



UNIVERSITÉ
CÔTE D'AZUR

Programmation impérative en Python

Cours 3. Boucles For et chaînes de caractères

Olivier Baldellon

Courriel : `prenom.nom@univ-cotedazur.fr`

Page professionnelle : `https://upinfo.univ-cotedazur.fr/~obaldellon/`

LICENCE I — FACULTÉ DES SCIENCES ET INGÉNIERIE DE NICE — UNIVERSITÉ CÔTE D'AZUR

- 🍃 Partie I. Compléments
- 🍃 Partie II. Boucles for
- 🍃 Partie III. Chaînes de caractères
- 🍃 Partie IV. Manipuler les chaînes
- 🍃 Partie V. Représentation des caractères
- 🍃 Partie VI. Cryptologie
- 🍃 Partie VII. Table des matières

- ▶ `print` ajoute automatiquement des espaces et un retour à la ligne.

```
def test():  
    print('12', '34', '56')  
    print('bon', 'jour')
```

SCRIPT

```
>>> test()  
12 34 56  
bon jour
```

SHELL

- ▶ On peut modifier ce comportement grâce aux options `sep` et `end`

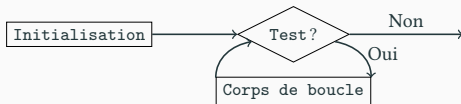
```
def test2():  
    print('12', '34', '56', sep=' - ', end=':')  
    print('bon', 'jour', sep='')
```

SCRIPT

```
>>> test2()  
12 - 34 - 56:bonjour  
>>> # Il y a bien eu nouvelle ligne après bonjour
```

SHELL

- 🍃 Partie I. Compléments
- 🍃 Partie II. Boucles `for`
- 🍃 Partie III. Chaînes de caractères
- 🍃 Partie IV. Manipuler les chaînes
- 🍃 Partie V. Représentation des caractères
- 🍃 Partie VI. Cryptologie
- 🍃 Partie VII. Table des matières



- ▶ De nombreuses boucles `while` sont de la forme :

```
i=0
while i<n:
    ...
    ...
    i=i+1
```

SCRIPT

- ▶ L'initialisation et le test sont toujours les mêmes quelque soit `n`.
- ▶ Dans la boucle, la variable `i` varie de 0 à `n-1`
- ▶ Il est facile de voir que pour `n>0`, un telle boucle termine toujours.

```
i=0
while i<n:
    ...
    ...
    i=i+1
```

SCRIPT

```
for i in range(n):
    ...
    ...
```

SCRIPT

- ▶ Plus besoin d'initialiser `i`
- ▶ Plus besoin d'incrémenter (`i=i+1`) la variable `i`
- ▶ Plus besoin d'écrire le test.
- ▶ Et on a la garantie que la boucle se termine après `n` passages !

```
def f():
    for i in range(10):
        print(i, end='')
    print('')
```

SCRIPT

```
>>> f()
0123456789
```

SHELL

- ▶ **Attention** : `range(10)` correspond à l'ensemble de 10 éléments `0, 1, ..., 9`

- ▶ On peut préciser le début :

```
def f():  
    for i in range(1,10):  
        print(i,end='')
```

SCRIPT

```
>>> f()  
123456789
```

SHELL

- ▶ Ainsi que le début et le pas

```
def f():  
    for i in range(1,10,3):  
        print(i,end='')
```

SCRIPT

```
>>> f()  
147
```

SHELL

- ▶ On part de $i=1$
- ▶ On incrémente avec $i=i+3$
- ▶ $1 \rightarrow 4 \rightarrow 7 \rightarrow$ **STOP** car i doit toujours vérifier $i < 10$

- ▶ Une boucle `for` est équivalent à une boucle `while`
 - ▶ début, fin et pas sont des entiers (`pas≠0`);
 - ▶ dans le cas où pas est négatif la condition est `i>fin`.

```
# début=0 par défaut  
# pas=1 par défaut  
for i in range(début,fin,pas):  
    ...  
    ...
```

SCRIPT

```
i=début  
while i < fin:  
    ...  
    ...  
    i=i+pas
```

SCRIPT

- ▶ Cette équivalence est vérifiée si :
 - ▶ On ne modifie pas `i` dans le corps de la boucle
 - ▶ On n'utilise pas `i` après la boucle
- ▶ De toutes façons, ne pas respecter ces usages est une mauvaise pratique !

▶ Que vaut i à la fin de la boucle ?

```

>>> i=0
>>> while i < 2:
...     print(i)
...     i=i+1
0
1
>>> i
2

```

SHELL

```

>>> for i in range(2):
...     print(i)
0
1
>>> i
1

```

SHELL

Que s'est-il passé ?

- ▶ La variable de boucle prend la valeur prévue au début de chaque boucle.
- ▶ Elle garde sa dernière valeur en sortant de la boucle.

```

>>> # Comportement étrange
>>> for i in range(30,25,-1):
...     print(i,end=';')
...     i=i+100
...     print(i)
30;130
29;129
28;128
27;127
26;126
>>> i
126

```

SHELL

- ▶ Afficher les entiers de 9 à 0 : méthode universelle

```
for i in range(10):  
    print(9-i, end=' -> ')
```

SCRIPT

```
9 -> 8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> 0 ->
```

SHELL

- ▶ Afficher les entiers de 9 à 0 : méthode plus directe

```
for i in range(9,-1,-1):# le test est i>-1  
    print(i,end=' -> ')
```

SCRIPT

```
9 -> 8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> 0 ->
```

SHELL

- ▶ Les pas flottants ($i=i+0.1$) sont interdits.

- ▶ Afficher les suites ci-dessous avec des boucles `for`.
 - ▶ 0.0 -> 0.1 -> 0.2 -> ... -> 0.9 -> 1.0
 - ▶ 0.0 -> 0.5 -> 1.0 -> 1.5 ... -> 19.5 -> 20.0
 - ▶ 10 -> 8 -> 6 -> ... -> -8 -> -10
- ▶ Même question avec des boucles `while`.

- ▶ On peut faire une boucle `for` dans une autre boucle `for`.

```
for i in range(5):  
    for j in range(i):  
        print('*',end='')  
    print(': i=',i,sep='')
```

SCRIPT

- ▶ La boucle du `j` affiche des étoiles.
- ▶ La boucle du `i` fait varier le nombre d'étoiles.

```
: i=0  
*: i=1  
**: i=2  
***: i=3  
****: i=4
```

SHELL

- ▶ `for i in range(0)` correspond à une boucle vide.

- ▶ `return` interrompt la fonction, donc toutes les boucles.

```
def plus_petit_div(n):  
    for i in range(2,n):  
        if n%i == 0:  
            return i  
    return n
```

SCRIPT

- ▶ `break` interrompt seulement la boucle en cours

```
for i in range(9,14):  
    for j in range(2, i+1):  
        print(j, end = '-')  
        if i%j == 0:  
            break  
    print('>',i, 'est divisible par', j)
```

SCRIPT

```
2-3-> 9 est divisible par 3  
2-> 10 est divisible par 2  
2-3-4-5-6-7-8-9-10-11-> 11 est divisible par 11  
2-> 12 est divisible par 2  
2-3-4-5-6-7-8-9-10-11-12-13-> 13 est divisible par 13
```

SHELL



SCRIPT

```
for i in range(10):      # dizaines
    for j in range(10): # unités
        print(i,j, sep='', end=' ')
    print('') # retour à la ligne
```

SHELL

```
00 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
```

- Comment pourrait-on écrire un programme équivalent mais avec une seule boucle `for`? Essayez de le faire!

- Partie I. Compléments
- Partie II. Boucles for
- Partie III. Chaînes de caractères
- Partie IV. Manipuler les chaînes
- Partie V. Représentation des caractères
- Partie VI. Cryptologie
- Partie VII. Table des matières

- ▶ Un texte est une suite finie de caractères : lettres capitales ou minuscules, chiffres, ponctuations, espaces, symboles mathématiques, etc.
- ▶ En programmation, un texte est une chaîne (de caractères) : anglais *string*

```
texte = 'Cette phrase est fausse'
```

SCRIPT

- ▶ Ne pas confondre chaînes et nombres.

```
>>> 12=='12'  
False  
>>> type(12)  
<class 'int'>  
>>> type('12')  
<class 'str'>
```

SHELL

- ▶ Il existe des fonctions de conversions entre `int` et `str`

```
>>> str(12)  
'12'  
>>> int('12')  
12
```

SHELL

- ▶ Un chaîne peut être définie avec des guillemets simples ' ou double "

```
>>> 'Coucou' == "Coucou"  
True
```

SHELL

- ▶ Mais comment représenter les symboles guillemets ?

```
>>> texte = 'C'est une bonne question !'  
File "<console>", line 1  
    texte = 'C'est une bonne question !'  
                                         ^
```

SHELL

```
SyntaxError: unterminated string literal (detected at line  
1)
```

- ▶ Deux solutions

- ▶ On utilise un échappement \'
- ▶ On utilise des guillemets doubles "

```
>>> 'C\'est la bonne réponse'  
"C'est la bonne réponse"  
>>> "C'est aussi la bonne réponse"  
"C'est aussi la bonne réponse"
```

SCRIPT

- ▶ Une chaîne est une suite de caractères
- ▶ La longueur (*length*) d'une chaîne est le nombre de caractères.

```
>>> texte='123 nous allons au bois :-)'  
>>> len(texte)  
27
```

SHELL

- ▶ Les caractères sont indexés de 0 à `len(texte)-1`.

```
>>> texte[0]  
'1'  
>>> texte[3]  
' '  
>>> texte[len(texte)-1]  
)'  
>>> texte[len(texte)]  
Traceback (most recent call last):  
  File "<console>", line 1, in <module>  
IndexError: string index out of range
```

SHELL

- ▶ Techniquement, sous Python, les chaînes sont indexées :

de `-len(texte)` à `len(texte)-1`

Exemple : `texte='123 soleil'`

<code>texte[i]</code>	'1'	'2'	'3'	' '	's'	'o'	'l'	'e'	'i'	'l'
<code>indice</code>	0	1	2	3	4	5	6	7	8	9
<code>indice</code>	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> texte='123 soleil'
>>> texte[-1]
'1'
>>> texte[-10]
'1'
>>> texte[-11]
Traceback (most recent call last):
  File "<console>", line 1, in <module>
IndexError: string index out of range
```

SHELL

- ▶ En pratique `texte[-k]` est un raccourci pour `texte[len(texte)-k]`.

- ▶ On ne peut pas modifier le contenu d'une chaîne : c'est un objet **immuable**

```
>>> texte[0] = 'Z'  
Traceback (most recent call last):  
  File "<console>", line 1, in <module>  
TypeError: 'str' object does not support item assignment
```

SHELL

- ▶ On peut, par contre, affecter une autre chaîne à une variable

```
>>> texte='123 soleil'  
>>> texte='Z23 soleil'
```

SHELL

- ▶ On peut concaténer plusieurs chaînes

```
>>> "0+0" + '=' + 'θττ'  
'0+0=θττ'
```

SCRIPT

- ▶ Et les répéter plusieurs fois

```
>>> 'A' + 20*'a' + 'h' + '!'*5  
'Aaaaaaaaaaaaaaaaaaaaaah!!!!'
```

SHELL

```
def nombre_caractères(lettre, chaîne):  
    compteur = 0  
    for i in range(len(chaîne)):  
        if chaîne[i] == lettre :  
            compteur = compteur + 1  
    return compteur
```

SCRIPT

```
>>> nombre_caractères('e', 'exceptionnellement')  
5
```

SHELL

- ▶ Un caractère est une chaîne de caractères de longueur 1

```
>>> 'e'[0]=='e' # 1er caractère de 'e' == La chaîne 'e'  
True  
>>> type('e'[0])  
<class 'str'>
```

SHELL

- ▶ On teste l'égalité des chaînes avec ==

```
>>> '0'=='0' # lettre 0 == chiffre Zéro  
False
```

SHELL

- ▶ Index de première apparition d'un caractère (-1 si absent)

```
def position(lettre, chaîne):  
    for i in range(len(chaîne)):  
        if chaîne[i] == lettre:  
            return i  
    return -1
```

SCRIPT

```
>>> position('a', 'Koala')  
2  
>>> position('A', 'Koala')  
-1
```

SHELL

- ▶ Le caractère est-il présent dans la chaîne ?

```
def appartient(lettre, chaîne):  
    return position(lettre, chaîne) >= 0
```

SCRIPT

```
>>> appartient('a', 'Koala')  
True  
>>> appartient('A', 'Koala')  
False
```

SHELL

- ▶ On peut itérer directement sur les caractères! (`for car in chaîne:`)

```
def appartient(lettre, chaîne):  
    for car in chaîne:  
        if car == lettre:  
            return True  
    return False
```

SCRIPT

- ▶ Mais ce n'est pas aussi généraliste.

```
def miroir(chaîne):  
    n=len(chaîne)  
    for i in range(n):  
        print(chaîne[n-1-i],end='')  
    print('')
```

SCRIPT

```
>>> miroir('UN RATS REPUS')  
SUPER STAR NU
```

SHELL

- ▶ Écrire une fonction qui affiche une chaîne de caractères, en séparant les symboles par des tirets.

```
>>> tiret('abcdef')  
a-b-c-d-e-f
```

SHELL

- ▶ On fera attention à ce qu'il n'y ait pas de tiret après le dernier caractère.
- ▶ Même exercice mais cette fois-ci en renvoyant la chaîne.

```
>>> renvoie_tiret('abcdef')  
'a-b-c-d-e-f'
```

SHELL

- 🍃 Partie I. Compléments
- 🍃 Partie II. Boucles for
- 🍃 Partie III. Chaînes de caractères
- 🍃 **Partie IV. Manipuler les chaînes**
- 🍃 Partie V. Représentation des caractères
- 🍃 Partie VI. Cryptologie
- 🍃 Partie VII. Table des matières

Python permet d'extraire une tranche (*slice*) d'une chaîne de caractères, repérée par ses positions extrêmes.

```
>>> texte = '123 nous allons au bois'
>>> texte[4]
'n'
>>> texte[12]
'o'
>>> texte[4:12] # de texte[4] à texte[11] !
'nous all'
>>> texte[4:]
'nous allons au bois'
>>> texte[:12]
'123 nous all'
```

SHELL

► `texte[a:b]` signifie la sous-chaîne d'indice $i \in [a, b - 1]$

- On peut même préciser un pas.

```
texte='abcABC123abcABCfff'
```

```
>>> texte[::3]
```

```
'aA1aAf'
```

```
>>> texte[4:12:3]
```

```
'B2b'
```

```
>>> texte[2::3]
```

```
'cC3cCf'
```

SHELL

```
def extraire(chaîne,début,fin,pas):
```

```
    tranche=''
```

```
    for i in range(début,fin,pas):
```

```
        tranche=tranche+chaîne[i]
```

```
    return tranche
```

SCRIPT

```
>>> extraire(texte,0,len(texte),3)
```

```
'aA1aAf'
```

```
>>> extraire(texte,4,12,3)
```

```
'B2b'
```

```
>>> extraire(texte,2,len(texte),3)
```

```
'cC3cCf'
```

SHELL

- ▶ La fonction `appartient` n'est autre que l'opérateur `in` de Python.

```
>>> 'a' in 'Koala'  
True
```

SHELL

- ▶ `for i in range(len(chaîne))` : `i` entier de [0, `len(chaîne)-1`]

```
def voyelles(chaîne): # compte le nombre de voyelles  
    res=0  
    for i in range(len(chaîne)):  
        if appartient(chaîne[i], 'aeiouy'):  
            res=res+1  
    return res
```

SCRIPT

- ▶ `for c in chaîne` : `c` caractère de « chaîne »

```
def voyelles(chaîne):  
    res=0  
    for c in chaîne:  
        if c in 'aeiouy':  
            res=res+1  
    return res
```

SCRIPT

```
>>> voyelles('Python')  
2  
>>> voyelles("Ajourné") # Oups...  
2
```

SHELL

- ▶ Python est un langage dont les types sont des **classes**.

```
>>> type(23)
<class 'int'>
>>> type('Coucou')
<class 'str'>
```

SHELL

- ▶ Un membre d'une classe est un **objet**.
 - ▶ 23 est un objet de la classe int
 - ▶ 'Coucou' est un objet de la classe str
- ▶ La programmation **orientée objet** est un style de programmation qui dépasse le cadre de ce cours, mais nous en verrons les rudiments.
- ▶ Une des principales différences est que des fonctions peuvent être attachées à un objet. On parle de **méthodes** et on utilise une syntaxe particulière.

```
>>> position('c', 'On appelle une fonction')
18
>>> 'On appelle une méthode'.find('c')
-1
```

SHELL

SHELL

```
>>> '123'.isdigit()           # que des chiffres ?
True
>>> 'Monde'.isalpha()        # que des lettres ?
True
>>> 'rayon+2'.islower()      # lettres minuscules ?
True
>>> 'RAYON+2'.isupper()      # lettres capitales ?
True
>>> 'RAYON+2'.lower()        # mettre en minuscules
'rayon+2'
>>> 'rayon+2'.upper()        # mettre en capitales
'RAYON+2'
>>> '  Olivier  '.strip()     # enlever les blancs aux bouts
'Olivier'
>>> 'abracadabra'.strip('ba') # enlever aux bouts b et a
'racadabr'
>>> 'Zip chic'.replace('ip','o') # remplacement
'Zo chic'
```

- ▶ Il y a 3 délimiteurs de chaînes : `'toto'` == `"toto"` == `"""toto"""`
 - ▶ Les guillemets triples permettent de documenter une fonction.
 - ▶ Cette `docstring` peut tenir sur plusieurs lignes.

SCRIPT

```
def somme_chiffre(n):  
    """Retourne la somme des chiffres de n en base 10  
    Exemple 621 -> 6+2+1 -> 9"""  
    res=0  
    while n>0:  
        res=res+n%10  
        n=n//10  
    return res
```

- ▶ N'intervient pas dans l'exécution, mais permet d'obtenir de l'aide.

SHELL

```
>>> somme_chiffre(3041) # 3 + 0 + 4 + 1  
8  
>>> help(somme_chiffre)  
Help on function somme_chiffre:  
  
somme_chiffre(n)  
    Retourne la somme des chiffres de n en base 10  
    Exemple 621 -> 6+2+1 -> 9
```



Déconseillé aux moins de 18 ans.



- ▶ Dans certaines applications avancées, on peut évaluer une expression stockée dans une chaîne.

```
>>> eval('1+1')
2
```

SHELL

- ▶ Application amusante

```
def évaluer(f,x):
    """ Méthode de goret """
    c=f.replace('x',str(x))
    return eval(c)
```

SCRIPT

```
>>> f='2*x**2+x+1'
>>> évaluer(f,5)
56
>>> évaluer(f,0)
1
>>> évaluer(f,'0 or méchant()#')
Autodestruction de la machine
```

SHELL

- ▶ Peut poser des problèmes de sécurité.
 - ▶ En python `False==0!` (Oui, je sais, c'est très moche...)
 - ▶ On a pu exécuter le code malicieux voulu — ici la fonction `méchant()`.

- 🌿 Partie I. Compléments
- 🌿 Partie II. Boucles for
- 🌿 Partie III. Chaînes de caractères
- 🌿 Partie IV. Manipuler les chaînes
- 🌿 **Partie V. Représentation des caractères**
- 🌿 Partie VI. Cryptologie
- 🌿 Partie VII. Table des matières

- ▶ Un ordinateur ne comprend que les nombres.
- ▶ Un caractère est codé sur la machine comme un nombre.
- ▶ Les caractères américains sont numérotés de 0 à 127, c'est le code ASCII (AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE).

```
>>> ord('A')
65
>>> ord('a')
97
>>> ord('1')
49
>>> ord(' ')
32
```

SHELL

```
>>> chr(65)
'A'
>>> chr(64)
'@'
>>> chr(10) # new line
'\n'
>>> chr(7) # bip !
'\x07'
```

SHELL

SHELL

```
>>> ascii()
 32 33 ! 34 " 35 # 36 $ 37 % 38 & 39 '
 40 ( 41 ) 42 * 43 + 44 , 45 - 46 . 47 /
 48 0 49 1 50 2 51 3 52 4 53 5 54 6 55 7
 56 8 57 9 58 : 59 ; 60 < 61 = 62 > 63 ?
 64 @ 65 A 66 B 67 C 68 D 69 E 70 F 71 G
 72 H 73 I 74 J 75 K 76 L 77 M 78 N 79 O
 80 P 81 Q 82 R 83 S 84 T 85 U 86 V 87 W
 88 X 89 Y 90 Z 91 [ 92 \ 93 ] 94 ^ 95 _
 96 ` 97 a 98 b 99 c 100 d 101 e 102 f 103 g
104 h 105 i 106 j 107 k 108 l 109 m 110 n 111 o
112 p 113 q 114 r 115 s 116 t 117 u 118 v 119 w
120 x 121 y 122 z 123 { 124 | 125 } 126 ~
```

- ▶ L'affichage de la table est programmée en Python.
- ▶ Vivement le prochain TP : ce sera à vous de le faire !

- ▶ Il y a seulement 127 caractères ASCII (chacun codé sur 7 bits) ce qui n'utilise que la moitié des 256 octets disponibles (de 0 à 127).
- ▶ Il reste l'autre moitié (de 128 à 255) pour faire n'importe quoi !
- ▶ Chaque pays, chaque région du monde, a codé les caractères dont il avait besoin sur les 128 autres. Notre norme à nous était l'ISO-Latin-1

SHELL

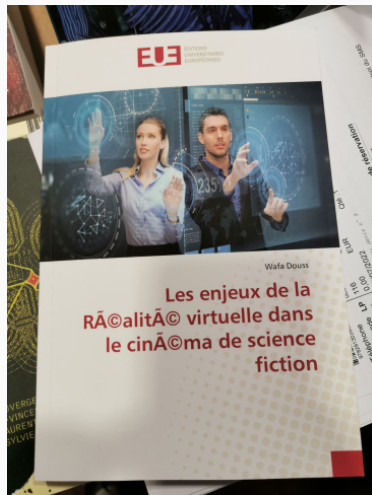
>>> iso_latin_1()

160	161	¡	162	¢	163	£	164	¤	165	¥	166	¦	167	§	
168	¨	169	©	170	ª	171	«	172	¬	173		174	®	175	¯
176	°	177	±	178	²	179	³	180	´	181	µ	182	¶	183	·
184	¸	185	¹	186	º	187	»	188	¼	189	½	190	¾	191	¸
192	À	193	Á	194	Â	195	Ã	196	Ä	197	Å	198	Æ	199	Ç
200	È	201	É	202	Ê	203	Ë	204	Ì	205	Í	206	Î	207	Ï
208	Ð	209	Ñ	210	Ò	211	Ó	212	Ô	213	Õ	214	Ö	215	×
216	Ø	217	Ù	218	Ú	219	Û	220	Ü	221	Ý	222	Þ	223	ß
224	à	225	á	226	â	227	ã	228	ä	229	å	230	æ	231	ç
232	è	233	é	234	ê	235	ë	236	ì	237	í	238	î	239	ï
240	ð	241	ñ	242	ò	243	ó	244	ô	245	õ	246	ö	247	÷
248	ø	249	ù	250	ú	251	û	252	ü	253	ý				

- ▶ Ce fut un âge obscur avec de nombreux problèmes de codage.

Unicode ou Latin-1 ?

De l'importance des accents



- ▶ Unicode (≈ 1990) a proposé d'abandonner la limitation à 255 caractères, en traitant toutes les langues du monde (> 65000 caractères!).

```

grec = "Πυθαγόρας και Πύθων"
espéranto = "Ĉirkaŭaĵo en esperanto"
emoji = "J'💕🐸"
chinois = "\u6211\u662f\u6cd5\u56fd\u4eba"
arabe="حفاث"
sumérien = "𐎶𐎵 𐎶"

print(grec,espéranto,chinois,sumérien,arabe,emoji,sep='\n\n')

print()
for i in range(0x0661,0x066a):
    print(chr(i), end=" ")
print()

```

```

Πυθαγόρας και Πύθων
Ĉirkaŭaĵo en esperanto
我是法国人
𐎶𐎵 𐎶
حفاث
J'💕🐸
٩ ٨ ٧ ٦ ٥ ٤ ٣ ٢ ١

```

- ▶ Grec, Arabe, Chinois, Géorgien, Inuit, Cunéiforme, Emoji!
- ▶ Chaque caractère a un numéro unicode unique.

- ▶ La base 16 utilise 16 chiffres : **0123456789abcdef** ($a = 10, b = 11 \dots f = 15$)
- ▶ Chaque caractère unicode peut être représenté par son numéro en base 16.

```

>>> a # a='\u20ac'
'€'
>>> ord('€') # Numéro unicode de '€' en base 10
8364
>>> 2*16**3 + 10*16**1 + 12*16**0 # a=10 et c=12
8364
>>> chr(8364)
'€'
>>> chr(0x20ac)
'€'

```

SHELL

- ▶ '€' a pour numéro unicode $8364_{10} = 20ac_{16}$ obtenue
 - ▶ avec `ord('€')`
 - ▶ sur internet <http://www.unicode.org/>
- ▶ On peut le saisir sous différents formats :

'\u20ac' chr(8364) chr(0x20ac) ou simplement '€'

- ▶ Comme les caractères Unicode ne tiennent plus tous sur un seul octet, certains caractères sont représentés sur 2, 3, voire 4 octets. Il y a plusieurs conventions d'encodage possibles pour un même caractère.
- ▶ UTF-8 : le standard.
 - ▶ code les 127 premiers caractères comme ASCII
 - ▶ tout texte ASCII est un texte UTF-8!
 - ▶ Les autres caractères sont sur plusieurs octets.
 - ▶ 'é' : deux octets. Caractères chinois : trois octets.
- ▶ UTF-16 : Windows (qui n'aime pas faire comme les autres)
- ▶ UTF-32 : Tous les caractères ont la même taille : 32 bits.
- ▶ Le numéro unicode ne correspond pas (exactement) au codage UTF-8!
- ▶ Unicode est un annuaire de caractères ; UTF-8 est un codage machine.

- 🍃 Partie I. Compléments
- 🍃 Partie II. Boucles for
- 🍃 Partie III. Chaînes de caractères
- 🍃 Partie IV. Manipuler les chaînes
- 🍃 Partie V. Représentation des caractères
- 🍃 **Partie VI. Cryptologie**
- 🍃 Partie VII. Table des matières

- ▶ La **cryptographie** est l'art d'écrire et de lire des codes secrets.
 - ▶ Alice **chiffre** un message pour Bob.
 - ▶ Bob **déchiffre** le message d'Alice.

- ▶ La **cryptanalyse** est l'art de casser les codes secrets sans connaître la clé.
 - ▶ Eve, la méchante, **décrypte** la conversation entre Alice et Bob.

- ▶ La **cryptologie** réunit la cryptographie et la cryptanalyse.

- ▶ Remarque : crypter un message n'a aucun sens...
...sauf apparemment pour les journalistes.

Une des premières formes de chiffrements d'un message est le chiffrement par décalage circulaire, appelé code de César.

- ▶ César l'aurait utilisé pour ses correspondances durant la Guerre des Gaules.
- ▶ Principe : on transforme lettre par lettre le message.

Exemple : $A \rightarrow S, B \rightarrow T, C \rightarrow U, \dots, Z \rightarrow R.$

Le message ZEBRA devient RWTjS.

- ▶ Ici on décale de 18 lettres.
- ▶ $k = 18$ est la clé de chiffrement.
- ▶ On peut choisir n'importe quelle clé entre 1 et 25.
- ▶ César utilisait la clé $k = 3$.



- ▶ On ne code que les CAPITALES.
- ▶ Les étapes pour chiffrer UNE lettre.
 - ▶ On calcule l'index i de la lettre de l'alphabet avec `ord`.
 - ▶ On ajoute la clé à l'index, modulo 26 (la taille de l'alphabet) : $(i+k)\%26$
 - ▶ On calcule la lettre à cet index dans l'alphabet avec `chr`, en utilisant le code ASCII.
- ▶ Pour chiffrer un message de plusieurs lettres : on chiffre lettre à lettre en stockant le résultat dans une chaîne.

- ▶ Le déchiffrement est très simple :
 - ▶ il suffit de connaître la clé k utilisée pour le chiffrement
 - ▶ On décale alors de $-k$ (modulo 26) pour retrouver le message de départ
- ▶ Et si on ne connaît pas la clé k ? La cryptanalyse du code de César est possible...en testant toutes les clés (vivement le TP!).
- ▶ On parle de **cryptographie symétrique** (ou à clé secrète), la même clé permet de chiffrer et déchiffrer. Mais il faut échanger cette clé secrète à l'avance (et ne pas la révéler à tout le monde).
- ▶ Pour échanger la clé symétrique, les techniques modernes reposent sur la **cryptographie asymétrique** (ou à clé publique).
 - ▶ Bob a deux clés : sa clé privée et sa clé publique.
 - ▶ Alice utilise la **clé publique** de Bob pour lui écrire un message **chiffré**.
 - ▶ Bob utilise sa **clé privée** pour **déchiffrer**.
 - ▶ La clé publique ne permet pas de déchiffrer le message.

Merci pour votre attention

Questions



Cours 3 — Boucles For et chaînes de caractères

Partie I. Compléments

Compléments sur le `print`

Partie II. Boucles `for`

Rappels sur la boucle `while`

La boucle `for` : principe

La boucle `for` : début et pas

Boucle `for` : synthèse

Différence entre `while` et `for`

Les pas négatifs

 Exercice

Boucles imbriquées

Sorties de boucles

 Afficher les entiers de 0 à 99

Partie III. Chaînes de caractères

Caractères et chaînes de caractères

Guillemets simples et doubles

De la chaîne aux caractères

Index de chaîne

Opérations sur les chaînes

Parcours d'une chaîne de caractères

Exemples

Variantes

 Exercices

Partie IV. Manipuler les chaînes

Extraction d'une sous-chaîne (1/2)

Extraction d'une sous-chaîne (2/2)

Parcours de chaîne : variante

Des fonctions étranges

Exemples de méthodes de la classe `str`

Documenter une fonction

Exécuter une chaîne

Partie V. Représentation des caractères

Le codage ASCII

La table ASCII

Autres caractères

Âges obscures

Unicode

Codage unicode

Des caractères aux codages machines

Partie VI. Cryptologie

Définitions

Le code de César

César en Python

Déchiffrer un message

Partie VII. Table des matières