Programmation en langage Python

Pr. K. Doumi

karim.doumi@gmail.com

Introduction

Le langage Python est crée en 1991 par Guido van Rossum (chez Google maintenant)

- Largement utilisé par :
  - Google
  - Yahoo
  - Youtube
  - ...
- Utilisé dans différentes disciplines, en particulier :
  - Big Data
  - Science de données
  - ..

Introduction

Quelques caractéristiques

- Typage dynamique
- Langage de script et orienté objet
- Code moins volumineux : 5 à 10 fois moins de ligne de code que C/C++/Java/...
- Gestion automatique de la mémoire(ramasse-miettes)
- Multiplateformes
- Vitesse de développement
- Erreurs facilement repérables
- Open Source ⇒ Grande communauté
- Riche en terme de bibliothèque spécialisées
- Facile à appréhender

L'ensemble minimal nécessaire :

- Un interpréteur Python (deux versions à ce jour sont 3.5 et 2.7)
- Un éditeur, pour écrire des programmes qui seront exécutés par l'interpréteur (Idle, Pyzo, Spyder, Jupyter, etc)

On travaille en général avec devant soi deux fenêtres principales : l'interpréteur et l'éditeur.

Éléments de base

Opérateur	Opération
+	Addition
-	Soustraction
*	Multiplication
/	Division
**	Puissance
//	Division entière
%	Reste de la division entière
+=	a+=b ➔ a=a+b
-=	a-=b ➔ a=a-b
*=	a*=b ➔ a=a*b
/=	a/=b ➔ a=a/b

Opérateur	Opération
==	Comparaison
>	Supérieur strictement
<	Inférieur strictement
>=	Supérieur ou égal
<=	Inférieur ou égal
!=	Différent
and	Et logique
or	Ou logique
not	Négation logique
=	Affectation

Les opérateurs

```
In [29]: 2+3
Out[29]: 5

In [30]: "ABC"+"DEF"
Out[30]: 'ABCDEF'

In [31]: 2*3
Out[31]: 6

In [32]: 3*'ABC'
Out[32]: 'ABCABCABC'

In [33]: 2**3
Out[33]: 8

In [34]: 10/3
Out[34]: 3.3333333333333335
```

```
In [9]: 10//3
Out[9]: 3

In [10]: 10%3
Out[10]: 1

In [11]: round(10/3,2)
Out[11]: 3.33

In [12]: round(10/3)
Out[12]: 3

In [13]: round(2/3)
Out[13]: 1

In [14]: round(1/3)
Out[14]: 0
```

```
In [30]: a,b=5,6

In [31]: a==b
Out[31]: False

In [32]: a,b
Out[32]: (5, 6)

In [33]: a=b

In [34]: a,b
Out[34]: (6, 6)
```

Les instructions élémentaires

Les instructions élémentaires qui figurent le plus souvent dans tous les algorithmes sont au nombre de trois :

- L'affectation.
- Les instructions d'entrée des données.
- Les instructions de sortie des données.

Les instructions élémentaires

L'affectation

L'opération d'affectation permet d'assigner la valeur d'une expression à un objet.

Syntaxe :

```
objet = expression
```

Exemple :

```
a = 5
b = 6
a , b = 5 , 6
a = a + b
```

Exercice :

Soient x et y deux variables telles que $x = 5$ et $y = 6$.

Donner le script permettant d'échanger leurs contenues.

Les instructions élémentaires

Écriture de données

L'écriture de données permet d'afficher une valeur, une expression ou un message sur l'écran.

Syntaxe :

```
1 print("texte")
2 print(expression)
3 print("texte", expression, "texte", ...)
```

Exemple :

```
1 age, nom = 25, "Python"
2 print("Bonjour")
3 print("Je m'appelle ",nom)
4 print('J\'ai ', age , ' ans')
```

Les instructions élémentaires

Lecture de données

La lecture des données permet d'affecter aux variables, les valeurs lues au clavier. Cette instruction interrompt le programme et attend que l'utilisateur saisisse ce qu'il veut puis appuie sur la touche *Entrée*.

Syntaxe :

```
1 variable = input()
2 variable = input("Message")
```

Attention : Les valeurs retournées par la fonction *input* sont toujours de type *chaine de caractères(texte)*. Ainsi, pour convertir une variable vers un autre type, il faut utiliser le nom du type comme une fonction .

Les instructions élémentaires

Lecture de données

Exécutons les scripts suivants :

Exemple 1 :

```
1 x = input("Entrer le premier nombre :")
2 y = input("Entrer le deuxième nombre :")
3 s = x + y
4 print("La somme est :",s)
```

Exemple 2 :

```
1 x = int(input("Entrer le premier nombre :"))
2 y =int(input("Entrer le deuxième nombre :"))
3 s = x + y
4 print(La somme est :',s)
```

Remarque : La fonction *int()* est remplacée par *float()* dans le cas où le nombre à lire est un réel.

Les instructions conditionnelles

L'instruction if ... : ...

Dans de nombreuses applications, on exige une exécution des parties du programme selon des conditions. C'est, ainsi, le rôle des instructions conditionnelles.

Syntaxe :

```
1 if condition:
2     Bloc_instructions
```

- Si *condition* est vérifiée, le *bloc d'instructions* sera exécuté, sinon le programme continue sans l'exécuter.
- Il est possible de définir plusieurs conditions à remplir avec les opérateurs *and* et *or*

Les instructions conditionnelles

L'instruction if ... : ...

Exemple : Soit le script suivant qui calcule et affiche la valeur absolue d'un entier.

```
1 x = float(input("Entrer un nombre :"))
2 if x < 0 :
3     x = -x
4 print(x)
```

Les instructions conditionnelles

L'instruction if ... : ... else : ...

Syntaxe :

```
1 if condition:
2     Bloc_instructions_1
3 else:
4     Bloc_instruction_2
```

Si la *condition* est vérifiée, c'est le *Bloc_instructions_1* qui sera exécuté, sinon c'est le *Bloc_instructions_2*

Exemple :

```
1 note = float(input())
2 if note < 10 :
3     print("Non  dmis")
4 else :
5     print("Admis")
```

Les instructions conditionnelles

L'instruction if ... : ... else : ...

Exercice :

Écrire un script qui demande un nombre réel, et qui vérifier si ce nombre est, strictement négatif, strictement positif ou nul.

Les instructions conditionnelles

L'instruction if ... : ... elif ... : ... elif ... : ... else : ...

Syntaxe :

```
1 if condition 1:
2     Bloc_instructions_1
3 elif condition 2:
4     Bloc_instruction_2
5 ....
6 elif condition n:
7     Bloc_instruction_n
8 else:
9     Bloc_instruction_n+1
```

Les instructions conditionnelles

L'instruction if ... : ... elif ... : ... elif ... : ... else : ...

Exemple :

```
1 x = float(input())
2 if x < 0 :
3     print(x, "est négatif")
4 elif x > 0 :
5     print(x, "est positif")
6 else:
7     print(x, "est nul")
```

Exercice 1 :

Soit x un nombre entier lu au clavier. Écrire un script qui vérifier si x est un nombre *pair* ou *impair*.

Exercice 2 :

Écrire un script qui lit les coefficients d'une équation $ax+b=0$ et qui affiche les solutions possibles (selon les cas, c'est : (1) $\frac{-b}{a}$, (2) aucune solution ou (3) tout $x \in \mathrm{R}$ est une solution)

Exercice 3 :

Écrire un script de résolution d'une équation du deuxième degré

$$ax^2 + bx + c = 0$$

Les instructions répétitives

La boucle while

Syntaxe :

```
1 while condition:
2     Bloc_instructions
```

Les instructions seront exécutées tant que la condition est vérifiée.

Exemple :

```
1 mdp = input("Entrer le mot de passe :")
2 while mdp != "abc":
3     print("Mot de passe incorrect")
4     mdp = input("Entrer le mot de passe :")
5 print("Bienvenue..")
```

Exercice 1 :

Comme le nombre de tentatives autorisées dans ce cas est illimité, proposer une deuxième version où ce nombre sera limité à trois tentatives.

Exercice 2 :

Donner un script qui demande la valeur de n (un entier) et qui calcule la valeur de la série : $1 + 2 + ... + n$

Exercice 3 :

Donner un script qui demande la valeur de n (un entier) et qui calcule $n! = n * (n - 1) * ... * 1$

Les instructions répétitives

La boucle for

Syntaxe :

```
1 for element in sequence :
2     Bloc_instructions
```

Exemple 1 :

```
1 for k in [2 , 5 , 8 , 3 . 4] :
2     print(k)
```

Exemple 1 :

```
1 for k in "BONJOR" :
2     print(k)
```

Exercice 1 :

En utilisant la boucle *for*, écrire un script calculant la somme : 1 + 2 + 3 + 4 + 5

Question : Comment faire maintenant pour calculer la somme $1+2+...+n$, avec n un entier quelconque à lire au clavier.

$\Rightarrow$ On aura besoin d'une séquence de valeurs 1, 2, 3, ...n que nous ne pouvons pas générer explicitement.

Les instructions répétitives

La fonction range

La fonction *range* permet de générer une séquence de nombres entiers.

Exemples :

- range(10) ⇒ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- range(5,10) ⇒ 5, 6, 7, 8, 9
- range(4,13,2) ⇒ 4, 6, 8, 10, 12
- range(10,5) ⇒ séquence vide
- range(10,5,-1) ⇒ 10, 9, 8, 7, 6

Exemple :

En utilisant la boucle *for*, écrire un script permettant de calculer la somme $1 + 2 + ... + n$, avec n un entier quelconque à demander.

Exercice 1 : On se propose d'écrire un script de test de nombres parfaits. Un nombre est dit parfait s'il est égal à la somme de ses diviseurs, 1 compris.

Exemple :

- $6 = 1 + 2 + 3$, est un nombre parfait.
- $28 = 14 + 7 + 4 + 2 + 1$, est un nombre parfait

Écrire un script qui permet de lire un entier au clavier, et de vérifier s'il est parfait ou pas.

Exercice 2 :

Écrire un script qui vérifier si un nombre lu au clavier est un nombre premier ou non.

Exercice 3 : Écrire un script qui détermine la nième valeur u_n (n étant fourni en donnée) de la suite définie comme suit :

$$\begin{cases} u_0 = 2 \\ u_{n+1} = u_n + 3 \quad si \quad n \geq 0 \end{cases}$$

Exercice 4 : La suite de *Fibonacci* est dénie par :

- $f_0 = 0$
- $f_1 = 1$

$$f_n = f_{n-1} + f_{n-2} \quad si \quad n \geq 2$$

Écrire un script calculant le néme terme (lu au clavier) de cette suite

Exercice 5 :

Écrire un script permettant de calculer la valeur approchée de π en utilisant le développement suivant :

$$\pi = 4 \sum_{k=0}^{+\infty} \frac{(-1)^k}{2k+1}$$

et en s'arrêtant lorsque le terme $\frac{1}{2k+1}$ du développement soit plus petit qu'un réel *epsilon*(de votre choix).

Les fonctions

Les fonctions permettent de regrouper plusieurs instructions dans un bloc qui sera appelé grâce à un nom. Cette programmation dite "modulaire" se justifie pour de multiples raisons :

- Un programme écrit dans un seul bloc devient difficile à comprendre dès qu'il dépasse une ou deux pages de code.
- La programmation modulaire permet d'éviter des séquences d'instructions répétitives.

Les fonctions

Syntaxe :

```
1 def    nom_de_la_fonction(paramètres):
2     Bloc_instructions
```

Si l'on décrit la ligne de définition de la fonction, on trouve :

- *def* : mot-clé qui est l'abréviation de '*define*' (définir, en anglais).
- *Le nom de la fonction* : qui se nomme exactement comme une variable.
- *La liste des paramètres ou arguments* : qui seront fournis lors de l'appel de la fonction.
- *Les deux points* : qui clôturent la ligne.

Les fonctions

Exemples

Exemple 1 :

```
1 def carre(x):
2     return  x*x
```

Exemple 2 :

```
1 def constante():
2       return  5
```

Exemple 3 :

```
1 def saluer(): #Fonction sans paramètres
2     print('bonjour')
```

Chaque argument de la définition de la fonction correspond, dans l'ordre, à un paramètre de l'appel. La correspondance se fait par affectation.

Exemple :

```
1 def fonction(x, y, z):
2     return (x, y, z)
```

Appels :

```
1 f(5, 3, 7) # x=5, y=3 et z=7
2 f(2, "abc", [1,2,3]) # x=2, y="abc" et z=[1,2,3]
```

Les fonctions

return

L'instruction return précise la valeur que fournira la fonction à la fin de son travail. Elle a pour effet d'interrompre le déroulement de la fonction.

Soit la fonction suivante qui vérifier si un nombre est pair ou impair :

Exemple :

```
1 def parite(x):
2     if x%2==0:
3         return True
4     else:
5         return False
```

```
1 def parite(x):
2     if x%2==0:
3         return True
4     return False
```

Les fonctions

Exemple

Exercice :

Un nombre est dit premier si il n'a que deux diviseur 1 et le nombre lui-même.

Autrement dit, il suffit de trouver un autre diviseur entre 2 et $n - 1$ pour conclure que le nombre n'est pas premier.

Écrire la fonction *premier* qui reçoit un entier en paramètres et qui retourne *True* si ce dernier est un nombre premier et *False* sinon.

Les fonctions

Exemple

```
1 def premier(n):
2     for i in range(2,n) :
3         if n%i==0 :
4             return False
5     return True
```

Si un entier *i* devise *n*, la valeur *False* sera retournée ce qui interrompra ainsi la fonction. Sinon c'est la valeur *True* qui est retournée.

Les fonctions

print vs return

Exemple 1 :

```
1 def carre(x):
2     return  x*x
```

Exécution :

```
1 a=carre(5)
2 print(a)
3 m=a+4
4 print(m)
```

Résultat :

25

9

Exemple 2 :

```
1 def carre(x):
2     print(x*x)
```

Exécution :

```
1 a=carre(5)
2 print(a)
3 m=a+4
4 print(m)
```

Résultat :

25

None

Erreur !

Les fonctions

les valeurs par défaut

Il est possible de préciser des valeurs par défaut pour les paramètres de la fonction.

Exemple :

```
1 def f(a=1, b=2, c=3):
2     print ("a =", a, "b =", b, "c =", c)
```

Exécution :

```
1 f() # a=1 b=2 c=3
2 f(6) # a=6 b=2 c=3
3 f(4,5) # a=4 b=5 c=3
4 f(4,5,6) # a=4 b=5 c=6
5 f(b=5) # a=1, b=5, c=3
6 f(c=8, b=4) # a=1 b=4 c=8
```

Les fonctions

Porté des variable

Les noms des objets sont crées lors de leur première affectation, mais ne sont visibles que dans certaines régions de la mémoire.

- **Variables globales** : ils ont une portée qui s'étend sur l'ensemble du programme.
- **Variables locales** : ils ont une portée qui se limite au corps de la fonction où il sons définis.

Les fonctions

Porté des variable

```
1  x = 5
2  def f():
3      x = 7
4  f()
5  print("x=",x)
6  Résultat :
7  x = 5
```

```
1  x = 5
2  def f():
3      x = x + 3
4  f()
5  print(x)
```

Résultat :

Erreur, x est appelé avant affectation

```
1  x = 5
2  def f():
3      x = 7
4      print("x de f =",x)
5  def g():
6      x = 8
7      print("x de g =",x)
8  f()
9  g()
10 print("x global=",x)
11 Résultat :
12 x de f = 7
13 x de g = 8
14 x global = 5
```

Référencement d'une variable global dans une fonction

Si on souhaite faire référence à une variable globale dans une fonction, il suffit de la précéder par le mot clé *global*

```
1 x=5
2 def f():
3     x=7
4 f()
5 print("x=",x)
```

Résultat :

```
1 x = 5
```

```
1 x = 5
2 def f():
3     global x
4     x=7
5 f()
6 print("x=",x)
```

Résultat :

```
1 x = 7
```

Exemple : afficher le nombre d'appel d'une fonction

```
1 n=1
2 def f():
3     print("appel",n)
4     n += 1
5 f()
6 f()
7 f()
```

erreur, n est appelé avant affectation

```
1 def f():
2     n=1
3     print('appel',n)
4     n += 1
5 f()
6 f()
7 f()
```

appel1

appel1

appel1

```
1 n=1
2 def f():
3     global n
4     print('appel',n)
5     n += 1
6 f()
7 f()
8 f()
```

appel1

appel2

appel3

Une fonction est dite récursive si elle est définie par une relation de récurrence. Autrement-dit, il fait appel à elle-même.

Exemple : La fonction factorielle

Définition itérative :

$$fact(n) = \prod_{k=1}^{n} k$$

Définition récursive :

$$fact(n) = \begin{cases} 1 & \text{si } n = 0 \\ n * fact(n-1) & \text{sinon} \end{cases}$$

Exemple : La fonction factorielle

Définition itérative :

```
1  def fact(n):
2      p=1
3      for k in range(1,n+1):
4          p=p*k
5      return  p
```

Définition récursive :

```
1  def fact(n):
2      if n==0:
3          return  1
4      return  n*fact(n-1)
```

Attention :

Les fonctions récursives sont très couteuses en terme d'espace mémoire.

Explication :

Chaque appel de *fact* entraîne une allocation d'espace pour les variables et les paramètres de la fonction. Or chaque nouvel appel de *fact*, "à l'intérieur" de *fact*, provoque une telle allocation, sans que les emplacements précédents soient libérés.

Exercice 1 :

Écrire la fonction définie par :

$$somme(n) = 1 + 2 + 3 + \ldots + n$$

1. En utilisant une fonction itérative.
2. En utilisant une fonction récursive.

Exercice 2 :

Sans utiliser l'opérateur **, Écrire une fonction qui reçoit en arguments un nombre réel et un entier, et qui renvoi le premier à la puissance du 2éme.

1. En utilisant une fonction itérative.
2. En utilisant une fonction récursive.

Exercice 3 :

Écrire une fonction qui vérifie si un entier passé en argument est un nombre premier ou non (return *True* si oui et *False* sinon).

Exercice 4 :

En utilisant la fonction définie dans l'exercice *N°3*, écrire une deuxième fonction qui affiche tous les nombres premiers compris entre deux nombres entiers fournis en argument.

Exercice 5 :

La suite de Fibonacci est dénie par :

$$\begin{cases} f_0 = 0 \\ f_1 = 1 \\ f_n = f_{n-1} + f_{n-2} \qquad si \quad n \geq 2 \end{cases}$$

1. Ecrire une fonction récursive $f(n)$ calculant le terme d'indice n de cette suite.
2. Quel est le nombre d'appels à la fonction f sont-ils faits pour calculer $f(5)$?

Les listes

Définition

C'est une suite de valeurs où chaque élément est repéré par un "indice" précisant sa position au sein de l'ensemble.

Initialisation :

```
1 L=[]
2 L=list()
3 L=[1,4,3,7]
4 L=[4,'ABC',[4,7,5]]
5 L=list(range(2,10))
6 L=list((4,5,6))
7 L=[2*x for x in range(5)]
```

```
1 L=[c for c in 'ABCDE']
2 L=3*[3,4]
3 L=[3,4,5]+[70,80,90]
4 L=[int(input()) for i in range(5)]
```

Les listes

Accès aux éléments d'une liste

```
1 | L=[10,20,30,40,50,60,70]
2 | - L[0] # 10
3 | - L[len(L)] # Erreur
4 | - L[len(L)-1] # 70
5 | - L[-1] # 70
```

Slicing :

```
1 | - L[2:5] # [30, 40, 50]
2 | - L[4:] # [50,60,70]
3 | - L[:4] # [10,20,30,40]
4 | - L[:] # [10,20,30,40,50,60,70]
5 | - L[2:7:2] # [30,50,70]
6 | - L[::2] # [10, 30, 50, 70]
7 | - L[::-1] # [70,60,50,40,30,20,10]
```

Les listes

Deux noms pour un même tableau

```
1 L=[1,2,3]
2 T=L
3 T[0]=5
4 print(L)
5 print(T)
6
7 [5,2,3]
8 [5,2,3]
```

```
1 L=[1,2,3]
2 T=L[:]
3 T[0]=5
4 print(L)
5 print(T)
6
7 [1,2,3]
8 [5,2,3]
```

```
1 L=[1,2,3]
2 T=L.copy()
3 T[0]=5
4 print(L)
5 print(T)
6
7 [1,2,3]
8 [5,2,3]
```

Les listes

Parcours d'une liste

- for indice in range(0, Taille)

```
1 def somme(L):
2     s=0
3     for i in range(0, len(L)):
4         s=s+L[i]
5 return  s
```

- for element in Liste

```
1 def somme(L):
2     s=0
3     for i in L
4         s=s+i
5 return  s
```

Les listes

Parcours d'une liste

- for indice in range(0, Taille)

```
1 L=[10, 20, 30, 40]
2 for i in L:
3     print(i)
4
5 10
6 20
7 30
8 40
```

- for element in Liste

```
1 L=[10, 20, 30, 40]
2 for i in reversed(L):
3     print(i)
4
5 40
6 30
7 20
8 10
```

Les listes

Modifier une liste passée en paramètres

Exemple : Remettre une liste à zéro

```
1 def raz(L):
2     for e in L:
3         e=0
4 L=[1,2,3]
5 raz(L)
6 print(L)
7
8 Résultat:
9 [1, 2, 3]
```

```
1 def raz(L):
2     for i in range(len(L)):
3         L[i]=0
4 L=[1,2,3]
5 raz(L)
6 print(L)
7
8 Résultat:
9 [0, 0, 0]
```

Les listes

Les méthodes et opérateurs

```
1 - x <not> in L # True si x est élément de L et False sinon
2 - L.count(x) # nombre d'occurrences de x
3 - L.index(x) # Indice de la première occurrence de x (erreur si x
      n'est pas dans L)
4 - L.insert(i, x) # insère l'élément x en position i dans la liste
5 - L.remove(x) # Supprime la première occurrence de x (erreur si x
      n'est pas dans L.
6 - L.pop(i) # Retourne et supprime l'élément L[i] de la liste L ou
      le dernier si aucun indice n'est fournit.
7 - L.append(x) # ajouter x à la fin de la liste
8 - L.sort(<reversed=True>): # trier la liste par ordre croissant
      (par defaut)
```

NB : Taper *dir(list)* pour afficher les autres fonctions et *help(fonction)* pour comprendre.

Les listes

Exercices

Exercice 1 :

Écrire une fonction *smul(m,l)* à deux paramètres, un nombre et une liste de nombres, qui multiple chaque élément de la liste par le nombre et renvoie une nouvelle liste.

Exemple : *smul*(2, [1, 2, 3]) renvoie [2, 4, 6].

Exercice 2 :

Écrire une fonction *vsom(u,v)* qui prend en paramètre deux listes de nombres de même longueur et qui renvoie une nouvelle liste constituée de la somme terme à terme de ces deux listes.

Exemple : *vsom*([1, 2, 3], [4, 5, 6]) renvoie [5, 7, 9].

Exercice 3 :

Écrire la fonction *binaire*(*n*) permettant de retourner la représentation binaire, sous forme d'une liste, correspondant au nombre *n* passé en arguements.

- Donner une version itérative.
- Donner une version récursive.

Exercice 4 :

Sans utiliser la fonction prédéfinie *max*, définir la fonction *maximum(l)* permettant de calculer le maximum d'une liste *l*.

- Donner une version itérative.
- Donner une version récursive.

Les listes de listes

Les matrices

On peut choisir de représenter une matrice de dimensions (n, p) par une liste de taille n, dont les éléments sont des listes de taille p.

Exemple : Soit la matrice suivante :

$$\begin{array}{ccc} & & ! \\ 0 & 1 & 2 \\ 3 & 4 & 5 \end{array}$$

peut être définie en Python par le tableau M ci-après :

```
1 M = [[0, 1, 2], [3, 4, 5]]
```

La valeur de l'élément m_{ij} est obtenue avec l'expression $M[i][j]$.

Les listes de listes

Les matrices

```
1 - v=[1,2,3]
2 - L=v+v # ou L=2*v ou M=[v for i in range(3)]
3 - L # [1,2,3,1,2,3]
4 - M=[v]+[v]+[v] # ou M=3*[v] ou M=[v for i in range(3)]
5 - M # [[1,2,3],[1,2,3],[1,2,3]]
6 - M[0][0]=44
7 - M # [[44,2,3],[44,2,3],[44,2,3]]
```

Les listes de listes

Les matrices

Création :

```
1 - M=[[1,2,3]+[1,2,3]+[1,2,3]] # ou M=3*[[1,2,3]] ou M=[[1,2,3]
      for i in range(3)]
2 - M # [[1,2,3],[1,2,3],[1,2,3]]
3 - M[0][0]=4
4 - M # [[4,2,3],[1,2,3],[1,2,3]]
```

Une autre méthode :

```
1 - M=[[i+j for i in range(1,4)] for j in range(1,4)]
2 - [[2, 3, 4], [3, 4, 5], [4, 5, 6]]
```

Exercice 1 : La trace d'une matrice carrée est la somme des valeurs situées dans la diagonale. Écrire la fonction *trace*(A) qui calcule la trace de la matrice A passée en paramètres.

Exercice 2 :

Une matrice $A = (a_{i,j})$ est dite symétrique si $\forall i, j \; a_{i,j} = a_{j,i}$

Écrire la fonction *symetrie*(A) qui retourne *True* si la matrice carrée A passée en arguments est une matrice symétrique et *False* sinon.

Exercice 3 :

1. Écrire une fonction qui calcule et retourne la somme de deux matrices passées en paramètres.
2. Tester votre fonction sur des exemples de votre choix

Exercice 4 :

1. Écrire une fonction qui calcule et retourne la transposé d'une matrice passée en paramètres.
2. Tester votre fonction sur une matrice de votre choix

Les tuples

Définition

Un tuple est une séquence non modifiable de valeurs. C'est une séquence comme les listes, la seule différence, c'est qu'il n'est pas modifiable.

Les tuples

Création

On peut créer un tuple par insertion de valeurs entre deux parenthèses séparées par virgules.

Exemple :

```
1 >>> tuple1 = ('E1234', 'Naciri Karim', 18)
2 >>> tuple2 = ('Rabat', 'Casablanca')
3 >>> tuple3 = (1, 2, 3, 4, 6, 50)
4 >>> tuple4 = (12,) # n'oubliez pas la virgule après la valeur.
5 >>> tuple5= () # tuple vide
```

Les tuples

Accès

Comme toutes les autres séquences, pour accéder aux valeurs d'un tuple, on utilise des crochets :

Exemple :

```
1 >>> T = ('E1234', 'Naciri Karim', 18)
2 >>> T[0]
3 E1234
4 >>> T[1]
5 Naciri Karim
6 >>> T[2]
7 18
```

Les tuples

Slicing dans les tuples

Comme pour toutes les autres séquences, on peut accéder aux sous-tuples d'un tuple en utilisant deux indices séparés par " : "

Exemple :

```
1  >>> T = ('AB',56,'Rabat','Maroc',14.6,55,'CPGE','Informatique')
2  >>> T[2:5]
3  ('Rabat', 'Maroc', 14.6)
4  >>> T[3:]
5  ('Maroc', 14.6, 55, 'CPGE', 'Informatique')
6  >>> T[:3]
7  ('AB', 56, 'Rabat')
8  >>> T[-1:]
9  ('Informatique',)
10 >>> T[-1:3:-1]
11 ('Informatique', 'CPGE', 55, 14.6)
```

Les tuples

Opérations de base

Comme pour les listes et chaines de caractères, on peut également utiliser les opérateurs "+" et "*" sur les tuples :

Exemple :

```
1 >>> len((1,2,3)) # Nombre d'éléments d'un tuple # 3
2 >>> (1,2,3)+(4,5,6) # Concaténation de tuples #(1,2,3,4,5,6)
3 >>> 3*(1,2)
4 (1, 2, 1, 2, 1, 2)
5 >>> 'Rabat' in ('Rabat', 'Agadir', 'Marrakech') # True
6 >>> for x in ('Rabat', 'Agadir', 'Marrakech') : # Parcourir un
      tuple
7                 print(x)
8 Rabat Agadir Marrakech
```

Les ensembles

Définition

Un ensemble en Python est une collection d'objets sans répétition et sans ordre.

- Ce n'est pas une séquence !
- C'est une suite d'éléments séparés par des virgules et délimités par des accolades.
- Les éléments peuvent être de types quelconques.

Exemples :

```
1 >>> E1 = {5,3,-8,2}
2 >>> E2 = {'o','e','y','u','a','i'}
3 >>> E3 = {5,'CPGE',(3,-2),7.4}
4 >>> E4 = set() # Ensemble vide (Attention : {} qui crée un
      dictionnaire)
```

Les ensembles

Création

```
1  >>> E = {5,2,3,1}
2  >>> E
3  {1, 2, 3, 5}
4  >>> {x*x for x in range(20) if x %3 == 0}
5  >>> E
6  {0, 9, 36, 81, 144, 225, 324}
7  >>> et('fabfcdefg')
8  {'a', 'b', 'c', 'd', 'e', 'f', 'g'}
9  >>> set([5,2,5,6,2])
10 {2, 5, 6}
```

Remarque : Les éléments d'un ensemble ne sont pas numérotés. D'où, il est impossible d'utiliser une notation comme *e*[*i*] puisque parler de l'élément numéro *i* n'a pas de sens!

Les ensembles

Méthodes

- **add** : Add an element to a set.
- **clear** : Remove all elements from this set.
- **copy** : Return a shallow copy of a set.
- **difference** : Return the difference of two or more sets as a new set.
- **discard** : Remove an element from a set if it is a member.
- **intersection** : Return the intersection of two sets as a new set.
- **isdisjoint** : Return True if two sets have a null intersection.
- **issubset** : Report whether another set contains this set.
- **issuperset** : Report whether this set contains another set.
- **pop** : Remove and return an arbitrary set element. (Raises KeyError if the set is empty).
- **remove** : Remove an element from a set; it must be a member(If the

Question : Afficher toutes les méthodes des ensembles ainsi que leurs description tout en essayant de comprendre comment fonctionnent tout en les manipulant avec des exemples de votre choix.

Les dictionnaires

Définition

Un dictionnaire est une structure de données qui permet de stocker plusieurs valeurs y compris des tuples, des listes ou même encore d'autres dictionnaires. L'utilité des dictionnaires, c'est qu'ils nous permettent de stocker des valeurs en utilisant nos propres indices, ainsi il suffit de fournir l'indice pour accéder à l'information stockée avec cet indice.

Les dictionnaires

Création

Un dictionnaire, appelé aussi un "*tableau associatif* ", est une séquence de paires "*clé : valeur*" séparées par virgules et mises entre accolades.

Exemple :

```
1 dict1 = {'C451236' : 'Anabih issa', 'E985477' : 'Amal Abbassi' }
2 dict2 = {'a': 1, 'b':2, 'c':3, 'd':4}
3 dict3 = {'name': 'Ansari Imad'}
4 dict4 = {} # dictionnaire vide
```

Les dictionnaires

Accès

Comme toutes les autres séquences pour accéder aux valeurs d'un dictionnaire, on utilise les crochets sauf qu'au lieu d'un indice, on utilise une clé :

Exemple :

```
>>> etudiant = {'CNE': 'E1234', 'Nom': 'Naciri Karim', 'Age': 18}
>>> etudiant['CNE']
'E1234'
>>> etudiant['Nom'] # Récupérer le nom de l'étudiant
'Naciri Karim'
>>> etudiant['Age']
18
```

NB : Une Erreur KeyError est générée si on fourni une clé qui n'existe pas dans le dictionnaire

Les dictionnaires

Les dictionnaires sont modifiables (mutables)

On peut mettre à jour un dictionnaire par ajout de nouvelle valeur, modifier des valeurs existantes ou supprimer des valeurs.

Les dictionnaires

Les dictionnaires sont modifiables (mutables)

- **Ajout de valeurs :**

```
1 >>> etudiant = {'CNE': 'E1234', 'Nom': 'Naciri Karim',
      'Age': 18}
2 >>> etudiant['Ville'] = 'Rabat'
```

⇒ La clé 'Ville' n'existe pas dans le dictionnaire *etudiant* donc elle va être ajoutée avec la valeur *Rabat*.

Les dictionnaires

Les dictionnaires sont modifiables (mutables)

- **Modification de valeurs :**

```
1 >>> etudiant = {'CNE': 'E1234', 'Nom': 'Naciri Karim',
      'Age': 18}
2 >>> etudiant['Age'] = 22
```

⇒ La valeur de la clé *age* va être modifiée en 22.

Les dictionnaires

Les dictionnaires sont modifiables (mutables)

- **Suppression de valeurs :**

```
1 >>> etudiant = {'CNE': 'E1234', 'Nom': 'Naciri Karim',
      'Age': 18}
2 >>> del etudiant['age'] # supprimer la paire de cle age
3 >>> etudiant.clear() # vider le dictionnaire
4 >>> del etudiant # supprimer le dictionnaire tout entier
```

Exemple :

```
>>> etudiant = {'CNE': 'E1234', 'Nom': 'Naciri Karim', 'Age': 18}
>>> len(etudiant) # Nombre d'éléments d'un dictionnaire
3
>>> 'CNE' in etudiant # Teste d'appartenance d'une clé
True
>>> for i in etudiant : # Parcourir les clés du dictionnaire
        print(i)
CNE Nom Age
>>> for i in etudiant : # Parcourir les valeurs du dictionnaire
        print(etudiant[i])
E1234 Naciri Karim 18
```

Les dictionnaires

Manipulation des dictionnaires (méthodes)

Python inclut des méthodes de manipulation des dictionnaires :

Soit le dictionnaire suivant :

```
1 D = {'CNE': 'E1234', 'Nom': 'Naciri Karim', 'Age': 18}
```

Expression	**Description**
D.clear()	Vide le dictionnaire etudiant
D.copy()	Retourne une copie du dictionnaire etudiant
D.fromkeys(keys, values)	Crée et retourne un dictionnaire à partir des listes *keys* et *values*

Les dictionnaires

Manipulation des dictionnaires (méthodes)

Expression	**Description**
D.get(key, default)	Retourne la valeur de la clé *key* si ça existe dans le dictionnaire, si non la fonction retourne la valeur *default*
D.items()	Retourne la séquence de paires *(clé, valeur)*
D.keys()	Retourne la liste des clés du dictionnaire
D.values()	Retourne la liste de valeurs stockées dans le dictionnaire

Les dictionnaires

Exercice

Exercice 1 : Écrire la fonction *toListe(D)* qui reçoit un dictionnaire en paramètres et qui permet de le convertir en liste de listes dont la première colonne sont les clés et la deuxième colonne les valeurs.

Exercice 2 :

Écrire la fonction *frequence(L)* qui reçoit une liste de nombres en paramètres et qui retourne un dictionnaire dont les clés sont les différents nombres de la liste et les valeurs la fréquence de chaque nombre.

Les chaines de caractères

Définition

Une chaîne de caractères est une suite quelconque de caractères délimités soit par des apostrophes (simple quote), soit par des guillemets (double quote) ou par triples guillemets ou de triples apostrophes dans le cas où la chaîne de caractères est sur plusieurs lignes.

Les chaines de caractères

Exemple :

```
1 ch1 = "" # chaine vide
2 ch2 = 'Je suis en classe MPSI3'
3 ch3 = "J'ai une bonne note en informatique"
4 ch4 = '"Python" est langage de programmation'
5 ch5 = """ Trois principales types de séquences existent en python:
6             - Les listes,
7             - Les tuples,
8             - Les chaines """
```

Les chaines de caractères

Exemple

Remarques :

Les chaînes de caractères ont beaucoup de points communs avec les listes, à savoir : la longueur avec *len*, l'accès aux éléments, la concaténation, le slicing, le parcours... etc.

```
1 >>> s ='bonjour'
2 >>> len(s)
3 7 # il y a bien 7 caractères dans la chaîne s
4 >>> s[1]
5 'o' # comme les listes, les indices des chaînes commencent àzéro
6 >>> s + ' UM6P'# remarquer l'espace avant UM6P
7 'bonjour UM6P'
8 >>> s[1:4] # sous chaîne situé entre le caractère 1 et 4 (exclu)
9 onj
```

Les chaines de caractères

Exemple

Une différence importante avec les listes : c'est que les chaînes de caractères sont non modifiables.

```
1 >>> ch='BONJOUR'
2 >>> ch[0]='D' # tentation de modifier 'B' par 'D'
3 TypeError: 'str' object does not support item assignment
```

Les chaines de caractères

Les caractères spéciaux

Le caractère '\' (antislash) permet d'insérer un certain nombre de codes spéciaux dans une chaîne de caractères :

- \n : insérer des sauts de ligne dans une chaîne.
- \' : insérer des apostrophes dans une chaîne délimitée par des apostrophes.
- \" : insérer des guillemets dans une chaîne délimitée par des guillemets.
- \\ : insérer des antislashs dans une chaîne.

On attribue à chaque caractère un nombre appelé code, qui permet de le repérer. La norme d'encodage que l'on utilisera est UTF-8, qui est rétro-compatible avec les codages ASCII et unicode. Par exemple, les lettres majuscules 'A', 'B',..., 'Z' ont pour code 65, 66, . . . , 90.

La fonction ord

La fonction ***ord*** («ordinal») de Python permet d'obtenir le code correspondant à un caractère.

Exemple :

```
1 >>> ord('A')
2 65
3 >>> ord('a')
4 97
5 >>> ord('#')
6 35
```

La fonction chr

Inversement, on peut récupérer le caractère correspondant à un code avec la fonction ***chr***.

Exemple :

```
1 >>> chr(65)
2 A
3 >>> chr(97)
4 a
5 >>> chr(35)
6 #
```

La méthode ***count*** est une méthode prédéfinie en python. Elle permet de compter l'occurence d'un caractère dans une chaîne de caractères.

Exemple :

```
1 >>> s = 'Bonjour python'
2 >>> s.count('o')
3 3
```

La méthode **isupper** est une méthode prédéfinie en python. Elle permet de vérifier si une chaine de caractères est en majuscule.

Exemple :

```
1 >>> s = "J'AI 20 ANS"
2 >>> s.isupper()
3 True
4 >>> ch = "J'ai 20 ANS"
5 >>> ch.isupper()
6 False
```

La méthode **upper** permet de convertir une chaine de caractères en majuscule.

Exemple :

```
1 >>> s="J'AI 20 ANS"
2 >>> s.upper()
3 J'AI 20 ANS
4 >>> s="J'ai 20 ANS"
5 >>> s.upper()
6 J'AI 20 ANS
```

La méthode **split** est aussi une méthode prédéfinie en python. Elle permet de transformes une chaine de caractères en une liste.

Exemple :

```
1 >>> s="Je suis en prepas"
2 >>> s.split()
3 ['Je','suis','en','prepas']
4 >>> s="Je:suis:en:prepas"
5 >>> s.split(':')
6 ['Je','suis','en','prepas']
```

De la même façon pour ***isupper*** *et* ***upper****, deux méthodes supplémentaires permettent d'effectuer l'opération inverse. Autrement-dit, elles permettent de vérifier si une chaine est minuscule ou de convertir une chaîne en minuscule. Il s'agit des méthodes* ***islower*** *et* ***lower****.*

Exercice 1 :

Une chaîne de caractères est dite palindrome si elle se lit de la même façon dans les deux sens (Exemple : 'laval', 'ELLE','ressasser').

Définir la fonction *palindrome* permettant de vérifier si la chaîne passée en argument est un palindrome (Retourne True si oui et False sinon).

Exemple :

```
1 >>> s = 'ELLE'
2 >>> palindrome(s)
3 True
4 >>> s = "J'ai 20 ANS"
5 >>> palindrome(s)
6 False
```

Exercice 2 :

En utilisant la fonctions chr, écrire une fonction *ascii* recevant deux entiers en paramètres et retournant la table des caractères dont leurs codes ascii est entre les deux entiers passés en paramètres.

Exemple :

```
1 >>> ascii(66,70)
2 B 66
3 C 67
4 D 68
5 E 69
6 F 70
```

Les fichiers

Définition

Un fichier texte est un fichier qui contient des caractères et des espaces organisés en lignes successives. Ces lignes étant séparées les unes des autres par un caractère spécial non imprimable appelé «marqueur de fin de ligne».

Les fichiers

Opérations sur les fichiers

- Ouvrir le fichier en appelant la fonction *open* qui renvoie un objet particulier fournissant des méthodes spécifiques.
- Accéder au fichier pour lire son contenu et à écrire dedans.
- Fermeture à la fin en appelant la fonction close.

Les fichiers

Ouverture d'un fichier : la fonction open.

Syntaxe :

```
1 fichier=open(chemin, mode)
```

- **chemin** : l'emplacement où se trouve fichier
- **mode** : mode d'ouverture
  - 'r' : Ouverture en lecture (READ).
  - 'w' : Ouverture en écriture (WRITE), à chaque ouverture le contenu du fichier est écrasé. Si le fichier n'existe pas python il sera crée.
  - 'a' : pour une ouverture en mode ajout à la fin du fichier (APPEND). Si le fichier n'existe pas python le crée.

Les fichiers

Ecriture dans fichier : la méthode write.

La méthode *write* réalise l'écriture dans un fichier texte.

Exemple :

```
1 f=open('Monfichier.txt','a')
2 f.write('Bonjour, fichier !')
3 f.write("Quel beau temps, aujourd'hui !")
```

Remarque :

- Lafonction *write* reçoit exactement un unique argument qui doit obligatoirement être une chaine de caractère.
- La fonction write ne peut être appelée que si le fichier est ouvert en mode '**w**' ou '**a**'

Les fichiers

Lecture séquentielle d'un fichier : la méthode **read**

- La méthode **read()** permet de lire d'un fichier.
- Si on utilise cette méthode sans aucun argument, la totalité du fichier est transférée.
- Elle peut également être utilisée avec un argument indiquant le nombre de caractères à lire, a partir de la position déjà atteinte dans le fichier.

Remarque :

- La fonction *read()* ne peut être appelée que si le fichier est ouvert en mode '**r**'
- S'il ne reste pas assez de caractères pour satisfaire la demande, la lecture s'arrête tout simplement a la fin du fichier.
- Si la fin du fichier est déjà atteinte,la fonction renvoie une chaîne vide.

Les fichiers

Fermeture d'un fichier : la méthode close.

Cette méthode permet de refermer le fichier après usage.

Exemple :

```
1 f.close()
```

Les fichiers

Méthodes supplémentaires

- La méthode **readline** : Cette méthode permet de extraites les lignes séparément les unes des autres.
- La méthode **readlines** : Cette méthode transfère les lignes restantes d'un fichier dans une liste de chaines.
- La méthode **seek(position)** : Elle permet de placer le curseur dans une position donnée.

Remarque :

Une fois arrivé à la fin du fichier, la méthode readline renvoie une chaine vide, tandis que la méthode readlines renvoie une liste vide.

Exercice 1 :

Écrire la fonction *copier* qui reçoit en paramètres les noms de deux fichiers, dont le premier est supposé existant et le deuxième est à créer en y copiant le contenu du premier tout en remplaçant les espaces par le caractère "*".

Exercice 2 :

Écrire une version modifiée *copier_2* de la fonction *copier* de l'exercice précèdent permettant de copier le contenu du premier dans le deuxième, en omettant, toutes les lignes qui commencent par le caractère "#".

Exercice 3 :

Écrire la fonction permettant de compter le nombre de caractères d'un fichier tout en omettant les espaces et les retours à la ligne (\n).

Exercice 4 :

Sans utiliser la méthode *upper* (n'utiliser que les codes ASCII), écrire la fonction *toMaj* qui reçoit en paramètres les noms de deux fichiers, dont le premier est supposé existant et le deuxième est à créer en y copiant le contenu du premier tout en le convertissant en majuscule.

Exercice : Soit la classes suivante :

```
1  class Complexe: #Définition de la classe Complexe
2  """Classe définissant un nombre complexe par sa partie réelle et
       sa partie imaginaire"""
3      #Constructeur
4      def __init__(self,x,y):
5          ...
6      #Afficheur
7      def __repr__(self):
8          ...
9      # Conversion en chaine de caractères
10     def __str__(self):
11         ...
```

1. Implémenter les méthodes de la classes ci-dessus.
2. Écrire les méthodes suivantes pour la classe **Complexe**
   1. *addition* permettant de calculer et de retourner la somme du complexe lui-même et d'un autre complexe.
   2. *oppose* qui renvoie le complexe opposé du complexe lui-même ;
   3. *conjugue* qui renvoie le complexe conjugué du complexe lui-même ;
   4. *inverse* qui renvoie le complexe inverse du complexe lui-même ;
   5. *produit* qui renvoie le complexe produit du complexe lui-même et d'un autre complexe.
   6. *difference* qui renvoie le complexe différence du complexe lui-même et d'un autre complexe
   7. *distance* qui renvoie la distance entre deux complexe $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Connexion aux bases de données (Mysql)

Aperçu des fonctions clés du connecteur MySQL pour Python : **mysql.connector.connect()** : Cette fonction permet d'établir une connexion à une base de données MySQL. Elle prend en charge plusieurs paramètres tels que l'hôte, le nom d'utilisateur, le mot de passe et le nom de la base de données.

```python
import mysql.connector

# Établissement de la connexion
connexion = mysql.connector.connect(
    user='votre_utilisateur',
    password='votre_mot_de_passe',
    host='localhost',
    database='votre_base_de_donnees'
)

# Vérification de l'état de la connexion
if connexion.is_connected():
    print('Connexion à la base de données MySQL établie avec succès.')

# Fermeture de la connexion
connexion.close()
```

Connexion aux bases de données (Mysql)

cursor.execute() : Cette fonction est utilisée pour exécuter une requête SQL sur la base de données via un curseur. Elle prend en charge l'exécution de requêtes de lecture et de modification.

```
# Création du curseur
curseur = connexion.cursor()

# Exécution d'une requête de lecture
curseur.execute("SELECT * FROM votre_table")
resultats = curseur.fetchall()

# Affichage des résultats
for resultat in resultats:
    print(resultat)
```

Connexion aux bases de données (Mysql)

cursor.fetchall() : Cette fonction permet de récupérer tous les enregistrements résultant d'une requête de sélection. Elle renvoie les données sous forme de tuples.

```
# Création du curseur
curseur = connexion.cursor()

# Exécution d'une requête de lecture
curseur.execute("SELECT * FROM votre_table")
resultats = curseur.fetchall()

# Affichage des résultats
for resultat in resultats:
    print(resultat)
```

Connexion aux bases de données (Mysql)

cursor.executemany() : Cette fonction est utilisée pour exécuter plusieurs requêtes avec des données différentes de manière efficace. Elle est utile lorsqu'il est nécessaire d'effectuer des opérations sur plusieurs enregistrements en une seule opération.

```
# Création du curseur
curseur = connexion.cursor()

# Liste de données à insérer
donnees = [
    ('Ali', 25, 'ali@example.com'),
    ('Karim', 30, 'karim@example.com'),
    ('Yassine', 28, 'yassine@example.com')
]

# Requête d'insertion avec executemany
requete = "INSERT INTO votre_table (nom, age, email) VALUES (%s, %s, %s)"
curseur.executemany(requete, donnees)

# Validation des modifications
connexion.commit()
```

Connexion aux bases de données: Fonctions Clés

Fonctions clés du connecteur MySQL pour Python :

1. **connect()** : Établit une connexion à la base de données MySQL.

2. **cursor**() : Crée un curseur pour exécuter des requêtes avec la base de données.

3. **execute()** : Exécute une requête SQL donnée.

4. **fetchall()** : Récupère toutes les lignes d'un résultat de requête.

5. **fetchone()** : Récupère la prochaine ligne du jeu de résultats.

Connexion aux bases de données: Fonctions Clés

7. **commit()** : Valide les transactions en cours.

8. **rollback()** : Annule les transactions en cours depuis le dernier commit.

9. **close()** : Ferme le curseur et la connexion à la base de données.

Connexion aux bases de données: Exercice 1

Vous devez créer un script Python pour interagir avec une base de données MySQL. Suivez les étapes ci-dessous :

1. Établissez une connexion à une base de données MySQL locale.

2. Créez une table appelée "clients" avec les colonnes suivantes :

- id (entier, clé primaire, auto-incrémenté)
- nom (chaîne de caractères, longueur maximale de 255)
- age (entier)

3. Insérez un enregistrement dans la table "clients" avec des données de votre choix.

4. Affichez tous les enregistrements de la table "clients".

5. Supprimez l'enregistrement avec le nom « XXX" de la table "clients".

6. Assurez-vous de fermer la connexion à la base de données après avoir terminé toutes les opérations.

Connexion aux bases de données: Exercice 2

Vous devez écrire un script Python pour gérer une base de données de stock pour un magasin. Voici les étapes :

1. Connectez-vous à une base de données MySQL locale.
2. Créez une table "stock" avec les colonnes suivantes :
   - id (entier, clé primaire, auto-incrémenté)
   - nom du produit (chaîne de caractères, longueur maximale de 255)
   - prix unitaire (décimal, 10 chiffres au total, 2 chiffres après la virgule)
   - quantité en stock (entier)
3. Insérez cinq enregistrements de produits avec des noms, des prix et des quantités en stock variés.
4. Affichez tous les produits dont le prix est supérieur à 50.
5. Mettez à jour la quantité en stock pour un produit de votre choix.
6. Supprimez un produit de votre choix de la base de données.
7. Fermez la connexion à la base de données.

Connexion aux bases de données: Exercice 3

1. Établissez une connexion à une base de données MySQL locale.
2. Créez deux tables : "clients" et "commandes" avec les schémas suivants :
   - Table "clients" : id (entier, clé primaire, auto-incrémenté), nom (chaîne de caractères, longueur maximale de 255), email (chaîne de caractères, longueur maximale de 255).
   - Table "commandes" : id (entier, clé primaire, auto-incrémenté), id_client (entier, clé étrangère faisant référence à l'id du client), montant_total (décimal, 10 chiffres au total, 2 chiffres après la virgule), date_commande (date).
3. Insérez au moins cinq clients dans la table "clients" et au moins dix commandes dans la table "commandes" en vous assurant que chaque commande est associée à un client existant.
4. Pour chaque client, calculez et affichez le montant total des commandes passées.
5. Supprimez tous les clients qui n'ont aucune commande associée.
6. Fermez la connexion à la base de données.

Connexion aux bases de données: Exercice 4

1. Écrire une fonction pour établir une connexion à une base de données MySQL en prenant en compte les informations d'authentification telles que l'hôte, la base de données, le nom d'utilisateur et le mot de passe.
2. Écrire une fonction pour créer une table dans la base de données MySQL en prenant en paramètre la connexion établie et le nom de la table. Les champs sont : ID, Nom, prenom, Tel, DDN, Adresse.
3. Écrire une fonction pour insérer des données dans une table spécifique en prenant la connexion à la base de données, le nom de la table, ainsi que les données à insérer en tant que paramètres.
4. Écrire une fonction pour afficher les données d'une table spécifiée en prenant la connexion à la base de données et le nom de la table en paramètres.
5. Écrire une fonction pour supprimer des données d'une table spécifique en utilisant une condition fournie en tant que paramètre.
6. Écrire une fonction pour mettre à jour des données dans une table spécifiée en utilisant une condition fournie ainsi que les nouvelles valeurs à mettre à jour en tant que paramètres.

Connexion aux bases de données:

Dictionnaire

un dictionnaire est une structure de données qui permet de stocker des paires clé-valeur. Chaque élément dans un dictionnaire est une paire composée d'une clé et de sa valeur associée.

```python
# Création d'un dictionnaire
mon_dictionnaire = {'prenom':'kamal', 'nom':'Alami', 'age': 30, 'ville': 'Rabat'}

# Accès aux éléments du dictionnaire
print(f"Le prénom est : {mon_dictionnaire['prenom']}")
print(f"L'âge est : {mon_dictionnaire['age']}")
```

Connexion aux bases de données:

Dictionnaire

```python
mon_dictionnaire = {'prenom':'kamal', 'nom':'Alami', 'age': 30, 'ville': 'Rabat'}

# Accès aux éléments du dictionnaire
print(f"Le prénom est : {mon_dictionnaire['prenom']}")
print(f"L'âge est : {mon_dictionnaire['age']}")

# Parcours des clés et des valeurs dans le dictionnaire
for cle, valeur in mon_dictionnaire.items():
    print(f"La clé {cle} a pour valeur {valeur}")

# Ajout d'un nouvel élément au dictionnaire
mon_dictionnaire['profession'] = 'Ingénieur'

# Suppression d'un élément du dictionnaire
del mon_dictionnaire['ville']

# Vérification de l'existence d'une clé dans le dictionnaire
if 'nom' in mon_dictionnaire:
    print("La clé 'nom' existe dans le dictionnaire.")
```

Connexion aux bases de données: Exercice 4

1. Créez une base de données vide dans votre système de gestion de base de données MySQL en utilisant le nom de base de données fourni comme paramètre.
2. Créez une table spécifiée avec ses champs dans la base de données créée précédemment. Les paramètres de cette question doivent inclure le nom de la table ainsi que les champs avec leurs types de données respectifs.
3. Insérez au moins cinq enregistrements dans la table précédemment créée avec des données de votre choix.
4. Affichez tous les enregistrements de la table spécifiée sur la console.
5. Mettez à jour un enregistrement spécifié dans la table en modifiant des valeurs spécifiques.
6. Supprimez un enregistrement spécifié de la table.

Analyse de données avec Python

Corrélation, régression linéaire simple

Introduction : précisions sémantiques

La corrélation indique le degré de linéarité entre deux variables (quantitatives). **La régression** simple indique la nature de la liaison linéaire entre deux variables (quantitatives). Ainsi l'analyse de **régression** fournit une fonction entière (une droite par exemple) alors que l'analyse de corrélation fournit un simple nombre – un indice qui renseigne sur l'intensité avec laquelle 2 variables évoluent ensemble. Ces 2 techniques sont donc complémentaires. **L'analyse causale** enfin va plus loin en précisant le sens de la relation, le chemin de la cause à l'effet.

Corrélation, régression linéaire simple

Méthode et but
2 variables numériques (quantitatives)
Identifier la nature des *variables* : *indépendante x* et *dépendante y*.
Décrire la relation entre les variables

- graphiquement
- en utilisant une équation

Utiliser l'équation pour prévoir une valeur y_i à partir d'une valeur x_i.
Etablir le degré de fiabilité de l'estimation (relation probabiliste seulement)

La relation entre deux variables peut être :

- ***déterministe*** (Ceci ne nous concerne pas ici)
- ***probabiliste*** (C'est ce dont on va parler)

Corrélation, régression linéaire simple

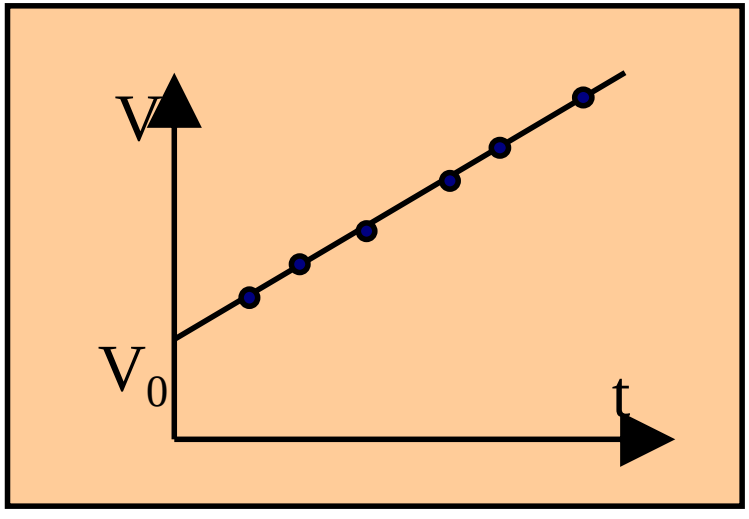
Relation déterministe: La valeur de la variable *y* peut être <u>précisement</u> prédite à partir de la valeur de la variable *x*.

***Exemples*:**

- Prix d'une maison et taxe due.

- Vitesse d'un corps en chute libre et temps.

$V=V_0+gt$



Corrélation, régression linéaire simple

Relation probabiliste: La valeur d'une variable *y* ne peut pas être précisement prédite à partir de la valeur de la variable *x* - à cause d'autres facteurs.

***Exemples*:**

1. Consommation en eau et une population

***x* = nombre d'habitants**

***y* = eau consommée**

2. Nombre d'heures passées à réviser un examen et la note obtenue.

***x* = heures passées à réviser**

***y* = note obtenue**

Regression possible avec une relation *probabiliste*.

Corrélation, régression linéaire simple

Coefficient de corrélation de Bravais-Pearson

$$r = \frac{Cov(x,y)}{s_x s_y} = \frac{s_{xy}}{s_x s_y}$$

$$r = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2} \sqrt{\sum (y_i - \bar{y})^2}}$$

Un exemple...

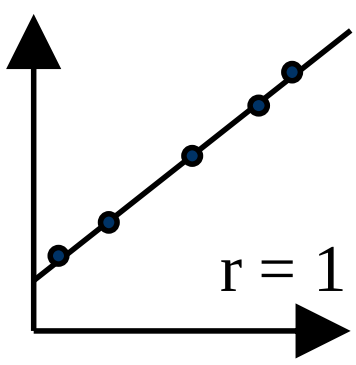
Numéro de l'essai i	Masse m_i x_i	Long. l_i y_i	$(x_i-\bar{x})$	$(x_i-\bar{x})^2$	$(y_i-\bar{y})$	$(y_i-\bar{y})^2$	$(x_i-\bar{x})(y_i-\bar{y})$
1	2	42.0	-4.0	16.0	-9.3	86.9	37.28
2	4	48.4	-2.0	4.0	-2.9	8.5	5.84
3	6	51.3	0.0	0.0	0.0	0.0	0
4	8	56.3	2.0	4.0	5.0	24.8	9.96
5	10	58.6	4.0	16.0	7.3	53.0	29.12
n=5	$\bar{X}=6$	$\bar{Y}=51.32$	$\sum=0.0$	$\sum=40$	$\sum=0.0$	$\sum=173.2$	$\sum=82.2$

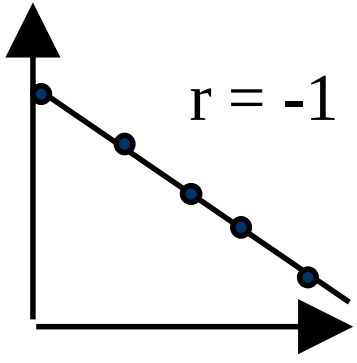
Statistiques

$$r=\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum(x_i-\bar{x})^2}\sqrt{\sum(y_i-\bar{y})^2}}=\frac{82,2}{\sqrt{173,2\times 40}}=0,987$$

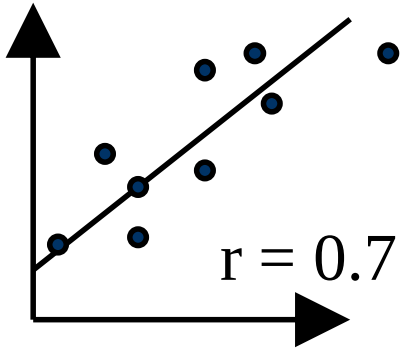
Corrélation, régression linéaire simple

$$|r| \leq 1 \quad \text{ou} \quad -1 \leq r \leq 1$$

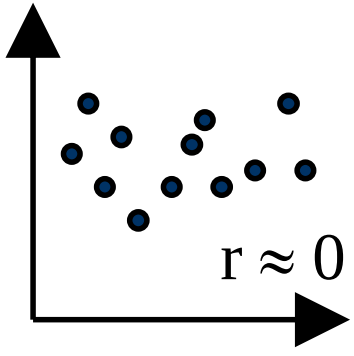




Liaisons absolues (déterministe)



Liaison stochastique (probabiliste)



Pas de liaison

Exemple

vous disposez d'un jeu de données représentant la taille (en centimètres) et le poids (en kilogrammes) de 10 personnes. Ces données sont stockées dans deux listes Python, une pour la taille et une pour le poids.

1. Création des données :

- Créez deux listes, une pour la taille et une pour le poids, chacune contenant 10 valeurs synthétiques.

2. Calcul de la corrélation :

- Utilisez ces listes pour créer un DataFrame pandas.
- Calculez la corrélation entre la taille et le poids en utilisant la fonction corr().

Exemple

3. Affichage des résultats :

- Affichez un nuage de points pour visualiser la relation entre la taille et le poids.
- Affichez la valeur de corrélation calculée.

4. Interprétation :

- À la lumière des résultats, comment interprétez-vous la corrélation entre la taille et le poids ? Est-ce que cela confirme ou infirme une relation directe entre ces deux variables ?

NB:
- Utilisez la bibliothèque matplotlib pour le nuage de points.
- Assurez-vous que le code est suffisamment commenté pour expliquer chaque étape.

Pandas - Traitement et Analyse de Données

- Pandas est une bibliothèque open-source pour la manipulation et l'analyse de données en Python.
- Développée sur la base de NumPy, Pandas offre des structures de données flexibles et performantes.
- Principales structures de données : Series (séries temporelles) et DataFrame (tableaux bidimensionnels).

Pandas - Traitement et Analyse de Données

Puissance de Pandas

- **Lecture de Données :** Importation de données à partir de divers formats (CSV, Excel, SQL, etc.).
- **Manipulation de Données :** Filtres, tris, groupes, pivotages et fusion de données.
- **Nettoyage de Données :** Gestion des valeurs manquantes, suppression de doublons et remplacement de valeurs.
- **Analyse Statistique :** Calcul de statistiques descriptives et opérations mathématiques.

Pandas - Traitement et Analyse de Données

Visualisation et Exportation

- **Visualisation de Données :** Pandas s'intègre avec d'autres bibliothèques comme Matplotlib pour créer des graphiques et des visualisations.
- **Exportation de Données :** Exportation des données traitées vers divers formats (CSV, Excel, etc.).
- **Intégration avec d'autres Librairies :** Utilisation en tandem avec des outils comme NumPy, Matplotlib et Scikit-Learn pour des analyses avancées.

Pandas - Traitement et Analyse de Données

Exploration des Fonctions de Pandas

Lecture de Données :
pd.read_csv('fichier.csv') : Importation de données depuis un fichier CSV.
pd.read_excel('fichier.xlsx') : Importation de données depuis un fichier Excel.
*pd.read_sql('SELECT * FROM table', connection) : Importation de données depuis une base de données SQL.*

***Manipulation de Données** :*
df.head(n) : Affiche les premières lignes du DataFrame.
df.tail(n) : Affiche les dernières lignes du DataFrame.
df.describe() : Statistiques descriptives pour les données numériques.

Pandas - Traitement et Analyse de Données

Exploration des Fonctions de Pandas

Sélection et Filtre :

df['colonne'] : Sélectionne une colonne spécifique.
df[['colonne1', 'colonne2']] : Sélectionne plusieurs colonnes.
df[df['colonne'] > seuil] : Filtre les données en fonction d'une condition.

Manipulation de Valeurs :

df.drop(['colonne'], axis=1) : Supprime une colonne.
df.fillna(valeur) : Remplace les valeurs manquantes par une valeur spécifiée.
df.replace(valeur_a, valeur_b) : Remplace une valeur par une autre.

Pandas - Traitement et Analyse de Données

Groupement et Agrégation :

df.groupby('colonne').mean() : Agrégation par moyenne.
df.groupby(['colonne1', 'colonne2']).sum() : Agrégation multiple.

Visualisation de Données :

df.plot(x='colonne_x', y='colonne_y', kind='scatter') : Crée un graphique de dispersion.
df['colonne'].hist() : Affiche un histogramme.

Exportation de Données :

df.to_csv('nouveau_fichier.csv', index=False) : Exportation vers un fichier CSV.
df.to_excel('nouveau_fichier.xlsx', index=False) : Exportation vers un fichier Excel.

Solution

```python
import pandas as pd
import matplotlib.pyplot as plt

# 1. Création des données synthétiques
taille = [160, 165, 155, 170, 168, 162, 175, 180, 158, 172]
poids = [55, 60, 50, 70, 65, 58, 75, 80, 52, 68]

# 2. Calcul de la corrélation
donnees = pd.DataFrame({'Taille': taille, 'Poids': poids})
correlation = donnees['Taille'].corr(donnees['Poids'])

# 3. Affichage des résultats
# Nuage de points
donnees.plot.scatter(x='Taille', y='Poids', title=f'Nuage de points - Corrélation : {correlation}')
plt.show()

# Affichage de la corrélation
print(f"\nCorrélation entre Taille et Poids : {correlation}")

# 4. Interprétation
# La corrélation positive suggère qu'il existe une relation directe entre la taille et le poids.
# Cela signifie généralement que les personnes plus grandes ont tendance à avoir un poids plus élevé.
```

Corrélation & régression

Exemple

Si je m'intéresse au lien entre le temps hebdomadaire moyen passé à travailler (X) et la note obtenue au partiel (Y) :

• **L'analyse de régression** permet de déterminer une fonction qui lie les deux variables :

ex : « $Y = aX + b$ »

• **L'analyse de corrélation** renseigne sur l'intensité du lien entre les deux variables : ex : « le lien est fort et très significatif ».

r et r^2 :

• Comme r indique le degré de la relation entre la variation d'une variable et celle d'une autre variable, il peut également représenter la décomposition de la variation totale (en étant au carré). On retiendra que

r^2 = variation expliquée ÷ variation totale

➔ **r^2 mesure la proportion de la variation d'une variable qui est expliquée par l'autre.**

• r et r^2 sont des mesures symétriques d'association : la corrélation entre X et Y est la même que la corrélation entre Y et X. Il n'est pas important de savoir quelle est la variable indépendante et quelle est la variable dépendante.

Interprétation du R^2 :

Variance expliquée : R^2, coefficient de détermination (proportion de variance totale de Y qui n'est pas due à l'erreur, ou encore proportion de la variance de Y expliquée par la variance de X)

- $R^2 = 0$: la variable indépendante n'explique rien
- $R^2 = 1$: la variable explique complètement Y
- $R^2 = 0{,}11$: 11% des variations de Y sont expliquées par le modèle

Le coefficient de corrélation linéaire r renseigne sur l'intensité du lien entre 2 variables quantitatives. Il doit être complété afin de déterminer si l'éventuel lien mis à jour est significatif ou non. On utilise pour cela un test t :

$$t = r.\sqrt{\frac{n-2}{1-r^2}}$$

Remarque : sous SPPS, la probabilité critique du test est fournie par la rubrique « *sig. (bilatérale)* »

Exercice

BDD Employes de SPSS : y'a-t-il une corrélation positive significative entre salaire actuel et salaire à l'embauche ? Entre salaire actuel et nombre d'année d'ancienneté ?

- H0 : il n'y a aucun lien entre ces deux variables (r=0)
- H1 : il existe un lien entre ces deux variables (r≠0)

Analyse ➔ Corrélation ➔ Bivariée

La régression simple :

Elle consiste à déterminer une équation qui relie 2 variables quantitatives. Contrairement à la corrélation simple, elle nécessite d'identifier l'une des 2 variables comme étant dépendante (à expliquer) et l'autre comme étant indépendante (explicative). Remarquons tout de même que cette méthode n'implique pas de causalité.

Le modèle type est de la forme :

$\mathbf{Y_i = \beta_0 + \beta_1 X_i + e_i}$ avec Y = variable dépendante (à expliquer)

X = variable indépendante (ou explicative)

β_0 = ordonnée à l'origine de la droite β_1 = pente de la droite

e_i = terme d'erreur associé à la ième observation

La régression simple, vocabulaire :

•**Coefficient de détermination r^2 :** proportion de la variation totale de Y expliquée par la variation de X

• **Valeur estimée (ou prédite) de Y_i :** $\hat{Y}_i = a + bx$ avec $\hat{Y}_i$ la valeur estimée de Y_i et a et b les estimateurs respectifs de β_0 et β_1.

• **Coefficient de régression :** le paramètre b est appelé coefficient de régression.

• **L'écart-type résiduel (SEE) :** c'est l'écart-type des erreurs (valeurs réelles Y moins valeurs estimées $\hat{Y}$).

La régression simple, vocabulaire (suite) :

•**Coefficient de régression standardisé (coefficient bêta) :** il correspond à la pente obtenue par la régression de Y sur X lorsque les données sont standardisées.

• **Somme des erreurs au carré :** les distances de tous les points à la droite de régression sont élevées au carré et additionnées pour obtenir la somme des erreurs au carré, qui est une mesure de l'erreur totale

• **Statistique t :** valeur du t de Student à n-2 degrés de liberté, afin de rejeter ou non H0. Cette statistique est associée à sa probabilité critique (significative lorsqu'elle est < 0,05)

Les étapes d'une analyse de régression simple :

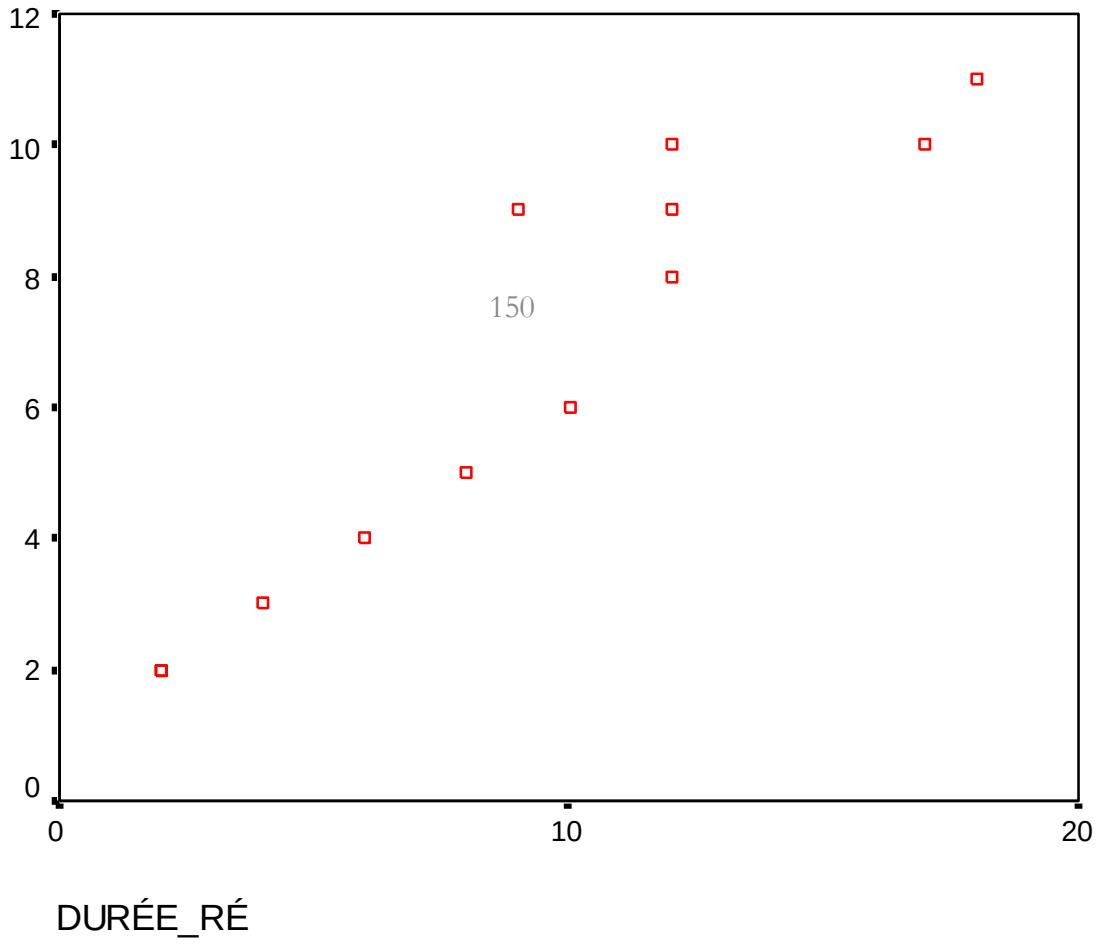
1. La première étape consiste à **représenter le nuage de points**, variable dépendante sur l'axe vertical et variable indépendante sur l'axe horizontal.

Cela permet de se faire une idée sur le type de lien (est-ce linéaire ?) et de détecter les éventuelles valeurs extrêmes qui risquent de perturber l'analyse.

Sous *SPSS :* Graph ➔ Diagramme de dispersion ➔ Simple

Corrélation & régression

Analyse bivariée



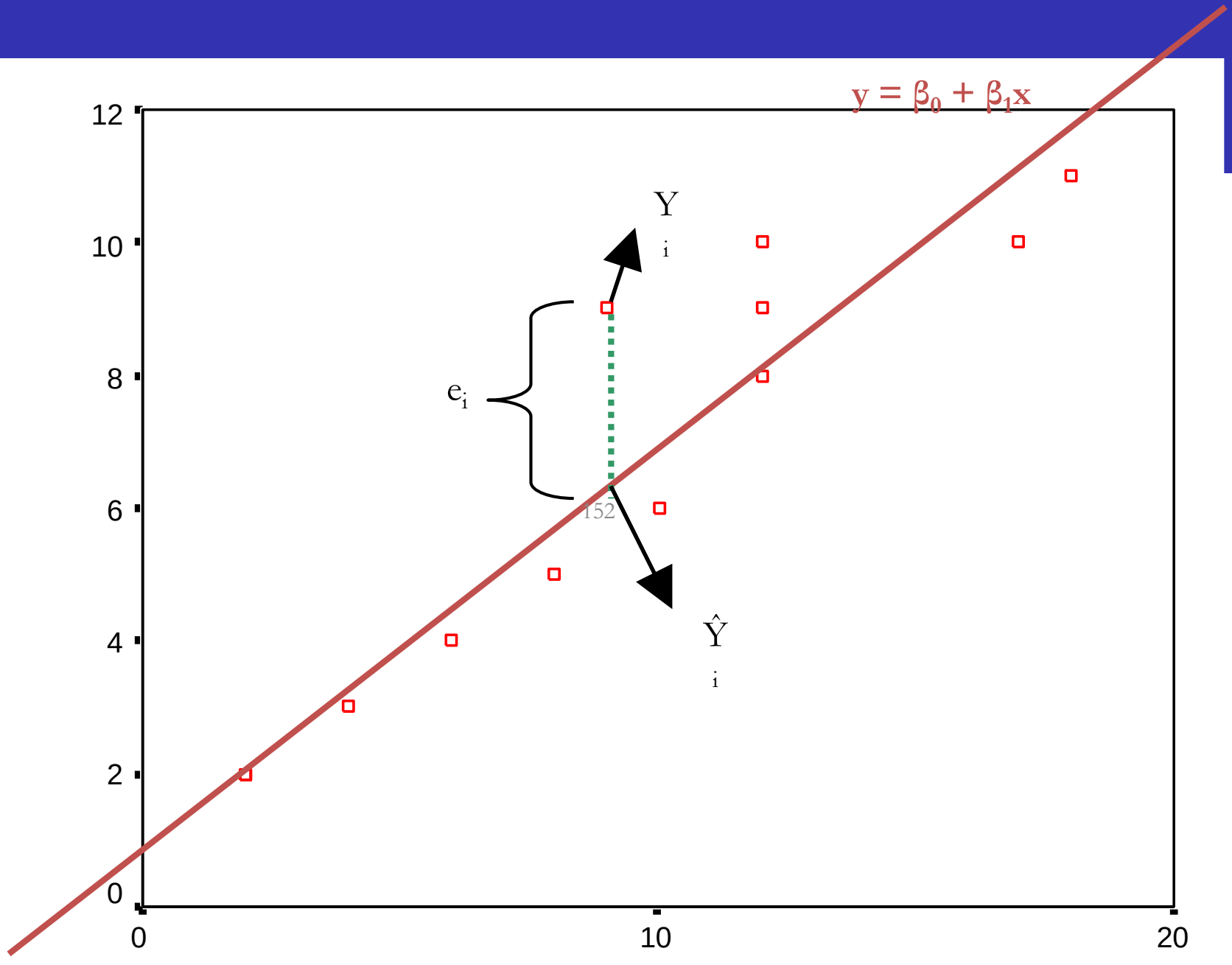
2. Il s'agit ensuite de **trouver les caractéristiques de la droite** qui décrit le mieux les données. On utilise généralement la méthode des moindres carrés. Elle consiste à déterminer la droite de régression qui minimise le carré des distances verticales entre les points et la droite.

Avec une équation du type $\mathbf{Y_i = \beta_0 + \beta_1 X_i + e_i}$ la distance verticale du point à la droite est représenté par $\mathbf{e_i}$.

Les distances de tous les points à la droite élevés au carrés et additionnés forment la somme des carrés des erreurs, ou « erreur totale », notée

$$\sum e_j^2$$

➔ Le but est que cette valeur soit minimale (que les distances verticales soient minimisées)


$y = \beta_0 + \beta_1 x$
12
10
8
6
4
2
0
0
10
20
Y_i
e_i
152
$\hat{Y}_i$
DURÉE_RÉ

La méthode des moindres carrés

Critère des moindres carrés

$$\min \sum (y_i - \hat{y}_i)^2$$

où:

y_i = valeur <u>observée</u> de la variable dépendante pour pour la *I* ème observation

$\hat{y}_i$ = valeur <u>estimée</u> de la variable dépendante pour la *I* ème observation

La méthode des moindres carrés …

$$SCres = e_1^2 + e_2^2 + e_3^2 + \ldots + e_n^2 = \sum_{i=1}^{n} e_i^2$$

$$SCres = \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2$$

$$SCres = \sum_{i=1}^{n} \left(y_i - b_0 - b_1 x_i \right)^2$$

Cette mesure donne l'ordre de grandeur de la dispersion des observations Y_i autour de la droite de régression

Il s'agit de trouver b_o et b_1 de sorte que la somme des carrés des résidus *SCres* soit la plus petite possible (minimale).

Principes de la méthode des moindres carrés …

Les estimations ponctuelles des paramètres de la droite de régression obtenues par la méthode des moindres carrés sont :

$$b_0 = \bar{y} - b_1 \bar{x}$$

$$b_1 = \frac{\sum_{i=1}^{n} x_i y_i - n\,\bar{x}\,\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2}$$

Taille de l'échantillon

Autre formule pour b_1

$$b_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

À partir des données ci-dessous, déterminez les estimations ponctuelles des paramètres de la droite de régression selon la méthode des moindres carrés :

y_i	x_i
24	10
40	20
36	30
45	40
55	50

y_i	x_i	$x_i y_i$	x_i^2
24	10	240	100
40	20	800	400
36	30	1080	900
45	40	1800	1600
55	50	2750	2500
$\bar{y} = \frac{\sum y_i}{5} = 40$	$\bar{x} = \frac{\sum x_i}{5} = 30$	$\sum x_i y_i = 6670$	$\sum x_i^2 = 5500$

$$b_1 = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2} = \frac{6670 - 5 \times 30 \times 40}{5500 - 5 \times (30)^2} = 0,67$$

$$b_0 = \bar{y} - b_1 \bar{x} = 40 - 0,67 \times 30 = 19,9$$

$$\hat{y} = 19,9 + 0,67x$$