MEEF

Python

pour le professeur de mathématiques

Centre Universitaire
MAYOTTE

Sommaire

Les travaux dirigés

Sommaire :

- TD 1 : Arithmétique et calcul avec Python 1
  Notions : Variable, importation de module, `print`, `input`, `type`
- TD 2 : Boucles itératives 4
  Notions : Représentation des nombre flottant (Norme), boucle `for`
- TD 3 : Boucles conditionnelles 7
  Notions : Boucle `while`, Algorithme du seuil
- TD 4 : Fonctions 10
  Notions : Création (`def`) et appel d'une fonction.
- TD 5 : Instructions conditionnelles 16
  Notions : structure `if`, `if ... else`, , `if ... elif .... else`
- TD 6 : Liste 19
  Notions : Accés aux élements, ajout d'éléments, parcours
- TD 7 : Aléatoire et graphique 23
  Notions : Modules `random` et `matplotlib`

TP 1 : Arithmétique

Description

Objectifs

- ✔ Entrée et sortie
- ✔ Calculer avec Python
- ✔ Variable

Prérquis

— Aucun

Mémo de cours

Variable

Affectation

L'affectation d'une valeur à une varaible se fait à l'aide du signe égal.

```
ma_variable = valeur
```

Régles de nommage

Le nom d'une variable :

— doit commencer par une lettre

— ne doit pas contenir un signe opératoire + - * / %, d'espace ni le caractère # (*Il est réserver aux commentaires*)

— ne doit pas être un mot clé de Python (`if`, `in`, `as` ;), ...)

Type

Variable	Type	Remarques
`2`	`int` : Les nombres entiers relatifs	
`3.5`	`float` : Les nombres réels	Le séparateur est le . et non la ,
`"Bonjour"`	`str` : Les chaines de caractères	Entourer par des guillemets (`"`) ou des apostrophes (`'`) ou des triples guillemets (`"""`) ou des triples apostrophes (`'''`)
`True` ou `False`	`bool` : Les booléens True ou Fasle	

•• De Scratch à Python


```
1  ma_variable = 10
```

• Entrée-Sortie

•• L'instruction `print`

En Python, pour afficher du texte, on utilise l'instruction **`print`**.
Si l'on veut afficher une chaine de caractère, on la placera entre guillemets :

```
print("Bonjour")
```

•• Les `f-string`

Il peut être utile de placer une variable ou un calcul à évaluer dans une chaine de caractères. C'est possible de le faire en utilisant les **`f-string`**. Les guillemets ouvrants encadrant la chaine de caractères doivent être précédés d'un **`f`** et les éléments à évaluer encadrés par des accolades.

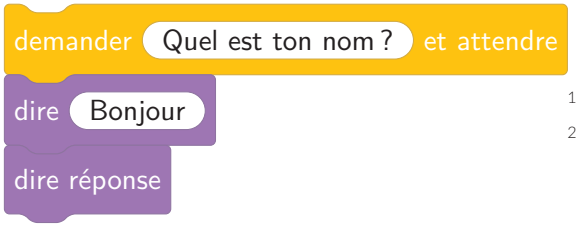
```
f" texte { expression à évaluer }"
```

•• L'instruction `input`

L'instruction permettant à l'utilisateur de faire une saisie clavier est l'instruction input. Il faut changer le type de la saisie si l'on veut des nombres.

Instruction	Description
chaine = input("Entrer un nombre")	Demander à l'utilisateur de saisir une chaine de caractères et l'affecter à la variable chaine
entier = int(input("Entrer un nombre"))	Demander à l'utilisateur de saisir un nombre entier et l'affecter à la variable entier
nombre = float(input("Entrer un nombre"))	Demander à l'utilisateur de saisir un nombre flottant et l'affecter à la variable nombre

•• De Scratch à Python




```
1  nom = input("Quel est ton nom ?")
2  print(f"Bonjour {nom}")
```

Calculs

Opérations de bases

Opérateur	Description
`+ - * /`	Opérateur arithmétique classique
`a**b`	a^b
`a%b`	reste de la division entière de `a` par `b` (a modulo b)
`a//b`	division entière de `a` par `b`

Le module `math`

Pour les opérations plus complexes, il faut importer le module `math`

```
from math import *
```

Opérateur	Description
`sqrt(a)`	$\sqrt{a}$
`exp(a)`	e^a
`log(a, b)`	$\log_b(a)$ le log de a dans la base b. *Par défaut e*
`pi`	π
`e`	Constante d'Euler
`degrees(a)`	Convertir un angle de radians à degrés
`radians(a)`	Convertir un angle de degrés à radians

TP 2 : Boucles itératives

Description

Objectif :

✔ Boucles itératives : `for ... in ... :`

Prérquis

– Variables

Memo de cours

Boucles itératives

En pyhton, les boucles itératives servent à répeter un bloc d'instructions un nombre prédéfini de fois.

for variable **in** itérable **:**
    bloc d'instructions

De Scratch à Python


```
1  for i in range(5):
2      print("Bonjour !")
```

Itérables

L'instruction `range`

Opérateur	Description
`range(nb)`	génère un itérable contenant les `n` entiers de 0 à `nb`-1
`range(nb1, nb2)`	génère un itérable contenant les entiers de `nb1` à `nb2`-1
`range(nb1, nb2, pas)`	génère un itérable contenant les entiers de `nb1` à `nb2`-1 en allant de `pas` en `pas`

— Les chaines de caractères : La variable de boucle prendra la valeur de chaque caractère de la chaine.
— Les listes : La variable de boucle prendra la valeur de chaque eélément de la liste.

Exercices

Exercice 1 Le compte en banque de Fry

Dans la série Futurama, Fry a déposé 0,93sur un compte bancaire à intéret composée au taux de 2,25%.

Mille ans plus tard, quel sera le montant disponnible ?

Exercice 2 Echiquier de Sissa

Le roi Belkib (Indes) promit une récompense fabuleuse à qui lui proposerait une distraction qui le satisferait.
Lorsque le sage Sissa, fils du Brahmine Dahir, lui présenta le jeu d'échecs, le souverain, demanda à Sissa ce que celui-ci souhaitait en échange de ce cadeau extraordinaire.
Sissa demanda au prince de déposer un grain de riz sur la première case, deux sur la deuxième, quatre sur la troisième, et ainsi de suite pour remplir l'échiquier en doublant la quantité de grain à chaque case.

Écrire un programme qui permet de calculer le nombre total de grains de riz nécessaire pour remplir ainsi l'échiquier.

Exercice 3 ★Fibonacci

En mathématiques, la suite de Fibonacci (F_n) est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent.

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \end{cases}$$

Écrire un programme permetttant de calculer F_{100}.

Exercice 4 ★

Pour étudier la probabilité qu'un fumeur arrête de fumer, on modélise la situation fumeur-non fumeur de la manière suivante :

- si le fumeur ne fume pas un jour donné, il ne fume pas le jour suivant avec une probabilité de 0,9 ;
- s'il fume un jour donné, il fume le jour suivant avec une probabilité de 0,6.

On note p_n la probabilité que le fumeur ne fume pas le jour$n$ et q_n la probabilité que le fumeur fume le jour n.
On admet que l'on peut calculer les termes des suites (p_n) et (q_n) avec les égalités.

- $p_0 = 0$
- $q_0 = 1$
- $\begin{cases} p_{n+1} = 0,9p_n + 0,4q_n \\ q_{n+1} = 0,1p_n + 0,6q_n \end{cases}$

Quelle est la probabilité que le fumeur ne fume pas le jour 30 ? Arrondir le résultat au centième.

Source : Cahier d'algorithmique et de programmation. 1ère. Barbazo

Exercice 5 Les nombres flottants

Des algues prolifèrent dans un étang.
Pour s'en débarrasser, le propriétaire installe un système de filtration. En journée, la masse d'algues est multiplier par 6, puis à la nuit tombée, le propriétaire actionne pendant une heure le système de filtration qui retire 7 kg d'algues. On admet que les algues ne prolifèrent pas la nuit.
Le propriétaire estime que la masse d'algues dans l'étang au matin de l'installation du système de filtration est de 1,4 kg.

On modélise par a_n la masse d'algues dans l'étang, exprimée en tonnes, après utilisation du système de filtration pendant n jours ; ainsi, $a_0 = 1,4$.

Écrire un programme python qui permet d'afficher a_{100}.

TP 3 : Boucles conditionnelles

Description

Objectifs

- ✔ Boucles conditionnelles : `while ... :`
- ✔ Opérateurs de comparaison et opérateurs booléens

Prérquis

- Variables

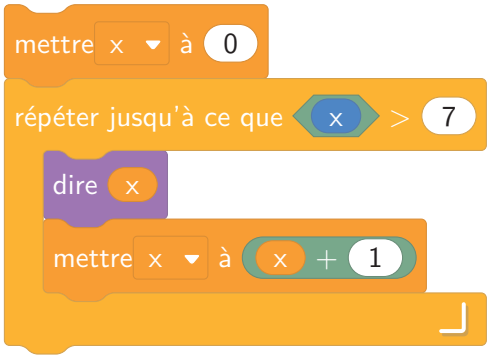
Memo de cours

Instruction `while`

Les boucles conditionnelles (utilisant l'instruction **`while`**) permettent de répéter un bloc de code tant qu'une condition est vraie.

```
while booleen :
    bloc instructions
```

De Scratch à Python




```
1  x = 0
2  while x <= 7:
3      print(x)
4      x += 1
```

! Entre Scratch et Python, les opérateurs de comparaison sont opposés.

• Opérateurs de comparaison

Les opérateurs de comparaison retournent soit `True` soit `False`.

Test	Sens	Remarques
a == 2	Test d'égalité	Le signe `=` étant réservé à l'affectation on utilise ici le double égal `==`
a != 2	Test de différence	
a > 3.5	Test de stricte supériorité	
a >= 3.5	Test de supériorité large	
a <	Test de stricte infériorité	
a <=	Test d'infériorité large	
"B" in "Bonjour"	Test d'appartenance	Nécessite un itérable

• Opérateurs booléens

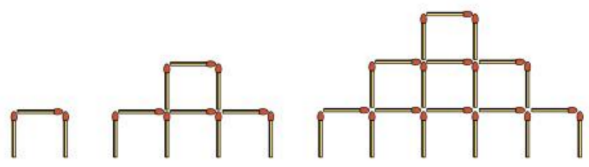
Les opérateurs booléens permettent de lier plusieurs expressions booléennes.

Test	Sens	Remarques
AND	et	retourne `True` si les deux conditions sont vraies et `False` sinon.
OR	ou	retourne `False` si les deux conditions sont fausses et `True` sinon.
NOT	non	transforme `True` en `False` et inversemment

Exercices

Exercice 6 Chateau d'allumettes

On s'intéresse à des pyramides construites avec des allumettes comme ci-dessous. En poursuivant ainsi, on obtient des pyramides à autant d'étages que l'on souhaite à condition, bien sûr, d'avoir assez d'allumettes.



Écrire un programme qui permet connaître le nombre maximal d'étages que l'on peut construire avec 1000 allumettes.

Source : Barbazo 1ère

Exercice 7 Algorithme de Babylone

L'algorithme de Babylone, est un algorithme d'approximation de racine carrée.
Pour déterminer une valeur approchée de la racine R du nombre A. On considère les suites (a_n) et (b_n) définie par :

$$\begin{cases} a_0 = A \\ a_{n+1} = \dfrac{a_n + b_n}{2} \end{cases} et\ b_n = \frac{A}{a_n}$$

La suite (a_n) (resp. (b_n)) converge par valeur supérieure (resp. inférieure) vers R
Écrire un programme qui permet d'obtenir une approximation au millième prés de $\sqrt{3}$

Exercice 8 ★

Dans cet exercice, on considère une feuille de papier A4 que l'on plie en deux.

Déterminer le nombre de pliages nécessaires pour que l'épaisseur d'une feuille de papier dépasse la distance Terre-Lune
Données :

- épaisseur d'une feuille de papier $\approx 0,1$ mm
- Distance Terre-Lune $\approx 384\ 400$ km

Exercice 9 Malthus

i Thomas-Robert Malthus, économiste anglais du début du XIX^e siècle, a vécu à une époque où l'industrialisation, couplée aux premières heures d'un marché dominant, ont permises l'expansion économique de la Grande Bretagne.
Durant celle-ci de grands chamboulements sociaux et des nouveaux moyens de productions créèrent autant de richesses que de pauvretés.
Au cours de ces recherches, il a travaillé sur l'évolution de la population en Angleterre.

En 1800, la population anglaise était de 8,3 millions d'habitants.
Bien que très pauvre en majorité, toute la population arrivait tant bien que mal à se nourrir.
On admettra par la suite qu'en 1800 la quantité de nourriture issue de la production agricole permettait de nourrir exactement les 8,3 millions d'habitants.

Thomas Malthus prévoit que cette situation ne pourra pas durer au cours du temps. Il émet les hypothèses suivantes :

- la population en Angleterre augmente chaque année de 2 % ;
- la production agricole anglaise aidée par des avancées techniques, permet de nourrir 400 000 habitants de plus par an.

Ecrire un programme Pyhton qui prédit ,selon ce modèle, en quelle année faut-il craindre une famine en Angleterre ?

TP 4 : Fonction

Description

Objectifs

✔ Définition et utilisation de fonctions

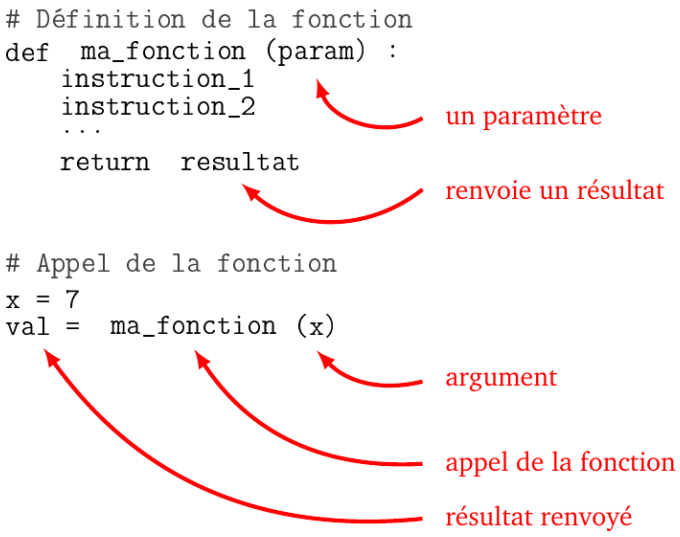
Prérquis

- Variables
- Boucles itératives

Memo de cours

i Il y a deux processus distincts dans l'utilisation des fonctions :

- La définition : étape pendant laquelle on définit les instructions à éxécuter par la fonction.
- Les appels : l'utilisation de la fonction avec des arguments particuliers.



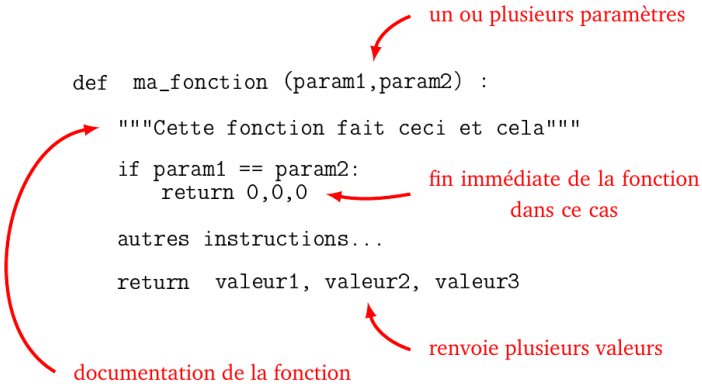

```
# Définition de la fonction
def  ma_fonction (param) :
    instruction_1
    instruction_2
    ...
    return  resultat

# Appel de la fonction
x = 7
val =  ma_fonction (x)
```

Définition d'une fonction

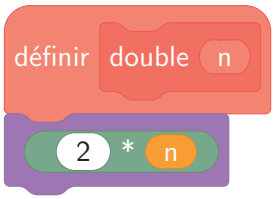
Pour **déf**inir une fonction on utilise le mot clé `def`.
Les parenthèses qui suivent le nom de la fonction accueillent les paramètres si nécessaire.
Les instructions de la fonction sont regroupées dans un bloc indenté.
Le mot `return` (optionnel) indique la fin de la fonction et précède les valeurs renvoyées par la fonction.

def nom_fonction **(** parametre_1**,** parametre_2 **):**
    bloc instructions
    return resultat



Source : Apprendre Python au lycée

•• De Scratch à Python

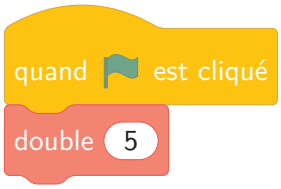



```
1  def double(n):
2      print(2*n)
```

• Appel d'une fonction

Pour utiliser une fonction il suffit de l'appeler par le nom qu'on lui a donné au moment de sa déclaration. Lors de cet appel, il faut lui fournir les arguments nécessaires.

•• De Scratch à Python




```
1  double(5)
```

• Paramètres obligatoires et paramètres optionnels

Il est possible de mettre des paramètres optionnels, pour cela il suffit de leur affecter une valeur lors de la déclaration de la fonction.

! Dans la délaration de la fonction, les paramètres optionnels doivent être en dernier

Exemple

```
1  def modulo(a,  b = 2):
2      mod = a % b
3      return mod
4
5  modulo(15)
6  modulo(15, 3)
```

— Le premier appel à la fonction `modulo`, c'est à dire `modulo(15)` renverra 1, car le second paramètre n'étant pas fourni, l'interpréteur pyhton prendra la valeur par défaut donc 2.

— Le second appel à la fonction `modulo`, c'est à dire `modulo(15, 3)` renverra 0, car le second paramètre est fourni, l'interpréteur pyhton renverra le reste de la division euclidienne de 15 par 3.

Exercices

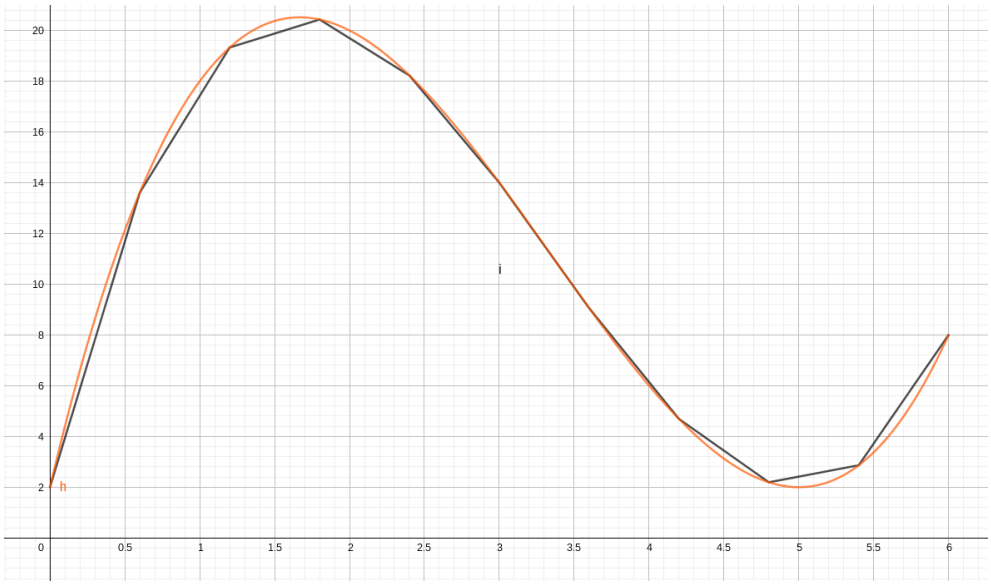
Exercice 10 Discriminant

Définir une fonction `discriminant` qui prend en paramètre `a`,`b` et `c`, des nombres flottants représentant les coefficients d'un polynôme de second degré (ax^2+bx+c) et qui renvoie le discriminant Δ.

nom	`discriminant`
paramètre.s	`a, b, c` nombres flottants
retour.s	Nombre flottant
description	fonction qui renvoie le discriminant du polynôme de second degré $ax^2 + bx + c$.

Exercice 11 Longueur

✔ Objectif : Implémenter une méthode d'approximation de la longueur de la courbe représentative d'une fonction sur un intervalle borné en l'approchant par la somme des longueurs des segments ayant pour extrémités des points de la courbe.



1. Définir une fonction `f`
   La fonction `f` prend en paramètre x un nombre flottant et renvoie l'image de x par la fonction f définit par :

$$f(x) = x^3 - 10x^2 + 25x + 2$$

nom	`f`
paramètre.s	`x` un nombre flottant
retour.s	nombre flottant
description	fonction qui renvoie l'image de x par la fonction f.

2. Définir une fonction `distance(xA, yA, xB, yB)`.
   La fonction `distance` qui prend en argument les 4 nombres xA, yA, xB et yB, représentant les coordonnées des points $A(xA,\ yA)$ et $B(xB,\ yB)$ dans un repère othonormal, est renvoie la longueur AB.

nom	`distance`
paramètre.s	`xA, yA,xB,yB` nombres flottants
retour.s	Nombre flottant
description	fonction qui renvoie la distance entre les points $A(xA,\ yA)$ et $B(xB,\ yB)$.

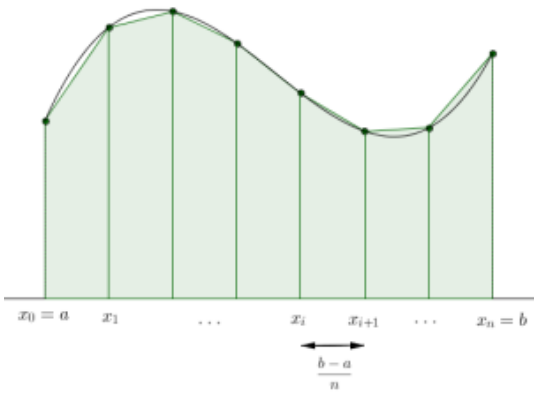
3. Définir une fonction `longueur_courbe(f, a, b, n)` prend en paramètre `f` une fonction, `a` un nombre flottant, `b` un nombre flottant et `n` le nombre de segments utilisés pour l'approximation et renvoie la longueur.

nom	`longueur_courbe`
paramètre.s	`f` une fonction `a` la borne inférieure de l'intervalle `b` la borne supérieure de l'intervalle `n` le nombre de segments utilisés
retour.s	nombre flottant
description	fonction qui renvoie `True` si `n` admet une décomposition telle que le présente la conjecture de Goldbach et `False` sinon.

4. A l'aide de la fonction `longueur_courbe`, estimer la longueur de la courbe représentative de la fonction f sur l'intervalle $[\,0\,;\,6\,]$ avec 50 segments ?

Exercice 12 Aire sous la courbe

✔ Objectif : Implémenter la méthode des trapèzes pour calculer l'aire sous la courbe d'une fontion f continue positive (c'està dire l'aire de la surface comprise entre l'axe des abcsisses, les droites d'équation $x = a$ et $x = b$ et la courbe représentative de la fonction f).



1. Définir une fonction `aire_trapeze(xA, yA, xB, yB)`.
   La fonction `aire_trapeze` qui prend en argument les 4 nombres xA, yA, xB et yB (), renvoie l'aire d'un trapèze dont les coordonnées des sommets seraient $(xA\,;\,0), (xA\,;\,yA), (xB\,;\,0)$ et $(xB\,;\,yB)$.

nom	`aire_trapeze`
paramètre.s	`xA, yA, xB, yB` nombres flottants
retour.s	Nombre flottant
description	l'aire d'un trapèze dont les coordonnées des sommets seraient $(xA\,;\,0), (xA\,;\,yA), (xB\,;\,0)$ et $(xB\,;\,yB)$.

2. Définir une fonction `f`
   La fonction `f` prend en paramètre x un nombre flottant et renvoie l'image de x par la fonction f définit par :

$$f(x) = \frac{10}{5 + \mathrm{e}^{-2x+9}}$$

nom	`f`
paramètre.s	`x` un nombre flottant
retour.s	nombre flottant
description	fonction qui renvoie l'image de x par la fonction f.

3. Définir une fonction `aire_sous_courbe(f, a, b, n)` prend en paramètre `f` une fonction, `a` un nombre flottant, `b` un nombre flottant et `n` le nombre de trapèzes utilisés pour l'approximation et qui renvoie l'approximation de l'aire sous la courbe de f.

nom	`aire_sous_courbe`
paramètre.s	`f` une fonction `a` la borne inférieure de l'intervalle `b` la borne supérieure de l'intervalle `n` le nombre de trapèzes utilisés
retour.s	nombre flottant
description	fonction qui renvoie .

4. A l'aide de la fonction `aire_sous_courbe`, estimer la longueur de la courbe représentative de la fonction f sur l'intervalle $[\,0\,;\,6\,]$ avec 50 segments ?

TD 5 : Instructions conditionnelles

Description

Objectifs

- ✔ Mise en place de structures conditionnelles : `if ...`, `if ... else ...`, `if ... elif ... else`,

Prérquis

- Variables
- Fonctions

Memo de cours

i Un instruction conditionnelle est composée d'un test (généralement construit autour d'un opérateur de comparaison) puis d'un bloc d'instructions qui sera executé, ou non, en fonction de la validité du test.

Syntaxe

En Python, le test commence par le mot clef **if** suivi d'un test à valeur booléenne (`True` ou `False`) et se termine par le symbole **:**
Le bloc d'instructions qui suit s'exécute si et seulement si le test a pour valeur `True`

3 cas de figure

S'il n'y a qu'un seul cas à distinguer, on utilisera uniquement l'instruction `if` :

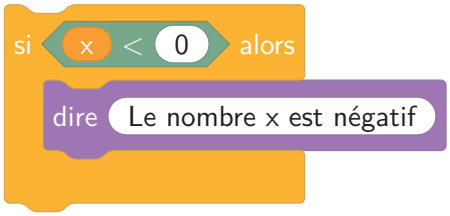
```
if test:
    bloc_d_instructions
```

S'il n'y a que deux cas à distinguer, on utilisera le couple `if` et `else` :

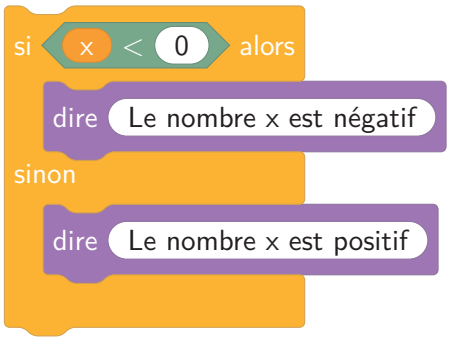
```
if test:
    bloc_d_instructions_1
else:
    bloc_d_instructions_2
```

S'il y a plus de deux cas, on utilisera `elif` pour ajouter des tests.

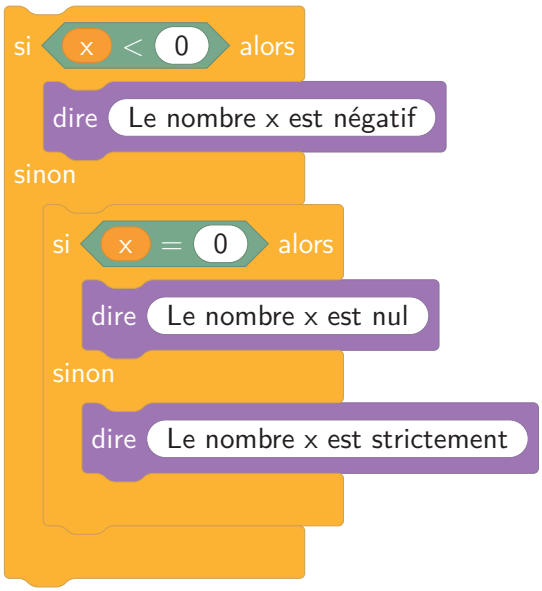
```
if test_1:
    bloc_d_instructions_1
elif test_2:
    bloc_d_instructions_2
elif test_3:
    bloc_d_instructions_3
else:
    bloc_d_instructions_4
```




```
1 if x < 0 :
2     print("Le nombre x est négatif)
```




```
1 if x < 0 :
2     print("Le nombre x est négatif")
3 else :
4     print("Le nombre x est positif")
```




```
1 if x < 0 :
2     print("Le nombre x est négatif")
3 elif x ==  0:
4     print("Le nombre x est nul")
5 else:
6     print("Le nombre x est positif")
```

Exercices

Exercice 13 Alignement de 3 points

Définir une fonction `alignement` qui prend en paramètre `xA, yA, xB, yB, xC` et `yC` les coordonnées respectives des points A, B et C et qui **affiche** le cas échéant :

- `True` si les points A, B et C sont alignés.
- `False` si les points A, B et C ne sont pas alignés.

Exercice 14 Racine d'un trinôme

Dans cet exercice, on considère le trinôme ax^2+bx+c.
L'objectif du programme est d'afficher, si elles existent, les racines du trinôme arrondi au centième. Définir une fonction racine qui prend en paramètre 3 nombres a, b et c et qui renvoie :

- Les valeurs x_1, x_2 où x_1 et x_2 sont les valeurs des racines arrondies au centième
- La valeur x_1 où x_1 est la valeur de la racine arrondie au centième
- `none` si le trinôme n'a pas de racine.

Exercice 15 Pythagore

Définir une fonction `pythagore` qui prend en paramètre 3 nombres `a, b` et `c` et qui renvoie :

- `True` si le triangle dont les longueurs des côtés sont a, b et c est rectangle.
- `False` sinon.

⚠ On ne sait pas quel est le plus grand des paramètres de la fonction.

TD 6 : Les listes

Description

Objectifs

- ✔ Liste

Prérequis :

- Boucle itérative
- Fonction (définition,utilisation)
- Instructions conditionnelles

Mémo de cours

i Une liste ou tableau est un type de variable permettant de garder en mémoire une collection de données.

Syntaxe

Définition — Syntaxe d'une liste

En python, les listes sont contenues dans des crochets [...]
Les éléments sont séparés par des virgules ,

```
ma_liste = [ elmt1, elmt2, elmt3 ]
```

Exemple

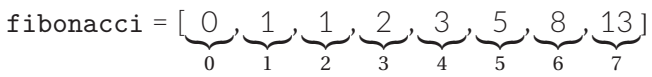
```
1    fibonacci = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Accéder à un élément

Pour accéder à un élément, on utilise son **index** (sa position dans la liste)

```
ma_liste [ index ]
```

⚠ L'index d'une liste commence à 0.



Exemple

```
fibonacci = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
print(fibonacci(6))
```

L'instruction ci-dessus affichera 8.

•• Modifier à un élément

Pour modifier à un élément, il suffit de le désigner par son index dans la liste puis on lui affecte une nouvelle valeur à l'aide du signe égal.

⚠ L'index d'une liste commence à 0.

```
ma_liste[index] = nouvelle_valeur
```

Exemple

```
fibonacci = [0, 1, 1, 2, 3, 5, 8, 1, 21, 34]
fibonacci[7] = 13
print(fibonacci)
```

L'instruction ci-dessus affichera `[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]`.

•• Longueur d'une liste

Il peut être utile de connaître la longueur d'une liste. En python, la fonction **`len`** renvoie la longueur de la liste.

Exemple

```
fibonacci = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
longueur = len(fibonacci)
```

Le code ci-dessus créera une variable **`longueur`** et lui affecte la valeur `12`.

•• Minimum / maximum /somme des valeurs d'une liste

Il existe des fonctions pour déterminer le maximum, le minimum et la somme des éléments d'une liste.

Exemple

```
fibonacci = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
maximum = max(fibonacci)
minimum = min(fibonacci)
somme = sum(fibonacci)
```

Le code ci-dessus créera les variables **`maximum`**, **`minimum`** et **`somme`** et leurs affectera respectivement les valeurs `89`, `0` et `232`.

i Il existe deux manières de parcourir les valeurs d'une liste à l'aide d'une boucle `for`.

- Parcourir les valeurs
- Parcourir les indices

De manière générale, on principalement le parcours des valeurs. Le parcours à l'aide des indices est utilisé si la position d'un élement doit être récupérer ou si l'on veut modifier la liste au cours de la boucle.

— Parcourir les valeurs

Exemple

```
1  # Création d'une variable somme initialisée à 0
2  somme = 0
3  # Création d'une liste de notes
4  notes = [10, 15, 8, 12]
5  # Parcour de chacune des notes contenues dans la liste notes
6  for valeur in notes:
7      # A chaque itération la variable valeur prend succésivement les valeurs
       ↪ de la liste notes
8      somme += valeur
9  print(somme)
```

— Parcourir les indices

Exemple

```
1  # Création d'une variable somme initialisée à 0
2  somme = 0
3  # Création d'une liste de notes
4  notes = [10, 15, 8, 12]
5  # Parcour de chacune des notes contenues dans la liste notes
6  for indice in range(len(notes)):
7      # A chaque itération la variable indice prendra les valeurs 0, 1, 2 et 3
8      somme += notes[indice]
9  print(somme)
```

i Le parcours d'une liste peut être nécessaire si :

- La position des éléments est importantes (ex : on compare un élément et son successeur)
- Si l'on veut modifier la valeur des élements

Exercices

Exercice 16 ★

On dispose d'une liste de notes, nommée `notes`, et d'une liste de coefficients, nommée `coeffs`, qui correspont au coefficient.

```
notes = [10, 8, 15, 14, 12, 18, 6, 20]
coeffs = [1, 0.5, 2, 1, 1, 0.5, 1.5, 2]
```

Ainsi le coefficient de la note 10 et 1, celui de la note 8 est 0,5.
Déterminer la moyenne pondérée de cette série de notes.

Exercice 17 ★

La suite de Syracuse d'un nombre entier $N > 0$ est définie par récurrence, de la manière suivante :

- $u_0 = N$
- et pour tout entier naturel n :

$$u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair,} \\ 3u_n + 1 & \text{si } u_n \text{ est impair.} \end{cases}$$

On déterminer pour une suite de Syracuse donnée, on détermine :

- le `temps de vol` comme étant le plus petit indice n tel que $u_n = 1$
- `l'altitude maximale` comme étant la valeur maximale de la suite.

Les premiers termes de la suite Syracuse 15 sont :

$$\underbrace{15}_{u_0}, \underbrace{46}_{u_1}, \underbrace{23}_{u_2}, \underbrace{70}_{u_3}, \underbrace{35}_{u_4}, \underbrace{106}_{u_5}, \underbrace{53}_{u_6}, \underbrace{160}_{u_7}, \underbrace{80}_{u_8}, \underbrace{40}_{u_9}, \underbrace{20}_{u_{10}}, \underbrace{10}_{u_{11}}, \underbrace{5}_{u_{12}}, \underbrace{16}_{u_{13}}, \underbrace{8}_{u_{14}}, \underbrace{4}_{u_{15}}, \underbrace{2}_{u_{16}}, \underbrace{1}_{u_{17}}, \underbrace{4}_{u_{18}}, \underbrace{2}_{u_{19}}, \underbrace{1}_{u_{20}}, \ldots$$

On a donc pour la suite Syracuse 15 :

- le temps de vol de cette suite est 17
- l'attitude maximale est 160

Déterminer une fonction `syracuse` qui prend en paramètre `n` le premier terme de la suite de syracuse et renvoie une liste contenant tous les termes de la suites jusqu'au premier terme de valeur 1.
A l'aide de la fonction `syracuse`, déterminer le temps de vol et l'attitude maximale de la suite Syracuse 27.

Exercice 18 Doublon

Déterminer une fonction `doublon` qui prend en paramètre une liste, nommée `tab` et qui renvoie :

- `True` si la liste comporte des doublons
- `False` sinon

TD 7 : Aléatoire & visualisation

Description

Objectifs

- ✔ Utilisation d'une bibliothèque de générateur de nombres pseudo-aléatoires : `random`
- ✔ Utilisation d'une bibliothèque de visualisation : `matplotlib`

Prérequis :

- Boucle itérative
- Fonction (définition,utilisation)
- Instructions conditionnelles
- Liste

Mémo de cours

Le module random

Le module `random` implémente des générateurs de nombres pseudo-aléatoires.

Importation du module

Il y a deux façons d'importer le module `random`.

- L'import étoilé permet d'importer l'intégralité des fonctions du module `random`.

```
1 from random import *
```

- L'importation ci-dessous nécessitera de préfixé toutes les fonctions du nom du module, (i.e. `random`).

```
1 import random
```

Fonction du module

Fonction	Description
`random()`	Renvoie un nombre compris entre 0 et 1 ($0 \leq x < 1$)
`randint(nb1, nb2)`	Renvoie un entier compris entre les entiers `nb1` et `nb2` (nb1 $\leq x \leq$ nb2)
`choice(liste)`	Effectue un tirage aléatoire parmi les éléments de la liste.
`choices(liste,k=n)`	Effectue `n` tirages aléatoires avec remise parmi les éléments de la liste `liste`, `n` étant un nombre entier.
`sample(liste, n)`	Effectue `n` tirages aléatoires sans remise parmi les éléments de la liste `liste`, `n` étant un nombre entier.
`suffle(liste)`	Renvoi une liste comportant les élèments de la liste `liste` mais mélangé.

Visualisation

Pour visualiser les tirages, nous allons utiliser le module `matplotlib`.

• Tracer une courbe (`simplePlot`)

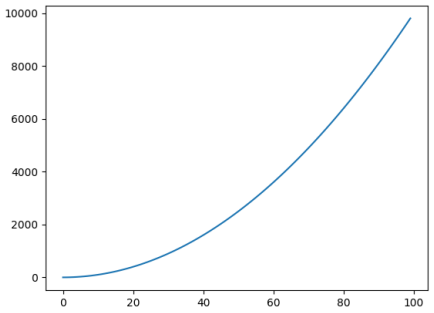
Pour tracer la courbe représentative d'une fonction à l'aide du module `matplotlib`, on utilise à la fonction `plot` qui prend en paramètre une liste d'abscisses, et une liste d'ordonnées.
La fonction accepte aussi des paramètres optionnels :

— `color` la couleur

Exemple

Tracer de la fonction $x \mapsto x^2$

```
1  import matplotlib.pyplot as plt
2
3  # Création des listes contenant les
   ↪  abscisses et les ordonnées
4  abscisses = [ x for x in range(100)] #
   ↪  crée une liste contenant les
   ↪  entiers de 0 à 99 (inclus).
5  ordonnees = [x**2 for x in abscisses]
6
7  # Création du graphique
8  plt.plot(abscisses, ordonnees)
9  plt.show()
```



• Tracer un point ou un nuage de point (scatterPlot)

Pour tracer un nuage de point à l'aide du module `matplotlib`, on fournit à la fonction `scatter` une liste d'abscisses, et une liste d'ordonnées.
La fonction accepte aussi comme paramètres optionnels :

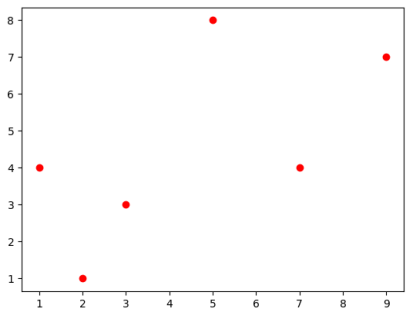
— `c` la couleur

— `f` la forme du point

Exemple

Nuage de points

```
1  import matplotlib.pyplot as plt
2  x = [1,2,3,7,9,5]
3  y = [4,1,3,4,7,8]
4
5  plt.scatter(x,y,c="red")
6  plt.show()
```



Tracer un diagramme en barres

Pour tracer un diagramme en barres à l'aide du module **matplotlib**, on fournit à la fonction `bar` une liste d'abscisses, et une liste d'ordonnées.
La fonction accepte aussi comme paramètres optionnels :

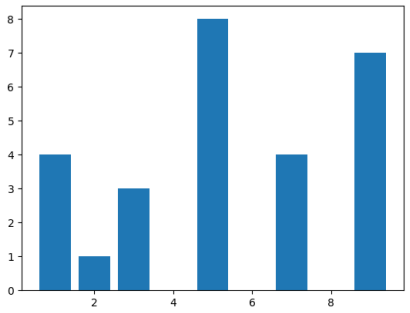
— **color** la couleur

Exemple

```
import matplotlib.pyplot as plt
x = [1,2,3,7,9,5]
y = [4,1,3,4,7,8]

plt.bar(x,y)
plt.show()
```

Diagramme en barres



Tracer un diagramme en circulaire

Pour tracer un diagramme à l'aide du module **matplotlib**, on fournit à la fonction `pie` une liste d'abscisses, et une liste d'ordonnées.
La fonction accepte aussi comme paramètres optionnels :

— **color** la couleur

Exemple

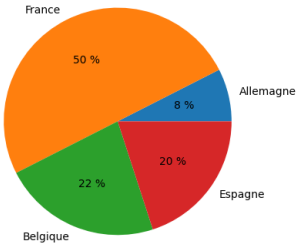
```
import matplotlib.pyplot as plt

labels = 'Allemagne', 'France',
↪  'Belgique', 'Espagne'
sizes = [15, 100, 45, 40]

plt.pie(sizes,
    labels=labels,
    autopct=lambda x: f"{round(x)} %")

plt.show()
```

Diagramme circulaire



Exercices

Exercice 19 ★

Le paradoxe des anniversaires affirme que, dans une population de 23 personnes, la probabilité qu'au moins deux d'entre elles aient leur anniversaire le même jour (mais pas la même année) est supérieur à 50

On supposera par la suite que :

- les années sont toutes de 365 jours (pas de naissance le 29 février)
- chaque date entre le 1er janvier et le 31 décembre est équiprobable

Définir une fonction `experience`, sans paramètre, qui crée une liste de 23 nombres compris en 1 et 365 et qui renvoie 0 si cette liste ne contient pas de doublon et 1 si celle-ci conteint des doublons.

Répéter 5000 fois l'expériences et afficher la fréquence des expériences ayant eu des doublons.

Le paradoxe semble-t'il vérifier ?

Exercice 20 ★

On considère deux dés à six faces, parfaitement équilibrés. On considère la variable aléatoire X qui prend pour valeurs la somme des deux dés.

Définir une fonction `experience` qui choisit aléatoirement deux nombres aléatoirement entre 1 et 6 et retourne leur somme.

On cherche à répéter l'expérience, tout en gardant en mémoire les différentes sommes obtenues.
Créer une liste nommée `compteurs` contenant 11 zéros (l'instruction `compteurs = [0]*11` peut être utilisée).

Cette liste servira à stocker le nombre d'occurences des différentes sommes.
Ainsi la liste `[0, 4, 1, 0, 7, 9, 0, 9, 12, 4, 3]` signifie que lors des répétitions de l'expériences, on a eu :

- La somme 2 est apparue 0 fois
- La somme 3 est apparue 4 fois
- La somme 4 est apparue 1 fois
- La somme 5 est apparue 0 fois
- .....

Répéter 5000 fois l'expérience du lancé de 2 dès, en incrémentant à chaque fois le compteur correspondant au résultat de l'expérience.
Créer un diagramme en barres représentant les résultats obtenus.
Quelle somme apparait le plus fréquemment.

Exercice 21 ★

Un puce se déplace sur un axe gradué. A chaque saut, elle avance ou recule de façon aléatoire d'une unité. Elle part de l'origine de l'axe O et effectue 4 sauts. On note X l'abscisse sur laquelle se trouve la suite à la fin de ces 4 sauts.

Les abcsisses $\{4;3;2;1;0;1;2;3;4\}$ sont-elles équiprobables ?
Définir une fonction `experience` sans paramètre qui choisit 4 fois aléatoirement le nombre 1 ou le nombre -1 et qui renvoie la somme des quatre nombres.

On cherche à répéter l'expérience, tout en gardant en mémoire les différentes abscisses obtenues.
Créer une liste nommée `compteurs` contenant 9 zéros (l'instruction `compteurs = [0]*9` peut être utilisée).

Cette liste servira à stocker le nombre d'occurences des différentes abscisses.
Ainsi la liste `[0, 4, 1, 0, 7, 9, 0, 9, 12]` signifie que lors des répétitions de l'expérience, on a eu :

- L'abscisse -4 est apparue 0 fois
- L'abscisse -3 est apparue 4 fois
- L'abscisse -2 est apparue 1 fois
- L'abscisse -1 est apparue 0 fois
- .....

Répéter 5000 fois l'expérience du saut de puce, en incrémentant à chaque fois le compteur correspondant au résultat de l'expérience.

Créer le diagramme en barres dont les abscisses sont la position de la puce à la fin de l'expérience et les ordonnées sont le nombre d'occurences chaque issue.