

La boucle for

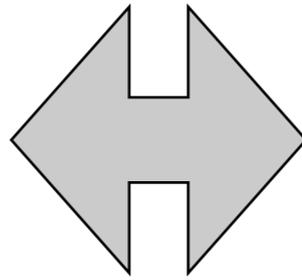
&

Les string



La boucle `for` utilisée avec `range`

```
i = 1
while i < 10 :
    print(i)
    i = i + 2
```



```
for i in range(1,10,2) :
    print(i)
```

On peut omettre le **pas** et la **valeur initiale**

```
>>> for i in range(1,10) : print(i, end = ' ')
123456789
>>> for i in range(10) : print(i, end = ' ')
0123456789
```

```
>>> i = 42
>>> for i in range(10,0,-1) :
...     i = i - 1
...     print(i, end = ' ')
9876543210
>>> i
0
```

La **variable de boucle** prend une nouvelle valeur à chaque tour de boucle

Boucles imbriquées, sortie de boucle

On peut mettre une boucle for à l'intérieur d'une autre

```
>>> for i in range(3) :  
...     for j in range(i) :  
...         print(j , end = ' ')  
...     print()
```

0

01

break et **return** causent une sortie de boucle

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

```
>>> for i in range(10) :  
...     for j in range(2,i + 1) :  
...         if i % j == 0 :  
...             break  
...     print(j , end = ' ')  
0023252723
```

Différence importante : **break** interrompt la boucle,
return interrompt la *fonction* : donc toutes les boucles

Les chaînes de caractères

- Un **texte** est une **suite finie de caractères** : lettres majuscules ou minuscules, chiffres, ponctuations, symboles mathématiques, etc.
- Dans les langages de programmation, un texte est une **chaîne** (de caractères). *In the world of programming languages, a text is a **string**.*

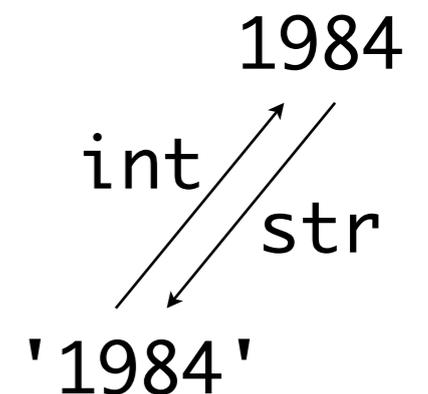
```
>>> texte = '1984 : Orwell !'  
>>> texte  
'1984 : Orwell !'
```

'Notez les apostrophes !'

- Ne confondez pas 1984 qui est un entier et '1984' qui est une chaîne.

```
>>> type(1984)  
<class 'int'>  
>>> type('1984')  
<class 'str'>
```

```
>> str(1984)  
'1984'  
>>> int('1984')  
1984
```



- Les apostrophes simples sont préférées, mais on peut aussi utiliser des guillemets :

```
>>> texte = "1984 : Orwell !"
>>> texte
'1984 : Orwell !'
```

- Pour inclure une apostrophe dans une chaîne, attention !

```
>>> 'Mais j'ai froid !'
SyntaxError: invalid syntax
```

- Deux méthodes :

- utiliser des guillemets pour délimiter la chaîne :

```
>>> "Mais j'ai froid !"
"Mais j'ai froid !"
```

- utiliser le caractère d'échappement *backslash* \

```
>>> 'Mais j\'ai froid !'
"Mais j'ai froid !"
```

Une chaîne est une suite de caractères

- Combien y a-t-il de caractères dans une chaîne ?

```
>>> texte = '1984 : Orwell !'  
>>> len(texte)  
15
```

length...

- Comment accéder au caractère numéro k (k = 0, 1, 2, ...)?

```
>>> texte[0]  
'1'  
>>> texte[3]  
'4'  
>>> texte[len(texte)-1]  
'!'  
>>> texte[-1]  
'!'  
>>> texte[len(texte)]  
IndexError: string index out of range
```

Hello

0 1 2 3 4
-5 -4 -3 -2 -1

```
>>> s = 'Hello'  
>>> s[-1]  
'o'  
>>> s[-5]  
'H'
```

- Puis-je *modifier* le caractère numéro k d'une chaîne ?

NON : **une chaîne est un objet constant, non mutable**.

On peut modifier la valeur de la variable texte avec une affectation, mais pas l'un de ses caractères individuellement.

```
>>> texte = 'Je dis ' + texte
>>> texte
'Je dis 1984 : Orwell !'
>>> texte[0] = 'K'    #j'essaye de transformer J en K
TypeError: 'str' object does not support item assignment
```

- L'opérateur + entre chaînes se nomme la **concaténation**.

```
>>> 'Sa' + '-' + 'lut'
'Sa-lut'
```

```
>>> '17' + '89'
'1789'
```

- L'opérateur * permet de répéter une chaîne :

```
>>> 3 * 'guili'    # 'guili' + 'guili' + 'guili'
'guiliguiliguili'
```

Parcourir une chaîne de caractères

- Exemple : nombre d'apparitions du caractère c dans une chaîne s

```
def nb_char(c,s) :  
    res = 0  
    for i in range(len(s)) :  
        if s[i] == c :  
            res = res + 1  
    return res
```

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

- Que remarque-t-on ?

Un caractère est identifié à une chaîne de longueur 1.

Le caractère 'e'

On teste l'égalité de deux caractères avec ==

```
>>> 'e' == 'E'  
False
```

Parcourir une chaîne de caractères

- Exemple. Quelle est la position de la première apparition du caractère `c` dans la chaîne `s` ? Ou bien `-1` s'il n'est pas dans `s`...

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

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

```
def appartient(c,s) :  
    return position(c,s) >= 0
```

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

astuce pythonnesque : on peut itérer directement sur les caractères!

```
def appartient(c,s) :  
    for c' in s :  
        if c == c' : return True  
    return False
```


Extraction d'une tranche de sous-chaîne

- Python propose un moyen d'extraire une **tranche** (*slice*) d'une chaîne repérée par ses positions extrêmes :

```
>>> texte
'Salut le monde !'
>>> texte[2:5]
'lut'
```

```
>>> texte[:8]
'Salut le'
>>> texte[2:]
'lut le monde !'
```

- Attention donc, $s[i:j]$ signifie la tranche de la chaîne s située entre les indices i et $j-1$, donc dans $[i, j[$. Pour prendre seulement 1 caractère sur k , demander $s[i:j:k]$.

```
>>> texte
'abcdefghijklmnopqrstuvwxyz'
>>> texte[:6] + texte[6:]
'abcdefghijklmnopqrstuvwxyz'
```

```
>>> texte[3:15:3]
'dgj m'
>>> texte[::3]
'adgj mpsvy'
>>> texte[3::3]
'dgj mpsvy'
```

Beaucoup de primitives sur les chaînes...

- Un langage de programmation vaut aussi par l'étendue de ses fonctionnalités. Avant de programmer, cherchez dans la doc si la fonction convoitée n'est pas une primitive !
- EXEMPLE. La fonction `appartient(c, s)` précédente n'est autre que l'opérateur primitif `in` de Python :

`appartient(c, s) <==> c in s`

- Le nombre de voyelles d'une chaîne `s` se programmerait (naïvement) :

```
def nb_voyelles(s) :  
    res = 0  
    for i in range(len(s)) :  
        if s[i] in 'aeiouy' : res = res + 1  
    return res
```

```
>>> nb_voyelles('Ajourné')  
2
```

naïf...

...mais certaines ont une drôle de tête !

- Nous commençons à pénétrer dans un continent qu'il faudra tôt ou tard aborder : celui des **OBJETS**. Nous garderons pour l'instant une idée naïve de ce qu'est un *objet*. Il suffit de savoir qu'on peut lui envoyer un **message** à l'aide d'une **méthode**.
- **EXEMPLE**. Je trouve dans la documentation la primitive `find` qui réalise ma fonction `position(c, s)`. Mais ce n'est pas une *fonction*, c'est une **méthode** ! Elle envoie un message à un objet de la classe `str`.

fonction
~~`find(c, s)`~~

méthode

`s.find(c)`

```
>>> texte = 'Salut le monde !'  
>>> texte.find('l')  
2  
>>> texte.find('mon')  
9
```

s, donne-moi l'indice de c !

- Autres exemples de méthodes dans la classe str des chaînes :

```
>>> '123'.isdigit()           # que des chiffres ?  
True  
>>> 'Monde'.isalpha()        # que des lettres ?  
True  
>>> 'rayon+2'.islower()      # lettres minuscules ?  
True  
>>> 'RAYON+2'.isupper()      # lettres majuscules ?  
True  
>>> 'RAYON+2'.lower()        # mettre en minuscules  
'rayon+2'  
>>> 'rayon+2'.upper()        # mettre en majuscules  
'RAYON+2'  
>>> '    www.unice.fr    '.strip() # enlever les blancs aux bouts  
'www.unice.fr'  
>>> 'www.unice.fr'.strip('wrf.') # enlever aux bouts  
'unice'  
>>> 'Zip chic'.replace('i','o')  # remplacement  
'Zop choc'
```

Le codage ASCII

- Les caractères américains (dits *internationaux*) sont numérotés de 0 à 127, c'est le **code ASCII** (*American Standard Code for Information Interchange*). Par exemple le code ASCII de 'A' est égal à 65 mais on n'a pas besoin de le mémoriser :

```
>>> ord('A')
65
>>> ord('a')
97
```

```
>>> ord(' ')
32
>>> ord('?')
63
```

```
>>> chr(65)
'A'
>>> chr(63)
'?'
```

- Les caractères dont le code ASCII tombe entre 0 et 31, ainsi que le caractère numéro 127 ne sont pas *affichables*. Ce sont les **caractères de contrôle** nommés symboliquement NUL, LF, BEL, CR, ESC, DEL...

```
>>> chr(10)
'\n'
```

← LF

symbolique

```
>>> chr(27)
'\x1b'
```

← ESC

hexadécimal

| | | | | | | | | | | | | | | | |
|-----|---|-----|---|-----|---|-----|---|-----|----|-----|---|-----|---|-----|---|
| 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 ci-dessus est programmée en Python :

```
def ascii() :
    for k in range(32,127) :
        ...
```

cf TP!

Le codage ASCII étendu

- Le code du caractère accentué 'é' est 233, il tombe entre 128 et 255. Il s'agit du **code ASCII étendu**, qui contient des caractères propres à chaque pays. Nous utilisons le codage *Iso-Latin-1*, qui couvre les langues de l'Europe de l'Ouest, avec le é, à, ç, ñ, Ö, ß, etc. Les russes utilisent KOI8-R, les chinois sont bien ennuyés, etc.

| | | | | | | | | | | | | | | | |
|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|
| 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 | ± | 186 | ° | 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 | 254 | þ | | |

Une partie standard de Iso-Latin-1

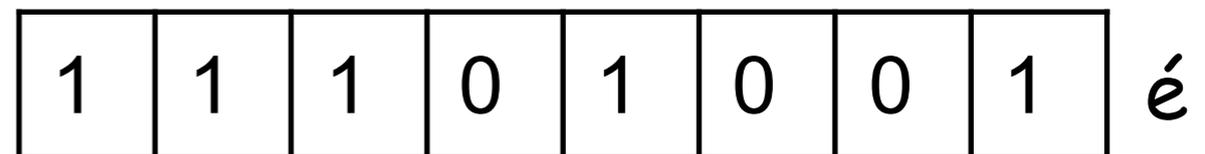
Le codage UNICODE

- Et les chinois furent contents ! En effet, le consortium **Unicode** (\approx 1990, initié par Xerox et Apple) proposait d'abandonner la limitation des 255 caractères en traitant d'un seul coup toutes les langues du monde (plus de 65000 caractères !)...

- *Pourquoi ce nombre magique 255 ?*

- *Parce que l'unité d'information sur un ordinateur est l'**octet** : un bloc de 8 bits (un **bit** = 0 ou 1).*

- Sur les 8 bits, un bit était utilisé pour la vérification de la bonne transmission (le bit de parité), d'où un codage sur 7 bits, c'est l'ASCII. Avec la fiabilité des transmissions, le 8ème bit fut finalement utilisé pour l'ASCII étendu...*



- Les caractères Unicode sont numérotés en **hexadécimal** (base 16).
Par exemple, le symbole *euro* a pour numéro 20AC en hexa :

```
>>> 0x20ac
8364
>>> hex(8364)
'0x20ac'
```

```
>>> '\u20ac'
'€'
>>> chr(8364)
'€'
>>> chr(0x20ac)
'€'
```

```
>>> ord('€')
8364
```

- Sur www.unicode.org, on trouve les *page charts* donnant les numéros de toutes les langues du monde. Par exemple le **chinois** :

```
>>> phrase = '\u6211\u662F\u6cd5\u56fd\u4eba'
>>> phrase
'我是法国人' ←————— Je suis Français.
```

- ou l'**arabe** (mais attention à l'écriture *gauche* ← *droite*)...

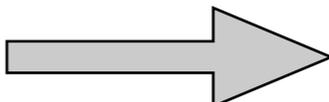
```
for i in range(0x0661,0x066a) :
    print(chr(i),end=' ')
```

٩ ٨ ٧ ٦ ٥ ٤ ٣ ٢ ١

Jules César et les messages secrets

- La **cryptographie** est la science des codes secrets. Une des premières formes de codage d'un message provient de Jules César durant la Guerre des Gaules. On ne code que les MAJUSCULES...
- Pour coder un message MSG :
 - Choisir une clé de codage k dans $[1,25]$.
 - Initialiser la chaîne résultat à la chaîne vide
 - Pour chaque caractère c du message à coder :
 - Si le caractère c est une lettre majuscule :
 - coder le caractère c en le décalant de k positions à droite
 - insérer le nouveau caractère dans la chaîne résultat
 - sinon : insérer c dans la chaîne résultat
 - Rendre en résultat la chaîne résultat

cf TP3...

$k = 3$
MSG = 'TOUS au ZOO !'  'WRXV au CRR !'

Documenter une fonction

- Il y a une troisième manière de délimiter une chaîne : les guillemets triples ! Il s'agit d'une *docstring* permettant de documenter une fonction. Elle peut tenir sur plusieurs lignes.

```
→ def somch(n) :  
    """Retourne la somme des chiffres de n en base 10"""  
    res = 0  
    while n > 0 :  
        res = res + (n % 10) ; n = n // 10  
    return res
```

- Cette *docstring* a été enregistrée et associée au nom de la fonction :

```
> help(somch)  
Help on function somch in module __main__:  
somch(n)  
    Retourne la somme des chiffres de n en base 10
```

Une chaîne peut contenir du code Python !

- Dans certaines applications avancées de Python ou pour les maths, on place une expression Python dans une chaîne str et on provoque son évaluation avec la fonction `eval`.

```
>>> x = 3
>>> eval('x+1')
4
>>> eval('x+1', {'x':10}) + x
14
```

```
def val0() :
    f = eval(input('Entrez une fonction -> '))
    print('Sa valeur en 0 est',f(0))
```

```
>>> val0()
Entrez une fonction -> lambda x : (x+1)/2
Sa valeur en 0 est 0.5
```

- L'usage de cette fonction est réservée à des cas très rares. En effet si la chaîne contient du code Python malicieux, la **sécurité** vacille...

Chaînes de formatage

Au lieu d'écrire

```
'5 * ' + str(i) + ' = ' + str(5 * i)
```

ce qui est peu lisible, on peut écrire

```
'5 * {} = {}'.format(i, 5 * i)
```

On utilise une chaîne de formatage. Outil utile et puissant!

Exemples : on peut aussi numérotter les arguments

```
'5 * {1} = {0}'.format(5 * i, i)
```

On peut les nommer

```
'5 * {b} = {a}'.format(a = 5 * i, b = i)
```

```
d = {'a':5, 'b':5*i}
```

```
'5 * {a} = {b}'.format(**d)
```

Chaînes de formatage

On peut imposer une taille au texte et un style d'alignement.

```
>>> '{0:<30}'.format('aligné gauche')
'aligné gauche'
>>> '{0:>30}'.format('aligné droit')
'                aligné droit'
>>> '{0:^30}'.format('centré')
'                centré'
>>> '{0:*^30}'.format('centré') # remplit avec '*'
'*****centré*****'
```

Enfin, on peut choisir entre différents formats pour afficher un flottant

```
>>> '{pi:.2f}; {a:.3e}; {b:.1%}'.format(pi=pi, a=7**7, b=0.1234)
'3.14; 8.235e+05; 12.3%'
```

et encore d'autres choses... voir la doc en ligne!