Exercice 1

October 22, 2019

1 Algorithmique et Pensée Computationnelle

2 ACT Week 5 (Python)

2.1 Exercice 1

2.2 Fonctions

2.2.1 Rôle

Jusqu'à maintenant, nous avons vu les `variables`, `les structures de données` et les `boucles`. Aujourd'hui nous allons nous intéresser aux `fonctions`.

Les `fonctions` permettent d'enregistrer du code dans une variable afin de réutiliser celui-ci à plusieurs endroits, et ainsi éviter de devoir le réécrire.

Les `fonctions` en `Python` fonctionnent comme des fonctions mathématiques, c'est-à-dire qu'on les définit une fois et qu'on peut les réutiliser autant de fois que l'on veut. La différence avec les fonctions mathématiques, c'est qu'on peut se servir de nos fonctions pour d'autres choses que des calculs (en maths, on ne verra jamais la fonction `lancer_un_missile(cible)`, par exemple.)

2.2.2 Syntaxe

Pour définir une fonction, on utilise le mot `def` suivi du nom de notre `fonction`, puis des parenthèses `()`. Ces parenthèses peuvent contenir ou nom des noms d'`arguments`, mais nous reviendrons dessus plus tard. Pour appeler une fonction, il suffit d'écrire son `nom` suivi de parenthèses `()`.

Dans l'exemple suivant, nous déclarons une fonction du nom `print_hello` qui a pour unique utilité de `print` le mot `"Hello"`. Puis nous appelons cette fonction

```
def print_hello():
    print("Hello")

print_hello()
```

[2]:
```
def print_hello():
    print("Hello")

print_hello()
```

```
Hello
```

2.2.3 Exercice

Créez une `fonction` du nom de votre choix qui affiche votre prénom et appelez cette fonction

[1]:
```
# VOTRE CODE ICI
def print_nom():
    print("Tanja")

print_nom()
```

```
Tanja
```

2.2.4 Arguments

Comme dit précédemment, une fonction peut avoir un ou plusieurs `arguments`. Comme en maths, les arguments sont des valeurs que l'on passe à notre `fonction` et c'est avec ces valeurs que la fonction va effectuer ses opérations.

Par exemple en maths, lorsqu'on écrit `f(x) = x+2`, l'argument de la fonction `f` est `x`, je peux maintenant simplement écrire `f(2)`, ce qui signifie **remplacer x par 2 dans la fonction f**.

En `python`, les arguments fonctionnent de la même façon. Dans l'exemple suivant, nous créons une `fonction` du nom `print_name` qui prend un `argument` que nous appelons `name`. Nous nous servons de cet argument pour faire `print("Mon nom est", name)`. Nous appelons ensuite cette fonction avec un argument.

```
def print_name(name):
    print("Mon nom est", name)

print_name("Linus")
```

[ ]:
```
def print_name(name):
    print("Mon nom est", name)

print_name("Linus")
```

2.2.5 Exercice

Créez une fonction du nom de votre choix qui prend un `argument x` et qui `print x+1`

[6]:
```
# VOTRE CODE ICI
def fan(x):
    print(x+1)
fan(2)
```

```
3
```

2.2.6 Return

Il est très commun que nous voulions enregistrer le résultat d'une fonction dans une variable. Par exemple, en maths, si nous avons une fonction `f(x) = x + 15`, nous pouvons faire `y = f(10)` et

nous savons donc que `y` vaut `25`. En python, si nous écrivons:

```
def f(x):
    x + 15

y = f(10)
print(y)
```

Le `print(y)` va afficher `None`, car le résultat de `f(10)` ne vaut rien.

Pour résoudre ce problème, nous avons le mot-clef **`return`**, celui-ci permet de retourner une valeur de la fonction pour permettre d'enregistrer le résultat dans une variable. Pour reprendre l'exemple précédent:

```
def f(x):
    return x + 15

y = f(10)
print(y)
```

Cette fois ci `y` vaut bien `25`, car nous avons fait `return x + 15`

[7]:
```
# SANS RETURN
print("SANS RETURN")

def f(x):
    x + 15

y = f(10)
print(y)

print("\n--------------------\n")


# AVEC RETURN
print("AVEC RETURN")

def f(x):
    return x + 15

y = f(10)
print(y)
```

```
SANS RETURN
None

--------------------

AVEC RETURN
25
```

2.2.7 Exercice

Ecrivez une fonction de nom `f` qui prend un argument `x` et qui retourne `x * 10 + 2`, appelez cette fonction et enregistrer le résultat de celle-ci dans une variable `y`, `print y`.

[8]:
```
# VOTRE CODE ICI
def f(x):
    return (x*10)+2
y=f(2)
print(y)
```

```
22
```

2.2.8 Arguments par défaut

Une `fonction` peut avoir des arguments par défaut, c'est-à-dire des arguments optionnels qui prennent soit une valeur par défaut, soit une valeur donnée par l'utilisateur. Par exemple, la fonction suivante a un argument par défaut de nom `name`, qui vaut `"Bjarne"` par défaut.

```
def print_name(name = "Bjarne"):
    print("Mon nom est", name)

print_name("Ken")
print_name()
```

La première fois, nous appelons la fonction avec `"Ken"` comme argument, nous affichons donc `Mon nom est Ken`, la deuxième fois nous ne donnons par d'argument, l'argument par défaut est donc utilisé et nous affichons `Mon nom est Bjarne`.

[13]:
```
def print_name(name = "Bjarne"):
    print("Mon nom est", name)

print_name("Ken")
print_name()
```

```
Mon nom est Ken
Mon nom est Bjarne
```

2.2.9 Exercice

Ecrivez une fonction `f` qui prend un argument `x` avec une valeur par défaut `0` et qui retourne `x` au carré. Appelez cette fonction avec un argument puis sans.

[ ]:
```
# VOTRE CODE ICI
```

[19]:
```
def f(x=0):
    return x*x
f()
f(2)
```

[19]: 4

2.2.10 *args

Il peut arriver que nous ne sachions pas à l'avance combien d'arguments nous allons avoir, heureusement nous avons les `*args` et `**kwargs` dont nous pouvons nous servir pour permettre à l'utilisateur d'utiliser un nombre indéterminé d'arguments.

Par exemple, nous pourrions vouloir une fonction `somme` qui prend un nombre non déterminé d'arguments et qui retourne leur somme, pour ce faire, nous utiliserions les `*args`. Les `*args` permettent de faire en sorte que tous les arguments avec lesquels la fonction est appelée soient mis dans un `tuple` de nom `args` (on aurait pu appeler `args` d'une autre manière du moment qu'il y a une `*` devant le nom).

```
def somme(*args):
    print(args)
    total = 0
    for i in args:
        total = total + i
    return total

t = somme(3, 4, 1, 10)
print(t)
```

[20]:
```
def somme(*args):
    print(args)
    total = 0
    for i in args:
        total = total + i
    return total

t = somme(3, 4, 1, 10)
print(t)
```

```
(3, 4, 1, 10)
18
```

2.2.11 Exercice

Ecrivez une fonction de nom `max` qui peut recevoir un nombre indéterminé d'arguments, retournez l'élément le plus grand passé en argument.

[24]:
```
### VOTRE CODE ICI
def max(*args):
    maxi= 0
    for i in args:
        if i>maxi:
            maxi=i
        else:
```

```
            continue
    return maxi
max(3,5,6,3,5,9)
```

[24]: 9

2.2.12 **kwargs

De la même manière que les `*args` enregistrent un nombre indéterminé d'arguments dans un `tuple`, les `**kwargs` enregistrent un nombre indéterminé d'arguments dans un `dictionnaire`. L'intérêt c'est que nous pouvons nommer nos arguments dynamiquements, exemple:

```
def print_names(**kwargs):
    print(kwargs)
    for key in kwargs:
        print(key, "est", kwargs[key])

print_names(Sandrine = "docteur", Marc = "programmeur", Josephine = "avocate")
```

[25]:
```
def print_names(**kwargs):
    print(kwargs)
    for key in kwargs:
        print(key, "est", kwargs[key])

print_names(Sandrine = "docteur", Marc = "programmeur", Josephine = "avocate")
```

```
{'Sandrine': 'docteur', 'Marc': 'programmeur', 'Josephine': 'avocate'}
Sandrine est docteur
Marc est programmeur
Josephine est avocate
```

2.2.13 Exercice

Ecrivez une `fonction` qui reçoit des `**kwargs` de type `nom = profession`. Affichez uniquement les noms dont la profession est `"programmeur"`.

[26]:
```
### VOTRE CODE ICI
def hasard(**kwargs):
    print(kwargs)
    for key in kwargs:
        if kwargs[key]=="programmeur":
            print (key)
hasard(Sandrine = "docteur", Marc = "programmeur", Josephine = "avocate")
```

```
{'Sandrine': 'docteur', 'Marc': 'programmeur', 'Josephine': 'avocate'}
Marc
```

[ ]:

Exercice 2

October 22, 2019

1 Algorithmique et Pensée Computationnelle

2 ACT Week 5

2.1 Exercise 2

2.1.1 Partie A

Petits exercices sur les listes et dictionnaires

Afin de créer une liste `my_list` contenant 5 chiffres et afficher uniquement les 3 premiers chiffres de `my_list`, vous faites comme ci-dessous. Pour ajouter ou enlever des éléments à une liste, vous pouvez utiliser la méthode .append() ou .pop(). Un "+=" peut également être utilisé pour ajouter des éléments. Pour modifier les deux premiers éléments d'une liste, vous pouvez utiliser l'indexation comme ci-dessous. Pour vider une partie de la liste, il suffit de rien mettre entre les crochets.

[1]:
```
my_list = [1,3,5,7,11]
print(my_list[0:3])
my_list.append(15)
my_list.pop()
my_list += [17,30]
my_list[0:2] = [4]
my_list[0:3] = []
```

```
[1, 3, 5]
```

Après l'exécution du code ci-dessus, à quoi va ressembler `my_list`?

[2]:
```
print(my_list)
```

```
[11, 17, 30]
```

Ci-dessous, une liste `your_list` contenant 5 ingrédients de cuisine a été crée pour vous.

[3]:
```
your_list=["pain","lait","farine","sucre","oeufs"]
```

Veuillez faire les étapes suivantes dans la cellule code ci-dessous:

- Ajoutez 2 ingrédients de cuisine à `your_list` à l'aide du "+=". Les ingrédients sont "poivrons" et "courgettes".

- Ajoutez 1 ingrédients de cuisine à `your_list` en utilisant la méthode append(). L'ingrédient est "chocolat".
- Affichez l'ingrédient qui se trouve à la position 3 de la liste 'your_list.
- Supprimez le dernier élément de `your_list` en utilisant la fonction pop().
- Remplacez les deux premiers éléments de `your_list` par "fruits".
- Supprimez les deux premiers éléments de `your_list`.
- Supprimez toute la liste `your_list`.
- Vérifiez que la liste est vide grâce à la fonction len().

[7]:
```
your_list=["pain","lait","farine","sucre","oeufs"]
print(your_list)
your_list += ["poivrons","courgettes"]
print(your_list)
your_list.append("chocolat")
print(your_list)
print(your_list[3])
your_list.pop()
print(your_list)
your_list[0:2]=["fruits"]
print(your_list)
your_list[0:2]=[]
print(your_list)
your_list[:]=[]
print(your_list)
print(len(your_list))
```

```
['pain', 'lait', 'farine', 'sucre', 'oeufs']
['pain', 'lait', 'farine', 'sucre', 'oeufs', 'poivrons', 'courgettes']
['pain', 'lait', 'farine', 'sucre', 'oeufs', 'poivrons', 'courgettes',
'chocolat']
sucre
['pain', 'lait', 'farine', 'sucre', 'oeufs', 'poivrons', 'courgettes']
['fruits', 'farine', 'sucre', 'oeufs', 'poivrons', 'courgettes']
['sucre', 'oeufs', 'poivrons', 'courgettes']
[]
0
```

Afin de créer un dictionnaire contenant 3 paires de clé-valeur, ajouter une quatrième paire clé-valeur et effacer une paire, vous pouvez faire comme ci-dessous. Afin d'affichez uniquement les clés ou valeurs, vous pouvez faire comme ci-dessous.

[ ]:
```
annuaire = {"Shioban": 111, "Tyson": 222, "Shawn": 333 }
annuaire["Steven"] = 444
del annuaire["Tyson"]
print(annuaire.keys())
```

```
print(annuaire.values())
```

Ci-dessous, un dictionnaire `mes_courses` ayant comme clés 5 ingrédients et comme valeurs les quantités respectives à acheter a été crée pour vous.

[ ]:
```
mes_courses= {"pain":1,"lait":3,"farine":2,"sucre":1,"oeufs":6}
```

Veuillez faire les étapes suivantes dans la cellule code ci-dessous:

- Ajoutez une nouvelle paire clé-valeur à `mes_courses`. La clé est "chocolat" alors que la valeur est 3.
- Supprimez la première paire clé-valeur ayant comme clé "pains".
- Affichez toutes les clés de `mes_courses`.
- Affichez toutes les valeurs de `mes_courses`.
- Vérifiez si l'ingrédient "lait" est dans `mes_courses` en une seule ligne avec la méthode print(). La réponse doit être True ou False.
- Trouvez une façon de vérifier que l'ingrédient "lait" se trouve effectivement dans votre liste de clés.

[6]:
```
mes_courses= {"pain":1,"lait":3,"farine":2,"sucre":1,"oeufs":6}
mes_courses["chocolat"]=3
del mes_courses["pain"]
print(mes_courses.keys())
print(mes_courses.values())
print("lait"in mes_courses.keys())
print("lait"in mes_courses)
```

```
dict_keys(['lait', 'farine', 'sucre', 'oeufs', 'chocolat'])
dict_values([3, 2, 1, 6, 3])
True
True
```

2.1.2 Partie B

Petits exercises sur les fonctions

Afin de définir une fonction, il faut utiliser le keyword `def` suivi du nom de la fonction et de ses paramètres entre paranthèses. Le keyword `return` est une instruction qui a pour but de renvoyer une information à la fin de l'exécution d'une fonction. L'appel d'une fonction peut se faire dans la déclaration d'une variable afin que le résultat soit stocker dans cette variable comme ci-dessous.

[9]:
```
def multip(a,b):
  return a*b
result = multip(3,7)
```

Complètez la fonction ci-dessous afin qu'elle retourne le cube de l'argument x.

[10]:
```
def cube(x):
    # complètez ici
    return x**3

cube(3)
```

[10]: 27

Complètez la fonction ci-dessous afin qu'elle renvoie la somme du tuple *args*.

[11]:
```
def somme(*args):
    resultat = 0
    # complètez ici
    for i in args:
        resultat = resultat + i
    return resultat

print(somme(23))
print(somme(48,29,19))
```

```
23
96
```

Une fonction `weirdo` qui prend a comme argument a été crée pour vous. Complètez la fonction afin que:

- si a est impair, "weirdo" soit affiché.
- si a est pair et compris entre 10 et 20, "weirdo" soit affiché.
- sinon, "no weirdo" doit être affiché.

[13]:
```
X = int(input())
def weirdo (a):
    # complètez ici
    if (a%2==1) or (a%2==0) and (6<=a<=20):
        print("weirdo")
    else:
        print("no weirdo")

weirdo(X)
```

```
20
weirdo
```

Une fonction peut avoir aucun paramètre, un paramètre ou plusieurs paramètre. Il est également possible d'utiliser une variable comme argument pour appeler la fonction. Les prochains exercices illustrent ceci.

Complètez la fonction ci-dessous afin qu'elle affiche les nombres de 0 à 10.

[14]:
```
def counting():
    i = 0
    # complètez ici
    while i<=10:
        print(i)
        i=i+1

counting()
```

```
0
1
2
3
4
5
6
7
8
9
10
```

Complètez la fonction ci-dessous qu'elle affiche les nombres de 0 à n.

[15]:
```
def counting2(n):
    i = 0
    # complètez ici
    while i<=n:
        print(i)
        i=i+1
z = 20
counting2(z)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

```
16
17
18
19
20
```

Appelez la fonction counting en utilisant la variable zci-dessus.

Complètez la fonction ci-dessous afin qu'elle affiche les nombres de aà bavec l'intervalle c.

[17]:
```
def counting3(a,b,c):
    i = a
    # complètez ici
    while i<=b:
        print(i)
        i=i+c

counting3(5,20,3)
```

```
5
8
11
14
17
20
```

[ ]:
```

```

Exercice 3

October 22, 2019

1 Algorithmique et Pensée Computationnelle

2 ACT Week 5

2.1 Exercise 3 : Calculez les complexités

Pour chaque algorithmes ci-dessous, indiquez en une phrase, ce que font ces algorithmes et calculez leur complexité temporelle avec la notation O( ).

[ ]:
```
#Entrée: list de nombre entiers et M un nombre entier
def algo1(L, M):
    i = 1
    while i < len(L) and L[i] <= M:
        i += 1
    s = i-1
    return s
```

Solution: écris ta solution ici

La complexite temporelle est O(n): dans le pire des cas (à savoir quand tous les nombres de la liste L sont plus petits ou´egaux à M), la boucle “tant que” est executee n fois, car le nombre i est incrementé de 1 a chaque passage de la boucle.

len() -> O(1) K/2 -> O(logN)

When complexity time linear, it means that the fct goes through all the elements of the list

[ ]:
```
#Entrée: n un nombre entier
def algo2(n):
    s = 0
    for i in range(2*n):
        s += i
    return s
```

Solution: écris ta solution ici

La complexite temporelle est´egalement O(n): de l’ordre de 2n opérations sont effectuees, mais le facteur 2 ne nous importe pas ici.

[ ]:
```python
#Entrée: L and M are two list de nombre entiers
def algo3(L, M):
    n = len(L)
    m = len(M)
    for i in range(n):
        L[i] = L[i]%2
    for j in range(m):
        M[j] =
```

Solution: écris ta solution ici

La complexité temporelle est ici de O(n) + 0(m) = O(max{n, m})

def for for i=0-> i=1-> j=0-> j=1->

-> O(n+n)

[ ]:
```python
# Entrée: une matrice M carré (même nombre de lignes et de colonnes)
def algo4(M):
    for i in range(len(M)):
        for j in range(len(M)):
            if M[i][j] < 0:
                M[i][j] = 0
    return M
```

Solution: écris ta solution ici

La compléxity temporelle est de 0(n2)

def for for

i=0, j=0 j=1 i=1, j=0 j=1

-> O(n* n)

Exercice 4

October 23, 2019

1 Algorithmique et Pensée Computationnelle

2 ACT Week 5

2.1 Exercise 4

2.2 Problème 1 : Exprimer des dates

Ecrivez un programme python pour aider le videur du XIIIème siècle à calculer l'age des étudiants voulant entrer dans le bar. Le programme doit contrôler si un étudiant peut rentrer ou pas (minimum 18 ans) selon sa date de naissance. Le programme doit faire la conversion requise pour que les inputs de l'utilisateur soient perçus comme des nombres entiers. Assumez que toutes les années ont 365 jours. Le programme doit affiché à la fin "Old enough" ou "too young". Les inputs suivant seraient demandés à l'utilisateurs:

`Enter the student birthday: Year: Month: Day:`

[5]:
```
from datetime import date #import datetime aussi possible
#ajouter un print ici, puis complètez le code ci-dessous
year = int(input("Enter a year:"))
month = int(input("Enter a month:"))
day = int(input("Enter a day:"))
today = date.today() #par défaut sous format "année-mois-jours"
birthday = date(year, month, day)
age = today-birthday
if (age.days/365)>=18: #pour age en jours
    print("Old enough")
else:
    print("too young")
```

```
Enter a year:1996
Enter a month:09
Enter a day:03
Old enough
```

Veuillez porter attention à la manière dans le package `datetime` est importé pour pouvoir utiliser la méthode `today()` et `date`. Il est également intéressant de voir qu'on peut convertir `age` en jours pour résoudre le problème.

2.3 Problème 2 : Discriminant

Ecrivez un programme python pour trouver les racines d'une équation de 2ème degré (aX^2 + bX + c) en utilisant le discriminant.

Le discriminant (= b^2 − 4ac) dérivé des coefficients determine le type de racines: - Si le discriminant est égal à 0, l'équation a une racine (x = -b/2a) - Si le discriminant est supérieur à 0, l'équation a deux racines (x = (-b ± √)/2a) - Si le discriminant est inférieur à 0, l'équation n'a aucune racine.

Pensez à importer le module `math` pour utiliser le `math.sqrt()`:

> `from math import sqrt` est plus ou moins équivalant à `import math` sauf que l'appel de la méthode se fait autrement. Dans le premier cas, vous devez appeler uniquement `sqrt()` alors que dans le deuxième cas vous devez toujours mentionner le module comme `math.sqrt()`.

[6]:
```
#importer le module nécessaire ici, puis complètez le code ci-dessous
from math import sqrt #import math aussi possible
a = float(input("Enter a: "))
b = float(input("Enter b: "))
c = float(input("Enter c: "))
discriminant = b**2-4*a*c
if discriminant > 0:
    num_roots = 2
    x1 = (((-b)+sqrt(discriminant))/(2*a))
    x2 = (((-b)-sqrt(discriminant))/(2*a))
    print("2 roots: %f and %f"%(x1, x2))

elif discriminant == 0:
    num_roots = 1
    x = ((-b)/(2*a))
    print("1 roots: %f"%(x))

else:
    num_roots = 0
    print("No roots")
    exit() #sort "process finished with exit code" mais pas obligatoire
```

```
Enter a: 10
Enter b: 20
Enter c: 5
2 roots: -0.292893 and -1.707107
```

2.4 Problème 3 : Gestion stock

Cet exercice, divisé en plusieurs parties, consiste à mettre en place un système qui permet à une petite boulangerie de gérer son stock.

Nous avons déjà créé pour vous un système qui demande répétitivement un `sales keywords` au boulanger (afin que le boulanger puisse interagir avec le système). Ainsi, si le boulanger entre `exit`,

le programme doit se terminer. Sinon, le programme doit demander un autre `sales keywords` (voir les lignes du code ci-dessous correspondantes).

Nous avons également défini un dictionnaire nommé `stock` contenant le nombre de sandwichs disponibles selon leur type.

Partie A Vous devez ajouter maintenant une fonctionnalité qui permet au boulanger de gérer son stock avec une boucle `while`. Ainsi, lorsque le boulanger entre un `sales keywords`, il y a trois cas de figure :

1. Il entre le nom d'un sandwich (`saumon`, `salami` ou `fromage`) qui est en stock => le programme réduit le `stock` d'une unité
2. Il entre le nom d'un sandwich mais il n'est plus en stock => le programme affiche « out-of-stock »
3. Le `sales keywords` ne correspond pas au nom d'un sandwich => le programme affiche « invalid input »

Partie B Ajoutez une fonctionnalité qui permet, lorsque le boulanger entre `stock`, d'afficher l'état du stock pour chaque sandwich.

Partie C Ajoutez une fonctionnalité qui permet, lorsque le boulanger entre `analytics` d'afficher l'ordre dans lequel les sandwichs ont été vendus. Vous pouvez créer une liste vide `historySandwich` à laquelle chaque fois qu'un sandwich est vendu, le type du sandwich est ajouté.

[1]:
```
inputUser = ''
stock = {"salami": 3, "saumon": 3, "cheese":3,}
# complètez le code ici
historySandwichsold =[]
while inputUser != 'exit':
    inputUser = input("sales keywords: ")
    if inputUser in stock.keys():  # if s'execute une fois while s'execute tant␣
↪que c'est vrai
        if stock[inputUser]>0:
            stock[inputUser]-=1
            historySandwichsold.append(inputUser) #ajout dans la liste
        else:
            print("Out-of-order")
    elif inputUser =="stock":
        for type_sandwich in stock.keys():
            print("{0}:{1}".format(type_sandwich,stock[type_sandwich])) #sort␣
↪la clé et la valeur
    elif inputUser =="analytics":
        print("Today,sales order: ")
        for sandwich_sold in historySandwichsold:
            print(sandwich_sold) #plusieurs lignes
    elif inputUser == "exit":
        exit()
    else:
```

```
        print("invalid input") #au moins un des cas doit correspondre au if,␣
↪elif, else
```

```
sales keywords: salami
sales keywords: exit
```

```
ERROR:root:Invalid alias: The name clear can't be aliased because it is another
magic command.
ERROR:root:Invalid alias: The name more can't be aliased because it is another
magic command.
ERROR:root:Invalid alias: The name less can't be aliased because it is another
magic command.
ERROR:root:Invalid alias: The name man can't be aliased because it is another
magic command.
```

2.5 Problème 4 : Bubble sort

Implémentez l'algorithme « Bubble sort » en vous aidant du descriptif ci-dessous:

"Le principe de l'algorithme du tri à bulles est très simple à assimiler. Il est, et de loin, l'un des algorithmes de tri les plus simples qui soient. Pour vous expliquer le principe, je vais d'abord vous donner une courte explication écrite puis nous allons concrètement trier une liste de nombres. Le principe du tri à bulles est de comparer deux valeurs adjacentes et d'inverser leur position si elles sont mal placées. Alors, qu'entend-t-on par"mal placé" ? C'est très simple et surtout, c'est logique : si un premier nombre x est plus grand qu'un deuxième nombre y et que l'on souhaite trier l'ensemble par ordre croissant, alors x et y sont mal placés et il faut les inverser. Si, au contraire, x est plus petit que y, alors on ne fait rien et l'on compare y à z, l'élément suivant. C'est donc itératif. Et on parcourt ainsi la liste jusqu'à ce qu'on ait réalisé n-1 passages (n représentant le nombre de valeurs à trier) ou jusqu'à ce qu'il n'y ait plus rien à inverser lors du dernier passage. Avec de la logique, on s'aperçoit qu'au premier passage, on place le plus grand élément de la liste au bout du tableau, au bon emplacement. Pour le passage suivant, nous ne sommes donc plus obligés de faire une comparaison avec le dernière élément ; et c'est bien plus avantageux ainsi. Donc à chaque passage, le nombre de valeurs à comparer diminue de 1."

source : https://openclassrooms.com/fr/courses/1160591-le-tri-a-bulles

[4]:
```
liste = [1,7,4,-2,10,36,-6,97]
count = 0
while count != len(liste)-1:
    count = 0
    # complètez le code ici
    for i in range (0,len(liste)-1):
        if liste[i]>liste[i+1]:
            (liste[i], liste[i+1]) =  (liste[i+1], liste[i])
    for i in range (0,len(liste)-1):
        if liste[i]<liste[i+1]:
```

```
            count += 1 # pour savoir quand terminer, 7 comparaisons nécessaires␣
 ↪pour avoir tout juste

print(liste)

# le plus lent des algos en termes de rapidité et de nombres d'échanges ou␣
 ↪comparaisons
# n(n-1)/4 opérations pour n valeurs à trier
#O(n^2) -> quadratique pour bubble sort
# O(nlog2 (n)) pour merge sort
```

```
[-6, -2, 1, 4, 7, 10, 36, 97]
```

2.6 Problème 5 : Selection sort et recursion

Implémentez l'algorithme de « selection sort » en vous aidant du descriptif ci-dessous :

- Imaginez deux listes: La liste A contient des éléments entiers en désordre et la liste B est une liste vide
- Scannez la liste A à la recherche du plus petit élément. Supprimez cet élément de la liste A et insérez le à la fin de la liste B.
- Répétez l'étape ci-dessus jusqu'à ce que A soit vide.

[7]:
```
def select_sort(l,index):
    min_liste = l[index]
    # complèter la fonction ici
    for i in range (index,len(l)-1):
        if min_liste > l[i+1]:
            min_liste = l[i+1]
    l.remove(min_liste)
    l.insert(0,min_liste)
    index += 1
    if index < len(l):
        select_sort(l,index)
    else:
        print(l)


l = [3,5,6,9,-35,-7, 47]
select_sort(l,0)
```

```
[47, 9, 6, 5, 3, -7, -35]
```

2.7 Problème 6 : Merge Sort

(On l'appelle **Tri Fusion** en français)

Implémentez l'algorithme de **merge sort**. Les pas de l'algorithme sont comme suit [1]:

1. Si le tableau n'a qu'un élément, il est déjà trié.

2. Sinon, séparer le tableau en deux parties à peu près égales.
3. Trier récursivement les deux parties avec l'algorithme du tri fusion.
4. Fusionner les deux tableaux triés en un seul tableau trié.

[1] : https://fr.wikipedia.org/wiki/Tri_fusion

[1]:
```
def merge_sort(la_liste):
    # complèter la fonction
    longueur = len(la_liste)    # calculer la longueur de la liste
    # s'il n'y a pas plus d'un élément, retourner la liste
    if longueur == 1 or longueur == 0:
        return la_liste
    # sinon, diviser la liste en deux
    elif longueur > 1:
        index_milieu = int(longueur / 2)    # convertir la variable en nombre␣
↪entier (l'index ne peut pas être un nombre à virgule)
        partie_gauche = la_liste[0:index_milieu]    # la partie gauche va du 1er␣
↪élément à celui du milieu
        partie_droite = la_liste[index_milieu:longueur]    # la partie droite va␣
↪du milieu à la fin de la liste

        partie_gauche_triee = merge_sort(partie_gauche)    # appeler la fonction␣
↪merge_sort à nouveau sur la partie gauche (récursivité)
        partie_droite_triee = merge_sort(partie_droite)    # même chose pour la␣
↪partie droite

        liste_fusionnee = merge(partie_gauche_triee, partie_droite_triee)    #␣
↪enfin, joindre les 2 parties

        return liste_fusionnee    # retourner le résultat
```

[3]:
```
def merge(partie_gauche, partie_droite):
    liste_fusionnee = []    # créer la liste qui sera retournée à la fin

    compteur_gauche = 0    # définir un compteur pour l'index de la liste de␣
↪gauche
    compteur_droite = 0    # pareil pour la liste de droite

    longueur_gauche = len(partie_gauche)    # calculer la longueur de la partie␣
↪gauche
    longueur_droite = len(partie_droite)    # pareil pour la partie droite

    # continuer jusqu'à ce que l'un des index (ou les deux) atteigne l'une des␣
↪longueurs (ou les deux)
    while compteur_gauche < longueur_gauche and compteur_droite <␣
↪longueur_droite:
```

```
        # comparer les éléments actuels, ajouter le plus petit à la liste␣
↪fusionnée
        # et augmenter le compteur de cette liste
        if partie_gauche[compteur_gauche] < partie_droite[compteur_droite]:
            liste_fusionnee.append(partie_gauche[compteur_gauche])
            compteur_gauche += 1
        else:
            liste_fusionnee.append(partie_droite[compteur_droite])
            compteur_droite += 1

    # s'il y a encore des éléments dans les listes, il faut les ajouter à la␣
↪liste fusionnée
    liste_fusionnee += partie_gauche[compteur_gauche:longueur_gauche]
    liste_fusionnee += partie_droite[compteur_droite:longueur_droite]

    return liste_fusionnee   # retourner la liste fusionnée
```

[4]:
```
print(merge_sort([38, 27, 43, 3, 9, 82, 10]))
```

```
[3, 9, 10, 27, 38, 43, 82]
```

- Cliquez sur le lien ci-dessous pour analyser le merge sort pas à pas.

https://upload.wikimedia.org/wikipedia/commons/thumb/e/e6/Merge_sort_algorithm_diagram.svg/1024px-Merge_sort_algorithm_diagram.svg.png

- Cliquez sur le lien suivant pour observer la performance de merge sort sur différents types de listes.

https://www.toptal.com/developers/sorting-algorithms/merge-sort

[ ]:
```

```

Exercice 4_merge_sort

October 22, 2019

0.0.1 Merge Sort

(On l'appelle **Tri Fusion** en français)

Implémentez l'algorithme de **merge sort**. Les pas de l'algorithme étaient comme suit [1]:

1. Si le tableau n'a qu'un élément, il est déjà trié.
2. Sinon, séparer le tableau en deux parties à peu près égales.
3. Trier récursivement les deux parties avec l'algorithme du tri fusion.
4. Fusionner les deux tableaux triés en un seul tableau trié.

[1] : https://fr.wikipedia.org/wiki/Tri_fusion

[4]:
```
def merge_sort(la_liste):
    longueur = len(la_liste)    # calculer la longueur de la liste
    # s'il n'y a pas plus d'un élément, retourner la liste
    if longueur == 1 or longueur == 0:
        return la_liste
    # sinon, diviser la liste en deux
    elif longueur > 1:
        index_milieu = int(longueur / 2)    # convertir la variable en nombre␣
↪entier (l'index ne peut pas être un nombre à virgule)
        partie_gauche = la_liste[0:index_milieu]    # la partie gauche va du 1er␣
↪élément à celui du milieu
        partie_droite = la_liste[index_milieu:longueur]    # la partie droite va␣
↪du milieu à la fin de la liste

        partie_gauche_triee = merge_sort(partie_gauche)    # appeler la fonction␣
↪merge_sort à nouveau sur la partie gauche (récursivité)
        partie_droite_triee = merge_sort(partie_droite)    # même chose pour la␣
↪partie droite

        liste_fusionnee = merge(partie_gauche_triee, partie_droite_triee)    #␣
↪enfin, joindre les 2 parties

        return liste_fusionnee    # retourner le résultat
```

[5]:
```
def merge(partie_gauche, partie_droite):
    liste_fusionnee = []    # créer la liste qui sera retournée à la fin
```

```
    compteur_gauche = 0   # définir un compteur pour l'index de la liste de␣
↪gauche
    compteur_droite = 0   # pareil pour la liste de droite

    longueur_gauche = len(partie_gauche)   # calculer la longueur de la partie␣
↪gauche
    longueur_droite = len(partie_droite)   # pareil pour la partie droite

    # continuer jusqu'à ce que l'un des index (ou les deux) atteigne l'une des␣
↪longueurs (ou les deux)
    while compteur_gauche < longueur_gauche and compteur_droite <␣
↪longueur_droite:
        # comparer les éléments actuels, ajouter le plus petit à la liste␣
↪fusionnée
        # et augmenter le compteur de cette liste
        if partie_gauche[compteur_gauche] < partie_droite[compteur_droite]:
            liste_fusionnee.append(partie_gauche[compteur_gauche])
            compteur_gauche += 1
        else:
            liste_fusionnee.append(partie_droite[compteur_droite])
            compteur_droite += 1

    # s'il y a encore des éléments dans les listes, il faut les ajouter à la␣
↪liste fusionnée
    liste_fusionnee += partie_gauche[compteur_gauche:longueur_gauche]
    liste_fusionnee += partie_droite[compteur_droite:longueur_droite]

    return liste_fusionnee   # retourner la liste fusionnée
```

[6]:
```
print(merge_sort([38, 27, 43, 3, 9, 82, 10]))
```

```
[3, 9, 10, 27, 38, 43, 82]
```

- Cliquez sur le lien ci-dessous pour analyser le merge sort pas à pas.

https://upload.wikimedia.org/wikipedia/commons/thumb/e/e6/Merge_sort_algorithm_diagram.svg/1024px-Merge_sort_algorithm_diagram.svg.png

- Cliquez sur le lien suivant pour observer la performance de merge sort sur différents types de listes.

https://www.toptal.com/developers/sorting-algorithms/merge-sort

[ ]:
```

```

Exercice 5

October 23, 2019

1 Algorithmique et Pensée Computationnelle

2 ACT Week 5

2.1 Exercise 5 : Fibonacci

Voici le code python de l'algorithme de Fibonacci vu en cours. Il calcule le nème nombre de Fibonnacci.

1) Calculez la complexité de cet algorithme: trouvez la borne temporelle maximale de l'exécution de cet algorithme avec la notation O( ) .

[2]:
```
def fibonacci(n):
    if n == 0 or n == 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

[3]:
```
def fibonacci(n):
    if n == 0 or n == 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)
fibonacci(10)
```

[3]: 55

Solution: écris ta solution ici

O(2^n)

2) Essayez de calculer le 10ème puis le 40ème (cette dernière peut prendre du temps). La commande %%time au début de la cellule permet d'obtenir le temps d'éxécution de la cellule.

[12]:
```
%%time
print(fibonacci(10))
```

```
55
CPU times: user 153 µs, sys: 3 µs, total: 156 µs
```

```
Wall time: 163 µs
```

[18]:
```
%%time
print(fibonacci(20))
```

```
6765
CPU times: user 3.68 ms, sys: 3.94 ms, total: 7.62 ms
Wall time: 7.07 ms
```

3) Comme vous pouvez le voir le temps de calcul est très long lorsque n est grand. Maintenant il est temps d'optimiser l'algorithme vu en cours pour qu'il soit efficace. Celui que nous allons faire ne sera plus récursif mais sera tout simplment un algorithme itératif. Le problème de l'algorithme vu en cours et que dans les différents appels récursifs beaucoup de valeurs sont calculées plusieurs fois ce qui rend l'algorithme très **glouton**.

Indice: pour écrire votre algorithme, il est conseillé d'utiliser des variables supplémentaires.

[4]:
```
def fibonacci_fast(n):
    result=[]
    a = 0
    b = 1
    while a < n:
        result.append(a)
        tmp = b
        b = a + b # a=old, b=new
        a = tmp

    return result
print(fibonacci_fast(10))
```

```
[0, 1, 1, 2, 3, 5, 8]
```

[13]:
```
def fibonacci_fast(n):
    x = 0
    y = 1
    for i in range(2, n+1):
        tmp =  x + y #tmp = old et new variables
        x = y #old = new
        y= tmp #new =tmp

    return tmp
```

4) Essayez de calculer le 10ème puis le 40ème. Comparez les résultats avec ceux obtenus précédemment.

[14]:
```
%%time
print(fibonacci_fast(10))
```

```
55
CPU times: user 100 µs, sys: 2 µs, total: 102 µs
Wall time: 108 µs
```

[19]:
```
%%time
print(fibonacci_fast(20))
```

```
6765
CPU times: user 111 µs, sys: 0 ns, total: 111 µs
Wall time: 117 µs
```

Note: Ce que vous venez d'apprendre est le compromis temps-memoire très commun en programmation souvent il est possible d'utiliser de la mémoire supplémentaire pour réduire le temps de calcul !!! THIS IS AMAAAAAZZZZIINNNNG!!!