



INFO

- Expr^e booléenne $[a, b] \& [c, d]$ st disjointe.
 > $a \leq b$ and $c \leq d$ and ($b < c$ or $d < a$)

• Sans le and:
 > if a
 > if b
 > instructions

• Sans le or:
 > if a
 > instructions
 > elif b
 > instructions

- Nombres de Armstrong : $153 = 1^3 + 5^3 + 3^3$

> def unité(n):
 return $n \% 10$

 > def dizaine(n):
 return $\frac{unité(n)}{10}$

 > def centaine(n):
 return unité($n // 100$)

$$\begin{array}{r} 153 \\ 3 \mid 15 \\ \hline \end{array}$$

> def est_amstrong(n):

> d = centaine(n)
 > e = dizaine(n)
 > f = unité(n)
 > return $n == d^{**3} + e^{**3} + f^{**3}$.

Chaines de caractères

→ Lettre, chiffre, ponctuation, fin de ligne, tableau, retour arrière.

Concatenat : $\ggg \text{"key"} + \text{max}$ Repetit 9: $\ggg \text{ah ahah} * 3$

6 pts affichage: print(racine, repertoire, fichier, sep = "\n")
où racine = "c:" & fichier = "prog.py".

Comparaison: Unicode $\rightarrow [A, Z] \rightarrow [65, 90]$; $[a, z] \rightarrow [97, 122]$

`>>> 'a' < 'b'` True `>>> 'a' < 'B'` False If `a` è un dictiom.

Accès aux caractères d'une chaîne : []

```

>>> ch = "informatique"
>>> ch[2]
'y'
>>> ch[-len(ch)-1]
'e'
>>> ch[-len(ch)]

```

Parcours d'une chaîne

$s = \text{"une chaîne"}$

```
> for c in s:  
>     print(c)
```

mechanical

① > print ("\\n" * 30)

② > def miroir (s) :

> for c in s :

> mire = c + mire

> return mire

③ > def my_len (s) :

> compteur = 0

> for c in s :

> compteur = compteur + 1

> return compteur

① > print ("\\n" * 30)

→ miroir ('manger')

② > def miroir (s):

> mire = "

> for c in s:

> mire = c + mire

> return mire

③ > def my_len (s):

> compteur = 0

> for c in s:

> compteur = compteur + 1

> return compteur

④ > def est_palindrome (s):

> return miroir (s) == s

⑤ > def contains (c, s):

"" " Renvoie vrai si c est dans s " " "

> resultat = False

> for car in s

> if car == c

> resultat = True

> return resultat

Énumérer des entiers : range(a, b, k)

⚠ b exclu, pas de k. (par défaut a=0)

> range(0, 6, 2)
> $\frac{0}{2}$

> $\frac{2}{4}$

> def affiche_vertical(s):
> for i in range(len(s)):
> print(s[i])

> def indice(c, s):

> pos = -1
> for i in range(len(s)):
> if s[i] == c:
> pos = i

> return pos

> def indice](c, s):
> pos = -1
> for i in range(len(s)):
> if s[i] == c and pos == -1
> pos = i
> return pos

def plusFrequent (chaine) :

"" "" "

alphabet = "abcdefghijklmnopqrstuvwxyz"

max_nb = 0

max_lettre = ''

for lettre in alphabet :

 nb = 0

 for car in chaine :

 if car == lettre :

 nb = nb + 1

 if max_nb < nb :

 max_nb = nb

 max_lettre = lettre

return max_lettre

def copie (chaine) :

"" "" "

resultat = ''

for indice in range (len(chaine)) :

 resultat = resultat + chaine [indice]

return resultat

Prédicat : savoir si l'expresstion est bien parenthésée.

def bien_parenthese(s):

i = 0

paren = 0

while i < len(s) and paren >= 0 :

 if s[i] == '(' :

 paren += 1

 elif s[i] == ')' :

 paren -= 1

return paren == 0

Calcul du pgcd.

def pgcd(a, b) :

 resultat = 0

 while b > 0 :

 mod = a % b

 a = b

 b = mod

 return a

$$\begin{cases} \text{pgcd}(a, 0) = a \\ \text{pgcd}(a, b) = \text{pgcd}(b, a \% b) \end{cases}$$

def plus-long-mot(s):

while i < len s:

if s[i] != "

mot = mot + s[i]

nb = nb + 1

else:

if nb > nb_max:

mot_max = mot

nb_max = nb

mot = "

nb = 0

i += 1

return mot_max

Variables à
initialiser:

i = 0

mot = "

nb = 0

mot_max = "

nb_max = 0

Tuple: 5^e séq^{ue}ll: m^{be} qq va p^{re}s n'importe q^ut type
mais da ordie précis.

`>>> (1, 2, 3)` | `>>> (1, 2, 3) == (2, 1, 3)` | `>>> ()` | `>>> (1,)`
`(1, 2, 3)` | `False` | `()` | `(1,)`

`>>> ("Timoleon", False, 1.0)`
`("Timoleon", False, 1.0)`

il faut un seul param-

`>>> tuple('Timoleon')` | `>>> tuple(1, 2, 3)`
`('T', 'i', 'm', 'o', 'l', 'e', 'o', 'n')` | `Error`

`>>> type((1, 2, 3))` | `>>> (1, 2) + (3,)` | `>>> (1, 2) * 3`
< class tuple > | `(1, 2, 3)` | `(1, 2, 1, 2, 1, 2)`

`>>> len((1, 2, 3))` | `>>> t = tuple('Hello')` | `t[-1]`
3 | `t[1]` | `'x'`
`'e'`

Tranches

`>>> t[-1:4]` |  exclu | `>>> t[:3]` | `>>> t[3:]`
`('e', 'l', 'l')` | `'o'` | `('H', 'e', 'l')` | `('l', 'o')`

for i in range(len(t)):
print(t[i])

H
e
l
l
o

→ f copie q' renvoie une copie du tuple passé en paramètre

def copie_1(t):
 return tuple(t)

def copie_3(t):
 return t[:]

par défaut jusqu'à bout

def copie_3(t):
 return tuple(t[i] for i in range(len(t)))

def indice(value, t):

i = 0

while i < len(t) and t[i] != value:

i = i + 1

if i == len(t):

return -1

else:

return i

def occure(value, t):

cpt = 0

for i in range(len(t)):

if t[i] == value:

cpt += 1

return cpt

def insert (value, pos, t):

return t[:pos] + value + t[pos:]