

Rappels : syntaxe python

- ▶ Affectation
- ▶ Expression booléenne
- ▶ Structures de contrôle
- ▶ Boucle

Qu'est-ce qu'un programme?

- ▶ C'est **une suite d'instructions** écrites dans un langage de programmation « compréhensible » par l'ordinateur.
- ▶ Cela permet à l'ordinateur d'appliquer un algorithme.
- ▶ Dans un programme l'ordinateur effectue les instructions **dans l'ordre**.
L'ordre des instructions est donc très important

Qu'est-ce qu'un programme

- ▶ Exemple : afficher les diviseurs de n

Algorithme :

```
si  $n > 0$  alors
  pour tout entier  $i$  entre 1 et  $n$  faire
    si  $n$  est divisible par  $i$  alors
      afficher  $i$ 
    finsi
  finpour
finsi
```

Programme :

```
if  $n > 0$ :
    for  $i$  in range(1,  $n+1$ ):
        if  $n \% i == 0$ :
            print ( $i$ )
```

Variables et affectation

- ▶ Une variable désigne un emplacement mémoire dans lequel on peut stocker (sauvegarder) une **valeur**.
- ▶ Une variable a toujours un **nom**. Vous pouvez choisir n'importe quel nom sauf...
- ▶ Pour **changer la valeur d'une variable** on utilise l'**affectation** représentée par le symbole = en python.

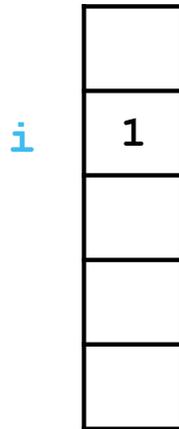
a=3

- ▶ Ce symbole n'a pas la même signification qu'en mathématiques. Il signifie **calculer la valeur à droite du symbole = et la ranger dans la variable dont le nom se trouve à gauche**.

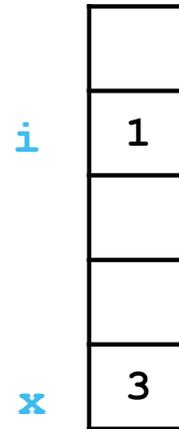
Variables et affectation

▶ Exemples :

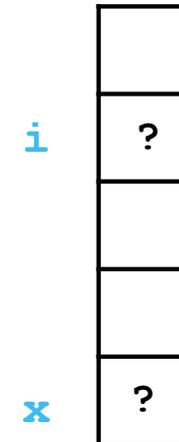
$$i = 1$$



$$x = 2*i+1$$



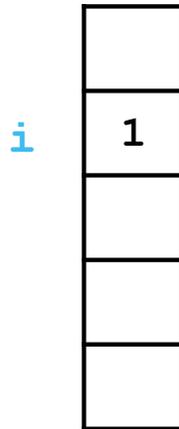
$$i = x+2$$



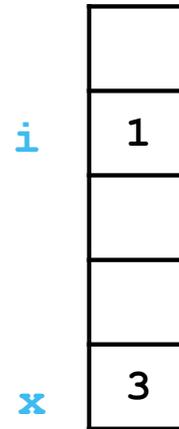
Variables et affectation

▶ Exemples :

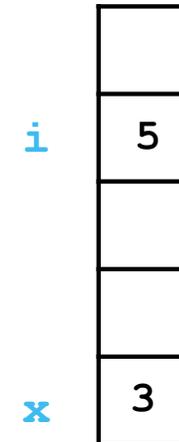
$$i = 1$$



$$x = 2*i+1$$



$$i = x+2$$



Division entière

- ▶ Le **quotient entier** q de deux entiers a et b positifs et le **reste** r sont définis par :

$$a = b * q + r \text{ avec } 0 \leq r < b$$

- ▶ En python :

Quotient entier de a par b : $a // b$

Reste ou modulo : $a \% b$

Exemple :

$19 // 5$ donne 3

$19 \% 5$ donne 4

Exercice : Que vaut $15 \% 21$?

15 car $15 = 0 * 21 + 15$

Expression booléenne

- ▶ Une expression booléenne est une expression qui n'a que 2 valeurs possibles :
 - True** (Vrai)
 - False** (Faux)
- ▶ Les tests sont des expressions booléennes
 - égalité **x==y**
 - inégalité **x != y**
 - comparaison **x<=y x>y** etc.
- ▶ On peut les combiner avec des opérateurs : **and**, **or** et **not**

Expressions booléennes : opérateurs logiques

and	V	F
V	V	F
F	F	F

or	V	F
V	V	V
F	V	F

	V	F
not	F	V

	or	and
not	and	or

Expression booléenne

- ▶ $x \geq -5$ and $x \leq 5$ vaut **True** si $x \in [-5, 5]$

Exercices :

1. Ecrire une expression booléenne qui vaut **True** si $x \notin [-5, 5]$.

$\text{not}(x \geq -5 \text{ and } x \leq 5)$

ou

$x < -5 \text{ or } x > 5$

2. Ecrire une expression booléenne qui vaut **True** si i est un nombre pair inférieur à 6.

$i \% 2 == 0 \text{ and } i < 6$

3. On considère deux variables n et p . Ecrire une expression booléenne qui vaut **True** si n divise p

$p \% n == 0$

Expression booléenne

Rappel loi de Morgan :

- ▶ `non (A ou B) = non (A) et non (B)`
- ▶ `non (A et B) = non (A) ou non (B)`

```
B = ((not (a or b)) == ((not a) and (not b)))  
# vaut True
```

```
B = ((not (a and b)) == ((not a) or (not b)))  
# vaut True
```

Expression booléenne

Exercice :

Soit r , g et b trois entiers quelle expression sera vraie si $(r,g,b) \neq (0,255,0)$:

1. $(r,g,b) \neq (0,255,0)$? ✓
2. $r \neq 0$ and $g \neq 255$ and $b \neq 0$? ✗
(0,0,0) produira un faux !
3. $\text{not}(r == 0 \text{ and } g == 255 \text{ and } b == 0)$? ✓
4. Autre idée ?
 $r \neq 0$ or $g \neq 255$ or $b \neq 0$ ✓

Expression booléenne : souvenir !

Ecrire une fonction

`incruster (imagePersonnage, imageFond)`

qui incruste l'image `imagePersonnage` dans l'image `imageFond`.

Cette dernière image sera modifiée par la fonction. On place l'image du personnage en haut à gauche de l'image de fond (on fait correspondre leurs coins supérieurs gauches).

On suppose que le fond vert qui ne doit pas apparaître dans l'image résultat est composé uniquement de pixels de couleur (0, 255, 0).

On suppose également que l'image de fond est assez grande pour contenir toute l'image à incruster. (DST Janvier 2017)

Expression booléenne : souvenir !

Ecrire une fonction

`incruster (imagePersonnage, imageFond)`



Idée :

Pour chaque pixel de `imagePersonnage` si sa couleur (r,g,b) est différente de $(0,255,0)$ on met sa couleur dans `imageFond` aux mêmes coordonnées.

Comment écrit-on $(r,g,b) \neq (0,255,0)$?

Expression booléenne : souvenir !

Ecrire une fonction

`incruster (imagePersonnage, imageFond`



Comment écrit-on $(r,g,b) \neq (0,255,0)$:

- ▶ $(r,g,b) \neq (0,255,0)$? ✓
- ▶ $r \neq 0$ and $g \neq 255$ and $b \neq 0$? ✗
- ▶ $\text{not}(r == 0$ and $g == 255$ and $b == 0)$? ✓
- ▶ Autre idée ?

Expression booléenne : souvenir !

Ecrire une fonction

`incruster (imagePersonnage, imageFond`



Comment écrit-on $(r,g,b) \neq (0,255,0)$:

- ▶ $(r,g,b) \neq (0,255,0)$? ✓
- ▶ $r \neq 0$ and $g \neq 255$ and $b \neq 0$? ✗
- ▶ $\text{not}(r == 0$ and $g == 255$ and $b == 0)$? ✓
- ▶ Si on distribue le **not** :
 $r \neq 0$ or $g \neq 255$ or $b \neq 0$ ✓

Expression booléenne : souvenir !

Ecrire une fonction

`incruster (imagePersonnage, imageFond`



Comment écrit-on $(r,g,b) \neq (0,255,0)$:

- ▶ $(r,g,b) \neq (0,255,0)$? ✓
- ▶ $r \neq 0$ and $g \neq 255$ and $b \neq 0$? ✗ Le noir est vert !
- ▶ $\text{not}(r == 0 \text{ and } g == 255 \text{ and } b == 0)$? ✓
- ▶ Si on distribue le `not` :
 $r \neq 0$ or $g \neq 255$ or $b \neq 0$ ✓

Evaluation paresseuse

- ▶ L'évaluation des expressions se fait de gauche à droite
- ▶ Elle s'arrête dès que possible, on dit que **l'évaluation est paresseuse**.



```
a=0
```

```
b=(a!=0 and (d==c)) #vaut False
```

```
a=1
```

```
b=(a!=0 or (d==c)) #vaut True
```



Dans les 2 cas `d==c` n'est pas évaluée tant mieux car `c` et `d` ne sont pas définis



Evaluation paresseuse

- ▶ L'évaluation des expressions se fait de gauche à droite
- ▶ Elle s'arrête dès que possible, on dit que **l'évaluation est paresseuse**.



```
i=0
```

```
a=5
```

```
b=(i!=0 and a/i>0)
```

```
# pas d'erreur car a/i n'est pas évalué.
```

```
#b vaut False
```



Booléen : On s'entraîne...

Soit (r,g,b) la couleur d'un pixel.

Si les 3 composantes de ce triplet sont égales à 0 le pixel est de couleur noire. Si les 3 composantes de ce triplet sont égales à 255 le pixel est de couleur blanche.

Quelle condition permet de vérifier qu'un pixel n'est ni noir ni blanc ?

$(r,g,b) \neq (0,0,0)$ **or** $(r,g,b) \neq (255,255,255)$?

$(r,g,b) \neq (0,0,0)$ **and** $(r,g,b) \neq (255,255,255)$?

$(r \neq 0$ **or** $g \neq 0$ **or** $b \neq 0)$ **and** $(r \neq 255$ **or** $g \neq 255$ **or** $b \neq 255)$?

$(r \neq 0$ **or** $r \neq 255)$ **and** $(g \neq 0$ **or** $g \neq 255)$ **and** $(b \neq 0$ **or** $b \neq 255)$?

Booléen : On s'entraîne...

Soit (r,g,b) la couleur d'un pixel.

Si les 3 composantes de ce triplet sont égales à 0 le pixel est de couleur noire. Si les 3 composantes de ce triplet sont égales à 255 le pixel est de couleur blanche.

Quelle condition permet de vérifier qu'un pixel n'est ni noir ni blanc ?

$(r,g,b) \neq (0,0,0)$ or $(r,g,b) \neq (255,255,255)$? ❌

- Le triplet $(0,0,0)$ -> True

$(r,g,b) \neq (0,0,0)$ and $(r,g,b) \neq (255,255,255)$? ✔️

$(r \neq 0$ or $g \neq 0$ or $b \neq 0)$ and $(r \neq 255$ or $g \neq 255$ or $b \neq 255)$? ✔️

$(r \neq 0$ or $r \neq 255)$ and $(g \neq 0$ or $g \neq 255)$ and $(b \neq 0$ or $b \neq 255)$?

- Le triplet $(0,0,0)$ -> True

❌

Notion de fonction

En mathématique une fonction $f : x \rightarrow 2x^2 + 1$
décrit la façon de calculer $f(x)$ à partir de la donnée x

En Python :

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

paramètre

Définition

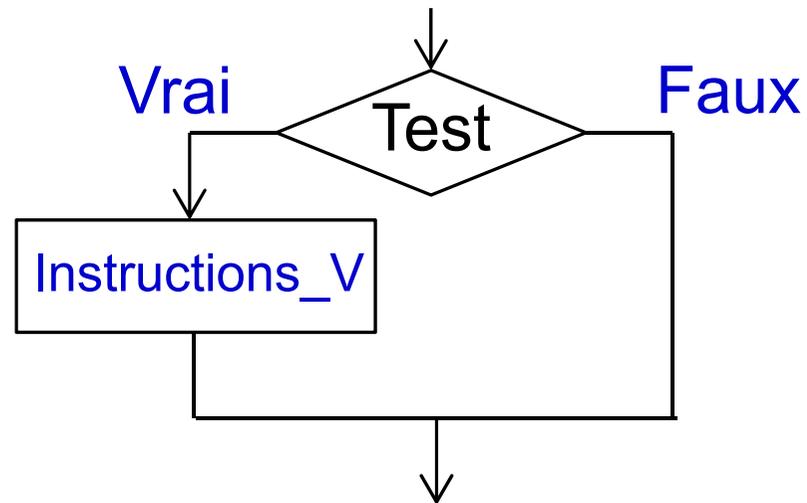
```
# exemples d'appel de la fonction  
y = 2 * f(2)  
print (f(5))
```

argument

Utilisations

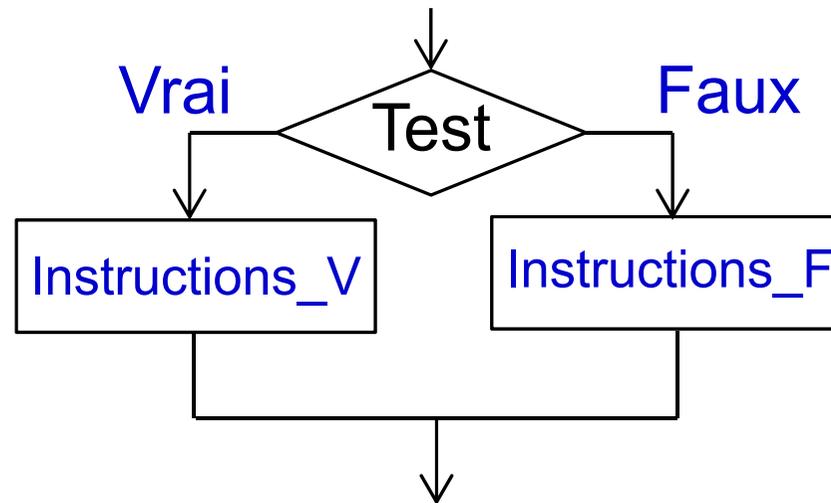
Instruction conditionnelle if...:

Instruction conditionnelle **si... alors...**



Instruction conditionnelle if...: else :...

Instruction conditionnelle **si... alors... sinon...**



Instruction conditionnelle

if...: else :...

Instruction conditionnelle **si... alors... sinon...**

```
if Test :  
↔ Instructions_V  
else :  
↔ Instructions_F
```

Instruction conditionnelle

if...: else :...

Instruction conditionnelle **si... alors... sinon...**

Exemple

```
i=10
```

```
x=6
```

```
if i > x :
```

```
↔ print ("test VRAI" )
```

```
↔ print (i, "est supérieur à", x)
```

```
else :
```

```
↔ print ("test FAUX")
```

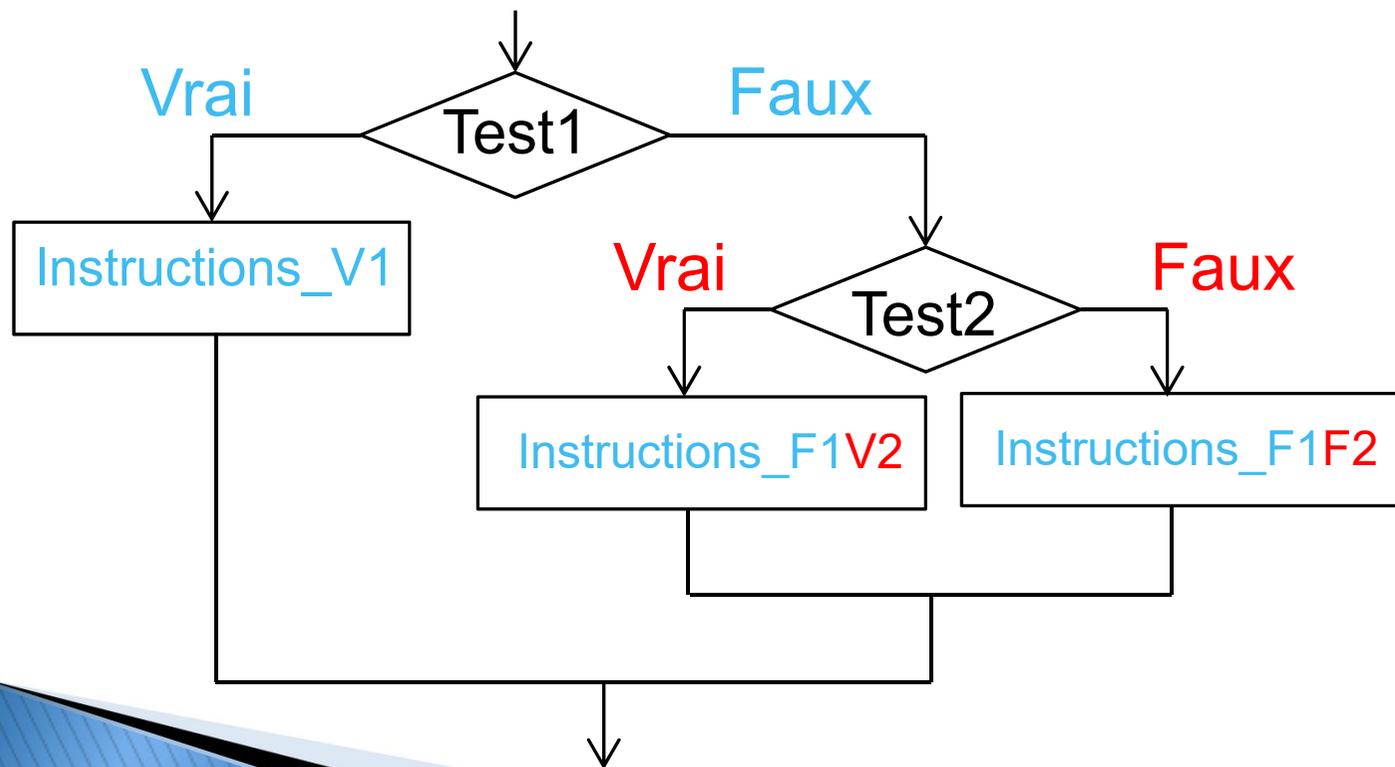
```
↔ print (i, "n'est pas supérieur à", x)
```

Attention à l'indentation !

Instruction conditionnelle

if... : elif... : else : ...

On peut « imbriquer » les **if... :else...**



Instruction conditionnelle

if... : elif... : else : ...

On peut « imbriquer » les `if... :else...`

```
if x<0 :  
↔ print ("negatif")  
else :  
↔ if x%2 ==0:  
↔↔ print ("pair" )  
↔ else :  
↔↔ print ("impair" )
```

```
if x<0 :  
↔ print( "negatif" )  
elif x%2 ==0:  
↔ print( "pair " )  
else :  
↔ print( "impair" )
```

Notion de fonction

```
def estLeProduitN_v1 (n, x, y) :  
    if n==x*y:  
        return True  
    else :  
        return False
```

```
def estLeProduitN_v2 (n, x, y) :  
    if n==x*y:  
        return True  
    return False
```

```
def estLeProduitN_v3 (n, x, y) :  
    return n==x*y
```

Listes d'éléments

On peut manipuler des listes d'éléments : $[x_0, x_1, \dots, x_{n-1}]$

Exemples :

```
[3, 7, 4, 1]  
['d', 'f', 'e']
```

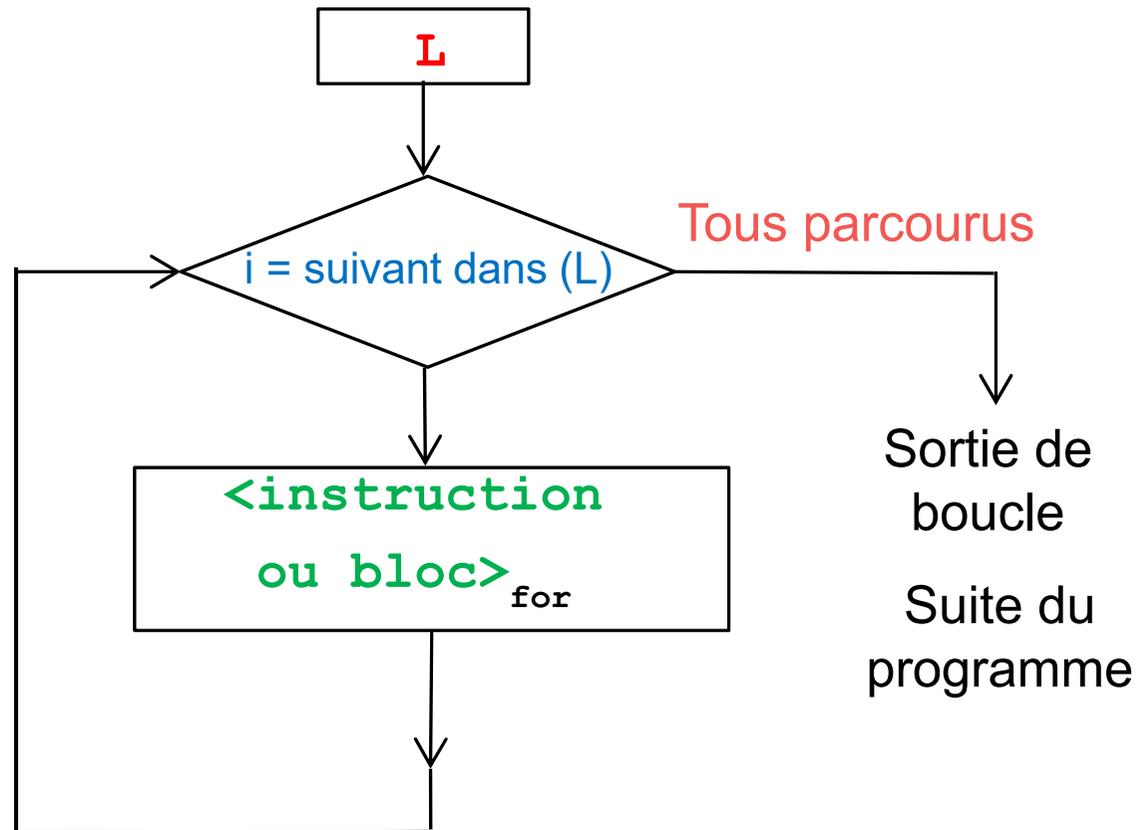
```
6 in [3, 7, 4, 1] => False
```

```
4 in [3, 7, 4, 1] => True
```

La fonction **len** retourne la taille de la liste.

```
len([3, 7, 4, 1]) => 4
```

Répétition : **for ... in Liste :**



Répétition : **for ... in Liste :**

Répétition : **pour... parcourant la liste ...**

La boucle **for** permet de parcourir les éléments d'une liste

parcourir et afficher les entiers de 0 à 9

```
L=[0,1,2,3,4,5,6,7,8,9]
```

```
for i in L :
```

```
    <=> print (i)
```

On peut imbriquer les boucles :

```
L=[0,1,2,3,4,5,6,7,8,9]
```

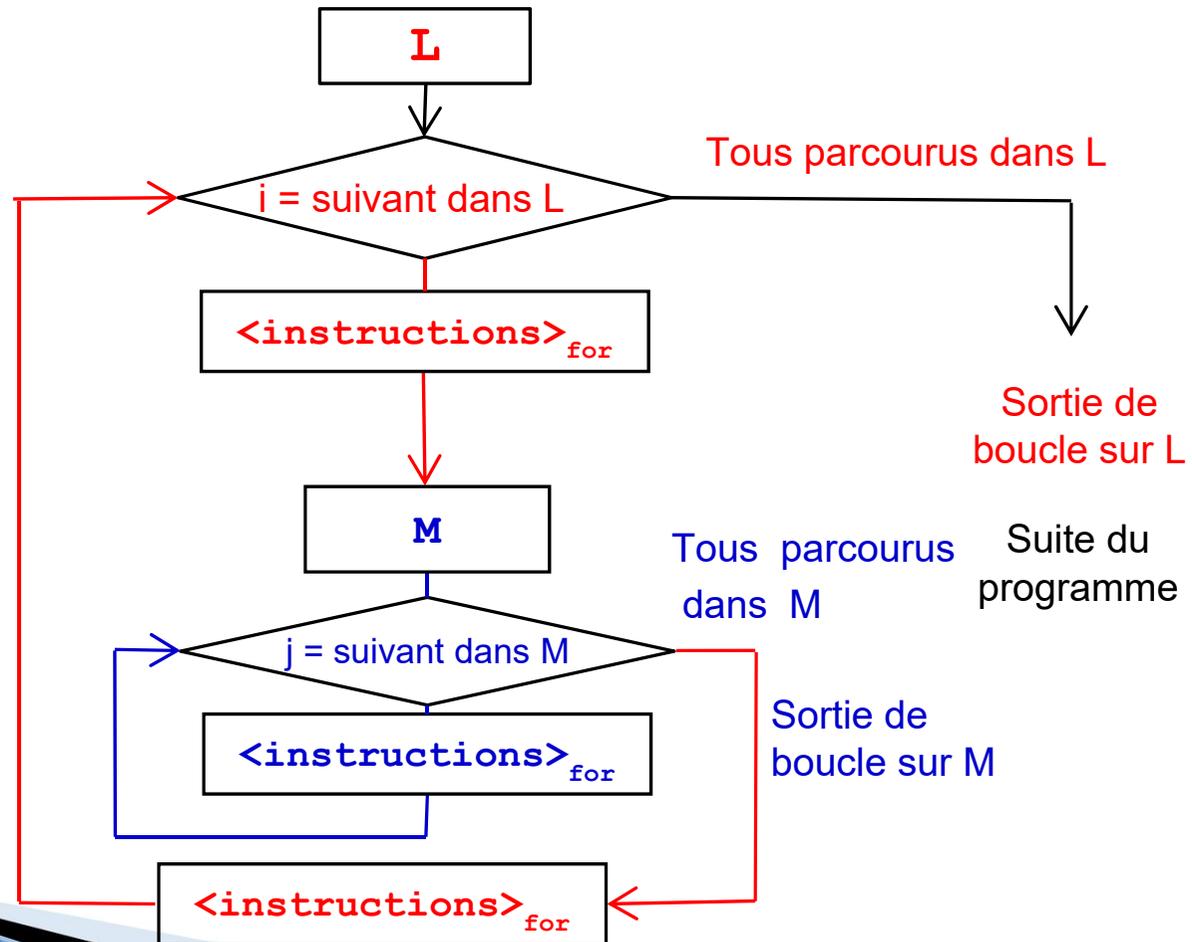
```
M=[0,1,2]
```

```
for i in L:
```

```
    <=> for j in M:
```

```
        <=><=> print(i,j)
```

Répétition : **for ... in Liste :**



Répétition : **for ... in Liste :**

Répétition : **pour... parcourant la liste ...**

On peut imbriquer les boucles :

```
L=[0,1,2,3,4,5,6,7,8,9]
```

```
M=[0,1,2]
```

```
for i in L:
```

```
    for j in M:
```

```
        print(i,j)
```

Quel sera l'affichage ?

```
(0,0) (0,1),(0,2),(1,0),(1,1),(1,2),..., (9,0),(9,1),(9,2)
```

Listes d'éléments

La fonction **range** permet de construire des listes d'éléments consécutifs

liste des entiers $\in [a, b[$ par pas de c

range(a, b, c) :



Debut Fin Pas

Remarque: **range(a, b) \Leftrightarrow range(a, b, 1) #c=1**
range(b) \Leftrightarrow range(0, b, 1) #a=0, c=1

Exemple :

`list (range(1, 9, 1)) => [1, 2, 3, 4, 5, 6, 7, 8]`

`list (range(1, 9, 2)) => [1, 3, 5, 7]`

Listes d'éléments

La fonction **range** permet de construire des listes d'éléments consécutifs

range (2, 14, 3)

-> [2, 5, 8, 11]

range (2, 14)

-> [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

range (14)

-> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

range (14, 3)

-> []

range (14, 3, -1)

-> [14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4]

Répétition : **for ... in ...** :

Répétition : **pour... parcourant la liste ...**

La boucle **for** permet de parcourir les éléments d'une liste

parcourir et afficher les entiers de 0 à 9

```
L=[0,1,2,3,4,5,6,7,8,9]
```

```
for i in L :
```

```
    print (i)
```

On peut imbriquer les boucles :

```
L=[0,1,2,3,4,5,6,7,8,9]
```

```
M=[0,1,2]
```

```
for i in L:
```

```
    for j in M:
```

```
        print(i,j)
```

Répétition : **for ... in ...** :

Répétition : **pour... parcourant la liste ...**

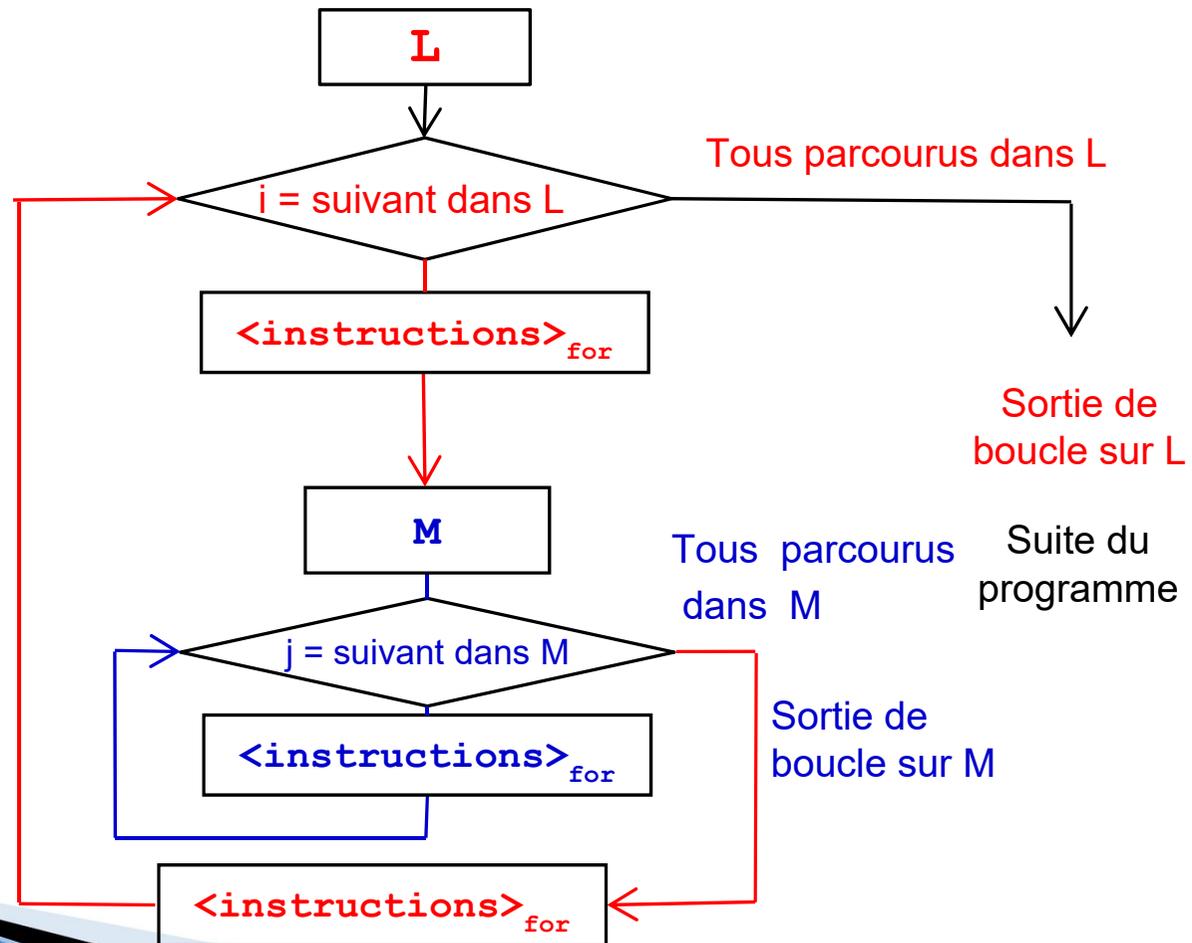
La boucle **for** permet de parcourir les éléments d'une liste

```
# parcourir et afficher les entiers de 0 à 9
for i in range(10) :
    print (i)
```

On peut imbriquer les boucles :

```
for i in range(10) :
    for j in range(3) :
        print(i,j)
```

Répétition : **for ... in Liste :**



Répétition : **for ... in ...** : Exemple

Proposer un exemple de boucles imbriquées que vous utilisez dans la vie courante :

pour toutes les heures de la journée :
pour toutes les minutes de chaque heure :
pour toutes les secondes de chaque minute : ...

Ou plus complexe :

pour toutes les années d'une vie :
pour toutes les semaines de l'année :
pour tous les jours de la semaine :
pour toutes les heures de la journée :
pour toutes les minutes de chaque heure :
pour toutes les secondes de chaque minute :

penser !

Sortie d'une boucle **for** : exemple 1

Proposer un exemple de boucles imbriquées que vous utilisez dans la vie courante :

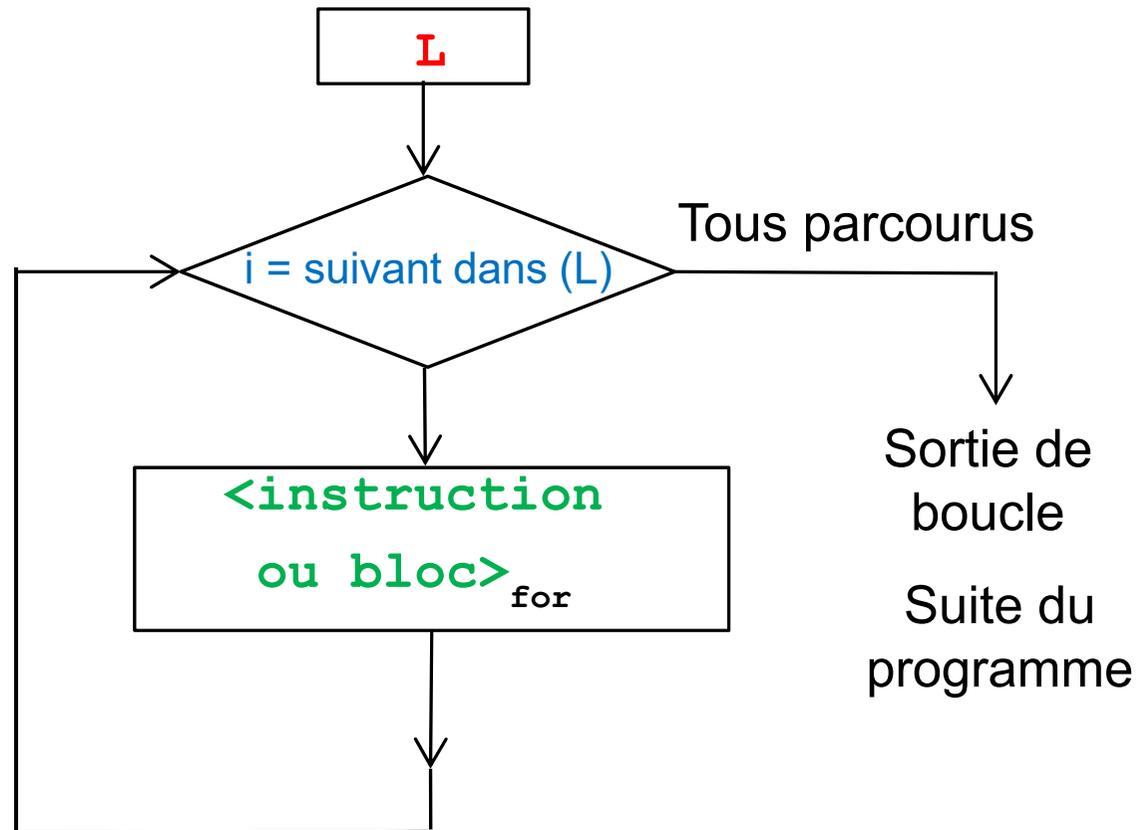
pour toutes les heures de la journée :
pour toutes les minutes de chaque heure :
pour toutes les secondes de chaque minute : ...

Ou plus complexe :

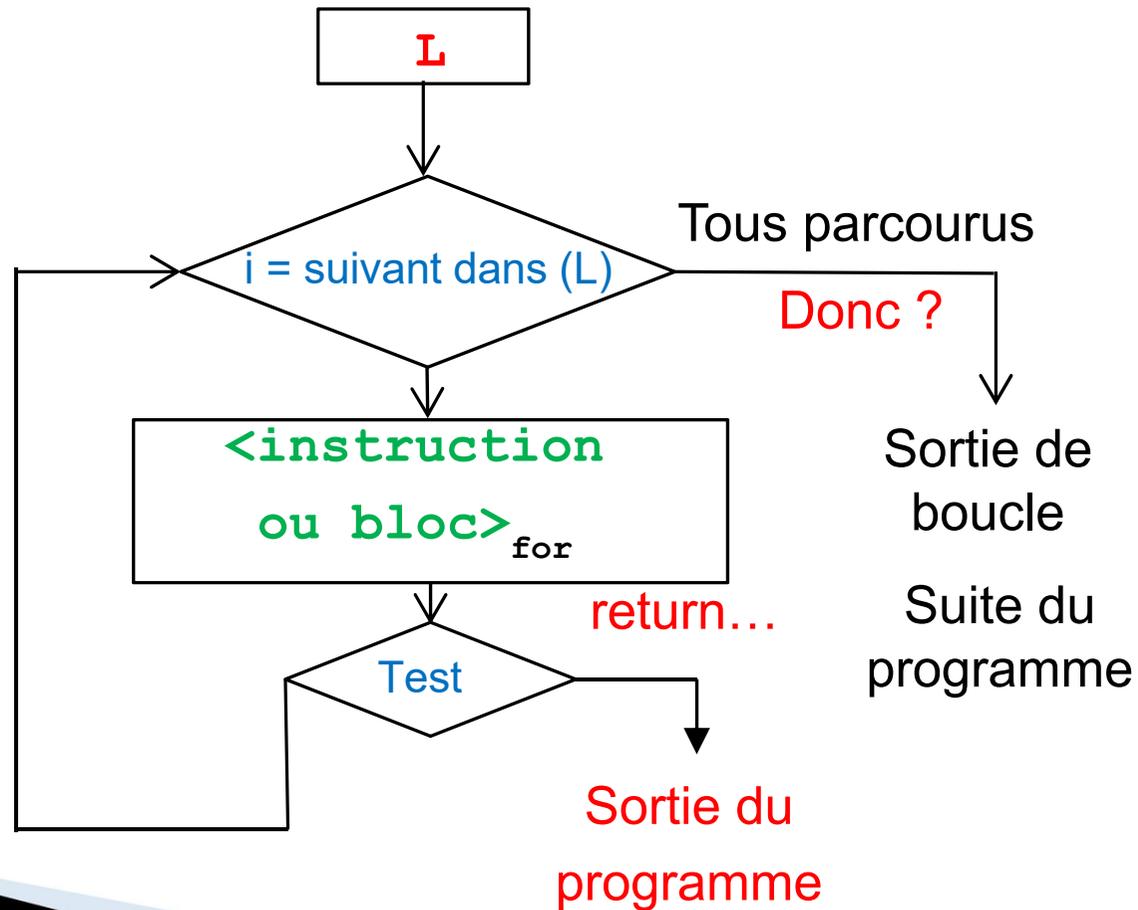
pour toutes les années d'une vie :
pour toutes les semaines de l'année :
pour tous les jours de la semaine :
pour toutes les heures de la journée :
pour toutes les minutes de chaque heure :
pour toutes les secondes de chaque minute :

penser !

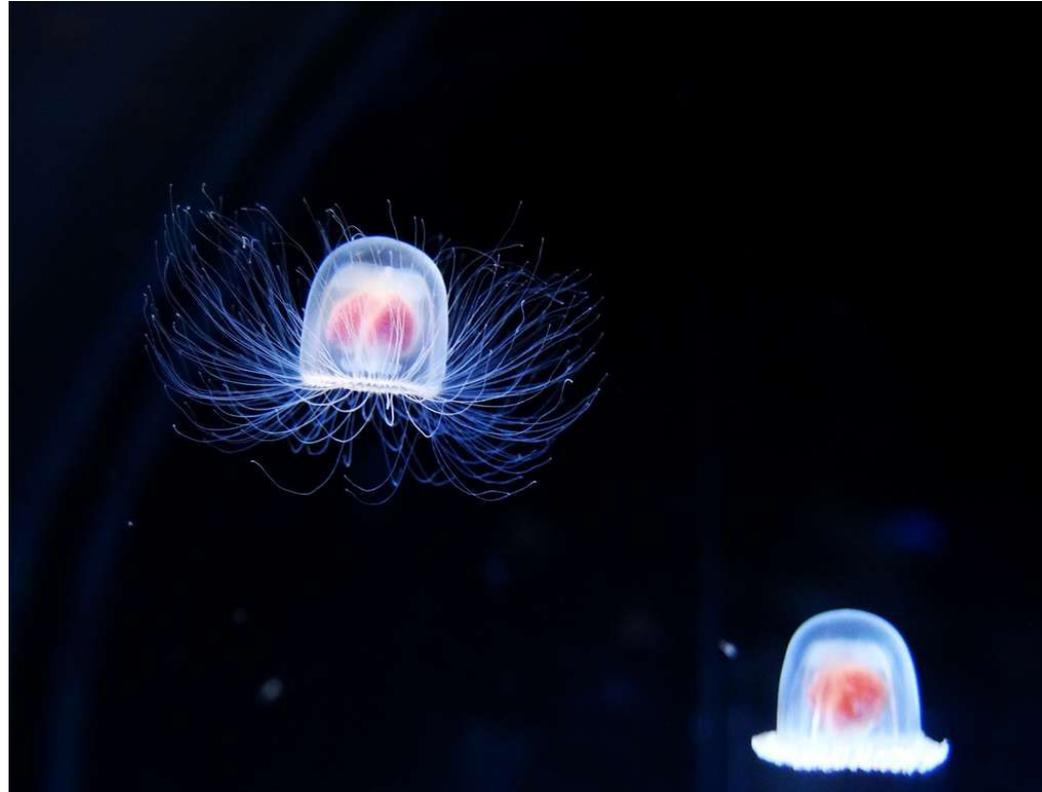
Répétition : **for ... in Liste :**



Répétition : **for ... in Liste : interruption**



Sortie d'une boucle **for** : exemple 1



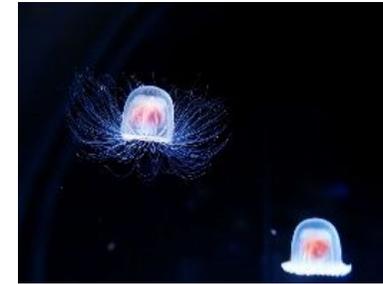
Une méduse urritopsis nutricula

Sortie d'une boucle **for** : exemple 1

penser = 0

```
for a in range (n): #pour toutes les années d'une vie
| for w in range (1,53): #pour toutes les semaines de l'année
| | for j in range (1,8): #pour tous les jours de la semaine
| | | for h in range (0,24): #pour toutes les heures de la journée
| | | | for m in range (0,60): #pour toutes les minutes de l'heure
| | | | | for s in range (0,60): #pour toutes les secondes d'une minute
| | | | | penser =penser+1
| | | | # fin de la boucle sur s
| | | # fin de la boucle sur m
| | # fin de la boucle sur h
| # fin de la boucle sur j
# fin de la boucle sur w
# fin de la boucle sur a
```

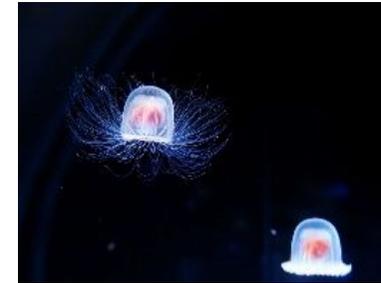
Sortie d'une boucle **for** : exemple 1



penser = 0

```
for a in range (n): #pour toutes les années d'une vie
  for w in range (1,53): #pour toutes les semaines de l'année
    for j in range (1,8): #pour tous les jours de la semaine
      for h in range (0,24): #pour toutes les heures de la journée
        for m in range (0,60): #pour toutes les minutes de l'heure
          for s in range (0,60): #pour toutes les secondes d'une minute
            penser =penser+1
          # fin de la boucle sur s
        # fin de la boucle sur m
      # fin de la boucle sur h
    # fin de la boucle sur j
  # fin de la boucle sur w
# fin de la boucle sur a
```

Sortie d'une boucle **for** : exemple 1



penser = 0

```
for a in range (n): #pour toutes les années d'une vie
  for w in range (1,53): #pour toutes les semaines de l'année
    for j in range (1,8): #pour tous les jours de la semaine
      for h in range (0,24): #pour toutes les heures de la journée
        for m in range (0,60): #pour toutes les minutes de l'heure
          for s in range (0,60): #pour toutes les secondes d'une minute
            penser =penser+1
          # fin de la boucle sur s
        # fin de la boucle sur m
      # fin de la boucle sur h
    # fin de la boucle sur j
  # fin de la boucle sur w
# fin de la boucle sur a
```

Comment utiliser ces informations ?

```
print(« je suis une méduse urritopsis nutricula »)
```

L'éphémère vit entre 1 et 24h :
disons 12h !



penser =0

ephemere=True

for a in range (n): #pour toutes les années d'une vie

for w in range (1,53): #pour toutes les semaines de l'année

for j in range (1,8): #pour tous les jours de la semaine

for h in range (0,24): #pour toutes les heures de la journée

if ephemere and h>=12:

return (penser)

for m in range (0,60): #pour toutes les minutes de l'heure

for s in range (0,60): #pour toutes les secondes par minute

penser = penser+1

Sans ajouter de test, où placer l'instruction

return (False) indiquant de façon sure que l'animal n'est pas un éphémère?

L'éphémère vit entre 1 et 24h :
disons 12h !



penser =0

for a in range (n): #pour toutes les années d'une vie

for w in range (1,53): #pour toutes les semaines de l'année

for j in range (1,8): #pour tous les jours de la semaine

for h in range (0,24): #pour toutes les heures de la journée

for m in range (0,60): #pour toutes les minutes de l'heure

for s in range (0,60): #pour toutes les secondes par minute

penser = penser+1

return (False)

L'éphémère vit entre 1 et 24h



penser =0

for a in range (n): #pour toutes les années d'une vie

for w in range (1,53): #pour toutes les semaines de l'année

for j in range (1,8): #pour tous les jours de la semaine

for h in range (0,24): #pour toutes les heures de la journée

for m in range (0,60): #pour toutes les minutes de l'heure

for s in range (0,60): #pour toutes les secondes par minute

penser = penser+1

return (False)

#sortie du programme

Quand l'algorithme atteint ce point c'est que plus de 24h ce sont écoulées.

La libellule vit environ 18 semaines



```
penser= 0
libellule=True
for a in range (n): #pour toutes les années d'une vie
    for w in range (1,53): #pour toutes les semaines de l'année
        if libellule and w>18:
            return (penser)
        for j in range (1,8): #pour tous les jours de la semaine
            for h in range (0,24): #pour toutes les heures de la journée
                for m in range (0,60): #pour toutes les minutes de l'heure
                    for s in range (0,60): #pour toutes les secondes par minute
                        penser= penser +1
```

Sans ajouter de test, où placer l'instruction
`return (False)` indiquant de façon sûre que
l'animal n'est pas une libellule

La libellule vit environ 18 semaines



```
penser= 0
```

```
for a in range (n): #pour toutes les années d'une vie
```

```
  for w in range (1,53): #pour toutes les semaines de l'année
```

```
    for j in range (1,8): #pour tous les jours de la semaine
```

```
      for h in range (0,24): #pour toutes les heures de la journée
```

```
        for m in range (0,60): #pour toutes les minutes de l'heure
```

```
          for s in range (0,60): #pour toutes les secondes par minute
```

```
            penser=penser+1
```

```
  return ( False)
```

Pour continuer :



Le hamster vit entre 2 et 4 ans



La plus vieille tortue géante a 184 ans

Sortie d'une boucle **for** : Exemple

```
def tousPositif (L):  
    for u in (L):  
        if u<0:  
            return False
```

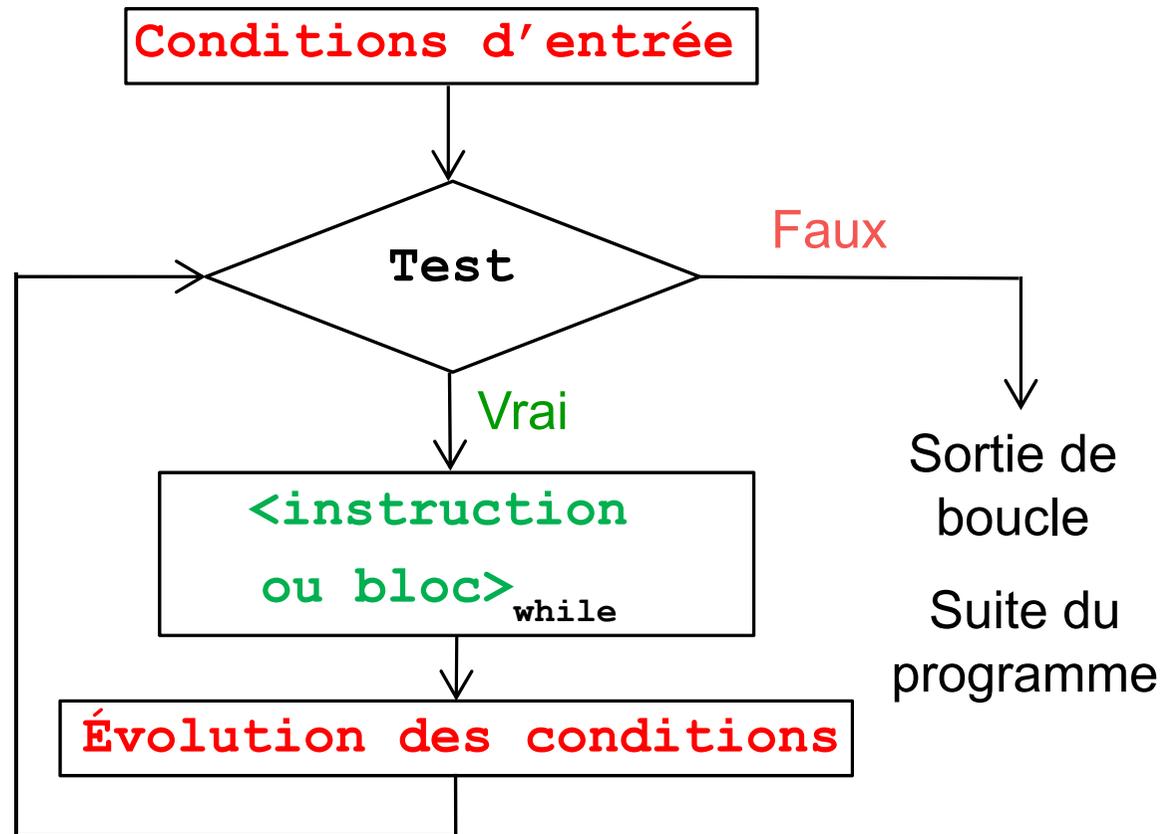
.....

Où écrit-on « **return True** » ?

Comment s'en servir ?

```
def tousPositif (L):  
    for u in (L):  
        if u<0:  
            return False  
    return True
```

Répétition : **while... :**



Répétition : **while... :**

Répétition : **tant que ... faire...**

```
i=10
```

```
while i > 0 :
```

```
  ↔ print (i)
```

```
  ↔ i = i - 1
```

```
10 9 8 7 6 5 4 3 2 1
```

Ou :

```
while i > 0 :
```

```
  ↔ i = i - 1
```

```
  ↔ print (i)
```

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

Attention à l'ordre des instructions !

Boucle `while` : Exemple 1

Afficher les puissance de k ($k > 1$) inférieures à `nMax`

```
def AfficherPuissances(k, nMax) :  
    i = 1 #initialization  
    while (i <= nMax) :  
        print(i)  
        i = i * k
```

Peut-on écrire cet algorithme avec une boucle `for` ?

Boucle `while` : Exemple 1

```
def afficherPuissancesAvecFor (k, nMax) :  
    p=1  
    for i in range (???) :  
        if (p<=???) :  
            print (p)  
            p=p*k  
        else :  
            return (None)
```

Peut-on écrire cet algorithme avec une boucle `for` ?

Pourquoi la boucle `while` est à privilégier ?

Boucle `while` : Exemple 1

```
def afficherPuissancesAvecFor (k, nMax) :  
    p=1  
    for i in range (nMax) :  
        if (p<=nMax) :  
            print(p)  
            p=p*k  
        else :  
            return (None)
```

Ecriture non
« naturelle »

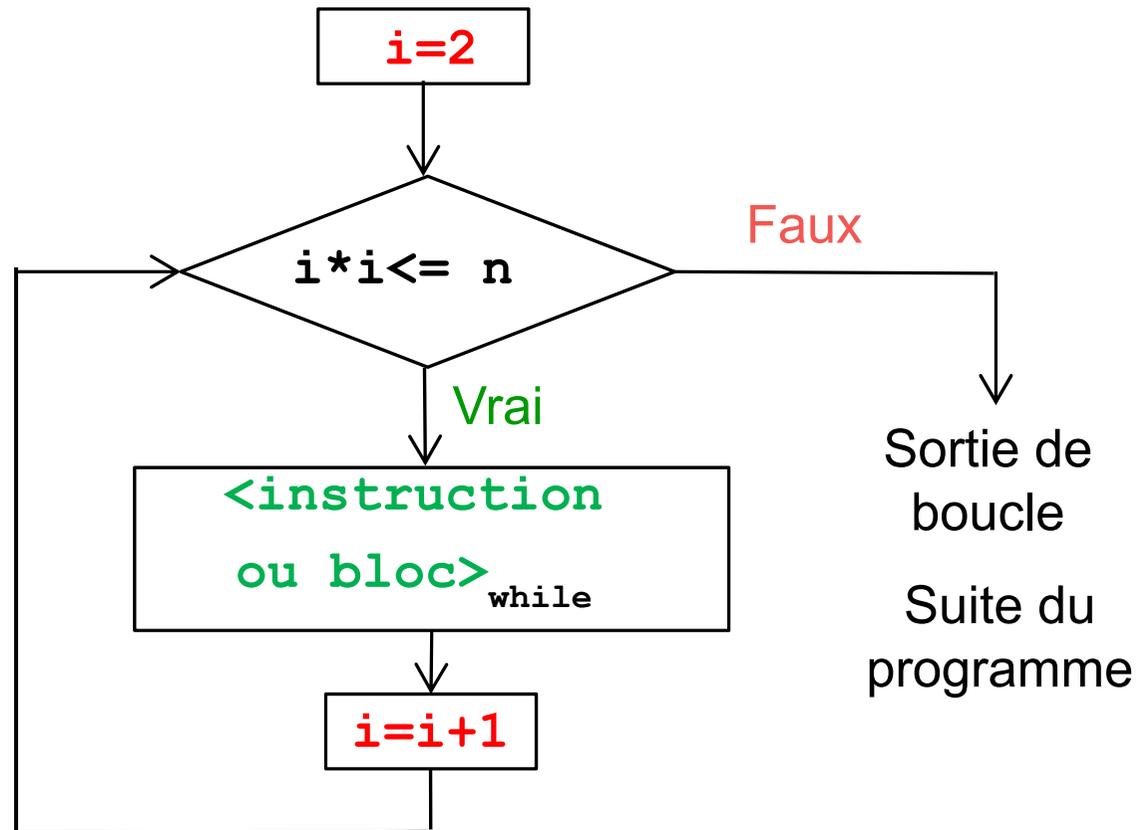
Peut-on écrire cet algorithme avec une boucle `for` ?

Pourquoi la boucle `while` est à privilégier ?

Boucle `while` : Exemple 2

```
def estPremier(n):  
    i = 2  
    while (i*i <= n):  
        if (n % i == 0):  
            return False  
        i=i+1  
    return True
```

Boucle while : Exemple 2



Boucle `while` : Exemple 3

```
def mystere (n) :  
    k = n  
    while (k%2==0) :  
        k=k//2  
    return k==1
```

Que vaut `mystere(240)` ?

Que vaut `mystere(256)` ?

Quelques pièges à éviter !

- Affichage ou sortie (**print** ou **return**)
- Position du **return** (dans la boucle ou après la boucle)

```
def f(x):  
    for i in range(x):  
        print (i)
```

```
def g(x):  
    for i in range(x):  
        return (i)
```

```
def h(x):  
    for i in range(x):  
        print (i)  
    return (i)
```

Quel sera le résultat de :
f, g et h pour x=3 ?