Travaux dirigés

Exercice 1 - Lecture de code

Vous en rencontrerez de deux sortes :

1. Du code python extrait d'un code source :

```
a = 3
b = 5
c = a * b
```

2. Du code python extrait de l'interpréteur :

```
>>> a = 3
>>> b = 5
>>> c = a * b
>>> c
15
>>> 2 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Les chevrons `>>>` indiquent une commande tapée et exécutée dans l'interpréteur. Une ligne qui n'en contient pas indique une réponse de l'interpréteur.

La dernière génère une erreur (on dit aussi qu'elle lève une exception). Les erreurs Python :

- commencent toujours par `Traceback ...`
- indiquent le fichier et le numéro de ligne où l'erreur est survenue (`<stdin>` pour l'interpréteur),
- se terminent par un type d'exception (ici `ZeroDivisionError`)

Questions

1. On a effacé les lignes de sortie de l'interpréteur. Écrire la sortie attendue.

```
>>> x = 8
>>> x + 2
>>> x ** 2
>>> x * x
>>> x = x + 1
>>> x
```

Exercice 2 - Opérations

On donne : `A=10, B=5, C=2, D=4`

1. Évaluer les expressions *valides*, rectifier celles qui sont fausses :

```
3 * A + 5 * B
3 A - 2 B
A / D
A // D
A += 2
A = 10
A == 12
B == A / 2
B == A // 2
C ** B
C ** -B
```

Les types de base : `int`, `float`, `bool`

Chaque objet python a un **type**. Les types de base que nous rencontrerons souvent sont : `int`, `float`, `bool`.

On accède au type d'un objet avec la fonction `type` :

```
>>> type(2)
<class 'int'>
>>> type(2.0)
<class 'float'>
>>> type(2 == 2.0)
<class 'bool'>
>>> type(2) == int
True
```

La fonction `type` est à réserver aux situations extrèmes où on n'a aucune idée du type d'une variable.

On préfère `isinstance` (est une instance de...) qui permet de vérifier un type précis :

`isinstance(obj, type) -> bool`

```
>>> isinstance(2, int)
True
>>> isinstance(2 == 3, bool)
True
```

Exercice 3 - Opérations sur les entiers et les flottants

- `int` : *integer*, les entiers, sans valeur maximale. Exemple : `-223`, `2**123`

Il existe 6 opérations de base sur les entiers en python :

operateur	signification
`+`	addition
`-`	soustration
`*`	produit
`//`	division entière
`%`	reste de la division
`**`	exposant

Ainsi :

- `7 // 2` vaut `3` et `7/2` vaut `3.5`.
- `7 % 2` vaut 1 (car $7 = 3 \times 2 + 1$).

1. Proposer une expression donnant le nombre de secondes écoulées entre le premier janvier de cette année à minuit et ce matin, à minuit.
2. Pierre, Paul et Jacques ont acheté par erreur `2 ** 11` gâteaux à la boulangerie. Ils les répartissent équitablement. Combien chacun en aura-t-il ? Combien en reste-t-il ?
3. Opération et affectation

   Rappel :

   ```
   >>> a = 3
   >>> a
   3
   >>> a += 1 # augmente a de 1, ne renvoie rien
   >>> a
   4
   ```

   On peut utiliser cette notation pour toutes les opérations : `-= *= /= //=` etc.

   Question : Evaluer les variables `a, b, c`

   ```
   >>> a = -5
   >>> a *= 2
   >>> a

   >>> b = 11
   >>> b //= 2
   >>> b

   >>> c = 3
   >>> c -= 1
   >>> c
   ```

- `float`: *float*, les nombres à virgule flottante. Exemple : `1.234`, `2.3e4`

  Grosso modo les réels. Retenez pour l'instant qu'il n'y aucun moyen de tester une égalité parfaite avec des flottants.

  ```
  >>> 0.1 + 0.2 == 0.3
  False
  ```

  Opérations : les mêmes que pour les entiers, la division réelle `/` en plus.

  Attention : lors d'une opération entre un entier et un flottant on obtient toujours un flottant.

  ```
  >>> 3 + 1.0
  4.0
  ```

  Question : Évaluer le type de sortie des opérations suivantes :

  ```
  2 + 5
  2.0 + 5
  3 // 2
  3 / 2
  4 ** 0.5     # a ** 0.5 = racine carrée de a
  ```

Exercice 4 - Opérations sur les booléens

Il existe trois opérateurs sur les booléens :

- `not` qui renvoie le contraire :

```
>>> not True
False
>>> not False
True
```

- `and` : le "et" logique : `b1 and b2` est vrai si, et seulement si `b1` est vrai ET `b2` est vrai.
- `or` : le "ou" logique : `b1 or b2` est faux si, et seulement si `b1` est faux ET `b2` est faux.

1. Evaluer les expressions booléennes suivantes :
   - `not (1 == 2)`
   - `(1 == 2) or (2 ** 2 == 4)`
   - `(4 <= 3) or (5 > 2)`
2. `x` est un variable de type `float`. Proposer une expression booléenne permettant de vérifier que :
   1. $x \in [1;9]$
   2. $x \in]-\infty;0[\cup]2;3]$
3. On veut savoir si l'entier n est divisible par trois sans qu'il ne vaille 0. Proposer une expression booléenne.
4. Une année est bissextile si elle est divisible par 4 mais non divisible par 100. Les années divisibles par 400 sont également bissextiles. Ecrire une condition portant sur une `annee` pour savoir si elle est bissextile.

Exercice 5 - Affectation

On rappelle le principe de l'affectation

```
x = 2
```

Une fois cette instruction réalisée, `x` est un identifieur qui pointe vers une case mémoire. Celle-ci contient l'entier 2.

Attention : Ne pas confondre l'affectation (=) et la comparaison "égalité" (==)

- Une *comparaison* retourne toujours un *booléen* `True` ou `False`.
- Une *affectation* ne retourne rien.

Affectations multiples :

```
a, b = 2, 3
ma_liste = [1, 2, 3]
x, y, z = ma_liste
```

1. À l'issue de ces affectations que contiennent les variables `a, b, x, y` et `z` ?
2. En deux lignes : affecter à `ma_liste` la liste des entiers pairs entre 2 et 12 inclus et affecter ces entiers aux lettres de l'alphabet.
3. Selon-vous que donnerait la série d'instruction suivante ?

   ```
   ma_liste = [1, 2, 3, 4]
   x, y, z = ma_liste
   ```

4. **Affectations impossibles**. Parmi les affectations suivantes, lesquelles vont générer une erreur ? Lorsque l'affectation est possible, quel est le type de la variable ?

```
a = "22"
b = 22
c = a + b
"d" = 22
"d" = "22"
ma_liste = [1, "2", trois]
ma_liste = [1, "2", trois]
ma_liste = ["1", 2, "trois"]
ma_liste[2] = 9
ma_liste[3] = 5
e = ("b" == 22)
f = True
```

Exercice 6 - Échanger deux valeurs

On part de :

```
a = 3
b = 6
```

Comment arriver à ce que la valeur de `a` soit 6 et celle de `b` 3 ?

Bien sûr, on s'interdit de faire :

```
a = 6
b = 3
```

Blocs d'instructions

En Python un **bloc d'instruction :**

1. Commence par une ligne qui se termine par le symbole `:`
2. Est contenu dans un niveau d'*indentation* (2 ou 4 espaces)
3. Se termine quand le niveau d'indentation décroît.

Fonctions - `def fonction(x, y, z):`

Une **fonction** :

- se définit avec `def func(...):` suivi d'un bloc indenté.
- s'appelle avec `func( ... )` après sa définition.

```
def f(x):
    return 2 * x + 3

f(4) == 11
```

Cette fonction renvoie (`return`) $2x+3$ pour tout x

Une fonction sans `return` renvoie toujours `None` (*rien*).

Exercice 7 - évaluer

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

Évaluer :

```
>>> func(12, 10)
>>> func("bonjour", "la famille")
>>> func(12, "bonjour")
>>> func(10, func(5, 7))
```

Exercice 8 - définir

1. Définir une fonction qui prend deux chaînes de caractères et renvoie les initiales :

   ```
   >>> initiales("Georges", "Garmin")
   "GG"
   ```

2. Définir une fonction qui prend deux chaînes et renvoie un booléen vrai si et seulement si la première chaîne est une sous-chaîne de la seconde

   ```
   >>> contient("def", "abcdefg")
   True
   >>> contient("zz", "zinedine zidane")
   False
   ```

   Améliorer la fonction pour qu'elle renvoie vrai ssi la première est dans la seconde ou la seconde est dans la première.

   ```
   >>> contient2("def", "abcdefg")
   True
   >>> contient2("abdedfg", "def")
   True
   ```

3. Programmer `est_pair` qui prend un entier et renvoie vrai s'il est pair.
4. `divmod` est une fonction *native* qui prend deux entiers `a` et `b` et renvoie le quotient de `a` par `b` ainsi que le reste de cette division. Programmer cette fonction.

Effets de bord

Une fonction est à *effet de bord* si elle modifie un état en dehors de son environnement local, c'est-à-dire a une interaction observable avec le monde extérieur autre que retourner une valeur.

`print` est une fonction native qui prend autant de paramètres qu'on veut et ne renvoie rien.

Son effet de bord est d'afficher les valeurs dans la console.

```
>>> a = print("super")
super
>>> a
None
```

Remarquez que lorsqu'on `print` une chaîne, celle-ci n'est pas entourée de guillemets.

Exercice 9 - Effets de bord

On considère :

```
def func(a):
    print(a + 5)
```

Compléter les sorties manquantes :

```
>>> func(5)
>>> b = 3
>>> b = func(8)
>>> b
```

Remplacer la deuxième ligne de la fonction `print(a + 5)` par `return a + 5` et recommencer.

***print* ne sert à rien, sauf pour afficher quelque chose**

Conditions - `if elif else`

Une **instruction conditionnelle** :

commence par `if condition:`
se poursuit par une série d'instructions précédées d'une indentation
se poursuit éventuellement par d'autres instructions condtionnelles (`elif condition:`)
se termine éventuellement par sinon... (`else:`)

Ne pas oublier le :

```
age = 23
if age > 18:
    majeur = True
else:
    majeur = False
print(majeur)
```

Le programme ci-dessus comporte une instruction conditionnelle : `if ... else`

- L'instruction `majeur = True` est *indentée* par 4 espaces. Elle n'est exécutée que si la condition `age > 18` est vraie.
- L'instruction `majeur = False` est après `else:`. Elle n'est exécutée que si `age > 18` est faux.
- L'instruction `print(majeur)` n'est pas indentée. Elle est *toujours* exécutée.
- Si d'autres conditions doivent être réalisées, on peut ajouter des instructions `elif condition:`

Exercice 10 - `if elif else`

1. Que verra-t-on à l'écran après avoir exécuté les lignes précédentes ?
2. Écrire un programme Python qui affecte à `nb_pommes` un entier. Ensuite, si le nombre de pommes est pair, vous affichez "divisible par 2". Si ce n'est pas le cas, vous affichez "non divisible par 2".
3. Écrire un programme Python qui affiche la mention obtenue au bac.

   ```
   moyenne = ... # obtenue plus tôt
   if ...                          :
   ```

   On *affiche* du texte avec `print("bonjour")`

   Attention, `"bonjour"` n'est pas une *variable* mais une chaîne de caractères.
4. Rectifier les erreurs d'indentation dans les instructions suivantes :

```
if 1 + 1 == 2:
a = True
else:
    c = 2
        d = 4
            e = 6
    f = 8
```

5. Proposez deux programmes différents qui répondent au problème suivant : Martin peut inviter ses copains s'il a fini ses devoirs et rangé sa chambre.

   On utilisera les variables booléennes `devoir_faits` et `chambre_rangee`

6. Écrire une fonction qui prend une note (nombre entre 0 et 20) et renvoie la mention associée.

Exercice 11 - Boucles non bornées : `while`

Python propose deux boucles : `while` et `for`.

La syntaxe d'un `while` est simple :

```
while condition:
    instruction
```

Tant que `condition` est vraie, on exécute `instruction`

Somme des entiers de 1 à 10 :

```
somme = 0
entier = 1
while entier <= 10:
    somme += entier
    entier += 1
```

Afficher son nom toutes les secondes :

```
from time import sleep
while True:
    print("Robert")
    sleep(1)
```

1. Robert ajoute 50 € à sa cagnotte tous les mois jusqu'à atteindre 1200 €.
   Écrire une boucle `while` Python qui calcule le nombre de mois nécessaires pour qu'il obtienne assez d'argent.
2. Le haricot magique de Jack double de hauteur tous les jours. Il mesure 1 cm le premier jour.
   Écrire une boucle while qui calcule le nombre de jours nécessaires pour qu'il atteigne 1 km de hauteur.
3. Le programme suivant est supposé afficher un point `.` toutes les secondes et s'arrêter après 10 secondes. Malheureusement il entre dans une boucle infinie. Rectifiez le.

   ```
   from time import sleep
   nb_points = 0
   while nb_points < 10:
       print('.')
       sleep(1)        # attend une seconde
   ```

4. On a affecté à `f` la fonction mathématique $f(x) = x^2 + 5x - 2$. Programmer une fonction `seuil` qui prend un nombre `a` et renvoie le premier entier `n` tel que $f(n) > a$.
5. Programmer une fonction `nb_annee_pour_doubler` qui prend un taux d'intérêt `t` en pourcentage (positif) et renvoie le nombre d'années nécessaires pour doubler un capital placé à intérêts composés avec le taux `t`.

Exercice 12 - Boucles bornées : `for`

Une boucle `for` itère sur une *collection* la syntaxe est :

```
for element in collection:
    instruction
```

La variable `element` prend pour valeurs successives chaque objet de `collection`

Exemple : produit des entiers de 3 à 9 :

```
produit = 1
for entier in [3, 4, 5, 6, 7, 8, 9]:
    produit = entier * produit
```

Le même résultat sans devoir décrire toute la liste des entiers :

```
produit = 1
for entier in range(3, 10):
    produit = entier * produit
```

Attention à `range` qui prend 1, 2 ou 3 paramètres :

- `range(10)` : 0, 1, 2, 3, . . . , **9**
- `range(4, 12)` : **4**, 5, 6, . . . , **11**
- `range(1, 10, 2)` : 1, **3**, **5**, 7, 9. Le dernier paramètre est le pas

1. Écrire une boucle `for` qui calcule la somme des entiers pairs plus petits que 100
2. Écrire une boucle `for` qui calcule le produit des entiers dont le reste est 2 dans la division par 3 et qui sont inférieurs à 200.
3. Le programme suivant affiche la table de multiplication par 5.

   ```
   >>> for x in range(3):
   ...     print('5 *', x, '=', 5 * x)
   ...
   5 * 0 = 0
   5 * 1 = 5
   5 * 2 = 10
   ```

   1. Modifier le pour qu'il affiche la table complète (de `5 * 0` à `5 * 10`).
   2. Modifier le pour qu'on puisse changer le nombre dont on veut la table.
   3. Écrire un nouveau programme qui affiche TOUTES les tables.
      Penser à ajouter une ligne de séparation entre les tables (`print()`)

On peut *itérer* (= faire une boucle qui parcourt quelque chose) dans n'importe quelle collection en Python. Par exemple pour afficher une lettre par ligne :

```
alphabet = "ABCDEFGHIJKLMNOPQRSTUVXYZ"
for lettre in alphabet:
    print(lettre)
```

4. Affecter à `mes_enfants` les prénoms de vos 6 futurs enfants (j'insiste). Écrire un prénom par ligne.

Lorsqu'on a besoin de connaître *l'indice* d'un élément, il existe deux approches :

```
mes_animaux = ["Lulu", "Lili", "Minouche"]
for i in range(len(mes_animaux)):
    print("Mon animal numero", i + 1, "est", mes_animaux[i])
```

Qui va afficher :

Mon animal numero 1 est Lulu
Mon animal numero 2 est Lili
Mon animal numero 3 est Minouch

On a utilisé la fonction `len(collection) -> int` qui retourne la longueur d'une collection.

Autre approche :

```
mes_animaux = ["Lulu", "Lili", "Minouche"]
for numero, animal in enumerate(mes_animaux):
    print("Mon animal numero", numero + 1, "est", animal)
```

On a utilisé `enumerate` qui parcourt une collection et renvoie à chaque étape un couple avec l'indice et l'élément.

5. Écrire de deux manière la comptine :

   Ma lettre numéro 1 est A
   Ma lettre numéro 2 est B
   Ma lettre numéro 3 est C
   Ma lettre numéro 4 est D
   Ma lettre numéro 5 est E
   ...

6. Écrire une fonction `nb_apparition` qui prend deux chaînes `lettre` et `mot` et renvoie le nombre de fois que `lettre` apparaît dans `mot`.

7. On simule un lancer de de avec `randint` :

   ```
   >>> from random import randint
   >>> randint(1, 6) # entier entre 1 et 6 inclus
   5
   ```

   Écrire une fonction qui prend deux paramètres entiers `n` et `f`, simule `n` lancers de dés et renvoie le nombre d'apparitions du `f`

8. Écrire une fonction `factorielle` qui calcule le produit des `n` premiers entiers non nuls : $1 \times 2 \times \cdots \times n$

   Rectifier la fonction pour respecter la définition mathématique : factorielle de 0 = 1.

9. Écrire une fonction `explose` qui prend deux paramètres `x` et `n` et renvoie :

   $$x^{x^{x^{x^{\cdot^{\cdot^{\cdot^{x}}}}}}}$$

   Avec n exposants.

Exercice 13 - Un premier types complexe : `list`

Un objet de type `list` Python est une collection d'objets, regroupés dans des `[]`.

```
ma_liste = [1, 2, 3]
ma_liste_vide = []
mes_enfants = ['Rambo 1', 'Rambo 2', 'Rambo 3']
```

On accède à un élément avec son indice : `ma_liste[indice]`. Attention : Python indexe les listes à partir de 0.

Comment s'appelle mon *second* fils, déjà ?

```
>>> mes_enfants[1]
'Rambo 2'
```

Ces objets sont *mutables* == modifiables.

```
>>> mes_enfants.append('Rambo 4') # on ajoute à la fin
>>> mes_enfants
['Rambo 1', 'Rambo 2', 'Rambo 3', 'Rambo 4']
```

On efface un élément avec `del ma_liste[indice]`

```
>>> del mes_enfants[0]     # il ne savait pas tirer à l'arc...
>>> mes_enfants
['Rambo 2', 'Rambo 3', 'Rambo 4']
```

On peut mesurer la longueur d'une liste avec `len(ma_liste)`

```
>>> len(["a", "b", "c"])
3
```

1. On exécute le programme suivant :

   ```
   mes_carres = []
   for i in range(100):
       if i % 2 == 0:
           mes_carres.append(i ** 2)
   ```

   1. Quels sont les premiers et derniers éléments de `mes_carres` ?
   2. Comment accéder à la longueur de la liste `mes_carres` ?
   3. Modifier le code pour déterminer les carrés des entiers divisibles par 3.
   4. Créer la liste de l'énoncé précédent sans utiliser `if`

2. Robert commence un régime. Le mardi et le vendredi, il ne se nourrit plus que de fruits.
   Écrire un programme qui affiche chaque jour de la semaine et le type d'alimentation de Robert :

   Le lundi tu peux manger ce que tu veux,
   Le mardi tu dois manger des fruits,

   On utilisera une liste pour enregistrer les jours `["lundi",...]`

Exercice 14 - Liste et boucle for

Dans cet exercice on répondra d'abord simplement à la question avant de proposer une fonction qui le fasse. On prendra garde aux types des paramètres en entrée et en sortie.

On itère sur une liste avec la syntaxe `for element in ma_liste:`

1. Nous avons prévu d'avoir encore 13 enfants. Compléter la liste de mes enfants à l'aide d'une boucle for.
2. On considère une liste d'entiers `entiers = [1, 2, 3, 4, ... , 1000]`. A l'aide d'une boucle `for` créer la liste des carrés des entiers : `[1, 4, 9, ...]`

Il arrive qu'on ait besoin de modifier un élément d'une liste.

Par exemple : remplacer tous les éléments d'indice *pair* par 0 :

```
for indice in range(len(ma_liste)):
    if indice % 2 == 0: # si l'indice est pair
        ma_liste[indice] = 0 # l'élément est maintenant 0
```

3. Le cinéma n'est plus mon art préféré depuis que j'ai découvert la chaîne youtube de Squeezie (j'ai dû ouvrir google pour taper son pseudo...)

   Modifier les noms de mes enfants pour qu'ils s'appellent `Squeezie 1`, `Squeezie 2` etc.

On peut tester si in élément est dans une liste avec `in`

```
>>> 1 in [1, 2, 3]
True
>>> 4 in [1, 2, 3]
False
```

3. On considère deux variables : `lettres = ['a', 'b', ..., 'z']` et `voyelles = ['a', 'e', 'i', 'o', 'u', 'y']`

   Écrire un programme python qui affiche chaque lettre de l'alphabet avec un commentaire à la manière de :

   ```
   a est une voyelle
   b n'est pas une voyelle
   c n'est pas une voyelle
   ...
   ```

4. Les albums des Beatles sortis au royaume uni sont :

Année	Nom
1963	Please Please Me
1963	With the Beatles
1964	A Hard Day's Night
1964	Beatles for Sale
1965	Help!
1965	Rubber Soul
1966	Revolver
1967	Sgt. Pepper's Lonely Hearts Club Band
1968	The Beatles
1969	Yellow Submarine
1969	Abbey Road
1970	Let It Be

   1. Écrire un programme affichant les titres des albums sortis une année paire.
      On créera deux listes : celle des années et celle des titres.

   Remarque : Il est possible d'itérer dans deux listes à la fois avec `zip` :

   ```
   fruits = ["banane", "fraise", "pastèque"]
   couleurs = ["jaune", "rouge", "verte"]
   for fruit, couleur in zip(fruits, couleurs):
       print(fruit, couleur)
   ```

   Dont l'exécution affiche :

   ```
   banane jaune
   fraise rouge
   pastèque verte
   ```

   2. Écrire le programme précédent à l'aide de la fonction `zip`
   3. Retour sur les albums des Beatles.
      On décide d'enregistrer les albums dans une liste de couples :

      ```
      albums = [(1963, "Please Please me"), (1963, "With the Beatles"), ...]
      ```

      Écrire une boucle qui affiche les années et titres des albums dont le titre contient la lettre "a".

5. Écrire une fonction qui prend une liste et renvoie sa longueur. On n'utilisera pas `len`
6. Écrire une fonction `index` qui prend une liste `l` et un objet `x` et renvoie :

- le premier indice de `x` dans `l`
- -1 si `x` n'est pas dans `l`

7. Écrire la fonction `copie_pairs` qui prend une liste d'entiers et renvoie la copie des entiers *pairs* de la liste.
8. Écrire une fonction qui prend une liste et renvoie une copie renversée de la liste.

Exercice 15 - Compléments sur les fonctions et les listes

1. Écrire une fonction `est_triee` qui prend en paramètre une liste et renvoie un booléen vrai si la liste est triée.
2. Écrire une fonction `concat` qui prend deux listes `l_1` et `l_2` et renvoie une nouvelle liste formée des éléments de `l_1` suivi des éléments de `l_2`
3. On considère la fonction :

```
def ajoute_paire(l_1, l_2):
    l_3 = []
    for i in ...:
        ...
    ...
```

Cette fonction prend deux listes, supposées de même taille et renvoie une nouvelle liste contenant les sommes des éléments de chaque liste.

Exemple :

```
ajoute_paire([1, 2, 3], [10, 12, 15])
[11, 14, 18]
```

Compléter la fonction.

4. On considère la fonction mystère suivante :

```
def mystere(l):
    z = 0
    t = 0
    for i in range(len(l)):
        z += l[i]
        t += 1
    return z / t
```

1. On suppose que le paramètre d'entrée `l` est une liste d'entiers. Quel est le type de sortie ?
2. Proposer une valeur de `l` qui provoque une erreur.
3. Faire tourner la fonction sur `l = [1, 2, 3]`
4. Que fait la fonction ?

5. ADN. Un brin d'ADN peut être modélisé par une chaîne de caractères ne contenant que des symboles `"a", "t", "g", "c"`
   1. Écrire une fonction `est_brin_ADN` qui reçoit en paramètre une chaîne et renvoie un booléen vrai si la chaîne ne contient que des caractères valides.

```
>>> est_brin_ADN("atcgdadt") # contient un "d"
False
>>> est_brin_ADN("atcgcadt") # c'est bon
True
```

   2. Lors de la réplication d'un brin d'ADN, l'organisme réalise une copie en utilisant l'association suivante :

```
"a" <-> "t"
"c" <-> "g"
```

Écrire une fonction association qui renvoie le complémentaire.

3. Écrire une fonction `replique` qui prend une chaîne d'ADN valide et renvoie le complémentaire.

6. Écrire une fonction `nb_div_par_2` qui prend un entier `n` et renvoie le nombre de fois où `n` est divisible par 2.

7. `split` est une méthode des chaînes de caractères qui permet de les découper en morceau. Écrire une fonction `split` qui prend deux chaines `phrase` et `symbole` et qui découpe la phrase en une liste de chaîne pour chaque `symbole`.

```
>>> split("salut les amis", " ")
["salut", "les", "amis"]
```

8. `join` fait le contraire de `split`. Elle regroupe une liste de chaines en une seule chaîne séparées par un caractère. Écrire une fonction `join` qui prend une liste de chaînes et une chaîne en paramètres et renvoie une seule chaîne formée des mots de la liste, séparés par le caractère.

```
>>> join(["salut", "les", "amis"], "-")
"salut-les-amis"
```

9. Écrire une fonction `ne_garder_que` qui prend deux paramètres `phrase` et `symboles`, des chaînes, et qui renvoie les symboles de `phrase` qui figurent dans `symboles` :

```
>>> ne_garder_que("mon num 0711223344 apl moi bb", "0123456789")
"0711223344"
>>> ne_garder_que("j'ai mange un cassoulet du tonnerre", "aeiouy")
"aieaoueuoee"
```

10. Écrire une fonction `premier_creu` qui reçoit une liste de nombres et renvoie l'indice du premier minimum local rencontré.

```
>>> premier_creu([7, 6, 4, 2, 3, 4, 9, 7, 4, 1, 0, 10])
3
```

Le premier minimum local est la valeur 2 (d'abord on descend : 7, 6, 4, 2 ; ensuite on monte : 2, 3, 4...). L'indice de la valeur 2 est 3. La fonction renvoie 3.

11. Écrire une fonction `est_palindrome` qui prend un chaîne et renvoie un booléen vrai si le mot est un parlindrome (*qui se lit de la même manière dans les deux sens, comme radar*).

```
>>> est_palindrome("radar")
True
>>> est_palindrome("radarzz")
False
```

12. Écrire une fonction qui renvoie le n^{ieme} terme de la suite $u_n = \dfrac{4n-2}{0.8^n}$.

Écrire une fonction qui calcule la somme des termes de (u_k) pour k entre 0 et n

13. Écrire une fonction qui renvoie le n^{ieme} terme de la suite $u_n = 2u_n - 2$ et $u_0 = 25$.

Exercice 16 - Dictionnaires

On considère le script suivant :

```
dict_eleve = {
    'nom': 'Figny',
    'prénom': 'Jean',
    'age': 16,
    }
```

1. Comment accéder au nom de l'élève ? À son prénom ?

2. Comment obtenir le nombre d'éléments de ce dictionnaire ?
3. Ajouter la moyenne de Jean, qui s'élève à 12.34.
4. On vient de célébrer l'anniversaire de Jean qui a maintenant 17 ans. Changer son age.
5. Jean vient de quitter l'établissement (renvoyé parce qu'il écoute du JuL). Supprimer sa moyenne du dictionnaire.

Exercice 17 - Depuis un dictionnaire vide.

1. Comment créer un dictionnaire vide ? Proposer deux syntaxes différentes.
2. Comment s'assurer qu'un objet enregistré dans une variable `d` est du type dictionnaire ? Proposer deux réponses différentes. Laquelle privilégier ?
3. Créer le dictionnaire `utilisateur` avec les couples clés, valeurs suivants :

clé	valeur
nom	Josephe
prenom	Apolline
age	22
password	juygvfesw

4. Écrire une fonction qui reçoit une liste de couples clés, valeurs et renvoie le dictionnaire correspondant :

   En entrée on reçoit une liste comme ceci :

```
entree = [('nom', 'Josephe'), ('prenom', 'Apolline'), ('age', 22),
          ('password', 'juygvfesw')]
```

   En sortie on veut :

```
sortie = {'nom': 'Josephe',
          'prenom': 'Apolline',
          'age': 22,
          'password': 'juygvfesw'}
```

   Écrire toutes les étapes à la main (création, ajout avec une boucle etc.)

   Remarque : La fonction `dict` transforme ce type de listes, contenant des couples, en un dictionnaire ayant exactement le format souhaité.

```
>>> dict( [(1: 2), (3: 4)] )
{1: 2, 3: 4}
```

5. Écrire une fonction `acceder` qui prend trois paramètres : `d` un dictionnaire, `k` une clé éventuellement présente et `defaut` un objet. Elle renvoie la valeur de `k` si `k` est une clé de `d` sinon la valeur par `defaut` :

```
>>> acceder({'a': 1, 'b': 5}, 'a', 2)
1
>>> acceder({'a': 1, 'b': 5}, 'c', 2)
2
```

6. Écrire une fonction `copier_sauf` qui prend deux paramètres : un dictionnaire `d` et une clé `k` et renvoie une copie de `d` privé de la clé `k`.

```
>>> copier_sauf({"a": 1, "b": 2, "c": 3}, "a")
{"b": 2, "c": 3}
```

Exercice 18 - Itérer sur un dictionnaire.

1. Quelles sont les trois manières d'itérer sur un dictionnaire en Python ?
2. Un site de jeux vidéos enregistre les scores de ses joueurs au jeu Pacman dans un dictionnaire :

```
scores_pacman = {'paul': 34,
                 'honorine': 456,
                 'marcel': 89,
                 'octave': 542,
                 'marine': 12,
                 'mélanie': 134,
                 'patricia': 631}
```

   Ajouter le score d'Amandine qui a 542 points. Attention à respecter la convention : tous les prénoms sont en minuscule.

3. Créer une fonction qui prend un dictionnaire tel que le précédent et le nom d'un joueur comme `'Amandine'` et retourne son score si le joueur est inscrit ou 0 sinon. À nouveau, attention à la convention d'enregistrement des noms.
4. Créer une fonction permettant d'inscrire un joueur. Elle respecte la signature ci-dessous.

```
incrire_joueur(score_jeu: dict, joueur: str) -> bool
```

   Elle retourne `True` si le joueur n'est pas déjà inscrit, `False` s'il est déjà inscrit. Un nouvel inscrit a un score de 0.

5. Depuis le dictionnaire précédent, créer à la liste des noms de joueurs. Proposer une fonction qui le fasse.
6. Depuis le dictionnaire précédent, calculer le score moyen des inscrits :
   a. En utilisant la fonction `sum`
   b. Sans utiliser la fonction `sum`
7. Créer une fonction qui prend le dictionnaire de joueurs en paramètre et retourne une chaîne de caractères contenant une série de phrases telles que celle ci-dessous, séparées par des retours à la ligne :
   `Le score de Paul est 34`

   À nouveau, prenez garde à la façon dont est noté le prénom.

   On pourra utiliser la méthode `upper` des chaînes de caractères :

```
>>> 'aBcD'.upper()
'ABCD'
>>> help(str.upper)
Help on method_descriptor:

upper(self, /)
    Return a copy of the string converted to uppercase.
```

8. La fonction précédente ne convient plus. On souhaite maintenant qu'elle retourne *une liste* de phrases (toujours une phrase par joueur, similaire à la précédente). Adapter votre fonction précédente.
9. Adapter la fonction précédente pour qu'elle retourne la liste des phrases :
   a. Triées par ordre alphabétique du prénom,
   b. Triées par score croissant.

   Trier une liste peut se faire avec la fonction `sorted` ou la méthode `sort` :

```
>>> ma_liste = [3, 2, 1]
>>> sorted(ma_liste)        # retourne une copie triée
[1, 2, 3]
>>> ma_liste
[3, 2, 1]
>>> ma_liste.sort()         # tri en place, ne retourne rien.
>>> ma_liste
[1, 2, 3]
```

Exercice 19 - Complément sur les boucles `while`

La syntaxe d'une boucle `while` est :

```
while condition:
    tour de boucle
```

Tant que `condition` est vraie, on tourne.

1. Écrire à l'aide d'une boucle `while` la boucle suivante :

   ```
   mot = "raisonable"
   c = 0
   for lettre in mot:
       if lettre == a:
           c = c + 1
   ```

   Intérêt ? Aucun, c'est même plus pénible.

2. On part d'un capital valant 1000€. Chaque année, il est augmenté de 5% (= multiplié par 1.05).

   Écrire une boucle `while` qui détermine le nombre d'année nécessaires pour doubler le capital.

3. La fonction `input(chaine)` :

   - affiche chaîne,
   - lit l'entrée standard (ce que tape l'utilisateur) et la renvoie sous la forme d'une chaîne :

   ```
   >>> nom = input("taper votre nom :")
   taper votre nom : MARCEL
   >>> nom
   "MARCEL"
   ```

   En enchaînant `int` et `input` on peut convertir une saisie en un entier.

   ```
   >>> nombre = int(input("votre nombre :"))
   votre nombre : 22
   >>> nombre
   22
   ```

   Remarquez que `nombre` est du type `int`.

   À l'aide des fonctions `randint` du module `random`, de `int` et de `input` écrire une fonction qui fait jouer l'utilisateur au "plus ou moins". Exemple de partie :

```
>>> plus_ou_moins()
votre nombre : 50
C est plus
votre nombre : 75
C est moins
votre nombre : 67
BRAVO !
Le nombre était 67, vous avez trouvé en 3 coups.
```

4. Écrire une fonction qui prend une liste d'entiers différents et compris entre 1 et 10 en paramètre et renvoie un entier aléatoire entre 1 et 10 *qui ne figure pas dans la liste reçue.*

```
>>> entier_aleatoire_un_dix([1, 2, 6, 7, 9, 10])
8
>>> entier_aleatoire_un_dix([1, 2, 6, 7, 9, 10])
5
```

Que se passe-t-il si tous les entiers entre 1 et 10 figurent dans la liste reçue ?

Améliorer votre fonction pour renvoyer -1 si tous les entiers sont choisis :

```
>>> entier_aleatoire_un_dix([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
-1
```

5. *Avec remise*

À l'aide de `random.randint` écrire une fonction qui prend trois paramètres entiers `a, b, n` et renvoie `n` valeurs entieres aléatoires entre `a` et `b` inclus.

```
>>> echantillon(10, 20, 5)
[12, 14, 11, 17, 17]
```

Sans remise

On souhaite maintenant que les valeurs soient toutes différentes. Ce n'est pas le cas de l'exemple précédent.

Il faut bien sûr avoir assez de valeurs pour en choisir `n`

Proposer une condition mathématique à vérifier entre `a`, `b` et `n` pour que ce soit possible.

Écrire une telle fonction qui :

- Renvoie `n` valeurs aléatoires *distinctes* si la condtion est respectée.
- Plante sinon (`assert condtion, "message"`)

```
>>> echantillon_sans_remise(10, 20, 5)
[15, 16, 18, 19, 10]
>>> echantillon_sans_remise(10, 12, 20)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in echantillon_sans_remise
AssertionError: pas assez de valeurs entre a et b
```

6. La conjecture de Syracuse s'intéresse au comportement des suites définies par :

- un premier terme u_0 entier positif,
- pour tout n, $u_{n+1} = \dfrac{u_n}{2}$ si u_n est pair et $u_{n+1} = 3n+1$ si u_n est impair.

La conjecture affirme que toute suite de ce type contient la valeur 1.

Par exemple, la suite des valeurs successives depuis 3 est :

$$3, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1 \ldots$$

Cette suite de valeurs contient bien 1.

a. Programmer une fonction `syracuse_vol` qui prend un entier positif `n` et renvoie la *liste des valeurs jusqu'à rencontrer 1 inclus*.
b. Programmer une fonction `syracuse_duree` prenant le même paramètre et qui renvoie *le nombre de termes* du vol.
c. Programmer une fonction `syracuse_hauteur` prenant le même paramètre et qui renvoie *la valeur maximale atteinte* durant le vol.