Conditions et boucles

Informatique pour tous

Semaine précédente

« = » est l'**affectation** en Python, pour modifier une variable :

`a = b` met la valeur de `b` dans `a`.

Exercice ICNA

Question 4 On considère le script Python suivant :

```
a=5
b=2
c=a
a=b
b=a
a=a+b
c=b+a
```

Quelle est la valeur de la variable c après l'exécution de ce script ?

A) 2

B) 4

C) 6

D) 8

Semaine précédente

Types en Python :

- **Nombre entier** (`int`)
- **Flottant** (`float`) : nombre à virgule : 2.718 -4.7564
- **Booléen ou condition** (`bool`) : `True` (vrai) ou `False` (faux)
  On peut utiliser le « et logique » **and** et le « ou logique » (inclusif) **or**.
- **Chaine de caractères** (`str`), entre guillemets :
  `"Ceci est une phrase"`
- ***n*-uplets** (`tuple`) : `(1, -2)` `(1, "blabla", False)`

« ou logique » / « et logique »

```
In [1]: True or True
Out[1]: True

In [2]: True or False
Out[2]: True

In [3]: False or True
Out[3]: True

In [4]: False or False
Out[4]: False
```

```
In [5]: True and True
Out[5]: True

In [6]: True and False
Out[6]: False

In [7]: False and True
Out[7]: False

In [8]: False and False
Out[8]: False
```

Semaine précédente

Opérations sur les nombres :

- `a * b` : multiplication de `a` et `b`.
- `a / b` : division de `a` par `b`.
- `a ** b` : `a` puissance `b`.
- `a // b` : quotient **entier** de la **division euclidienne** de `a` par `b`.
- `a % b` : reste de la division euclidienne de `a` par `b`.

On peut utiliser $==$, $!=$, $<$, $>$, $<=$, $>=$ pour comparer des nombres.

Divisibilité

On suppose définies deux variables `a` et `b`.

Comment savoir si `a` divise `b` ?

Divisibilité

On suppose définies deux variables `a` et `b`.

Comment savoir si `a` divise `b` ?

`a` divise `b` $\Longleftrightarrow$ le reste de la division de `b` par `a` est nul

Divisibilité

On suppose définies deux variables `a` et `b`.

Comment savoir si `a` divise `b` ?

`a` divise `b` $\iff$ le reste de la division de `b` par `a` est nul

$\iff$ `b % a == 0`

Chiffres d'un nombre

Quelle opération utiliser pour récupérer le chiffre des unités d'un entier n ?

Chiffres d'un nombre

Quelle opération utiliser pour récupérer le chiffre des unités d'un entier n ?

Si c est le chiffre des unités et d le nombre de dizaines, alors :

$$n = d \times 10 + c$$

Chiffres d'un nombre

Quelle opération utiliser pour récupérer le chiffre des unités d'un entier n ?

Si c est le chiffre des unités et d le nombre de dizaines, alors :

$$n = d \times 10 + c$$

Donc :

1. le chiffre des unités est donné par `n % 10`
2. le reste des chiffres est donné par `n // 10`

Instruction

En Python, une ligne = une instruction.

Il est nécessaire de sauter une ligne entre chaque instruction.

Blocs d'instructions

Un **bloc d'instructions** est une suite d'instructions avec la même **indentation** (décalage par rapport à la gauche).

```
    Ceci est un
    bloc d'instructions
        Un autre bloc
        d'instructions
```

Condition if

```
# instructions avant le if

if condition:
    # bloc
    # d'instructions

# instructions après le if
```

Exécute le bloc d'instructions qui suit **si** la condition est vraie (`True`).

`condition` doit être un **booléen** :
il peut contenir `==`, `!=`, `<`, `>`, `and`, `or`, `not`, des calculs...

Condition if

Exemple :

```
if p % 2 == 0:
    print("Ce message est affiché...")
    print("...seulement si p est pair")
print("Ce message est toujours affiché")
```

(print affiche une chaîne de caractères)

Condition if

On peut aussi utiliser une variable booléenne dans un `if` :

```
b = (p % 2 == 0)
if b:
    print("Ce message est affiché...")
    print("... seulement si p est pair")
print("Ce message est toujours affiché")
```

Condition if

Comment calculer le maximum de deux variables x et y ?

Condition if

Comment calculer le maximum de deux variables x et y ?

```
maxi = x
if y > maxi:
    maxi = y
```

Maximum de 3 nombres

Comment calculer le maximum de 3 variables `x`, `y`, `z` ?

Maximum de 3 nombres

Comment calculer le maximum de 3 variables `x`, `y`, `z` ?

```
maxi = x
if y > maxi:
    maxi = y
if z > maxi:
    maxi = z
```

Erreurs fréquentes

```
if p % 2 == 0:
    print("Ce message est affiché...")
     print("... seulement si p est pair")
print("Ce message est toujours affiché")
```

Erreurs fréquentes

```
if p % 2 == 0:
    print("Ce message est affiché...")
     print("... seulement si p est pair")
print("Ce message est toujours affiché")
```

```
  File "<tmp 2>+17", line 4
    print("... seulement si p est pair")
    ^
IndentationError: unexpected indent
```

Erreurs fréquentes

```
if p % 2 == 0:
    print("Ce message est affiché...")
     print("... seulement si p est pair")
print("Ce message est toujours affiché")
```

```
  File "<tmp 2>+17", line 4
    print("... seulement si p est pair")
    ^
IndentationError: unexpected indent
```

Attention à indenter de la même façon toutes les instructions du bloc

Erreurs fréquentes

```
if p % 2 == 0
    print("Ce message est affiché...")
    print("... seulement si p est pair")
print("Ce message est toujours affiché")
```

Erreurs fréquentes

```
if p % 2 == 0
    print("Ce message est affiché...")
    print("... seulement si p est pair")
print("Ce message est toujours affiché")
```

```
  File "<tmp 2>+17", line 2
    if p % 2 == 0
                ^
SyntaxError: invalid syntax
```

Erreurs fréquentes

```
if p % 2 == 0
    print("Ce message est affiché...")
    print("... seulement si p est pair")
print("Ce message est toujours affiché")
```

```
  File "<tmp 2>+17", line 2
    if p % 2 == 0
                ^
SyntaxError: invalid syntax
```

Attention à ne pas oublier « : »

Erreurs fréquentes

```
if p % 2 = 0:
    print("Ce message est affiché...")
    print("... seulement si p est pair")
print("Ce message est toujours affiché")
```

Erreurs fréquentes

```
if p % 2 = 0:
    print("Ce message est affiché...")
    print("... seulement si p est pair")
print("Ce message est toujours affiché")
```

```
    if p % 2 = 0:
               ^
SyntaxError: invalid syntax
```

Erreurs fréquentes

```
if p % 2 = 0:
    print("Ce message est affiché...")
    print("... seulement si p est pair")
print("Ce message est toujours affiché")
```

```
    if p % 2 = 0:
               ^
SyntaxError: invalid syntax
```

`a = b` est une **affectation** (modifie a)

`a == b` est un **booléen** (True ssi les valeurs de `a` et `b` sont égales)

Condition if

On peut aussi utiliser `else` (= sinon) :

```
if p % 2 == 0:
    print("p est pair")
else:
    print("p est impair")
```

Condition if

On peut aussi utiliser `else` (= sinon) :

```
if p % 2 == 0:
    print("p est pair")
else:
    print("p est impair")
```

Un `else` est forcément précédé d'un `if` mais un `if` n'est pas forcément suivi d'un `else`...

Condition if

On peut aussi utiliser `elif` pour enchaîner plusieurs if à la suite :

```
delta = b**2 - 4*a*c
if delta == 0:
    print("1 racine réelle")
elif delta < 0:
    print("pas de racine réelle")
else:
    print("2 racines réelles")
```

Condition if

On peut aussi utiliser `elif` pour enchaîner plusieurs if à la suite :

```
delta = b**2 - 4*a*c
if delta == 0:
    print("1 racine réelle")
elif delta < 0:
    print("pas de racine réelle")
else:
    print("2 racines réelles")
```

⚠ ce code marche si `a`, `b`, `c` sont entiers, car **les calculs sur les entiers sont exacts**.

Condition if

Cependant, il peut y avoir des **erreurs d'arrondis sur les flottants** :

```
In [11]: a, b, c = 1., 0.2, 0.01

In [12]: b**2 - 4*a*c
Out[12]: 6.938893903907228e-18

In [13]: b**2 - 4*a*c == 0.
Out[13]: False
```

Alors que $0.2^2 - 4 \times 0.01 = 0$ en mathématiques !

On expliquera plus tard pourquoi le calcul sur les entiers est exact mais pas le calcul sur les flottants.

Condition if

Il est possible d'imbriquer un if à l'intérieur d'un autre :

```
if delta == 0:
    print("1 racine réelle")
else:
    if delta < 0:
        print("pas de racine réelle")
    else:
        print("2 racines réelles")
```

Boucle while

Pour répéter un bloc d'instructions **tant que** `condition` est `True` :

```
# exécuté avant le while

while condition:
    # bloc
    # d'instructions

# exécuté après le while
```

Normalement, le bloc d'instructions peut modifier `condition`.
Il est possible que `condition` soit faux dès le début, auquel cas le bloc n'est pas exécuté du tout.

Boucle while

Exemple : afficher le plus petit entier n tel que $2^n \geq x$.

Boucle while

Exemple : afficher le plus petit entier n tel que $2^n \geq x$.

```
n = 0
while 2**n < 311:
    n = n + 1
print(n)
```

Boucle while

Exemple : afficher les entiers de 0 à 20.

Boucle while

Exemple : afficher les entiers de 0 à 20.

```
i = 0
while i <= 20:
    print(i)
    i = i + 1
```

Boucle while

Exemple : afficher les entiers de 0 à 20.

```
i = 0
while i <= 20:
    print(i)
    i = i + 1
```

Il est très courant de vouloir augmenter (on dit aussi incrémenter) une variable dans une boucle. Il existe une boucle spécifique pour ça : la boucle **for**.

Boucle for

Pour répéter un bloc d'instructions n fois :

```
for i in range(n):
    # bloc d'instructions
```

Boucle for

Pour répéter un bloc d'instructions n fois :

```
for i in range(n):
    # bloc d'instructions
```

`i` est une nouvelle variable qui prend la valeur 0 la 1ère fois que le bloc est exécutée, 1 la 2ème fois, ..., jusqu'à $n-1$ pour la dernière.

On peut utiliser un nom de variable quelconque pour un `for`, mais souvent on utilise `i`, `j`, `k`...

Boucle for

Pour répéter un bloc d'instructions n fois :

```
for i in range(n):
    # bloc d'instructions
```

`i` est une nouvelle variable qui prend la valeur 0 la 1ère fois que le bloc est exécutée, 1 la 2ème fois, ..., jusqu'à $n-1$ pour la dernière.

On peut utiliser un nom de variable quelconque pour un `for`, mais souvent on utilise `i`, `j`, `k`...

⚠ En Python, les indices commencent à **0**, et la borne supérieure est souvent **exclue** !
⚠ Il ne faut jamais modifier la variable d'un `for` !

Boucle for

Exemple : afficher les entiers de 0 à 20 avec une boucle `for`.

Boucle for

Exemple : afficher les entiers de 0 à 20 avec une boucle `for`.

```
for i in range(21):
    print(i)
```

Boucle for

Exemple : afficher 17 fois « je ne dois pas confondre = et == ».

Boucle for

Exemple : afficher 17 fois « je ne dois pas confondre = et == ».

```
for j in range(17):
    print("je ne dois pas confondre = et ==")
```

Remarque : ici on n'utilise pas la variable j.

Calcul de somme

Une somme peut facilement se calculer avec un `for`, par exemple la somme des 30 premiers carrés $\sum_{k=0}^{29} k^2$:

Calcul de somme

Une somme peut facilement se calculer avec un `for`, par exemple la somme des 30 premiers carrés $\sum_{k=0}^{29} k^2$:

```
somme = 0
for k in range(30):
    somme = somme + k**2
```

Calcul de suite récurrente

Soit (v_n) la suite définie par :

$$\begin{cases} v_0 = 5 \\ v_{n+1} = 2v_n + 3 \end{cases}$$

Pour afficher v_{10} :

Calcul de suite récurrente

Soit (v_n) la suite définie par :

$$\begin{cases} v_0 = 5 \\ v_{n+1} = 2v_n + 3 \end{cases}$$

Pour afficher v_{10} :

```
vn = 5
for i in range(10):
    vn = 2*vn + 3
print(vn)
```

`vn` prend successivement les valeurs de v_0, v_1, ... v_{10}.
Remarque : on n'utilise pas la variable du `for`.

Calcul de suite récurrente d'ordre 2

Exercice : calculer v_{10}, où (v_n) est définie par :

$$\begin{cases} v_0 = 1 \\ v_1 = 1 \\ v_{n+2} = v_{n+1} + v_n \end{cases}$$

Calcul de suite récurrente d'ordre 2

Exercice : calculer v_{10}, où (v_n) est définie par :

$$\begin{cases} v_0 = 1 \\ v_1 = 1 \\ v_{n+2} = v_{n+1} + v_n \end{cases}$$

Si v_{n+2} est défini en fonction de v_n et v_{n+1}, on peut utiliser deux variables : une pour v_n et une pour v_{n+1}.

```
vn = 1
vn_plus_1 = 1
for i in range(10):
    vn_plus_1, vn = vn + vn_plus_1, vn_plus_1
print(vn)
```

Boucle for

Il est aussi possible de commencer à partir d'un entier *m* **inclus** jusqu'à *n* **exclus** :

```
for i in range(m, n):
    # bloc d'instructions
```

Boucle for

Il est aussi possible de commencer à partir d'un entier *m* **inclus** jusqu'à *n* **exclus** :

```
for i in range(m, n):
    # bloc d'instructions
```

Question

Combien de fois le bloc d'instructions s'exécute dans ce cas ?

Boucle for

Il est aussi possible de commencer à partir d'un entier m **inclus** jusqu'à n **exclus** :

```
for i in range(m, n):
    # bloc d'instructions
```

Question

Combien de fois le bloc d'instructions s'exécute dans ce cas ?

Réponse : $n - m$

Boucle for

Boucle qui affiche tous les nombres divisant un entier n :

Boucle for

Boucle qui affiche tous les nombres divisant un entier n :

```
for d in range(2, n + 1):
    if n % d == 0:
        print(d)
```

Boucle for

Il est possible de parcourir les entiers de m à n en sautant de p en p, avec $p \in \mathbb{Z}$:

```
for i in range(m, n, p):
    # bloc d'instructions
```

Boucle for

Il est possible de parcourir les entiers de m à n en sautant de p en p, avec $p \in \mathbb{Z}$:

```
for i in range(m, n, p):
    # bloc d'instructions
```

Le bloc d'instruction est exécuté pour $i = m$, $i = m + p$, $i = m + 2p$... tant que $i < n$.

Boucle for

Question

Qu'affiche la boucle suivante ?

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

Boucle for

Question

Qu'affiche la boucle suivante ?

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

Réponse : 1, 3, 5, 7, 9 (entiers impairs de 1 à 9)

Boucle for

Afficher les 10 premiers multiples de 7 :

Boucle for

Afficher les 10 premiers multiples de 7 :

```
for m in range(0, 64, 7):
    print(m)
```

Boucle for

Autre possibilité, avec un `if` à l'intérieur du `for` :

Boucle for

Autre possibilité, avec un `if` à l'intérieur du `for` :

```
for m in range(0, 64):
    if m % 7 == 0:
        print(m)
```

Boucle for

Il est possible d'avoir un pas négatif.
Pour afficher les nombres de 20 à 0 :

Boucle for

Il est possible d'avoir un pas négatif.
Pour afficher les nombres de 20 à 0 :

```
for i in range(20, -1, -1):
    print(i)
```

Boucle for ou while ?

En général, on utilise une boucle `for` lorsque l'on connaît le nombre d'itérations à réaliser (ex : calcul de somme, du $n$ème terme d'une suite...).

Sinon, on utilise une boucle `while`.

Exercice

Écrire un programme Python pour calculer $20\,!$ $(= 1 \times 2 \times ... \times 20)$.

Exercice

Écrire un programme Python pour calculer $20\,!$ $(= 1 \times 2 \times ... \times 20)$.

Exercice

Écrire un programme Python pour trouver le nombre de diviseurs de 2018.

Exercice

Écrire un programme Python pour calculer $20\,!$ $(= 1 \times 2 \times ... \times 20)$.

Exercice

Écrire un programme Python pour trouver le nombre de diviseurs de 2018.

Exercice

Écrire un programme Python pour trouver le plus petit entier n tel que $n! \geq 1000$.

Boucle for

On peut utiliser une boucle à l'intérieur d'une autre boucle :

```
for i in range(10):
    for j in range(10):
        print(i, "*", j, "=", i*j)
```

Question

Que fait cet algorithme ?

Boucle for

On peut utiliser une boucle à l'intérieur d'une autre boucle :

```
for i in range(10):
    for j in range(10):
        print(i, "*", j, "=", i*j)
```

Question

Que fait cet algorithme ?

Il affiche les tables de multiplications.

Boucle for

On peut utiliser une boucle à l'intérieur d'une autre boucle :

```
for i in range(10):
    for j in range(10):
        print(i, "*", j, "=", i*j)
```

Question

Combien de `print` sont exécutés ?

Boucle for

On peut utiliser une boucle à l'intérieur d'une autre boucle :

```
for i in range(10):
    for j in range(10):
        print(i, "*", j, "=", i*j)
```

Question

Combien de `print` sont exécutés ?

$10 \times 10 = 100$