+a,y-b).

ProduitModuloP(k, n, p) :
Utilité de la fonction :

Cette fonction est utile pour alléger le code lorsqu'on calcule le reste de la division euclidienne d'un multiple dont on construit la table par le nombre de point,c'est à dire du modulo.

Explication du code :

On utilise juste les commandes natives de python a savoir :

- «*» la multiplication
- «%» le reste d'une division euclidienne

–
DessinerSegment(cnv, k, n, p, ListePoints, a, b):

Utilité de la fonction :

Il s'agit ici d'une fonction graphique, celle ci permet de créer des segments reliant deux points.

Explication du code :

Soit a l'entier dont on construit la table et p le nombre de points.

On commence par créer deux points en définissant leur coordonnées dans le repère centré, on commence par un point A, on lui associe un élément ' k ' de la liste contenant les coordonnées de chaque point du cercle, ensuite à un point B on associe les coordonnées du point de la liste correspondant au produit de ' k ' par a modulo b. On transpose les coordonnées des deux points dans le repère de Tkinter à l'aide de la fonction passageRepereTkinter(x,y, a, b). On utilise enfin la commande ' cnv.create_line(A, B) ' qui crée le segment reliant A à B.

CreerPoint(cnv, C, R=2, color='red') :

Utilité de la fonction :

Il s'agit encore ici d'une fonction graphique, celle ci permet de créer un cercle dont on choisira le rayon petits pour représentés les points du cercle.

DessinerPoint(cnv, ListePoints, a, b):

Utilité de la fonction :

Cette fonction va utiliser CreerPoint pour construire, si on note p le modulo du produit modulaire, les p sommets du cercles.

Explication du code :

On utilise ici une boucle for qui à chaque itérations va créer un point en parcourant chaque élément de la liste des coordonnées de chaque point du cercle et en utilisant la fonction CreerPoint.

show(cnv, n, p):

Utilité de la fonction :

Cette fonction est celle qui permet d'afficher à l'écran dans le canevas crées tout les éléments construits précédemment, à savoir : les points et les segments.

Explication du code :

On commence par supprimer tout ce qui était affiché sur le canevas pour éviter une superposition. On créée ensuite une variable ' t ' qui reçoit la valeur de l'angle séparant chaque point du cercle. Cette angle va nous servir à créer la liste de tout les points à partir des fonctions cos et sin importés du module math. Cette liste est créée à l'aide d'une instruction

for qui va crée les coordonnées de tout les points utiles à la construction. On soustraie de plus -pi/2 à l'intérieur des fonctions cos et sin pour redresser la figure puis on multiplie les valeurs par un rayon ' R '. On applique ensuite la fonction DessinerSegment à chaque point du cercle pour finaliser la construction. Enfin on relie tout les points en tracant un cercle de rayon ' R ' et de centre le centre de la fenêtre. Le calcul du centre est fait à partir de la taille de la fenêtre defini dans la fonction ProgrammeEntier(), on translate enfin ses coordonnées par le même calcul que celui de la fonction passageRepereTkinter(x,y, a, b) :

ProgrammeEntier() :

Utilité de la fonction :

Cette fonction est en quelque sorte le corps du programme. Elle permet d'ouvrir la fenêtre graphique sur lequel est dessiné la figure. Pour cela elle utilise les fonctions définis précédemment. Elle contient de plus la création de deux curseurs permettant de faire varier plus aisément les nombres utilisé pour la représentation.

Explication du code :

Tout d'abord, la première ligne est une déclaration de variable globale. Il est donc nécessaire d'expliquer ce choix : pour cnv, R, a, b, il s'agit d'un problème de lecture des variables. En effet, ces variables sont utilisés dans d'autre fonctions et sans déclaration global il faudrait redéfinir les fonctions et placer ces variables en paramètres puis leur assigner leur valeur en dehors des fonctions, mais alors on perdrait en lisibilité. Pour N et P, le problème est différent, en effet, l'utilisation d'un curseur impose une réactualisation de leur valeur à chaque fois que le curseur est bougé, et plus particulièrement dans les fonctions table et modulo.

Ensuite on assigne une valeur à R qui permettra de définir la taille des objets construits sur Tkinter, à savoir le rayon du cercle et les dimensions de la fenêtre (par intermédiaires des variables 'a' et 'b' qui valent 1,2 * R afin de laisser une marge par rapport au contour du cercle.

La suite du code est ensuite directement issue de la syntaxe du module Tkinter et permet la creation du canevas et des curseurs.

NB : les explications sont strictement identiques pour la fonction ProgrammeDecimaux() car seul un paramètre de la création du curseur (à savoir le pas) est modifié.

ChoixRepresentation() :

Utilité de la fonction :

Cette fonction permet à l'utilisateur qui lance le programme de choisir si les tables de multiplication modulaire seront construites à partir d'entier ou de nombre décimaux

Explication du code :

Il s'agit ici encore seulement d'une syntaxe du module Tkinter créant deux boutons lancant chacun une fonction différente, à savoir ProgrammeEntier() ou ProgrammeDecimaux().

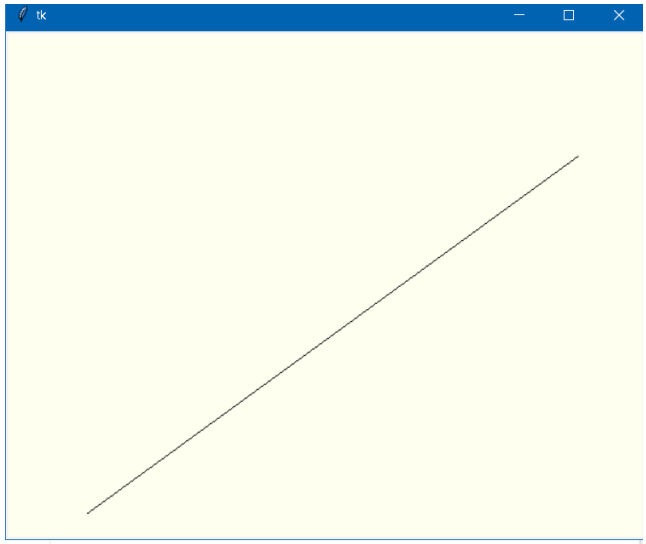
Ici je te met l'explication du floor dans la fonction ProduitModuloP() :

Pour le calcul des valeurs de la table de n, par exemple de k×n, on effectue à nouveau le produit, ce qui donne un flottant (puisque n est flottant cette fois) qu'on va appeler u. Prendre le reste d'un flottant par un entier a un sens, par exemple, le reste de 42,75 par 10 est le flottant 2,75 et se calcule par 42.75 % 10. Donc, on peut prendre le reste r de u modulo p, ce qui donne un flottant. Pour pouvoir placer le point correspondant sur le cercle, on prendra tout simplement la partie entière int(r) de ce reste.

Lorsqu'il s'agit de nombre décimaux, le reste de la divison euclidienne d'un flottant par un entier a toujours un sens et est un flottant, par exemple on peut écrire 15,5 = 3*5 +0,5 avec 0<=0,5<5 donc le couple quotient reste de la divison euclidienne de 15,5 par 5 est (3;0,5), on definit ainsi le produit modulaire décimal. La fonction ProduitModuloP() renvoie ainsi une flottant et non un entier, pour placer ce reste sur le cercle il faut donc prendre la partie entière de ce reste, or comme la partie entière d'un entier est le même entier, on peut rajouter dans la fonction ProduitModuloP() une instruction floor dans le return.

Ici je te met des screen de test :

```
from tkinter import Tk, Canvas, Scale , Button
from math import sin, cos, pi,floor
def DessinerSegment(cnv, k, n, p, ListePoints, a, b):
    """
    Entrée :
    cnv : canevas
    k : point du cercle d'indice k
    n : entier dont on construit la table
    p : nombre de sommet(modulo)
    base : liste contenant les coordonées de chaque sommet du cercle
    a,b : Coordonnées du centre du repère classique
    Sortie :
    le tracé d un segement reliant un point a son produit par n modulo p
    """
    prod=ProduitModuloP(k, n, p)
    xA, yA=ListePoints[k]
    xB, yB=ListePoints[prod]
    A=passageRepereTkinter(xA, yA, a, b)
    B=passageRepereTkinter(xB, yB, a, b)
    cnv.create_line(A, B)
def ProduitModuloP(k, n, p):
    """
    Entrée :
    k : k-ieme multiple de n
    n : entier dont on construit la table
    p : nombre de sommet(modulo)
    Sortie :
    La partie entiere du reste de la division de k*n par p
    """
    return floor(k*n % p)
def passageRepereTkinter(x,y, a, b):
    """
    Entrée :
    a,b : Coordonnées du centre du repère classique
    x,y : coordonées d'un point dans le repère 'classique'
    Sortie:
    Coordonées du point dans le repère Tkinter
    """
    return (a+x, b-y)
t=2*pi/p
R=300 # paramètre permettant de definir la taille de la fenêtre
N=P=2 #position de départ
a=b=1.2*R #pour avoir une marge sur les bords
master = Tk()
cnv=Canvas(master, width=2*a, height=2*b, bg='ivory') #création du canevas
cnv.pack(side="left")
p = 10
ListePoints=[(R*cos(k*t-pi/2), R*sin(k*t-pi/2)) for k in range(p)]
DessinerSegment(cnv, 4, 7, 10, ListePoints, a, b)
master.mainloop()
```



```
from tkinter import Tk, Canvas, Scale , Button
from math import sin, cos, pi,floor

def CreerPoint(cnv, C, R=2, color='red'):
    """
    Entrée :
    cnv : canevas
    C : Coordonées du centre du cercle dans le repère de Tkinter
    R : rayon du cercle
    color : couleur du cercle (rouge par defaut)
    Sortie
    un cercle de centre C(repère classique) et de rayon R
    """
    xC, yC=C
    return cnv.create_oval(xC-R,yC-R,xC+R, yC+R, fill=color,
                           outline=color)
def DessinerPoint(cnv, ListePoints, a, b):
    """
    Entrée:
    cnv : le canevas sur lequel on dessine
    ListePoints : liste contenant les coordonnées de chaque points
    a,b : Coordonnées du centre du repère classique
    Sortie :
    Le dessin sur le canevas de l'integralité des points de la liste
    """
    p=len(ListePoints)
    for k in range(p):
        x, y=ListePoints[k]
        X, Y=passageRepereTkinter(x, y, a, b)
        CreerPoint(cnv, (X, Y), R=3, color='black')
def passageRepereTkinter(x,y, a, b):
    """
    Entrée :
    a,b : Coordonnées du centre du repère classique
    x,y : coordonées d'un point dans le repère 'classique'
    Sortie:
    Coordonées du point dans le repère Tkinter
    """
    return (a+x, b-y)
t=2*pi/p
R=150 # paramètre permettant de definir la taille de la fenêtre
N=P=2 #position de départ
a=b=1.2*R #pour avoir une marge sur les bords
master = Tk()
cnv=Canvas(master, width=2*a, height=2*b, bg='ivory') #création du canevas
cnv.pack(side="left")
p = 10
ListePoints=[(R*cos(k*t-pi/2), R*sin(k*t-pi/2)) for k in range(p)]
DessinerPoint(cnv, ListePoints, a, b)
master.mainloop()
```