# 06-cours_python_fonctions

February 21, 2020

## 1 Les fonctions

### 1.1 Présentation

#### 1.1.1 Qu'est-ce qu'une fonction ?

Une fonction est un bloc d'un programme qui contient une série d'instructions.

- Une fonction peut être appelée n'importe où dans un programme.
- Les instructions contenues dans la fonction ne sont executées qu'à l'appel de la fonction.

Nous avons déjà utilisé des fonctions :

- `len(liste)` -> renvoie la longueur d'une liste
- `input(chaine)` -> renvoie une valeur saisie par l'utilisateur
- `print(valeur)` -> affiche une valeur à l'écran (mais ne renvoie rien)

Ces fonctions sont disponibles par défaut dans le langage Python.

#### 1.1.2 Ecriture d'une fonction

Outre l'utilisation des fonctions fournies par le langage, il est également possible pour le programmeur de définir des fonctions. Les avantages sont les suivants :

- Le code d'un programme peut être écrit par petits blocs, ce qui permet d'isoler correctement les différentes parties et de les modifier simplement. Cela est particulièrement utile lorsque le programme comporte de nombreuses lignes.
- Lorsqu'on répète une série d'instructions, on peut les isoler dans une fonction et l'appeler lorsqu'on en a besoin ailleurs dans le programme.

Une fonction s'écrit de la façon suivante :

```
def ma_fonction1():  # Fonction sans paramètre
    # Série d'instructions (indentées)
    ...

def ma_fonction2(a, b):  # Fonction avec deux paramètres
    ...
```

**Exemple : écriture d'une fonction mathématique** On peut aussi faire une analogie avec les fonctions mathématiques : on donne une ou des valeur(s) en entrée et on obtient une valeur en sortie. Par exemple, une fonction qui calcule le double d'un nombre donné s'écrira : $f(x) = 2x$. Si

on veut savoir quel est le double de 3, nous appellerons la fonction $f$ en lui indiquant que $x$ vaut 3 : $f(3)$. Le résultat, la valeur de retour de la fonction sera $2 \times 3$ soit 6. Commençons donc par écrire cette fonction en Python :

```
def f(x):
    resultat = 2*x
    return resultat
```

ou

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

La partie de définition du nom de la fonction et des paramètres (valeurs) qui lui seront transmis, se fait sur la première ligne par `def f(x):`. Notez que, comme pour les tests de conditions ou pour les boucles, nous débutons un nouveau bloc : toutes les instructions qui seront indentées (décalées par rapport à la marge de gauche), feront partie de la fonction.

La valeur de retour de la fonction est définie par l'instruction `return` (ici, ce sera $2*x$). Quand nous appellerons cette fonction, la valeur de retour sera calculée, renvoyée à la ligne de code ayant effectué l'appel, puis utilisée.

A noter que les instructions contenues dans la fonction **ne seront exécutées que lorsqu'on fera appel à la fonction**.

```
def f(x):
    # Contenu de la fonction
    return 2*x

# A partir d'ici le code se trouve hors de la fonction
# On appelle la fonction (et son code est exécuté)
f(3)
# Mais rien n'est visible à l'écran, car on ne demande pas d'affichage...
```

Si on souhaite un affichage:

```
def f(x):
    # Contenu de la fonction
    return 2*x

y = f(3)
print("Résultat : ", y)
```

Notons que si on ne souhaite pas stocker dans une variable le résultat de la fonction, on faire l'appel de la fonction directement dans la fonction `print`.

```
def f(x):
    # Contenu de la fonction
    return 2*x

print("Résultat : ", f(3))
```

Remarque : la transmission d'un paramètre à une fonction peut se faire également par une variable, par exemple :

```python
def f(x):
    # Contenu de la fonction
    return 2*x

a = 3
print("Résultat : ", f(a))
```

**Exercice : retour et affichage de valeur** Sans exécuter le programme ci-dessous, indiquez ce qu'affichent les instructions *print* ci-dessous :

```python
def fonction_polie():
    return "Bonjour"

# Programme principal
print("Début")
valeur = fonction_polie()
print("Appel #1", valeur)
print("Appel #2", fonction_polie())
fonction_polie()
print("C'est fini")
```

## 1.2 Remarque (très importante) sur l'organisation d'un programme

Par convention l'ordre d'un programme est le suivant :

```python
# Ajout des import éventuels
import ...

def ma_fonction1():
    ...

def ma_fonctionN():
    ...

# Programme "principal"
print("Bonjour je suis en forme")
ma_fonction1()
ma_fonctionN()
print("Ouf c'est fini !")
```

Si on écrit simplement :

```python
def ma_fonction1():
    ...

def ma_fonctionN():
    ...
```

Il ne se passe rien, car lorsque l'interpréteur Python exécute le code, **il n'exécute pas les fonctions tant qu'elles ne sont pas appelées**.

Que se passe t-il à l'exécution du programme suivant ?

```
def bonjour():
    print("Bonjour")

print("Mais que diable fais-je ici ?")

def aurevoir():
    print("Hasta luego")
```

Et pour ce programme ?

```
def bonjour():
    print("Bonjour")

print("Mais que diable fais-je ici ?")

def aurevoir():
    print("Hasta luego")

bonjour()
aurevoir()
```

### 1.3 Fonctions sans valeur de retour (procédure)

En informatique, une fonction ne prend pas forcément de paramètre(s) et ne renvoie pas forcément une valeur… Voici un exemple :

```
def ma_fonction():
    print("Bonjour")

ma_fonction()  # Appel de la fonction et affichage de bonjour
```

Remarque : une fonction qui ne comporte pas d'instruction return renvoie None par défaut, ce code est donc équivalent :

```
def ma_fonction():
    print("Bonjour")
    return None

ma_fonction()
```

### 1.4 Commentaire d'une fonction

Il est souvent judicieux de mettre une description de la fonction dans son entête. Pour cela, on va utiliser la possibilité en Python d'afficher une chaîne de caractères sur plusieurs lignes. C'est ce qu'on appelle la "docstring" de la fonction.

```
def racine_carree(valeur):
    """Fonction de calcul de la racine carrée
    @param valeur: la valeur dont on souhaite extraire la racine
    @return: la racine de la valeur ou None si le nombre est négatif
```

```
    """
    # Code de la fonction
    # ...
```

### 1.5 Exercice : écriture d'une fonction de calcul de racine carrée

Ecrire la fonction `racine_carree(valeur)`. Elle prend en paramètre une valeur et en retourne la racine carrée si la valeur est positive et `None` dans le cas contraire. On prendra soin d'écrire la *docstring* de la fonction.

[ ]:
```
def racine(valeur):
    """Fonction de calcul de la racine carrée
    valeur: La valeur dont on souhaite extraire la racine
    return: La racine de la valeur ou None si le nombre est négatif
    """
    if valeur >= 0:
        return valeur**0.5
    else:
        return None

# Programme principal
print(racine(4))
print(racine(-1))
```

[ ]:
```
# Remarque : on peut vérifier automatiquement le retour des valeurs avec␣
 ↪l'instruction assert
# Cela est particulièrement utile lorsqu'il y a beaucoup de cas à vérifier
def racine(valeur):
    """Fonction de calcul de la racine carrée
    valeur: La valeur dont on souhaite extraire la racine
    return: La racine de la valeur ou None si le nombre est négatif
    """
    if valeur >= 0:
        return valeur**0.5
    else:
        return None

# Vérification
assert racine(4) == 2
assert racine(-1) == None
## Une erreur Python va être déclenchée
assert racine(6) == 2
```

### 1.6 Exercices : appel de différentes fonctions de calcul

- Exercice 1 : écrire une fonction `affiche_surface_cercle` qui prend en paramètre le rayon et affiche la surface du cercle

- Exercice 2 : écrire une fonction `calcul_surface_cercle` qui prend en paramètre le rayon et renvoie la surface du cercle
- Exercice 3 : écrire une fonction `calcul_volume_cylindre` qui prend en paramètre le rayon du cercle ainsi que la hauteur et qui renvoie le volume

[ ]:
```python
# Exercice 1

import math

def affiche_surface_cercle(rayon_cm):
    val_arr = round(rayon_cm**2 * math.pi, 2)
    print(val_arr, "cm²")

# Programme principal
affiche_surface_cercle(20)
```

[ ]:
```python
# Exercice 2

import math

def calcul_surface_cercle(rayon_cm):
    return rayon_cm**2 * math.pi

# Programme principal
surface_cm2 = calcul_surface_cercle(20)
print(surface_cm2, "cm²")
```

[ ]:
```python
# Exercice 3 - v1

import math

def calcul_volume_cylindre(rayon_cm, h_cm):
    return rayon_cm**2 * math.pi * h_cm

# Programme principal
volume_cm3 = calcul_volume_cylindre(20, 10)
print(volume_cm3, "cm3")
```

[ ]:
```python
# Exercice 3 - v2

import math

def calcul_surface_cercle(rayon_cm):
    return rayon_cm**2 * math.pi

def calcul_volume_cylindre(rayon_cm, h_cm):
    # Question : que se passerait-il si on appelait
```

```
    # affiche_surface_cercle au lieu de calcul_surface_cercle
    # dans la ligne ci-dessous ?
    return calcul_surface_cercle(rayon_cm) * h_cm

# Programme principal
volume_cm3 = calcul_volume_cylindre(20, 10)
print(volume_cm3, "cm3")
```

### 1.7 Exercice : calcul de l'écart-type d'une liste de nombres

- Calculer l'écart-type pour les listes suivantes (ne pas utiliser le module *statistics*) :

```
liste1 = [10, 11, 2, 5, 13]
liste2 = [1, 18, 20, 10]
liste3 = [10, 10, 10, 10]
```

- Ecrire une fonction `ecart_type` qui prend en entrée une liste de valeurs et qui retourne l'écart-type
- Dans le programme principal, effectuer les appels avec `liste1`, `liste2` et `liste3` et afficher le résultat

[ ]:
```
# V1
def ecart_type(valeurs):
    somme = 0
    for elem in valeurs:
        somme += elem
    moyenne = somme / len(valeurs)
    somme_carre = 0
    for v in valeurs:
        somme_carre += (v - moyenne)**2
    return (somme_carre/len(valeurs))**0.5

# Programme principal
liste1 = [10, 11, 2, 5, 13]
liste2 = [1, 18, 20, 10]
liste3 = [10, 10, 10, 10]
ecart_type_1 = ecart_type(liste1)
ecart_type_2 = ecart_type(liste2)
ecart_type_3 = ecart_type(liste3)
print(ecart_type_1, ecart_type_2, ecart_type_3)
# Ou sans variable intermédiaire
print(ecart_type(liste1), ecart_type(liste2), ecart_type(liste3))
```

[ ]:
```
# V2
import math

def ecart_type(valeurs):
    moyenne = sum(valeurs) / len(valeurs)
```

```
    total = 0
    for v in valeurs:
        total += (v - moyenne)**2
    return math.sqrt(total/len(valeurs))

liste1 = [10, 11, 2, 5, 13]
liste2 = [1, 18, 20, 10]
liste3 = [10, 10, 10, 10]
ecart_type_1 = ecart_type(liste1)
ecart_type_2 = ecart_type(liste2)
ecart_type_3 = ecart_type(liste3)
print(ecart_type_1, ecart_type_2, ecart_type_3)
# Ou sans variable intermédiaire
print(ecart_type(liste1), ecart_type(liste2), ecart_type(liste3))
```

### 1.8 Exercice : fonction qui recherche le minimum dans une liste de valeurs

- Reprendre le code du chapitre sur les boucles qui permettait de trouver la valeur minimale dans une liste de nombres et le mettre dans une fonction `cherche_min`, qui prendra en paramètre une liste et qui retournera le minimum (si la liste est vide, on retrournera `None`).
- Dans le programme principal, définir plusieurs listes et effectuer l'appel à la fonction.
- Valider que le resultat obtenu est bien celui attendu.

[ ]:
```
def trouve_min(liste):
    if len(liste) == 0:
        return None

    val_min = liste[0]
    for elem in liste:
        if elem < val_min:
            val_min = elem
    return val_min

## PP
liste1 = [1, 2, 3, 4, 5]
liste2 = [6, 2, 4]
liste3 = [6, 4, 3]
liste4 = [8, 8, 8]
liste5 = [1]
liste6 = []

assert trouve_min(liste1) == 1
assert trouve_min(liste2) == 2
assert trouve_min(liste3) == 3
assert trouve_min(liste4) == 8
assert trouve_min(liste5) == 1
assert trouve_min(liste6) == None
```

### 1.9 Exercice : calcul de la vitesse

- Ecrire une fonction *calcul_vitesse_kmh* qui prend en argument la distance (en m) et le temps (minutes) et qui renvoie la vitesse en (en km/h).
- Dans le programme principal :
  - On demandera à l'utilisateur de saisir la distance et le temps et on fera ensuite appel à la fonction
  - On affichera ensuite le résultat à l'écran

[ ]:
```python
def calcul_vitesse_kmh(distance_m, temps_min):
    if temps_min == 0:
        return None  # On évite une erreur de division par 0
    return round((distance_m/1000) / (temps_min/60), 2)

# Programme principal
tps = int(input("Temps (min) : "))
dist = int(input("Distance (m) : "))
vitesse = calcul_vitesse_kmh(dist, tps)
print(f"La vitesse est de {vitesse} km/h")
```

### 1.10 Exercice : factoriser le code (éviter la duplication de code)

On souhaite demander à l'utilisateur son prénom, son nom et la ville où il habite pour ensuite l'afficher. Avant l'affichage on contrôlera que le prénom, le nom et la ville contiennent uniquement des caractères alphabétiques.

```python
# Du code dupliqué en pagaille !

saisie_ok = False
while not saisie_ok:
    prenom = input("Votre prénom :")
    if prenom.isalpha():
        saisie_ok = True
    else:
        print("Erreur de saisie")

saisie_ok = False
while not saisie_ok:
    nom = input("Votre nom :")
    if nom.isalpha():
        saisie_ok = True
    else:
        print("Erreur de saisie")

saisie_ok = False
while not saisie_ok:
    ville = input("Ville de résidence : ")
    if ville.isalpha():
```

```
        saisie_ok = True
    else:
        print("Erreur de saisie")

print(f"Bonjour {prenom} {nom}, vous habitez {ville}.")
```

Cette approche fonctionne mais est mauvaise :

- Duplication de code : les mêmes portions de code sont répétées et si on voulait faire un autre contrôle (ex. au moins 2 caractères saisis) il faudrait le refaire pour prénom, nom et ville.
- Autre désavantage : le code est long (et donc moins lisible)

On va donc opter pour une approche plus **élégante** qui va factoriser le code. Ecrire une fonction `saisie_chaine(question)` qui prend en paramètre une question et renvoie la valeur saisie par l'utilisateur.

[ ]:
```
def saisie_chaine(question):
    saisie_ok = False
    while not saisie_ok:
        reponse = input(f"{question} : ")
        if reponse.isalpha():
            saisie_ok = True
        else:
            print("Erreur de saisie")
    return reponse

# Programme principal
nom = saisie_chaine("Nom")
prenom = saisie_chaine("Prénom")
ville = saisie_chaine("Ville")
print(f"Bonjour {prenom} {nom}, vous habitez {ville}.")
```

[ ]:
```
# Version intermédiaire

def caracteres_ok(chaine_a_tester):
    return chaine_a_tester.isalpha()

def saisie_chaine(question):
    saisie_ok = False
    while not saisie_ok:
        reponse = input(f"{question} : ")
        if caracteres_ok(reponse):
            saisie_ok = True
        else:
            print("Erreur de saisie")
    return reponse

# Programme principal
nom = saisie_chaine("Nom")
```

```
prenom = saisie_chaine("Prénom")
ville = saisie_chaine("Ville")
print(f"Bonjour {prenom} {nom}, vous habitez {ville}.")
```

[ ]:
```
# Version qui accepte les "-" et les "'"

def caracteres_ok(chaine):
    for c in chaine:
        if not c.isalpha() and c not in ["'", "-"]:
            return False
    return True

def saisie_chaine(question):
    saisie_ok = False
    while not saisie_ok:
        reponse = input(f"{question} : ")
        if caracteres_ok(reponse):
            saisie_ok = True
        else:
            print("Erreur de saisie")
    return reponse

nom = saisie_chaine("Nom")
prenom = saisie_chaine("Prénom")
ville = saisie_chaine("Ville")
print(f"Bonjour {prenom} {nom}, vous habitez {ville}.")
```

[ ]:
```
# Remarque 2
# Si on désire améliorer le programme et contrôler qu'au moins deux caractères␣
 ↪sont saisis,
# il suffit de le rajouter dans la fonction :
def saisie_chaine(question):
    saisie_ok = False
    while not saisie_ok:
        valeur = input(f"{question} : ")
        if valeur.isalpha():
            if len(valeur) >= 2:
                return valeur
            else:
                print("Merci de saisir au moins deux caractères")
        else:
            print("Erreur de saisie")

# Programme principal
nom = saisie_chaine("Nom")
prenom = saisie_chaine("Prénom")
ville = saisie_chaine("Ville")
```

```
print(f"Bonjour {prenom} {nom}, vous habitez {ville}.")
```

### 1.11 Exercice génération de mot de passe

On souhaite créer un petit programme qui nous permettra d'avoir un mot de passe différent pour chaque site visité. Ainsi on créera une fonction qui entrée prendra l'identifiant utilisé pour se connecter sur le site ainsi que l'adresse du site. Elle renverra un mot de passe, élaboré à partir de ces deux paramètres. L'algorithme de génération du mot de passe est laissé à votre discrétion.

La structure du programme est la suivante :

```
def generer_mdp(identifiant, adresse_site):
    # Les lignes ci-dessous élaborent une méthode savante pour générer un mot de passe
    # à partir de l'identifiant et de l'adresse passés en paramètre
    ...
    return ...

# Programme principal

## Demander à l'utilisateur l'identifiant puis l'adresse du site
## Appeler la fonction et afficher le mot de passe généré
```

[ ]:
```
def generer_mdp(identifiant, adresse_site):
    # On inverse l'ordre des caractères
    valeur1 = identifiant[::-1]
    valeur2 = adresse_site[::-1]
    return valeur2 + valeur1

ident = input("Identifiant sur le site ")
adresse = input("Adresse du site")
mdp = generer_mdp(ident, adresse)
print(f"Mot de passe : {mdp}")
```

### 1.12 Exercice : calcul de la médiane

En théorie des probabilités et en statistiques, la médiane d'un ensemble de valeurs (échantillon, population, distribution de probabilités) est une valeur $x$ qui permet de couper l'ensemble des valeurs en deux parties égales : mettant d'un côté une moitié des valeurs, qui sont toutes inférieures ou égales à $x$ et de l'autre côté l'autre moitié des valeurs, qui sont toutes supérieures ou égales à x.

**Exemples** - Nombre impairs d'éléments : `[2, 4, 9]` -> médiane = 4 - Nombre pairs d'éléments : `[2, 4, 6, 8]` -> médiane = 5 (`(4+6)/2`)

**Indications**

- Reprendre l'exercice du tri à bulles dans le chapitre sur les listes et faire en sorte que le tri se fasse dans une fonction `tri_croissant(liste)`.
- Ecrire une fonction qui calcule la médiane en faisant appel à la fonction `tri_croissant`.

[5]:
```python
def tri_croissant(liste):
    """
    Effectue un tri croissant de la liste passé en paramètre
    Attention cette fonction modifie la liste originale
    @param liste: une liste de valeurs numériques
    @return None
    """
    permutation = True
    i = 0
    while permutation:
        permutation = False
        for elem in liste:
            if i < len(liste)-1:
                if liste[i] > liste[i+1]:
                    liste[i+1], liste[i] = liste[i], liste[i+1]  # Echange de␣
 ↪valeurs
                    permutation = True
            i += 1
        i = 0


def mediane(liste):
    """
    Calcule la médiane d'une liste
    @param liste: une liste de valeurs numériques
    @return: la médiane
    """
    tri_croissant(liste)
    lg_liste = len(liste)
    if lg_liste == 0:
        return None
    if lg_liste == 1:
        return liste[0]

    i = int(lg_liste / 2)
    if lg_liste % 2 == 0:  # Nb pair d'éléments
        return (liste[i-1] + liste[i]) / 2

    return liste[i]

# Programme principal
assert mediane([9, 2, 4]) == 4
assert mediane([6, 2, 4, 8]) == 5
assert mediane([]) == None
```

## 1.13 Exercice : la revanche des Dalton

Lucky Luke s'est endormi sur son cheval (Jolly Jumper). Les Dalton l'ont attrapé puis attaché sur les rails en gare de Guingamp. * Écrire un programme qui affiche un tableau me permettant de connaître l'heure à laquelle Lucky Luke sera déchiqueté par le train parti de la gare Montparnasse (Paris) à 9h (il y a 404 km entre la gare Montparnasse et Guingamp). * Le tableau prédira les différentes heures possibles pour toutes les vitesses de 200 km/h à 300 km/h, par pas de 10 km/h. * Écrire une fonction *ecrabouille* qui reçoit la vitesse du train et qui affiche l'heure du drame * Écrire le programme principal qui affiche le tableau demandé, à savoir :

```
À 200 km/h, Lucky Luke s'éteindra à 11h01
À 210 km/h, Lucky Luke s'éteindra à 10h55
À 220 km/h, Lucky Luke s'éteindra à 10h50
À 230 km/h, Lucky Luke s'éteindra à 10h45
À 240 km/h, Lucky Luke s'éteindra à 10h41
À 250 km/h, Lucky Luke s'éteindra à 10h37
À 260 km/h, Lucky Luke s'éteindra à 10h33
À 270 km/h, Lucky Luke s'éteindra à 10h30
À 280 km/h, Lucky Luke s'éteindra à 10h27
À 290 km/h, Lucky Luke s'éteindra à 10h24
À 300 km/h, Lucky Luke s'éteindra à 10h21
```

[ ]:
```python
def ecrabouille(vitesse):
    distance = 404
    heure_depart = 9
    heure = heure_depart + int(distance/vitesse)
    minutes = round((60 * distance / vitesse) % 60)
    print(f"À {vitesse} km/h, Lucky Luke s'éteindra à {heure}h{minutes:02d}")

for v in range(200, 301, 10):
    ecrabouille(v)
```

## 1.14 Exercice : deviner un nombre aléatoire

Objectif de l'exercice : tirer un nombre aléatoire entre *nb_min* et *nb_max* et le faire deviner à l'utilisateur. Voici quelques indications :

1. Ecrire une fonction `saisie_nombre(nb_min, nb_max)` :

- Elle demande à l'utilisateur de saisir un nombre entre *nb_min* et *_nb_max*
- Elle renvoie *None* si la valeur saisie par l'utilisateur est "f"
- Elle contrôle que la valeur est bien numérique et comprise entre *nb_min* et *nb_max* :
  - Si c'est le cas elle renvoie la valeur
  - Sinon elle indique l'erreur et demande à l'utilisateur de saisir une autre valeur

2. On appelera cette fonction depuis le "programme principal"

- On compte le nombre de tentatives
- Si la valeur retournée par la fonction `saisie_nombre` est *None* on affiche à l'utilisateur qu'il a perdu
- Sinon on effectue le test pour savoir si la valeur saisie est la valeur à deviner

- Si tel est le cas on lui affiche qu'il a gagné en X fois

[ ]:
```python
import random

def saisie_nombre(nb_min, nb_max):
    while True:
        valeur = input(f"Saisir une valeur entre {nb_min} et {nb_max} (ou 'f' pour terminer) :")
        if valeur == "f":
            return None
        if valeur.isnumeric():
            nombre = int(valeur)
            if nombre < nb_min or nombre > nb_max:
                print(f"Merci de saisir une valeur en {nb_min} et {nb_max}")
            else:
                return nombre
        else:
            print("Merci de saisir une valeur numérique")

# Programme principal

NB_MIN = 1
NB_MAX = 20
nb_tentatives = 0
nb_a_deviner = random.randint(NB_MIN, NB_MAX)

jeu_continue = True
while jeu_continue:
    valeur = saisie_nombre(NB_MIN, NB_MAX)
    nb_tentatives += 1
    if valeur == None:
        print(f"Vous avez abandonné au bout de {nb_tentatives}, le nombre à deviner était : {nb_a_deviner}")
        jeu_continue = False
    if valeur == nb_a_deviner:
        print(f"Bravo vous avez gagné en {nb_tentatives} fois !")
        jeu_continue = False
    elif valeur < nb_a_deviner:
        print("Valeur trop petite")
    else:
        print("Valeur trop grande")
```

### 1.15 Exercice : fonction d'extraction de valeurs numériques

On souhaite écrire la fonction `liste_valeurs_num` qui reçoit en paramètre une chaîne de caractères contenant des éléments quelconques (séparés par des espaces) et qui renvoie uniquement les valeurs numériques. On veillera également à ce que les valeurs ne soient pas dupliquées.

Exemples :

- `"123 1 3" => [123, 1, 3]`
- `"123 abc 123 def" => [123]`
- `"abc def" => []`

1) Commencer par réfléchir à l'algorithme

2) Indications techniques :

- La fonction `split(" ")` appliquée à une chaîne de caractères permet de mettre dans une liste les éléments d'une chaîne de caractères qui sont séparés par des espaces, par exemple : `"12 a 34".split(" ") -> ["12", "a", "34"]`
- La fonction `isnumeric()` appliquée à une chaîne de caractères permet de savoir si celle-ci contient une valeur numérique

[ ]:
```python
def liste_valeurs_num(chaine):
    liste_elem = chaine.split(" ")
    liste_res = []
    for elem in liste_elem:
        if elem.isnumeric() and int(elem) not in liste_res:
            liste_res.append(int(elem))
    return liste_res

# Programme principal
chaines_a_traiter = ["123 1 3", "123 abc 123 def", "abc def"]
for chaine in chaines_a_traiter:
    print(liste_valeurs_num(chaine))
```

### 1.16 Exercice : notes d'examen

```
notes_filles = [16, 13, 11, 12, 12.5, 16, 14.5, 15, 12, 14.5, 14, 14, 12, 10, 18, 15.5, 17, 16]
notes_garcons = [12, 11, 18, 15, 16, 20, 16, 17, 17, 14, 12.5, 15, 16, 14, 14, 14, 17, 10]
```

- Ecrire une fonction `moyenne(liste_valeurs)` qui retourne la moyenne en l'arrondissant
- Ecrire une fonction `ecart_type(liste_valeurs)` qui retourne l'écart-type en l'arrondissant
- Ecrire une fonction `affiche_valeurs(titre, moyenne, ecart_type)` qui affiche les résultats. Le titre sera soit "Général", "Filles" ou "Garçons"

Le programme aura la forme suivante :

```
import math

def calcul_moyenne(liste_valeurs):
    return ...

def calcul_ecart_type(liste_valeurs):
    return ...

def affiche_valeurs(titre, moyenne, ecart_type):
    print ...
```

```
# Programme principal
notes_filles = [16, 13, 15, 12, 12.5, 16, 17, 15, 12, 14.5, 14, 14, 12, 10, 18, 15.5, 17, 16]
notes_garcons = [12, 11, 18, 15, 16, 20, 16, 11, 12, 8, 12.5, 15, 16, 14, 14, 14, 17, 10]

## Appel des fonctions...
```

[ ]:
```
import math

def calcul_moyenne(liste_valeurs):
    return round(sum(liste_valeurs) / len(liste_valeurs), 2)

def calcul_ecart_type(liste_valeurs):
    moyenne = calcul_moyenne(liste_valeurs)
    total = 0
    for v in liste_valeurs:
        total += (v - moyenne)**2
    return round(math.sqrt(total / len(liste_valeurs)), 2)

def affiche_valeurs(titre, moyenne, ecart_type):
    print(titre)
    print("\tMoyenne :", moyenne)
    print("\tEcart-type :", ecart_type)

# Programme principal
notes_filles = [16, 13, 15, 12, 12.5, 16, 17, 15, 12, 14.5, 14, 14, 12, 10, 18,␣
↪15.5, 17, 16]
moyenne_filles = calcul_moyenne(notes_filles)
et_filles = calcul_ecart_type(notes_filles)

notes_garcons = [12, 11, 18, 15, 16, 20, 16, 11, 12, 8, 12.5, 15, 16, 14, 14,␣
↪14, 17, 10]
moyenne_garcons = calcul_moyenne(notes_garcons)
et_garcons = calcul_ecart_type(notes_garcons)

notes = notes_filles + notes_garcons  # Concaténation des deux listes

affiche_valeurs("Général", calcul_moyenne(notes), calcul_ecart_type(notes))
affiche_valeurs("Filles", moyenne_filles, et_filles)
affiche_valeurs("Garçons", moyenne_garcons, et_garcons)

if moyenne_filles >= moyenne_garcons:
    print("\nBravo mesdames")
else:
    print("\nLe masculin l'emporte")
```