TD2 : AP.

```python
def est_pair(n: int) -> bool:
    return n == 0 or not est_impair(n)

def est_impair(n: int) -> bool:
    return not (n == 0 or est_pair(n))
```

1. Que pensez-vous des expressions calculées par chacune des deux fonctions dans les deux cas de base et récursif ?

boucle infinie.

1.2 Juste Pair

Voici la déclaration d'une fonction `est_pair`

```python
def est_pair(n: int) -> bool:
    if n == 0:
        res = True
    else:
        res = est_pair(n - 2)
    return res
```

1. Que pensez-vous de cette fonction ?

Correct lorsque n est pair, boucle infinie sinon.

1. Proposez un algorithme récursif de calcul de la somme de deux entiers naturels a et b en supposant que les seules opérations de base dont vous disposez sont
   - l'ajout de 1 à un entier x : $x + 1$
   - le retrait de 1 à un entier x : $x - 1$
   - et les comparaisons à 0 d'un entier x : $x = 0$, $x > 0$ et $x < 0$.
2. L'algorithme proposé est-il récursif terminal ?
3. Donnez un variant de l'algorithme proposé.
4. Programmez cet algorithme en python pour en faire une fonction `add` à deux paramètres.
5. Tracez les appels récursifs de `add(2, 3)`
6. Comment étendre cette fonction aux entiers de signe quelconque ?

1. add : a, b. $a, b \geq 0$.

   si b = 0 :

   return a

   sinon

   Return add a+1, b-1

2. Oui, car la valeur finale de Renvoi est le résultat de l'appel de la fonct°.
3. b est un variant de add

4.
```
def add(a:int, b:int)-> int:
    if b == 0:
        return a
    else:
        return add(a+1, b-1)
```

5. add(2,3) → 5
   ↳ add(3,2) → 5
   ↳ add(4,1) → 5
   ↳ add(5,0) → 5

6.
```
def add(a:int, b:int)-> int:
    if b == 0:
        return a
    elif b < 0:
        return add(a-1, b+1)
    else:
        return add(a+1, b-1)
```

3 Produit de deux entiers

1. Proposez un algorithme récursif de calcul du produit de deux entiers naturels a et b en supposant que les seules opérations de base dont vous disposez sont

- la somme de deux entiers x et y : $x + y$
- le retrait de 1 à un entier x : $x - 1$
- et la comparaison à 0 d'un entier x : $x = 0$.

2. L'algorithme proposé est-il récursif terminal ?
3. Donnez un variant possible de votre algorithme.
4. Programmez cet algorithme en python pour en faire une fonction `mult` à deux paramètres.
5. Tracez les appels récursifs de `mult(0, 3)` et de `mult(3, 2)`.

fonction récursive produit a, b.
  si b == 0 alors
    retourne 0
  sinon
    retourne a + produit(a, b-1).

2. non, cet algorithme n'est pas récursif terminal.
3. b est un variant.

```
def mult(a:int, b:int):
    if b == 0:
        return 0
    elif b == 1:
        return a
    else:
        return a+add(a, b-1)
```

5. mult(0,3) → 0 + mult(0,2) → 0 + mult(0,1) → 0 + 0 → 0

mult(3,2) → 3 + mult(3,1) → 3 + 3 → 6

3.1 Puissance entière d'un nombre réel

1. Proposez un algorithme récursif de calcul de la puissance n-ième ($n \in \mathbb{N}$) d'un nombre réel a en supposant que les seules opérations de base dont vous disposez sont

- le produit de deux réels x et y : $x \times y$
- le retrait de 1 à un entier x : $x - 1$
- et la comparaison à 0 d'un entier x : $x = 0$

2. Exprimer le nombre de produit $c(n)$ effectués par votre algorithme en fonction de n.
3. (MI) En déduire un encadrement de $c(n)$ en fonction de la taille binaire t de n (*rappel* : $t(n) = \lfloor \log_2(n) \rfloor + 1$)
4. Nous allons améliorer la complexité de l'algorithme. En utilisant uniquement
   - une élévation au carré et
   - au plus une multiplication,

   exprimez a^n en fonction de a^k et a lorsque $n = 2k$, puis lorsque $n = 2k + 1$.
5. En déduire un autre algorithme récursif pour calculer a^n.

   Donnez, en fonction de n, un encadrement du nombre de multiplications effectuées par cet algorithme. Comparez avec l'algorithme précédent.

```
fonction Récursive exp a, n :
    si n-1 == 0 alors
        Retourner a
    sinon :
        retourne a * exp(a, n-1)
```

$c(n) = n$

3 $\lfloor \log_2(n) \rfloor + 1 \leq \log_2(n) + 1 \leq \lfloor \log_2(n+1) \rfloor + 1$

$\Rightarrow 2^t \leq 2n \leq 2^{t+1}$

$\Rightarrow 2^{t-1} \leq n \leq 2^t$.

4. $a^n = \begin{cases} (a^k)^2 & \text{si } n = 2k \\ (a^k)^2 * a & \text{si } n = 2k+1 \end{cases}$

```
5. def exp2(a, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return exp2(a, n//2) ** 2
    else:
        return a * exp2(a, n//2) ** 2.
```

4 Nombre de chiffres d'un entier

Réalisez une fonction récursive qui calcule le nombre de chiffres de l'écriture décimale d'un nombre entier passé en paramètre *sans utiliser de chaîne de caractères.*

```
nbre_chiffres(0)
1
nbre_chiffres(2020)
4
```

```
def nbre_chiffre(n:int) -> int:
    if n < 10:
        return 1
    else:
        return 1 + nbre_chiffre(n//10)
```