Base de l'informatique 1

Cours 0. Expressions, écriture binaire et logique

Olivier Baldellon

Courriel : prénom.nom@univ-cotedazur.fr

Page professionnelle : https://upinfo.univ-cotedazur.fr/~obaldellon/

Licence 1 — Faculté des sciences et ingénierie de Nice — Université Côte d'Azur

Sommaire

Avant de me contacter

- La réponse est-elle sur moodle ?
- sur `https://upinfo.univ-cotedazur.fr/~obaldellon/python` ?
- sur le site de la licence info : `https://upinfo.univ-cotedazur.fr`
- Puis-je attendre la fin d'un cours en amphi ?

Écrire à l'enseignant

- Sur mon email `@univ-cotedazur.fr` (surtout pas sur moodle)
- avec votre adresse étudiante `@etu.univ-cotedazur.fr`
- Objet : clair et précis
- Rappel du nom du cours, de votre groupe
- Rappel de votre nom (signature suffisante)
- Si la question vous concerne (choix d'IP, emploi du temps, lettre de recommandation) : numéro d'étudiant !

Exemple

```
Objet : TP Python annulé à cause des grèves ?

Bonjour Monsieur,

Je suis étudiant du groupe A2 du cours de Python. Le TP
du mercredi 29 juillet à 10h est-il annulé à cause des
grèves ?

Cordialement,

Nicolas Bourbaki (22314159)
```

- Écrivez des messages brefs qui vont à l'essentiel !

- CM, TD & TP
  - 10 séances de cours magistral.
  - 9 séances de 2h de TD (semaines de 0 à 8)
  - 9 séances de 2h de TP (semaines de 1 à 9) : semaine prochaine pas de TP

- **CM, TD & TP**
  - 10 séances de cours magistral.
  - 9 séances de 2h de TD (semaines de 0 à 8)
  - 9 séances de 2h de TP (semaines de 1 à 9) : semaine prochaine pas de TP
- **Évaluation**
  - un partiel en cours de semestre ≈ 50%.
  - un examen terminal à la fin du semestre ≈ 50%

- **CM, TD & TP**
  - 10 séances de cours magistral.
  - 9 séances de 2h de TD (semaines de 0 à 8)
  - 9 séances de 2h de TP (semaines de 1 à 9) : semaine prochaine pas de TP
- **Évaluation**
  - un partiel en cours de semestre ≈ 50%.
  - un examen terminal à la fin du semestre ≈ 50%
  - des interros surprises (peut-être… c'est une surprise)
  - un projet bonus et facultatif

- **CM, TD & TP**
  - 10 séances de cours magistral.
  - 9 séances de 2h de TD (semaines de 0 à 8)
  - 9 séances de 2h de TP (semaines de 1 à 9) : semaine prochaine pas de TP
- **Évaluation**
  - un partiel en cours de semestre ≈ 50%.
  - un examen terminal à la fin du semestre ≈ 50%
  - des interros surprises (peut-être… c'est une surprise)
  - un projet bonus et facultatif
- **En plus de vos notes manuscrites, vous trouverez sur mon site web :**
  - Transparents des cours magistraux (avec ou sans animations).
    - Version avec animations pour relire et retravailler le cours chez soi.
    - Version sans animations pour retrouver rapidement une information.
  - Sujets et corrigés des derniers exercices de TP/TD.

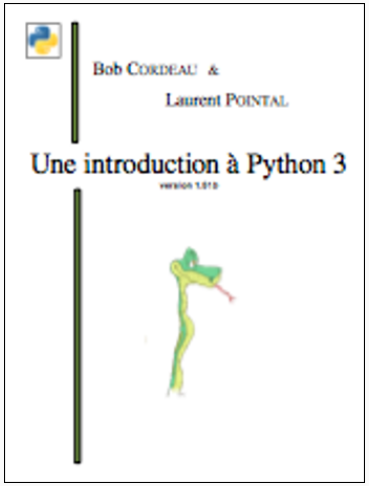
- **CM, TD & TP**
  - 10 séances de cours magistral.
  - 9 séances de 2h de TD (semaines de 0 à 8)
  - 9 séances de 2h de TP (semaines de 1 à 9) : semaine prochaine pas de TP
- **Évaluation**
  - un partiel en cours de semestre ≈ 50%.
  - un examen terminal à la fin du semestre ≈ 50%
  - des interros surprises (peut-être… c'est une surprise)
  - un projet bonus et facultatif
- **En plus de vos notes manuscrites, vous trouverez sur mon site web :**
  - Transparents des cours magistraux (avec ou sans animations).
    - Version avec animations pour relire et retravailler le cours chez soi.
    - Version sans animations pour retrouver rapidement une information.
  - Sujets et corrigés des derniers exercices de TP/TD.
    - pour avoir les corrections des premiers exercices : venez en TD/TP !

Quelques références pour ce cours

- La documentation officielle Python : `http://docs.python.org/py3k`

- Livre gratuit en ligne

- Poly. IUT Orsay



- Livre de mon prédecesseur

- De très nombreuses autres références en ligne ou à la BU

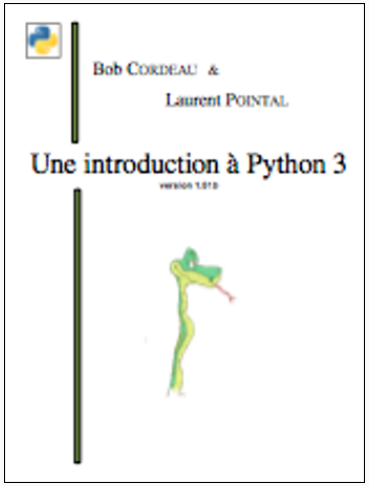
- En particulier le livre de Jean-Paul Roy (à l'origine de cette UE) est à la BU.

Quelques références pour ce cours

- La documentation officielle Python : `http://docs.python.org/py3k`

- Livre gratuit en ligne

- Poly. IUT Orsay



- Livre de mon prédecesseur

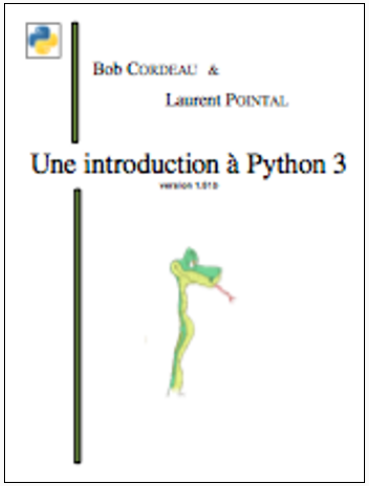
- De très nombreuses autres références en ligne ou à la BU
- En particulier le livre de Jean-Paul Roy (à l'origine de cette UE) est à la BU.
- La meilleure référence :

Quelques références pour ce cours

- La documentation officielle Python : `http://docs.python.org/py3k`

- Livre gratuit en ligne

- Poly. IUT Orsay



- Livre de mon prédecesseur

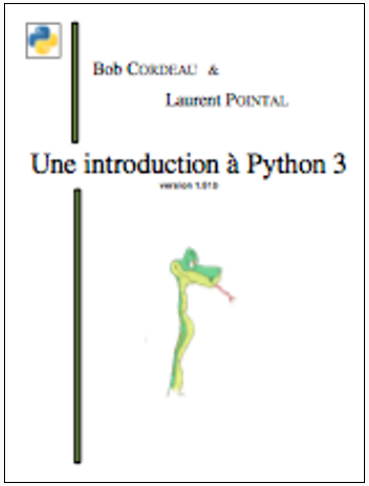
- De très nombreuses autres références en ligne ou à la BU
- En particulier le livre de Jean-Paul Roy (à l'origine de cette UE) est à la BU.
- La meilleure référence : c'est mon cours :)

Quelques références pour ce cours

- La documentation officielle Python : `http://docs.python.org/py3k`

- Livre gratuit en ligne

- Poly. IUT Orsay



- Livre de mon prédecesseur

- De très nombreuses autres références en ligne ou à la BU
- En particulier le livre de Jean-Paul Roy (à l'origine de cette UE) est à la BU.
- La meilleure référence : c'est mon cours :)
  - Pas dans l'absolue, mais pour préparer cette UE !

Algorithmique

- Comment résoudre un problème ?
- Branche des mathématiques (feuilles blanches et crayon)
- Ce problème peut ensuite être résolu par :
  - Un humain : chercher un mot dans le dictionnaire, poser une addition
  - Une machine : modifier une image, décider de votre orientation (parcoursup)

Algorithmique

- Comment résoudre un problème ?
- Branche des mathématiques (feuilles blanches et crayon)
- Ce problème peut ensuite être résolu par :
  - Un humain : chercher un mot dans le dictionnaire, poser une addition
  - Une machine : modifier une image, décider de votre orientation (parcoursup)

Programmation

- Donner des consignes (par exemple un algorithme) à une machine
- On utilise une langage artificiel (C, **Python**, OCaml, Java, Lisp, etc)
- Tout ce que fait un ordinateur est décrit sous forme de texte.
  - Les fichiers contenant ces textes s'appelle le code source
- Exemples de tâche : lire une vidéo, faire un calcul, modifier une image

C'est un cours de programmation avec Python version 3.

- Créé par le Néerlandais Guido van Rossum en **1989**.
- **Langage de haut niveau** :
  - bas niveau : proche du fonctionnement de la machine
  - haut niveau : proche du raisonnement humain
- **Langage multi-paradigmes** : impératif, fonctionnel, orienté objets.
- **Nombreuses bibliothèques**.
- **Libre** : code source disponible et modifiable, gratuit.
- **Attention** Python 2 est un **autre langage**. Très proche, certes.

Objectif du cours

- Ce n'est pas un cours de Python !
  - c'est un cours d'introduction à l'informatique (au sens large)
  - c'est un cours d'initiation à la programmation impérative.

Objectif du cours

- Ce n'est pas un cours de Python !
  - c'est un cours d'introduction à l'informatique (au sens large)
  - c'est un cours d'initiation à la programmation impérative.
- La programmation impérative ? Kézaco ?
  - `while`, `if`, fonction et variable
  - Tout le reste n'est que raccourcis.

Objectif du cours

- Ce n'est pas un cours de Python !
  - c'est un cours d'introduction à l'informatique (au sens large)
  - c'est un cours d'initiation à la programmation impérative.
- La programmation impérative ? Kézaco ?
  - `while`, `if`, fonction et variable
  - Tout le reste n'est que raccourcis.
- Apprendre à programmer ce n'est pas apprendre plein d'astuces
  - fonction prédéfinie « `max(liste)` »
  - méthode « `liste.sort()` »
  - mots-clé « `x in liste` »
  - construction syntaxique « `[f(x) for x in L]` »

Objectif du cours

- Ce n'est pas un cours de Python !
  - c'est un cours d'introduction à l'informatique (au sens large)
  - c'est un cours d'initiation à la programmation impérative.
- La programmation impérative ? Kézaco ?
  - `while`, `if`, fonction et variable
  - Tout le reste n'est que raccourcis.
- Apprendre à programmer ce n'est pas apprendre plein d'astuces
  - fonction prédéfinie « `max(liste)` »
  - méthode « `liste.sort()` »
  - mots-clé « `x in liste` »
  - construction syntaxique « `[f(x) for x in L]` »
- Pourquoi se focaliser sur les bases :
  - Permet d'apprendre rapidement d'autres langages
  - Permet d'apprendre à résoudre des problèmes plus complexes
  - Permet de programmer plus efficacement
  - Permet de comprendre que l'informatique ce n'est pas magique…

Objectif du cours

- Ce n'est pas un cours de Python !
  - c'est un cours d'introduction à l'informatique (au sens large)
  - c'est un cours d'initiation à la programmation impérative.
- La programmation impérative ? Kézaco ?
  - `while`, `if`, fonction et variable
  - Tout le reste n'est que raccourcis.
- Apprendre à programmer ce n'est pas apprendre plein d'astuces
  - fonction prédéfinie « `max(liste)` »
  - méthode « `liste.sort()` »
  - mots-clé « `x in liste` »
  - construction syntaxique « `[f(x) for x in L]` »
- Pourquoi se focaliser sur les bases :
  - Permet d'apprendre rapidement d'autres langages
  - Permet d'apprendre à résoudre des problèmes plus complexes
  - Permet de programmer plus efficacement
  - Permet de comprendre que l'informatique ce n'est pas magique… c'est juste vachement chouette !

Sommaire

- L'écriture en base 10 :
  - 527
  - $= 5 \times 100 + 2 \times 10 + 1$
  - $= 5 \times 10^2 + 2 \times 10^1 + 1 \times 10^0$

- L'écriture en base 2 :
  - 1010
  - $= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
  - $= 1 \times 8 + 1 \times 2 = 10$

Convertir en binaire

- Convertir 37 en binaire
  - $37 = 2 \times 18 + 1$

Convertir en binaire

- Convertir 37 en binaire
  - $37 = 2 \times 18 + 1$
  - $18 = 2 \times 9 + 0$

Convertir en binaire

- Convertir 37 en binaire
  - $37 = 2 \times 18 + 1$
  - $18 = 2 \times 9 + 0$
  - $9 = 2 \times 4 + 1$

Convertir en binaire

- Convertir 37 en binaire
  - $37 = 2 \times 18 + 1$
  - $18 = 2 \times 9 + 0$
  - $9 = 2 \times 4 + 1$
  - $4 = 2 \times 2 + 0$

Convertir en binaire

- Convertir 37 en binaire
  - $37 = 2 \times 18 + 1$
  - $18 = 2 \times 9 + 0$
  - $9 = 2 \times 4 + 1$
  - $4 = 2 \times 2 + 0$
  - $2 = 2 \times 1 + 0$

Convertir en binaire

- Convertir 37 en binaire
  - $37 = 2 \times 18 + 1$
  - $18 = 2 \times 9 + 0$
  - $9 = 2 \times 4 + 1$
  - $4 = 2 \times 2 + 0$
  - $2 = 2 \times 1 + 0$
  - $1 = 2 \times 0 + 1$

Convertir en binaire

- Convertir 37 en binaire
  - $37 = 2 \times 18 + 1$
  - $18 = 2 \times 9 + 0$
  - $9 = 2 \times 4 + 1$
  - $4 = 2 \times 2 + 0$
  - $2 = 2 \times 1 + 0$
  - $1 = 2 \times 0 + 1$
- On s'arrête lorsque que le quotient est nul

Convertir en binaire

- Convertir 37 en binaire
  - $37 = 2 \times 18 + 1$
  - $18 = 2 \times 9 + 0$
  - $9 = 2 \times 4 + 1$
  - $4 = 2 \times 2 + 0$
  - $2 = 2 \times 1 + 0$
  - $1 = 2 \times 0 + 1$
- On s'arrête lorsque que le quotient est nul
- On lit le résultat de bas en haut : $37_{10} = 100101_2$

- Que vaut 10110_2 ?

- Que vaut 10110_2 ?
  - $1_2 \qquad = 1$

Convertir depuis le binaire

- Que vaut 10110_2 ?
  - $1_2 = 1$
  - $10_2 = 2 \times 1 + 0 = 2$

Convertir depuis le binaire

- Que vaut 10110_2 ?
  - $1_2 = 1$
  - $10_2 = 2 \times 1 + 0 = 2$
  - $101_2 = 2 \times 2 + 1 = 5$

Convertir depuis le binaire

- Que vaut 10110_2 ?
  - $1_2 = 1$
  - $10_2 = 2 \times 1 + 0 = 2$
  - $101_2 = 2 \times 2 + 1 = 5$
  - $1011_2 = 2 \times 5 + 1 = 11$

Convertir depuis le binaire

- Que vaut 10110_2 ?
  - $1_2 = 1$
  - $10_2 = 2 \times 1 + 0 = 2$
  - $101_2 = 2 \times 2 + 1 = 5$
  - $1011_2 = 2 \times 5 + 1 = 11$
  - $10110_2 = 2 \times 11 + 0 = 22$

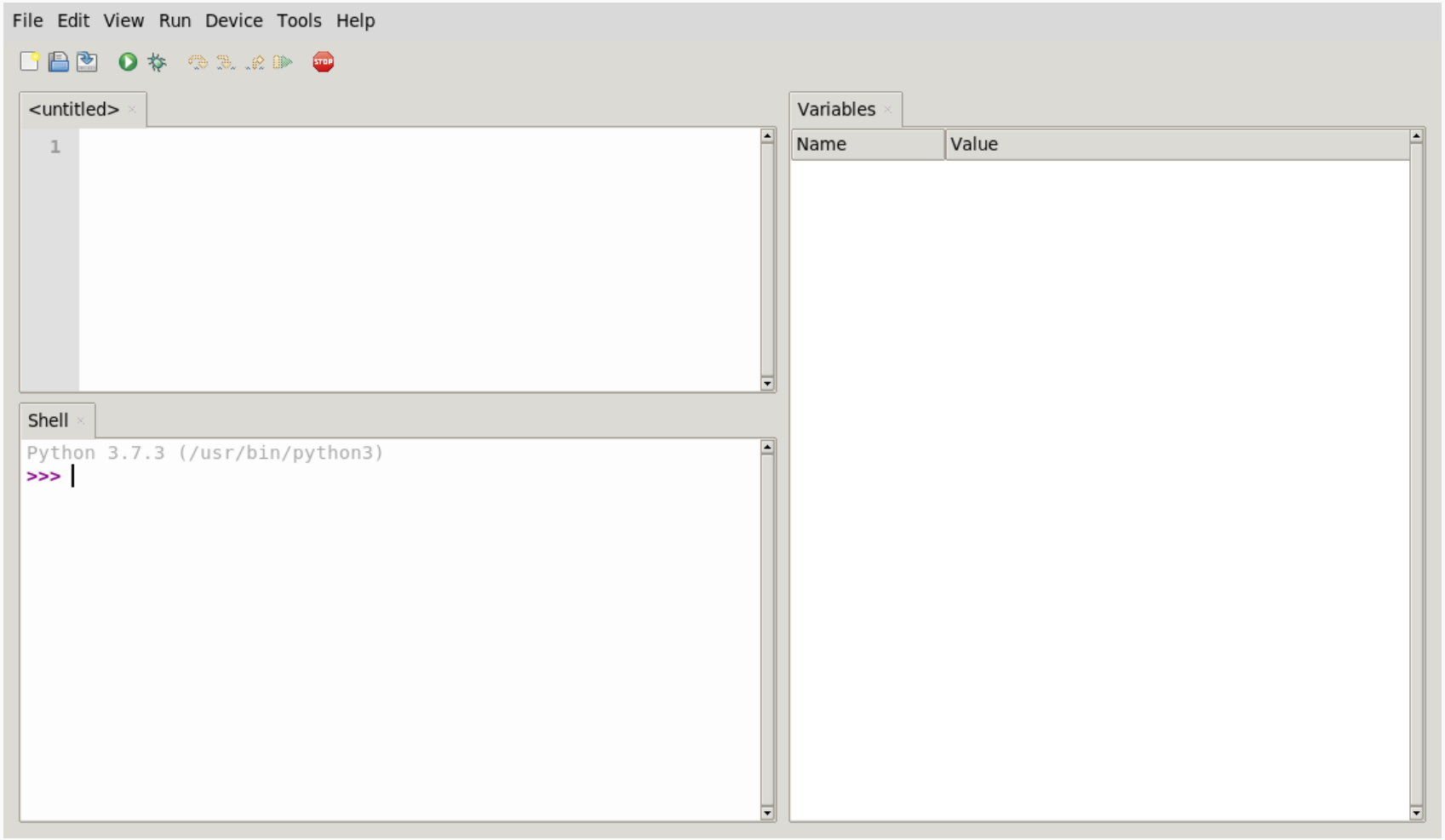
Convertir depuis le binaire

- Que vaut 10110_2 ?
  - $1_2 = 1$
  - $10_2 = 2 \times 1 + 0 = 2$
  - $101_2 = 2 \times 2 + 1 = 5$
  - $1011_2 = 2 \times 5 + 1 = 11$
  - $10110_2 = 2 \times 11 + 0 = 22$

- On a donc $10110_2 = 22_{10}$

Sommaire

À installer au plus vite sur votre ordinateur via le site `https://thonny.org`



Session interactive

Python a une console (ou shell, ou toplevel, ou REPL) avec laquelle on peut interagir. On peut lui soumettre un calcul comme à une calculatrice.

SHELL
```
>>>
```

Python a une console (ou shell, ou toplevel, ou REPL) avec laquelle on peut interagir. On peut lui soumettre un calcul comme à une calculatrice.

SHELL
```
>>> 1+1
```

Python a une console (ou shell, ou toplevel, ou REPL) avec laquelle on peut interagir. On peut lui soumettre un calcul comme à une calculatrice.

SHELL
```
>>> 1+1
2
>>>
```

Python a une console (ou shell, ou toplevel, ou REPL) avec laquelle on peut interagir. On peut lui soumettre un calcul comme à une calculatrice.

SHELL
```
>>> 1+1
2
>>> 123//10
```

Python a une console (ou shell, ou toplevel, ou REPL) avec laquelle on peut interagir. On peut lui soumettre un calcul comme à une calculatrice.

SHELL
```
>>> 1+1
2
>>> 123//10
12
>>>
```

Python a une console (ou shell, ou toplevel, ou REPL) avec laquelle on peut interagir. On peut lui soumettre un calcul comme à une calculatrice.

SHELL
```
>>> 1+1
2
>>> 123//10
12
>>> 2**10
```

Python a une console (ou shell, ou toplevel, ou REPL) avec laquelle on peut interagir. On peut lui soumettre un calcul comme à une calculatrice.

SHELL
```
>>> 1+1
2
>>> 123//10
12
>>> 2**10
1024
```

Python a une console (ou shell, ou toplevel, ou REPL) avec laquelle on peut interagir. On peut lui soumettre un calcul comme à une calculatrice.

SHELL
```
>>> 1+1
2
>>> 123//10
12
>>> 2**10
1024
```

- Une opération mal utilisée peut provoquer une erreur.

SHELL
```
>>> 5//0 # Bouh, la vilaine division par zéro
```

Python a une console (ou shell, ou toplevel, ou REPL) avec laquelle on peut interagir. On peut lui soumettre un calcul comme à une calculatrice.

SHELL
```
>>> 1+1
2
>>> 123//10
12
>>> 2**10
1024
```

- Une opération mal utilisée peut provoquer une erreur.

SHELL
```
>>> 5//0 # Bouh, la vilaine division par zéro
Traceback (most recent call last):
  File "<console>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

Python a une console (ou shell, ou toplevel, ou REPL) avec laquelle on peut interagir. On peut lui soumettre un calcul comme à une calculatrice.

SHELL
```
>>> 1+1
2
>>> 123//10
12
>>> 2**10
1024
```

- Une opération mal utilisée peut provoquer une erreur.

SHELL
```
>>> 5//0 # Bouh, la vilaine division par zéro
Traceback (most recent call last):
  File "<console>", line 1, in <module>
ZeroDivisionError:  integer division or modulo by zero
```

- Le code après # est un commentaire, non pris en compte par Python.

Python a une console (ou shell, ou toplevel, ou REPL) avec laquelle on peut interagir. On peut lui soumettre un calcul comme à une calculatrice.

SHELL
```
>>> 1+1
2
>>> 123//10
12
>>> 2**10
1024
```

- Une opération mal utilisée peut provoquer une erreur.

SHELL
```
>>> 5//0 # Bouh, la vilaine division par zéro
Traceback (most recent call last):
  File "<console>", line 1, in <module>
ZeroDivisionError:  integer division or modulo by zero
```

- Le code après # est un commentaire, non pris en compte par Python.
- Lisez-bien les messages d'erreurs ! Il sont souvent assez clairs.

En informatique, il existe de nombreux types.

- Les nombres :
  - le type entier `1987`
  - le type flottant `6.55957`
  - le type complexe `1+2j`

En informatique, il existe de nombreux types.

- Les nombres :
  - le type entier `1987`
  - le type flottant `6.55957`
  - le type complexe `1+2j`
- Les symboles et le texte :
  - les caractères `'a'` ou `'é'` ou `' '`
  - les chaînes `"Bonjour à toi"`

En informatique, il existe de nombreux types.

- Les nombres :
  - le type entier `1987`
  - le type flottant `6.55957`
  - le type complexe `1+2j`
- Les symboles et le texte :
  - les caractères `'a'` ou `'é'` ou `' '`
  - les chaînes `"Bonjour à toi"`
- Les booléans
  - `True` et `False`

En informatique, il existe de nombreux types.

- Les nombres :
  - le type entier `1987`
  - le type flottant `6.55957`
  - le type complexe `1+2j`
- Les symboles et le texte :
  - les caractères `'a'` ou `'é'` ou `' '`
  - les chaînes `"Bonjour à toi"`
- Les booléans
  - `True` et `False`
- Les collections : listes, tableau, dictionnaires

En informatique, toute valeur à un type

Sommaire

Opérations de base sur les entiers

- Les opérations classiques : `+` `-` `*`
- Le quotient `//` et le reste `%` de la division euclidienne
- `a**b` calcule la puissance a^b

À connaître par cœur ♥

- `a%b` se lit a modulo `b`
- `a` est un multiple de `b` si et seulement si `a%b=0`.
- en particulier, `n` est pair si et seulement si `n%2=0`

Exemple : Heures et minutes

- **Problème 1**

Il est 15h43, combien de minutes se sont-elles écoulées depuis minuit ?

- **Problème 1**

Il est 15h43, combien de minutes se sont-elles écoulées depuis minuit ?

- **Problème 2**

Sachant que 1024 minutes se sont écoulées depuis minuit, le cours de programmation impérative est-il terminé ?

- Les nombres à virgules sont représentés avec un point : `6.55957`

- Les nombres à virgules sont représentés avec un point : `6.55957`

- On parle de nombre à virgule flottante
  - ou simplement de flottant

- Les nombres à virgules sont représentés avec un point : `6.55957`
- On parle de nombre à virgule flottante
  - ou simplement de flottant
- Pour les grands nombres et les petits on utilise la notation scientifique :

```
>>>
```
SHELL

- Les nombres à virgules sont représentés avec un point : `6.55957`
- On parle de nombre à virgule flottante
  - ou simplement de flottant
- Pour les grands nombres et les petits on utilise la notation scientifique :

SHELL
```
>>> 12345678910111213.
```

- Les nombres à virgules sont représentés avec un point : `6.55957`
- On parle de nombre à virgule flottante
  - ou simplement de flottant
- Pour les grands nombres et les petits on utilise la notation scientifique :

SHELL
```
>>> 12345678910111213.
1.2345678910111212e+16
>>>
```

- Les nombres à virgules sont représentés avec un point : `6.55957`
- On parle de nombre à virgule flottante
  - ou simplement de flottant
- Pour les grands nombres et les petits on utilise la notation scientifique :

SHELL
```
>>> 12345678910111213.
1.2345678910111212e+16
>>> 5*10. **20
```

- Les nombres à virgules sont représentés avec un point : `6.55957`
- On parle de nombre à virgule flottante
  - ou simplement de flottant
- Pour les grands nombres et les petits on utilise la notation scientifique :

SHELL
```
>>> 12345678910111213.
1.2345678910111212e+16
>>> 5*10. **20
5e+20
>>>
```

- Les nombres à virgules sont représentés avec un point : `6.55957`
- On parle de nombre à virgule flottante
  - ou simplement de flottant
- Pour les grands nombres et les petits on utilise la notation scientifique :

SHELL
```
>>> 12345678910111213.
1.2345678910111212e+16
>>> 5*10. **20
5e+20
>>> 5/100000
```

- Les nombres à virgules sont représentés avec un point : `6.55957`
- On parle de nombre à virgule flottante
  - ou simplement de flottant
- Pour les grands nombres et les petits on utilise la notation scientifique :

SHELL
```
>>> 12345678910111213.
1.2345678910111212e+16
>>> 5*10. **20
5e+20
>>> 5/100000
5e-05
```

- Les nombres à virgules sont représentés avec un point : `6.55957`
- On parle de nombre à virgule flottante
  - ou simplement de flottant
- Pour les grands nombres et les petits on utilise la notation scientifique :

SHELL
```
>>> 12345678910111213.
1.2345678910111212e+16
>>> 5*10. **20
5e+20
>>> 5/100000
5e-05
```

- 5×10^6 devient `5e6` ou `5e+6`.

Opérations de base sur les flottant

- Les opérations classiques : `+` `-` `*`
- La division décimale `/`
- `a**b` calcule la puissance a^b

En cas d'opération entre flottant et entier, le résultat est flottant

SHELL
```
>>>
```

Opérations de base sur les flottant

- Les opérations classiques : `+` `-` `*`
- La division décimale `/`
- `a**b` calcule la puissance a^b

En cas d'opération entre flottant et entier, le résultat est flottant

SHELL
```
>>> 1 + 1.5
```

Opérations de base sur les flottant

- Les opérations classiques : `+` `-` `*`
- La division décimale `/`
- `a**b` calcule la puissance a^b

En cas d'opération entre flottant et entier, le résultat est flottant

SHELL
```
>>> 1 + 1.5
2.5
>>>
```

Opérations de base sur les flottant

- Les opérations classiques : `+` `-` `*`
- La division décimale `/`
- `a**b` calcule la puissance a^b

En cas d'opération entre flottant et entier, le résultat est flottant

SHELL
```
>>> 1 + 1.5
2.5
>>> 12/2
```

Opérations de base sur les flottant

- Les opérations classiques : `+` `-` `*`
- La division décimale `/`
- `a**b` calcule la puissance a^b

En cas d'opération entre flottant et entier, le résultat est flottant

SHELL

```
>>> 1 + 1.5
2.5
>>> 12/2
6.0
>>>
```

Opérations de base sur les flottant

- Les opérations classiques : `+` `-` `*`
- La division décimale `/`
- `a**b` calcule la puissance a^b

En cas d'opération entre flottant et entier, le résultat est flottant

SHELL

```
>>> 1 + 1.5
2.5
>>> 12/2
6.0
>>> 25**(0.5)
```

Opérations de base sur les flottant

- Les opérations classiques : `+` `-` `*`
- La division décimale `/`
- `a**b` calcule la puissance a^b

En cas d'opération entre flottant et entier, le résultat est flottant

SHELL

```
>>> 1 + 1.5
2.5
>>> 12/2
6.0
>>> 25**(0.5)
5.0
```

- Ordre de priorité des opérateurs.

moins prioritaire	plus prioritaire	très prioritaire
`+` `-`	`*` `//` `%` `/`	`**`

- Ordre de priorité des opérateurs.

moins prioritaire	plus prioritaire	très prioritaire
`+` `-`	`*` `//` `%` `/`	`**`

- En cas de doute, on peut utiliser des parenthèses inutiles.

SHELL
```
>>> 5 - 8 + 4 * 2 ** 3 # Attention à la priorité !
```

- Ordre de priorité des opérateurs.

moins prioritaire	plus prioritaire	très prioritaire
`+ -`	`* // % /`	`**`

- En cas de doute, on peut utiliser des parenthèses inutiles.

SHELL
```
>>> 5 - 8 + 4 * 2 ** 3 # Attention à la priorité !
29
>>>
```

- Ordre de priorité des opérateurs.

moins prioritaire	plus prioritaire	très prioritaire
`+ -`	`* // % /`	`**`

- En cas de doute, on peut utiliser des parenthèses inutiles.

SHELL
```
>>> 5 - 8 + 4 * 2 ** 3 # Attention à la priorité !
29
>>> (5 - 8) + (4 * (2 ** 3))
```

Opérations et priorités

- Ordre de priorité des opérateurs.

moins prioritaire	plus prioritaire	très prioritaire
`+ -`	`* // % /`	`**`

- En cas de doute, on peut utiliser des parenthèses inutiles.

SHELL

```
>>> 5 - 8 + 4 * 2 ** 3 # Attention à la priorité !
29
>>> (5 - 8) + (4 * (2 ** 3))
29
```

- Ordre de priorité des opérateurs.

moins prioritaire	plus prioritaire	très prioritaire
`+ -`	`* // % /`	`**`

- En cas de doute, on peut utiliser des parenthèses inutiles.

SHELL
```
>>> 5 - 8 + 4 * 2 ** 3 # Attention à la priorité !
29
>>> (5 - 8) + (4 * (2 ** 3))
29
```

- Remarque : associativité à gauche pour les opérations de même priorité.

SHELL
```
>>>
```

- Ordre de priorité des opérateurs.

moins prioritaire	plus prioritaire	très prioritaire
`+ -`	`* // % /`	`**`

- En cas de doute, on peut utiliser des parenthèses inutiles.

SHELL
```
>>> 5 - 8 + 4 * 2 ** 3 # Attention à la priorité !
29
>>> (5 - 8) + (4 * (2 ** 3))
29
```

- Remarque : associativité à gauche pour les opérations de même priorité.

SHELL
```
>>> 1-2+5
```

- Ordre de priorité des opérateurs.

moins prioritaire	plus prioritaire	très prioritaire
`+ -`	`* // % /`	`**`

- En cas de doute, on peut utiliser des parenthèses inutiles.

SHELL
```
>>> 5 - 8 + 4 * 2 ** 3 # Attention à la priorité !
29
>>> (5 - 8) + (4 * (2 ** 3))
29
```

- Remarque : associativité à gauche pour les opérations de même priorité.

SHELL
```
>>> 1-2+5
4
>>>
```

- Ordre de priorité des opérateurs.

moins prioritaire	plus prioritaire	très prioritaire
`+ -`	`* // % /`	`**`

- En cas de doute, on peut utiliser des parenthèses inutiles.

SHELL
```
>>> 5 - 8 + 4 * 2 ** 3 # Attention à la priorité !
29
>>> (5 - 8) + (4 * (2 ** 3))
29
```

- Remarque : associativité à gauche pour les opérations de même priorité.

SHELL
```
>>> 1-2+5
4
>>> (1-2)+5
```

- Ordre de priorité des opérateurs.

moins prioritaire	plus prioritaire	très prioritaire
`+ -`	`* // % /`	`**`

- En cas de doute, on peut utiliser des parenthèses inutiles.

SHELL
```
>>> 5 - 8 + 4 * 2 ** 3 # Attention à la priorité !
29
>>> (5 - 8) + (4 * (2 ** 3))
29
```

- Remarque : associativité à gauche pour les opérations de même priorité.

SHELL
```
>>> 1-2+5
4
>>> (1-2)+5
4
>>>
```

- Ordre de priorité des opérateurs.

moins prioritaire	plus prioritaire	très prioritaire
`+ -`	`* // % /`	`**`

- En cas de doute, on peut utiliser des parenthèses inutiles.

SHELL
```
>>> 5 - 8 + 4 * 2 ** 3 # Attention à la priorité !
29
>>> (5 - 8) + (4 * (2 ** 3))
29
```

- Remarque : associativité à gauche pour les opérations de même priorité.

SHELL
```
>>> 1-2+5
4
>>> (1-2)+5
4
>>> 1-(2+5)
```

- Ordre de priorité des opérateurs.

moins prioritaire	plus prioritaire	très prioritaire
`+ -`	`* // % /`	`**`

- En cas de doute, on peut utiliser des parenthèses inutiles.

SHELL
```
>>> 5 - 8 + 4 * 2 ** 3 # Attention à la priorité !
29
>>> (5 - 8) + (4 * (2 ** 3))
29
```

- Remarque : associativité à gauche pour les opérations de même priorité.

SHELL
```
>>> 1-2+5
4
>>> (1-2)+5
4
>>> 1-(2+5)
-6
```

- La taille des entiers n'est limitée que par la mémoire de la machine.

```
>>>
```
SHELL

- La taille des entiers n'est limitée que par la mémoire de la machine.

SHELL
```
>>> 87412192161147838437159141 9 ** 10
```

- La taille des entiers n'est limitée que par la mémoire de la machine.

SHELL
```
>>> 87412192161147838437159141 9 ** 10
260447454985660585245180084757921014509252146181223003455270
044550605023694309556482234857745408080127776885578607664767
325495386096105286268494775055260671252317663749619931275002
342815836733596266389781703659821356427940431697254607426485
624871372306773584664397463801
```

- Ce n'est pas le cas dans la plupart des autres langages

- La taille des entiers n'est limitée que par la mémoire de la machine.

SHELL
```
>>> 87412192161147838437159141 9 ** 10
260447454985660585245180084757921014509252146181223003455270
044550605023694309556482234857745408080127776885578607664767
325495386096105286268494775055260671252317663749619931275002
342815836733596266389781703659821356427940431697254607426485
624871372306773584664397463801
```

- Ce n'est pas le cas dans la plupart des autres langages

- Cette propriété est intéressante pour les problèmes de cryptologie où l'on manipule de grands nombres entiers.

- La taille des entiers n'est limitée que par la mémoire de la machine.

SHELL
```
>>> 87412192161147838437159141 9 ** 10
260447454985660585245180084757921014509252146181223003455270
044550605023694309556482234857745408080127776885578607664767
325495386096105286268494775055260671252317663749619931275002
342815836733596266389781703659821356427940431697254607426485
624871372306773584664397463801
```

- Ce n'est pas le cas dans la plupart des autres langages

- Cette propriété est intéressante pour les problèmes de cryptologie où l'on manipule de grands nombres entiers.

- Ne marche pas avec les nombres flottants.

SHELL
```
>>>
```

- La taille des entiers n'est limitée que par la mémoire de la machine.

SHELL
```
>>> 87412192161147838437159141 ** 10
260447454985660585245180084757921014509252146181223003455270
044550605023694309556482234857745408080127776885578607664767
325495386096105286268494775055260671252317663749619931275002
342815836733596266389781703659821356427940431697254607426485
624871372306773584664397463801
```

- Ce n'est pas le cas dans la plupart des autres langages
- Cette propriété est intéressante pour les problèmes de cryptologie où l'on manipule de grands nombres entiers.
- Ne marche pas avec les nombres flottants.

SHELL
```
>>> 87412192161147838437159141.9 ** 10
```

- Ce qui signifie $2,6 \times 10^{259}$

- La taille des entiers n'est limitée que par la mémoire de la machine.

SHELL
```
>>> 87412192161147838437159141 ** 10
260447454985660585245180084757921014509252146181223003455270
044550605023694309556482234857745408080127776885578607664767
325495386096105286268494775055260671252317663749619931275002
342815836733596266389781703659821356427940431697254607426485
624871372306773584664397463801
```

- Ce n'est pas le cas dans la plupart des autres langages
- Cette propriété est intéressante pour les problèmes de cryptologie où l'on manipule de grands nombres entiers.
- Ne marche pas avec les nombres flottants.

SHELL
```
>>> 87412192161147838437159141.9 ** 10
2.6044745498566053e+259
```

- Ce qui signifie $2,6 \times 10^{259}$

Sommaire

- En informatique on fait traditionnellement la différence entre :
  - Un caractère (un symbole) : `'B'`
  - Une chaîne de caractères (une suite de symbole) : `"Baobab"`
  - En anglais on parle de *string (of char)*

- En informatique on fait traditionnellement la différence entre :
  - Un caractère (un symbole) : `'B'`
  - Une chaîne de caractères (une suite de symbole) : `"Baobab"`
  - En anglais on parle de *string (of char)*
- En Python un caractère est une chaîne de longueur 1.

- En informatique on fait traditionnellement la différence entre :
  - Un caractère (un symbole) : `'B'`
  - Une chaîne de caractères (une suite de symbole) : `"Baobab"`
  - En anglais on parle de *string (of char)*
- En Python un caractère est une chaîne de longueur 1.
- Une chaîne de caractère peut être notée :
  - Avec des guillemets simples : `'Ceci est une chaîne'`
  - Avec des guillemets doubles : `"Ceci est une autre chaîne"`
  - Les deux notations sont équivalentes

SHELL

```
>>>
```

- En informatique on fait traditionnellement la différence entre :
  - Un caractère (un symbole) : `'B'`
  - Une chaîne de caractères (une suite de symbole) : `"Baobab"`
  - En anglais on parle de *string (of char)*
- En Python un caractère est une chaîne de longueur 1.
- Une chaîne de caractère peut être notée :
  - Avec des guillemets simples : `'Ceci est une chaîne'`
  - Avec des guillemets doubles : `"Ceci est une autre chaîne"`
  - Les deux notations sont équivalentes

SHELL
```
>>> "Salut" == 'Salut'
```

- En informatique on fait traditionnellement la différence entre :
  - Un caractère (un symbole) : `'B'`
  - Une chaîne de caractères (une suite de symbole) : `"Baobab"`
  - En anglais on parle de *string (of char)*
- En Python un caractère est une chaîne de longueur 1.
- Une chaîne de caractère peut être notée :
  - Avec des guillemets simples : `'Ceci est une chaîne'`
  - Avec des guillemets doubles : `"Ceci est une autre chaîne"`
  - Les deux notations sont équivalentes

SHELL
```
>>> "Salut" == 'Salut'
True
```

Opérations sur les chaînes

```shell
>>> "0+0" + '=' + 'θττ'
```

- Il existe une opération sur les chaînes de caractère : la concatenation

SHELL
```
>>> "0+0" + '=' + 'θττ'
'0+0=θττ'
```

- Il existe une opération sur les chaînes de caractère : la concatenation

Opérations sur les chaînes

SHELL

```
>>> "0+0" + '=' + 'θττ'
'0+0=θττ'
```

- Il existe une opération sur les chaînes de caractère : la concatenation
  - ce n'est pas l'addition
  - cette opération est associative
  - mais elle n'est pas commutative (`"olivier" != "voilier"`)

Opérations sur les chaînes

```
>>> "0+0" + '=' + 'θττ'
'0+0=θττ'
```
SHELL

- Il existe une opération sur les chaînes de caractère : la concatenation
  - ce n'est pas l'addition
  - cette opération est associative
  - mais elle n'est pas commutative (`"olivier" != "voilier"`)

- On peut multiplier une chaîne par un entier

```
>>>
```
SHELL

Opérations sur les chaînes

```
>>> "0+0" + '=' + 'θττ'
'0+0=θττ'
```
SHELL

- Il existe une opération sur les chaînes de caractère : la concatenation
  - ce n'est pas l'addition
  - cette opération est associative
  - mais elle n'est pas commutative (`"olivier" != "voilier"`)

- On peut multiplier une chaîne par un entier

```
>>> "5" + "5" + "5"
```
SHELL

```
>>> "0+0" + '=' + 'θττ'
'0+0=θττ'
```
SHELL

- Il existe une opération sur les chaînes de caractère : la concatenation
  - ce n'est pas l'addition
  - cette opération est associative
  - mais elle n'est pas commutative (`"olivier" != "voilier"`)

- On peut multiplier une chaîne par un entier

```
>>> "5" + "5" + "5"
'555'
>>>
```
SHELL

Opérations sur les chaînes

```
>>> "0+0" + '=' + 'θττ'
'0+0=θττ'
```
SHELL

- Il existe une opération sur les chaînes de caractère : la concatenation
  - ce n'est pas l'addition
  - cette opération est associative
  - mais elle n'est pas commutative (`"olivier" != "voilier"`)

- On peut multiplier une chaîne par un entier

```
>>> "5" + "5" + "5"
'555'
>>> 3 * "5"
```
SHELL

```
>>> "0+0" + '=' + 'θττ'
'0+0=θττ'
```
SHELL

- Il existe une opération sur les chaînes de caractère : la concatenation
  - ce n'est pas l'addition
  - cette opération est associative
  - mais elle n'est pas commutative (`"olivier" != "voilier"`)

- On peut multiplier une chaîne par un entier

```
>>> "5" + "5" + "5"
'555'
>>> 3 * "5"
'555'
>>>
```
SHELL

```
>>> "0+0" + '=' + 'θττ'
'0+0=θττ'
```
SHELL

- Il existe une opération sur les chaînes de caractère : la concatenation
  - ce n'est pas l'addition
  - cette opération est associative
  - mais elle n'est pas commutative (`"olivier" != "voilier"`)

- On peut multiplier une chaîne par un entier

```
>>> "5" + "5" + "5"
'555'
>>> 3 * "5"
'555'
>>> 'A' + 20*'a' + 'h' + '!'*5
```
SHELL

```shell
>>> "0+0" + '=' + 'θττ'
'0+0=θττ'
```

- Il existe une opération sur les chaînes de caractère : la concatenation
  - ce n'est pas l'addition
  - cette opération est associative
  - mais elle n'est pas commutative (`"olivier" != "voilier"`)

- On peut multiplier une chaîne par un entier

```shell
>>> "5" + "5" + "5"
'555'
>>> 3 * "5"
'555'
>>> 'A' + 20*'a' + 'h' + '!'*5
'Aaaaaaaaaaaaaaaaaaaah!!!!!'
```

Sommaire

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>>
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>>
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>>
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs `True` et `False` s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs `True` et `False` s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

Il existe d'autres opérateurs de comparaison : <, >, <=, >= et !=.

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs `True` et `False` s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

Il existe d'autres opérateurs de comparaison : <, >, <=, >= et !=.

SHELL
```
>>>
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs `True` et `False` s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

Il existe d'autres opérateurs de comparaison : <, >, <=, >= et !=.

SHELL
```
>>> 1<2
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs `True` et `False` s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

Il existe d'autres opérateurs de comparaison : <, >, <=, >= et !=.

SHELL
```
>>> 1<2
True
>>>
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs `True` et `False` s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

Il existe d'autres opérateurs de comparaison : <, >, <=, >= et !=.

SHELL
```
>>> 1<2
True
>>> 1+1 >= 2 # 1+1 ≥ 2
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs `True` et `False` s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

Il existe d'autres opérateurs de comparaison : <, >, <=, >= et !=.

SHELL
```
>>> 1<2
True
>>> 1+1 >= 2 # 1+1 ≥ 2
True
>>>
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs `True` et `False` s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

Il existe d'autres opérateurs de comparaison : <, >, <=, >= et !=.

SHELL
```
>>> 1<2
True
>>> 1+1 >= 2 # 1+1 ≥ 2
True
>>> 1+1 != 2 #  a ≠ 2
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs `True` et `False` s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

Il existe d'autres opérateurs de comparaison : <, >, <=, >= et !=.

SHELL
```
>>> 1<2
True
>>> 1+1 >= 2 # 1+1 ≥ 2
True
>>> 1+1 != 2 #  a ≠ 2
False
```

SHELL
```
>>>
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs `True` et `False` s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

Il existe d'autres opérateurs de comparaison : <, >, <=, >= et !=.

SHELL
```
>>> 1<2
True
>>> 1+1 >= 2 # 1+1 ≥ 2
True
>>> 1+1 != 2 #  a ≠ 2
False
```

SHELL
```
>>> 2>0
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs `True` et `False` s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

Il existe d'autres opérateurs de comparaison : <, >, <=, >= et !=.

SHELL
```
>>> 1<2
True
>>> 1+1 >= 2 # 1+1 ≥ 2
True
>>> 1+1 != 2 #  a ≠ 2
False
```

SHELL
```
>>> 2>0
True
>>>
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs `True` et `False` s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

Il existe d'autres opérateurs de comparaison : <, >, <=, >= et !=.

SHELL
```
>>> 1<2
True
>>> 1+1 >= 2 # 1+1 ≥ 2
True
>>> 1+1 != 2 #  a ≠ 2
False
```

SHELL
```
>>> 2>0
True
>>> 1+1<=3   # 1+1 ≤ 3
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs `True` et `False` s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

Il existe d'autres opérateurs de comparaison : <, >, <=, >= et !=.

SHELL
```
>>> 1<2
True
>>> 1+1 >= 2 # 1+1 ≥ 2
True
>>> 1+1 != 2 #  a ≠ 2
False
```

SHELL
```
>>> 2>0
True
>>> 1+1<=3   # 1+1 ≤ 3
True
>>>
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs True et False s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

Il existe d'autres opérateurs de comparaison : <, >, <=, >= et !=.

SHELL
```
>>> 1<2
True
>>> 1+1 >= 2 # 1+1 ≥ 2
True
>>> 1+1 != 2 #  a ≠ 2
False
```

SHELL
```
>>> 2>0
True
>>> 1+1<=3   # 1+1 ≤ 3
True
>>> (1+1>3)==False # F = F
```

En comparant des valeurs entre elles on obtient des booleéans.

- Pour tester l'égalité on utilise l'opérateur ==
- Le symbole = correspond à l'affectation.

SHELL
```
>>> 2+2==4
True
>>> "2" + "2" == "22"
True
>>> True == False
False
```

Les valeurs True et False s'appellent des booléens (George Boole, Royaume Unie, 1815-1864)

Il existe d'autres opérateurs de comparaison : <, >, <=, >= et !=.

SHELL
```
>>> 1<2
True
>>> 1+1 >= 2 # 1+1 ≥ 2
True
>>> 1+1 != 2 #  a ≠ 2
False
```

SHELL
```
>>> 2>0
True
>>> 1+1<=3   # 1+1 ≤ 3
True
>>> (1+1>3)==False # F = F
True
```

Il y a trois opérateurs booléans :

-
-
-

p	q	p and q	p or q
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Il y a trois opérateurs booléans : négation

- La négation s'écrit `not`
-
-

p	q	p and q	p or q
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Il y a trois opérateurs booléans : négation ,« et logique »

- La négation s'écrit `not`
- `p and q` est vrai ⇔ p et q sont tous les deux vrais.
-

p	q	p and q	p or q
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Il y a trois opérateurs booléans : négation ,« et logique » , « ou logique »

- La négation s'écrit `not`
- `p and q` est vrai ⇔ p et q sont tous les deux vrais.
- `p or q` est faux ⇔ p et q sont tous les deux faux.

p	q	p and q	p or q
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Il y a trois opérateurs booléans : négation ,« et logique » , « ou logique »

- La négation s'écrit `not`
- `p and q` est vrai ⇔ p et q sont tous les deux vrais.
- `p or q` est faux ⇔ p et q sont tous les deux faux.

p	q	p and q	p or q
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

SHELL

```
>>>
```

Il y a trois opérateurs booléans : négation ,« et logique » , « ou logique »

- La négation s'écrit `not`
- `p and q` est vrai ⇔ p et q sont tous les deux vrais.
- `p or q` est faux ⇔ p et q sont tous les deux faux.

p	q	p and q	p or q
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

SHELL

```
>>> not True
```

Il y a trois opérateurs booléans : négation ,« et logique » , « ou logique »

- La négation s'écrit `not`
- `p and q` est vrai ⇔ p et q sont tous les deux vrais.
- `p or q` est faux ⇔ p et q sont tous les deux faux.

p	q	p and q	p or q
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

SHELL

```
>>> not True
False
>>>
```

Il y a trois opérateurs booléans : négation ,« et logique » , « ou logique »

- La négation s'écrit `not`
- `p and q` est vrai ⇔ p et q sont tous les deux vrais.
- `p or q` est faux ⇔ p et q sont tous les deux faux.

p	q	p and q	p or q
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

SHELL

```
>>> not True
False
>>> (3>1) or (6>2)   # True or True
```

Il y a trois opérateurs booléans : négation ,« et logique » , « ou logique »

- La négation s'écrit `not`
- `p and q` est vrai ⇔ p et q sont tous les deux vrais.
- `p or q` est faux ⇔ p et q sont tous les deux faux.

p	q	p and q	p or q
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

SHELL

```
>>> not True
False
>>> (3>1) or (6>2)   # True or True
True
>>>
```

Il y a trois opérateurs booléans : négation ,« et logique » , « ou logique »

- La négation s'écrit `not`
- `p and q` est vrai ⇔ p et q sont tous les deux vrais.
- `p or q` est faux ⇔ p et q sont tous les deux faux.

p	q	p and q	p or q
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

SHELL

```
>>> not True
False
>>> (3>1) or (6>2)  # True or True
True
>>> True and not (1+1==2)  # True and False
```

Il y a trois opérateurs booléans : négation ,« et logique » , « ou logique »

- La négation s'écrit `not`
- `p and q` est vrai ⇔ p et q sont tous les deux vrais.
- `p or q` est faux ⇔ p et q sont tous les deux faux.

p	q	p and q	p or q
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

SHELL

```
>>> not True
False
>>> (3>1) or (6>2)  # True or True
True
>>> True and not (1+1==2)  # True and False
False
```

Il y a trois opérateurs booléans : négation ,« et logique » , « ou logique »

- La négation s'écrit `not`
- `p and q` est vrai ⇔ p et q sont tous les deux vrais.
- `p or q` est faux ⇔ p et q sont tous les deux faux.

p	q	p and q	p or q
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

SHELL

```
>>> not True
False
>>> (3>1) or (6>2)  # True or True
True
>>> True and not (1+1==2)  # True and False
False
```

Attention : le « ou logique » n'a pas le même sens qu'en français :

Mange ta soupe ou tu t'en prendras une !

- Lors d'un calcul booléen, les deux termes ne sont pas forcément calculés.

On parle d'évaluation paresseuse.

SHELL
```
>>>
```

- Lors d'un calcul booléen, les deux termes ne sont pas forcément calculés.

On parle d'évaluation paresseuse.

SHELL
```
>>> 3%0 == 0
```

- Lors d'un calcul booléen, les deux termes ne sont pas forcément calculés.

On parle d'évaluation paresseuse.

SHELL
```
>>> 3%0 == 0
Traceback (most recent call last):
  File "<console>", line 1, in <module>
ZeroDivisionError:  integer modulo by zero
>>>
```

- Lors d'un calcul booléen, les deux termes ne sont pas forcément calculés.

On parle d'évaluation paresseuse.

SHELL
```
>>> 3%0 == 0
Traceback (most recent call last):
  File "<console>", line 1, in <module>
ZeroDivisionError:  integer modulo by zero
>>> (1>0) or (3%0 == 0)
```

- Lors d'un calcul booléen, les deux termes ne sont pas forcément calculés.

On parle d'évaluation paresseuse.

SHELL
```
>>> 3%0 == 0
Traceback (most recent call last):
  File "<console>", line 1, in <module>
ZeroDivisionError:  integer modulo by zero
>>> (1>0) or (3%0 == 0)
True
```

- Lors d'un calcul booléen, les deux termes ne sont pas forcément calculés.

On parle d'évaluation paresseuse.

SHELL
```
>>> 3%0 == 0
Traceback (most recent call last):
  File "<console>", line 1, in <module>
ZeroDivisionError: integer modulo by zero
>>> (1>0) or (3%0 == 0)
True
```

- `(3%0 == 0)` n'est pas évalué car « `True or` … » est toujours vrai.

- Lors d'un calcul booléen, les deux termes ne sont pas forcément calculés.

On parle d'évaluation paresseuse.

SHELL
```
>>> 3%0 == 0
Traceback (most recent call last):
  File "<console>", line 1, in <module>
ZeroDivisionError: integer modulo by zero
>>> (1>0) or (3%0 == 0)
True
```

- `(3%0 == 0)` n'est pas évalué car « `True or` … » est toujours vrai.
- En pratique, on peut faire la même chose en mathématique.

$$0 \times \int_{log(2)}^{7\pi+3} \frac{3e^{4x\pi}}{\sqrt{x+\sqrt{x}} \cdot cos(18)} \sum_{n=0}^{\infty} \frac{1}{n^2} dx = \ ?$$

On peut voir le `and` et le `or` comme deux opérations.

- Commutativité : `a and b == b and a`
- Associativité : `(a or b) or c == a or (b or c)`
- Distributivité : `a or (b and c) == (a or b) and (a or c)`
- Distributivité : `a and (b or c) == (a and b) or (b and c)`

On peut voir le `and` et le `or` comme deux opérations.

- Commutativité : `a and b == b and a`
- Associativité : `(a or b) or c == a or (b or c)`
- Distributivité : `a or (b and c) == (a or b) and (a or c)`
- Distributivité : `a and (b or c) == (a and b) or (b and c)`

Lois de Morgan

- `not (a and b) == (not a) or (not b)`
- `not (a or b) == (not a) and (not b)`

Sommaire

- Pour définir la fonction $f : n \mapsto 2n + 1$.

SHELL
```
>>>
```

- Pour définir la fonction $f : n \mapsto 2n + 1$.

SHELL
```
>>> def f(n):

```

Définir sa propre fonction

- Pour définir la fonction $f : n \mapsto 2n+1$.

SHELL
```
>>> def f(n):
...     return 2*n+1 # Attention : indentation
```

- Le contenu de la fonction doit être indenté (tabulation)
- le mot-clef `return` définit le résultat de la fonction.

Définir sa propre fonction

- Pour définir la fonction $f : n \mapsto 2n+1$.

SHELL
```
>>> def f(n):
...     return 2*n+1 # Attention : indentation
>>>
```

- Le contenu de la fonction doit être indenté (tabulation)
- le mot-clef `return` définit le résultat de la fonction.

- Pour définir la fonction $f : n \mapsto 2n+1$.

SHELL
```
>>> def f(n):
...     return 2*n+1 # Attention : indentation
>>> f(3)
```

- Le contenu de la fonction doit être indenté (tabulation)
- le mot-clef `return` définit le résultat de la fonction.

- Pour définir la fonction $f : n \mapsto 2n+1$.

SHELL
```
>>> def f(n):
...     return 2*n+1 # Attention : indentation
>>> f(3)
7
```

- Le contenu de la fonction doit être indenté (tabulation)
- le mot-clef `return` définit le résultat de la fonction.

SHELL
```
>>>
```

- Pour définir la fonction $f : n \mapsto 2n+1$.

SHELL
```
>>> def f(n):
...     return 2*n+1 # Attention : indentation
>>> f(3)
7
```

- Le contenu de la fonction doit être indenté (tabulation)
- le mot-clef `return` définit le résultat de la fonction.

SHELL
```
>>> f(2.5) # n n'est pas forcément entier
```

- Pour définir la fonction $f : n \mapsto 2n+1$.

SHELL
```
>>> def f(n):
...     return 2*n+1 # Attention : indentation
>>> f(3)
7
```

- Le contenu de la fonction doit être indenté (tabulation)
- le mot-clef `return` définit le résultat de la fonction.

SHELL
```
>>> f(2.5) # n n'est pas forcément entier
6.0
>>>
```

- Pour définir la fonction $f : n \mapsto 2n+1$.

SHELL
```
>>> def f(n):
...     return 2*n+1 # Attention : indentation
>>> f(3)
7
```

- Le contenu de la fonction doit être indenté (tabulation)
- le mot-clef `return` définit le résultat de la fonction.

SHELL
```
>>> f(2.5) # n n'est pas forcément entier
6.0
>>> f(2+2j) # j correspond au i des mathématiques
```

Définir sa propre fonction

- Pour définir la fonction $f : n \mapsto 2n+1$.

SHELL
```
>>> def f(n):
...     return 2*n+1 # Attention : indentation
>>> f(3)
7
```

- Le contenu de la fonction doit être indenté (tabulation)
- le mot-clef `return` définit le résultat de la fonction.

SHELL
```
>>> f(2.5) # n n'est pas forcément entier
6.0
>>> f(2+2j) # j correspond au i des mathématiques
(5+4j)
>>>
```

- Pour définir la fonction $f : n \mapsto 2n+1$.

SHELL
```
>>> def f(n):
...     return 2*n+1 # Attention : indentation
>>> f(3)
7
```

- Le contenu de la fonction doit être indenté (tabulation)
- le mot-clef `return` définit le résultat de la fonction.

SHELL
```
>>> f(2.5) # n n'est pas forcément entier
6.0
>>> f(2+2j) # j correspond au i des mathématiques
(5+4j)
>>> f('Deux') #  le type doit être cohérent
```

- Pour définir la fonction $f : n \mapsto 2n+1$.

SHELL
```
>>> def f(n):
...     return 2*n+1 # Attention : indentation
>>> f(3)
7
```

- Le contenu de la fonction doit être indenté (tabulation)
- le mot-clef `return` définit le résultat de la fonction.

SHELL
```
>>> f(2.5) # n n'est pas forcément entier
6.0
>>> f(2+2j) # j correspond au i des mathématiques
(5+4j)
>>> f('Deux') #  le type doit être cohérent
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "<console>", line 1, in f
TypeError:  can only concatenate str (not "int") to str
```

Écrire les fonctions suivantes

- Une fonction prenant trois notes et renvoyant la moyenne
- Une fonction prenant trois notes et renvoyant `True` si la moyenne est supérieure à 10.
- Une fonction prenant deux flottants et renvoyant `True` s'ils sont presques égaux (à 10^{-2} près)

Merci pour votre attention
Questions

?

Cours 0 — Expressions, écriture binaire et logique

Licence