T exp

Interrogation écrite du jeudi 5 janvier 2023 (30 minutes)

Numéro : ….. **Prénom et nom :** ……

Note : ….. / 20

I. (6 points : 1 point + 1 point + 1 point + 1 point + 1 point + 1 point)

On considère la suite complexe (z_n) définie sur $\mathbb{N}$ par son premier terme $z_0 = 1+\mathrm{i}$ et par la relation de récurrence $z_{n+1} = z_n^{\,2} + 4$ pour tout entier naturel n.
On considère le programme Python donné dans le cadre ci-dessous qui a pour objectif de trouver le plus petit entier naturel n tel que $\mathrm{Im}\, z_n < 0$.
Compléter les pointillés.

```
z=1+1j
n=....
while z.imag ..... 0:
      z=.....
      n=.......
print(....)
```

Quelle valeur obtient-on lorsque l'on exécute ce programme ? … (répondre sans écrire d'égalité)

II. (7 points : 1°) 2 points + 1 point ; 2°) 2 points + 2 points)

Dans cet exercice, on s'intéresse à deux jeux pour lesquels on dispose d'une urne contenant n boules indiscernables au toucher numérotées de 1 à n, n étant un entier naturel supérieur ou égal à 1.
Dans les programmes Python considérés, on suppose que la fonction `randint` de la bibliothèque `random` a été préalablement importée.

1°) **Jeu 1**
Le joueur tire au hasard une boule dans l'urne. Il gagne si le numéro de la boule tirée est un multiple de 3 ou de 4.

• On considère la fonction Python d'en-tête `def jeu_1(n):` donnée dans le cadre ci-dessous qui simule une partie de ce jeu. Compléter les pointillés au niveau du `if`.

```
def jeu_1(n):
    r=randint(1,n)
    if .....==0 or .....==0:
      print('gagné')
    else :
      print('perdu')
```

• On prend $n = 30$. Quelle est la probabilité de gagner à ce jeu ? …….

2°) **Jeu 2**

Le joueur tire au hasard successivement, avec remise, deux boules dans l'urne. Il gagne si les numéros des boules tirées sont deux entiers consécutifs.

• On considère la fonction Python d'en-tête `def jeu_2(n):` donnée dans le cadre ci-dessous à gauche qui simule une partie de ce jeu. Compléter les pointillés au niveau du `if`.

```
def jeu_2(n):
    r1=randint(1,n)
    r2=randint(1,n)
    if r1==..... or r2==.....:
      print('gagné')
    else :
      print('perdu')
```

```
def jeu_2_var(n):
    r1=randint(1,n)
    r2=randint(1,n)
    if abs(.....)==...:
      print('gagné')
    else :
      print('perdu')
```

• On considère la fonction Python d'en-tête `def jeu_2_var(n):` donnée dans le cadre à droite ci-dessus qui est une variante de la fonction précédente utilisant la fonction `abs(...)` donnant la valeur absolue d'un nombre. Compléter les pointillés au niveau du `if`.

• **Bonus sur 1 point :** Quelle est la probabilité de gagner à ce jeu ? .....

On donnera la réponse sous la forme d'un quotient en fonction de *n*, sans écrire d'égalité.

III. (6 points : 1 point + 1 point + 2 points + 1 point + 1 point)

Pour tout entier naturel $n \geqslant 1$, on note u_n l'entier naturel dont l'écriture en base dix comporte *n* fois le chiffre 1. On peut donc écrire $u_n = \overline{11...1}^{(\text{dix})}$, le chiffre 1 étant répété *n* fois.

On considère la fonction Python d'en-tête `def rep_unit(n):` donnée dans le cadre ci-dessous qui prend pour argument un entier naturel *n* supérieur ou égal à 1 et qui a pour objectif de renvoyer la liste des entiers u_1, u_2, …, u_n dans cet ordre.

Compléter les pointillés. On pourra observer que pour tout entier naturel $n \geqslant 1$, on a $u_{n+1} = 10u_n + 1$.

```
def rep_unit(n):
    u=1
    L=[u]
    for i in range(...,...):
        u=.............
        L.append(...)
    return ...
```

IV. (1 point)

On considère la fonction Python d'en-tête `def restes_div_eucl_par_3(L):` donnée dans le cadre ci-dessous qui prend pour argument une liste L d'entiers relatifs et qui a pour objectif de renvoyer la liste M des restes dans la division euclidienne par 3 des éléments de L, dans le même ordre.

Compléter les pointillés.

```
def restes_div_eucl_par_3(L):
    M=[.......... for x in L]
    return M
```

Corrigé de l'interrogation écrite du 5-1-2023

I.

On considère la suite complexe (z_n) définie sur $\mathbb{N}$ par son premier terme $z_0 = 1+\mathrm{i}$ et par la relation de récurrence $z_{n+1} = {z_n}^2 + 4$ pour tout entier naturel *n*.
On considère le programme Python donné dans le cadre ci-dessous qui a pour objectif de trouver le plus petit entier naturel *n* tel que $\mathrm{Im}\, z_n < 0$.
Compléter les pointillés.

```
z=1+1j
n=0
while z.imag>=0:
      z=z**2+4
      n=n+1
print(n)
```

Quelle valeur obtient-on lorsque l'on exécute ce programme ? 5 (répondre sans écrire d'égalité)

On peut obtenir ce résultat en effectuant le programme sur la calculatrice ou en calculant les premiers termes de la suite, avec la calculatrice pour gagner du temps.

L'onglet « Suites » de la calculatrice Numworks ne fonctionne que pour des suites réelles
On ne peut donc pas l'utiliser ici.

On va dans « Calculs » et on utilise la touche [Ans].

On tape 1 + i puis on appuie sur la touche [EXE].

Ensuite, on appuie sur la touche [Ans] puis sur la touche [x^2] (affichage : $\mathrm{Ans}^2 + 4$).
En appuyant plusieurs fois de suite sur la touche [EXE], on obtient les termes successifs de la suite.

$z_1 = 4 + 2\mathrm{i}$
$z_2 = 16 + 16\mathrm{i}$
$z_3 = 4 + 512\mathrm{i}$
$z_4 = -262128 + 4096\mathrm{i}$
$z_5 = 68094311168 - 2147252576\mathrm{i}$

On va dire que le plus petit entier naturel *n* tel que $\mathrm{Im}\, z_n < 0$ est 5.
On ne peut pas dire « à partir de l'indice 5 ».

II.

Dans cet exercice, on s'intéresse à deux jeux pour lesquels on dispose d'une urne contenant *n* boules indiscernables au toucher numérotées de 1 à *n*, *n* étant un entier naturel supérieur ou égal à 1.
Dans les programmes Python considérés, on suppose que la fonction `randint` de la bibliothèque `random` a été préalablement importée.

1°) **Jeu 1**
Le joueur tire au hasard une boule dans l'urne. Il gagne si le numéro de la boule tirée est un multiple de 3 ou de 4.

• On considère la fonction Python d'en-tête `def jeu_1(n):` donnée dans le cadre ci-dessous qui simule une partie de ce jeu. Compléter les pointillés au niveau du `if`.

```
def jeu_1(n):
    r=randint(1,n)
    if r%3==0 or r%4==0:
      print('gagné')
    else :
      print('perdu')
```

L'instruction `r=randint(1,n)` permet de choisir un entier au hasard entre 1 et *n*.
Elle permet de simuler le tirage d'une boule au hasard dans l'urne car les boules sont numérotées de 1 à *n*.

• On prend $n = 30$. Quelle est la probabilité de gagner à ce jeu ? $\frac{15}{30} = \frac{1}{2}$

Les résultats possibles pour l'événement considéré sont : 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 4, 8, 16, 20, 28.

2°) **Jeu 2**
Le joueur tire au hasard successivement, avec remise, deux boules dans l'urne. Il gagne si les numéros des boules tirées sont deux entiers consécutifs.
• On considère la fonction Python d'en-tête `def jeu_2(n):` donnée dans le cadre ci-dessous à gauche qui simule une partie de ce jeu. Compléter les pointillés au niveau du `if`.

```
def jeu_2(n):
    r1=randint(1,n)
    r2=randint(1,n)
    if r1==r2+1 or r2==r1+1:
      print('gagné')
    else :
      print('perdu')
```

```
def jeu_2_var(n):
    r1=randint(1,n)
    r2=randint(1,n)
    if abs(r1-r2)==1:
      print('gagné')
    else :
      print('perdu')
```

Dans le programme de gauche, on peut aussi écrire `r1==r2-1 or r2==r1-1:` .

`r1` désigne le numéro de la première boule tirée.
`r2` désigne le numéro de la deuxième boule tirée.

• On considère la fonction Python d'en-tête `def jeu_2_var(n):` donnée dans le cadre à droite ci-dessus qui est une variante de la fonction précédente utilisant la fonction `abs(...)` donnant la valeur absolue d'un nombre. Compléter les pointillés au niveau du `if`.

• **Bonus sur 1 point :** Quelle est la probabilité de gagner à ce jeu ? $\dfrac{2(n-1)}{n^2}$

On donnera la réponse sous la forme d'un quotient en fonction de *n*, sans écrire d'égalité.

L'univers des possibles de l'expérience aléatoire est $\Omega = [\![1\,;n]\!]^2$.

On note *P* la probabilité uniforme sur Ω (loi d'équiprobabilité).

On note A l'événement : « Le joueur gagne au jeu ».

Calculons $P(\mathrm{A})$.

Une manière de modéliser l'expérience aléatoire consiste à utiliser un tableau à double entrée ou un arbre.

On a donc $\operatorname{card}\Omega = n^2$.

1er cas : On suppose que *n* est quelconque supérieur ou égal à 2.

Les tirages possibles pour A sont : $(1\,;2)$, $(2\,;1)$, $(2\,;3)$, $(3\,;2)$, $(3\,;4)$, $(4\,;3)$, …, $(n-1\,;n)$, $(n\,;n-1)$.

$(1\,;2)$, $(2\,;3)$, $(3\,;4)$,…, $(n-1\,;n)$: $n-1$ couples
$(2\,;1)$, $(3\,;2)$, $(4\,;3)$, …, $(n\,;n-1)$: $n-1$ couples

L'ordre intervient dans les couples.

$(n-1)+(n-1)=2(n-1)$

Comme *P* est la loi de probabilité uniforme sur Ω, on a $P(\mathrm{A}) = \dfrac{\operatorname{card}\mathrm{A}}{\operatorname{card}\Omega} = \dfrac{2(n-1)}{n^2}$.

2e cas : On suppose que $n=1$.

Dans ce cas, la formule précédente fonctionne encore car A est l'événement impossible donc sa probabilité est nulle.

Si on remplace *n* par 1, on obtient $\dfrac{2(1-1)}{1^2} = \dfrac{0}{1} = 0$.

III.

Pour tout entier naturel $n \geqslant 1$, on note u_n l'entier naturel dont l'écriture en base dix comporte *n* fois le chiffre 1. On peut donc écrire $u_n = \overline{11...1}^{(\text{dix})}$, le chiffre 1 étant répété *n* fois.
On considère la fonction Python d'en-tête `def rep_unit(n):` donnée dans le cadre ci-dessous qui prend pour argument un entier naturel *n* supérieur ou égal à 1 et qui a pour objectif de renvoyer la liste des entiers u_1, u_2, …, u_n dans cet ordre.
Compléter les pointillés. On pourra observer que pour tout entier naturel $n \geqslant 1$, on a $u_{n+1} = 10u_n + 1$.

```
def rep_unit(n):
    u=1
    L=[u]
    for i in range(1,n):
        u=10*u+1
        L.append(u)
    return L
```

IV.

On considère la fonction Python d'en-tête `def restes_div_eucl_par_3(L):` donnée dans le cadre ci-dessous qui prend pour argument une liste L d'entiers relatifs et qui a pour objectif de renvoyer la liste M des restes dans la division euclidienne par 3 des éléments de L, dans le même ordre.
Compléter les pointillés.

```
def restes_div_eucl_par_3(L):
    M=[x%3 for x in L]
    return M
```