35. Algorithmique et Python

1. algorithme

1.1. Algorithme et Python

Définition 1.

Un algorithme est une succession d'un nombre fini d'étapes réalisées dans un ordre précis et qui a pour but de résoudre un problème.

Les instructions doivent être formulées dans un langage compréhensible par l'exécutant.

- humain ⇒ langage courant.
- ordinateur ⇒ langage de programmation (Python).
- algorithmique ⇒ pseudo-code. C'est un langage entre le langage courant et un langage de programmation, sans norme précise, qui met l'accent sur la logique de l'algorithme.

Python C'est un langage de programmation interprété. Pour écrire un programme en Python :

1. on écrit un texte contenant les instructions (avec vocabulaire et grammaire précises).
2. on demande à Python d'exécuter le programme.
3. Python lit le texte et suit les instructions.
4. Si tout va bien, il va au bout. Si il rencontre une erreur, il s'arrête et renvoi un message d'erreur.

Il faut une distribution Python et un éditeur de code pour l'utiliser. On va privilégier les tout-en-un.

- Sur ordinateur : Installer un logiciel (Spyder, Idle, Anaconda, WinPython)

— Sur internet, dans un navigateur :
https://www.programiz.com/python-programming/online-compiler/
https://repl.it/languages/python3
— Sur un téléphone ou une tablette : installer une application

1.2. Ecrire un programme en Python

Dans les logiciels cités précédemment, on trouve deux fenêtres.

— une console. On peut taper directement dedans des instructions simples après les symboles >>>, valider pour les faire éxecuter et voir le résultat apparaitre en-dessous. C'est aussi là que s'affichent par défaut les résultats produits par Python lorsqu'il exécute un programme.
— Un éditeur de texte. On l'utilise pour taper des programmes plus évolués, les sauvegarder, ouvrir des programmes enregistrés... Les mots-clés Python apparaissent souvent en couleur pour les repérer. Quand on exécute (Run) ces programmes, Python les lit et produit un résultat qui peut s'afficher dans la console.

1) Editeur. le code est un texte contenant les instructions Run/Exécuter On demande à Python d'exécuter le programme. Python lit le texte et suit les instructions.	2) Console. Les résultats ou les erreurs s'affichent.

On peut aussi taper le code dans un logiciel autre, et le faire exécuter ultérieurement par un compiler Python.
Quelques éléments de base en Pyhthon :

— Les commentaires (mots, phrases que python ne lit pas) commencent par le symbole #. Les commentaires sont utiles pour expliquer le fonctionnement du

programme et se répérer dans les programme complexes.

— Les blocs d'instructions sont identifiés par l'indentation (les espaces en début de ligne).

— Des mots sont réservés à Pyhton et ne peuvent pas être utilisé pour nommer autre chose

if, else, while, def, int, string, bool, print, input.....

Exemple.

Python

```
a= 5
def Calcul(n):  # Ceci est un commentaire
  if n<2 :
    return 1
  else:
    return (n*9 -2)
b=2+a
c =Calcul(b)
print(c)
```

La fonction print petiteboitevioletprint est une fonction Python. Elle permet d'afficher quelque chose dans la console, car Python n'affiche pas ce qu'il fait par défaut (sauf les erreurs). print peut afficher beaucoup de types de variables (textes, nombres, matrices, booléen etc.). print(♡) affiche ♡ :

— Si ♡ est une variable, ça affiche le contenu de la variable

— Si ♡ est une chaine de caractère, ça affiche le texte

— Si ♡ est une opération, ça affiche le résultat de l'opération

On peut afficher plusieurs éléments à la suite en les séparant par des virgules dans la parenthèses : print(♡, ♠, ♢)

La fonction input La commande ♠ = input(♡) va afficher le message ♡ dans la console et attendre. Il faut entrer quelque chose au clavier et valider. Ce qu'on a tapé

au clavier est alors stocké dans la variable ♠ au format string.

Remarque : La variable créée par input est de type *string*. Si on veut qu'elle soit un nombre, il faut la convertir en appliquant une fonction supplémentaire : `int(input("Entrer un nombre "))` (conversion en entier), ou `float(input("Entrer un nombre "))` (conversion en nombre décimal).

Ecrire le programme dans un éditeur permet de faire des instructions plus longue qu'une ligne, de sauvegarder, modifier plus facilement et organiser son programme.

2. Les variables

2.1. Définition de variable

Définition 2.

Une variable est un espace de stockage dans un emplacement de la mémoire (de l'ordinateur).

Caractéristiques :

- nom : permet d'identifier la variable ;
- type : définit quelle donnée peut contenir la variable ;
- valeur : utilisée par le programme, peut éventuellement changer ;
- (adresse : l'endroit de l'ordinateur où est stockée la variable.)

Remarque : Python gère tout seul les types de variables (c'est quand même bien de savoir avec quel type on travaille).

Définition 3.

Déclarer une variable, c'est attribuer un nom et un type à une "boite mémoire".

Affecter une variable, c'est associer une valeur à la boite.

En Python, on donne le nom et la valeur. Le type est déduit automatiquement dans les cas simples.

$$\heartsuit = \diamondsuit$$

avec $\heartsuit$ un nom de variable et $\diamondsuit$ la valeur qu'on veut mettre dedans

Cette instruction met la valeur $\diamondsuit$ dans la variable nommée $\heartsuit$. Si la variable n'existait pas avant, ça crée la variable.

Exemple. A, b, prix et Total ci-dessous sont des variables. A, b, prix et Total sont des variables. A est un int, prix et Total des float et b un string

Python

```
A= 50
prix = 1.52
Total = A * prix
b= "prix de l'essence"
```

Propriété 4.

Bonnes pratiques pour nommer une variable :
- Commencer par une lettre minuscule.
- Choisir un nom explicite pour bien comprendre le programme
- On peut utiliser des majuscules au milieu du nom ou bien utiliser le symbole _ pour séparer des parties du nom.
- Eviter les lettres accentuées mais les chiffres sont autorisés.

Remarque : majuscules $\neq$ minuscules. La variable `prix` n'est pas la même que la variable `Prix`.

Remarque : Python permet d'affecter plusieurs variables à la fois. $a, b = 1, "bonjour"$ met directement 1 dans la variable a et "*bonjour*" dans la variable b. Donc pour échanger les valeurs de a et b, il suffit de faire $a, b = b, a$.

Fonctions utiles pour les variables

- `type(♡)` donne le type de la variable ♡
- `print(♡)` affiche le contenu de la variable ♡.

2.2. Types de nombres

Les nombres peuvent être de différents types :
- `int` entiers relatifs (entiers négatifs, entiers positifs et 0).
- `float` flottant (nombres décimaux). Ils s'écrivent avec un point à la place de la virgule.
- `complex` Nombres complexes de la forme $a + bj$. La partie imaginaire se note avec un j(remplace le i), a et b sont des float. Il est obligatoire de mettre un nombre devant le j, même 0 ou 1.

Exemple. -6.9 est un float. -69 est un int. 3 est un int, mais 3.0 est un float. 2+5.1j est un complex. -9.7 + 0j aussi

Conversion de type Il est possible de convertir quelque chose en un type de nombre.

- `int(♡)` converti ♡ en entier
- `float(♡)` converti ♡ en nombre flottant.
- `complex(♡, ♠)` crée le nombre complexe $♡ + i♠$

Exemples. int(3.6) → 3, float(5) → 5.0, complex(2.3)→ 2.3 + 0j, complex(2.3, 6)→ 2.3 + 6.0j

Remarque : Les trop petits nombres ($< 10^{-323}$) deviennent 0. Les nombres trop grands ($> 10^{308}$) renvoient une erreur.

Opérations

- `+` addition : $3 + 4 \to 7$
- `-` soustraction : $5 - 6.7 \to -1.7$
- `*` multiplication : $12 * (-4)$
- `/` division avec résultat à virgule : $81/10 \to 8.1$
- `//` quotient de la division euclidienne $81//10 \to 8$
- `%` reste de la division euclidienne $81\%10 \to 1$
- `**` Puissance : $2 * *3 \to 2^3$, on peut aussi utiliser pow(2,3)

Remarque :

- Les variables de type entier et de type flottant sont compatibles pour les opérations arithmétiques. Mais une opération entre entiers et flottants donnera un résultat flottant.
- Raccourci d'opérations : on change la valeur d'une variable x par des opérations simples. `x += a` correspond à l'opération $x = x + a$, x a été remplacée par $x + a$. ça marche aussi avec $- =$, $* =$, $/ =$, $** =$, $\% =$.

Pour faire d'autres fonctions, il faut charger un package mathématique. On le fait une fois en début de programme par la commande `from math import *`
On peut alors utiliser (entre autre) :

— la racine carrée sqrt(♡)
— les nombres π pi et e e
— la fonction exponentielle exp(♡)
— la fonction ln log(♡)
— les fonctions trigonométriques cos(♡), sin(♡), tan(♡)

imprécision avec les float

```
x=0.1
y=x+x
print("y est-il égal à
0.2 ?" , y==0.2)
z=x+x+x
print("z est-il égal à
0.3 ?" , z==0.3)
print("z=", z)
```

y est-il égal à 0.2 ? True
z est-il égal à 0.3 ? False
z= 0.30000000000000004

Les float en Python sont stockés d'une manière "non exacte" en mémoire, et Python affiche une valeur approchée.

Exemple. 0.1 est stockée en python avec une valeur de

$$0.1000000000000000055511151231257827021181583404541015625$$

il faut s'en souvenir si on veut faire des test d'égalité avec des float (ça ne marche pas !). Pour éviter ça, on peut utiliser le mode fraction from fractions import Fraction

— Fraction(♡,♠) permet d'utiliser la fraction $\frac{♡}{♠}$ de manière exacte.
— On peut convertir un float ♠ en fraction avec Fraction(♠) mais ça se base sur la valeur (approchée) du float ! Donc écrire Fraction('♠') est préférable, les

guillemets font que Python considère que ♠ est une chaine de caractère (donc valeur exacte).

On a aussi le mode décimal, plus précis mais plus gourmand en mémoire `from decimal import Decimal`.

Si on met Decimal(♠), on retrouve l'imprécision, il vaut mieux utiliser `Decimal('♠')`.

2.3. Chaine de caractères

Définition 5.

Une chaine de caractère, type string `str`, s'écrit entre des "guillemets " (ou des 'guillemets').

Exemple. "Bonjour" est un string, "landfql r gfmk " aussi. "3" aussi

Opérations

- la commande `\n` dans une chaine de caractère produit un retour à la ligne
- `+` concatène (accole) deux chaines de caractère : "chaine1"+ "chaine2" donne "chaine1chaine2"
- `♡[n]` donne le caractère numéro n de la chaine de caractère ♡, en comptant à partir de 0.
- `♡[n :m]` extrait de ♡ la sous-chaine allant du caractère numéro n(inclus) au caractère numéro m (exclus), en comptant à partir de 0.
- `print(♡)` affiche la chaine de caractère ♡
- `len(♡)` donne la longueur de la chaine de caractère ♡
- `str(♡)` convertit ♡ en chaine de caractère (à récupérer dans une variable).

Exemple. "Bonjour" + "le monde" donnera "Bonjourle monde", "Bonjourle monde"[4] donne la lettre o.

2.4. Booléen et Condition

Définition 6.

Un booleen bool ne peut prendre que deux valeurs :
True (vrai) ou False (faux)

Par défaut, la majorité des objets Python sont considérés comme Vrai.

- Les chaines de caractères non vides sont True. Une chaine de caractère vide est False.
- Un nombre non nul est True. Le nombre 0 est False.
- Une liste non vide est True. Une liste vide est False.
- La valeur None est False

Les booléens servent à écrire des conditions et des tests.

Définition 7.

Une condition est une opération qui renvoie un booléen True ou False.

Comparaisons

1. == (est égal) renvoie True si les deux valeurs sont égales, et False sinon
2. != (est différent) renvoie True si les deux valeurs ne sont pas égales, et False sinon.
3. > (est strictement supérieur) renvoie True si la première valeur est strictement plus grande que la deuxième, et False sinon.
4. >= (est supérieur ou égal) renvoie True si la première valeur est plus grande ou égale à la deuxième, et False sinon.
5. < (est strictement inférieur) renvoie True si la première valeur est strictement plus petite que la deuxième, et False sinon.

6. `<=` (est inférieur ou égal) renvoie True si la première valeur est plus petite ou égale à la deuxième, et False sinon.

Exemple. Pour $x = 3$ et $y = 7$: $x == 2 \rightarrow$ False, $y! = 5 \rightarrow$ True, $x > 1 \rightarrow$ True, $y <= x \rightarrow$ False

Opérations de booléens Si ($Condition1$) et ($Condition2$) sont deux conditions :

- ($Condition1$) `and` ($Condition2$) est True si les deux conditions sont vraies, et False sinon
- ($Condition1$) `or` ($Condition2$) est True si une des deux conditions est vraie, et False sinon
- `not`($Condition1$) est True si ($Condition1$) est False. (not est la négation). Et inversement.

Exemple. Pour $x = 3$ et $y = 7$,

- $(x == 3)$ and $(y == 7) \rightarrow$ True
- $(x > 1)$ and $(y > 10) \rightarrow$ False
- $(x > 1)$ or $(y > 10) \rightarrow$ True
- not$(x == 1) \rightarrow$ True.

3. Instructions conditionnelles

Définition 8.

Une instruction conditionnelle est une instruction qui dépend d'une condition. Certaines parties du code sont exécutées ou ignorées selon que la condition est vérifiée ou non

if *Une condition* :

> instructions à faire si la condition est True
> avec indentation obligatoire

else :

> instructions à faire si la condition est False
> Facultatif, mais indentation obligatoire

Exemple. On veut créer un suite d'instruction qui teste si un nombre A est pair, puis qui calcule $A/2$ si c'est le cas et $A+1$ si A est impaire.

Pour savoir si A est pair, on teste si le reste de sa division euclidienne par 2 `A % 2` est nul.

Python

```
A=
if (A% 2)== 0 :
  A=A/2
else :
  A= A+1
print(A)
```

Si on rentre $A = 27$ au début, le programme affiche 28. Si on rentre $A = 26$ au début,

le programme affiche 13.0.

Remarque : Si... alors... sinon... permet de séparer deux cas. Si on veut faire plus de cas, on peut imbriquer plusieurs **Si** (ou utiliser d'autres structures).

Exemple. Une suite d'instruction qui compare deux nombres x et y, qui stocke dans une variable max le plus grand des deux si il sont différents, et qui affiche un message si il sont égaux. Il y a trois cas : $x < y$, $x = y$ et $x > y$.

Python

```
if (x==y):
  print("Les valeurs sont égales")
else :
  if (x>y):
    max= x
  else :
    max= y
```

Si on teste avec $x = 6$ et $y = 2 * 3$, on voit "Les valeurs sont égales". Si on teste avec $x = 95$ et $y = 76$, alors max prend la valeur 95

Plutôt que d'utiliser des si-alors-sinon imbriqués, on peut utiliser elif en Python (raccourci de else if)

Python

```
if (x==y):
  print("Les valeurs sont égales")
elif (x>y) :
  max= x
else :
  max= y
```

4. Fonctions

Définition 9.

Une fonction est une suite d'instructions qui définissent un sous-programme et qui renvoient un résultat pouvant être utilisé autant de fois que nécessaire dans un programme plus général.

Une fonction peut avoir besoin de paramètres : des valeurs à entrer dans la fonction (pas obligatoirement). Elle peut renvoyer un résultat (pas obligatoirement).

Exemple. print() et sqrt() sont des fonctions.

- print() est une fonction qui nécessite un ou plusieurs arguments : print ("Hello"), print(a, b, c). Elle affiche quelque chose à l'écran, mais elle ne renvoie pas de résultat.
- sqrt() est une fonction qui nécessite un argument : sqrt(9). Elle renvoie un résultat sqrt(9) $\rightarrow$ 3.

Créer une fonction en Python

def *NomDeLaFonction* (parametre1, parametre2, ...) :

```
    instructions de la fonction
    avec un décalage vertical (indentation) obligatoire
    tout ce qui est décalé est dans la fonction
    return résultat
```

Remarque :

- return n'est pas obligatoire. Si il y est, la fonction s'arrête à cet endroit et renvoie le résultat. Si il n'y est pas, la fonction s'arrêtera à la fin des instructions

et renvoie None

— (parametre1, parametre 2, ...) sont les arguments de la fonction. S'il n'y a pas d'arguments, on écrit NomDeLaFonction()

Utiliser une fonction en Python On a défini la fonction... mais on ne l'a pas exécutée ! ! Elle ne fait rien.

— NomDeLaFonction (valeur1, valeur2 ...) va appeler la fonction en remplaçant les paramètres par les valeurs données et effectuant les instructions.

— Pour récupérer le résultat, il faut le stocker dans une variable : *MaVariable* = NomDeLaFonction (valeur1, valeur2 ...).

Exemple. Calculer le périmètre d'un carré de coté 3, de coté 4, de coté 5,6 et de coté 13,1.

Python

```
def  perimetreCarre(cote):
  perimetre=4*cote
  return perimetre
print("le perimetre pour 3 est ", perimetreCarre(3))
print("le perimetre pour 4 est ", perimetreCarre(4))
print("le perimetre pour 5,6 est ", perimetreCarre(5.6))
print("le perimetre pour 13,1 est ", perimetreCarre(13.1))
```

Exemple. Définir la fonction $f(x) = 3x^2 + 2x - 4$ et récupérer dans une variable $f(-1)$ et $f \circ f(2)$.

Python

```
def f(x):
  return (3*x**2+2*x -4)
a=f(-1)
b= f(f(2))
```

print() ≠ return () print(Truc) se contente d'afficher Truc à l'écran, mais Truc n'est sauvegardé en mémoire nulle part et on ne peut pas l'utiliser. return(Truc) ne se trouve qu'à l'intérieur d'une fonction. ce que renvoie la fonction quand elle est appelée. Par exemple, perimetreCarre(5) peut être utilisé dans un calcul par la suite.

Exemple. Une fonction sans return

Python

```
def Rien():
  a=2+4
  b=a*3
  print("il ne se passe rien")

Rien()  # donne "il ne se passe rien"
print(Rien())  # donne "il ne se passe rien" puis None
```

4.1. TP Maîtriser les fonctions en Python

1) Définir des fonctions Dans un programme Python, commencer par la commande print("Debut"), puis définir les fonctions suivantes les unes après les autres :

— une fonction toto1 sans argument et qui affiche "Mon enfant, ma soeur"

— une fonction toto2 sans argument et qui renvoie "Songe à la douceur"

— une fonction toto3 recevant un argument n et qui affiche le résultat de $3 \times n$
— une fonction toto4 recevant un argument n et qui renvoie le résultat de $4 \times n$

Finir le programme avec la commande `print("Fin")`

Exécuter le programme. Que se passe-t-il ? Les fonctions ont-elles été exécutées ?

2) Exécuter des fonctions A chaque question, vous devrez insérer juste avant `print("Fin")` la ligne de commande indiquée. Exécuter le programme. Noter et interpréter le résultat. Puis effacer (ou passer en commentaire) la ligne insérée avant de passer à la question suivante.

1. `toto1`
2. `print(toto1)`
3. `toto1()`
4. `print(toto1())`
5. `toto2()`
6. `print(toto2())`
7. `toto3(n)`
8. `toto3(5)`
9. `print(toto3(5))`
10. `toto4(5)`
11. `print(toto4(5))`

Expliquer les différences de comportement des quatre fonctions.

4.2. TP variables et fonctions

Rentrer exactement le code suivant dans un programme Python.

Python

```
print("Debut")
a=1
b=10
c=100
#1

def bac(c):
    a=2
    d=a*c+b
    #5
    return d


#2
bac(c)
#3
d=bac(10)+ 2
#4
print("Fin")
```

Exécuter le code pour vérifier qu'il n'y a pas d'erreur. Seuls les messages "Début" et "Fin" doivent s'afficher.
Pour comprendre ce qui se passe, on va insérer une des deux commandes d'affichage suivantes à divers endroit, en remplaçant la lettre n) par le numéro de la question

Python

```
print(f"Question n) a={a}, b={b}, c={c}, d={d}")
# commande 1
print(f"Question n) a={a}, b={b}, c={c}")   # commande 2
```

Pour chacune des questions suivantes, entrer la commande à l'endroit indiqué.

1. remplacer #1 par la commande 2. Executer et observer.
2. remplacer #2 par la commande 1. Executer et observer. Que faut-il supprimer dans le message ? Corriger, executer et observer.
3. remplacer #3 par la commande 2. Executer et observer.
4. remplacer #4 par la commande 1. Executer et observer.
5. remplacer #5 par par la commande 1. Executer et observer.

Que pouvez-vous observer ? En déduire de bonnes pratiques avec les variables pour éviter les confusions.

5. Listes

5.1. Définition de liste

Définition 10.

Une liste (type list en Python) est une collection ordonnée d'objets que l'on groupe dans une seule variable. Les objets sont séparés par des virgules et l'ensemble est enfermé dans des crochets.

♡ =[objet0 , objet1 , objet 2 , ... , objetN]

On peut créer une liste vide en faisant ♡ =[ ]

Exemple. maListe=["a", "b", "c", 5, 8.3, "d"]

Extraire/modifier un élément d'une liste ♡ [n] donne l'élément numéro n de la liste ♡. Attention la numérotation des objets commence à zéro ! ♡[0] donne le premier objet de la liste ♡, ♡[1] donne le deuxième objet ... ♡[-1] donne le dernier objet ! On peut compter les objets dans la liste en partant de la fin.
♡[n] =♠ met ♠ en position n dans la liste ♡ (en effaçant ce qui s'y trouvait).

Exemple. maListe[1] renvoie "b", mais maListe[6] renvoie une erreur !
Si on fait maListe[4]=7, la liste devient maListe=["a", "b", "c", 5, 7, "d"]

Extraire/modifier une partie d'une liste La commande ♡[n :m] va créer une liste contenant les objets du numéro n (inclus) au numéro m (exclus).
On peut modifier les éléments du numéro n (inclus) au numéro m (exclus) de la liste en une seule opération : ♡[n :m]= la liste des nouveaux éléments

Exemple. maListe[1 :3] crée la liste ["b", "c", 5].

maListe[1 :3]= [4, 'u', 0] modifie la liste maListe=["a", 4, 'u', 0, 8.3, "d"]

Faire une liste à partir d'une chaine de caractère la commande list(♡) avec ♡ une chaine de caractère va créer une liste de tous les caractères de la chaine de caractère. Cette fonction liste peut aussi convertir en liste un objet de type *range*.

Exemple. list("bonjour) crée la liste ['b','o','n','j',o","u','r']

5.2. Opérations avec des listes

- in permet de tester si un objet appartient à une liste : ♠ in ♡ renvoie True si ♠ est dans la liste ♡ et False sinon.
- + permet de concaténer (coller) deux listes

Exemple. $[1, 2, 3] + [6, 5, 4]$ donne $[1, 2, 3, 6, 5, 4]$.

Python

```
ListeDeNombre=[2,3,6,9]
3 in ListeDeNombre       # donne True
7 in ListeDeNombre     # donne False
```

Fonctions Ces fonctions ne modifient pas la liste, on met la liste en paramètre de la fonction :

- len(♡) donne le nombre d'élément de la liste ♡
- min(♡) donne le plus petit élément de la liste ♡
- max(♡) donne le plus grand élément de la liste ♡

Exemple.

Python

```
UneListe=[3,2,5,7,5,4]
len(UneListe)      # donne 6
min(UneListe)      # donne 2
max(UneListe)      # donne  7
```

Méthodes C'est un peu différent d'une fonction et ça modifie la liste. On met le nom de la liste, suivi d'un point, puis le nom de la méthode suivi de parenthèses (contenant des paramètres supplémentaires éventuellement)

- ♡.index(x) donne la première position de x dans la liste ♡ (en commençant à compter à 0)
- ♡.append(x) rajoute x à la fin de ♡
- ♡.insert(n,x) insère x en position n dans la liste ♡ (en décalant ce qui est après)
- ♡.remove(x) supprime la première apparition de x dans ♡
- ♠= ♡.pop(n) retire l'objet en position n de la liste ♡ et le stocke dans ♠.
- ♡.sort() trie les objets dans l'ordre croissant dans la liste ♡.
- ♡.reverse() inverse l'ordre des éléments dans la liste ♡

Exemple.

Python

```
ListeDeNombre=[2,3,6,9]
ListeDeNombre.index(3)    # donne  1
ListeDeNombre.append(7) # modifie [2,3,6,9,7]
a= ListeDeNombre.pop(2) # modifie [2,3,9,7] et donne a=6
```

Attention Une liste est modifiable. Il faut être vigilant si on modifie une liste en cours de programme. Si on veut une liste non modifiable, il faut utiliser un tuple.

Attention (bis) Dupliquer une liste.

Python

```
liste1= [1,2,3]
liste2=liste1
liste2[0]=7
```

On obtient liste2 = [7,2,3], mais aussi liste1 = [7,2,3] ! !
En fait, la liste n'a pas été dupliquée, c'est juste que la variable liste2 fait référence au même objet que la variable liste1. Et cet objet est unique. Si on veut dupliquer une liste et la modifier indépendamment de l'original, il faut utiliser des procédés spéciaux de copie : `liste2=list(liste1)` ou `liste2=liste1[ :]` (crée une liste extraite de liste1 en prenant tous les éléments).

6. Boucles bornées

Définition 11.

Dans un algorithme, une boucle est une suite d'instructions qu'on répète en boucle un certain nombre de fois.

Définition 12.

on dit que la boucle est bornée lorsqu'on sait exactement combien de fois la boucle va se répéter.

for *une Variable* in *Liste De Valeur* :

> instructions à faire avec la variable
> indentation obligatoire
> Toutes les valeurs de la *Liste De Valeur* sont prises l'une après l'autre

Liste De Valeur peut-être :

- une liste de nombre, ou de tout autre objet.
- un string : la variable va parcourir les caractères.
- range(n) avec $n \in \mathbb{N}^*$. correspond à la liste des entiers de 0 à $n-1$
- range(k,n) avec $k, n \in \mathbb{N}$ correspond à la liste des entiers de k à $n-1$.
- range(k,n,p) avec $k, n, p \in \mathbb{N}$ correspond à la liste des entiers de k à $n-1$ avec un pas de p (c'est à dire un entier sur p)

Exemple.

Python

```python
for i in range(10) :
  print(i)
  # affiche la liste des entiers de 0 à 9
for i in ["a", "e", "r"]:
  print(i)
  # affiche successivement a, e et r
```

7. Boucle non bornée

Définition 13.

On dit qu'une boucle est non bornée lorsqu'on ne sait pas à l'avance le nombre d'itérations de la boucle nécessaire. Un test d'arrêt est effectué à chaque passage de la boucle : si la condition est True (vraie), on continue. Si la condition est False (faux), on s'arrête.

---

while *une Condition* :

    instructions à faire tant que *une Condition* est vraie avec indentation obligatoire

Danger ! le test d'arrêt est obligatoire et il faut s'assurer que la condition d'arrêt soit vérifiée à un moment. Sinon ça crée une boucle infinie qui fait planter le programme.

Remarque : Certaines instructions peuvent modifier le comportement d'une boucle. break provoque l'arrêt immédiat de la boucle. continue permet de sauter immédiatement à la phase de test en ignorant ce qui suit. Il est évident qu'il faut réserver ces instructions à des cas spécifiques encadrés par une structure if/else.

Itérations et boucle Une manière d'utiliser *while* est de l'utiliser avec un compteur qu'on augmente (ou diminue) à chaque tour de la boucle. On peut utiliser la valeur de ce compteur pour le test d'arrêt.

Exemple. Le code suivant affiche tous les entiers de 0 à 99 (on aurait pu utiliser aussi une boucle *for*)

Python

```
n=0
while (n<=100) :
  print(n)
  n=n+1
```

Exemple. On cherche une valeur approchée à 10^{-2} de $x = \sqrt{13}$, c'est-à-dire x tel que $x^2 = 13$. On commence avec une valeur de $x = 3$ car $x^2 = 9$ est proche de 13 et on teste tous les nombres de 0,01 en 0,01 jusqu'à trouver celui dont le carré dépasse tout juste 13.

Python

```
x=3
while (x**2 <13) :
  x=x+0.01
print(x)     # résultat 3.609999999999987
```

Remarque : On ne peut pas mettre comme condition d'arrêt que $x^2 = 13$ car il n'y a aucune chance de tomber pile dessus !

8. Des algorithmes classiques

8.1. TP dichotomie

On recherche un objet x dans un ensemble ordonné. Par exemple : chercher un mot dans le dictionnaire, chercher un nombre particulier dans un intervalle. Plutôt que de parcourir l'ensemble ordonné de plus petit au plus grand à la recherche de x, on procède par dichotomie : on coupe en deux l'ensemble. On regarde si x est dans la première moitié

ou la deuxième et on garde l'ensemble correspondant. Puis on re-coupe en deux... etc. Diviser par deux la taille de l'ensemble dans lequel on cherche permet d'aller vite : Si on fait n étapes, on divise la taille de l'ensemble par 2^n. Par exemple, 10 étapes divise l'ensemble par 1024.

On considère la fonction réelle

$$f(x) = x^3 - 7x^2 + 2x + 5$$

Partie 1

1. Programmer la fonction f en Python de manière à calculer facilement ses valeurs.
2. Dresser le tableau de variation de f et en déduire le nombre de racines réelles de f. On pourra s'aider de Python pour calculer des valeurs approchées.
3. Afficher avec Python la liste des $f(i)$ pour i entier entre -7 et 7. En déduire un encadrement par des entiers de chacune des racines de f.

Partie 2 On note r la racine de f se situant entre 1 et 2. On veut donner une valeur approchée à 10^{-3} par défaut de r en utilisant un algorithme de dichotomie.

Dichotomie Si f s'annule entre a et b, on pose m le milieu de a et b et on calcule $f(m)$.

- Si la valeur d'annulation est entre a et m, on recommence avec $a = b$ et $b = m$.
- Sinon on recommence avec $a = m$ et $b = b$

On a donc coupé la longueur de l'intervalle en deux. On continue jusqu'à ce que la longueur de l'intervalle passe en-dessous de 10^{-3}. Alors a (ou b) est une valeur approchée de r à 10^{-3} par défaut.

On donne l'algorithme suivant

```
a=1
b=2
Tant que b − a > 10^−3 faire :
    m = (a+b)/2
```

```
    Si f(m)*f(a)<0, alors :
      b=m
    sinon
      a=m
afficher "la racine est"  a
```

1. Tester à la main la première exécution de la boucle en donnant les valeurs prises successivement par a, b, m.
2. A quoi sert la condition $f(m) * f(a) < 0$? Et la condition $b - a > 10^{-3}$?
3. Programmer l'algorithme en python et donner la valeur de r. Vérifier en calculant f(r).
4. Comment modifier l'algorithme pour obtenir à la fin le nombre d'étape du calcul ? Que pensez-vous du nombre d'étapes ?

Partie 3 Rajouter ces commandes à votre algorithme et executez-le.

Python

```
import numpy as np
from matplotlib import pyplot
xValue=np.arange(-1, 7, 0.1)
yValue=f(xValue)
pyplot.plot(xValue,yValue,'b-')
pyplot.plot([-1,7],[0,0],'-')
```

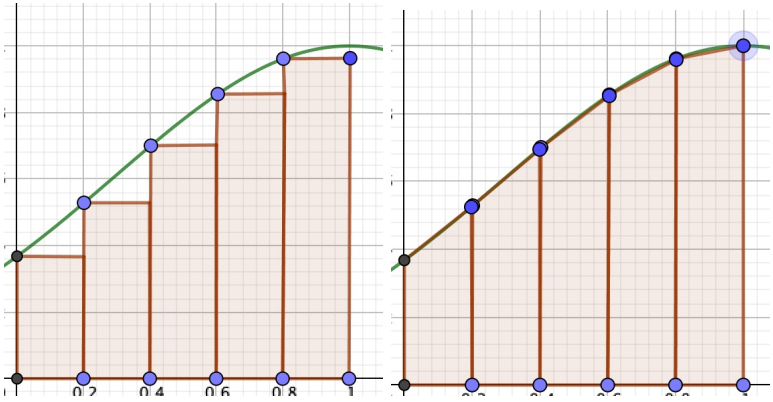
Si rien ne s'affiche, rajouter à la fin `pyplot.show()`.

1. Qu'est-ce qui est représenté ?
2. Dans le code, remplacer les valeurs -1 et 7 par 1 et 2, qu'est-ce qu'on obtient ?

8.2. TP Calcul d'intégrale

On veut calculer une valeur approchée de l'intégrale d'une fonction (aire sous la courbe) sur un intervalle $[a, b]$. On peut utiliser deux méthodes, la méthode des rectangles et

la méthodes des trapèzes, qui reposent sur le même principe : on découpe l'intervalle $[a, b]$ en n portions égales. Sur chaque portion, on approche l'aire sous la courbe par un rectangle ou un trapèze. Puis on additionne tout.



On considère la fonction f, qu'on veut intégrer entre a et b.

Partie 1 On coupe l'intervalle en n intervalles égaux de longueur $\frac{(b-a)}{n}$ et de forme $\left[a + k\frac{(b-a)}{n}, a + (k+1)\frac{(b-a)}{n}\right]$ avec k entier variant de 0 à n-1.

1. Sur l'intervalle $\left[a + k\frac{(b-a)}{n}, a + (k+1)\frac{(b-a)}{n}\right]$, donner l'aire du rectangle sous la courbe (figure 1).
2. Sur l'intervalle $\left[a + k\frac{(b-a)}{n}, a + (k+1)\frac{(b-a)}{n}\right]$, donner l'aire du trapèze sous la courbe (figure 2).

L'aire totale est constituée de l'addition des aires de trapèze ou de rectangle. Pour additionner ces aires, on utilise une boucle for qui va les ajouter les unes après les autres à partir de la gauche.

Partie 2 : Le code python On suppose qu'on a déjà programmé la fonction f, et les valeurs de a et b. La fonction suivante calcule l'aire par la méthode des rectangles et des trapèzes et affiche les deux résultats, avec paramètres n (nombre de rectangle ou trapèze), a, b, f

Python

```
def rectangletrapeze(n,a,b,f) :
  ban= (b-a)/n # largeur rectangle/trapeze
  rectangle=0 #aire de l'addition des rectangle
  trapeze=0 #aire de l'addition des trapèzes
  x1=a    # abscisse coin gauche rectangle/trapèze.
  # remarque x1+ban= coin droit
  for k in range(0,n):
    #f(x1)  ordonnée coin gauche/droite rectangle
    rectangle= rectangle + ban*f(x1)
    #f(x1)  ordonnée coin gauche trapeze
    #et f(x1+ban) ordonnée coin droit
    trapeze= trapeze+ban*(f(x1) + f(x1+ban))/2
    x1=x1+ban #décalage au rectangle suivant.
  print( "L'aire par rectangle est", rectangle)
  print( "L'aire par trapèze est", trapeze)
  return [rectangle, trapeze]
```

1. Programmer en python cette fonction.
2. Calculer à la main $\int_0^1 x^2 dx$. Calculer avec python l'aire correspondante par rectangletrapeze en choisissant $n = 10$, $n = 100$, $n = 1000$. Estimer l'erreur correspondante à chaque fois.
3. Calculer à la main $\int_0^1 4x + 5dx$. Calculer avec python l'aire correspondante par rectangletrapeze $n = 10$, $n = 100$, $n = 1000$. Estimer l'erreur correspondante à chaque fois.
4. Calculer avec python l'aire de $\int_0^1 e^{-x^2} dx$ par rectangletrapeze avec $n = 1000$. Peut-on faire cette intégrale à la main ?

9. TD 35 Algorithmes et Python

Exercice 1

Ouvrir un programme permettant de faire du Python. Identifier la console et l'éditeur. Observer ce qui se passe dans les cas suivants.

1. Dans la console, taper `1+2` et exécuter.
2. Dans la console, taper `print(1+2)` et exécuter.
3. Dans la console, taper `print(Bonjour)` et exécuter.
4. Dans la console, taper `print('Bonjour')` et exécuter.

Exercice 2

Dans la console Python, taper les commandes suivantes, en validant après chaque ligne pour exécuter. Observer.

Python

```
1+2
print(1+2)
x=1+2
print(x)
print("x est égal à ",x)
```

Exercice 3

Dans la console Python, taper les commandes suivantes, en validant après chaque ligne pour exécuter. Observer.

Python

```
y=5
print("y est égal à ", y)
y=2*y
print("y est égal à ", y)
z=2**y
print("z est égal à ", z)
print(y==z)
```

Exercice 4

Dans la console Python, taper les commandes suivantes, en validant après chaque ligne pour exécuter, rentrer votre nom au moment où on vous le demande. Observer.

Python

```
nom=input("Entrer votre nom")
print("Bonjour, ", nom, "comment vas-tu ?")
```

Exercice 5

Dans la console Python, taper les commandes suivantes, en validant après chaque ligne pour exécuter, rentrer le chiffre 23 au moment où on vous demande un nombre. Observer.

Python

```
nombre=input("Entrer un nombre ")
print(nombre)
print(nombre+nombre)
print(3*nombre)
```

Exercice 6

Refaire l'exercice précédent en utilisant l'éditeur et exécuter le programme une seule fois à la fin.

Exercice 7

Jean veut faire un programme inversant le contenu de deux variables. Il propose le programme suivant :

Python

```
a=7
b=2
a=b
b=a
```

1. Expliquer pourquoi le programme ne marche pas.
2. modifier le programme pour que la valeur des variables a et b soient inversées.

Exercice 8

Quelles sont les valeurs finales de a, b, c et d après les instructions suivantes (à la main, puis en Python) ?

$$a = 53; b = 10; c = a/b; d = a\%b; a = a//b; b = b//a$$

Exercice 9

Quelles sont les valeurs finales de a, b, c après les instructions suivantes (à la main, puis en Python) ?

$a = 2; a = a+2; b = a*2+a; c = (b-a)//2; c = c+2*a-b; a = (b-a)*c; b = (a+c)*b$

Exercice 10

Ecrire un programme en Python qui demande de rentrer un prix hors-taxe et qui renvoie alors la valeur du prix TTC, en sachant que la TVA est de 20%. Tester avec les valeurs de prix 10, 9.5 et 231.

Exercice 11

Ecrire un programme en Python qui demande à l'utilisateur de rentrer son année de naissance et qui renvoie son âge (entier) à la fin de l'année. L'année courante sera mise dans une variable.

Exercice 12

Si a et b sont des entiers relatifs non nuls, les expressions $int(a/b)$ et $a//b$ sont-elles égales ?

Exercice 13

Ecrire un programme demandant à l'utilisateur de saisir deux variables entières, et qui affiche leur somme et leur moyenne. Tester avec 12 et 28.

Exercice 14

1. Afficher la valeur de la variable pi dans la console Python.
2. Écrire un algorithme en Python qui calcule et affiche le périmètre d'un cercle de rayon donné et l'aire d'un disque de même rayon.

Exercice 15

Effectuer l'opération suivante :

$$(0,1+0,1+0,1-0,3)\times 10^{20}$$

d'abord à la main, puis en Python en n'utilisant que des float, puis en utilisant le mode fraction, puis le mode décimal. Comparer les résultats.

Exercice 16

Jean veut faire un programme affichant le résultat de l'opération $732+27$. Il propose trois programmes différent :

```
print("732 + 27") #programme 1
print("732" + "27") #programme 2
print(732 + 27) #programme 3
```

1. Quel programme affiche le résultat souhaité ?
2. Donner le résultat affiché par les autres programmes.

Exercice 17

Ecrire un programme qui demande à l'utilisateur d'entrer son nom (simple), puis son prénom (simple), et qui affiche ses initiales. Que se passe-t-il en cas de nom ou prénom composé.

Exercice 18

Ecrire un programme qui demande à l'utilisateur d'entrer une chaine de caractère, et qui affiche la même chaine de caractère encadrée par deux astérisques. La phrase *Bonjour le monde* doit s'afficher $*$*Bonjour le monde*$*$.

Exercice 19

Donner la valeur et le type des variables ci-dessous

Python

```
A = 5.2
B = "Bonjour"
C =6
D = "Choisir un nombre entier"
E =False
F =A+C
G= C - 10
H =(26 + 4*2 - (12 + 6*3))/2
```

Exercice 20

Déterminer la valeur (True ou False) de chacun des tests suivants :

- $(1 < 2)$ and $(6 < 5)$
- $(4 == 2 + 2)$ and $(6! = 7)$
- $(3 <= 2 + 1)$ and $(2 > 3)$
- $(3 <= 2 + 1)$ or $(2 > 3)$
- not $(6 !=7)$

Exercice 21

Ecrire une suite d'instructions en Python :

1. Créer les variable *longueur* et *largeur* (on rentrera manuellement les valeurs plus tard) qui représentent la longueur et la largeur d'un rectangle
2. Ecrire une instruction qui affichera : "c'est un carré" si le rectangle est un carré et "ce n'est pas un carré" sinon
3. Tester avec $longueur = 6$, $largeur = 5$, puis $largeur = 2 * 3$

Exercice 22

Considérons une suite d'instructions en Python

Python

```
if (a-b)>=c*b :
  c=c+1
  a=a-b
else :
  c=c-1
  a=a+b
```

Donner les valeurs de variables a, b, c à la fin de ces instruction (à la main, puis vérification en Python)

1. pour $a = 14, b = 3, c = 3$
2. pour $a = 11, b = 3, c = 4$

Exercice 23

Corriger les erreurs du programme suivant

Python

```
input("Entrer un mot ")
If len(mot)!=6
  print('le mot n'a pas 6 lettres')
  Else :
  print("le mot a 6 lettres")
```

Exercice 24

1. Écrire en Python une fonction aireCarre() avec un argument *cote* qui renvoie l'aire d'un carré de côté *cote*.
2. Calculer l'aire d'un carré de coté $5,7$, de coté $1,1$ et de coté $3,45$

Exercice 25

1. Programmer en Python la fonction g défini sur $\mathbb{R}$ avec $g(x) = -2x^3 + 4x - 7$
2. Afficher l'image des nombres -3 ; 0 ; $\frac{5}{2}$ et $\frac{5}{7}$ par la fonction g.

Exercice 26

1. Ecrire une fonction **EstPaire(nombre)** qui retourne Vrai si le nombre est un multiple de 2 et faux dans le cas contraire.
2. Programmer une fonction **MultipleCinq(Nombre)** qui retourne Vrai si le nombre est un multiple de 5 et faux dans le cas contraire.
3. On considère le programme ci-dessous :

Python

```
def Mystere(Nombre):
  return EstPaire(Nombre) and MultipleCinq(Nombre)
```

Quel est le but de cette fonction ?

Exercice 27

On considère deux points A et B de coordonnées respectives (x_A, y_A) et (x_B, y_B). On recherche la droite passant par ces deux points.

1. Programmer une fonction en Python qui prend argument les coordonnées des deux points, et qui renvoie Vrai si les deux points ont la même abscisses et Faux sinon.
2. Utiliser cette fonction pour créer un programme qui reçoit en argument les coordonnées de deux points et qui renvoie l'équation de la droite (AB) sous la forme $x = k$ ou $y = mx + p$.

Exercice 28

On considère la liste suivante

$$\text{maListe1} = [15,12,22,16]$$

Que renvoie chacune des instructions suivantes ?

- maListe1[2]
- maListe1[4]

Comment la liste est-elle modifié par les instructions suivantes ? (on considérera que chaque instruction est indépendante et que la liste est remise à son état initial entre chaque instruction)

- maListe1.append(11)
- maListe1.remove(16)
- maListe1.sort()

Exercice 29

Compléter la fonction tri suivante qui prend en paramètre deux listes de nombre et renvoie la liste obtenue en rangeant dans l'ordre décroissant la concaténation des deux listes.

Python

```
def tri(A,B) :
  out=.........    #concatenation des listes
  out .........      # tri des listes
  out .........
  return out
```

Exercice 30

Que renvoie la fonction suivante ?

Python

```
def mystery():
  A=[k-7 for k in [8,9,10]]
  A.append(0)
  A.append(22)
  A.reverse()
  A=A+[3,5,11,9]
  A.sort()
  return A
```

Exercice 31

1. Proposer une fonction **mult** prenant en parametre liste1 (une liste de nombre) et k un réel, et qui renvoie une autre liste contenant les éléments de liste1 multiplié par k.
2. Tester avec $liste1 = [2, 4, 7, 1, 5]$ et $k = 3$. On doit obtenir $[6, 12, 21, 3, 15]$

Exercice 32

1. Ecrire un programme de deux lignes qui calcule et affiche $4 \times 0 = 0$, $4 \times 1 = 4$, ..., jusqu'à $4 \times 50 = 200$.
2. Ecrire un programme de deux lignes qui calcule et affiche $4 \times 30 = 120$, $4 \times 31 = 124$, ..., jusqu'à $4 \times 50 = 200$

Exercice 33

On considère la fonction f définie sur $\mathbb{R}$ par $f(x) = x^2$.

1. Définir cette fonction avec Python.
2. Ecrire un programme qui affiche les images des entiers entre -3 et 3 par la fonction f
3. Modifier ce programme pour n'afficher que les images des entiers pairs entre -3 et 3.

Exercice 34

On considère le programme suivant

Python

```
mot="MATHEMATIQUES"
n=0
for i in mot:
  if i=="A":
    n=n+1
```

1. Quel est le type de la variable **mot** ? Et quelle sont les valeurs successives prises par i ?
2. Que fais ce programme ?
3. Ecrire un programme Python qui affiche le nombre de lettre A et de lettre N dans le mot

INTERGOUVERNEMENTALISATIONS

Exercice 35

On considère le programme suivant

Python

```
for i in range(5):
  for j in range(4):
    print(i+j)
```

1. Quelle est la première valeur prise par i et j ? Quelle est la première valeur affichée ?
2. Après l'affichage de la première valeur, quel compteur va changer en premier : i ou j ? pour prendre quelle valeur ? Quelle est alors la deuxième valeur affichée ?

Exercice 36

On considère la chaine de caractère "AEIOUY" représentant les voyelles de l'alphabet français. Ecrire un programme qui compte le nombre de voyelle dans un mot donné en majuscule. Tester avec MATHEMATIQUES et INTERGOUVERNEMENTALISATIONS.

Exercice 37

Ecrire une fonction som(n) qui prend un paramètre n entier strictement positif et qui renvoie la somme de tous les entiers de 1 à n. Tester avec $n = 34$.

Exercice 38

On fait une pyramide de sphères empilées avec une base carrée de taille $n \times n$. On veut connaître $S(n)$ le nombre total de sphères nécessaires.

1. Combien de sphère faut-il si $n = 1$? si $n = 2$?
2. Si on connait $S(n)$, combien de sphère doit-on rajouter pour obtenir $S(n+1)$?
3. Compléter la fonction Python ci-dessous pour qu'elle renvoie $S(n)$.

Python

```
def S(n):
  s=
  for i in range(n+1) :
    s=
  return s
print(som(34))
```

4. vérifier en calculant S(7)

Exercice 39

Python

```
i=0
while i<10:
  print(i)
  i=i+1
print("fin")
```

1. Qu'est-ce qu'affiche ce programme ?
2. Modifier le programme pour qu'il affiche les nombres de 1 à 15
3. Modifier le programme pour qu'il affiche les nombres pairs de 2 à 18
4. Modifier le programme pour qu'il affiche "fin" après chacun des nombre.

Exercice 40

Python

```
n=0
p=17
while n*p<1000 :
  print(n, n*p)
  n=n+1
```

Que fais ce programme ? Le tester.

Exercice 41

La fonction x^3 est continue, strictement croissante et on a $(-3)^3 = -27$ et $(-2)^3 = -8$. Comme $-27 < -20 < -8$, par le théorème des valeurs intermédiaires, il existe $c \in [-3, -2]$ tel que $c^3 = -20$. On veut déterminer une valeur approchée de c à 10^{-2}.
Ecrire un algorithme en python permettant de trouver la valeur approchée de c.

Exercice 42

Jonathan dispose de 10 000€ en banque avec un taux d'intérêt composé à 2%. (Il gagne 2% du montant de l'année précédente).
On suppose que 10 000€ correspond à l'année 0.

1. Calculer à la main le montant en banque lors de l'année 2.
2. Ecrire un programme avec Python qui calcule le montant en banque après la dixième année.
3. Jonathan souhaite avoir au moins 15 000€. Ecrire un programme qui utilise une boucle non bornée pour déterminer le nombre d'année à attendre.

Exercice 43

Un entier naturel est dit premier lorsqu'il n'est divisible que par 1 et lui-même. On veut écrire un programme qui déterminer si un entier naturel est un nombre premier. On définit le programme suivant :

Python

```
def EstPremier(n):
  i=2
  while n%i != 0:
    i=i+1
  return i
```

1. Justifier que la boucle n'est pas infinie.
2. Dans la ligne 2, pourquoi ne pas avoir commencé par $i = 0$ ou $i = 1$?
3. Comment utiliser cette fonction pour déterminer si n est premier ou non ? Ecrire une fonction **testPremier(n)** qui renvoie vraie si n est premier et Faux sinon.

Table des matières

1 algorithme **1**
- 1.1 Algorithme et Python . . . 1
- 1.2 Ecrire un programme en Python . . . 2

2 Les variables **4**
- 2.1 Définition de variable . . . 4
- 2.2 Types de nombres . . . 6
- 2.3 Chaine de caractères . . . 9
- 2.4 Booléen et Condition . . . 10

3 Instructions conditionnelles **12**

4 Fonctions **14**
- 4.1 TP Maîtriser les fonctions en Python . . . 16
- 4.2 TP variables et fonctions . . . 18

5 Listes **20**
- 5.1 Définition de liste . . . 20
- 5.2 Opérations avec des listes . . . 21

6 Boucles bornées **23**

7 Boucle non bornée **25**

8 Des algorithmes classiques **26**
- 8.1 TP dichotomie . . . 26
- 8.2 TP Calcul d'intégrale . . . 28

9 TD 35 Algorithmes et Python **31**