INF 1

D. Lenne

ALGORITHMIQUE ET PROGRAMMATION

INF1C2P22

université de technologie de compiègne

INF 1

D. Lenne

ALGORITHMIQUE ET PROGRAMMATION

INF1C2P22

université de technologie de compiègne

INF1

Algorithmique et programmation

Dominique Lenne

Printemps 2022

Table des matières

• Diapositives du cours

• Annales d'examens

• Mémento Python

INF1

Algorithmique et programmation

Dominique Lenne

utc
SORBONNE UNIVERSITES

Documentation

Polycopié

- Supports de cours
- Annales d'examens

Plate-forme pédagogique

- Moodle : http://moodle.utc.fr/course/view.php?name=INF1
  - ✓ Equipe pédagogique
  - ✓ Planning
  - ✓ Supports de cours
  - ✓ Enoncés de TD
  - ✓ Annales corrigées
  - ✓ …

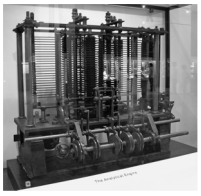
Langage de programmation

- Python : https://www.python.org/

utc
SORBONNE UNIVERSITES

Précurseurs

Al-Khwarizmi (vers 780 – vers 850)
Mathématicien persan du IX^{e} siècle.
A écrit un livre regroupant des méthodes claires, à suivre pas à pas, pour résoudre des problèmes mathématiques.



Ada Lovelace (1815 – 1852)
Femme de sciences anglaise.
A proposé la première série d'instructions exécutables par la machine analytique de Charles Babbage.

D'après : https://interstices.info/famille-algorithmes-programmation/

utc
SORBONNE UNIVERSITES

Objectifs

Savoir écrire un algorithme

- Déterminer une séquence d'actions primitives permettant de réaliser un traitement informatique

Savoir écrire un programme

- Coder un algorithme à l'aide d'un langage, en respectant la syntaxe de ce langage

Langage utilisé

- Python

utc
SORBONNE UNIVERSITES

Exemple

Pb : calcul du périmètre d'un cercle à partir de son rayon

Quelques actions primitives (réalisables par un ordinateur)

- Lire (une valeur entrée au clavier par l'utilisateur)
- Afficher (un message et/ou une valeur à l'écran)
- Calculer la valeur d'une expression
- Affecter une valeur à une variable

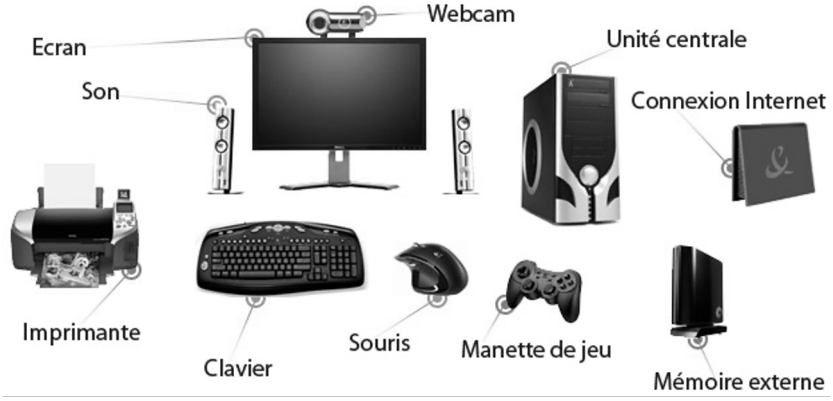
Solution ?

Solution

```
constante
   Pi = 3.1416
variables
   rayon, p : réels
début
    afficher "Rayon du cercle ? "
    lire (rayon)
    p ← 2 * Pi * rayon
    afficher   "Périmètre du cercle :  ", p
fin
```

Structure et fonctionnement d'un ordinateur

L'ordinateur



Architecture d'un ordinateur

Trois constituants essentiels

- Mémoire centrale (mémoire vive)
  - Contient les programmes pendant leur exécution
  - Contient aussi les informations temporaires (données, informations intermédiaires, résultats)
- Processeur (unité centrale)
  - Lit les instructions et les traite
  - Possibilité de branchements
- Périphériques
  - Stockage (permanent)
    - Disque dur, clé USB, lecteur/graveur CD, DVD, …
  - Communication avec l'utilisateur
    - Ecran, Clavier, Souris, …

Système d'exploitation

Programme permettant

- la gestion de la mémoire,
- la gestion des périphériques,
- l'exécution des programmes,
- la gestion des fichiers.

Exemples

- Windows, macOS, Unix, Linux, …

Codage de l'information

Unités de capacité

Unités élémentaires

bit (binary digit) : 0 ou 1

octet (byte) : 8 bits

Unités de capacité

Ko, Mo, Go, To, Po

10^3 , 10^6 , 10^9 , 10^{12} , 10^{15}

Codage de l'information

Dépend de la nature des données

- Booléens : Vrai/Faux
  - 1 bit suffirait
- Caractères
  - Exemple : code ASCII
- Nombres
  - Divers codages possibles
- Images, sons
  - Echantillonnage, Numérisation

utc
SORBONNE UNIVERSITES

Table ASCII
(7 bits : 0-127)

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20	sp	64	40	@	96	60	`
^A	1	01	☺	SOH	33	21	!	65	41	A	97	61	a
^B	2	02	☻	STX	34	22	"	66	42	B	98	62	b
^C	3	03	♥	ETX	35	23	#	67	43	C	99	63	c
^D	4	04	♦	EOT	36	24	$	68	44	D	100	64	d
^E	5	05	♣	ENQ	37	25	%	69	45	E	101	65	e
^F	6	06	♠	ACK	38	26	&	70	46	F	102	66	f
^G	7	07	•	BEL	39	27	'	71	47	G	103	67	g
^H	8	08	◘	BS	40	28	(	72	48	H	104	68	h
^I	9	09	○	HT	41	29	)	73	49	I	105	69	i
^J	10	0A	◙	LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B	♂	VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C	♀	FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D	♪	CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E	♫	SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F	☼	SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10	►	SLE	48	30	0	80	50	P	112	70	p
^Q	17	11	◄	CS1	49	31	1	81	51	Q	113	71	q
^R	18	12	↕	DC2	50	32	2	82	52	R	114	72	r
^S	19	13	‼	DC3	51	33	3	83	53	S	115	73	s
^T	20	14	¶	DC4	52	34	4	84	54	T	116	74	t
^U	21	15	§	NAK	53	35	5	85	55	U	117	75	u
^V	22	16	▬	SYN	54	36	6	86	56	V	118	76	v
^W	23	17	↨	ETB	55	37	7	87	57	W	119	77	w
^X	24	18	↑	CAN	56	38	8	88	58	X	120	78	x
^Y	25	19	↓	EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A	→	SUB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B	←	ESC	59	3B	;	91	5B	[	123	7B	{
^\	28	1C	∟	FS	60	3C	<	92	5C	\	124	7C	|
^]	29	1D	↔	GS	61	3D	=	93	5D	]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^_	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	⌂

ASCII étendu (8 bits)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	▒	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▒	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	n
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Langages

Langage machine

- Les instructions sont des codes binaires
- Ex : 10110000 01100001

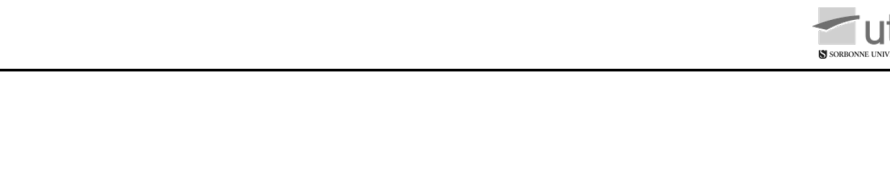
Langage assembleur

- Les instructions sont de type STO, ADD, JMP, MOV
- Ex : mov %al,$0x61

Langages de programmation

- Plus ou moins indépendants de la machine
- Ex : C, C++, Pascal, Python, Java, Lisp, Prolog, Kotlin …
- Ex d'instructions :
  - x =x+1
  - for i in range(10):
    print(i)

Premiers éléments d'algorithmique



Ensemble d' instructions
- Opérations élémentaires
- Enchainement entre ces opérations

Données, constantes, variables

Donnée

- Valeur introduite pendant l'exécution du programme

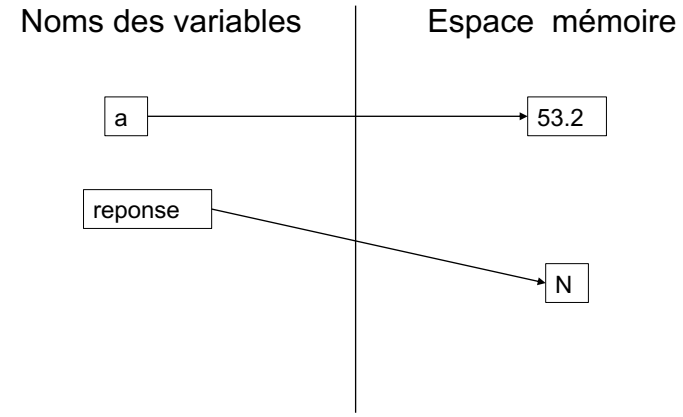
Constante

- Valeur fixe utilisée par le programme

Variable

- Valeur susceptible de changer au cours de l'exécution d'un programme

Représentation en mémoire



Instructions de base

Entrées / sorties

- lire, écrire, afficher

Affectation

de la valeur d'une expression à une variable

- ←

Sélection

- si … alors ... sinon

Répétition

- tant que, répéter … jusque, pour

Affectation

Opération consistant à attribuer à une variable la valeur d'une expression

Exemples

- z ← 1
- resultat ← 2*3 + 5
- solution ← -b/a
- nb ← nb + 1

Sélection

SI <condition> **ALORS**
  <instructions>
SINON
  <instructions>
FSI

Exemple

si a ≥ b alors
  max ← a
sinon
  max ← b
fsi

Répétition

3 possibilités :

Tant que <condition> **faire**   <séquence d'instructions > **Ftq**
Répéter   <séquence d'instructions> **Jusqu'à** <condition>
Pour i **allant de** n1 **à** n2 **faire**   <séquence d'instructions> **Finpour**

Exemples

Exemple 1

répéter

afficher " Entrez un nombre inférieur à 10 "

lire n

jusqu'à n < 10

Exemple 2

s ← 0

pour i **allant de** 1 **à** 100 **faire**

s ← s + i

fin pour

Conception d'algorithmes

Propriétés importantes pour un algorithme

- non ambigu
- combinaison d'opérations élémentaires
- capable de fournir un résultat en un nombre fini d'opérations, quelles que soient les données d'entrée

Méthodes

- Analyse descendante
- Analyse ascendante
- Analyse mixte

Exemple

Algorithme pour déterminer le maximum de deux nombres :

```
variables
   x, y, max : réels
début
   afficher 'Entrez les deux nombres :'
   lire x et y
   si x > y  alors
         max ← x
   sinon
         max ← y
   fsi
   afficher 'Le maximum est', max
fin
```

Deuxième version

Algorithme pour déterminer le maximum de deux nombres (v2) :

```
variables
   x, y, max : réels
début
   afficher 'Entrez les deux nombres :'
   lire x et y
   max ← x
   si y > max alors
         max ← y
   fsi
   afficher 'Le maximum est', max
fin
```

Programmation

Le langage Python

Quelques caractéristiques

Inventeur : Guido Von Rossum

Gratuit, libre, portable, compatible avec les autres langages

Interprété

Utilisable dans de nombreux domaines

- Apprentissage machine, IA, jeux video, 3D

Outils de développement

Installation de Python

- https://www.python.org/
- Pour être guidé, voir éventuellement : https://www.youtube.com/watch?v=HWxBtxPBCAc&feature=youtu.be

idle

- Environnement de développement de base
- Inclus dans la distribution de Python

Autres EDI (inclus dans la distribution anaconda orientée ML)

- Spyder
- PyCharm

L'interpréteur python

idle
(prompt : >>>)

```
>>> 3 + 4
7
>>> x = 5
>>> x
5
>>> x = x+1
>>> x
6
>>> y = 2 * x + 4
>>> y
16
>>>
```

iPython

```
In [1]: 3 + 4
Out[1]: 7

In [2]: x = 5

In [3]: x
Out[3]: 5

In [4]: x = x + 1

In [5]: x
Out[5]: 6

In [6]: y = 2 * x + 4

In [7]: y
Out[7]: 16
```

utc SORBONNE UNIVERSITES

L'interpréteur python (suite)

```
>>> print(y)
16

>>> print('Le carré de ',x,' est', x**2)
SyntaxError: invalid syntax

>>> print('le carré de ',x,' est', x**2)
le carré de  6  est 36

>>> z=input("Entrez un nombre ")
Entrez un nombre 45

>>> z
'45'

>>> int(z) // 9
5

>>> int(z) / 9
5.0
```

Code python de l'algorithme du maximum

```
x = float(input('1er nombre ? '))
y = float(input('2ème nombre ? '))
maxi = x
if y > maxi :
    maxi = y
print('Le maximum est ',maxi)
```

N.B. : en fait une fonction max existe en python :

```
>>> max(4,5)
5
```

Instructions de base

INF1

DOMINIQUE LENNE

utc
SORBONNE UNIVERSITES

Sommaire

Algorithmique

- Expressions
- Sélection
- Entées/Sorties

Programmation Python

- Outils de développement
- Premiers éléments de python
- Entrées/Sorties
- Sélection
- Exercices

utc
SORBONNE UNIVERSITES

Expressions

Une expression est constituée

- d'opérateurs
- d'opérandes (constantes et variables)
- et de fonctions

Exemples

- nbEleves +1
- 10
- 4 / 3 * PI * R ^ 3
- (x > 0) and (x < 1)

Rappels

Affectation

Opération consistant à attribuer à une variable la valeur d'une expression

Exemples

- z ← 1
- resultat ← 2*3 + 5
- solution ← -b/a
- nb ← nb + 1

Sélection

Rappels

SI <condition> **ALORS**
  <instructions>
SINON
  <instructions>
FSI

Exemple

si a ≥ b alors
  max ← a
sinon
  max ← b
fsi

Sélection (suite)

Sans alternative :
SI <condition> ALORS
  <instructions>
FinSI

Exemple : (valeur absolue de x)

```
valAbs ← x
SI x < 0 ALORS
    valAbs ← - x
FinSI
```

Sélection multiples

SI … ALORS

…..

SINON SI … ALORS

….

SINON SI … ALORS

…?.

SINON

….

Entrées / sorties

- **Lire (au clavier, ou dans un fichier)**
- **Ecrire (à l'écran, ou dans un fichier)**

- **Ou Afficher (pour écrire à l'écran)**

Programmation en python

utc
SORBONNE UNIVERSITES

Rappels

Outils de développement

Installation de Python

- https://www.python.org/
- Pour être guidé, voir éventuellement : https://www.youtube.com/watch?v=HWxBtxPBCAc&feature=youtu.be

idle

- Environnement de développement de base
- Inclus dans la distribution de Python

Autres EDI (inclus dans la distribution anaconda orientée ML)

- Spyder
- Pycharm

utc
SORBONNE UNIVERSITES

Hello World

```
>>> print('Bonjour à tous !')
Bonjour à tous !
```

Avec une procédure ?

```
def bonjour() :
    print('Bonjour à tous !')
```

Exécution ?

```
bonjour()
Bonjour à tous !
```

Remarques

- **Une fonction ou une procédure**
  - est définie à l'aide du mot-clé **`def`**
  - a un nom
  - est suivie de parenthèses et du caractère ':'
- **Des paramètres peuvent figurer ou non à l'intérieur des parenthèses** (voir prochain cours)
- **Les instructions doivent être indentées ensuite.**
- **Une fonction retourne une valeur, une procédure n'en retourne pas**
- **Ex de fonction :**

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

Identificateurs

Un identificateur est un nom donné à un élément du programme (par le programmeur)

Règles de formation des identificateurs en python

- Suite alphanumérique
- Ne peut pas commencer par un chiffre
- Ne comporte pas d'espaces
- Peut contenir un tiret de soulignement : _
- Sensible à la « casse »

Exemples

salaireNet ~~salaire-net~~ salaire_net x2 ~~2x~~ moyenne _x

Conventions de nommage

Constantes : toutes les lettres en majuscules

- Ex : NMAX PI AN_MIN

Variables

- Minuscules
  - Ex : v prix choix mois
- si 2 mots, majuscule au début du deuxième mot
  - Ex : nbJours prixVente debutListe
- Ou tiret bas de séparation
  - Ex : nb_jours prix_vente debut_liste

Mots réservés

Ne peuvent pas être redéfinis par le programmeur :

def, return, yield
if, elif, else
while, for, from
or, and, in, not
global
import
try, finally, except, raise
break, continue
del
…

Fonctions standard

print() : affiche le texte en paramètre
input() : retourne la saisie de l'utilisateur (sous forme de texte)
len() : longueur (taille) du paramètre
range(n) : liste des entiers de 0 à n-1
ord(car) : retourne le code correspondant au caractère car

int(x) convertit x en entiers
str(x) convertit x en chaine de caractères

open(), close(), read(), readline() …

Opérateurs

Opérateurs arithmétiques

+ - * /

****** *(puissance)* *II (division entière***)** **%** *(modulo)*

Exemples

```
a = a + 1;
delta = b * b – 4 * a * c;
k = i % j
```

Opérateurs de comparaison

== < > <= >= !=

Exemple

```
if (i % 2 == 0):
    print("i est pair")
```

Opérateurs composés

On peut "composer" l'addition avec l'affectation

Exemple :

x += 1 est équivalent à : x = x + 1

Deux opérations sont composées en une seule (incrémentation de x) :

- ajout de 1 à la valeur de x
- affectation du résultat à x

Les autres opérateurs peuvent être composés avec l'affectation de la même façon :

-= *= /= %= //= **=

Instructions de base

- **Affectation**
  =
- **Sélection**
  if <condition> :
  …
  else :
  …
- **Iteration (voir prochain cours)**
  while
  for
- **Entrées/sorties**
  print()
  input()

Documentation des programmes

Commentaires

```
# Ceci est un commentaire sur une ligne

" " "
Ceci est un commentaire
sur
plusieurs lignes
" " "
```

Les commentaires ne sont visibles que des programmeurs

Ils sont ignorés à l'exécution

Affectation

Syntaxe

<identificateur de variable> = <expression>

Deux rôles :

- Evaluation (calcul) de l'expression
- Puis, affectation (rangement) à la variable (identificateur)

Exemples

```
x =10
nbEleves = nbEleves +1;
delta = b*b –  4*a*c
```

Entrées / Sorties

- **Entrée clavier : input**

```
>>> x=input('Entrez un nombre : ')
Entrez un nombre : 25
>>> x
'25'
```

- **Attention !**

  le type de la valeur retournée est une **chaine de caractères**

Lecture de nombres

- **Pour lire une variable d'un autre type, il faut « transtyper » (ou « caster » anglicisme)**

- **Pour un entier :**

```
x = input('Entrez un nombre : ')
x = int(x)
```

Ou directement :

```
x = int(input('Entrez un nombre : '))
```

- **Pour un réel :**

```
alpha = float(input('Entrez un nombre : '))
```


Affichage

Affichage simple à l'écran :

```
>>> print('Valeur du nombre : ',x)
Valeur du nombre : 25
```

Affichage avec des chaines formatées

La chaine de caractères à afficher est précédée de la lettre f.
Les variables et les expressions se notent entre accolades

```
Exemples :
>>> print(f'Valeur du nombre : {x}')
Valeur du nombre : 25
>>> print(f'Le carré de {x} est {x**2}')
Le carré de 25 est 625
```

Affichage sans retour à la ligne

Sans retour à la ligne

```
x = 3
y = 5
print(x , end='')
print(y)
```

Affichage :

```
3 5
```


Formatage d'un nombre à l'écran

Nombre entier

On précise le nombre total de positions après ':'

Ex : Affichage de la valeur de n sur 6 positions

```
print(f'Valeur de n {n:6}')
```

Nombre réel

On utilise `x.yf (virgule fixe)` ou `x.ye (notation scientifique)` avec x : nb total de positions et y : nb de chiffres décimaux

Ex : Affichage de la valeur de la variable surface sur 6 positions, dont 2 décimales

```
print(f'Surface : {surface:6.2f}')
```

Sélection

```
if  <condition> :
    <bloc d'instructions>
```

ou

```
if  <condition> :
    <bloc d'instructions>
else:
    <bloc d'instructions>
```

Ou encore

```
if  <condition> :
    <bloc d'instructions>
elif <condition> :
    <bloc d'instructions>
…
else:
    <bloc d'instructions>
```

Instruction composée, bloc d' instructions

Ligne d'en-tête :
1ère instruction
…..
….

Bloc d'instructions

Instruction composée

Indentation

Le bloc est indenté par rapport à la ligne d'en-tête

Exemple :

```
if delta == 0 :
    r = - b / (2 * a)
    print ("Une solution double : ", r)
```

Expressions booléennes

Opérateurs : `and` `or` `not` `in`

Valeurs booléennes : `True, False`

Exemples d'expressions booléennes

```
(x > 0) and (x < 100)  # x doit être strictement compris entre 0 et 100

not ((x <= 0) or (x >= 100))  # idem

y in {1, 3, 5, 7, 9}  # True si y impair entre 0 et 10, False sinon
```

utc
SORBONNE UNIVERSITES

Exemple

```
# Grand ou petit

sexe = input("Quel est votre sexe (M/F) ? ")
age = int(input("Quel est votre âge ? "))
taille = int(input("Quelle est votre taille ?"))

femme = sexe == 'F'
homme = not femme
majeur = age >= 18

# les valeurs ci-dessous sont assez arbitraires
if femme :
    petit = taille < 150
    grand = taille > 170
else:
    petit = taille < 160
    grand = taille > 180

print(majeur, femme, homme)
print(age, petit, grand)
```

Exécution
Entrées :
F
24
165
Sortie ?
True True False
24 False False

Choix multiples

Exemple

- Simulation d'une calculatrice 4 opérations
- 2 opérandes, 1 opérateur (+ - * ou /)

utc
SORBONNE UNIVERSITES

```
# calculette

a = float(input("a ? "))
op = input("opérateur ? ")
b = float(input("b ? "))

if (op == '+'):
    resultat = a + b
elif (op == '-'):
    resultat = a - b
elif (op == '*'):
    resultat = a * b
elif (op == '/'):
    resultat = a / b

print(resultat)
```

Exercice

Intérêts d'un livret bancaire sur un an

- Variables
- Données
- Constantes

Algorithme

Lire la *somme initiale* entrée au clavier par l'utilisateur
Multiplier la *somme initiale* par le *taux d'intérêt*
Diviser ce produit par 100
Nommer *intérêts* le quotient obtenu.
Additionner *intérêts* et la *somme initiale*
Nommer *valeur acquise* la somme obtenue
Afficher la *valeur acquise* et les *intérêts*

Algorithme plus formel

Constante

T : réel constant (le taux d'intérêt)

Variables

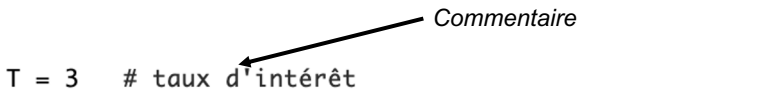
c : réel (le capital initial)

Début

lire c
interets ← T * c /100
valeur ← c + interets
afficher ('Valeur acquise :', valeur, ' intérêts : ', interets)

Fin

En python




```
T = 3   # taux d'intérêt
c = float(input('capital initial'))
interets = c * T / 100
c = c + interets
print('Valeur acquise : ',c,' intérêts : ',interets)
```

utc SORBONNE UNIVERSITES

Itérations

(ou boucles)

INF1

Motivation

Exemple

Ecrire un algorithme permettant de calculer la moyenne de n nombres

- n sera entré par l'utilisateur
- les n nombres aussi

Une première proposition

Lire n # n étant le nombre de nombres
Lire les n nombres
Faire la somme s des n nombres
moyenne ← s / n
Afficher moyenne

Problèmes ?
Algorithme plus détaillé ?

Algorithme plus détaillé

```
variables
   n et i des entiers,
   donnee, moyenne et somme  des réels

afficher (' Combien de données ? ')
lire n
si n>0 alors
   somme ← 0
   pour i allant de 1 à n faire
         lire(donnee)
         somme ← somme + donnee
   fin pour
   moyenne ← somme / n
   afficher ('la moyenne est ', moyenne)
sinon
   afficher ('pas de de données')
fsi
```

Instructions itératives (ou boucles)

Différents types de boucles

Boucles à bornes définies

Pour i allant de .. à …
    < instructions >
Fin Pour

Boucles à bornes indéfinies

Tant que <condition> faire
    < instructions >
Ftq

Répéter
    <instructions>
jusqu'à <condition>

Boucles à bornes définies

Forme générale

Pour v allant de val1 à val2

< instructions >

Fin Pour

En Python

for i in range(n) : # i prend alors les valeurs de 0 à n-1

<bloc d'instructions>

---

for i in range(n, m) : # i prend les valeurs de n à m-1

<bloc d'instructions>

N.B. : la boucle n'est exécutée que si n < m

Différents cas

- Répétition simple
- Utilisation d'un accumulateur
- Utilisation de la variable de boucle
- Imbrication de boucles

Répétition simple

Exemple :

Pour i allant de 1 à 5

Afficher ('Bonjour')

Exécution :

Bonjour
Bonjour
Bonjour
Bonjour
Bonjour

La variable i ne sert que comme compteur

En Python :

```
for i in range(5)
    print('Bonjour')
```

i prend les valeurs de 0 à 4

La variable de boucle n'étant pas utilisée directement, on peut écrire :

```
for _ in range(5)
    print('Bonjour')
```

_ est aussi une variable de boucle qui varie de 0 à 4

Utilisation d'un accumulateur

Exemple : calcul de la somme des n premiers carrés

Algorithme ?

```
afficher 'n ? '
lire n
s ← 0
pour  i allant de 1 à n faire
    s ← s + i * i
fin pour
afficher 'Somme : ', s
```

En python ?

```
n = int (input ('n ? '))
s = 0
for  i in range (1, n+1) :
    s = s + i * i
print ('Somme :', s)
```

Analyse de l'exécution

On numérote les lignes

```
1  n = int (input ('n ? '))
2  s = 0
3  for  i in range (1, n+1) :
4     s = s + i * i
5  print ('Somme :', s)
```

Ligne	Etat	Interactions
1	n 4	n ? Saisie : 4
2	s 0 n 4	
3a 4a	s 0 n 4 i 1 s 1 n 4 i 1	
3b 4b	s 1 n 4 i 2 s 5 n 4 i 2	
3c 4c	s 5 n 4 i 3 s 14 n 4 i 3	
3d 4d	s 14 n 4 i 4 s 30 n 4 i 4	
5	s 30 n 4 i 4	Affichage : Somme : 30

Et la moyenne ?

```
somme  =  0
n = int(input('Combien de nombres ? '))
for i in range(n) :
    donnee = int(input('Entrez un nombre'))
    somme = somme + donnee
moyenne = somme / n

print('moyenne =',moyenne)
```

Utilisation de la variable de boucle

Exemple

On souhaite afficher à l'écran le carré des 5 premiers entiers :

le carré de 1 est : 1
le carré de 2 est : 4
le carré de 3 est : 9
le carré de 4 est : 16
le carré de 5 est : 25

Algorithme ?

Pour i allant de 1 à 5
    afficher 'le carré de ', i, 'est : ', i*i

En Python

```
# Carrés de 1 à 5
for i in range(1, 6):
    print(f'le carré de {i} est {i*i:3}')
```

Exécution ?

```
le carré de  1 est :   1
le carré de  2 est :   4
le carré de  3 est :   9
le carré de  4 est : 16
le carré de  5 est : 25
```

Dans l'ordre inverse

`for i in range(a, b, -1):`

N.B. : la boucle n'est exécutée que si a > b

Exemple

```
for i in range(5, 0, -1):
    print('Le carré de',i,'est',i*i)
```

Exécution

```
Le carré de 5 est 25
Le carré de 4 est 16
Le carré de 3 est 9
Le carré de 2 est 4
Le carré de 1 est 1
```

utc SORBONNE UNIVERSITES

Portée

Attention aux indentations !

```
somme = 0
for i in range (1,5):
  somme = somme + i**2
print(somme)
```

Exécution ?

30

```
somme = 0
for i in range (1,5):
    somme = somme + i**2
    print(somme)
```

Exécution ?

1
5
14
30

Imbrication de boucles

Exemple : Comment afficher une matrice unité ?

```
1 0 0 0 0 0
0 1 0 0 0 0
0 0 1 0 0 0
0 0 0 1 0 0
0 0 0 0 1 0
0 0 0 0 0 1
```

Algorithme

Afficher les lignes ?

Pour i allant de 1 à n
    Afficher la ligne i

Afficher la ligne i ?

Pour j allant de 1 à n
    Si j=i alors afficher ('1')
        sinon afficher ('0')

 # affichage d'une matrice unité;

```
N=8

for i in range(N):
    for j in range(N):
        if j == i:
            print('1', end=' ')
        else:
            print('0', end=' ')
    print()
```

Boucles à bornes indéfinies

utc
SORBONNE UNIVERSITES

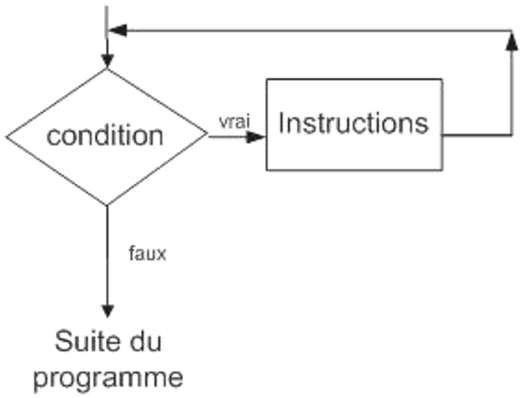
Boucles à bornes indéfinies

Tant que <expression> **faire**
    <instruction>
Ftq

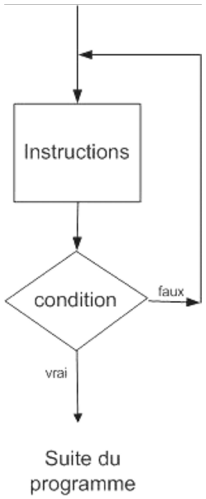
Répéter
  <instruction>
Jusqu'à <expression>

utc
SORBONNE UNIVERSITES

Tant que



Répéter … jusqu'à



En Python

- **Boucle « tant que »**

  while <condition> **:**

  <bloc d'instructions>

- **La boucle « Répéter … jusqu'à … »**
  n'existe pas en python mais on peut la simuler

Exemple

Contrôle d'une saisie :

On demande à l'utilisateur de rentrer un nombre strictement inférieur à 10. Comment contrôler ?

répéter
    afficher " Entrez un nombre inférieur à 10 "
    lire n
jusqu'à n < 10

En python ?

Problème : la boucle « `répéter … jusqu'à …` » n'existe pas

Solution ?

Une première saisie avant la boucle while avec inversion de la condition :

```
n = int (input ('Entrez un nombre inférieur à 10 : '))
while   (n >= 10) :
    print(n,  " n'est pas inférieur à 10")
    n = int (input ('Entrez un nombre inférieur à 10 : '))
```

Autre exemple

```
# lecture d'un nombre pair
fini = False
while not fini :
    n = int(input('Entrez un nombre pair '))
    fini = n % 2 == 0
```

Simulation d'une boucle pour

Comment afficher le carré des n premiers entiers avec une boucle Tant que ou Répéter … jusqu'à ?

le carré de 1 est : 1
le carré de 2 est : 4
le carré de 3 est : 9
le carré de 4 est : 16
le carré de 5 est : 25

Algorithme ?

carrés avec while

```
N = 5

i = 1
while i <= N:
    print ('Le carré de ', i, 'est', i*i)
    i = i + 1
```

Remarques :
- Le programme doit gérer l'incrémentation de i
- La boucle for est plus adaptée dans ce cas

Style de codage en Python

INF1

utc
SORBONNE UNIVERSITES

Style de codage

Défini dans PEP8 :

https://www.python.org/dev/peps/pep-0008

Principales recommandations :

- Utiliser des indentations de 4 espaces et pas de tabulation
- Utiliser des lignes vides pour séparer les fonctions ou pour scinder de gros blocs de code à l'intérieur de fonctions
- Lorsque c'est possible, placez les commentaires sur les lignes commentées
- Utiliser des espaces autour des opérateurs et après les virgules, mais pas juste à l'intérieur des parenthèses
- Exemple : a = f(1, 2) + g(3, 4)
- N'utiliser que des caractères ASCII pour vos noms de variables
- Utiliser des minuscules_avec_trait_bas pour les noms de fonctions. Le camelCase est possible aussi
- Faites en sorte que les lignes ne dépassent pas 79 caractères, au besoin en insérant des retours à la ligne.

utc
SORBONNE UNIVERSITES

Retours à la ligne

**Pour couper une ligne de code utiliser le caractère : **

```
>>> x = 'on peut couper une chaine de \
caractères si elle est trop \
longue'
>>> x
'on peut couper une chaine de caractères si elle
est trop longue'
```

Il est possible aussi de couper n'importe quelle ligne de code avec ce moyen

Types

Chaines de caractères

INF1

Dominique Lenne

utc
SORBONNE UNIVERSITES

Sommaire

Notion de type

Type entier

Type réel

Type caractère

Type chaîne de caractères

- Fonctions et opérations sur les chaînes
- Méthodes de la classe string

Traitement des erreurs

utc
SORBONNE UNIVERSITES

Notion de type

- Un **type** définit la nature des valeurs que peut prendre une donnée ou une variable, ainsi que les opérateurs qui peuvent lui être appliqués.
- Dans de nombreux langages de programmation, les variables doivent être « déclarées » et associées à un type. Par exemple en langage C, une variable x de type entier doit être déclarée de la façon suivante : `int x;`
- **En python, le typage est « dynamique »**
  - il n'y a pas de déclarations
  - le type d'une variable est inféré à partir de sa valeur
- **La fonction type** permet de connaître le type d'une donnée ou d'une variable

Exemples

```
>>> type('bonjour')
<class 'str'>

>>> message = 'bonjour'
>>> type(message)
<class 'str'>

>>> n = 43
>>> type(n)
<class 'int'>
```

Remarques :

- On peut changer le type d'une variable en changeant sa valeur

```
>>> n = 3.5
>>> type(n)
<class 'float'>
```

Principaux types

Entier

int (pour integer)

Réel

float

Complexe

complex

Chaines de caractères

str (pour string)

Caractère

str (en python, un caractère est une chaine à un caractère)

Booléen

bool (pour boolean)

N.B. : nous verrons d'autres types dans la suite du cours (e.g. list)

Le type entier

- Type **int** en python
- Exemple

```
>>> t = 0
>>> type (t)
<class 'int'>
```

- Python peut traiter des entiers de taille illimitée (sauf par la taille de la mémoire)
- Dans d'autres langages, les entiers sont représentés sur 4 ou 8 octets (Plus grand entier : 2^31 – 1 ou 2^63 -1)
- Mais, en python, le temps de traitement des opérations augmente pour de très grands nombres

Le type réel

- Type **float** en python (virgule "flottante")
- Un nombre est de type float s'il contient un point ou une puissance de 10

  Exemples : 3.14159 -273.15 1.83e+17

- La taille maximale dépend de la machine sur laquelle le programme est en cours d'exécution.

  Exemple :

  max =1.7976931348623157 e+308

  min = 2.2250738585072014 e-308

Caractères

Diversité des caractères

- Lettres majuscules
- Lettres minuscules
- Caractères de ponctuation
- Espace
- Chiffres
- Caractères non imprimables

Problème des caractères nationaux

- Caractères accentués

Table ASCII
(7 bits : 0-127)

Ctrl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01	☺	SOH
^B	2	02	☻	SIX
^C	3	03	♥	EIX
^D	4	04	♦	EOI
^E	5	05	♣	ENQ
^F	6	06	♠	ACK
^G	7	07	•	BEL
^H	8	08	◘	BS
^I	9	09	○	HI
^J	10	0A	◙	LF
^K	11	0B	♂	VI
^L	12	0C	♀	FF
^M	13	0D	♪	CR
^N	14	0E	♫	SO
^O	15	0F	☼	SI
^P	16	10	►	SLE
^Q	17	11	◄	CS1
^R	18	12	↕	DC2
^S	19	13	‼	DC3
^I	20	14	¶	DC4
^U	21	15	§	NAK
^V	22	16	▬	SYN
^W	23	17	↨	EIB
^X	24	18	↑	CAN
^Y	25	19	↓	EM
^Z	26	1A	→	SIB
^[	27	1B	←	ESC
^\	28	1C	∟	FS
^]	29	1D	↔	GS
^^	30	1E	▲	RS
^_	31	1F	▼	US

Dec	Hex	Char
32	20	sp
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂

ASCII étendu
(8 bits)

Dec	Hex	Char
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	ö
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	₧
159	9F	ƒ

Dec	Hex	Char
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	▒
177	B1	▒
178	B2	▒
179	B3	│
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐

Dec	Hex	Char
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	─
197	C5	┼
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	═
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	█
220	DC	▄
221	DD	▌
222	DE	▐
223	DF	▀

Dec	Hex	Char
224	E0	α
225	E1	ß
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	µ
231	E7	τ
232	E8	Φ
233	E9	Θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	φ
238	EE	ε
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
245	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	∙
250	FA	·
251	FB	√
252	FC	n
253	FD	2
254	FE	■
255	FF	

Fonctions de conversion (entre caractère et code)

Fonction ord

- ord (caractère) renvoie un entier correspondant au code ASCII (et plus généralement UTF-8)

Exemples :
ord('A') vaut 65, ord('a') vaut 97, ord ('€') vaut 8364

Fonction chr

- chr (entier) renvoie le caractère correspondant au code UTF-8 de l'entier

Exemples :
chr(65) vaut 'A', chr(8364) vaut '€'

utc SORBONNE UNIVERSITES

Conversion Minuscule-Majuscule

Soit *c* une variable de type *string*

Si sa valeur correspond à une lettre minuscule, comment la convertir en majuscule ?

```
if (c >= 'a') and (c <= 'z'):
        c = chr(ord(c) - ord('a') + ord('A'))
```

N.B. : la "méthode" upper() de la classe string permet de faire cette conversion :
c = c.upper()

utc SORBONNE UNIVERSITES

Chaines de caractères

Définition

- Une chaîne de caractères est une suite de caractères regroupés dans une même variable
- En python, le type est str (pour string)

```
>>> type('hello')
<class 'str'>
```

Remarques

- Il s'agit d'un type dit « composite »
- On parle aussi de « séquence ». Nous verrons plus tard qu'il existe d'autres types de séquences.

Les chaînes de caractères en python

Deux possibilités pour définir une chaine de caractères :

- Soit des simples quotes : 'Une chaîne de caractères'
- Soit des doubles quotes : "Une chaîne de caractères"
- Mais pas les deux !

Si la chaîne contient une apostrophe, on utilise les doubles quotes :

- "L'étudiant s'attend à être reçu"

Si la chaîne contient des guillemets, on utilise les simples quotes :

- 'Il dit : "je pense être reçu" '

On peut aussi banaliser un caractère à l'aide d'un anti-slash

- "L'étudiant dit \"Je m'attends à être reçu\""

Chaîne vide : " ou ""

Accès à un caractère

- On peut accéder à chaque caractère au moyen d'un index (commençant à 0)

  Exemple

  ```
  s = 'INF1'   # s[1] vaut N
  ```

- Attention ! Les chaînes sont "immuables"
- On ne peut donc pas modifier un caractère directement :

  Exemple

  `s[3] = '2'` provoquerait une erreur

  'str' object does not support item assignment

Extraction d'une sous-chaîne (slicing)

- `chaine[n:m]` # extrait la sous-chaine de `chaine` de l'index n à l'index m, # m non compris

  Exemple

  ```
  >>> ch = 'Université'
  >>> ch[3:7]
  'vers'
  ```

- `chaine[:m]` # extrait la sous-chaine de `chaine` de l'index 0 à l'index m, # m non compris

  ```
  >>> ch[:7]
  'Univers'
  ```

- `chaine[n:]` # extrait la sous-chaine de `chaine` de l'index n à la fin de chaine

  ```
  >>> ch[6:]
  'sité'
  ```

Longueur d'une chaîne

Fonction len()

Renvoie un entier représentant la longueur de la chaîne.

```
>>> len('INF1')
4
>>> s = 'INF1'
>>> len(s)
4
```

Comparaisons de chaînes

Opérateurs

==, <, >, <=, >=, !=

Ordre

lexicographique

Exemples

```
s1 = 'Dupont'
s2 = 'Dupond'
s3 = 'Du pont'
s4 = 'du Pont'
```

Vrai ou faux ?

s2 > s1	False
s3 > s1	False
s4 > s1	True

Concaténation

Définition

La concaténation est une opération qui permet d'accoler 2 ou plusieurs chaines de caractères

Syntaxe Python

s = s1 + s2 + s3 ...

Exemple

```
s1 = 'du'
s2 = 'pont'
s3 = s1 + s2    # s3 vaut 'dupont'
s4 = s1 + ' ' + s2    # s4 vaut 'du pont'
```

utc
SORBONNE UNIVERSITES

Appartenance ou inclusion

On peut tester l'appartenance d'un caractère ou l'inclusion d'une sous-chaîne dans une chaîne à l'aide de "**in**"

Exemples :

```
>>> ch = 'Université'
>>> 'v' in ch
True
>>> 'u' in ch
False
>>> 'Univers' in ch
True
```

utc
SORBONNE UNIVERSITES

Quelques méthodes de la classe string

En python, tout est "objet"

- Un objet regroupe des propriétés et des méthodes

Une chaine de caractère est donc un objet

Quelques méthodes de la classe string :

- `upper(), lower(), replace(old, new), split(), …`
- Exemples :

```
>>> print('bonjour'.upper())
'BONJOUR'
>>> 'toto'.replace('o','i')
'titi'
```

utc
SORBONNE UNIVERSITES

Traitement des erreurs

utc
SORBONNE UNIVERSITES

Problème

Certaines erreurs peuvent se produire à l'exécution

Par exemple, si l'utilisateur ne saisit pas une valeur du type attendu, cela génère une erreur :

```
>>> n=int(input('Entrez un entier : '))
Entrez un entier : 4.7
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    n=int(input('Entrez un entier : '))
ValueError: invalid literal for int() with base 10: '4.7'
```

Comment éviter cela ?

utc
SORBONNE UNIVERSITES

Récupération des erreurs

On peut anticiper certaines erreurs. On parle alors de "gestion des exceptions".

En python, on dispose des instructions try ... except ... else

```
try:
    n=int(input('Entrez un entier : '))
except:
    print('Attention ! Erreur de saisie')
```

Pour répéter la saisie :

```
correct = False
while not correct:
    try:
        n = int(input('Entrez un entier : '))
    except:
        print('Attention ! Erreur de saisie')
    else:
        correct = True
```

utc
SORBONNE UNIVERSITES

Tableaux

INF1

Dominique Lenne

utc
SORBONNE UNIVERSITES

Motivation

Pourquoi a-t-on besoin des tableaux ?

Exemple : tri de n nombres

Comment représenter et stocker les n nombres en mémoire ?

- 1ère solution : n variables (pas très pratique !)
- 2ème solution : tableau de nombres
  - Un seul nom pour tout le tableau
  - Accès aux nombres à l'aide d'un indice

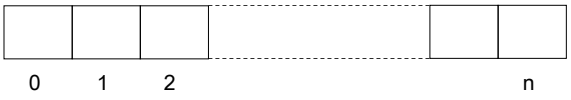
utc
SORBONNE UNIVERSITES

Définition

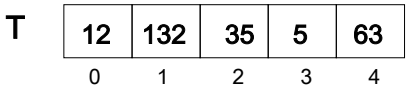
Un tableau est une collection ordonnée d'éléments ayant tous le même type

On accède à chacun de ces éléments individuellement à l'aide d'un indice

Tableaux à une dimension



Exemple : tableau d'entiers



Dimension : 1
Longueur : 5

Valeur de T [1] : 132

Représentation d'un tableau

En python, un tableau peut être représenté par une liste :

- Une liste se note entre crochets
- Les éléments sont séparés par des virgules
- Exemple (tableau d'entiers) :

```
>>> t = [5, 17, 8, 24, 45]
>>> type(t)
<class 'list'>
```

- Tableau vide (2 crochets accolés)
  tab = [ ]

Remarques

- En fait, contrairement à un tableau, une liste en python peut contenir des éléments de types différents
- Exemple : ['Anna', 24, 1.68]

- Mais, pour représenter un tableau, on se limitera à des éléments de **même type**

Taille d'un tableau

len(t) retourne la taille du tableau t
On parle plutôt de longueur (length) pour une liste

Exemples :

```
>>> t = ['a', 'b', 'c', 'd']
>>> len(t)
4

>>> t = []
>>> len(t)
0
```

Accès aux éléments d'un tableau

Accès à l'aide d'un index

```
>>> villes = ['Compiègne', 'Belfort', 'Troyes']
>>> villes[0]
'Compiègne'
```

Contrairement aux chaînes de caractères, les listes sont "mutables" :

Modification du $i^{éme}$ élément du tableau

```
>>> villes[1] = 'Belfort-Montbéliard'
>>> villes
['Compiègne', 'Belfort-Montbéliard', 'Troyes']
```

Initialisation

Initialisation

- initialisation directe (les éléments sont connus)
  Ex: t = [1, 4, -1, 0]
- initialisation à 0

```
>>> tab = [0] * 5
>>> print(tab)
[0, 0, 0, 0, 0]
```

Lecture d'un tableau

On ne peut pas écrire directement :

```
for i in range(10):
    t[i] = int(input("Entrez un entier : "))
```

Ni t, ni sa taille ne sont alors connus

Lecture d'un tableau

Méthode append() : ajout des éléments un à un

```
N = 5
t = []

for i in range(N):
    nombre = int(input("Entrez un entier : "))
    t.append(nombre)
```

Autre possibilité : initialisation préalable du tableau

```
N = 5
t = [0] * N

for i in range(N):
    t[i] = int(input("Entrez un entier : "))
```

Parcours d'un tableau

Affichage d'un tableau

```
>>> t = ['a', 'b', 'c', 'd']
>>> print(t)
['a', 'b', 'c', 'd']
```

Parcours élément par élément

```
>>> for i in range(len(t)):
        print(t[i], end='')
abcd
```

Autre solution : les listes sont des itérables

```
>>> for element in t :
        print(element, end=',')
a,b,c,d,
```

Exemple

Ecrire un programme permettant de calculer la somme de 2 vecteurs de R_N

Exemple **:** R_3

(3.5, 12, -6) + (-1, 6.3, 0) = (2.5, 18.3, -6)

Algorithme ?

```
# somme de deux vecteurs de R3

N = 3
u = []
v = []
w = []

print('1er vecteur')
for i in range(N):
    x = float(input(f'Composante {i} : '))
    u.append(x)


print('2ème vecteur')
for i in range(N):
    x = float(input(f'Composante {i} : '))
    v.append(x)

for i in range(N):
    somme = u[i] + v[i]
    w.append(somme)

print('Somme :', w)
```

utc SORBONNE UNIVERSITES

Concaténation

Opérateur +

Exemple :

```
>>> t = ['Paris', 'Marseille']
>>> t2 = ['Lyon', 'Toulouse', 'Nice']
>>> print(t + t2)
['Paris', 'Marseille', 'Lyon', 'Toulouse', 'Nice']
```

utc SORBONNE UNIVERSITES

Extraction

On peut extraire des parties de liste (comme pour les chaines) :

Exemples :

```
>>> t = ['Paris', 'Marseille', 'Lyon', 'Toulouse', 'Nice']
>>> t[2:5]
['Lyon', 'Toulouse', 'Nice']
>>> t[:3]
['Paris', 'Marseille', 'Lyon']
>>> t[3:]
['Toulouse', 'Nice']
>>> t[:]
['Paris', 'Marseille', 'Lyon', 'Toulouse', 'Nice']
```

Copie d'un tableau

Copie de t dans t2

t2 = t[:]

Dans ce cas la copie est effective. On peut ensuite modifier t2 sans que t ne soit modifié.

Attention ! On serait tenté d'écrire tout simplement :

t2 = t

Mais dans ce cas, t2 et t référencent la même adresse.

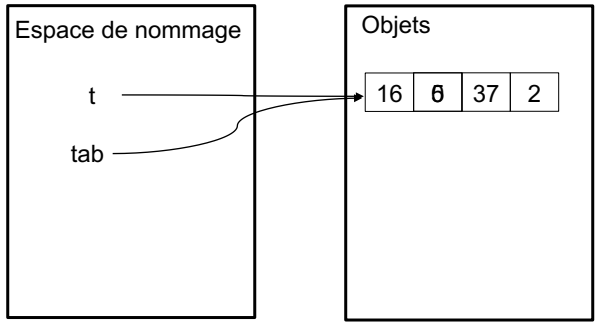
Si un élément est modifié dans t2, il le sera aussi dans t

Exemple :

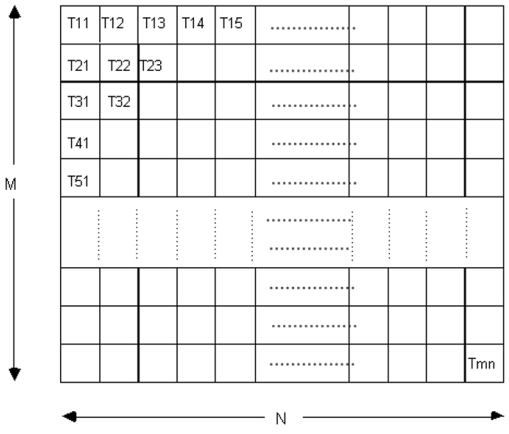
```
>>> t = ['Paris', 'Marseille', 'Lyon', 'Toulouse', 'Nice']
>>> t2 = t
>>> t2[3] = 'Strasbourg'
>>> t2
['Paris', 'Marseille', 'Lyon', 'Strasbourg', 'Nice']
>>> t
['Paris', 'Marseille', 'Lyon', 'Strasbourg', 'Nice']
```

Représentation en mémoire

```
>>> t = [16, 5, 37, 2]

>>> tab = t

>>> tab[1] = 0

>>> tab
[16, 0, 37, 2]

>>> t
[16, 0, 37, 2]
```



Tableaux à 2 dimensions



Les t_{ij} (i = 1 .. M, j = 1 .. N) étant de même type
En fait en python, on commencera à 0 : i = 0 .. M-1, j = 0 .. N-1

Représentation en python

- Tableau à une dimension de lignes (ou de colonnes)
- Liste de listes

Exemple :

```
  2      6      7
 -1     21     12
 10    -23      0
```

```
matrice = [[2, 6, 7], [-1, 21, 12], [10, -23, 0]]
```

Accès à un élément

Accès à l'élément i , j
tab [i][j]

Exemples :

```
tab[2][4] = 5   # affectation
      …
if matrice[i][j] == 0 :  # test
      …
```

Lecture d'une matrice

Il faut initialiser avec une liste vide la matrice et chacune des lignes qui la composent

```
N = 3
M = 2
mat=[]

for i in range(N):
    mat.append([])
    for j in range(M):
        mat[i].append(int(input(f'Elément {i}{j} : ')))


# verification
for ligne in mat:
    print(ligne)
```

Affichage

Directement

```
def afficheMatrice(mat):
    for ligne in mat :
        for element in ligne:
            print(f'{element:6.2f}',end='')
        print()
```

Avec des indices

```
def afficheMatrice2(mat):
    for i in range(N) :
        for j in range(M):
            print(f'{mat[i][j]:6.2f}',end='')
        print()
```

Exercice

Ecrire un programme permettant d'initialiser une matrice unité

Exemple : Matrice unité de dimension 5

1 0 0 0 0

0 1 0 0 0

0 0 1 0 0

0 0 0 1 0

0 0 0 0 1

Compréhension de listes

Exemples :

- **Avec range**

```
>>> [2 * x + 1 for x in range(10)]
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

- **En parcourant un tableau**

```
>>> t = [2 * x + 1 for x in range(10)]
>>> [x * x for x in t]
[1, 9, 25, 49, 81, 121, 169, 225, 289, 361]
```

- **Avec une condition**

```
>>> [x * x for x in t if x * x % 3 == 0]
[9, 81, 225]
```

Fonctions et procédures

INF1

Dominique Lenne

Sommaire

- Notion de fonction (et de procédure)
- Les fonctions en Python
- Structure d'un programme Python
- Importation de modules
- Le module Turtle

Exemple introductif

On dispose de données sur un ensemble d'individus : taille, poids, âge, …

Comment calculer :

- la taille moyenne,
- le poids moyen,
- l'âge moyen ?

Solution ?

1ère solution

- Calcul de la somme des tailles
- Division du résultat par le nombre d'individus
- Calcul de la somme des poids
- Division du résultat par le nombre d'individus
- Calcul de la somme des âges
- Division du résultat par le nombre d'individus

```
tailles = [171, 165, 187]
poids = [62, 53, 85]
ages = [24, 21, 35]

somme = 0
n = len(tailles)
for i in range(n):
    somme = somme + tailles[i]
moyenne_tailles = somme / n

somme = 0
n = len(poids)
for i in range(n):
    somme = somme + poids[i]
moyenne_poids = somme / n

somme = 0
n = len(ages)
for i in range(n):
    somme = somme + ages[i]
moyenne_ages = somme / n

print(f'Taille moyenne : {moyenne_tailles:6.2f} \
        Poids moyen : {moyenne_poids:6.2f} \
        Age moyen : {moyenne_ages:6.2f}')
```

Portions de code analogues

Exécution :

Taille moyenne : 174.33 Poids moyen : 66.67 Age moyen : 26.67

Une meilleure solution ?

Ecrire une fonction permettant de calculer la moyenne de n nombres

Passer les valeurs des tailles, poids et âges en paramètres

Fonctions et procédures

Une fonction permet de définir un traitement autonome

- nommé par un identificateur
- appelable par cet identificateur

Une fonction retourne généralement une valeur

Lorsqu'une fonction ne retourne pas de valeur, on parle parfois de procédure

En python, seule la notion de fonction existe

utc
SORBONNE UNIVERSITES

Objectifs

Eviter la répétition d'instructions

correspondant à des traitements analogues
(Cf. exemple de la moyenne)

Structurer les programmes

Exemple :

- lecture
- calcul
- affichage

utc
SORBONNE UNIVERSITES

Pour la moyenne

```
Fonction moyenne(tab : tableau de réels) : réel
Début
    somme ← 0
    n = taille(tab)
    pour chaque élément e de tab
        somme ← somme + e
    moyenne ← somme / n
    retourner moyenne
Fin

Début
    lecture ou initialisation des tableaux tab_tailles,
                                    tab_poids, tab_ages
    moyenne_taille ← moyenne(tab_tailles)
    moyenne_poids ← moyenne(tab_poids)
    moyenne_ages ← moyenne(tab_ages)

    afficher moyenne_taille, moyenne_poids, moyenne_ages
Fin
```

utc SORBONNE UNIVERSITES

Les fonctions en python

Nous avons déjà rencontré des fonctions en python :

Ex : print(), input(), len(), range(), chr(), ord(), …

Les méthodes associées à des objets sont des fonctions :

Ex : ch.upper(), tab.append()

On peut aussi définir des fonctions

utc SORBONNE UNIVERSITES

Définition d'une fonction en python

Le mot-clé def permet de définir une fonction

- Il est suivi d'un nom de fonction, d'une liste de paramètres formels entre parenthèses, et du caractère ':'
- Les instructions qui définissent la fonction doivent être indentées

Syntaxe

```
def nomFonction(liste de paramètres):
    <bloc d'instructions>
```

Valeur retournée

- Pour retourner explicitement une valeur, on utilise l'instruction **return**
- Si aucune valeur n'est retournée explicitement, la fonction retourne quand même la valeur None

Exemple

Ecrire une fonction retournant le cube d'un nombre x

```
def cube(x) :
    return x**3
```

Quelques exemples d'utilisation

```
y = cube(3)

print(cube(3*t + 1))

volume = 4/3 * Pi * cube(r)
```

Exemple
(nouvelle version de la moyenne)

```
def moyenne(tab):
    n = len(tab)
    somme = 0
    for i in range(n):
        somme = somme + tab[i]
    moy = somme / n
    return moy


tailles = [171, 165, 187]
poids = [62, 53, 85]
ages = [24, 21, 35]

moyenne_tailles = moyenne(tailles)
moyenne_poids = moyenne(poids)
moyenne_ages = moyenne(ages)

print(f'Taille moyenne : {moyenne_tailles:6.2f} \
        Poids moyen : {moyenne_poids:6.2f} \
        Age moyen : {moyenne_ages:6.2f}')
```

Fonction sans paramètre

Une fonction peut ne contenir aucun paramètre

Exemple :

```
def alphabet():
    debut = ord('A')
    for i in range(26):
        print(chr(debut+i), end='')
```

Test :

>>> alphabet()

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Fonctions avec un ou plusieurs paramètres

- Une **liste de paramètres** est constituée d'un ou plusieurs noms entre parenthèses
- Chaque nom correspond à un paramètre. Les paramètres sont séparés par des virgules
- Les paramètres d'une fonction sont parfois appelés paramètres formels
- Un paramètre se comporte comme une variable pour la fonction
- En python, le type de chaque paramètre est déterminé dynamiquement lors de l'appel de la fonction

utc SORBONNE UNIVERSITES

Utilisation d'une fonction

Une fonction qui retourne une valeur s'utilise au niveau d'une expression

Ex :

- m = moyenne(t)
- z = 3 * cube(x) + 1

Une fonction qui ne retourne pas de valeur (procédure) s'utilise au niveau d'une instruction

Ex :

- alphabet()
- (ou, par exemple, exercices en td)

utc SORBONNE UNIVERSITES

Paramètres et arguments

Paramètres : interviennent dans la définition d'une fonction

Ex :

```
def affiche(message): # paramètre : message
    print(message)
def cube(x):          # paramètre x
    return x**3
```

Arguments : utilisés lors de l'appel de la fonction

```
affiche('Bonjour à tous') # argument :'Bonjour à tous'
y = 3
print cube(y)   # argument : y
```

N.B. : on aurait pu écrire x à la place de y

Dans ce cas, l'argument x n'a rien à voir avec le paramètre x. Seule la valeur de l'argument x est transmise au paramètre x

Documentation d'une fonction

Il est souvent utile de "documenter" une fonction à l'aide d'une chaine de caractères (docstring)

Exemple :

```
def cube(x) :
    "Retourne le cube d'un nombre x"
     return x**3

>>> help(cube)
Help on function cube in module __main__:

cube(x)
    Retourne le cube d'un nombre x

>>> cube(
        (x)
        Retourne le cube d'un nombre x
```

Importation de modules

Un module est un fichier qui regroupe un ensemble de fonctions (et éventuellement de variables, de constantes, de classes)

- Pour importer un module : `import` nom_du_module
- Liste des fonctions disponibles : `dir`(nom_du_module)

Exemple :

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'acos
', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor
', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite',
'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan
', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau
', 'trunc']
```

Exemple d'utilisation :

```
r1 = (- b + math.sqrt(delta)) / (2 * a)
perimetre = 2 * math.pi * r
```

Autres formes d'importation

On peut choisir de n'importer qu'un ou plusieurs éléments :

```
Ex : from math import pi, sin, cos, tan
```

On peut aussi importer complètement un module par :

```
from nom_module import *
```

Avantage : utilisation simplifiée

```
from math import *
…
r1 = (- b + sqrt(delta)) / (2 * a)
…
perimetre = 2 * pi * r
```

Inconvénient : tous les noms sont importés. Il faut donc veiller à ne pas les réutiliser

Structure d'un programme Python

- # commentaires
- Importation de modules
- Définition éventuelle de variables globales
- Définition de fonctions
- Corps principal du programme

Les fonctions doivent être définies avant leur utilisation

Une fonction peut faire appel à une autre fonction (imbriquée ou non)

Imbrication de fonctions

On peut imbriquer une fonction dans une autre.

Exemple :

```
def f(x):

    # fonction cube imbriquée dans f1
    def cube(z):
        return z**3

    # corps de la fonction f
    y = 3 * cube(x) + 1
    return y

# corps du programme principal
y = f(2)
print(y)
```

Test :
25

N.B. : la fonction cube n'est pas utilisable dans le programme principal

Variable locale/globale

Une variable du corps principal du programme est dite **globale**
Une variable "déclarée" à l'intérieure d'une fonction est dite **locale**

Exemple :

```
def nbCar(c,ch):
    "Renvoie le nombre de caractères c dans la chaine ch"
    nb = 0
    for car in ch:
        if c == car :
            nb = nb +1
    return nb

# test
mot = 'anticonstitutionnellement'
print(nbCar('n', mot))
```

nb est une variable locale à la fonction nbCar

mot est une variable globale

Portée

Portée d'une variable globale

- Une variable globale peut être utilisée dans tout le programme
- Y compris à l'intérieur d'une fonction (mais c'est à éviter !)

Portée d'une variable locale

- Une variable locale n'a d'existence que dans la fonction où elle est définie
- Sa portée s'étend toutefois aux fonctions imbriquées

Et si une variable locale a le même nom qu'une variable globale ?

- C'est la variable locale qui l'emporte
- Tout se passe comme si on avait 2 variables différentes

Local ou global ?

Eviter autant que possible les variables globales

Une procédure ou une fonction doit

- effectuer la tâche qui lui a été confiée,
- en ne modifiant que l'état de ses variables locales.

Le module turtle

Le module `turtle` permet de dessiner des graphiques à l'aide des fonctions suivantes :

- reset() : effacer l'écran
- goto(x, y) : aller à l'endroit de coordonnées x, y
- forward(n) : avancer de n
- backward(n) : reculer de n
- color(couleur) : retourne ou fixe la couleur trait ou remplissage ('green', 'red', 'blue', …)
- fillcolor(couleur) : retourne ou fixe la couleur de remplissage
- left(angle) : tourner à gauche d'un angle donné (en degrés)
- right(angle) : tourner à droite
- begin_fill : commencer à remplir un contour fermé à l'aide de la couleur sélectionnée
- end_fill : terminer le remplissage
- width(epaisseur) : choisir l'épaisseur du trait
- up() : lever le crayon (et pouvoir avancer sans dessiner)
- down() : baisser le crayon

Exemple

Module figures (avec une seule fonction pour l'instant)

```
from turtle import *

def carre(cote, couleur):
    "fonction qui dessine un carré de côté et de couleur donnés"
    color(couleur)
    begin_fill()
    for i in range(4):
        forward(cote)
        right(90)
    end_fill()
```

Utilisation (pour dessiner 5 carrés bleus)

```
from figures import carre
from turtle import forward, up, down

for i in range(5):
    carre(50, 'blue')
    up()
    forward(60)
    down()
```

utc SORBONNE UNIVERSITES

Compléments

INF1

Dominique Lenne

utc SORBONNE UNIVERSITES

Sommaire

Le module random

print et return

Typage des paramètres

Variables locales

Passage de paramètres

Le module random

Le module random donne accès à différentes fonctions produisant des nombres aléatoires

En particulier :

- random.random()
  retourne un nombre décimal aléatoire entre 0 et 1
- random. randrange(m)
  retourne un entier compris entre 0 et m (m non compris)
- random.randrange(n, m)
  retourne un entier compris entre n et m (m non compris)
- `random.randint(n, m)`
  retourne un nombre entier aléatoire entre n et m (**m compris**)

Exemple

Tirage simplifié du loto (sans numéro complémentaire)

```
import random

t = []
for i in range(6):
    nouveau = False
    while not nouveau:
        n = random.randint(1, 49)
        nouveau = True
        for c in t:
            if n == c:
                nouveau = False

    t.append(n)

print(t)
```

print et return

Attention à ne pas confondre print et return.

- Exemple avec print

```
def cube(x):
    print(x**3)    # dans ce cas la valeur retournée est None

>>> x = 2
>>> y = 3 * cube(x) + 1

Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    y = 3 * cube(x) + 1
TypeError: unsupported operand type(s) for *: 'int' and 'NoneType'
```

- Avec return

```
def cube(x):
    return x**3

>>> x = 2
>>> y = 3 * cube(x) +1
>>> 25
```

Variables locales / globales

Rappels

Variable globale

- Déclaration dans le programme principal
- Portée : tout le programme

Variable locale

- Déclaration à l'intérieur d'une fonction
- Portée : fonction où elle est déclarée (et dans les fonctions imbriquées s'il y en a)

Variables locales

Exemple : fonction factorielle

```
def fact(n):
    f = 1
    for i in range(2, n+1):
        f = f * i
    return f
```

f est une variable locale à la fonction fact. Elle n'existe que pendant l'exécution de cette fonction. Elle disparaît ensuite.

Test

```
>>> fact(5)
120
>>> f
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    f
NameError: name 'f' is not defined
```

Paramètres et arguments

Rappels :

- Paramètre (ou paramètre formel) :
  Intervient dans la définition de la fonction
  Est utilisé comme une variable dans le corps de la fonction
- Argument (ou paramètre réel) :
  Est utilisé dans l'appel de la fonction
  Peut être une expression ou une variable

Exemple

```
def cube(x) :            # x est le paramètre de la fonction cube
    return x * x * x

t = 2

y = 3 * cube(t)      # t est l'argument. Sa valeur est transmise à x

z = cube(3 * t + 1)  # 3*t+1 est l'argument. Sa valeur est transmise à x
```

Typage des paramètres

En python, les paramètres sont typés dynamiquement.

Exemple :

```
def foo(x, y):
    return x + y
```

L'opérateur + dépend du type des arguments transmis

Test

```
>>> foo('bonjour', ' ca va')
'bonjour ca va'
>>> foo(3, 5)
8
>>> foo(3.5, 4.2)
7.7
>>> foo(['a', 'b', 'c'], ['d', 'e'])
['a', 'b', 'c', 'd', 'e']
```

Passage de paramètres

On distingue généralement deux modes de passage de paramètres :

- Par valeur

  Seule la valeur de l'argument est transmise au paramètre. L'argument ne peut donc pas être modifié.
- Par référence (ou par adresse, ou encore par variable)

  Lorsque l'argument est une variable, tout se passe comme si la fonction travaillait directement avec cette variable. L'argument peut alors être modifié.

En python, aucun de ces deux modes ne s'applique vraiment :

- Il s'agit plutôt d'un passage par référence d'objet
- Lorsque l'argument est une variable, il ne peut en général pas être modifié par la fonction (ou la procédure)
- En revanche, les éléments d'une liste (et donc d'un tableau) peuvent être modifiés

utc SORBONNE UNIVERSITES

Passage de paramètre en python

Si l'argument est une variable globale ayant le même nom que le paramètre correspondant, la variable globale n'est généralement pas modifiée

(sauf éventuellement pour une liste)

Exemple :

```
def plus4(x):      # ici x est le paramètre de la fonction plus4
    x += 4
    return x

x = 3              # ici x est une variable globale
print(plus4(x))
print("Après l'appel de la fonction, x vaut", x)
```

Test :

```
7
Après l'appel de la fonction, x vaut toujours 3
```

Pour modifier la variable globale x :

```
x = plus4(x)
```

x vaut alors 7

utc SORBONNE UNIVERSITES

Modification d'un tableau (effet de bord)

Exemple

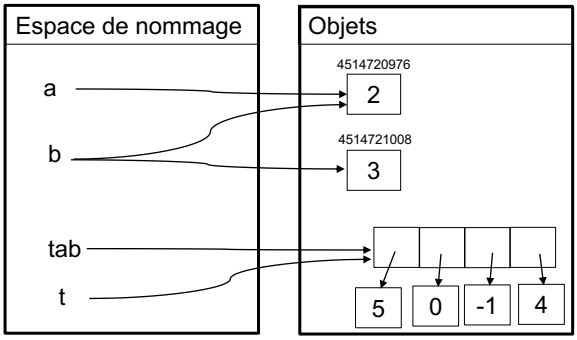
```
# modification d'un tableau dans une fonction

def replace(old, new, tab):
    "remplace les valeurs old par new dans tab"
    for i in range(len(tab)):
        if tab[i] == old:
            tab[i] = new

tab = ['et', 'ou', 'ni', 'mais', 'hors', 'car', 'donc']
replace('hors', 'or', tab)
print(tab)
```

Test

```
['et', 'ou', 'ni', 'mais', 'or', 'car', 'donc']
```

Représentation en mémoire

```
>>> a = 2
>>> b = a
>>> id(a)
4514720976
>>> id(b)
4514720976
>>> b = b + 1
>>> id(a)
4514720976
>>> id(b)
4514721008
>>> tab = [5, 0, -1]
>>> t = tab
>>> t.append(4)
>>> tab
[5, 0, -1, 4]
```

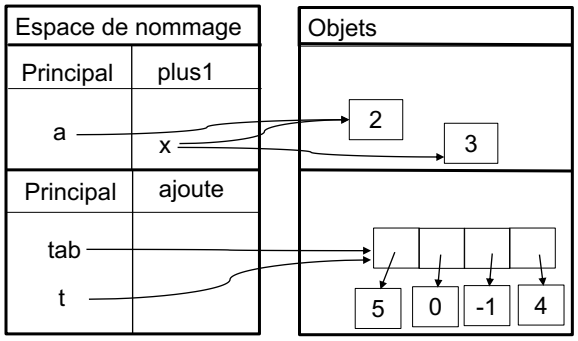


N.B. : La fonction id retourne l'identifiant (adresse) de l'objet associé au nom

Représentation en mémoire

```
def plus1(x):
    x += 1
    print(x)
Test :
>>> a = 2
>>> plus1(a)
3
>>> a
2
```

```
def ajoute(elt, t):
    t.append(elt)
Test :
>>> tab = [5, 0, -1]
>>> ajoute(4, tab)
>>> tab
[5, 0, -1, 4]
```



N.B. : a n'est pas modifié, alors que tab l'est

A retenir

Une variable ne peut en général pas être modifiée lorsqu'on la passe en paramètre

Elle peut être modifiée si son type est "mutable" (ou muable)

Un tableau passé en paramètre peut être modifié. Il s'agit alors d'un "effet de bord".

Les effets de bord sont à proscrire. Ils nuisent à la lisibilité du programme et à la réutilisabilité des fonctions et procédures.

Structures de données

Dictionnaires

INF1

Dominique Lenne

Sommaire

Bilan sur les types

- Types simples et types composites
- Types composites en Python

Tuples

Dictionnaires

Types simples et types composites

Types simples

- Entier : int
- Réel : float
- Booléen : bool
- Caractère

Types Composites

- Chaine de caractères : string
- Tableau : list, tuple
- Structure : dict (dictionnaires), class (classe)
- Ensemble : set

Remarques :

- La correspondance n'est pas toujours exacte (e.g. une liste est plus générale qu'un tableau)
- Les types set et class seront étudiés dans un prochain cours

utc SORBONNE UNIVERSITES

Types composites en python

Séquences : chaines, listes, tuples

- suites ordonnées d'éléments
- accessibles par un index (nombre entier)

Dictionnaires

- éléments non ordonnés
- accessibles à l'aide d'une "clé"

Ensembles

- éléments non ordonnés, uniques
- seront vus dans un prochain cours

utc SORBONNE UNIVERSITES

Tuples

Un tuple

- est semblable à une liste
- mais n'est pas modifiable (mutable)

Collection d'éléments

- séparés par des virgules
- encadrés de préférence par des parenthèses

Exemples :

```
coord = (3, 5, 1)
axes = ('x', 'y', 'z')
```

Les parenthèses ne sont pas indispensables mais conseillées pour la lisibilité du code

Opérations sur les tuples

Similaires aux opérations sur les listes

```
>>> t = (4, -5, 1, 0, -3)
>>> len(t)
5
>>> print(t[1:4])
(-5, 1, 0)
```

Mais les tuples ne sont pas modifiables

```
>>> t[:2] = (-5, 4)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    t[:2] = (-5, 4)
TypeError: 'tuple' object does not support item assignment
```

L'ajout (append) et et le retrait (del) sont donc impossibles

Pour modifier, il faut créer un nouveau tuple

```
>>> t = (-5, 4) + t[2:]
>>> print(t)
(-5, 4, 1, 0, -3)
```

Structures

Problème

Comment modéliser une entité ayant plusieurs caractéristiques de types différents ?

Exemples :

Personne
- Nom
- Prenom
- Age
- …

Voiture
- Marque
- Type
- Cylindree
- …

Définitions

Une variable de type **structure** est une variable composée de plusieurs champs

Les **champs** sont les attributs ou caractéristiques de la structure

Remarque

- Tableau : éléments de même type
- Structure : les champs peuvent être de types différents

En Python, on peut utiliser des **dictionnaires** ou des objets (à voir plus tard)

N. B. : si les éléments sont ordonnés, on peut dans certains cas utiliser des listes ou des tuples

Le type dictionnaire

Ensemble de paires clé-valeur

- Clés
  Type accepté : tout type non modifiable (chaine de caractères, entier, réel, tuple)
- Valeurs
  Type accepté : tout type (valeurs numériques, chaines, listes, tuples, dictionnaires, fonctions, classes, instances)

Les dictionnaires sont modifiables (mutables)

Syntaxe

- Un dictionnaire se note entre accolades
- Les éléments sont séparés par des virgules
- Chaque élément est une paire d'objets (clé : valeur) séparés par ":"

Création d'un dictionnaire

Exemple avec des clés de type chaine de caractères

```
d = {"marque":"Renault", "modèle":"Kadjar", "cv":7}
```

Exemple avec des clés de type entier

```
d = {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Les clés sont : 0, 1, 2, 3 , 4. Les valeurs sont les carrés des clés

- Remarque : définition en compréhension possible dans ce cas

```
>>> d = { x*x : x for x in range(5)}
>>> print(d)
{0: 0, 1: 1, 4: 2, 9: 3, 16: 4}
```

Création d'un dictionnaire vide

```
d = {}
```

Remarque : il existe aussi un constructeur dict() lorsque les clés sont des chaines :

```
d = dict(marque = "Renault", modèle = "Kadjar", cv = 7)
```

Opérations sur les dictionnaires

dico = {"marque":"Renault", "modèle":"Kadjar", "cv":7}

Accès à la valeur associée à une clé par : dico[clé]

```
>>> print(dico["marque"])
'Renault'
```

Ajout : dico[nouvelleClé] = valeur

```
>>> dico = {'a':1, 'b':2, 'd':4}
>>> print(dico)
{'a': 1, 'b': 2, 'd': 4}
>>> dico['c'] = 3
>>> print(dico)
{'a': 1, 'b': 2, 'd': 4, 'c': 3}
```

Suppression : del dico[cle]

```
>>> del dico['d']
>>> dico
{'a': 1, 'b': 2, 'c': 3}
```

utc SORBONNE UNIVERSITES

Et si la clé n'existe pas ?

dico = {"marque":"Renault", "modèle":"Kadjar", "cv":7}

Accès à une clé qui n'existe pas

```
>>> print(dico["cylindrée"])
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    print(dico["cylindrée"])
KeyError: 'cylindrée'
```

Tentative de destruction d'une clé-valeur qui n'existe pas

```
del dico['couleur']        # génère aussi une erreur
```

Solution : tester l'existence de la clé : in

```
if "couleur" in dico: print(dico["couleur"])
```

Ou utiliser la méthode `get(key, default)`

```
>>> print(dico.get("couleur","Couleur non précisée"))
Couleur non précisée
```

utc SORBONNE UNIVERSITES

Autre exemple

```
d = {"nom" : "Da Vinci", "naissance":1452, "mort":1519}
```

- Clés : "nom", "jour", "naissance", "mort"
- Valeurs : "Da Vinci", 1452, 1519

Tests :

```
>>> type(d)
<class 'dict'>
>>> print(d["naissance"])
1452

>>> if not "prénom" in d:
        d["prénom"] = "Leonardo"

>>> for cle in d:
        print(f'{cle} : {d[cle]}'

nom : Da Vinci
naissance : 1452
mort : 1519
prénom : Leonardo
```

Clés, valeurs, éléments

Liste des clés : méthode keys()

```
>>> list(dico.keys())
['a', 'b', 'c']
```

Liste des valeurs : méthode values()

```
>>> list(dico.values())
[1, 2, 3]
```

Liste des éléments : méthode items()

```
>>> list(dico.items())
[('a', 1), ('b', 2), ('c', 3)]
```

Parcours d'un dictionnaire

```
for cle in dico:
    print(cle, dico[cle])
```

Ou mieux

```
for cle, valeur in dico.items():
    print(cle, valeur)
```

Test

```
a 1
b 2
c 3
```

Exercice

Compter toutes les occurrences des caractères d'une chaine de caractères donnée

Algorithme ?

- On construit un dictionnaire d dont les clés sont les lettres et les valeurs le nombre d'occurrences
- Pour chaque caractère c dans chaine
  - Si c est une clé de d, incrémenter sa valeur (nombre d'occurrences)
  - Sinon l'initialiser à 1

Tableau de dictionnaires

Création

- Le tableau peut être réparti sur plusieurs lignes. Un dictionnaire par ligne par exemple.
- Attention à la syntaxe : les éléments (dictionnaires) du tableau doivent être séparés par des virgules

Exemple

```
peintres = [{"nom":"Da vinci", "naissance":1452, "mort":1519},
            {"nom":"Michelangelo", "naissance":1475, "mort":1564},
            {"nom":"Raphael", "naissance":1483, "mort":1520},
            {"nom":"Bellini", "naissance":1430, "mort":1516}]
```

Accès au champ du $i^{ème}$ élément du tableau : d[i][cle]

Exemple : affichage du nom du $3^{ème}$ élément du tableau ?

```
>>> print(peintres[2]["nom"])
Raphael
```

Dictionnaires et listes

Les dictionnaires et les listes sont modifiables

Les clés d'un dictionnaire sont analogues aux indices d'un tableau mais :

- Ajout de `elt` à un tableau : `tab.append(elt)`
- Ajout de `cle:valeur` à un dictionnaire : `dico[cle] = valeur`

Les dictionnaires ne sont pas des séquences

Eléments non ordonnés (contrairement à un tableau)

- Slice impossible : print(dico[1:3]) provoque une erreur
- Intéressant pour l'ajout non consécutif de données

Ex :
```
td = {}
td[25] = 'Claire'
td[3] = 'Zheqin'
td[10] = 'Maria'
```

Test
```
>>> print(td)
{25: 'Claire', 3: 'Zheqin', 10: 'Maria'}
```

Copie de dictionnaires

Attention !

Comme pour les listes, l'instruction :

```
d = dico      # d étant un dictionnaire
```

ne crée pas une copie de dico, mais seulement de sa référence

d et dico référencent alors le même dictionnaire

Pour faire une copie effective utiliser la méthode : `copy()`

```
dico = {'a':1, 'b':2, 'd':4}
d = dico.copy()
d['c'] = 3
print('Dictionnaire dico',dico)
print('Dictionnaire d',d)
```

Test :
```
Dictionnaire dico {'a': 1, 'b': 2, 'd': 4}
Dictionnaire d {'a': 1, 'b': 2, 'd': 4, 'c': 3}
```

Fichiers

INF1

Dominique Lenne

utc
SORBONNE UNIVERSITES

Problème

Jusqu'à maintenant,

- Entrées / sorties limitées au clavier et à l'écran
- Impossible de traiter des données en grand nombre
- Impossible de conserver les données et résultats produits : après exécution, tout est perdu …

Solution ?

- Lire ou écrire dans un **fichier** (sur un disque dur par exemple)

utc
SORBONNE UNIVERSITES

Définition

Un fichier est

- une collection d'informations
- stockée sur un support physique (disque dur, bande, CD, DVD, clé USB ...).

Un fichier permet de conserver durablement l'information

- Données, résultats, programmes, …

L'information persiste à l'arrêt du programme.

Différents types de fichier

- **Fichiers texte**

  Ne contiennent que des caractères imprimables (et des marqueurs de fin de ligne ou de fin de fichier)

- **Fichiers binaires**

  Contiennent des octets. Lorsqu'on enregistre des données, il faut conserver leur type pour pouvoir les relire.

Python utilise par défaut des fichiers texte

- On peut toutefois utiliser aussi des fichiers binaires
- Le module pickle permet d'enregistrer les données en conservant leur type;

Fichiers de texte

Texte

- Suite de caractères alphanumériques (et éventuellement de marqueurs)
- Ensemble de lignes

Marqueur de fin de ligne ?

- Un ou deux caractères spéciaux suivant le système d'exploitation
  - Windows : CR LF (0D, 0A en hexadécimal) ou \r \n
  - Linux ou macOs : LF seulement
- Python unifie cela avec \n qui fonctionne pour tout système d'exploitation

utc SORBONNE UNIVERSITES

Principe général d'accès aux fichiers

1. Ouverture du fichier et création d'un "objet fichier"
   - Lecture
   - Ecriture
   - Ajout
2. Instruction(s) de lecture ou d'écriture (ou de positionnement)
3. Fermeture du fichier

utc SORBONNE UNIVERSITES

Fichiers texte en Python

Ouverture

```
objetFichier = open(nom_fichier, mode, encoding = '…')
```

N.B. : l'encodage est optionnel. S'il n'est pas précisé, c'est l'encodage de la plateforme qui est utilisé.
Pour l'interopérabilité, il est préférable d'utiliser 'utf-8' : encoding = 'utf-8'

Mode

- Lecture : 'r'
- Ecriture : 'w'
- Ajout : 'a'

Exemple : ouverture du fichier 'my_file.txt' en lecture

```
infile = open('my_file.txt', 'r', encoding = 'utf-8')
```

Remarque

Si le fichier n'est pas dans le répertoire courant, il faut indiquer le chemin d'accès complet (ou relatif). Ex :

```
infile = open('D:/my_dir/my_file.txt', 'r')
```

utc SORBONNE UNIVERSITES

Principe général

1. **Ouverture du fichier** et création d'un "objet fichier"
   - Lecture
     ```
     file = open('my_file.txt', 'r')
     ```
   - Ecriture
     ```
     file = open('my_file.txt', 'w')
     ```
     Le fichier est créé s'il n'existe pas et écrasé sinon
   - Ajout
     ```
     file = open('my_file.txt', 'a')
     ```
     Les enregistrements sont ajoutés à la fin du fichier
2. Instruction(s) de **lecture** (ou itération sur chaque enregistrement du fichier)
   ou d'écriture
3. Fermeture du fichier
   ```
   file.close()
   ```

utc SORBONNE UNIVERSITES

Lecture séquentielle d'un fichier texte

Principe général

- Boucle de lecture : on lit successivement des éléments jusqu'à ce que la fin du fichier soit atteinte
- Question : comment détecter la fin du fichier ?

Plusieurs modes possibles

- Un ou plusieurs caractères à la fois
- Ligne par ligne
- Lecture du fichier complet en une seule fois

utc
SORBONNE UNIVERSITES

Algorithmes de lecture

Lecture avec détection de la fin de fichier

```
Lire une ligne
Tant que non fin-de-fichier
    # traitement de la ligne
    …
    Lire une ligne
Fin tant que
```

Lecture si le fichier est un itérable

```
Pour chaque ligne du fichier
    lire une ligne
    # traitement de la ligne
    …
Fin Pour
```

utc
SORBONNE UNIVERSITES

Lecture ligne à ligne en python

Fin de fichier

```
ligne = ""
```

Lecture avec la méthode readline()

```
infile = open('poeme.txt','r')
ligne = infile.readline()
while ligne != "":
    print(ligne)
    ligne = infile.readline()
infile.close()
```

Pour enlever \n :
`ligne = ligne.strip()`
Ou
`ligne = ligne[:-1]`

Chaque ligne se termine par \n.
=> une ligne vide après chaque ligne.

Test

```
C'est un trou de verdure où chante une rivière

Accrochant follement aux herbes des haillons

D'argent. Où le soleil de la montagne fière
```

Lecture avec readlines

Exemple

```
infile = open('poeme.txt','r')
lignes = infile.readlines()
for ligne in lignes:
    print(ligne)
infile.close()
```

Remarque

readlines() retourne les lignes du fichier dans une liste

Exemple

```
inputfile = open('villes','r')
print(inputfile.readlines())
inputfile.close()
```

Test

```
['Paris\n', 'Marseille\n', 'Lyon\n', 'Toulouse\n', 'Nice\n']
```

L'objet fichier est un itérable

Un objet fichier est un "itérable" en python

On utilisera donc de préférence la méthode suivante :

```
infile = open('my_file.txt', 'r')
for line in infile:
     # traitement de chacune des lignes
     …
infile.close()
```

Exemple

```
inputfile = open('poeme.txt')
for line in inputfile:
    print(line)
inputfile.close()
```

Et si le fichier n'existe pas ?

Dans ce cas une erreur se produit.

Il faut donc sécuriser la lecture en interceptant cette erreur

Exemple

```
try:
    inputfile = open('poeme.txt')
    for line in inputfile:
        print(line)
    inputfile.close()
except IOError:
    print("Erreur : le fichier n'existe pas")
```

Fonction de test

On peut aussi écrire une fonction pour tester l'existence d'un fichier

```
def existe(filename):
    try:
        f = open(filename, 'r')
        f.close()
        return True
    except:
        return False

nom = input('Nom du fichier : ')
if existe(nom):
    print("Ce fichier existe")
else:
    print("Ce fichier n'existe pas")
```

utc SORBONNE UNIVERSITES

Ecriture dans un fichier

Exemple

Saisie d'un texte par l'utilisateur et écriture dans un fichier

Pb : convention d'arrêt pour l'utilisateur

- Ligne vide par exemple

Algorithme ?

Lire une ligne
Tant que ligne non vide
    Ecrire la ligne
    Lire ligne suivante

N.B. : Si le texte contient une ligne vide, il faut une autre convention

utc SORBONNE UNIVERSITES

Ecriture des éléments d'un tableau

Exemple : écriture d'un tableau de chaines de caractères

```
t = ['Paris', 'Marseille', 'Lyon', 'Toulouse', 'Nice']

out = open('villes','w')
for element in t:
    out.write(element)
out.close()
```

Contenu du fichier après écriture

```
ParisMarseilleLyonToulouseNice
```

Ecriture avec séparateur de ligne

On ajoute \n :

```
t = ['Paris', 'Marseille', 'Lyon', 'Toulouse', 'Nice']

out = open('villes','w')
for element in t:
    out.write(f'{element}\n')
out.close()
```

Contenu du fichier après écriture

```
Paris
Marseille
Lyon
Toulouse
Nice
```

Ecriture de données de types divers

Pour lire et écrire des données d'un autre type que string, il faut les convertir.

- Ex : écriture de notes dans un fichier

```
t = [12.5, 12, 10, 9.5, 8.25, 17.75, 15]

outputfile = open('notes','w')
for note in t:
    outputfile.write(str(note)+'\n')
outputfile.close()
```

- Lecture des notes et calcul de la moyenne

```
somme = 0
nb = 0
infile = open('notes', 'r')
for note in infile:
    somme += float(note)
    nb += 1
infile.close()

moyenne = somme / nb
print('Moyenne : ', moyenne)
```

utc SORBONNE UNIVERSITES

Exercice

Ecrire une fonction permettant de recopier un fichier texte en enlevant les lignes commençant par #

```
def sansCommentaires(source, destination):
    "enlever les lignes commençant par #"
    if existe(source) and not existe(destination):
        inputfile = open(source, 'r')
        outputfile = open(destination, 'w')
        for line in inputfile:
            if line[0] != '#':
                outputfile.write(line)
        inputfile.close()
        outputfile.close()
    return
```

utc SORBONNE UNIVERSITES

Récursivité

INF1

Dominique Lenne

utc
SORBONNE UNIVERSITES

Définition

Une fonction ou une procédure est dite récursive s'il est fait appel à cette fonction ou à cette procédure dans le corps d'instructions qui la définit.

i.e., la fonction (ou la procédure) s'appelle elle-même.

utc
SORBONNE UNIVERSITES

Exemple

Fonction factorielle

- n! = n*(n-1)* …….1

Mais aussi

- n! = n * (n-1)! ← Formulation récursive
- n! = 1 si $0 \leqslant n \leqslant 1$ ← Condition d'arrêt

Factorielle : algorithme

```
si (n > 1) alors
    fact ← n * fact (n - 1)
sinon
    fact ← 1
fsi
retourner fact
```

Factorielle récursive en Python

```
def factorielle(n):
    " facorielle de n, pour n >= 0 "

    if n > 1 :
        return n * factorielle(n - 1)
    else:
        return 1
```

Tests :

```
>>> factorielle(0)
1
>>> factorielle(5)
120
>>> factorielle(20)
2432902008176640000
```

Somme des n premiers entiers

1 + 2 + …+ (n-1) + n

Fonction itérative ?

Fonction récursive ?

Somme des n premiers entiers

Fonction itérative

```
Fonction somme(n:entier):entier
Variables
  s, i : entier
Début
  s = 0
  pour i allant de 1 à n faire
     s = s + i
  retourner s
Fin
```

Somme des n premiers entiers

Fonction récursive ?

Définition récursive : S(n) = S (n - 1) + n

Condition d'arrêt : S(0) = 0

```
def somme(n):
    if n == 0 :
        return 0
    else:
        return somme(n-1) + n
```

```
somme(4)
début
| appel somme(3) {début de l'empilement}
|   début
|   |   appel somme(2)
|   |      début
|   |      |   appel somme(1)
|   |      |      début
|   |      |      |   appel somme(0) {fin de l'empilement}
|   |      |      |      début
|   |      |      |      |   somme ← 0
|   |      |      |      fin {début du dépilement}
|   |      |      |   somme ← somme (0) + 1 = 0 + 1 = 1
|   |      |      fin
|   |      |   somme ← somme (1) + 2 = 1 + 2 = 3
|   |      fin
|   |   somme ← somme (2) + 3 = 3 + 3 = 6
|   fin
| somme ← somme (3) + 4 = 6 + 4 = 10
fin {fin du dépilement}

Résultat : somme (4) = 10
Fin
```

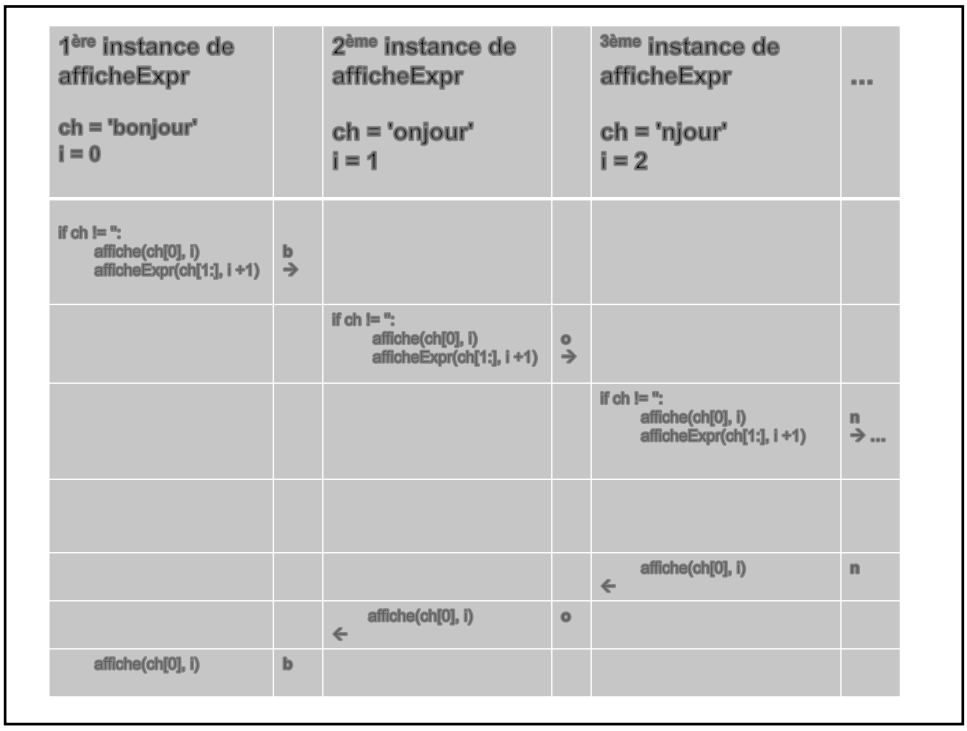
Exercice

Qu'affiche ce programme si l'utilisateur saisit : "bonjour" ?

```
def affiche(car, i):
    "Affiche car avec i espaces avant"
    print(' '*i, car)

def afficheExpr(ch, i):
    if ch != '':
        affiche(ch[0], i)
        afficheExpr(ch[1:], i + 1)
        affiche(ch[0], i)

chaine = input("Entrez une chaine de caractère : ")
afficheExpr(chaine, 0)
```

Exécution :

```
b
 o
  n
   j
    o
     u
      r
      r
     u
    o
   j
  n
 o
b
```



Exercice

Ecrire une fonction récursive permettant de calculer cos(x) et sin(x)

Principe ?

- Exprimer en fonction du cosinus et/ou du sinus d'un nb plus petit

Condition d'arrêt ?

Principe

sinus

Si x est petit alors
sin(x) ≃ x
sinon
sin(x) = 2 sin(x/2) * cos(x/2)

cosinus

Si x est petit
cos(x) ≃ 1
sinon
cos(x) = $\cos^2(x/2) - \sin^2(x/2)$

utc
SORBONNE UNIVERSITES

```
EPS = 1E-6

def sinus(x):
    if x <= EPS:
        return x
    else:
        return 2 * sinus(x/2) * cosinus(x/2)

def cosinus(x):
    if x <= EPS:
        return 1
    else:
        return cosinus(x/2) ** 2 - sinus(x/2) ** 2
```

N.B. : les fonctions du module math sont beaucoup plus efficaces.

```
from math import sin, cos
```

Exercice

- Ecrire une fonction récursive qui retourne une chaine de caractères constituée uniquement des voyelles d'une chaine *ch* passée en paramètre.

Principe

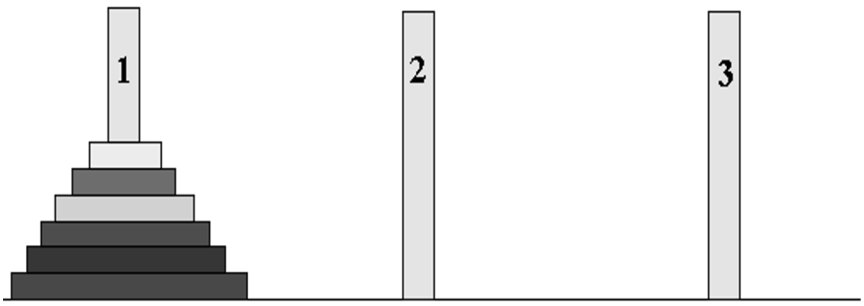
Soit `voyelles(ch)` **cette fonction :**

```
si ch est vide alors
   retourner vide
sinon
   si 1er caractère de ch est une voyelle alors
      retourner 1er caractère + voyelles de la suite de ch
   sinon
      retourner voyelles de la suite de ch
   finsi
finsi
```

Code python

```
def voyelles(ch):
    if ch == '' :
        return ''
    else:
        if ch[0] in "aeiouy":
            return ch[0] + voyelles(ch[1:])
        else:
            return voyelles(ch[1:])
```

Tours de Hanoï



BUT *:* Déplacer la pile de la **tour 1** à la **tour 3**, en ne déplaçant qu'un disque à la fois, et en s'assurant qu'aucun disque ne repose sur un disque de plus petite dimension.

Algorithme

Déplacer la pile des n-1 premiers disques de la tour 1 à la tour 2

Déplacer le dernier disque de la tour 1 à la tour 3

Déplacer les n-1 disques de la tour 2 à la tour 3

Condition d'arrêt : nb de disques = 0 (il n'y a plus de disque à déplacer)

```
def hanoi(nbDisques, t1, t3, t2):
    if nbDisques > 0 :
        hanoi(nbDisques - 1, t1, t2, t3)
        print('Déplacer disque de ', t1,' à ', t3)
        hanoi(nbDisques - 1, t2, t3, t1)
```

Arrêt quand
il n'y a plus de disque

Test :

hanoi(4,1,3,2)

```
Déplacer disque de  1  à  2
Déplacer disque de  1  à  3
Déplacer disque de  2  à  3
Déplacer disque de  1  à  2
Déplacer disque de  3  à  1
Déplacer disque de  3  à  2
Déplacer disque de  1  à  2
Déplacer disque de  1  à  3
Déplacer disque de  2  à  3
Déplacer disque de  2  à  1
Déplacer disque de  3  à  1
Déplacer disque de  2  à  3
Déplacer disque de  1  à  2
Déplacer disque de  1  à  3
Déplacer disque de  2  à  3
```

Comparaison des temps de calcul

```
def factorielleRec(n):
    " factorielle (version récursive) de n, pour n >= 0 "

    if n > 1 :
        return n * factorielleRec(n - 1)
    else:
        return 1

def factorielleIt(n):
    " factorielle (version itérative) de n, pour n >= 0 "

    fact = 1
    for i in range(2, n + 1):
        fact = fact * i
    return fact

def temps(f, n):
    from time import time
    t0 = time()
    f(n)
    t1 = time()
    print('temps : ',f.__name__,round(1E6 * (t1 - t0),5),' microsecondes')

n = int(input("n ? "))
temps(factorielleRec, n)
temps(factorielleIt, n)
```

Résultats

n	15	100	900
récursive	19,00	80,61	1196,62
itérative	13,59	21,71	288,40
math.factorial	1,19	5,00	48.16

Remarques :

- Valeurs indicatives en microsecondes (dépendant de la machine)
- Nb d'appels récursifs limités. Pour connaître la limite :

```
import sys
sys.getrecursionlimit()
```

Algorithmes de tri

INF1

Dominique Lenne

utc
SORBONNE UNIVERSITES

Problème

Trier un tableau d'éléments de même type

Le type des éléments doit être muni d'une relation d'ordre

Exemple

Avant : 8 12 5 35 21 3
Après : 3 5 8 12 21 35

Algorithme (s) ?

N.B. : dans la suite de ce cours, les exemples seront présentés à l'aide de tableaux d'entiers ou de chaines de caractères. Les algorithmes peuvent cependant être appliqués à tout type de données muni d'une relation d'ordre.

utc
SORBONNE UNIVERSITES

Tri par sélection

Principe

- On suppose qu'on connaît le nb d'éléments *n*
- On cherche le minimum
- On le place en premier élément (on échange)
- On cherche le minimum suivant
- On le place en 2ème position
- Et ainsi de suite

Simulation

8 12 5 35 21 3
3 12 5 35 21 8
3 5 12 35 21 8
3 5 8 35 21 12
3 5 8 12 21 35
3 5 8 12 21 35

Algorithme ?

Pour i allant de 1 à n-1

- Chercher le minimum
  - dans la partie du tableau commençant à l'indice i
  - (avec son indice)
- Permuter l'élément d'indice i avec le minimum

Algorithme

```
pour i allant de 1 à n-1
  indiceMin ← i
  pour j allant de i+1 à n
     si t[j] < t[indiceMin] alors
          indiceMin ← j
     fin si
   fin pour
   echanger t[i] et t[indiceMin]
fin pour
```

N.B. : le tableau est modifié. On dit qu'il s'agit d'un tri "en place".

Complexité ?

Complexité

Temps de calcul du tri par sélection

- En fonction du nombre n d'éléments à trier

Il faut comparer $t[i]$ à $t[i+1], t[i+2] \ldots t[n]$

- soit $n - i$ comparaisons à chaque passage dans la boucle interne
- donc $\sum_1^{n-1}(n-i)$ soit $\frac{n*(n-1)}{2}$

La complexité est donc en $O(n^2)$

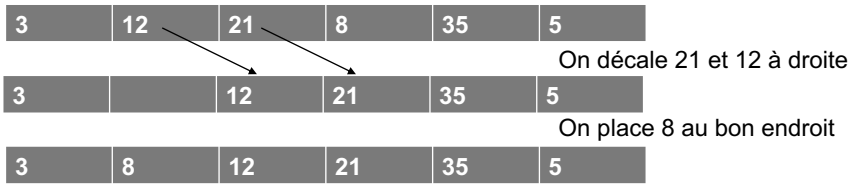
Tri par insertion

Principe

- On met le minimum en tête
- On considère qu'une partie des éléments est ordonnée (seul un élément est ordonné au début)
- On insère au bon endroit

Exemple

- Si les 3 premiers éléments sont bien placés. Placement de 8 :



Simulation (1)

8 12 5 35 21 3
3 12 5 35 21 8
3 5 12 35 21 8
3 5 12 35 21 8
3 5 12 21 35 8
3 5 8 12 21 35

Simulation (2)

35	12	21	8	3	5
3	12	21	8	35	5
3	12	21	8	35	5
3	8	12	21	35	5
3	8	12	21	35	5
3	5	8	12	21	35

Algorithme

```
On cherche le min, on le met au début.
Pour i allant de 2 à n-1
  elt ← T[i+1]
  k ← i
  tant que elt < T[k] faire   {on décale à droite
      T[k+1] ← T[k]            les éléments > = elt }
      k ← k-1
  fin tant que
  T[k+1] ← elt              {on met elt au bon endroit}
fin pour
```

Complexité ? **$O(n^2)$**

En python

```python
def triInsert(t):

    minAuDebut(t)  # fonction qui permute t[0] et le minimum

    for i in range(1, len(t)-1):
        elt = t[i+1]
        k = i
        while elt < t[k]:
            t[k+1] = t[k]
            k -= 1

        t[k+1] = elt

t = ['Rayane', 'Armand', 'Mathieu', 'Clara', 'Mehdi', 'Runzhao']
triInsert(t)
print(t)
```

Test :

```
['Armand', 'Clara', 'Mathieu', 'Mehdi', 'Rayane','Runzhao']
```

Tri d'un tableau de dictionnaires

Exemple

```
t = [{'id':'03','name':'Rayane'},{'id':'08','name':'Armand'},{'id':'35','name':'Mathieu'},
    {'id':'12','name':'Clara'}, {'id':'05','name':'Mehdi'},{'id':'20','name':'Runzhao'}]
```

Il n'y a pas de relation d'ordre entre les dictionnaires.

- Il faut trier en fonction d'une clé (ici 'id' ou 'name')

Fonctions auxiliaires

```
def cle_id(x):
    return x['id']

def cle_name(x):
    return x['name']
```

Ajout d'une fonction de tri

```
def triInsert(t, cle):

    minAuDebut(t, cle)  # fonction qui permute t[0] et le minimum
                        # en fonction de la cle
    for i in range(1, len(t)-1):
        elt = t[i+1]
        k = i
        while cle(elt) < cle(t[k]):
            t[k+1] = t[k]
            k -= 1

        t[k+1] = elt
```

Exemple :

```
def cle_id(x):
    return x['id']
```

```
>>> triInsert(cle_id)
>>> print(t)
[{'id': '03', 'name': 'Rayane'}, {'id': '05', 'name': 'Mehdi'}, {'id': '08', 'name
': 'Armand'}, {'id': '12', 'name': 'Clara'}, {'id': '20', 'name': 'Runzhao'}, {'id
': '35', 'name': 'Mathieu'}]
```

Tri par échange (tri à bulles)

Principe

- On échange les éléments 2 à 2 en les réordonnant

=> Les éléments mal classés remontent dans la liste comme des bulles à la surface d'un liquide

Efficacité

- Dépend du tableau initial
- Efficace si le tableau est presque trié

Simulation (1)

8 12 5 35 21 3
8 5 12 21 3 35
5 8 12 3 21 35
5 8 3 12 21 35
5 3 8 12 21 35
3 5 8 12 21 35

Algorithme

```
nbElts ← n          {nb elts restant à classer }
echange ← vrai   {echange est vrai s'il y a eu un échange}
tant que echange faire
   echange ← faux
   max ← nbElts
   pour i allant de 1 à max-1 faire
      si T[i] > T[i+1] alors
         echanger T[i] et T[i+1]
         echange ← vrai
         nbElts ← i          { nbElts = dernier elt permuté }
      fsi                    {à partir de NbElts, le tri est correct }
   fpour
ftq
```

Complexité ? **$O(n^2)$ en moyenne**

Simulation (2)

8 5 3 12 35 21	max = 6
5 3 8 12 21 35	max = 5
3 5 8 12 21 35	max = 1

Simulation

Interstices

http://interstices.info/jcms/c_6973/les-algorithmes-de-tri

Tri rapide

Tri récursif

Principe

On choisit une valeur pivot

On sépare dans le tableau les éléments inférieurs à cette valeur et les éléments supérieurs

On trie ensuite chacune des deux parties du tableau en suivant le même principe

Tri rapide

```
tri_rapide(t, premier, dernier)
   début
     si premier < dernier alors
       i ← partitionner(t, premier, dernier)
       tri_rapide(t, premier, i - 1)
       tri_rapide(t, i + 1, dernier)
     fin si
   fin
```

```
fonction Partitionner(t, premier, dernier)
i,j,v : entier

v ← t[dernier]            { pivot }
i ← premier; j ← dernier-1
tant que i <= j faire
   tant que i < dernier et t[i] <= v faire i ← i + 1
   tant que j >= premier et t[j] >= v faire j ← j - 1
   si i < j alors echanger t[i] et t[j]
fin tant que
t[dernier] ← t[i]
t[i] ← v
retourner i
```

Complexité ? **En moyenne : O(n * log n)**

Dans le pire des cas : O(n^2)

Les tris en Python

Fonction sorted(t)

- retourne un tableau trié sans modifier le tableau initial
- Exemple :

```
>>> t = [8, 12, 5, 35, 21, 3]
>>> print(sorted(t))
[3, 5, 8, 12, 21, 35]
>>> t
[8, 12, 5, 35, 21, 3]
```

Méthode t.sort()

- modifie le tableau directement

```
>>> t = [8, 12, 5, 35, 21, 3]
>>> t.sort()
>>> t
[3, 5, 8, 12, 21, 35]
```

utc SORBONNE UNIVERSITES

Avec une fonction de tri

On précise la fonction à l'aide du paramètre key :

```
t = [{'id':'03','name':'Rayane'},{'id':'08','name':'Armand'},{'id':'35','name':'Mathieu'},
     {'id':'12','name':'Clara'}, {'id':'05','name':'Mehdi'},{'id':'20','name':'Runzhao'}]

def ident(x):
    return x['id']

def name(x):
    return x['name']

t.sort(key=ident)
print(t)
```

utc SORBONNE UNIVERSITES

Introduction à la programmation orientée objet

INF1

Dominique Lenne

utc
SORBONNE UNIVERSITES

Classes et objets

Nous avons déjà rencontré les notions de classe et d'objet

En Python :

- les variables sont des objets,
- le type d'une variable correspond à une classe
- à laquelle sont associées des méthodes

Exemple : liste

```
>>> liste = [8, 12, 26]
>>> type(liste)
<class 'list'>
>>> liste.append(53)
>>> print(liste)
[8, 12, 26, 53]
>>> print(liste.index(26))
2
```

Nous allons voir comment définir de nouvelles classes et de nouveaux objets

utc
SORBONNE UNIVERSITES

Notion de classe et d'objet

Une classe regroupe :

- les caractéristiques de l'entité qu'elle représente (attributs)
- les méthodes effectuant des traitements sur cette entité

Exemple : la classe Cercle

- Attributs : centre, rayon
- Méthodes : perimetre, surface, …

Un objet est une instance d'une classe

Exemple :

l'objet cercle0 de centre O et de rayon 5 est une instance de la classe Cercle

utc
SORBONNE UNIVERSITES

Définition d'une classe

Une classe "encapsule" en général

- des variables d'instance
- des méthodes
- un constructeur, qui est une méthode particulière, appelée à chaque fois qu'une instance est créée

En python

- Une classe est définie à l'aide du mot-clé **class**
- Le nom du constructeur est obligatoirement **__init__**
- L'instance courante est désignée par **self**
- self doit être le premier paramètre des méthodes et donc aussi du constructeur

utc
SORBONNE UNIVERSITES

Un exemple en python

La classe Point

Un point peut être représenté par ses coordonnées :

```
class Point:
    "Définition d'un point à partir de ses coordonnées"
    def __init__(self, abscisse, ordonnee):
        self.x = abscisse
        self.y = ordonnee
    def affiche(self):
        print(f'abscisse : {self.x}, ordonnée : {self.y}')

p = Point(3, 4)
p.affiche()
```

Constructeur

Création de p, une instance de la classe Point

Test

```
abscisse : 3, ordonnée : 4
```

Vérifications

```
>>> Point
<class '__main__.Point'>
>>> p
<__main__.Point object at 0x7fbf4306cc18>
>>> help(Point)
Help on class Point in module __main__:

class Point(builtins.object)
 |  Point(abscisse, ordonnee)
 |
 |  Définition d'un point à partir de ses coordonnées
 |
 |  Methods defined here:
 |
 |  __init__(self, abscisse, ordonnee)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  affiche(self)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

>>> p.__dict__
{'x': 3, 'y': 4}
```

La méthode constructeur

Méthode exécutée automatiquement lors de la création d'une instance

Des valeurs "par défaut" des paramètres peuvent être précisées. Exemple :

```
class Point:
    "Définition d'un point à partir de ses coordonnées"

    def __init__(self, abscisse = 0, ordonnee = 0):
        self.x = abscisse
        self.y = ordonnee

    def affiche(self):
        print(f'abscisse : {self.x}, ordonnée : {self.y}')

p = Point()
p.affiche()
```

Test

```
abscisse : 0, ordonnée : 0
```

Définition d'une méthode

Définition analogue à celle d'une fonction mais :

- Doit toujours être placée à l'intérieur de la définition d'une classe
- Contient au moins un paramètre placé en premier : self
- Le paramètre self est une référence à l'instance courante

Exemple

```
class Point:
    "Définition d'un point à partir de ses coordonnées"

    def __init__(self, abscisse, ordonnee):
        self.x = abscisse
        self.y = ordonnee

    def affiche(self):
        print(f'abscisse : {self.x}, ordonnée : {self.y}')

    def translate(self, dx, dy):
        self.x += dx
        self.y += dy
```

Test

```
>>> p = Point(3, 4)
>>> p.translate(2, 0)
>>> p.affiche()
abscisse : 5, ordonnée : 4
```

Accès aux attributs depuis l'extérieur d'une classe

Accès direct possible mais déconseillé. Exemples :

```
print(p.x)
…
p.y += p.y + 1
```

Il est fortement conseillé de définir si nécessaire des méthodes :

- d'accès (getter)
- de modification (setter)

Similitude et unicité

Similitude

- Si deux objets sont créés à partir de la même classe avec les mêmes valeurs d'attributs, ils sont similaires, mais pas identiques
- Exemple :

```
p1 = Point(3, 4)
p2 = Point(3, 4)
```

- p1 et p2 référencent deux objets distincts

Unicité

- Mais si p2 est définie à partir de p1 :
  - p1 = Point(3, 4)
  - p2 = p1
- Alors p1 et p2 référencent le même objet

Objets composés d'objets

Un objet peut être composé d'autres objets

- Exemple : la classe Cercle contient un objet Point (son centre) créé indépendamment

```
from Point3 import Point
from math import pi

class Cercle:
    "Définition d'un cercle et de son centre"

    def __init__(self, x_centre, y_centre, rayon):
        self.centre = Point(x_centre, y_centre)
        self.rayon = rayon

    def perimetre(self):
        return pi * self.rayon ** 2
```

- Test

```
>>> p = Point(0, 0)
>>> c = Cercle(p, 5)
>>> print(round(c.perimetre(), 2))
78.54
```

Objets composés d'objets

Création d'un objet lors de l'initialisation d'un autre objet

```
from Point3 import Point
from math import pi

class Cercle:
    "Définition d'un cercle et de son centre"

    def __init__(self, x_centre, y_centre, rayon):
        self.centre = Point(x_centre, y_centre)
        self.rayon = rayon

    def perimetre(self):
        return pi * self.rayon ** 2
```

Tests

```
>>> c = Cercle(0, 0, 5)
>>> print(c.centre.x)
0
>>> c.centre.affiche()
abscisse : 0, ordonnée : 0
>>> print(round(c.perimetre(), 2))
78.54
```

Notion d'héritage

Hors programme

Héritage

Une classe C2 peut "hériter" d'une autre classe C1

- On dit que C2 est une classe dérivée de C1
- La classe C2 hérite alors de toutes les propriétés (attributs et méthodes) de C1
- Elle peut en modifier certaines et en ajouter d'autres
- La classe C2 peut elle-même être dérivée bien sûr en une autre classe

Exemple

- Un compte bancaire peut être dérivé en un compte épargne
- Test

```
>>> compte1 = CompteBancaire('Monsieur X', 800)
>>> compte2 = CompteEpargne('Madame Y', 1000, 0.5)
>>> compte1.affiche()
Le solde du compte de Monsieur X est : 800 euros
>>> compte2.affiche()
Le solde du compte de Madame Y est : 1000 euros
Le taux d'intérêt est de 0.5%
```

Héritage

```
class CompteBancaire(object):
    def __init__(self, nom, solde):
        self.nom = nom
        self.solde = solde

    def affiche(self):
        print(f'Le solde du compte de {self.nom} est : {self.solde} euros')

class CompteEpargne(CompteBancaire):
    def __init__(self, nom, solde, taux):
        CompteBancaire.__init__(self, nom, solde)
        self.taux = taux

    def affiche(self):
        CompteBancaire.affiche(self)
        print(f"Le taux d'intérêt est de {self.taux}%")

    def interets_annuels(self):
        return self.solde * self.taux / 100
```

Remarques

- La classe CompteEpargne est dérivée de CompteBancaire
- La méthode affiche de CompteEpargne surcharge celle de CompteBancaire
- Elle appelle la méthode parente (de même que le constructeur)

Compléments

INF1

Dominique Lenne

Sommaire

Retour sur les fonctions et méthodes

Fonctions retournant un tuple

Dictionnaires et objets

Algorithmes gloutons

- Rendu de monnaie
- Parcours de villes

Retour sur les fonctions et méthodes

Une fonction retourne une valeur, une procédure n'en retourne pas

- Une fonction s'utilise au niveau d'une expression, une procédure au niveau d'une instruction
- En Python, une procédure est en fait une fonction qui retourne None

Une méthode s'applique à un objet. Elle peut le modifier.

objet.methode(paramètres)

En python, une méthode retourne une valeur ou la valeur None

- Si elle retourne une valeur elle s'utilise au niveau d'une **expression**, sinon au niveau d'une **instruction**
- Par exemple, pour un cercle c (voir cours précédent),

```
p = c.perimetre()   # la méthode perimetre retourne un réel
c.translate(3, 5)  # la méthode translate s'utilise au niveau
                   d'une instruction. Elle modifie l'objet p
```


Fonctions retournant un tuple

Il est parfois nécessaire qu'une fonction retourne plus d'une valeur.

Une solution est alors de retourner un tuple

Exemple :

Ecrire une fonction permettant de convertir en heure, minutes, secondes une durée exprimée en secondes

```
h, m, s = conversion(duree)
print(f'{h} heures, {m} minutes, {s} secondes')
```

Principe

Calculer les 3 valeurs

- heures = duree // 3600
- minutes = (duree - nb_heures * 3600) // 60
- secondes = (duree – nb_heures * 3600 - nb_minutes * 60) // 60

Puis retourner le tuple des 3 valeurs

retourner (heures, minutes, secondes)

Code Python

```
def conversion(duree):
    "conertit la durée exprimée en secondes en h, m, s"
    heures = duree // 3600
    minutes = (duree % 3600) // 60
    secondes = duree - heures * 3600 - minutes * 60

    return (heures, minutes, secondes)

temps = int(input("durée en secondes ? "))
h, m, s = conversion(temps)
print(f'{h} heures, {m} minutes, {s} secondes')
```

Les parenthèses peuvent être omises

Test

```
durée en secondes ? 12543
3 heures, 29 minutes, 3 secondes
```

Dictionnaires et objets

Dictionnaire ou objet ?

On peut représenter une structure composite par un dictionnaire ou par un objet

Exemple : représentation des notes d'un étudiant

Nom, prénom, note_median, note_final

- dictionnaire

```
etu = {'nom' : 'xxx', 'prenom : 'yyy', 'médian' : 9,
       'final' : 15}
```

- objet

```
class Etudiant:
    def __init__(self, n, p, m, f):
        self.nom = n
        self.prenom = p
        self.median = m
        self.final = f
    def get_final(self):
        return self.final

etu = Etudiant('xxx', 'yyy', 9, 15)
```

Dictionnaire ou objet

Pour représenter les étudiants d'une UV :

- Tableau de dictionnaires

```
UV = [{'nom':'xxx', 'prenom':'yyy', 'médian':9, 'final':15},
      { …                                                  },
      …
      { …                                                  }]
```

- Ou objet contenant un tableau d'étudiants

```
class UV:
    def __init__ (self):
        self.tab_etu = []
    def ajoute(self, etudiant):
        self.tab_etu.append(etudiant)
```

Ajout d'un étudiant :

```
uv = UV()
etu = Etudiant('xxx', 'yyy', 9, 15)
uv.ajoute(etu)
```


Fonction et méthode

Calcul de la moyenne du final

- Fonction avec un tableau de dictionnaires

```
def moyenne_final(uv):
    somme = 0
    n = len(uv)
    for i in range(n):
        somme += uv[i]['final']
    return somme / n
```

- Méthode avec l'objet UV

```
def moyenne_final(self):          # à insérer dans la classe UV
    somme = 0
    n = len(self.tab_etu)
    for i in range(n):
        somme += self.tab_etu[i].get_final()
    return somme / n
```

Algorithmes gloutons

(hors programme)

Problème : rendu de monnaie

Un commerçant souhaite rendre la monnaie à l'un de ses clients en utilisant le moins de pièces et de billets possibles.

Ecrire un programme qui détermine la **combinaison optimale de pièces et de billets** en fonction de la somme à rendre

- On ne considère pas les centimes
- On considère seulement les coupures et pièces en euros :
  1, 2, 5, 10, 20, 50, 100, 200
- On suppose que le commerçant dispose d'une réserve suffisamment pour chaque espèce

Exemple

Somme à rendre : 9€

Combinaison	Pièces
9 x 1€	9
7 x 1€ + 1 x 2€	8
5 x 1€ + 2 x 2€	7
3 x 1€ + 3 x 2€	6
1 x 1€ + 4 x 2€	5
4 x 1€ + 1 x 5€	5
2 x 1€ + 1 x 2€ + 1 x 5€	4
2 x 2€ + 1 x 5€	3

Algorithme "glouton"

- On sélectionne les billets ou pièces à rendre un à un
- On choisit à chaque fois la meilleure solution vis-à-vis de l'objectif, c'est-à-dire la plus grande valeur possible
- Algorithme

```
euros = [ 200, 100, 50, 20, 10, 5, 2, 1]
soit s la somme à rendre
i = 0; nb = 0
tant que s > 0
    si euros[i] ≤ s
        s = s - euros[i]
        nb = nb +1
    sinon
        i = i + 1
```

En python

Avec une amélioration de l'algorithme précédent

```
def monnaie(s):
    euros = [200, 100, 50, 20, 10, 5, 2, 1]
    nb = 0
    i = 0
    while s > 0:
        if euros[i] <= s:
            n = s // euros[i]
            print(n, ' fois ', euros[i],' euros')
            s = s - n * euros[i]
            nb = nb + n
        else:
            i = i + 1
    return nb
```

On peut montrer que cet algorithme est optimal

Parcours de villes*

On considère un ensemble de villes.

Comment **visiter toutes les villes en minimisant la distance totale parcourue**, en partant d'une ville et en revenant à cette même ville ?

Exemple : Nancy, Metz, Paris, Reims, Troyes

S'il faut partir et revenir de Nancy, il y a en tout 4! = 24 circuits différents

Remarque

- Si le nombre de villes devient grand, le problème peut nécessiter un très grand nombre d'opérations
- On peut là-aussi appliquer un algorithme glouton, en choisissant à chaque étape la ville la plus proche

* Extrait de "Numérique et Sciences Informatiques", éditions Ellipses 2019

Circuit	Détail	Total
Metz – Paris – Reims- Troyes	55 + 306 + 142 + 123 + 183	809
Metz – Paris – Troyes - Reims	55 + 306 + 153 + 123 + 188	825
Metz – Troyes – Paris - Reims	55 + 203 + 153 + 142 + 188	741
Troyes – Metz - Paris - Reims	183 + 203 + 306 + 142 + 188	1022
Troyes – Metz – Reims - Paris	183 + 203 + 176 + 142 + 303	1007
Metz –Troyes - Reims - Paris	55 + 203 + 123 + 142 + 303	826
Metz - Reims – Troyes - Paris	…	810
Metz - Reims – Paris - Troyes	…	709
Reims – Metz– Paris - Troyes	…	1006
Reims – Metz – Troyes - Paris	…	1023
Reims – Troyes – Metz - Paris	…	1123
Troyes – Reims – Metz - Paris	…	1091

+ les douze circuits en sens inverse

Algorithme

Algorithme glouton

- on choisit à chaque étape la ville la plus proche
- cela permet de déterminer un circuit parmi les meilleurs, mais pas forcément le meilleur

Données

- Tableau des noms de villes
- Tableau booléen des villes visitées
- Matrice des distances entre villes

```
villes = ['Nancy','Metz', 'Paris', 'Reims', 'Troyes']
dist = [[0, 55, 303, 188, 183],
        [55, 0, 306, 176, 203],
        [303, 306, 0, 142, 153],
        [188, 176, 142, 0, 123],
        [183, 203, 153, 123, 0]]
```

Détermination d'un circuit acceptable

```
def circuit_glouton(villes, dist, depart):
    n = len(villes)
    visitees = [False] * n
    distance_totale = 0
    courante = depart
    for i in range(n - 1):
        visitees[courante] = True
        suivante = plus_proche(courante, dist, visitees)
        distance_totale += cumul_etape(courante, suivante, dist)
        courante = suivante
    distance_totale += cumul_etape(courante, depart, dist)
    print('distance totale :', distance_totale)
```

Test

```
on va de Nancy à Metz en 55 km
on va de Metz à Reims en 176 km
on va de Reims à Troyes en 123 km
on va de Troyes à Paris en 153 km
on va de Paris à Nancy en 303 km
distance totale : 810
```

Pour rappel, le circuit le plus court était de 709 km

Fonctions annexes

Indice de la ville non encore visitée la plus proche

```
def plus_proche(ville, dist, visitees):
    pp = None
    for i in range(len(visitees)):
        if not visitees[i]:
            if pp == None or dist[ville][i] < dist[ville][pp]:
                pp = i
    return pp
```

Affichage d'une étape et retour de la distance

```
def cumul_etape(i, j, dist):
    distance = dist[i][j]
    print("on va de", villes[i], "à", villes[j], "en", distance, "km")
    return distance
```

Annales de médians

(corrections sur Moodle)

NF01 – P18

Examen Médian

Durée : 1 heure 30

Feuille A4 recto-verso autorisée, autres documents interdits.
Calculatrices, téléphones, traducteurs et ordinateurs interdits.

Attention : chaque partie doit être rédigée sur une copie séparée

1ère partie : choix

1.1. Quelle salle pour le médian ?

Ecrire un programme en Python qui demande à un étudiant de INF1 la première lettre de son nom de famille et qui lui indique la salle dans laquelle a lieu le médian. On supposera que les étudiants dont la première lettre du nom est comprise entre 'A' et 'H' doivent aller dans la salle FA501, ceux dont la première lettre est comprise entre 'I' et 'P' dans la salle FA502 et les autres dans la salle FA503.

1.2. Un programme mystérieux

On considère le programme Python suivant :

```
print('Saisir deux entier compris entre 0 et 20')
boule = int(input())
de = int(input())
gomme = boule
if boule < 10 :
    gomme = -boule
print(gomme)
if de < 5 :
    gomme = gomme - de
print('Résultat', gomme)
```

a) Quel est le résultat de ce programme si l'utilisateur saisit 2 et 3 ?
b) Quel est le résultat de ce programme si l'utilisateur saisit 15 et 2 ?

1.3. Le jeu de Chifoumi

Le jeu de Chifoumi, aussi appelé caillou-ciseaux-papier, se joue à deux joueurs avec les mains. Simultanément, les deux joueurs font un signe avec leur main qui représente soit un caillou, soit des ciseaux, soit un papier. Si on nomme les joueurs A et B, les règles sont les suivantes :

- Si A et B font le même signe, il y a égalité, aucun des deux joueurs ne marque de point.
- Si le joueur A joue Caillou et le joueur B Ciseaux, A marque un point, car « le caillou émousse les ciseaux », et réciproquement.
- Si A joue Papier et B joue Caillou, A marque un point, car « le papier enveloppe le caillou », et réciproquement.
- Si A joue Ciseaux et B joue Papier, A marque un point car « les ciseaux coupent le papier », et réciproquement.

1) Ecrire un programme Python qui permet à l'utilisateur de saisir les deux coups joués par A et B et qui affiche le nom du joueur qui marque un point.*)*

2) Modifier le programme précédent pour ajouter un score à A et B, et arrêter la partie quand l'un des joueurs atteint 5 en indiquant qui est le gagnant.

---------- ***{ prendre une nouvelle copie }*** ----------

2ème partie : sapin numérique

On souhaite réaliser un programme en Python qui donne la possibilité à l'utilisateur d'afficher à l'écran une figure 'sapin numérique' (voir Figure 1 ci-dessous).

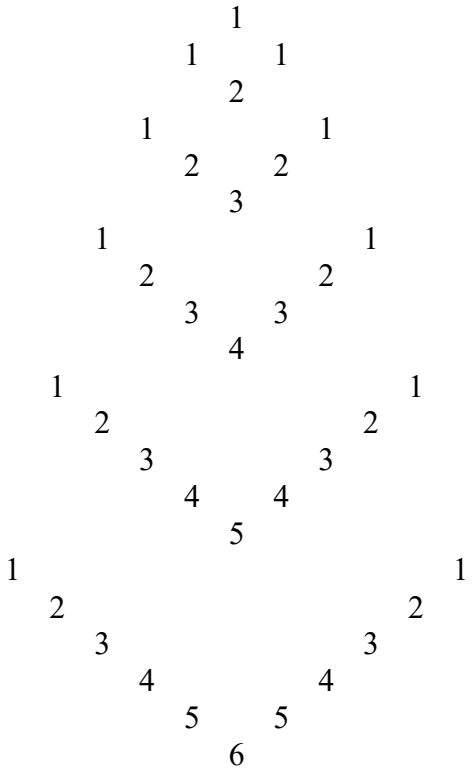
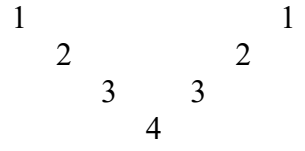


Figure 1. Sapin numérique

L'utilisateur doit pouvoir définir la taille du sapin (le nombre d'étages est 6 pour la Figure 1).

Pour écrire le programme, on procédera en 3 étapes :

1. Définir le nombre *n* (1 < *n* < 10) d'étages du sapin. L'utilisateur peut se tromper dans sa demande (*n* en dehors du domaine de définition), mais pas plus de 5 fois.
2. Ecrire un programme Python permettant d'afficher à l'écran un étage de branche. Par exemple pour l'étage 4, le programme affichera :



3. Afficher le sapin avec le nombre d'étages demandé par l'utilisateur en utilisant l'étape 2.

---------- ***{ prendre une nouvelle copie }*** ----------

3ème partie : Tableaux

a) Ecrire un algorithme permettant à l'utilisateur de saisir un tableau de 8 caractères, puis réalisant un décalage circulaire vers la gauche des éléments de ce tableau.

Exemple :
Tableau initial : D E C A L A G E
Tableau modifié (décalage à gauche) : E C A L A G E D

b) Ecrire un algorithme permettant à l'utilisateur de tester si un tableau est trié en ordre croissant.

INF1 – P21

Examen Médian

Durée : 1 heure 30

.

Document autorisé : une feuille de notes A4 recto-verso
Calculatrices, téléphones, traducteurs et ordinateurs interdits.

Attention : chaque partie doit être rédigée sur une copie séparée

N.B. : on s'attachera à fournir tout commentaire utile et à écrire de façon claire et lisible.

1ère partie : scrabble *(6 points)*

Au jeu de Scrabble, les joueurs doivent former des mots à partir des 7 lettres dont ils disposent. On appellera `jeu` cet ensemble de 7 lettres dans la suite.

1. Écrire en Python une fonction booléenne `acceptable(mot, jeu)` qui retourne `True` si un mot est acceptable pour un jeu donné et `False` sinon.

   Indications :

   - Un mot est dit acceptable si on peut le former à partir des lettres du jeu, même si ce mot n'existe pas en français.
   - On considérera pour cela que `jeu` et `mot` sont constitués uniquement de lettres majuscules non accentuées.

   Exemples : ELEVE, WVB ou BLE sont des mots acceptables avec le jeu BEWEELV

   Conseils :

   - Pour écrire cette fonction, on pourra remarquer qu'un mot est acceptable si le nombre d'occurrences de chacune des lettres du mot est inférieur ou égal au nombre d'occurrences de chacune des lettres du jeu.
   - On pourra utiliser la méthode `count(car)` d'une chaine de caractères, qui retourne le nombre d'occurrences du caractère `car` dans cette chaine. Par exemple, si le mot est ELEVE, `mot.count('E')` retourne 3.

2. Au scrabble, un nombre de points est associé à chaque lettre de la façon suivante :

1 point	EAINORSTUL
2 points	DMG
3 points	BCP
4 points	FHV
8 points	JQ
10 points	KWXYZ

Écrire une fonction `nbPoints(mot)` qui retourne le cumul du nombre de points des lettres du mot `mot`.

3. En utilisant les deux fonctions précédentes, écrire un programme qui :
   a. demande à l'utilisateur le jeu dont il dispose et le mot qu'il souhaite former à l'aide ce jeu,
   b. si le mot est acceptable, affiche le nombre de points qu'il rapporterait et sinon affiche qu'il n'est pas acceptable.

   On supposera que l'utilisateur saisit le mot et le jeu en majuscules.

---------- ***{ prendre une nouvelle copie }*** ----------

2ème partie : alternatives *(7 points)*

1. Un magasin de reprographie facture 20 centimes pièce les dix premières photocopies, 15 centimes les vingt suivantes et 5 centimes au-delà.

   Écrire un algorithme qui demande à l'utilisateur le nombre de photocopies à effectuer puis affiche le montant correspondant en centimes.

2. Une fréquence cardiaque normale au repos se situe entre 50 et 90 bpm (battements par minutes). Durant l'activité physique, la fréquence cardiaque normale peut augmenter jusqu'à 120 bpm.

   Écrire un algorithme permettant de déterminer si une fréquence cardiaque donnée est normale.
   L'utilisateur saisira au clavier la fréquence cardiaque (HR) et indiquera si la mesure a été faite dans le cadre d'une activité physique ou non (variable booléenne activite)

   Exemples :

   Si l'utilisateur saisit HR = 110 et activite = True
   => L'algorithme affiche : Fréquence cardiaque normale
   Si l'utilisateur saisit HR = 95 et activite = False
   => L'algorithme affiche : Fréquence cardiaque anormale

3. On considère le programme Python suivant, qui décrit une procédure de filtrage de valeurs RR, RR correspondant à la durée (en millisecondes) entre deux pics R d'un signal électrique du cœur. RR0, RR1, RR2 et RR3 sont 4 intervalles successifs d'un signal cardiaque.

```
RR0 = float(input('RR0'))
RR1 = float(input('RR1'))
RR2 = float(input('RR2'))
RR3 = float(input('RR3'))

RR_f = RR2 + RR3
e1 = abs(RR_f-RR1)/RR1

print(f' e1 = {e1}')

if e1 < 0.40:
    print(f'RR_f = RR2 + RR3 = {RR_f}')
    print(f'RR2 est fusionnée avec la valeur suivante, RR3')

else:
    RR_f= RR1 + RR2
    e2 = abs(RR_f-RR0)/RR0
    print(f' e2 = {e2}')

    if e2 < 0.40:
        print(f'RR_f = RR2 + RR1 = {RR_f}')
        print(f'RR2 est fusionnée avec la valeur précédente, RR1')
    else:
        print('Les deux erreurs sont supérieures à 40%.')
        print('Il est donc impossible de fusionner les valeurs.')
```

Quel est l'affichage obtenu si l'utilisateur saisit les valeurs suivantes ?

a) RR0 = 0.4, RR1 = 0.4, RR2 = 0.2, RR3 = 0.7

b) RR0 = 0.4, RR1 = 0.6, RR2 = 0.22, RR3 = 0.35

---------- ***{ prendre une nouvelle copie }*** ----------

3ème partie : lancés de dés *(7 points)*

Dans les jeux de plateau, les joueurs doivent souvent lancer un ou deux dés pour déterminer le joueur qui commencera à jouer. Par exemple, le premier joueur à faire un 6 ou un double 6 commence à jouer. Nous allons simuler ce(s) lancé(s) de dé.

1. Dans un premier temps, on souhaite simplement simuler les lancés d'un seul dé sans déterminer le joueur qui fait un 6 en premier.

   Proposer un programme Python qui remplit un tableau avec les valeurs des lancés. Par exemple, s'il y a 5 joueurs, le tableau pourrait contenir les valeurs suivantes : `[4, 3, 2, 6, 1]`

   On utilisera pour cela la fonction `random.randint(1, 6)`. Cette fonction permet de retourner un entier entre 1 et 6.

2. On souhaite maintenant que les joueurs lancent successivement un dé jusqu'à ce que l'un d'eux fasse un 6.

Proposer un programme Python qui affichera un tableau contenant le lancé de chaque joueur, puis le numéro du premier joueur à faire 6. Pour reprendre l'exemple 1, on s'arrêtera au joueur 4 et on affichera *:*

```
[4, 3, 2, 6]
Le premier joueur à faire 6 est le joueur 4
```

Si aucun joueur n'obtient 6, le programme affichera le tableau contenant les valeurs obtenues et signalera qu'aucun joueur n'a obtenu 6.

3. Modifier ce programme (en Python ou sous forme d'algorithme), pour que :
   a. Si personne ne fait 6, on relance des tours de lancés de dé jusqu'au premier 6.
   b. Le programme affiche le numéro du premier joueur à obtenir 6 et le tour pendant lequel il l'a obtenu.

   Par exemple, l'affichage pourrait être :

```
==== Tour 1 ====
[3, 1, 5, 5, 1]
==== Tour 2 ====
[2, 6]
Le joueur qui a fait un 6 en premier est le joueur 2 au tour 2
```

4. Les joueurs lancent maintenant 2 dés jusqu'à ce que l'un d'eux obtienne un double 6. Par exemple, au 1er lancé, si le joueur 1 fait 4 et 1 et le joueur 2 fait 2 et 4, le tableau se remplit ainsi : [[4, 1], [2, 4]…].
   Proposer un algorithme, sur le principe de la question 3, qui déterminera le 1er joueur à faire un double 6.
   Par exemple, l'affichage pourra être :

```
==== Tour 1 ====
[[4, 1], [2, 4], [3, 2], [2, 1], [4, 5]]
==== Tour 2 ====
[[4, 3], [2, 6], [2, 5], [6, 6]]
Le joueur qui a fait un double 6 en premier est le joueur 4 au
tour 2
```

Annales de finaux

(corrections sur Moodle)

Examen Final P17

Sujet adapté pour Python

1ère Partie : Gestion d'un annuaire *(5 points)*

Vous devez proposer un programme Python de gestion d'un annuaire. Pour chaque personne stockée dans l'annuaire, les informations suivantes sont renseignées : nom, prénom, numéro (dans la rue), nom de la rue, numéro de téléphone, code postal, ville. Votre programme doit permettre la saisie des informations, des recherches et des extractions suivant des critères choisis par l'utilisateur.

1. Définissez la structure correspondant à la personne à l'aide des informations citées précédemment. Définissez l'annuaire en tant que tableau contenant ce type de données.
2. Votre programme de gestion de l'annuaire doit contenir certaines fonctionnalités structurées sous forme de fonctions et procédures appelées dans le programme principal. Vous devez définir :
    a. Une fonction *saisie_tab* permettant la saisie de toutes les données pour toutes les entrées de l'annuaire et retournant le tableau correspondant à cet annuaire.
    b. Une fonction *critere_recherche*, sans arguments, qui permet à l'utilisateur de choisir le critère de recherche (nom, prénom, nom de la rue, numéro de téléphone, code postal, ville). Cette fonction doit retourner le choix de l'utilisateur.
    c. Une fonction *recherche* à deux arguments : le tableau annuaire et le critère de recherche (Cf. 2.b). L'utilisateur doit pouvoir donner la valeur de recherche (par exemple Compiègne si le critère de recherche est 'ville'). Cette fonction doit retourner un tableau booléen de la taille du tableau annuaire contenant la valeur `True` pour les entrées de l'annuaire qui correspondent à la recherche demandée.
    d. Une procédure *affiche_tab* à deux arguments : le tableau annuaire et un tableau du type de celui retourné par la fonction *recherche* précédente. Toutes les informations relatives aux personnes correspondant au critère de recherche doivent être affichées à l'écran.

2ème Partie : Récursivité *(5 points)*

2. 1 : La fonction itérative suivante, `divin(a,b)`, retourne le résultat de la division entière de *a* par *b* :

```
def divin(a,b):
    d = 0
    while a >= b :
        d = d + 1
        a = a - b
     return d
```

Il s'agit en fait du calcul `a // b` en Python.

Vous devez écrire la version récursive de cette fonction. Pour cela :

a) Déterminez l'expression récursive et le critère d'arrêt de la séquence des appels récursifs.

b) Ecrivez la fonction récursive `divinRec(a,b)` qui fournit le résultat de `a div b,` en utilisant uniquement les opérateurs "+" et/ou "-".

2.2 : On souhaite écrire une fonction récursive prenant en paramètres deux valeurs, *a* un réel et *b* un entier, et retournant la valeur réelle a^b

a) Quel(s) critère(s) d'arrêt proposez-vous ?
b) Ecrivez la fonction
c) Combien y a-t-il d'appels récursifs pour *a=2* et *b=3* ? Décrivez graphiquement les appels récursifs dans ce cas.

{ ---------- pensez à changer de copie ---------- }

3ème Partie : Fichiers *(5 points)*

Dans cette partie, on suppose que l'on dispose d'un fichier texte composé exactement d'un mot par ligne, en minuscules et sans espacement. Chaque procédure ou fonction fait les opérations d'ouverture et de fermeture des fichiers dont le nom est passé en paramètre sous forme de chaine de caractères.

3.1 – Ecrire la fonction `nbMotsAvecVoyelle(nomf)` :

qui renvoie le nombre de mots commençant par une voyelle et présents dans le fichier de nom `nomf`.

3.2 – On considère maintenant un fichier de nom `nomf1` trié par ordre lexicographique (*i.e.* alphabétique) croissant.

Ecrire la procédure `compterChaqueMot(nomf1,nomf2)` qui écrit chaque mot du fichier `nomf1` de façon unique dans le fichier `nomf2`, suivi sur la même ligne d'un espace et du nombre d'occurrences de ce mot.

N.B. : La procédure ne fera qu'un seul passage sur le fichier `nomf1`. Celui-ci étant trié, on utilisera le fait que les occurrences d'un même mot sont forcément consécutives.

{ ---------- pensez à changer de copie ---------- }

4ème Partie : Retour vers le futur *(5 points)*

1) On considère le programme Python suivant :

```
On souhaite représenter une personne à l'aide des champs
suivants :
   • nom  # nom de la personne (chaine de caractères)
   • annee # année actuelle de la personne (entier)
   • temps (entier) # temps (en secondes) nécessaire pour
                    # revenir en 2017
```

Ecrire en Python une fonction `Saisie()` permettant de saisir le nom des personnes et retournant ces noms dans un tableau `t`
Le nombre de personnes n'est pas connu à l'avance. Ce tableau sera par la suite utilisé pour calculer et ajouter les champs `annee` et `temps` de chaque personne.

2) Chaque personne doit choisir une période d'au moins 10 ans au cours de laquelle elle souhaite faire un voyage dans le temps. Le programme choisit aléatoirement une année de départ au cours de cette période.

   Ecrire une procédure `calculAnnee` en Python qui pour chaque personne du tableau `t` :

   - demande dans quelle période, donnée par `annee_min` et `annee_max`, elle souhaite faire un voyage dans le temps. Cette période sera comprise entre -10 000 ans et 10 000 ans.
   - appelle la fonction `random.randint(annee_min, annee_max)` et stocke l'année renvoyée par cette fonction dans le champ `annee` de la personne.

3) Le retour en l'an 2017 se fait également de manière aléatoire par essais successifs. Ecrire une procédure `calculTemps` en Python qui, pour chaque personne du tableau `t`, calcule le temps nécessaire pour revenir en l'an 2017 et le stocke dans le champ `temps` de cette personne, sachant que :

   - la période de saut est forcément de 10 ans autour de 2017, i.e. entre 2012 et 2022,
   - le temps pour chaque saut nécessite 10 secondes.

4) Prendriez-vous ce risque ?

INF1 – P21

Examen Final

Durée : 1 heure 30

.

Document autorisé : une feuille de notes A4 recto-verso
Calculatrices, téléphones, traducteurs et ordinateurs interdits.

Attention : chaque partie doit être rédigée sur une copie séparée

N.B. : on s'attachera à fournir tout commentaire utile et à écrire de façon claire et lisible.

1ère partie : vente à distance *(7 points)*

Un site internet de vente à distance gère les commandes et le stock à l'aide des classes suivantes :

- La classe `Produit` permet de représenter un produit par sa référence (`ref`), son prix unitaire (`prix`), et la quantité de ce produit en stock (`stock`) :

```
class Produit :
    def __init__(self, ref, stock, prix) :
        self.ref = ref
        self.stock = stock
        self.prix = prix

    def toString(self) :
        return f'{self.ref}({self.stock}) - prix : {self.prix}'
```

  N.B. : on considérera que la référence d'un produit est son nom tout simplement.

- La classe `Stock` permet de gérer la liste des produits :

```
class Stock :
    def __init__(self) :
        self.produits = []
    def afficheStock(self) :
        for produit in self.produits :
            print(produit.toString())
```

- La classe `ItemCommande` représente un item de commande, c'est-à-dire le nom d'un produit et la quantité désirée :

```
class ItemCommande :
    def __init__(self, ref, nb) :
        self.ref = ref
        self.nb = nb
```

Une commande est simplement une liste constituée d'un ou plusieurs items de commande.

Voici un exemple d'utilisation :

```
stock = Stock()
stock.ajouteListeProduits([Produit('stylo bleu', 1500, 1.1),
                           Produit('gomme', 200, 0.55),
                           Produit('cahier A4', 350, 2.3)
                           Produit('petit agenda', 589, 13)])
commande = [ItemCommande('gomme', 100),
            ItemCommande('cahier A4', 300),
            ItemCommande('petit agenda', 200)]
if stock.commandePossible(commande) :
    print('Montant :', stock.montantCommande(commande))
    stock.fournitCommande(commande)
    stock.afficheStock()
```

On vous demande d'écrire les méthodes de la classe Stock suivantes :

1. Écrire la méthode `ajouteListeProduits(self, listeProduits)` qui permet d'ajouter une liste de produits à la liste `self.produits`.
2. Écrire la méthode `trouveProduit(self, ref)` qui retourne un produit à partir de sa référence. On supposera que la référence `ref` est bien celle d'un des produits du stock.
3. Écrire la méthode booléenne `commandePossible(self, commande)` qui détermine si une commande est possible ou non en fonction des stocks disponibles. On supposera que les références demandées sont correctes, que les quantités sont positives et que deux items ne correspondent pas à la même référence.
4. Écrire la méthode `montantCommande(self, commande)` qui retourne le montant d'une commande.
5. Écrire la méthode `fournitCommande(self, commande)` qui modifie le stock après la commande.

N.B. les questions sont indépendantes entre elles. La méthode `trouveProduit` de la question 2 pourra éventuellement être utilisée dans les autres questions.

---------- ***{ prendre une nouvelle copie }*** ----------

2ème partie : récursivité *(6 points)*

1. On donne la définition récursive suivante du PGCD de deux nombres :
   « Le PGCD de deux nombres entiers est égal au PGCD du plus petit et du reste de la division du plus grand par le plus petit. Le PGCD est alors le dernier reste non nul. »

   Ainsi, partant des nombres 50 et 46, on obtient successivement 46 et 4, puis 4 et 2, puis 2 et 0. Le pgcd est donc 2.

   Écrire une fonction récursive retournant le pgcd de deux nombres suivant cette règle.

2. Une liste l_1 est une sous-liste d'une liste l si tous les éléments de l_1 sont des éléments de l et si l'ordre des éléments de l_1 est respecté dans l. Ainsi :

[2, 10, 4, 8] est une sous-liste de [9, 2, 11, 10, 7, 3, 4, 8, 25]
[2, 10, 4, 8] n'est pas une sous-liste de [9, 10, 7, 3, 4, 8, 25]

Écrire une fonction récursive indiquant si une liste l_1 est une sous-liste d'une liste l.

Indication : pour écrire la fonction on pourra considérer le premier élément des deux listes et appeler récursivement sur la fin d'au moins une des deux listes. On obtient un succès si l'on a réussi à vider l1 avant l.

---------- ***{ prendre une nouvelle copie }*** ----------

3ème partie : fichiers de joueurs *(7 points)*

Le sélectionneur Didier Deschamps a fait appel à 26 joueurs pour composer l'équipe de France qui participera à l'Euro2021 de football.
Il dispose d'un fichier texte contenant le nom et le prénom de ses joueurs. Chaque ligne de ce fichier contient le nom et le prénom séparés par une virgule (pas d'espace avant ou après la virgule), de la manière suivante :

…
Benzema,Karim
...
Mbappe,Kylian
…

Toutefois, le sélectionneur souhaite avoir plus d'informations sur chaque joueur. Pour cela, il envoie son fichier à la fédération française de football, et demande qu'on y ajoute, pour chaque joueur : la date de naissance et la liste des clubs où le joueur a évolué en parcours professionnel, chaque club étant suivi du nombre de buts marqués par le joueur dans ce club.

Chaque ligne du fichier final aura la structure suivante :

…
Benzema,Karim,19/12/1987,Real Madrid,279,Olympique Lyonnais,81,…
...

1) Donnez une structure de données de type dictionnaire qui peut contenir toutes les données relatives à un joueur, souhaitées par le sélectionneur. Utilisez les données relatives au joueur cité comme exemple.

2) Écrire une fonction `creer_joueur(nom, prenom)`, qui prend en entrée le nom et prénom d'un joueur, demande à l'utilisateur les autres informations, et retourne le dictionnaire correspondant au joueur.

3) Écrire une fonction `creer_liste_joueurs(nom_fichier)`, qui prend en entrée le nom du fichier d'origine du sélectionneur et retourne la liste des joueurs. Cette fonction fera appel à `creer_joueur(nom, prenom)` pour créer les joueurs.

4) Écrire une fonction `fich_select_final(liste_joueurs, nom_fichier)`, qui prend en entrée la liste de joueurs et crée le fichier final de Didier Deschamps.

Mémento Python 3

Dernière version sur :
https://perso.limsi.fr/pointal/python:memento

Types de base

entier, flottant, booléen, chaîne, octets

```
int 783  0 -192  0b010 0o642 0xF3
          zéro   binaire octal hexa
float 9.23 0.0  -1.7e-6
                        ×10^-6
bool True  False
str "Un\nDeux"
     retour à la ligne échappé
    'L\'âme'
      ' échappé
bytes b"toto\xfe\775"
          hexadécimal  octal
```

Chaîne multiligne :
`"""X\tY\tZ`
`1\t2\t3"""`
tabulation échappée

immutables

Types conteneurs

- **séquences ordonnées**, accès par index rapide, valeurs répétables

```
list [1,5,9]   ["x",11,8.9]   ["mot"]   []
tuple (1,5,9)  11,"y",7.4     ("mot",)  ()
```

Valeurs non modifiables (immutables) — *expression juste avec des virgules →* `tuple`

`str` `bytes` *(séquences ordonnées de caractères / d'octets)* `""` `b""`

- **conteneurs clés**, sans ordre *a priori*, accès par clé rapide, chaque clé unique

```
dictionnaire dict {"clé":"valeur"}   dict(a=3,b=4,k="v")   {}
(couples clé/valeur)  {1:"un",3:"trois",2:"deux",3.14:"π"}
ensemble     set  {"clé1","clé2"}    {1,9,3,0}            set()
```

clés=valeurs hachables (types base, immutables…) `frozenset` *ensemble immutable* *vide*

Identificateurs

pour noms de variables, fonctions, modules, classes…

`a…zA…Z_` suivi de `a…zA…Z_0…9`

- accents possibles mais à éviter
- mots clés du langage interdits
- distinction casse min/MAJ

☺ `a toto x7 y_max BigOne`
☹~~`8y and for`~~

Variables & affectation

`=` affectation ⇔ **association** d'un *nom* à une *valeur*

1) évaluation de la valeur de l'expression de droite
2) affectation dans l'ordre avec les noms de gauche

```
x=1.2+8+sin(y)
a=b=c=0      affectation à la même valeur
y,z,r=9.2,-7.6,0   affectations multiples
a,b=b,a      échange de valeurs
a,*b=seq     dépaquetage de séquence en
*a,b=seq     élément et liste
x+=3   incrémentation ⇔ x=x+3      et *= /= %= …
x-=2   décrémentation ⇔ x=x-2
x=None valeur constante « non défini »
del x  suppression du nom x
```

Conversions

`type(`*expression*`)`

```
int("15") → 15
int("3f",16) → 63      spécification de la base du nombre entier en 2nd paramètre
int(15.56) → 15        troncature de la partie décimale
float("-11.24e8") → -1124000000.0
round(15.56,1)→15.6    arrondi à 1 décimale (0 décimale → nb entier)
bool(x)  False pour x zéro, x conteneur vide, x None ou False ; True pour autres x
str(x)→"…"   chaîne de représentation de x pour l'affichage (cf. Formatage au verso)
chr(64)→'@'  ord('@')→64   code ↔ caractère
repr(x)→"…"  chaîne de représentation littérale de x
bytes([72,9,64]) → b'H\t@'
list("abc") → ['a','b','c']
dict([(3,"trois"),(1,"un")]) → {1:'un',3:'trois'}
set(["un","deux"]) → {'un','deux'}
str de jointure et séquence de str → str assemblée
    ':'.join(['toto','12','pswd']) → 'toto:12:pswd'
str découpée sur les blancs → list de str
    "des mots  espacés".split() → ['des','mots','espacés']
str découpée sur str séparateur → list de str
    "1,4,8,2".split(",") → ['1','4','8','2']
séquence d'un type → list d'un autre type (par liste en compréhension)
    [int(x) for x in ('1','29','-3')] → [1,29,-3]
```

Indexation conteneurs séquences

pour les listes, tuples, chaînes de caractères, bytes…

```
index négatif  -5  -4  -3  -2  -1
index positif   0   1   2   3   4
lst=[10, 20, 30, 40, 50]
tranche positive 0   1   2   3   4   5
tranche négative -5  -4  -3  -2  -1
```

Nombre d'éléments `len(lst)→5`

index à partir de 0 (de 0 à 4 ici)

Accès individuel aux **éléments** par `lst[`*index*`]`

`lst[0]→10` ⇒ *le premier* `lst[1]→20`
`lst[-1]→50` ⇒ *le dernier* `lst[-2]→40`

Sur les séquences modifiables (`list`), suppression avec `del lst[3]` *et modification par affectation* `lst[4]=25`

Accès à des **sous-séquences** par `lst[`*tranche début*`:`*tranche fin*`:`*pas*`]`

```
lst[:-1]→[10,20,30,40]   lst[::-1]→[50,40,30,20,10]   lst[1:3]→[20,30]   lst[:3]→[10,20,30]
lst[1:-1]→[20,30,40]     lst[::-2]→[50,30,10]         lst[-3:-1]→[30,40]  lst[3:]→[40,50]
lst[::2]→[10,30,50]      lst[:]→[10,20,30,40,50]  copie superficielle de la séquence
```

Indication de tranche manquante → à partir du début / jusqu'à la fin.
Sur les séquences modifiables (`list`), suppression avec `del lst[3:5]` *et modification par affectation* `lst[1:4]=[15,25]`

Logique booléenne

Comparateurs: `<` `>` `<=` `>=` `==` `!=` (résultats booléens) ≤ ≥ = ≠

`a and b` et logique — *les deux en même temps*

`a or b` ou logique — *l'un ou l'autre ou les deux*

piège : `and` *et* `or` *retournent la valeur de* `a` *ou de* `b` *(selon l'évaluation au plus court).*
⇒ s'assurer que `a` *et* `b` *sont booléens.*

`not a` non logique

`True` `False` constantes Vrai/Faux

Blocs d'instructions

instruction parente :
bloc d'instructions 1…
instruction parente :
bloc d'instructions 2…
instruction suivante après bloc 1

indentation !

régler l'éditeur pour insérer 4 espaces à la place d'une tabulation d'indentation.

Imports modules/noms

module `truc` ⇔ *fichier* `truc.py`

```
from monmod import nom1,nom2 as fct
```
→*accès direct aux noms, renommage avec* `as`

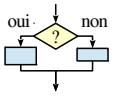
```
import monmod
```
→*accès via* `monmod.nom1` …

modules et packages cherchés dans le *python path* (cf. `sys.path`)

Instruction conditionnelle

un bloc d'instructions exécuté, uniquement si sa condition est vraie

`if` *condition logique* `:`
→ *bloc d'instructions*



Combinable avec des *sinon si, sinon si…* et un seul *sinon* final. Seul le bloc de la première condition trouvée vraie est exécuté.

avec une variable `x`*:*

```
if bool(x)==True:   ⇔ if x:
if bool(x)==False:  ⇔ if not x:
```

```
if age<=18:
    etat="Enfant"
elif age>65:
    etat="Retraité"
else:
    etat="Actif"
```

Maths

nombres flottants… valeurs approchées !

angles en radians

Opérateurs : `+` `-` `*` `/` `//` `%` `**`
Priorités `(…)` × ÷ ↑ ↑ a^b
÷ entière reste ÷

`@` → × matricielle *python3.5+* `numpy`

```
(1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6
pow(4,3)→64.0
```

priorités usuelles

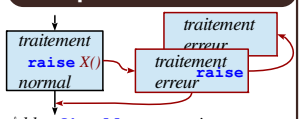
```
from math import sin,pi…
sin(pi/4)→0.707…
cos(2*pi/3)→-0.4999…
sqrt(81)→9.0      √
log(e**2)→2.0
ceil(12.5)→13
floor(12.5)→12
```

modules `math`, `statistics`, `random`, `decimal`, `fractions`, `numpy`, etc.

Exceptions sur erreurs

Signalisation :
`raise` *ExcClass(…)*

Traitement :
`try:`
→ *bloc traitement normal*
`except` *ExcClass* `as e:`
→ *bloc traitement erreur*



bloc `finally` *pour traitements finaux dans tous les cas.*

Instruction boucle conditionnelle

*bloc d'instructions exécuté **tant que** la condition est vraie*

```
while condition logique :
    bloc d'instructions
```


```
s = 0          # initialisations avant la boucle
i = 1
while i <= 100:          # condition avec au moins une valeur variable (ici i)
    s = s + i**2
    i = i + 1            # faire varier la variable de condition !
print("somme:",s)
```

Algo : $s = \sum_{i=1}^{i=100} i^2$

Contrôle de boucle

`break` sortie immédiate
`continue` itération suivante
bloc `else` en sortie *normale* de boucle.

attention aux boucles sans fin !

Affichage

```
print("v=",3,"cm :",x,",",y+4)
```

éléments à afficher : valeurs littérales, variables, expressions
Options de `print` :

- `sep=" "` séparateur d'éléments, défaut espace
- `end="\n"` fin d'affichage, défaut fin de ligne
- `file=sys.stdout` print vers fichier, défaut sortie standard

Saisie

```
s = input("Directives:")
```

`input` retourne toujours une **chaîne**, la convertir vers le type désiré (cf. encadré *Conversions* au recto).

Opérations génériques sur conteneurs

`len(c)` → nb d'éléments
`min(c)` `max(c)` `sum(c)`
`sorted(c)` → `list` *copie* triée
`val in c` → booléen, opérateur `in` de test de présence (`not in` d'absence)
`enumerate(c)` → *itérateur* sur (index, valeur)
`zip(c1,c2…)` → *itérateur* sur tuples contenant les éléments de même index des c_i
`all(c)` → `True` si **tout** élément de `c` évalué vrai, sinon `False`
`any(c)` → `True` si **au moins un** élément de `c` évalué vrai, sinon `False`
`c.clear()` supprime le contenu des dictionnaires, ensembles, listes

*Note: Pour dictionnaires et ensembles, ces opérations travaillent sur les **clés**.*

*Spécifique aux **conteneurs de séquences ordonnées** (listes, tuples, chaînes, bytes…)*
`reversed(c)` → *itérateur inversé* `c*5`→ duplication `c+c2`→ concaténation
`c.index(val)` → *position* `c.count(val)` → *nb d'occurences*

```
import copy
copy.copy(c) → copie superficielle du conteneur
copy.deepcopy(c) → copie en profondeur du conteneur
```

Opérations sur listes

modification de la liste originale

`lst.append(val)`	ajout d'un élément à la fin
`lst.extend(seq)`	ajout d'une séquence d'éléments à la fin
`lst.insert(idx,val)`	insertion d'un élément à une position
`lst.remove(val)`	suppression du premier élément de valeur *val*
`lst.pop([idx])`→*valeur*	supp. & retourne l'item d'index *idx* (défaut le dernier)
`lst.sort()` `lst.reverse()`	tri / inversion de la liste sur place

Opérations sur dictionnaires

`d[clé]=valeur` `del d[clé]`
`d[clé]` → *valeur*
`d.update(d2)` mise à jour/ajout des couples
`d.keys()` `d.values()` `d.items()` →*vues itérables sur les clés / valeurs / couples*
`d.pop(clé[,défaut])` → *valeur*
`d.popitem()` → *(clé,valeur)*
`d.get(clé[,défaut])` → *valeur*
`d.setdefault(clé[,défaut])` →*valeur*

Opérations sur ensembles

Opérateurs :
`|` → union (caractère barre verticale)
`&` → intersection
`-` `^` → différence/diff. symétrique
`<` `<=` `>` `>=` → relations d'inclusion
Les opérateurs existent aussi sous forme de méthodes.

`s.update(s2)` `s.copy()`
`s.add(clé)` `s.remove(clé)`
`s.discard(clé)` `s.pop()`

Fichiers

stockage de données sur disque, et relecture

```
f = open("fic.txt","w",encoding="utf8")
```

- **variable** fichier pour les opérations
- **nom** du fichier sur le disque (+chemin…) — cf modules `os`, `os.path` et `pathlib`
- **mode** d'ouverture
  - `'r'` lecture (read)
  - `'w'` écriture (write)
  - `'a'` ajout (append)
  - …`'+'` `'x'` `'b'` `'t'`
- **encodage** des caractères pour les *fichiers textes*: `utf8` `ascii` `latin1` …

en **écriture**

```
f.write("coucou")
f.writelines(list de lignes)
```

en **lecture**

lit chaîne vide si fin de fichier

```
f.read([n])          → caractères suivants
                       si n non spécifié, lit jusqu'à la fin !
f.readlines([n])     → list lignes suivantes
f.readline()         → ligne suivante
```

*par défaut mode texte `t` (lit/écrit `str`), mode binaire `b` possible (lit/écrit `bytes`). **Convertir de/vers le type désiré** !*

`f.close()` ne pas oublier de **refermer le fichier** après son utilisation !

`f.flush()` écriture du cache `f.truncate([taille])` retaillage
lecture/écriture progressent séquentiellement dans le fichier, modifiable avec :
`f.tell()` →*position* `f.seek(position[,origine])`

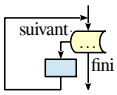
Très courant : ouverture en bloc gardé (fermeture automatique) et boucle de lecture des lignes d'un fichier texte.

```
with open(…) as f:
    for ligne in f :
        # traitement de ligne
```

Instruction boucle itérative

*bloc d'instructions exécuté **pour chaque** élément d'un conteneur ou d'un itérateur*

```
for var in séquence :
    bloc d'instructions
```



Parcours des **valeurs** d'un conteneur

```
s = "Du texte"      # initialisations avant la boucle
cpt = 0
                    # variable de boucle, affectation gérée par l'instruction for
for c in s:
    if c == "e":
        cpt = cpt + 1
print("trouvé",cpt,"'e'")
```

Algo : comptage du nombre de `e` dans la chaîne.

boucle sur dict/set ⇔ boucle sur séquence des clés
utilisation des *tranches* pour parcourir un sous-ensemble d'une séquence

Parcours des **index** d'un conteneur séquence

- changement de l'élément à la position
- accès aux éléments autour de la position (avant/après)

```
lst = [11,18,9,12,23,4,17]
perdu = []
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        perdu.append(val)
        lst[idx] = 15
print("modif:",lst,"-modif:",perdu)
```

Algo: bornage des valeurs supérieures à 15, mémorisation des valeurs perdues.

Parcours simultané **index** et **valeurs** de la séquence :

```
for idx,val in enumerate(lst):
```

bonne habitude : ne pas modifier la variable de boucle

Séquences d'entiers

```
range([début,] fin [,pas])
```

début défaut 0, *fin* non compris dans la séquence, *pas* signé et défaut 1

`range(5)` → 0 1 2 3 4 `range(2,12,3)` → 2 5 8 11
`range(3,8)` → 3 4 5 6 7 `range(20,5,-5)` → 20 15 10
`range(len(séq))` → *séquence des index des valeurs dans séq*

range fournit une séquence immutable d'entiers construits au besoin

Définition de fonction

nom de la fonction (identificateur) — paramètres nommés

```
def fct(x,y,z):
    """documentation"""
    # bloc instructions, calcul de res, etc.
    return res
```

`return res` ← valeur résultat de l'appel, si pas de résultat calculé à retourner : `return None`

les paramètres et toutes les variables de ce bloc n'existent que *dans* le bloc et *pendant* l'appel à la fonction (penser "*boîte noire*")

Avancé : `def fct(x,y,z,*args,a=3,b=5,**kwargs):`
**args nb variables d'arguments positionnels (→`tuple`), valeurs par défaut, **kwargs nb variable d'arguments nommés (→`dict`)*

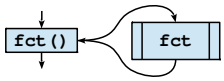
Appel de fonction

```
r = fct(3,i+2,2*i)
```

stockage/utilisation de la valeur de retour — *une valeur d'argument par paramètre*

c'est l'utilisation du nom de la fonction *avec les parenthèses* qui fait l'appel

*Avancé: *séquence **dict*



Opérations sur chaînes

`s.startswith(prefix[,début[,fin]])` `s.endswith(suffix[,début[,fin]])` `s.strip([caractères])`
`s.count(sub[,début[,fin]])` `s.partition(sep)` → *(avant,sep,après)*
`s.index(sub[,début[,fin]])` `s.find(sub[,début[,fin]])`
`s.is…()` *tests sur les catégories de caractères (ex. `s.isalpha()`)*
`s.upper()` `s.lower()` `s.title()` `s.swapcase()`
`s.casefold()` `s.capitalize()` `s.center([larg,rempl])`
`s.ljust([larg,rempl])` `s.rjust([larg,rempl])` `s.zfill([larg])`
`s.encode(codage)` `s.split([sep])` `s.join(séq)`

Formatage

directives de formatage — valeurs à formater

```
"modele{} {} {}".format(x,y,r) → str
"{sélection:formatage!conversion}"
```

- **Sélection** :
  `2`
  `nom`
  `0.nom`
  `4[clé]`
  `0[2]`
- **Formatage** :
  car-rempl. *alignement* *signe* *larg.mini*.*précision~larg.max* *type*
  `< > ^ =` `+ - espace` `0` au début pour remplissage avec des 0
  entiers : `b` binaire, `c` caractère, `d` décimal (défaut), `o` octal, `x` ou `X` hexa…
  flottant : `e` ou `E` exponentielle, `f` ou `F` point fixe, `g` ou `G` approprié (défaut),
  chaîne : `s` … `%` pourcentage
- **Conversion** : `s` (texte lisible) ou `r` (représentation littérale)

Exemples :

```
"{:+2.3f}".format(45.72793)
→'+45.728'
"{1:>10s}".format(8,"toto")
→'      toto'
"{x!r}".format(x="L'ame")
→'"L\'ame"'
```