Algorithmique et programmation 1

DE MIE 1re année

2019-2020

DAUPHINE UNIVERSITÉ PARIS | MIDO MATHEMATIQUES ET INFORMATIQUE DE LA DECISION ET DES ORGANISATIONS

PSL RESEARCH UNIVERSITY PARIS

Objectif : Apprendre à programmer pour résoudre un problème

- Conception d'algorithmes
- Mise en œuvre informatique avec le langage Python

Organisation

3h cours/TD et 3h de cours/TP par semaine
(Pensez à activer votre compte ENT ! !)

Évaluation

- Partiel : Semaine du 4 novembre 2019 (note P)
- TP noté : Semaine du 6 janvier 2020 (note TP)
- Examen : Semaine du 6 janvier 2020 (note E)

Note finale $= (0,2P + 0,3TP) + 0,5E$

Contenu du cours

1. Introduction et premières instructions
2. Instructions répétitives
3. Fonctions et programmes modulaires
4. Séquences, tableaux et instructions POUR
5. Chaînes de caractères
6. Listes
7. Notions avancées sur les listes
8. Portée des variables en Python
9. Récursivité
10. Tuples et dictionnaires
11. Fichiers

Plan du cours

Chapitre 1 - Introduction et premières instructions

1.1 Algorithmes et programmation : principaux concepts

1.2. Valeurs, types et variables

1.3. Affectation et expression

1.4. Instruction de lecture

1.5. Instruction d'écriture

1.6. Instructions conditionnelles et expressions logiques

Exemple

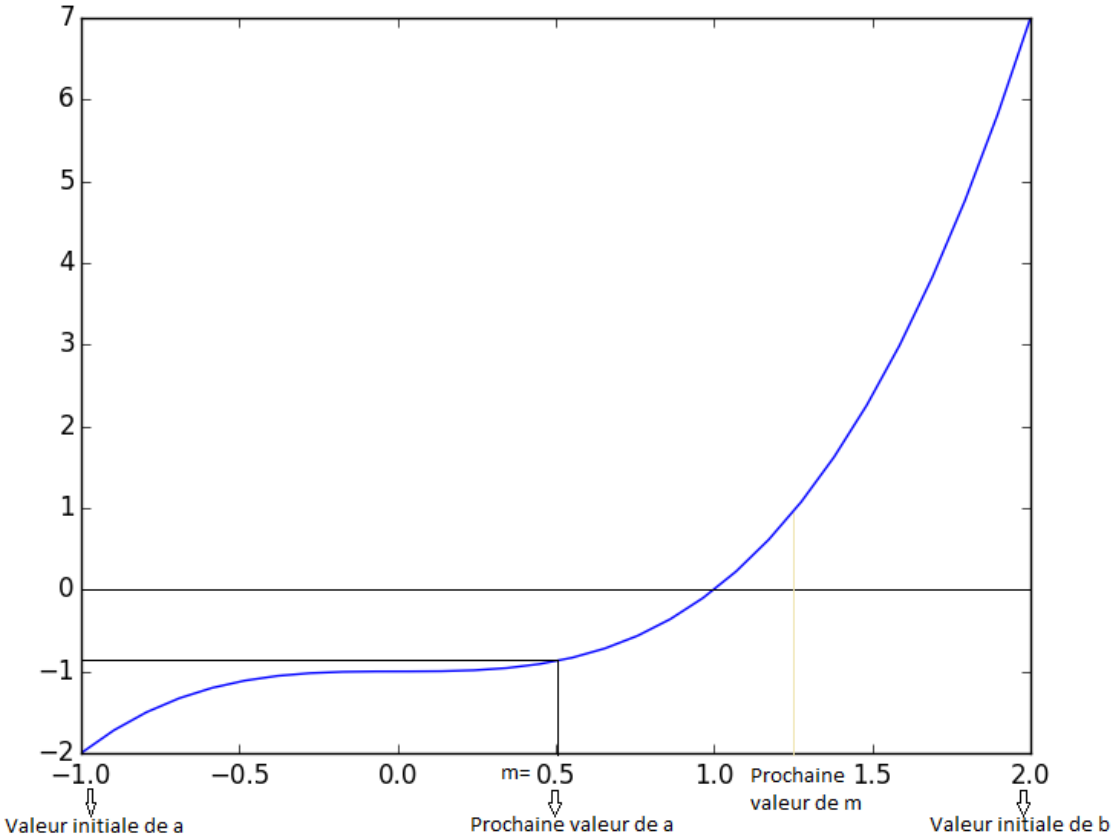


FIGURE – $f(x) = x^3 - 1$

Modèle d'ordinateur

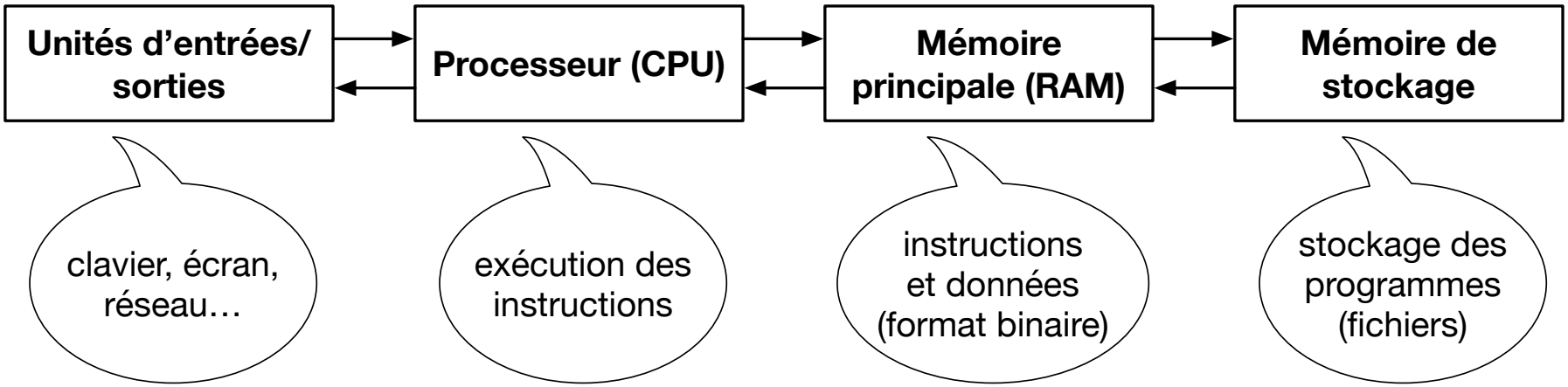


Table ASCII (*American Standard Code for Information Interchange*)

Character	Decimal Number	Binary Number	Character	Decimal Number	Binary Number
blank space	32	0010 0000	^	94	0101 1110
!	33	0010 0001	-	95	0101 1111
“	34	0010 0010	`	96	0110 0000
#	35	0010 0011	a	97	0110 0001
$	36	0010 0100	b	98	0110 0010
A	65	0100 0001	c	99	0110 0011
B	66	0100 0010	d	100	0110 0100
C	67	0100 0011	e	101	0110 0101
D	68	0100 0100	f	102	0110 0110
E	69	0100 0101	g	103	0110 0111
F	70	0100 0110	h	104	0110 1000
G	71	0100 0111	i	105	0110 1001
H	72	0100 1000	j	106	0110 1010
I	73	0100 1001	k	107	0110 1011
J	74	0100 1010	l	108	0110 1100
K	75	0100 1011	m	109	0110 1101
L	76	0100 1100	n	110	0110 1110
M	77	0100 1101	o	111	0110 1111
N	78	0100 1110	p	112	0111 0000
O	79	0100 1111	q	113	0111 0001
P	80	0101 0000	r	114	0111 0010
Q	81	0101 0001	s	115	0111 0011
R	82	0101 0010	t	116	0111 0100
S	83	0101 0011	u	117	0111 0101
T	84	0101 0100	v	118	0111 0110
U	85	0101 0101	w	119	0111 0111
V	86	0101 0110	x	120	0111 1000
W	87	0101 0111	y	121	0111 1001
X	88	0101 1000	z	122	0111 1010
Y	89	0101 1001	{	123	0111 1011
Z	90	0101 1010	|	124	0111 1100
[	91	0101 1011	}	125	0111 1101
/	92	0101 1100	~	126	0111 1110
]	93	0101 1101			

Mots réservés de Python

A ne pas utiliser comme identificateur !!

`False`	`None`	`True`	`and`	`as`
`assert`	`break`	`class`	`continue`	`def`
`del`	`elif`	`else`	`except`	`finally`
`for`	`from`	`global`	`if`	`import`
`in`	`is`	`lambda`	`nonlocal`	`not`
`or`	`pass`	`raise`	`return`	`try`
`while`	`with`	`yield`		

Instruction d'écriture

- Fonction `print()` avec affichage par défaut :

```
>>> pi = 3.14
>>> print("2*pi=", 2 * pi)
```

- Modification de l'affichage par défaut avec le paramètre `sep` :

```
>>> pi = 3.14
>>> print(pi, 2 * pi, 4 * pi)

>>> print(pi, 2 * pi, 4 * pi, sep = ",")

>>> print(pi, 2 * pi, 4 * pi, sep = "\n")
```

- Modification de l'affichage par défaut avec le paramètre `end` :

```
>>> print(pi, 2 * pi, 4 * pi, end = "Fin")
```

Instructions conditionnelles : exemple

Algorithme qui détermine si une équation du second degré, $ax^2 + bx + c = 0$, n'admet aucune solution réelle :

```
ALGO
VARIABLES
a TYPE NOMBRE
b TYPE NOMBRE
c TYPE NOMBRE
d TYPE NOMBRE
DEBUT
    ECRIRE "Quel est le parametre a?"
    LIRE a
    ECRIRE "Quel est le parametre b?"
    LIRE b
    ECRIRE "Quel est le parametre c?"
    LIRE c
    d <- (b*b - 4*a*c)
    SI (d<0) ALORS
        DEBUT
            ECRIRE "L'equation n'admet aucune racine reelle"
        FIN
FIN
```

Reprise de l'exemple du transparent 10 avec un `SINON` :

```
ALGO
VARIABLES
a, b, c, d TYPE NOMBRE
DEBUT
    ECRIRE "Quel est le parametre a?"
    LIRE a
    ECRIRE "Quel est le parametre b?"
    LIRE b
    ECRIRE "Quel est le parametre c?"
    LIRE c
    d <- (b*b - 4*a*c)
    SI (d<0) ALORS
        DEBUT
            ECRIRE "L'equation n'admet aucune racine reelle"
        FIN
    SINON
        DEBUT
            ECRIRE "L'equation admet au moins une racine reelle"
        FIN
FIN
```

Imbrications d'instructions conditionnelles : exemple

```
ALGO
VARIABLES
a TYPE NOMBRE
DEBUT
    ECRIRE "Quel est le parametre a?"
    LIRE a
    SI (a!=0) ALORS
        DEBUT
            SI (a%2==0) ALORS
                DEBUT
                    SI (a>0) ALORS
                        DEBUT
                            ECRIRE "C'est un nombre pair positif"
                        FIN
                    ECRIRE "C'est un nombre pair"
                FIN
        FIN
    SINON
        DEBUT
            ECRIRE "Il est nul"
        FIN
FIN
```

Instructions conditionnelles : exemple en Python

Traduction de l'exemple du transparent 10 en Python :

```
a = float(input("Donner le parametre a : "))
b = float(input("Donner le parametre b : "))
c = float(input("Donner le parametre c : "))
d = b * b - 4 * a * c
if d < 0:
    print("L'equation n'admet pas de racine reelle")
    print("La valeur du discriminant est :", d)
```

Bloc d'instructions

Que se passe-t-il lors de l'exécution du programme ci-dessous pour $x = 4$?

```
x = int(input("Saisir une valeur >= 0 : "))
if x >= 10:
    print("x est constitue de plus d'un chiffre")
print("parce qu'il est plus grand que 10")
```

Instruction conditionnelle `if...else`

Traduction de l'exemple du transparent 11 en Python :

```
a = float(input("Donner le parametre a : "))
b = float(input("Donner le parametre b : "))
c = float(input("Donner le parametre c : "))
d = b * b - 4 * a * c
if d < 0:
    print("L'equation n'admet pas de racine reelle")
    print("La valeur du discriminant est :", d)
else:
    print("L'equation admet au moins "
          "une racine reelle")
    print("Le discriminant est positif :", d)
```

Instruction conditionnelle `if...else`

Que fait le programme Python suivant ?

```
if True:
    print("Tout juste")
else:
    print("Pas bon")
```

Instruction conditionnelle `if...elif...else`

```
a = float(input("Donner le parametre a : "))
b = float(input("Donner le parametre b : "))
c = float(input("Donner le parametre c : "))
d = b * b - 4 * a * c
if d < 0:
    print("Le discriminant est negatif :", d)
    print("L'equation n'admet pas de racine reelle")
elif d == 0:
    print("Le discriminant est nul")
    print("L'equation admet une seule racine reelle")
else:
    print("Le discriminant est positif :", d)
    print("L'equation admet deux racines reelles")
```

Expressions logiques : exercice

Dans l'algorithme ci-dessous, le texte de "message2" peut-il être plus explicite quant à la parité et au signe de *a* ?

```
ALGO
VARIABLES
a TYPE NOMBRE
DEBUT
    ECRIRE "Quel est le parametre a?"
    LIRE a
    SI (a>0 ET a%2==0) ALORS
        DEBUT
            ECRIRE "C'est un nombre pair positif"
        FIN
    SINON
        DEBUT
            SI a%2==0 ALORS
                DEBUT
                    ECRIRE "message2"
                FIN
        FIN
FIN
```

Plan du cours

Chapitre 2 - Instruction répétitive

2.1. Instruction répétitive `TANT_QUE` en pseudo-code

2.2. Instruction `while` en Python

Instruction TANT_QUE

Algorithme affichant les entiers de 1 à 10

```
ALGO
VARIABLES
x TYPE NOMBRE
DEBUT
   x <- 1
   TANT_QUE (x<=10) FAIRE
     DEBUT
       ECRIRE x
       x <- x+1
     FIN
FIN
```

Que fait cet algorithme ?

```
ALGO
VARIABLES
n TYPE NOMBRE
DEBUT
  n <- 0
  TANT_QUE (n<=10) FAIRE
    DEBUT
      ECRIRE n
      n <- n+2
    FIN
FIN
```

Exercice

On veut connaître le plus petit entier n tel que $2^n \geq 100$.

Compléter les lignes 6 et 8 de l'algorithme ci-dessous pour qu'il réponde au problème.

```
1    ALGO
2    VARIABLES
3    n TYPE NOMBRE
4    DEBUT
5      n <- 1
6      TANT_QUE (2**n  ???? ) FAIRE
7        DEBUT
8          n <- ????
9        FIN
10      ECRIRE n
11   FIN
```

Ex. repris de `http://www.xm1math.net/algobox/exemples/dichotomie.html`

Instruction `while` en Python

Programme Python affichant les entiers de 1 à 10

```
n = 1
while n <= 10:
    print(n)
    n = n + 1
```

Que se passe-t-il dans les programmes Python ci-dessous ?

```
n = 1
while n <= 10:
    print(n)
n = n + 1
```

```
n = 1
while True:
    print(n)
```

Instruction `break`

Programme qui lit 10 valeurs strictement positive saisies au clavier et affichage leur somme. Interruption en cas d'erreur de saisie.

```
cpt = 0
somme = 0
while cpt < 10:
    x = int(input())
    if x <= 0:
        break
    somme += x
    cpt += 1
if cpt < 10:
    print("Erreur de saisie")
else:
    print("La somme des 10 valeurs lues est :", somme)
```

Instruction non autorisée en algorithmique

Réécriture sans instruction `break`

```
cpt = 0
somme = 0
while cpt < 10:
    x = int(input())
    if x <=0 :
        cpt=10
    else :
        somme += x
        cpt += 1
if x <= 0:
    print("Erreur de saisie")
else:
    print("La somme des 10 valeurs lues est :", somme)
```

Instruction `continue`

Programme qui lit 10 valeurs strictement positive saisies au clavier et affichage leur somme. En cas d'erreur de saisie (valeur <= 0), poursuivre la lecture jusqu'à obtenir 10 valeurs >0.

```
cpt = 0
somme = 0
while cpt < 10:
    x = int(input())
    if x <= 0:
        continue
    somme += x
    cpt += 1
print("La somme des 10 valeurs lues est :", somme)
```

Instruction non autorisée en algorithmique

Réécriture sans l'instruction `continue`

```
cpt = 0
somme = 0
while cpt < 10:
    x = int(input())
    if x > 0:
        somme += x
        cpt += 1
print("La somme des 10 valeurs lues est :", somme)
```

Plan du cours

Chapitre 3 - Fonctions

3.1 Introduction

3.2 Utiliser des fonctions appartenant à des modules

3.3 Pourquoi définir et utiliser vos propres fonctions ?

3.4. Fonctions en pseudo-code

- Définition
- Appel

3.5. Fonctions en Python

- Définition
- Appel
- Structure d'un programme Python
- Documenter l'aide d'une fonction

Exemples de fonctions déjà utilisées en Python :

1. `x = sqrt(4)`
2. `nom = input()`
3. `nom = input("Saisir votre nom")`
4. `ch = "Saisir votre nom"`
   `nom = input(ch)`
5. `print("Nom :", nom, "Prénom :", prenom, note)`

Exemples de modules Python :

- `math`
- `os`
- `random`
- `time`
- `calendar`
- `numpy`

Algorithme listant tous les nombres parfaits allant de 1 à *n*, *n* étant donné par l'utilisateur

```
ALGO
VARIABLES
  n TYPE NOMBRE
DEBUT
  LIRE n
  TANT_QUE (n>1) FAIRE
    DEBUT
      SI (sommeDiviseurs(n)==n) ALORS
          DEBUT
              ECRIRE n
          FIN
      n <- n-1
    FIN
FIN
```

ou en Python :

```
n = int(input())
while n > 1:
    if sommeDiviseurs(n) == n:
        print(n)
    n -= 1
```

Exemple de définition de fonction

Fonction calculant la somme des diviseurs d'un entier passé en paramètre

```
FONCTION sommeDiviseurs(e TYPE NOMBRE) TYPE NOMBRE
  VARIABLES_LOCALES
    somme, d TYPE NOMBRE
  DEBUT
    d <- 1
    somme <- 0
    TANT_QUE (d<e) FAIRE
      DEBUT
        SI ((e%d)==0) ALORS
          DEBUT
            somme <- somme+d
          FIN
        d <- d+1
      FIN
    RENVOYER somme
  FIN
```

Exercice : écrire une fonction prenant un nombre en argument et teste si ce nombre est pair : la fonction doit renvoyer une valeur booléenne `VRAI` si ce nombre est pair et `FAUX` sinon.

Appel de fonction

Exemple de fonction :

```
FONCTION carre(v TYPE NOMBRE) TYPE NOMBRE
    DEBUT
        RENVOYER  v*v
    FIN
```

Exemples d'appel de la fonction `carre()` :

```
c <- carre(2)
ECRIRE carre(2)
SI carre(2)==6 ALORS DEBUT ... FIN
```

Exemple de fonction

Fonction Python calculant la somme des diviseurs d'un entier passé en paramètre

```
def sommeDiviseurs(e):
    d = 1
    somme = 0
    while d < e:
        if e % d == 0:
            somme = somme + d
        d = d + 1
    return somme
```

Exercice : Analyser les fonctions suivantes

```
def afficheBonjour():
    print("Le résultat est Bonjour")

def affiche(v):
    print("Le résultat est", v)

def somme(v1, v2):
    return v1 + v2

def resteEtQuotient(num, denom):
    return num % denom, num // denom
```

Appel et utilisation des fonctions

Analyser ce programme et indiquer ce qu'il va afficher

```
def image(x):
    res = 2 * x + 1
    print("L'image de", x, "sur la fonction 2*x+1 est",
          res)

image(3)
```

Exemple de programme :

```
def plusUn(n):
        n = n + 1
        return n
m = 5
print("Valeur renvoyée :", plusUn(m))
print("Valeur de m :", m)
```

Analyser ce programme et indiquer ce qu'il va afficher

```
def image(x):
  res = 2 * x + 1
  print("L'image de", x, "sur la fonction 2*x+1 est", res)

def image(x):
  res = 10 * x
  print("L'image de", x, "sur la fonction 10*x est", res)

image(3)
```

Appels croisés

```
def f1(x):
    if x == 0:
        print(1)
        f2(0)
    else:
        print(2)
        f2(1)

def f2(x):
    if x == 0:
        print(3)
        f1(1)
    else:
        print(4)

f1(0)  # affiche 1 3 2 4
```

```
def f1(x):
    if x == 0:
        print(1)
        f2(0)
    else:
        print(2)
        f2(1)

f1(0)   # erreur d'exécution

def f2(x):
    if x == 0:
        print(3)
        f1(1)
    else:
        print(4)
```

Aide d'une fonction

```
>>> help(afficheBonjour)
Help on function afficheBonjour in module __main__:
afficheBonjour()
```

Pour personnaliser le message affiché lors de l'appel de l'aide sur une fonction :

```
def afficheBonjour():
    """Fonction qui affiche \"Bonjour\""""
    print("Bonjour")

def afficherValeur(v)
    """Fonction qui affiche la valeur du paramètre v"""
    print(v)
```

Documenter l'aide d'une fonction

Affichage de l'aide des fonctions précédentes :

```
>>> help(afficheBonjour)
Help on function afficheBonjour in module __main__:
afficheBonjour()
    Fonction qui affiche "Bonjour"

>>> help(afficherValeur)
Help on function afficherValeur in module __main__:
afficherValeur(v)
    Fonction qui affiche la valeur du paramètre v
```

Plan du cours

Chapitre 4 - Séquences, tableaux et instruction `POUR`

- 4.1 Introduction - Motivation
- 4.2 Séquences
- 4.3 Type de données tableau
- 4.4 Boucle `POUR`

Type de données tableau

'Stockage' de la séquence $S = a_1, \ldots, a_{p-1}, a_p, \ldots, a_n$ dans un tableau T :

a_1	a_2	$\ldots$	a_p	a_{p+1}	$\ldots$	a_n
$T[0]$	$T[1]$	$\ldots$	$T[p-1]$	$T[p]$	$\ldots$	$T[n-1]$

Tableau à deux dimensions

$T[0][0]$	$T[0][1]$	$\ldots$	$T[0][j]$	$\ldots$	$T[0][m-1]$
$T[1][0]$	$T[1][1]$	$\ldots$	$T[1][j]$	$\ldots$	$T[1][m-1]$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$T[i][0]$	$T[i][1]$	$\ldots$	$T[i][j]$	$\ldots$	$T[i][m-1]$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$T[n-1][0]$	$T[n-1][1]$	$\ldots$	$T[n-1][j]$	$\ldots$	$T[n-1][m-1]$

Algorithme en pseudo-code affichant le carré de tous les entiers de 1 jusqu'à 10 :

```
ALGO
VARIABLES
i TYPE NOMBRE
carre TYPE NOMBRE
DEBUT
    POUR i ALLANT_DE 1 A 10
        DEBUT
            carre <- i*i
            ECRIRE carre
        FIN
FIN
```

Que faut-il modifier dans ce pseudo-code pour afficher tous les carrés des *n* premiers entiers (*n* n'étant pas fixé) ?

Algorithme en pseudo-code permettant de remplacer chaque nombre d'un tableau par son cube :

```
ALGO
VARIABLES
i, n TYPE NOMBRE
cube TYPE NOMBRE
T TYPE TABLEAU DE NOMBRE
DEBUT
    LIRE n
    T <- CREER_TABLEAU(n)
    POUR i ALLANT DE 0 A (n-1)
        DEBUT
            LIRE T[i]
        FIN
    POUR i ALLANT_DE 0 A (n-1)
        DEBUT
            cube <- T[i]*T[i]*T[i]
            T[i] <- cube
        FIN
FIN
```

Exercice :

Écrire en pseudo-code un algorithme qui demande à l'utilisateur de remplir un tableau de NOMBRES (en lui demandant au préalable sa taille), lui demande ensuite un élément x, puis détermine la dernière position (entre 1 et n) de cet élément dans le tableau.

```
ALGO
VARIABLES
x TYPE NOMBRE
i, taille, derniere_indice TYPE NOMBRE
T TYPE TABLEAU DE NOMBRE
DEBUT
    LIRE taille
    T <- CREER_TABLEAU(taille)
    POUR i ALLANT DE 0 A (taille-1)
        DEBUT
            LIRE T[i]
        FIN
    LIRE x
    POUR i ALLANT DE 0 A (taille-1)
        DEBUT
            SI (T[i]==x) ALORS
                DEBUT
                    derniere_indice <- i
                FIN
        FIN
    ECRIRE "La dernière position rencontrée est "
    ECRIRE derniere_indice+1
FIN
```

Plan du cours

Chapitre 5 - Chaînes de caractères en Python

5.1 Type Python `str`
5.2 Opérateurs Python sur les chaînes de caractères
5.3 Fonction et méthodes prédéfinies en Python

Extraction

```
>>> ch = 'Python'
>>> ch[1:3]

>>> ch[2:2]

>>> ch[:4]

>>> ch[4:]

>>> ch[::-1]
```

Concaténation

```
>>> ch = 'Python' + ' pour tous'
>>> ch

>>> ch1 = 'Python'
>>> ch2 = ' pour tous'
>>> ch = ch1 + ch2
>>> ch

>>> ch = 'p' + ch[1:]
>>> ch
```

Duplication et appartenance

Duplication

```
>>> ch = 'bon'
>>> ch = ch * 2
>>> ch
```

Appartenance

```
>>> ch = 'python pour tous'
>>> 'thon' in ch

>>> 'toutes' in ch
```

Fonctions et méthodes prédéfinies en Python

La fonction `len()` :

```
>>> ch = 'python'
>>> longueur = len(ch)
>>> longueur

>>> car = ch[longueur-1]
>>> car
```

Fonctions et méthodes prédéfinies en Python

La méthode `replace()` :

```
>>> ch = "Langage C"
>>> ch.replace("C", "Python")

>>> ch

>>> ch = ch.replace("C", "Python")
>>> ch
```

Plan du cours

Chapitre 6 - Listes en Python et boucle `for`

6.1 Création de liste en Python, affichage et taille
6.2 Accès aux éléments et modification d'un élément
6.3 Parcours
6.4 Méthodes de liste

Exemple de listes en Python

- Liste contenant des éléments de même type :

```
L1 = [1, 15, -8]
L2 = ["bob", "lulu"]
```

- Liste contenant des éléments de types différents :

```
L3 = ["Python", 76, 3.8, False, "XXI"]
L4 = ['a', [2, True], 22]
```

- En utilisant la fonction `range()` :

```
L5 = list(range(5))
L6 = list(range(3, 7))
L7 = list(range(1, 9, 2))
```

- Liste vide

```
L8 = []
```

Indices des éléments d'une liste

indices négatifs	$-len(L)$	$-len(L)+1$		-2	-1
`L`	élément 1	élément 2	...	élément $n-1$	élément n
indices positifs	0	1		$len(L)-2$	$len(L)-1$

Exercice

Expliquer ce qui se passe lorsque l'on exécute le programme suivant :

```
L = ["Python", 76, 3.8, False, "XXI"]
print(L[4])
L[4] = 21
print(L)
```

Exercice

Expliquer ce qui se passe lorsque l'on exécute les instructions suivantes :

```
>>> L = ["Bob", 76, [2, 3]]
>>> print(L[0][2])

>>> print(L[2][1])

>>> print(L[1][0])
```

Itérer sur les indices

Exemple : affichage des éléments et de la longueur des éléments d'une liste

```
L = ["Zoe", "Thomas", "Manon"]
i = 0
while i < len(L):
    print('longueur de ', L[i], " = ", len(L[i]))
    i += 1
```

Exemple : affichage un à un les caractères d'une chaîne

```
ch = "abcd"
for i in range(len(ch)):
    print(ch[i])
```

Itérer sur les éléments

Exemple : affichage des éléments et de la longueur des éléments d'une liste

```
L = ["Zoe", "Thomas", "Manon"]
for prenom in L:
    print('longueur de ', prenom, " = ", len(prenom))
```

Exemple : affichage un à un les caractères d'une chaîne

```
ch = "abcd"
for car in ch:
    print(car)
```

Méthodes de liste ne renvoyant rien

Exemples :

```
>>> L = [1, 2, 3]
>>> L.append("Python")
>>> print(L)

>>> L = [9, 5, 1, 7]
>>> L.sort()
>>> print(L)

>>> L = [1, 2, 3]
>>> L2 = [4, 5]
>>> L.extend(L2)
>>> print(L)
```

Méthodes de liste ne renvoyant rien

Attention : Contrairement aux chaînes, les méthodes de listes modifient la liste elle-même sans en renvoyer de nouvelle

```
>>> liste1 = [3, 1, 8]
>>> liste2 = liste1.append(7)
>>> print(liste1)

>>> print(liste2)
```

Plan du cours

Chapitre 7 - Notions avancées sur les listes

7.1 Opérateurs de listes

- Opérateur d'extraction : *slice*
- Opérateur d'appartenance : `in`
- Opérateurs `*` et `+`

7.2 Type `list` : un type modifiable

7.3 Passage d'arguments

7.4 Des chaînes aux listes et inversement

Opérateur d'extraction

Quels sont les différents affichages de ces instructions ?

```
>>> L = [15, 3, 6, 7, 58, 4, 0, 3, 1]
>>>print(L[2:4])

>>>print(L[2:])

>>>print(L[1:8:3])

>>>print(L[:4])

>>>print(L[1::3])

>>>print(L[::5])

>>>print(L[:4:-1])
```

Opérateur d'appartenance

Quels sont les différents affichages ?

```
>>> L = ["bob", "tutu",5 ,6 ,10]
>>> 5 in L

>>> 22 in L

>>> ["bob", "tutu"] in L
```

Opérateur *

Quels sont les différents affichages ?

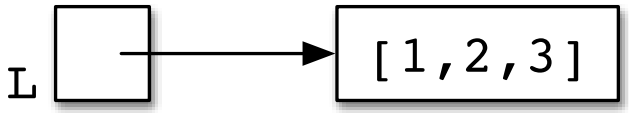
```
>>> print(list(range(3)) * 2)

>>> L = [0] * 5
>>> print(L)
```

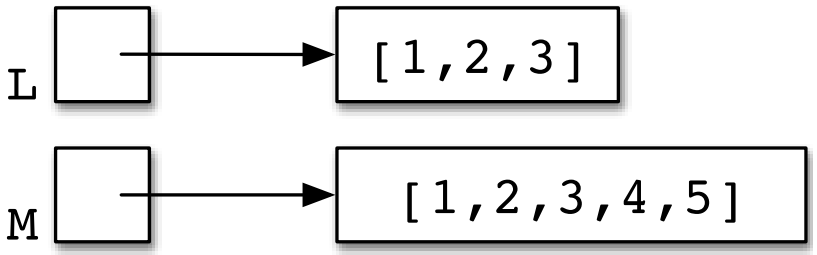
Différence entre `extend()` et l'opérateur `+`

L'opérateur + crée une nouvelle liste :

```
L = [1,2,3]
```

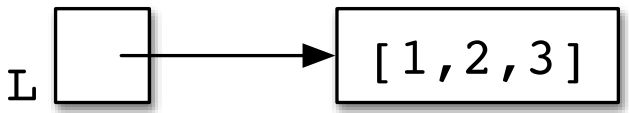


```
M = L + [4,5]
```

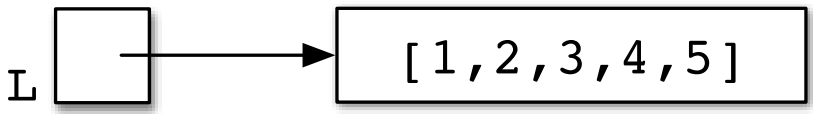


La méthode `extend()` étend une liste existante :

```
L = [1,2,3]
```



```
L.extend([4,5])
```



Tri des éléments d'une liste

Expliquer ce qui va se passer dans le programme ci-dessous :

```
L = [ 5, 1, 8]
L1 = L[:]
L.sort()
print("la liste L est devenue ", L)
L2 = sorted(L1)
print("la liste L1 est devenue ", L1)
print("la liste L2 est ", L2)
```

Attention à ne pas modifier une liste dans une boucle `for` !!

Exemple 1 :

```
L = [1, -2, -3, 4]
for i in L:
    if i < 0:
        L.remove(i)
print(L)
```

Attention à ne pas modifier une liste dans une boucle `for`!!

Exemple 2

```
L = [1, -2, -3, 4]
for i in range(len(L)):
    if L[i] < 0:
        L.remove(L[i])
print(L)
```

Passage d'argument de type `str`

Que va afficher le programme suivant ?

```
def modifierChaine(chaine):
    chaine = chaine + "!"
    print("Chaine dans la fonction : ", chaine)

ch = "bonjour"
modifierChaine(ch)
print("Valeur de ch :", ch)
```

Passage d'argument de type `str`

Que va afficher le programme suivant ?

```
def concat(arg1, arg2):
    arg1 = arg1 + ' ' + arg2
    return arg1

ch1 = 'Monty'
ch2 = 'Python'
concat(ch1, ch2)
```

Passage d'argument de type `list`

Que va afficher le programme suivant et quelle est la valeur référencée par `L` ?

```
def modifList(arg1, arg2):
  arg1 = arg1[:len(arg1)-1] + [arg2]
  return arg1

#programme principal
L = ['a', 1, False]
x = 3
modifList(L,x)
```

Passage d'argument de type `list`

Que va afficher le programme suivant et quelle est la valeur référencée par `L` ?

```
def modif(arg1, arg2):
    arg1.pop()
    arg1.append(arg2)

#Programme principal
L = ['a', 1, False]
x = 3
modif(L,x)
```

Passage d'argument de type `list`

Que va afficher le programme suivant et quelle est la valeur référencée par `L` ?

```
def fonction1(M):
  M = [4] + M[1:]

def fonction2(M):
  M[0] = 4

L = [1, 2, 3]
fonction1(L)
print(L)
fonction2(L)
print(L)
```

Des chaînes aux listes et inversement

Exemple 1 :

```
>>> ch1 = "Le langage Python"
>>> L1 = ch1.split()
>>> print(L1)
['Le', 'langage', 'Python']
>>> print(ch1)
Le langage Python
>>> L2 = ch1.split('a')
>>> print(L2)
['Le l', 'ng', 'ge Python']
```

Exemple 2 :

```
>>> s = "_"
>>> s.join(['Le', 'langage', 'Python'])
'Le_langage_Python'
```

Plan du cours

Chapitre 8 - Portée des variables en Python

8.1 Variables locales

8.2 Variables globales

- Accès à une variable globale dans une fonction
- Modification d'une variable globale dans une fonction
- Conseils d'utilisation

8.3 Quid des listes

Variables locales

Que fait le programme ci-dessous ?

```
def f(x):
    res = 2 * x + 1
    return res

print(f(3))
```

L'instruction `print(res)` peut-elle s'exécuter correctement dans le programme principal ?

Variables locales

Exemple avec 2 variables locales de même nom :

```
def f(x):
    res = 2 * x + 1
    return res

def g(x):
    res = 5 * x + 3
    return res

print(f(3))
print(g(3))
```

Accès à une variable globale dans une fonction

Exemple :

```
def image(x):
    res = a * x + b
    return res

a = 2
b = 1
print(image(3))
```

Priorité aux variables locales

```
def imageV2(x):
    a = 5
    b = 10
    res = a * x + b
    return res

a = 2
b = 1
print(imageV2(3))
print(a, b)
```

Impossible de modifier une variable globale dans une fonction

```
def h():
    vg = vg + 1
    return vg + 1

vg = 1
print(h())
print(vg)
```

... à moins d'utiliser le mot-clé `global`

Exemple :

```
def hV2():
    global vg
    vg = vg + 1
    return vg + 1

vg = 1
print(hV2())
print(vg)
```

Conseil : Utiliser les arguments pour modifier une variable globale

```
def hV3(v):
    v = v + 1
    return v, vg + 1

vg = 1
vg, val_retour = hV3(vg)
print(val_retour)
print(vg)
```

Et quand les variables globales sont de type `list`...

Programme 1 :

```
def ajout1(x):
    M.append(x)

M = [1]
ajout1(2)
print(M)
```

Programme 2 :

```
def ajout2(x):
    M = M + [x]

M = [1]
ajout2(2)
print(M)
```

Exemple :

```
def ajout3(x):
    global M
    M = M + [x]

M = [1]
ajout3(2)
print(M)
```

Plan du cours

Chapitre 9 - Récursivité

9.1 Définition
9.2 Principes généraux d'une fonction récursive
9.3 Compléments sur la récursivité

Exemple : Fonction Python récursive qui `suite(n)` qui prend en argument le rang n et renvoie la valeur U_n

```
def suite(n):
    if n <= 0:
        return 3
    else:
        return 5 * suite(n-1) + 2
```

Fonction factorielle : version itérative

$\text{fac}(n) = n \times (n-1) \times \ldots \times 2 \times 1$

```
FONCTION fac(n TYPE NOMBRE) TYPE NOMBRE
VARIABLES_LOCALES
    cpt, res TYPE NOMBRE
DEBUT
    res <-1
    POUR cpt ALLANT DE 2 A n
        DEBUT
            res <- res*cpt
        FIN
    RENVOYER res
FIN
```

Fonction factorielle : version récursive

$\text{fac}(n) = n \times \text{fac}(n-1)$ avec la valeur initiale $\text{fac}(0) = 1$

```
FONCTION fac_rec(n TYPE NOMBRE) TYPE NOMBRE
DEBUT
    SI (n==0 OU n==1) ALORS
        DEBUT
            RENVOYER 1
        FIN
    SINON
        DEBUT
            RENVOYER n*fac_rec(n-1)
        FIN
FIN
```

Exercice : Écrire la fonction en Python et simuler l'exécution de `fac_rec(3)`

Exercice

Soit la fonction `longueur(ch)` ci-dessous :

```
def longueur(ch):
    ch = ch[1:]
    return 1 + longueur(ch)
```

L'appliquer à la main sur la chaîne `ch="algo"`. Que devrait-elle faire ?

Exercice 1

Sans utiliser de méthode de liste ni de test `in` :

- Écrire une fonction récursive qui permet de tester si l'élément *x* appartient à une liste *L*.

  Cette fonction `appartient(L,x)` aura pour paramètres *L* et *x* et renverra vrai ou faux.
- Simuler l'appel de `appartient([1,3,5,7],5)` et celui de `appartient([1,3,5,7],2)`.

Exercice 2

Sans utiliser de méthode de liste :

- Écrire une fonction récursive qui permet de compter le nombre d'occurrences de l'élément *x* dans une liste *L*.

  Cette fonction `nbOccu(L,x)` aura pour paramètres *L* et *x* et renverra le nombre d'occurrences.
- Simuler l'appel de `nbOccu([1,3,3,7,3,5],3)` et celui de `nbOccu([1,3,5,7],5)`.
- Partant de la fonction précédente, proposer une nouvelle fonction, appelée `sansOccu()` qui renvoie la liste dans laquelle toutes les occurrences de l'élément *x* auront été supprimées.
- Simuler l'appel de `sansOccu([1,3,3,7,3,5],3)` et celui de `sansOccu([1,3,5,7],5)`.

Récursivité terminale

Donner une version itératice de la fonction récursive `CompteRebourg(n)` suivante :

```
def CompteRebourg(n):
    if n == 0:
        print("Partez")
    else:
        print(n)
        CompteRebourg(n-1)
```

Récursivité multiple

```
def suite2(n):
    if n <= 0:
        return 2
    elif n == 1:
        return 5
    else:
        return 3 * suite2(n-1) - suite2(n-2)
```

Récursivité multiple

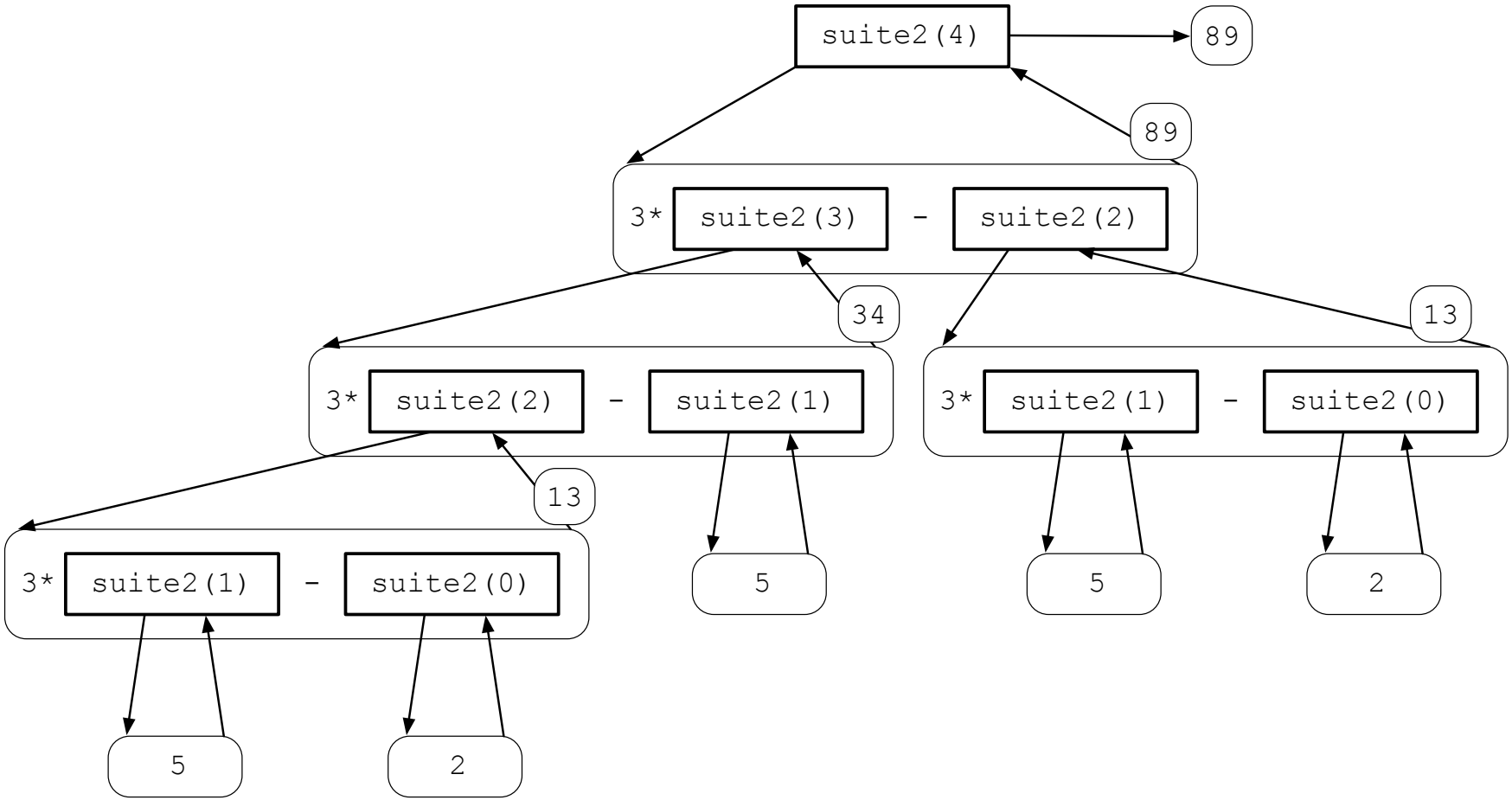


FIGURE – Appels récursifs de `suite2(4)`

Plan du cours

Chapitre 10 - Tuples et dictionnaires

10.1 Tuples en Python

10.2 Dictionnaires en Python

`tuple` : Création et accès

```
>>> x = (1 ,2, "a") # parenthèses optionnelles
>>> x

>>> x[1]

>>> y = (1)

>>> y = (1,)


>>> x[2] = 3
```

Opérateurs sur les valeurs de type `tuple`

```
>>> t = ('a', 'b', 3)
>>> z = x + t
>>> print(z)

>>> 'a' in z

>>> print(z[1:4])
```

Le type `tuple` utile pour l'affectation multiple

```
>>> a = 2
>>> b = 5
>>> z = (a, b)
>>> print(z)

>>> z = (3, 2)
>>> print(a)

>>> print(b)
```

Le type `tuple` utile pour retourner plusieurs valeurs

```
def fct():
    a = 10
    b = 20
    return a, b

v1, v2 = fct()
print(v1, v2)
v = fct()
print(v[0], v[1])
```

Parcours des valeurs d'un `tuple`

```
for x in (1, 2, 3):
    print(x)

z = (2, 3, 'a')
for e in z:
    print(e)
```

Conversion `tuple` vers `list` et inversement

Quels vont être les affichages ?

```
>>> t = (1, 2, 3)
>>> L = list(t)
>>> L

>>> L[0] = 4
>>> L

>>> t = tuple(L)
>>> t
```

Le type `dict` : création et accès

```
agenda = {"Tom": "0606", "Zoe": "0605",
          "Lucie": "0604"}
```

ou

```
agenda = {}
agenda["Tom"] = "0606"
agenda["Zoe"] = "0605"
agenda["Lucie"] = "0604"
```

Comparer deux dictionnaires

```
>>> d1 = {'Tom': 4, 'Agathe': 10}
>>> d2 = {'Agathe': 10, 'Tom': 4}

>>> d1 == d2

>>> d1 != d2
```

Exemple : différence entre les listes et les dictionnaires

```
>>> L = [1, 2, 3]
>>> L1 = [1, 2, 3]
>>> L2 = [2, 1, 3]

>>> L == L1

>>> L == L2
```

Copie de dictionnaires : même chose que pour les listes !

```
>>> d1 = {'Tom': 4, 'Agathe' : 10}
>>> d2 = d1
>>> d2['Billy'] = 12
>>> print(d1)
```

Les dictionnaires sont modifiables !

```
def ajouter(dico, cle, valeur):
    dico[cle] = valeur

d1 = {'Tom': 4, 'Agathe': 10}
ajouter(d1, 'Eliott', 5)
print(d1)
```

Parcours de dictionnaire

```
agenda = {"Tom": "0606", "Zoe": "0605", "Lucie": "0604

for cle in agenda:
    print(cle)
```

Exercices

1. Écrire une fonction qui prend comme argument un dictionnaire et qui affiche, à raison d'une paire par ligne, toutes les clés et toutes les valeurs du dictionnaire.
2. Écrire une fonction qui prend comme argument un dictionnaire, une valeur et une clé et qui ajoute, si la clé n'apparaît pas déjà dans le dictionnaire, la clé et sa valeur associée.
3. Écrire une fonction qui prend comme argument un dictionnaire, et un caractère `c` et supprime du dictionnaire toutes les entrées correspondant à des clés qui commencent par le caractère `c`.

Exemple de liste de dictionnaires

```
listeEtudiant = []
listeEtudiant.append({"nom": "Durand",
                      "prenom": "Agathe",
                      "age": 19,
                      "bac": "S"})
listeEtudiant.append({"nom": "Dupont",
                      "prenom": "Louis",
                      "age": 20,
                      "bac": "S"})
```

Plan du cours

Gestion de fichiers en Python

1. Travailler avec des fichiers
2. Où sont mes fichiers ?
3. Manipuler des fichiers texte
   - Écriture séquentielle dans un fichier
   - Lecture depuis un fichier

Exemple d'utilisation de la fonction `chdir()` (*CHange DIRectory)* du module `os` :

```
>>> from os import chdir
>>> chdir("/users/lazard/Documents/python/")
#ou chdir("Documents/python/")
```

Exercice

Dans l'interpréteur :

1. demander quel est le répertoire courant
2. changer de répertoire pour vous positionner dans votre répertoire `'Documents'`
3. vérifier que vous êtes bien dans ce répertoire
4. changer de répertoire courant pour vous positionner dans le répertoire où vous sauvegardez votre TP
5. vérifier que vous êtes bien dans ce répertoire

Écriture séquentielle dans un fichier

Exemple 1 :

```
f = open("test.txt", "w")
f.write("Bonjour\n")
f.write("Ceci est un test d'ecriture")
f.close()
```

Exemple 2 :

```
f = open("test1.txt", "w")
f.writelines(["ligne 1\n", "ligne 2"+"\n", "FIN"])
f.close()
```

Exercice

Écrire un programme qui demande à l'utilisateur son nom, puis crée un fichier dont le nom sera `"TestNom.txt"` (où `Nom` est celui de l'utilisateur), dans lequel vous écrirez sur une première ligne `"Ceci est un test"`, puis sur une seconde ligne `"Je m'appelle "`, en ajoutant le nom de l'utilisateur, et, sur une dernière ligne, vous écrirez `"Test termine"`. Pour cela vous utiliserez dans un premier temps la méthode `write()`.

Lecture dans un fichier

Exemple 1 :

```
f = open("test.txt", "r")
s = f.read()
f.close()
print(s)
```

Exemple 2 :

```
f = open("test.txt", "r")
t = f.readline()
f.close()
print(t)
```

Exemple 3 :

```
f = open("test.txt", "r")
l = f.readlines()
f.close()
print(l)
```