Planche 1 :

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import random as rd
from math import sqrt, pi

# Q1  :

def creation (n)  :
    return [rd.randint(1,n+1) for i in range(n+1)]

# principe des tiroirs.

# Q2  :

# naif  :
def indice (L)  :
    n = len(L)
    for i in range(1,n)  :
        for j in range(i)  :
            if L[j] == L[i]  :
                return i
    return(0)

# recursif

def indice_rec(L)  :
    if L ==[]  :
        return 0
    else  :
        ind = indice_rec (L[ :-1])
        if ind ==0  :
            if L[-1] in L[ :-1]  : return (len(L)-1)
            else  : return 0
        else  : return ind

# Q3  :

def moyenne (m,n)  :
    somme = 0
    for j in range(m)  :
        L = creation(n)
        somme += indice(L)
    return somme/m

m = 1000
X = [n for n in range(10,101)]
```

```
Y = [moyenne(m,n) for n in X]
Z = [sqrt(n) for n in X]

plt.clf()
plt.plot(X,Y,label="moyenne de l'indice en fonction de n")
plt.plot(X,Z,label="racine carrée de n")
plt.legend()
plt.savefig("Q3.png")

# Conjecture  : la moyenne est égale à sqrt(n)*constante légérement
# supérieure à 1

# Q4

# P(X_n = 1) = 1/n
# P(X_n > 1) = 1-P(X_n=1) = 1- 1/n
# P(X_n > 2) = P(X_n > 2 | X_n > 1)*P(X_n > 1)=(1 - 2/n)*(1 - 1/n)
# P(X_n > j+1) = P(X_n > j+1 | X_n > j)*P(X_n > j) = (1-j/n)*P(X_n > j)
# et on récurre.

# Q5.a

# pour l'inégalité, simple étude de fonction
# (ou alors on utilise la série entière)
# P(X_n>j) <= produit(exp(-i/n))=exp((-1/n)*somme i)=exp(-(j(j+1))/n)
# <= exp(-j**2/n)

# Q5.b

# E(X_n) = somme de i=1..n ( i*P(X_n=i)) = somme(i*(P(X_n>i-1) - P(X_n>i)),i=1..n)
# = somme((i-1)*P(X_n>i-1)-i*P(X_n>i) + P(X_n>i-1),i=1..n)
# = 0 -nP(X_n>n) + somme(P(X_n>i),i=0..n-1) = somme(P(X_n>i), i=0..n) car P(X_n>n)=0

# Q5.c  : Comparaison série / intégrale
"""
for i in range(10,301,20)  :
    print(moyenne(100,i), sqrt(i*(pi/2)))
"""
```

Planche 2 :

```
import numpy as np
from scipy.integrate import quad
from math import pi, log, exp
import matplotlib.pyplot as plt

# Q1  :  t -> abs(P(exp(it)) est continue et strictement positive
# donc son ln est une fonction définie et continue, donc
# M(P) existe.
```

```
# Q2  : S  : nombre de racines distinctes, z_k les racines, m_k leurs
# multiplicités, A 2pi*log du module du coefficient dominant de P.

# Q3  :



def Mahler(r,theta)  :
    G = lambda t  : log(abs(np.exp(t*1j)-r*np.exp(theta*1j)))
    return quad(G, 0, 2*pi)[0]

"""
n = 1000
a = 0
b = 40
R = [2.5,5,100,2017]
X = [a+k*(b-a)/n for k in range(n+1)]
plt.clf()
for r in R  :
    Y = [Mahler(r,theta) for theta in X]
    plt.plot(X,Y)
plt.savefig("Q3.png")
"""

# Ca a l'air constant tout ça.

# Q4  :

n = 1000
a = 0
b = 100
Theta = [1,10,50,90]
X = [a+k*(b-a)/n for k in range(n+1)]
plt.clf()
for theta in Theta  :
    Y = [Mahler(r,theta) for r in X]
    plt.plot(X,Y)
plt.savefig("Q4.png")

# Ca se confirme.

# Q5  : on passe à partie réelle + partie imaginaire, on développe
# on arrive à intégrale de 0 à 2pi de (1/2)ln(1+r**2-2rcos(t+theta) dt
# on pose u = t+theta, theta n'apparaît plus que dans les bornes de l'intégrale
# de theta à 2pi+theta, et par 2pi-périodicité, ça ne dépend pas de theta.
```

Planche 3 :

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
```

```
import random as rd

# Q1  :

def Q1()  :
    L =[1,1]
    for i in range(18)  :# invariant  : L contient les i+2 premières valeurs de F
        L.append(L[i] + L[i+1])
        # invariant sortie  : L contient les i+3 premières valeurs de F
    print(L)
    return L

# Q2  :

F = Q1()

def Q2()  :
    for n in range(10)  :
        alpha = F[n+1]/F[n+2]
        beta = F[n]/F[n+3]
        for p in range(10)  :
            S = 4*sum([(-1)**k*(alpha**(2*k+1)+beta**(2*k+1))/(2*k+1) for k in range(p+1)])
            print("S("+str(n)+","+str(p)+")="+str(S))

# on conjecture que pour tout n, S(n,p) -> pi quand p -> + infinity

# Q3  :

for n in range(9)  :
    T = (F[n+2]+F[n+1]*1j)*(F[n+3]+F[n]*1j)
    print(T)

# T = entier strictement positif * pi/4

# démo  :  on développe  :
# Re(T) = F[n+2]F[n+3]-F[n]F[n+1]=F[n+2](F[n+1]+F[n+2])-F[n]F[n+1]
#       = F[n+2]**2 + F[n+1]**2
# Avec un calcul du même genre on trouve la même chose pour Im(T), donc
# la conjecture sur est vérifiée, et arg(T) = pi/4.
# Or arg(T) = arg(F[n+2]+iF[n+1]) + arg(F[n+3]+iF[n])=arctan alpha + arctan beta
# et là on reconnaît que Sn est la série entière tendant vers
# arctan(alpha)+arctan(beta), donc vers arg(T), donc vers pi/4.


# Q4  :

# Rp = arctan x - somme( (-1)**k x**(2k+1)/(2k+1) , k=0..p
# = integrale(1/(1+t**2) - somme( [-t**2]**k , k=0..p) , t = 0..x)
# = integrale ( 1/(1+t**2) - (1-(-t**2)**(p+1))/(1-(-t)**2), t=0..x)
# = ce qu'on voulait.
```

```
# Q5 : on encadre : t**(2p+2)/(1+t**2)<=t**(2p+2)
# et de plus : 1-t**2 <= 1/(1+t**2) donc t**(2p+2)-t**(2p+4) <= t**(2p+2)/(1+t**2)
# on intègre, on a un équivalent : 4*[(-1)**(p+3)/(2p+3)*(alpha_n**(2p+3)+beta_n**(2p+3)))]
# car il faut bien remarquer que alpha_n et beta_n tendent vers 0
```

Planche 4 :

```
import matplotlib.pyplot as plt
import numpy as np
import random as rd
import numpy.linalg as lin


# Q1 ;

def position(n) :
    pos=[1]
    for i in range(n-1) :
        if pos[-1] == 1 : pos.append(rd.randint(1,4))
        else : pos.append(pos[-1]-1)
    return pos

def trace_pos() :
    plt.clf()
    N=[10,50,100]
    for n in N :
        X=[k for k in range(n)]
        Y=position(n)
        plt.plot(X,Y,label="n="+str(n))
    plt.legend()
    plt.savefig("position.png")


# Conjecture : plus la valeur est basse, plus souvent elle est atteinte

# Q3.a : A = [[1/4, 1, 0, 0],
#             [1/4, 0, 1, 0],
#             [1/4, 0, 0, 1],
#             [1/4, 0, 0, 0]]
A=np.array([[1/4,1,0,0],[1/4,0,1,0],[1/4,0,0,1],[1/4,0,0,0]])

# Q3.b : c'est un matrice compagnon de polynôme caractéristique
# X**4-1/4(X**3+X**2+X+1)

poca = np.poly(A)
lin.eigvals(A) # on trouve 4 racines distinctes, dont deux ne sont pas réelles
# donc diagonalisable dans C mais pas dans R.


# Q3.c : ces racines sont : 1, et les autres de module <1
```

```
# donc A**n converge vers P((1,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0))P-1.
# Donc U_n converge vers cte * le vecteur propre associé à la vp 1, et la somme
# des coordonnées vaut 1. On résout. On trouve 1/10(4,3,2,1).

# Q4.a  :

def compte(n)  :
    a = position(n+1)
    Y = [0, 0, 0, 0]
    for p in a  :
        Y[p-1] += 1
    return Y

# Q5.b  :

def test()  :
    C = []
    for i in range(100)  :
        C.append(compte(100))
    return C

# Q5.c  :

def esperance()  :
    C=test()
    return [(1/101)*sum([C[j][i] for j in range(len(C))]) for i in range(4)]

# ça colle.
```