- TD 1 -

Bekkali Hamza

09/3/2024

1 **

✣ - **Q1** - Ecrire les algorithmes permettant de générer et d'afficher les séries de valeurs suivantes :
a) 5 5 5 5 5 5 5 5 5 5
b) 0 1 2 3 4 5 6 7 8 9
c) 20 21 22 23 24 25 26 27 28 29 30
d) 10 9 8 7 6 5 4 3 2 1 0
e) 1 2 4 8 16 32 64 128 256 512 1024
f) 1 1 2 3 5 8 13 21 34 55

```python
# Série de 5
serie_a = [5] * 10
print(serie_a)
#
# Série de nombres croissants
serie_b = list(range(10))
print(serie_b)
#
# Série de nombres croissants à partir de 20
serie_c = list(range(20, 31))
print(serie_c)
#
# Série de nombres décroissants à partir de 10
serie_d = list(range(10, -1, -1))
print(serie_d)
#
# Série de puissances de 2
serie_e = [2 ** i for i in range(11)]
print(serie_e)
#
# Série de Fibonacci
serie_f = [1, 1]
for i in range(2, 10):
    serie_f.append(serie_f[i-1] + serie_f[i-2])
print(serie_f)
```

✣ - **Q2** - Une tortue effectue 100 pas sur un chemin rectiligne. A chaque pas, elle avance d'une distance aléatoire d (en cm), avec $0 < d < 5$.
1- Ecrire l'algorithme qui affiche, pour chacun des 100 pas, le numéro du pas et la distance

parcourue depuis le départ.
2- Ajouter un message qui indique si la tortue a parcouru au moins 250 cm après avoir marché 100 pas.
3- Modifier l'algorithme pour que la tortue s'arrête dès qu'elle a parcouru 250 cm, et afficher alors le nombre de pas qui ont été nécessaires.
4- Ecrire les programmes correspondant aux questions 2 et 3.

```python
import random

distance_totale = 0
for pas in range(1, 101):
    distance = random.uniform(0, 5)
    distance_totale += distance
    print(f"Pas {pas}: Distance parcourue depuis le départ =
    ↪ {distance_totale} cm")
#########
if distance_totale >= 250:
    print("La tortue a parcouru au moins 250 cm après avoir marché 100 pas.")
else:
    print("La tortue n'a pas parcouru au moins 250 cm après avoir marché 100
    ↪ pas.")
#####
distance_totale = 0
pas_necessaires = 0
while distance_totale < 250 and pas_necessaires < 100:
    distance = random.uniform(0, 5)
    distance_totale += distance
    pas_necessaires += 1

print(f"La tortue a parcouru {distance_totale} cm en {pas_necessaires} pas.")
```

✣ - **Q3** - Le tableau ci-dessous donne la température (en degrés Celsius) enregistrée au mois de mars à Reykjavik (Islande) sur la période 1951-1960 :

Année	Temp.
1951	-3,9
1952	2,4
1953	-3,6
1954	-13,7
1955	4,5
1956	0,8
1957	5,3
1958	-1,9
1959	3,7
1960	-7,8

1- Ecrire l'algorithme permettant de calculer et afficher la température moyenne sur la période considérée.
2- Ecrire l'algorithme permettant de trouver et afficher la température la plus élevée et l'année de son enregistrement.
3- Modifier l'algorithme de la question 1 pour déterminer et afficher le nombre d'années ou la

température a été inférieure à la moyenne.
4- Ecrire les programmes correspondant aux algorithmes 2 et 3.

2 Les instructions itératives

✣ - **Q1** - Ecrire un programme en langage Python qui permet de calculer la somme suivante :

$$\sum_{i=1}^{N} i = 1 + 2 + ... + N$$

```
 n = int(input("entrez un Nombre : "))
sum = 0
for i in range(n+1):
      sum += i
print(" la somme vaut :",sum)
```

✣ - **Q2** - Ecrire un algorithme/programme en langage Python qui permet de calculer le produit suivant.

$$\Pi_{i=1}^{N} i = 1 * 2 * 3 * ...N$$

```
 n = int(input("entrez un Nombre:"))
pro = 1
for i in range(1,n+1):
     pro *= i
print(" le produit vaut :",pro)
```

✣ - **Q3** - Ecrire un programme en langage Python qui permet de calculer la valeur de $n!$ (on suppose $n >= 0$).

```
n = int(input("entrez un Nombre :"))
fac= 1
for i in range(1,n+1):
    fac *= i   #n!=1*2*3*....*(n-1)*n

print(" la somme vaut :",fac)
```

✣ - **Q4** - Ecrire un programme en langage Python qui permet de calculer la valeur de x^n (on Suppose $x! = 0$ et $n >= 0$).

```
x = int(input(" entrez le nombre x : "))
n = int(input(" entrez la puissance  n : "))
print(x**n)  #c'est facile on peut juste utilise ** pour calculer la
 ↪  puissance
res=1
# l'intérêt de l'exercice  maitrise les boucles
for i in range(1,n+1):
```

```
    res*= x
print(" resulat de ",x," a la puissance de ",n," vaut :",res)
```

✣ - **Q5** - Ecrire un programme en langage Python permettant de calculer et afficher la moyenne de valeur fournies au clavier.

```
N=int(input("entrez le nombre des valeurs : "))
somme = 0
for i in range(1,N+1):
    x=int(input("saisie une valeur : "))
    somme += x
print("moyenne : ",somme/N)
```

✣ - **Q6** - Ecrire un programme en langage Python permettant de calculer et afficher la somme d'une suite d'entiers entré au clavier. On arrêtera la saisie quand le nombre -1 est entré.

```
N=int(input("entrez une valeur : "))
somme = 0
while N != -1:
    somme += N
    N=int(input("entrez une valeur : ")
print("La Somme de la suite vaut  : ",somme)
```

✣ - **Q7** - Écrire un programme en langage Python qui affiche une matrice carrée rempli d'étoiles, s'étendant sur un nombre de lignes et un nombre de colonnes entré au clavier.

```
col=int(input("entrez le nombre des colonnes : "))
row=int(input("entrez le nombre des lignes    : "))

for i in range(row):
    for c in range(col):
        print("*",end='  ')
    print()
```

✣ - **Q8** - Ecrire un algorithme/ programme en langage Python qui affiche un triangle rempli d'étoiles, s 'étendant sur un nombre de lignes entré au clavier.

Exemple: $N = 4$

```
*
* *
* * *
* * * *
```

```
row=int(input("entrez le nombre des lignes: "))
for i in range(row):
    for c in range(i+1):
        print("*",end='  ')
    print()

# 2eme solution
row=int(input("entrez le nombre des lignes: "))
```

```
for i in range(row+1):
        print("* "*i)
```

✣ - **Q9** - Ecrire un programme en langage Python qui affiche le **pgcd** de deux entiers entrés au clavier.

$$\begin{aligned} a = 105; \quad b = 33 \quad & \texttt{if } a > b \\ a = b \cdot Q_1 + R_1 \quad \Rightarrow \quad & 105 = 33 \cdot 3 + 6 \\ b = R_1 \cdot Q_2 + R_2 \quad \Rightarrow \quad & 33 = 6 \cdot 5 + 3 \\ R_1 = R_2 \cdot Q_3 + R_3 \quad \Rightarrow \quad & 6 = 3 \cdot 2 + 0 \quad \leftarrow \texttt{arrêt} \end{aligned}$$

```
a = int(input("Entrez le premier nombre : "))
b = int(input("Entrez le deuxième nombre : "))
if b > a :
     a,b=b,a          # attention ceci est différent de a = b
                      #  b = a
while b != 0:     # la boucle s'arrêt quand b = 0
        q=a//b    # on peut néglige cette partie
        R= a%b
        print("    ",a," = ",b,"*",q," + ",R)
        a, b = b, a % b       # l'expression importante
print("Le PGCD  est", a)
```

✣ - **Q10** - Ecrire un programme en langage Python qui affiche le **ppcm** de deux entiers entrés au clavier.
La méthode la plus simple pour calculer **ppcm**:
`a * b = pgcd*ppcm`
`ppcm = (a*b)/pgcd`

```
a = int(input("Entrez le premier nombre : "))
b = int(input("Entrez le deuxième nombre : "))
if b > a :
     a,b=b,a
produit = a*b
while b != 0:
        a, b = b, a % b
print("Le PGCD  est", a)
print("Le PPCM  est", produit/a)
```

✤(La constante de PI)
Pi π peut être calculé à l'aide de la série suivante : 4(1- 1/3 + -1/5- 1/7+ 1/9- 1/l +..)
Méthode: $\pi = 4\sum_{i=0}^{N} \frac{(-1)^i}{2i+1} = 4 \cdot \frac{(-1)^1}{(2\cdot 1+1)} + \ldots$

```
n= int(input(" entrez l\' ordre de   : "))
pi = 0
```

```
for k in range( n+1):
    pi  += 4*((-1)**k / (2*k + 1))
print("Approximation de  avec", n, "itérations :", pi)
```

✤(La constante d'Euler e)
La constante d'Euler e peut s 'écrire comme :
1+1+1/2+1/6..... ...+1/n!+...
Méthode: $e = 1 + \sum_{i=1}^{N} \frac{1}{i!} = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \dots$

```
n= int(input(" entrez l\' ordre de e  : "))
e = 1
for k in range(1,n+1):
     fac = 1
     for i in range(1,k+1):
          fac *=i
     e  += (1 / fac)
print("Approximation de Euler avec", n, "itérations :", e)
```

- TD 2 -

Bekkali Hamza

09/03/2024

1 Définition des fonctions

✣ - **Q1** - `sommeDeuxVal` : qui permet de calculer la somme de deux valeurs.
✣ - **Q2** - `produitDeuxVal` : qui permet de calculer le produit de deux valeurs.
✣ - **Q3** - `sommeNVal` : qui permet de calculer la somme suivante : $\sum_{i=1}^{n} i$
✣ - **Q4** - `sommeCarre` : qui permet de calculer la somme suivante : $\sum_{i=0}^{n} i^2$
✣ - **Q5** - `difference(n)` : qui retourne la différence entre la somme des carrés et le carré de la somme.
✣ - **Q6** - `puissance(x, n)` : qui retourne la valeur de x^n (on suppose $x = 0$ et $n \geq 0$).

Quelle valeur renvoie :

- `sommeDeuxVal(2,5)` : ...
- `produitDeuxVal(2.5, 2.0)` : ...
- `sommeNVal(5)` : ...
- `sommeCarre(3)` : ...
- `puissance(2.0,3)` : ...

2 Développement limité de sinh()

- Définir la fonction d'entête :`puissance(x, n)` qui retourne la valeur de x^n (on suppose $x \neq 0$ et $n \geq 0$).
- Définir la fonction d'entête : `fact(n)` qui retourne la valeur de $n!$ (on suppose $n \geq 0$).
- Lorsque x est proche de 0, $\exp(x)$ peut être approximé à l'aide de la formule suivante : $\sum_{i=0}^{n} \frac{x^i}{i!}$.
- Définir la fonction d'entête : `exp(x, n)` qui retourne la valeur de $\exp(x)$.
- Lorsque x est proche de 0, $\sinh(x)$ peut être approximé à l'aide de la formule suivante : $\sum_{i=0}^{n} \frac{x^{2i+1}}{(2i+1)!}$.
- Définir la fonction d'entête : `sinh(x, n)` qui retourne la valeur de $\sinh(x)$.
- Définir la fonction d'entête : `combinaison(p, n)` qui retourne la valeur de $C_n^p = \frac{n!}{p!(n-p)!}$.

3 Nombres d'Armstrong

On appelle nombres d'Armstrong les nombres entiers tels que la somme des cubes de leurs chiffres (en base 10) est égale au nombre lui-même. Exemple :

$$1^3 + 5^3 + 3^3 = 153$$

.

✣ - **Q1** - Ecrire une fonction `amstrong(N)` qui retourne 1 si N est un nombre Armstrong et 0 sinon.
✣ - **Q2** - Ecrire une fonction `sommeAmstrong(N)` qui retourne la somme des nombres d'Armstrong inférieurs à un entier N.
✣ - **Q3** - Parmi tous les entiers supérieurs à 1 et inférieurs à 500, il y en a seulement 4 qui peuvent être représentés par la somme des cubes de leurs chiffres ($xyz = x + y + z$).
Écrire une fonction qui affiche ces 4 nombres?.

4 développements en séries

Écrire un programme se servant d'une fonction F pour afficher la table de valeurs de la fonction définie par :

$$f(x) = \sin(x) + \log(x) - \sqrt{x}$$

où x est un entier compris entre 1 et 10.

Méthode :

Il faut savoir avant tout que pour calculer les fonctions trigonométriques, logarithmiques (`log()`), et les racines carrées (`sqrt()`) en Python, on utilise les fonctions mathématiques intégrées de Python en important le module `math`.

Mais ici, l'intérêt de l'exercice est de savoir résoudre les problèmes et manipuler les instructions de l'algorithme. C'est pourquoi, pour calculer $\sin(x)$, $\log(x)$, et $\sqrt{(}x)$, vous pouvez également utiliser les expressions déjà apprises par le développement en série.

$$\sin(x) = \sum_{k=0}^{n} \frac{(-1)^k \cdot x^{2k+1}}{2k+1} + O(x^{2n+2})$$

5 Les nombres parfaits

Pour tout entier strictement positif x, un entier i est un diviseur strict de x si i divise x et i est différent de x.

Écrire la fonction `divStrict(x)` ?

La fonction `divStrict(x)` prend en paramètre un entier strictement positif x et affiche les diviseurs stricts de x.

Exemple : `divStrict(8)` affiche les nombres 1, 2 et 4.

illustration:

Pour $x = 8$, un entier strictement positif ($8 > 0$), on vérifie les diviseurs stricts de x :

$8/1 = 1$ avec $1 \neq 8$ et 1 est un nombre entier.

$8/2 = 4$ avec $2 \neq 8$ et 4 est un nombre entier (car 8 modulo 2 est égal à 0).

$8/4 = 2$ avec $4 \neq 8$ et 2 est un nombre entier (car 8 modulo 4 est égal à 0).

Écrire la fonction `somDivStrict(x)` ?

La fonction `somDivStrict(x)` prend en paramètre un entier strictement positif x et retourne la somme des diviseurs stricts de x.

Exemple : `somDivStrict(8)` retourne le nombre $7 = 1 + 2 + 4$.

Un entier strictement positif x est parfait, si x est égal à la somme de ses diviseurs stricts.

Écrire la fonction `parfait(x)` ?

La fonction `parfait(x)` prend en paramètre un entier strictement positif x et retourne `True` si x est parfait, sinon elle retourne `False`.

Exemple : `parfait(28)` retourne `True` ($28 = 1 + 2 + 4 + 7 + 14$).

- TD 2 Correction-

Bekkali Hamza

09/03/2024

1 Définition des fonctions

✣ - **Q1** - `sommeDeuxVal` : qui permet de calculer la somme de deux valeurs.
✣ - **Q2** - `produitDeuxVal` : qui permet de calculer le produit de deux valeurs.
✣ - **Q3** - `sommeNVal` : qui permet de calculer la somme suivante : $\sum_{i=1}^{n} i$
✣ - **Q4** - `sommeCarre` : qui permet de calculer la somme suivante : $\sum_{i=0}^{n} i^2$
✣ - **Q5** - `difference(n)` : qui retourne la différence entre la somme des carrés et le carré de la somme.
✣ - **Q6** - `puissance(x, n)` : qui retourne la valeur de x^n (on suppose $x = 0$ et $n \geq 0$).

Quelle valeur renvoie :

- `sommeDeuxVal(2,5)` : ...
- `produitDeuxVal(2.5, 2.0)` : ...
- `sommeNVal(5)` : ...
- `sommeCarre(3)` : ...
- `puissance(2.0,3)` : ...

```
def sommeDeuxVal(a,b):
      return a+b

def produitDeuxVal(a,b):
      return a*b
def sommeNVal(n):    # sum(i=1,n) = i
    somme = 0
    for i in range(1,n+1):
            somme +=i
    return somme

def sommeCarre(n):   # sum(i=1,n) = i**2
    sommeCarre = 0
    for i in range(n+1):
              sommeCarre += (i**2)   #   i*i
    return sommeCarre

def difference(n):
      somme_carres = sommeCarre(n)
      carre_somme = sommeNVal(n)* sommeNVal(n)
      return (somme_carres - carre_somme)
```

```
def puissance (x, n):      #x^n = x*x*x*x*x*..... n fois
    pow = 1
    for i in range(0,n):
         pow *= x
    return pow

print("La Somme            : ",sommeDeuxVal(2,5))
print("Le produit          : ",produitDeuxVal(2.5, 2.0))
print("La Somme de  Nvalue : ",sommeNVal(5))
print("La Somme des Carres : ",sommeCarre(3))
print("La difference       : ",difference(3))
print("La puissance        : ",puissance(2.0,3))
```

2 Développement limité de sinh()

- Définir la fonction d'entête :**puissance(x, n)** qui retourne la valeur de x^n (on suppose $x \neq 0$ et $n \geq 0$).
- Définir la fonction d'entête : **fact(n)** qui retourne la valeur de $n!$ (on suppose $n \geq 0$).
- Lorsque x est proche de 0, $\exp(x)$ peut être approximé à l'aide de la formule suivante : $\sum_{i=0}^{n} \frac{x^i}{i!}$.
- Définir la fonction d'entête : **exp(x, n)** qui retourne la valeur de $\exp(x)$.
- Lorsque x est proche de 0, $\sinh(x)$ peut être approximé à l'aide de la formule suivante : $\sum_{i=0}^{n} \frac{x^{2i+1}}{(2i+1)!}$.
- Définir la fonction d'entête : **sinh(x, n)** qui retourne la valeur de $\sinh(x)$.
- Définir la fonction d'entête : **combinaison(p, n)** qui retourne la valeur de $C_n^p = \frac{n!}{p!(n-p)!}$.

```
def puissance (x, n):      #x^n = x*x*x*x*x*..... n fois
    pow = 1
    for i in range(0,n):      #n itera
        pow *= x
    return pow

def fac(n):
    fac = 1
    for i in range(1,n+1):
            fac *= i                              #n!=1*2*3*....*(n-1)*n
    return fac

def exp(x,n):
        expo = 0
        for i in range(1,n+1):
             expo += (puissance(x,i)/fac(i))
        return expo

def sinh(x,n):
    sinsh = 0
    for i in range(n+1):
```

```
            sinsh += (puissance(x,(2*i+1))/fac(2*i+1))
    return sinsh

print("sinh(10) pour n =100 = ",sinh(10,100))

def combinaison(n,p):
    comb = (fac(n))/(fac(p)*fac(n-p))
    return comb

print("Combinaison pour n=4 et p=2 vaut :  ",combinaison(4,2))
```

3 Nombres d'Armstrong

On appelle nombres d'Armstrong les nombres entiers tels que la somme des cubes de leurs chiffres (en base 10) est égale au nombre lui-même. Exemple :

$$1^3 + 5^3 + 3^3 = 153$$

.

✣ - **Q1** - Ecrire une fonction **amstrong(N)** qui retourne 1 si N est un nombre Armstrong et 0 sinon.

✣ - **Q2** - Ecrire une fonction **sommeAmstrong(N)** qui retourne la somme des nombres d'Armstrong inférieurs à un entier N.

✣ - **Q3** - Parmi tous les entiers supérieurs à 1 et inférieurs à 500, il y en a seulement 4 qui peuvent être représentés par la somme des cubes de leurs chiffres ($xyz = x + y + z$).
Écrire une fonction qui affiche ces 4 nombres?.

```
def amstrong(N):
    to_str=str(N)
    somme=0
    for i in range(len(to_str)):
        somme += int(to_str[i])**3

    if N == somme :
        return 1
    else :
         return 0

print("153 : est un nombre d'amstrong ", amstrong(153))

def sommeAmstrong(N):
    ams=[]

    for i in range(2,N):
        if amstrong(i):
            ams.append(i)
    return ams
print("AMS ", sommeAmstrong(500))
```

4 développements en séries

Écrire un programme se servant d'une fonction F pour afficher la table de valeurs de la fonction définie par :

$$f(x) = \sin(x) + \log(x) - \sqrt{x}$$

où x est un entier compris entre 1 et 10.

Méthode :

Il faut savoir avant tout que pour calculer les fonctions trigonométriques, logarithmiques (`log()`), et les racines carrées (`sqrt()`) en Python, on utilise les fonctions mathématiques intégrées de Python en important le module `math`.

Mais ici, l'intérêt de l'exercice est de savoir résoudre les problèmes et manipuler les instructions de l'algorithme. C'est pourquoi, pour calculer $\sin(x)$, $\log(x)$, et $\sqrt{(x)}$, vous pouvez également utiliser les expressions déjà apprises par le développement en série.

$$\sin(x) = \sum_{k=0}^{n} \frac{(-1)^k \cdot x^{2k+1}}{2k+1} + O(x^{2n+2})$$

5 Les nombres parfaits

Pour tout entier strictement positif x, un entier i est un diviseur strict de x si i divise x et i est différent de x.

Écrire la fonction `divStrict(x)` ?

La fonction `divStrict(x)` prend en paramètre un entier strictement positif x et affiche les diviseurs stricts de x.

Exemple : `divStrict(8)` affiche les nombres 1, 2 et 4.

illustration:

Pour $x = 8$, un entier strictement positif $(8 > 0)$, on vérifie les diviseurs stricts de x :

$8/1 = 1$ avec $1 \neq 8$ et 1 est un nombre entier.

$8/2 = 4$ avec $2 \neq 8$ et 4 est un nombre entier (car 8 modulo 2 est égal à 0).

$8/4 = 2$ avec $4 \neq 8$ et 2 est un nombre entier (car 8 modulo 4 est égal à 0).

Écrire la fonction `somDivStrict(x)` ?

La fonction `somDivStrict(x)` prend en paramètre un entier strictement positif x et retourne la somme des diviseurs stricts de x.

Exemple : `somDivStrict(8)` retourne le nombre $7 = 1 + 2 + 4$.

Un entier strictement positif x est parfait, si x est égal à la somme de ses diviseurs stricts.

Écrire la fonction `parfait(x)` ?

La fonction `parfait(x)` prend en paramètre un entier strictement positif x et retourne `True` si x est parfait, sinon elle retourne `False`.

Exemple : `parfait(28)` retourne `True` $(28 = 1 + 2 + 4 + 7 + 14)$.

```
#1
def  divStrict(x):
        for i in range(1,x ):
```

```
                if x%i == 0 :
                        print(i , " diviseur strict de  ",x)

divStrict(8)

#2
def  somDivStrict(x):
        somme = 0
        for i in range(1,x ):
                if x%i == 0 :
                        somme += i
                        #print(i , " diviseur strict de  ",x)
        return somme
x =8
res= somDivStrict(x)
print(" La somme des diviseurs stricts de  ",x," est : ",res)
#3
def  parfait (x):
        somme = somDivStrict(x)
        if x != somme :
                return False
        else :
                return True
x =8
res= parfait(x)
print( x,"est un nombre parfait?  ",res)
#4
def  compte_parfaits (L):
      c = 0
      for i in L:
            etat = parfait(i)
            if etat:
                  c +=1
      return c
L = [25,28,123,10,6]
res= compte_parfaits(L)
print(res," nombres sont parfait")

#5
N = 500
def  liste_parfaits (n):
     P=[]
     for i in range(1,n ):
          etat = parfait(i)
          if etat :
                P.append(i)
     return P
liste=liste_parfaits(N)
print(" Liste des nombres parfait <",N," sont :",liste)
```

- Devoir -

Bekkali Hamza

09/03/2024

1 Exercice 1

Évaluez les listes suivantes et entourez la seule bonne réponse :
✣ - **Q1** - $[1+2, [3,3]]$ est évaluée (égale) à :

[3 , [3 , 3] **[[3] , [3, 3]]** **[3 , 3 , 3]** **[3 , [3 , 3]]**

✣ - **Q2** - $[[1,2], \text{print}(3), 2 > 1]$ est évaluée (égale) à :

[1 , 2 , 3 , True] [[1 , 2] , 3 , True] [[1 , 2],None,'2 > 1'] [[1 , 2],None,True]

2 Jeu de Juniper Green

Le jeu de Juniper Green, également connu sous le nom de "jeu des multiples et des diviseurs", est un jeu mathématique opposant deux joueurs. Il a été créé par Richard Porteous, un enseignant à l'école de Juniper Green, et tire son nom de cet établissement. Ce jeu a suscité un vif intérêt et a été l'objet de nombreux défis et discussions, notamment sur la manière de remporter la partie.

L'objectif de cet exercice est d'appliquer les connaissances acquises en cours de Python (listes, fonctions, boucles) pour implémenter ce jeu.
Les règles du jeu de Juniper Green sont plutôt simples :
➙ **Règle 1** : Le jeu se joue avec les nombres entiers de 1 à N, où N est un entier naturel non nul. À l'origine du jeu, il semblerait que N valait 20.
➙ **Règle 2** : Le premier joueur choisit un nombre pair entre 1 et N.
➙ **Règle 3** : Une fois qu'un nombre a été choisi, il ne pourra plus jamais être utilisé.
➙ **Règle 4** : Chaque joueur doit choisir un nombre qui est soit un diviseur soit un multiple du nombre précédent.
➙ **Règle 5** : Le joueur qui ne peut plus jouer perd la partie.

Démonstration de jeu :
Exemple de jeu entre la machine et l'utilisateur : Prenons une grille de taille 20. Au premier temps, le tableau des coups possibles est le suivant :

1	**2**	**3**	**4**	**5**
6	**7**	**8**	**9**	**10**
11	**12**	**13**	**14**	**15**
16	**17**	**18**	**19**	**20**

-> carte=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

✤Selon les règles 1 et 2, la taille de la grille est de 20. Le premier joueur, par exemple la machine (`computer`), a la possibilité de choisir un nombre de 1 à 20 qui doit être **pair**. Par exemple, l'ordinateur choisit (`computer = 8`) . Ce nombre '8' doit être éliminé de la grille selon la règle 3. Le tableau devient alors : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20.

✤Selon la 4ème règle, le deuxième joueur "`user`" doit choisir un nombre qui est soit un diviseur, soit un multiple parmi la liste des coups possibles de ce nombre '8', le nombre choisi doit être éliminé de la grille.

1	**2**	**3**	**4**	**5**
6	**7**	X	**9**	**10**
11	**12**	**13**	**14**	**15**
16	**17**	**18**	**19**	**20**

Le jeu continue ainsi jusqu'à ce que l'un des deux joueurs n'ait aucun nombre disponible dans les coups possibles. Si c'est le cas de l'utilsateur, le programme affiche "*Vous avez perdu !*" Sinon, si c'est l'ordinateur qui échoue, le programme affiche "*Bravo, vous avez battu l'ordinateur !*". Le tableau suivant illustre toutes les possibilités pour chaque coup précédent des coups possibles.

Coupe précédente	Nb. Possibilités	Liste des coupes Possibles
1	**19**	[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
2	**10**	[1, 4, 6, 8, 10, 12, 14, 16, 18, 20]
3	**6**	[1, 6, 9, 12, 15, 18]
4	**6**	[1, 2, 8, 12, 16, 20]
5	**4**	[1, 10, 15, 20]
6	**5**	[1, 2, 3, 12, 18]
7	**2**	[1, 14]
8	**4**	[1, 2, 4, 16]
9	**3**	[1, 3, 18]
10	**4**	[1, 2, 5, 20]
11	**1**	[1]
12	**5**	[1, 2, 3, 4, 6]
13	**1**	[1]
14	**3**	[1, 2, 7]
15	**3**	[1, 3, 5]
16	**4**	[1, 2, 4, 8]
17	**1**	[1]
18	**5**	[1, 2, 3, 6, 9]
19	**1**	[1]
20	**5**	[1, 2, 4, 5, 10]

✣ - **Q1** - Déclarez une liste nommée `Carte` et initialisez-la avec deux méthodes pour générer des séquences de nombres de 1 à 20.

✣ - **Q2** - Définissez une fonction nommée `diviseurs_multiples(n:int)` qui prend comme argument un entier et renvoie une liste des valeurs multiples et diviseurs de ce nombre n.
e.g : diviseurs_multiples(8) ; [1, 2, 4, 16]

✣ - **Q3** - Pour générer des nombres aléatoires, nous pouvons utiliser le module random.

```
from random import *
```

* L'étoile indique que l'on importe toutes les fonctions contenues dans le module. On peut aussi énumérer les fonctions nécessaires, par exemple s'il ne nous en faut qu'une :

```
from random import randint
```

La fonction `randint(a,b)` retourne un nombre entier entre a et b (bornes comprises). Exemple:

```
 from random import randint
print(randint(1,6))
# ----
# 3
```

Soit l'expression "`2*randint(1,20/2)`" signifie que la variable "`computer`" est affectée à deux fois un nombre aléatoire compris entre 1 et 20 divisé par 2. Cela garantit que "`computer`" sera toujours un nombre pair, car le produit de tout nombre entier par 2 est pair.

- Déclarer une variable nommée `computer` et l'initialiser avec une valeur aléatoire paire comprise entre 1 et 20

✣ - **Q4** - **- Écrire une expression qui nous aide à supprimer la valeur de "`computer`" de la liste "`Carte`.**

✣ - **Q5** - créez une liste appelée "`carteDisponible`" et affectez le résultat de la fonction "`diviseurs_multip`
à cette liste.
Ensuite, affichez la liste "`carteDisponible`".

✣ - **Q6** - - Affichez le message "Jouons avec des nombres entre 1 et 20 " ; 20 comme résultat. - Affichez le message "Je choisis comme nombre de départ", suivi de la valeur de "computer". - Créez une boucle qui teste à chaque fois si la liste "carteDisponible" est vide ou non. - Ecrire une boucle qui demande à l'utilisateur de corriger la valeur saisie s'il entre une valeur incorrecte ?

✣ - **Q7** -

✣ - **Q1** - Créez une liste nommée `Carte` et initialisez-la avec deux méthodes pour générer des séquences de nombres de 1 à 20.

✣ - **Q2** - Définissez une fonction nommée `diviseurs_multiples(n:int)` qui prend un entier en argument et renvoie une liste des valeurs multiples et diviseurs de ce nombre n. Par exemple, pour diviseurs_multiples(8), la liste renvoyée serait [1, 2, 4, 16].

✣ - **Q3** - Pour générer des nombres aléatoires, utilisez le module random.

```
from random import *
```

L'étoile indique que toutes les fonctions du module sont importées. Vous pouvez également importer des fonctions spécifiques, par exemple :

```
from random import randint
```

La fonction `randint(a,b)` retourne un nombre entier entre a et b (bornes comprises). Par exemple :

```python
from random import randint
print(randint(1,6))
# ----
# 3
```

L'expression "`2*randint(1,20/2)`" signifie que la variable "`computer`" est affectée à deux fois un nombre aléatoire compris entre 1 et 20 divisé par 2. Cela garantit que "`computer`" sera toujours un nombre pair, car le produit de tout nombre entier par 2 est pair.

Déclarez une variable nommée `computer` et initialisez-la avec une valeur aléatoire paire comprise entre 1 et 20.

✣ - **Q4** - Écrivez une expression pour supprimer la valeur de "`computer`" de la liste "`Carte`".

✣ - **Q5** - Créez une liste appelée "`carteDisponible`" et lui assigner les diviseurs et multiples de le résultat de la fonction "`diviseurs_multiples(computer)`". Ensuite, affichez la liste.

✣ - **Q6** - Créer une boucle qui continue tant que la liste "carteDisponible" n'est pas vide.

✣ - **Q7** - proposer une instruction qui demander à l'utilisateur de saisir une valeur valide présente dans la liste "carteDisponible" et la supprimer de la liste "Carte" ?

✣ - **Q8** - Continuez l'écriture du programme afin que le résultat soit comme suit :

Jouons avec des nombres entre 1 et 20
Je choisis comme nombre de départ 20
Nombres valides : [1, 2, 4, 5, 10]
Que jouez-vous ? 2
Nombres valides : [1, 4, 6, 8, 10, 12, 14, 16, 18]
Je joue 1
Nombres valides : [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Que jouez-vous ? 14
Nombres valides : [7]
Je joue 7
Vous avez perdu!
