Introduction à l'algorithmique et à la programmation

Informatique pour tous

Algorithme

Définition

Un **algorithme** est une suite d'**instructions** permettant de résoudre un **problème** : il prend une **entrée** (les données du problème, on dit aussi **instance**) et renvoie une **sortie** (la réponse au problème).

Algorithme

Question

Connaissez-vous des algorithmes ?

Algorithme = Recette ?

L'algorithme suivant résout le problème « Faire un gâteau au chocolat pour 6 personnes » :

Entrée :

- 200g de chocolat noir
- 100g de beurre
- 3 oeufs
- 50g de farine
- 100g de sucre en poudre

Sortie :
Gâteau au chocolat pour 6 personnes.

Algorithme = Recette ?

L'algorithme suivant résout le problème « Faire un gâteau au chocolat pour 6 personnes » :

Instructions :

```
Faire fondre le chocolat et le beurre dans une casserole
Ajouter le sucre, les oeufs et la farine dans un saladier
Ajouter le mélange chocolat/beurre et mélanger
Versez la pâte à gâteau dans un moule
Faire cuire au four 20 minutes
```

Algorithme d'Euclide

Entrée :
Deux nombres entiers a et b, avec $a \geq b \geq 0$
Sortie :
Le PGCD de a et b

Algorithme d'Euclide

Entrée :
Deux nombres entiers a et b, avec $a \geq b \geq 0$
Sortie :
Le PGCD de a et b

Tant que $b > 0$:
    $r =$ reste de la division de a par b
    $a = b$
    $b = r$
Renvoyer a

Vocabulaire

a, ***b*** et ***r*** sont des **variables**.
Une variable possède :

1. un **nom** (par ex. *a*)
2. un **type** (par ex. entier)
3. une **valeur** (par ex. 42)

Tant que $\boldsymbol{b} > 0$:
    $\boldsymbol{r}$ = reste de la division de $\boldsymbol{a}$ par $\boldsymbol{b}$
    $\boldsymbol{a} = \boldsymbol{b}$
    $\boldsymbol{b} = \boldsymbol{r}$
Renvoyer $\boldsymbol{a}$

Vocabulaire

« $\boldsymbol{b = r}$ » est une **affectation**, qui change la valeur de b pour mettre la valeur de r à la place.
⚠ ne pas confondre avec l'égalité mathématique, qui sera représentée par ==

Tant que $b > 0$:
    $\boldsymbol{r =}$ **reste de la division de** $\boldsymbol{a}$ **par** $\boldsymbol{b}$
    $\boldsymbol{a = b}$
    $\boldsymbol{b = r}$
Renvoyer a

Vocabulaire

« $\boldsymbol{b > 0}$ » est une **condition** (ou **booléen**) : elle peut être soit vraie, soit fausse.

Tant que $\boldsymbol{b > 0}$:
    $r =$ reste de la division de a par b
    $a = b$
    $b = r$
Renvoyer a

Vocabulaire

« **Tant que $b > 0$** » est une **boucle**.

Tant que $b > 0$:

$r =$ reste de la division de a par b

$a = b$

$b = r$

Renvoyer a

Exemple d'exécution

Un bon moyen de comprendre un algorithme est de l'exécuter sur un exemple simple.
Suivons l'exécution de l'algorithme d'Euclide avec, initialement, $a = 30$ et $b = 12$.

Exemple d'exécution

$\boxed{a == 30}$ $\boxed{b == 12}$

Tant que $b > 0$:

$r =$ reste de la division de a par b

$a = b$

$b = r$

Renvoyer a

Exemple d'exécution

$\boxed{a == 30}$ $\boxed{b == 12}$ $\boxed{\boldsymbol{r == 6}}$

Tant que $b > 0$:

r = reste de la division de a par b

$a = b$

$b = r$

Renvoyer a

Exemple d'exécution

$\boxed{\boldsymbol{a == 12}}$ $\boxed{b == 12}$ $\boxed{r == 6}$

Tant que $b > 0$:

$r =$ reste de la division de a par b

$\boldsymbol{a = b}$

$b = r$

Renvoyer a

Exemple d'exécution

$\boxed{a == 12}$ $\boxed{\boldsymbol{b == 6}}$ $\boxed{r == 6}$

Tant que $b > 0$:

$r =$ reste de la division de a par b

$a = b$

$\boldsymbol{b = r}$

Renvoyer a

Exemple d'exécution

$\boxed{a == 12}$ $\boxed{b == 6}$ $\boxed{r == 6}$

Tant que $b > 0$:

$r =$ reste de la division de a par b

$a = b$

$b = r$

Renvoyer a

Exemple d'exécution

$\boxed{a == 12}$ $\boxed{b == 6}$ $\boxed{\boldsymbol{r == 0}}$

Tant que $b > 0$:

r = reste de la division de a par b

$a = b$

$b = r$

Renvoyer a

Exemple d'exécution

$\boxed{\boldsymbol{a == 6}}$ $\boxed{b == 6}$ $\boxed{r == 0}$

Tant que $b > 0$:

$r =$ reste de la division de a par b

$\boldsymbol{a = b}$

$b = r$

Renvoyer a

Exemple d'exécution

$\boxed{a == 6}$ $\boxed{\boldsymbol{b == 0}}$ $\boxed{r == 0}$

Tant que $b > 0$:

$r =$ reste de la division de a par b

$a = b$

$\boldsymbol{b = r}$

Renvoyer a

Exemple d'exécution

$\boxed{a == 6}$ $\boxed{b == 0}$ $\boxed{r == 0}$

Tant que $b > 0$:

$r =$ reste de la division de a par b

$a = b$

$b = r$

Renvoyer a

Exemple d'exécution

$\boxed{a == 6}$ $\boxed{b == 0}$ $\boxed{r == 0}$

Tant que $b > 0$:

$r =$ reste de la division de a par b

$a = b$

$b = r$

Renvoyer *a*

Autre exemple

Question

Que vaut x et y à la fin de l'algorithme suivant ?

$x = 3$
$y = 22$
$x = x + y$
$y = x - y$
$x = x - y$

Algorithmique

Il y a plusieurs questions fondamentales que l'on se pose quand on conçoit un algorithme :

Algorithmique

Il y a plusieurs questions fondamentales que l'on se pose quand on conçoit un algorithme :

1. Est-ce que mon algorithme s'arrête toujours (on dit aussi **termine**) ? Ou est-ce qu'il peut faire une « boucle infinie » ?

Algorithmique

Il y a plusieurs questions fondamentales que l'on se pose quand on conçoit un algorithme :

1. Est-ce que mon algorithme s'arrête toujours (on dit aussi **termine**) ? Ou est-ce qu'il peut faire une « boucle infinie » ?
2. Est-ce que mon algorithme est **correct** ? C'est à dire : renvoie t-il la bonne réponse ?

Algorithmique

Il y a plusieurs questions fondamentales que l'on se pose quand on conçoit un algorithme :

1. Est-ce que mon algorithme s'arrête toujours (on dit aussi **termine**) ? Ou est-ce qu'il peut faire une « boucle infinie » ?
2. Est-ce que mon algorithme est **correct** ? C'est à dire : renvoie t-il la bonne réponse ?
3. Est-ce que mon algorithme est **rapide** ? Existe t-il un algorithme plus efficace pour faire la même chose ?

Langage de programmation

Pour l'instant on a écrit nos algorithmes en français.
Un **langage de programmation** est un langage comportant un ensemble de règles et de mots clés, permettant de faire exécuter un algorithme par un ordinateur.

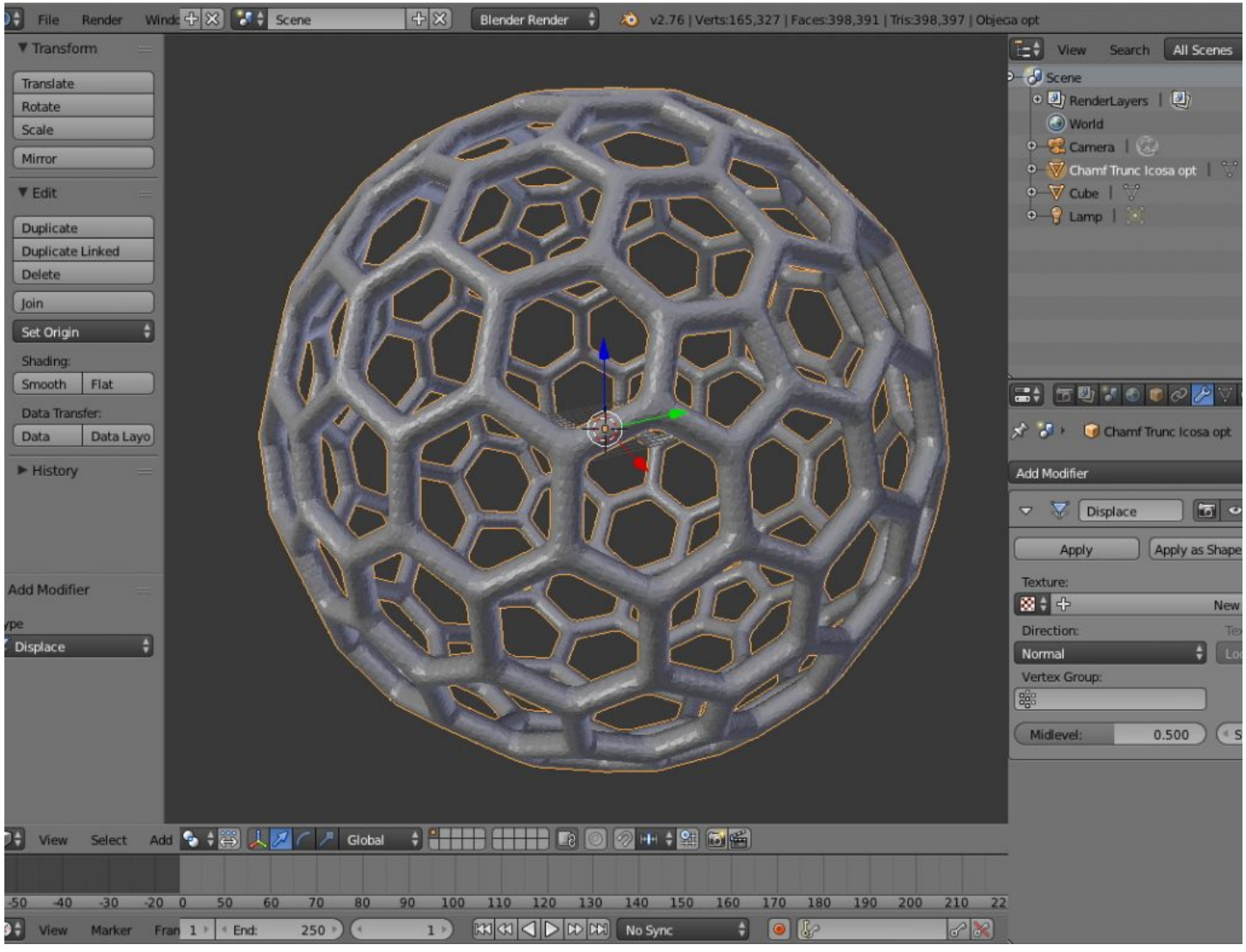
Python

Le langage de programmation utilisé pour le cours d'IPT est **Python**.
Il fait parti des langages les plus utilisés au monde, avec C/C++ et Java.
Il a de très nombreuses applications : industrie, web, scientifique...

Exemples de programmes Python

Civilization 4 (jeu vidéo)

Exemples de programmes Python



Blender (modélisation 3D)

Exemples de programmes Python

Dropbox (partage de fichiers)

Pyzo

Pour programmer confortablement, nous utilisons un éditeur qui s'appelle Pyzo et qui permet de :

1. sauvegarder/charger des fichiers .py contenant du code Python
2. colorier le code pour qu'il soit plus lisible
3. compléter automatiquement le nom des variables...

Pyzo


File Edit View Settings Shell Run Tools Help
M_pendule_double.py
tp_particule.py
simul_ressort.py
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation
sz = 300.
vmax = 50.
v_choc = 1.
dt = 0.1
Fg = np.array([0, 0])
ressort = np.array([20., 0., 300., 3.,
20.])
Editeur de fichier
n_particules = 50
particules = []
def init():
for i in range(n_particules):
p = []
p.append(ressort[0] +
np.random.random()*(sz - ressort[0]))
p.append(np.random.random()*sz)
p.append(-
np.random.random()*vmax)
Shells
Python
iew of IPython's features.
%quickref -> Quick reference.
help -> Python's own help syst
em.
object? -> Details about 'object'
, use 'object??' for extra details.
In [1]: a = 3 + 5
In [2]: a - 2
Out[2]: 6
Shell
In [3]:
Interactive help
abs
Print
Object: abs, class: builtin_function_or_method, repr: <built-in function abs>
Return the absolute value of the argument.

Déroulement de l'année

1. Vous devez installer Pyzo sur votre ordinateur (voir le lien sur hugoprépa), ou le télécharger sur clé USB.
2. Il est fortement conseillé de s'entraîner à programmer chez soi, en refaisant les exercices de cours/TP.
3. Les premiers mois d'IPT sont les plus importants (difficile à rattraper par la suite).

Types

Chaque variable possède un type en Python.

Voici les types de base :

- **Nombre entier** (`int`) : `0` `5` `-12`

Types

Chaque variable possède un type en Python.

Voici les types de base :

- **Nombre entier** (`int`) : 0 5 -12
- **Flottant** (`float`), nombre à virgule : 0.35 -1/3 -2.7564

Types

Chaque variable possède un type en Python.

Voici les types de base :

- **Nombre entier** (`int`) : 0 5 -12
- **Flottant** (`float`), nombre à virgule : 0.35 -1/3 -2.7564
- **Booléen** (`bool`), qui est soit vrai, soit faux : `True` `False`

Types

Chaque variable possède un type en Python.

Voici les types de base :

- **Nombre entier** (`int`) : 0 5 -12
- **Flottant** (`float`), nombre à virgule : 0.35 -1/3 -2.7564
- **Booléen** (`bool`), qui est soit vrai, soit faux : `True` `False`
- **Chaine de caractères** (`str`), entre guillemets :
  `"Hello World!"`

Types

Chaque variable possède un type en Python.

Voici les types de base :

- **Nombre entier** (`int`) : `0` `5` `-12`
- **Flottant** (`float`), nombre à virgule : `0.35` `-1/3` `-2.7564`
- **Booléen** (`bool`), qui est soit vrai, soit faux : `True` `False`
- **Chaine de caractères** (`str`), entre guillemets : `"Hello World!"`
- ***n*-uplets** (`tuple`), dont les éléments sont séparés par des virgules : `(1, -2)` `(1, "blabla", False)`

Types

Contrairement à de nombreux langages, en Python il n'y a pas besoin de dire de quel type est une variable : Python le déduit tout seul.

En cas de doute, il est possible d'écrire `type(a)`, qui donne le type de la variable `a`.

Types

Contrairement à de nombreux langages, en Python il n'y a pas besoin de dire de quel type est une variable : Python le déduit tout seul.

En cas de doute, il est possible d'écrire `type(a)`, qui donne le type de la variable `a`.

Il est important de donner des noms significatifs aux variables.
Évitez `a`, `b`, `c`, `d`, `e`...

Affectation

L'**affectation** en Python s'écrit :

$$\text{variable} = \text{expression}$$

« expression » est calculé, puis sa valeur est mise dans « variable ».

Affectation

L'**affectation** en Python s'écrit :

$$\text{variable} = \text{expression}$$

« expression » est calculé, puis sa valeur est mise dans « variable ».

Si cette variable n'existait pas encore, l'affectation s'appelle une **initialisation** et a pour effet de créer une nouvelle variable.

Affectation

L'**affectation** en Python s'écrit :

$$\text{variable} = \text{expression}$$

« expression » est calculé, puis sa valeur est mise dans « variable ».

Si cette variable n'existait pas encore, l'affectation s'appelle une **initialisation** et a pour effet de créer une nouvelle variable.

⚠ `=` n'est pas symétrique : `a = b` et `b = a` ne font pas la même chose.

Affectation

Exemple dans le Shell de Pyzo :

```
In [51]: a = 3

In [52]: a
Out[52]: 3

In [53]: a = 4*a + 2

In [54]: a
Out[54]: 14
```

Affectation

⚠ Une variable doit être initialisée avant d'être utilisée.

```
In [4]: b
---------------------------------
----------------------------
NameError
ck (most recent call last)
<ipython-input-4-3b5d5c371295> in
----> 1 b

NameError: name 'b' is not defined
```

Affectation

⚠ A gauche d'une affectation il ne peut y avoir qu'une variable, pas une expression.

```
In [15]: 2*a = 6
  File "<ipython-input-15-9a2a42313c6
    2*a = 6
        ^
SyntaxError: can't assign to operator
```

Affectation

```
In [61]: a = 11

In [62]: b = a

In [63]: a = 3
```

Que vaut `b` ?

Affectation

```
In [61]: a = 11

In [62]: b = a

In [63]: a = 3
```

Que vaut `b` ?

```
In [64]: b
Out[64]: 11
```

Quand on écrit `a = b` où `a` et `b` sont des nombres, `a` et `b` sont **indépendantes : modifier l'une ne modifie pas l'autre**.

Affectation

Il est possible d'affecter plusieurs variables en même temps :

```
In [17]: a, b, c = -4, True, 7.3
```

`a` pour effet de mettre simultanément la valeur `-4` dans `a`, `True` dans `b` et `7.3` dans `c`.

Affectation

Il est possible d'affecter plusieurs variables en même temps :

```
In [17]: a, b, c = -4, True, 7.3
```

a pour effet de mettre simultanément la valeur `-4` dans `a`, `True` dans `b` et `7.3` dans `c`.

C'est équivalent à `(a, b, c) = (-4, True, 7.3)` : initialisation d'un triplet.

Affectation

Question

Comment échanger les valeurs de deux variables x et y ?

Affectation

Question

Comment échanger les valeurs de deux variables x et y ?

On a déjà vu une possibilité...

Affectation

Question

Comment échanger les valeurs de deux variables x et y ?

On a déjà vu une possibilité...

2ème possibilité :

```
tmp = x
x = y
y = tmp
```

Affectation

Question

Comment échanger les valeurs de deux variables x et y ?

3ème possibilité :

```
x, y = y, x
```

Booléens

`==`, `!=`, `<`, `>`, `and`, `or`, `not` donnent des booléens.

Booléens

`==`, `!=`, `<`, `>`, `and`, `or`, `not` donnent des booléens.

`a == b` : booléen qui est `True` ssi la *valeur* de `a` est égale à la *valeur* de `b`.

Booléens

`==`, `!=`, `<`, `>`, `and`, `or`, `not` donnent des booléens.

`a == b` : booléen qui est `True` ssi la *valeur* de `a` est égale à la *valeur* de `b`.

⚠ Ne pas confondre avec l'affectation « `a = b` » !

Booléens

`==`, `!=`, `<`, `>`, `and`, `or`, `not` donnent des booléens.

`a == b` : booléen qui est `True` ssi la *valeur* de `a` est égale à la *valeur* de `b`.

⚠ Ne pas confondre avec l'affectation « `a = b` » !

⚠ `a == b` n'a de sens que si `a` et `b` sont de même type !

Booléens

`==`, `!=`, `<`, `>`, `and`, `or`, `not` donnent des booléens.

`a == b` : booléen qui est `True` ssi la *valeur* de `a` est égale à la *valeur* de `b`.

⚠ Ne pas confondre avec l'affectation « `a = b` » !

⚠ `a == b` n'a de sens que si `a` et `b` sont de même type !

`a != b` : booléen qui est `True` ssi la *valeur* de `a` est différente de la *valeur* de `b`.

Booléens

Si `b1` et `b2` sont des booléens :

- `b1` **and** `b2` : `True` ssi `b1` **et** `b2` sont `True`.
- `b1` **or** `b2` : `True` ssi `b1` **ou** `b2` sont `True`.
- **not** `b1` : `True` ssi `b1` est `False` (**négation**).

Booléens

Donner la valeur de :

```
True and (not ((False and True) or False))
```

Booléens

Donner la valeur de :

```
True and (not ((False and True) or False))
```

```
In [8]: True and (not ((False and True) or False))
Out[8]: True
```

Booléens

On peut aussi stocker le résultat dans une variable :

```
In [9]: a = True and (not ((False and True) or False))

In [10]: a
Out[10]: True
```

Booléens

Question

Comment écrire le **ou exclusif** de deux booléens `b1` et `b2` (vrai si `b1` ou `b2` est vrai, mais pas les deux) ?

Booléens

Question

Comment écrire le **ou exclusif** de deux booléens `b1` et `b2` (vrai si `b1` ou `b2` est vrai, mais pas les deux) ?

```
(b1 or b2) and (not (b1 and b2))
```

Booléens

<, >, <=, >= permettent de comparer des nombres.

```
In [9]: a = 7

In [10]: a > 2 and a < 7
Out[10]: False

In [11]: a > 2 and a <= 7
Out[11]: True
```

Flottants

Les flottants sont des décimaux : ils ont un nombre fini de chiffres après la virgule.

Flottants

Les flottants sont des décimaux : ils ont un nombre fini de chiffres après la virgule.

Il n'est pas possible de stocker un nombre réel exact :

Flottants

Les flottants sont des décimaux : ils ont un nombre fini de chiffres après la virgule.

Il n'est pas possible de stocker un nombre réel exact : il faudrait pour cela un nombre infini de chiffres après la virgule, ce qui n'est pas possible de stocker dans une mémoire finie.

Flottants vs n-uplet

⚠ La virgule d'un flottant est en fait un point « . »

⚠ « , » est utilisé pour séparer les éléments d'un n-uplet.

Flottants vs *n*-uplet

⚠ La virgule d'un flottant est en fait un point « . »

⚠ « , » est utilisé pour séparer les éléments d'un *n*-uplet.

```
In [23]: a = 1,2

In [24]: a
Out[24]: (1, 2)

In [25]: b = 1.2 + a
```

```
TypeError: unsupported operand type(s) for +: 'float' and 'tuple'
```

Opérations sur les nombres

- `a * b` : multiplication de `a` et `b`.

Opérations sur les nombres

- `a * b` : multiplication de `a` et `b`.
- `a ** b` : `a` puissance `b`.

Opérations sur les nombres

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

Opérations sur les nombres

- `a * b` : multiplication de `a` et `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`.

Opérations sur les nombres

- `a * b` : multiplication de `a` et `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`.
- `a / b` : quotient **flottant** de la division de `a` par `b`.

Erreur classique

On exécute le code suivant :

```
a = 3
b = 4a
```

Erreur classique

On exécute le code suivant :

```
a = 3
b = 4a
```

On obtient une erreur :

```
  File "<tmp 1>", line 2
    b = 4a
          ^
SyntaxError: invalid syntax
```

Erreur classique

On exécute le code suivant :

```
a = 3
b = 4a
```

On obtient une erreur :

```
  File "<tmp 1>", line 2
    b = 4a
          ^
SyntaxError: invalid syntax
```

oubli du `*` : il faut écrire `b = 4*a`

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` ?

$$\texttt{a} \text{ divise } \texttt{b} \iff \text{le reste de la division de } \texttt{b} \text{ par } \texttt{a} \text{ est nul}$$

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

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

Divisibilité

```
In [55]: 27 % 3
Out[55]: 0

In [56]: 27 % 3 == 0
Out[56]: True

In [57]: 54 % 5 == 0
Out[57]: False

In [58]: -54 % 2 == 0
Out[58]: True

In [59]: 3 % 0 == 0
-------------------------
-------------
ZeroDivisionError
```

QCM 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

QCM ICNA

Question 4 :

Parmi les scripts suivant lesquels échangent les valeurs de a et b ?

A)
```
c=a
b=c
c=a
```

B)
```
c=a
b=c
a=b
```

C)
```
a=a+b
b=a-b
a=a-b
```

D)
```
a=a+b
b=a-b
a=a+b
```