AGRO CAMPUS OUEST

Algorithmique pour l'informatique

et

Introduction à Python

Algorithmique pour l'informatique

- Algorithmique pour des tâches de traitement de l'information
- Algorithmique pour les ordinateurs
  - L'ordinateur est notre robot
    - C'est un exécutant qu'on va commander
  - « Faire faire » à un ordinateur
- Qu'est ce qu'on peut faire avec ?
  - Qu'est ce qu'il peut traiter comme information ?
  - Quelles sont les instructions élémentaires qu'il propose ?
  - Qu'est ce qu'il peut donner en retour ?

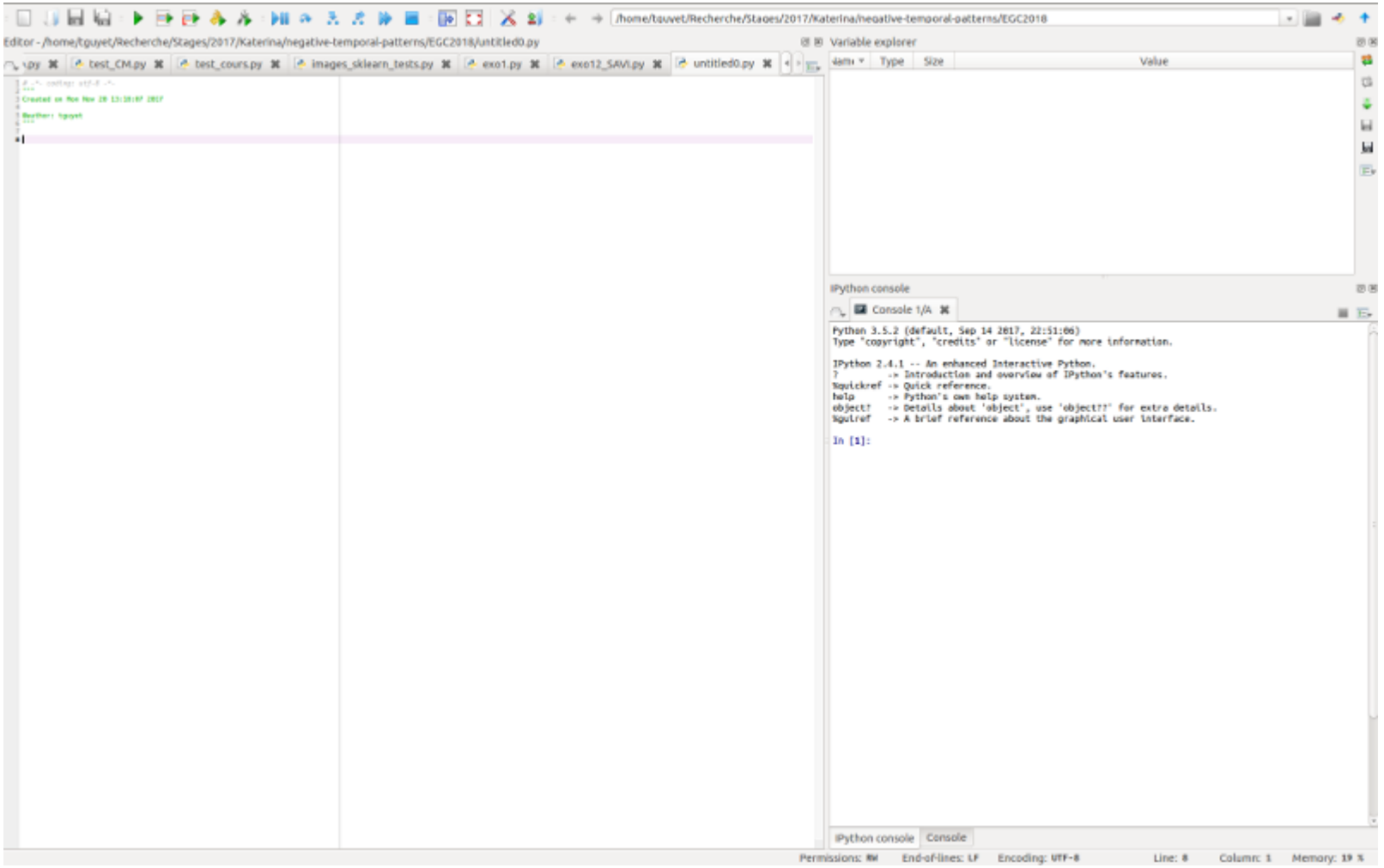
Python : un langage de programmation

- Un langage se définit par :
  - Vocabulaire (mots ayant un sens dans ce langage...)
  - Syntaxe (règles d'agencement des unités lexicales)
  - Sémantique (signification des instructions)
- Le langage Python
  - Python est un langage simple mais puissant
  - Python est un langage open source et multi-paradigmes (script, programmation objet, impérative...)
  - Il a été développé en 1993 par Guido Von Rossum
  - Parmis les plus populaires

Python : un langage de programmation

- Un langage : c'est aussi un environnement de programmation
- Environnement de travail
  - Python 3
  - Utilisation d'Anaconda (environnement d'installation et de gestion des options de Python)
  - Utilisation d'un IDE (Environnement intéractif de développement)
    - Intégration script et console
    - Coloration syntaxique
    - Complétion
    - Outils de déboggage
    - Aide en ligne très riche (nommée Google!)

Spyder 3 – courte démo

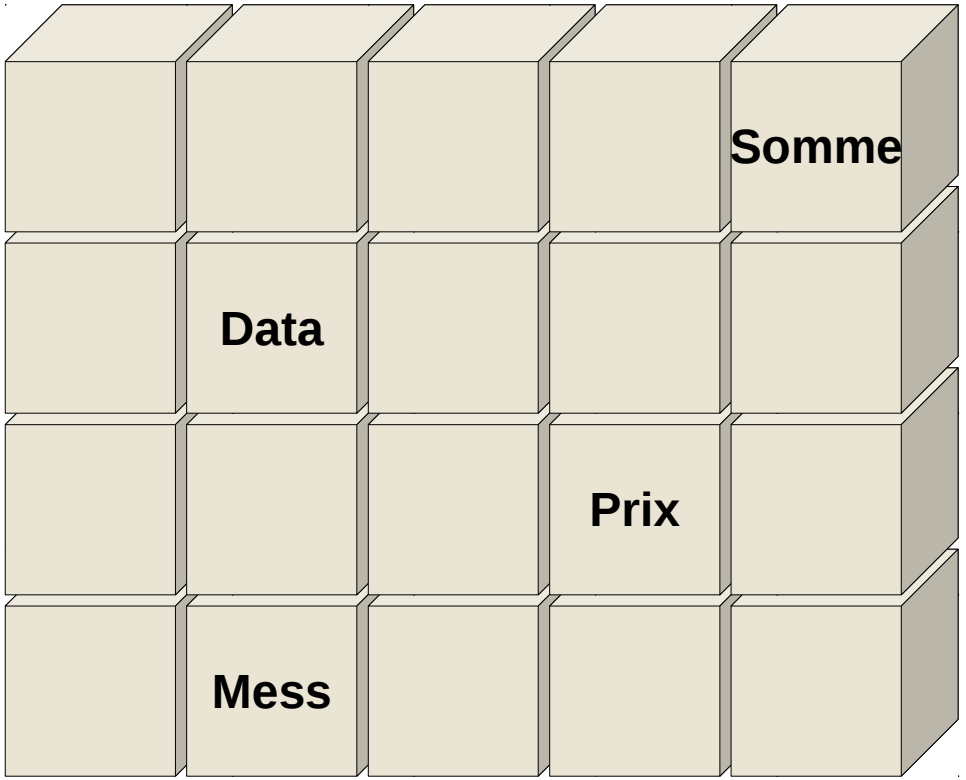


Description du robot-ordinateur

1) Ordinateur gestionnaire de casiers
    1) Présentation de la mémoire
    2) Déclaration et affectation de variables
    3) Opérateurs et conditions
    4) Communication avec l'extérieur
2) Tableaux d'évolution de variables

Image du fonctionnement de la mémoire

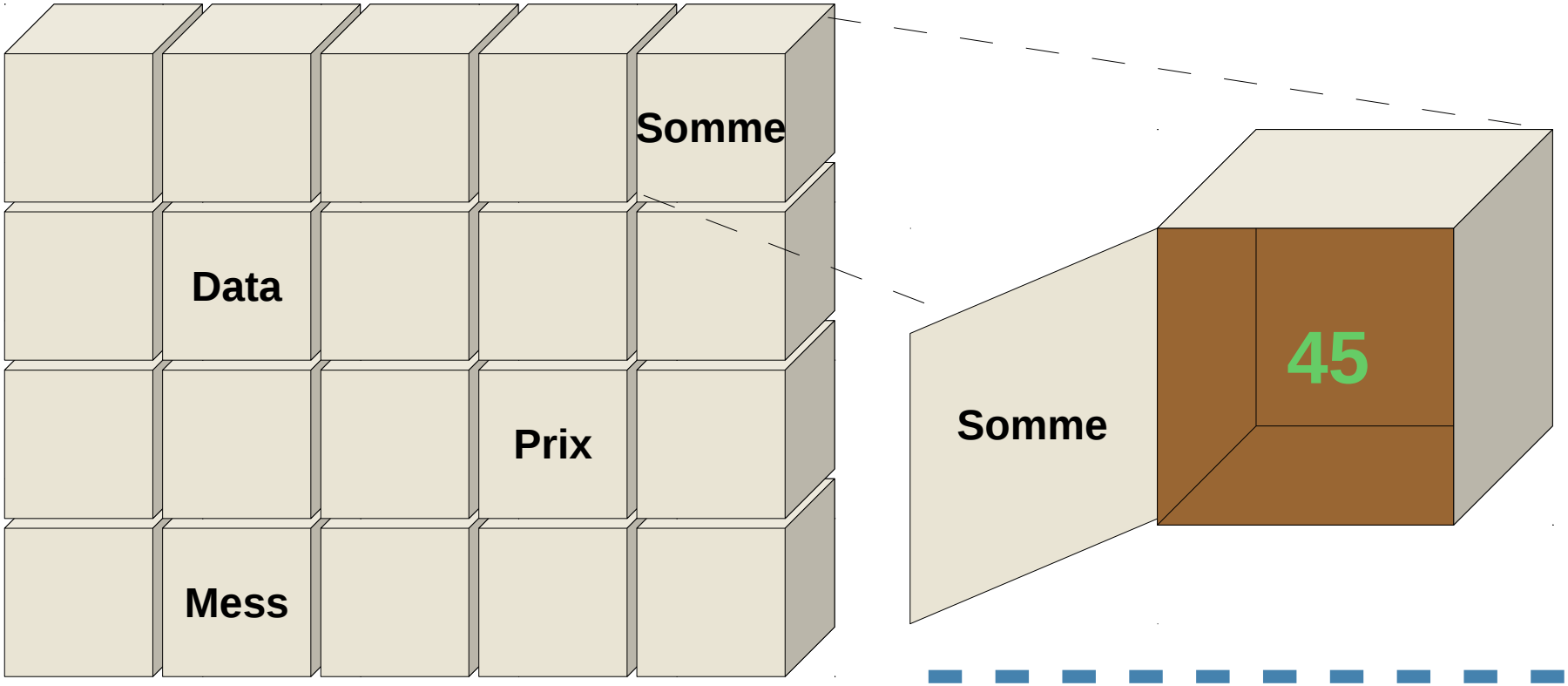
- Figuration de la mémoire de l'ordinateur : casiers
  - 1 casier = 1 **variable**



- L'ordinateur gère une multitude de casiers
- Les casiers utiles sont ceux qui portent un **nom**.
- Chaque nom désigne une **variable**.

Image du fonctionnement de la mémoire

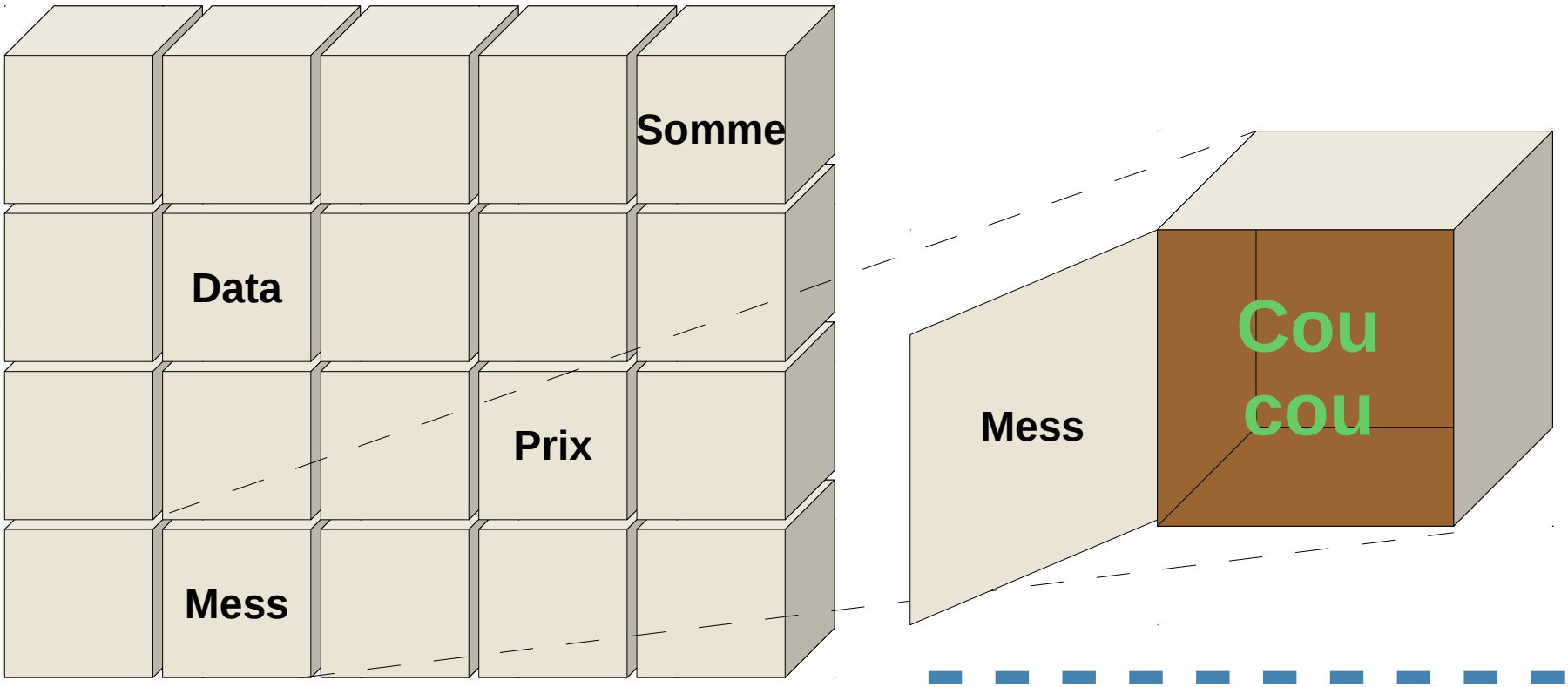
- Figuration de la mémoire de l'ordinateur : casiers



Une variable contient une **valeur**

Image du fonctionnement de la mémoire

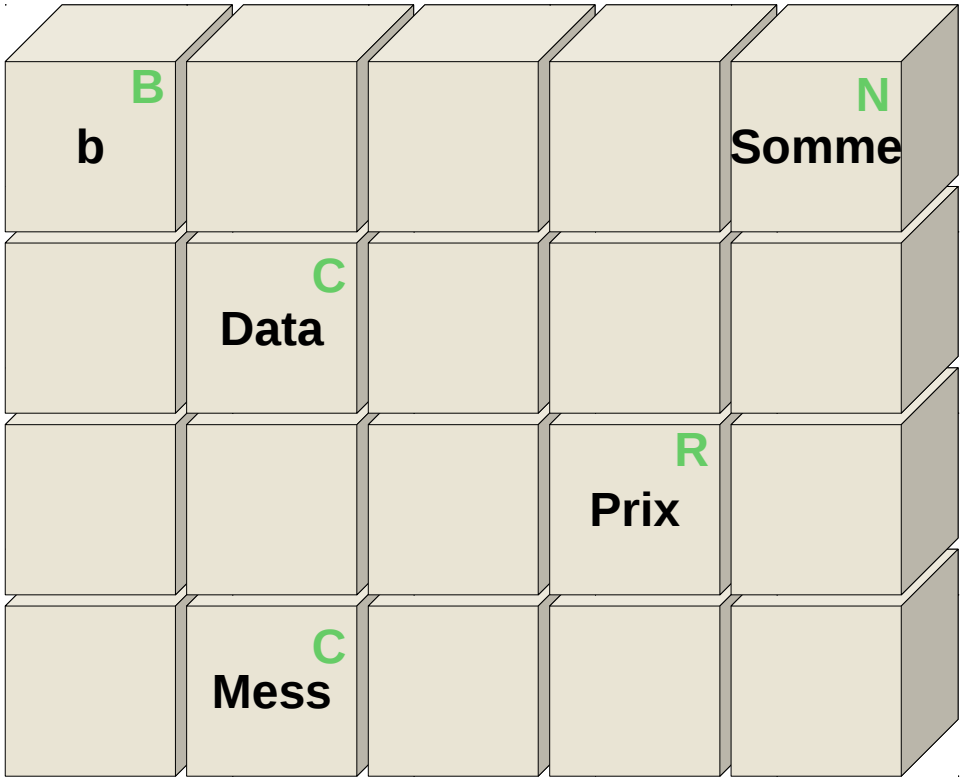
- Figuration de la mémoire de l'ordinateur : casiers



Une variable contient une **valeur**

Image du fonctionnement de la mémoire

- Figuration de la mémoire de l'ordinateur : casiers
  - Type d'une variable



Le **type** de variable indique ce qu'elle peut contenir comme valeur

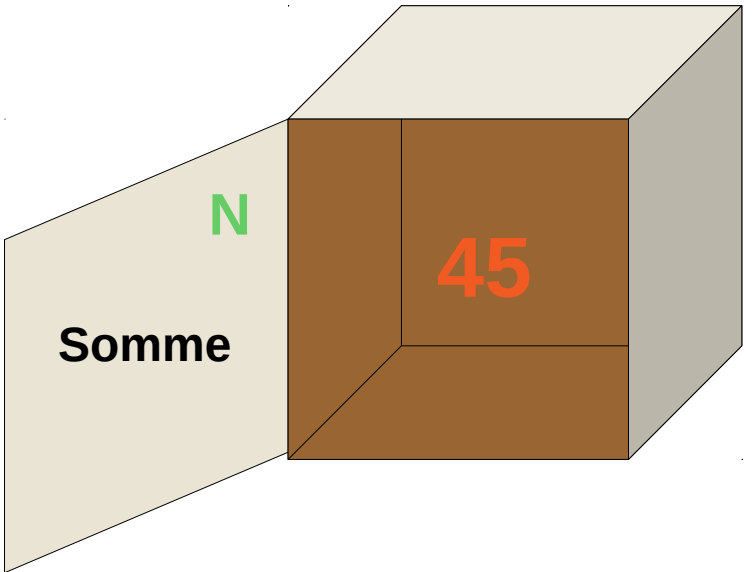
- un booléen

- un nombre (entier ou réel)

- une chaîne de caractères

Image du fonctionnement de la mémoire

- Caractéristiques d'une variable (un casier)
  - **Son nom**
    - Inamovible, choisi par le programmeur
  - **Son type**
    - Inchangeable, choisi par le programmeur
    - parmi : *booléen, entier*, *réel*, *chaîne de caractères*
  - **Son contenu** : ce qui est manipulé par l'ordinateur
    - une valeur booléenne
    - un nombre
    - une chaîne de caractères



Les types de données en informatique

- Dans un ordinateur, les données sont codées en binaire
  - Le codage des données imposent des contraintes sur les valeurs que peuvent contenir des casiers
  - Le codage binaire est un choix « technologique » plus que « théorique »

- Les grands types de données
  - Les booléens,
  - Les nombres,
  - Les chaînes de caractères

Les types de données en informatique

- Les **booléens** : une valeur binaire
  - Appelés aussi `bool`
  - Prend deux valeurs
    - **`False`** ou **`True`**
    - `0` ou `1`
  - les opérations possibles entre booléens sont les suivantes
    - opérateur ET, noté `&`
    - opérateur OU, noté `|`
    - opérateur NON (`not` en Python), noté parfois `!`

Les types de données en informatique

- Les nombres
  - Les **entiers**, notés également `int`
    - Ils sont limités par des valeurs maximale et minimale
  - Les **flottants** (nombres à virgules), notés `float`
    - Ils sont limités par des valeurs maximale et minimale
    - Ils ont une précision, par exemple, pour un ordinateur :

1.000000000000000 == 0.999999999999

- L'ordinateur sait bien traiter les nombres
  - Additions, soustractions, multiplications, divisions
  - Calcul de puissances, racines carrées ??
    - Il le peut plus ou moins facilement en fonction du langage

Les types de données en informatique

- Les **chaînes de caractères** = suite de caractères
  - noté `str`
  - C'est un « mot », un « texte » ou bien d'autres choses
    - `'bonjour'`
    - `'Ceci est un chaîne de caractères !'`
    - `'dlTTR34  -- 3'`
  - L'utilisation des chaînes de caractères n'est pas naturelle pour un ordinateur
    - pour lui, c'est juste
      - des caractères qui se suivent
      - un caractère est codé par un nombre entier
    - la bonne traduction des caractères est dépendante de conventions (tables de caractères, ex ASCII)
  - Voir cours spécifique

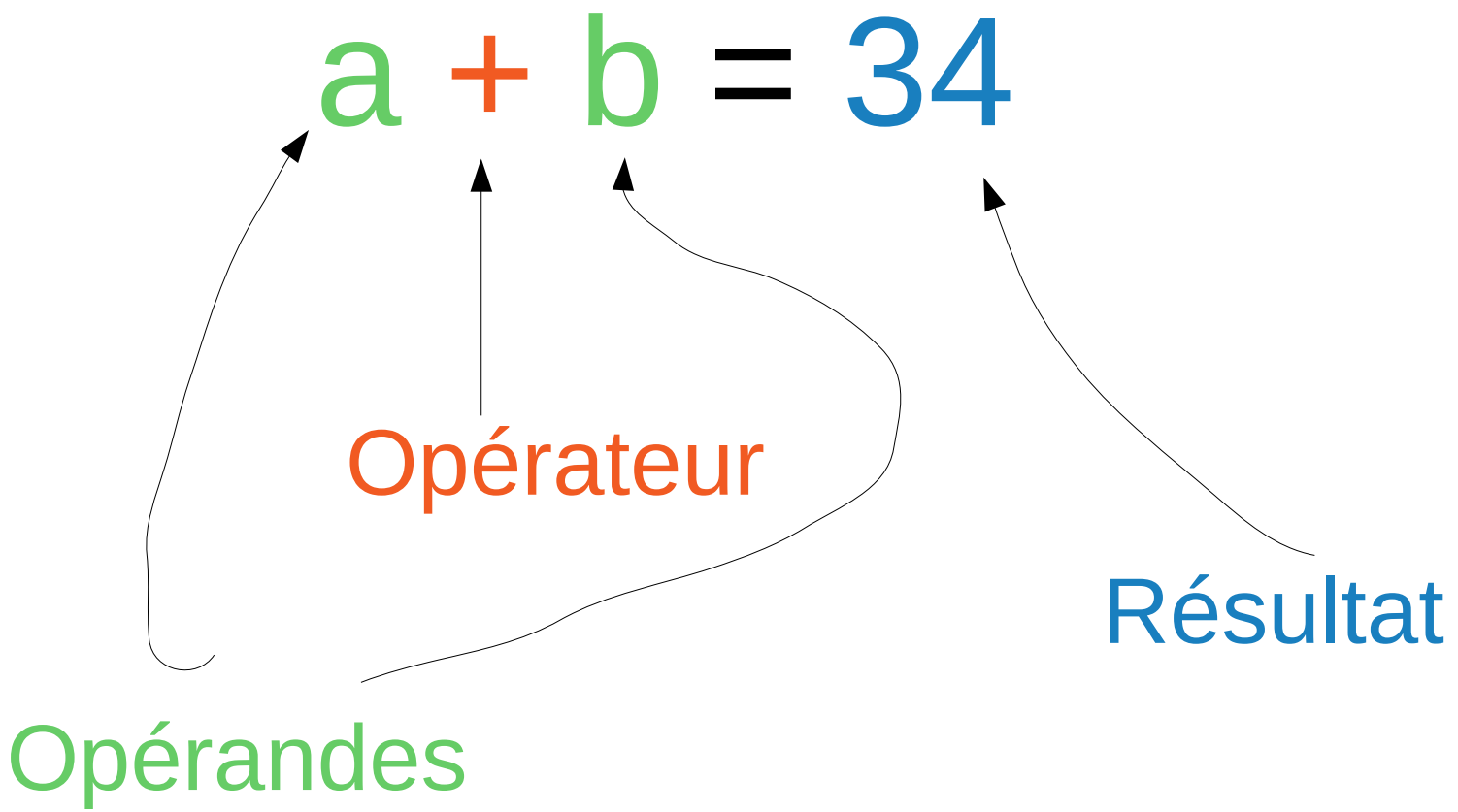
Avec Python

- Fonction type
  - Donne le type d'une expression
- Fonction isinstance
  - Tests le type d'une expression

```
>>> type(1)
>>> type(True)
>>> type(3.45)
>>> type('coucou')
>>> isinstance(3, int)
>>> isinstance(3, str)
>>> isinstance(3, float)
>>> isinstance(3.0,float)
```

Les types de données en informatique

- Rappel de définition



Vérification des types

- Attention : **une opération est *typée***
  - **chaque opérateur ne peut être utilisé qu'avec des opérandes d'un certain type**
  - la signification d'un opérateur peut changer en fonction du type des opérandes
- Exemples
  - On ne peut pas additionner des chaînes de caractères
    - `+, -, *, /` : les opérandes doivent être des nombres
  - On ne peut pas « concaténer » des nombres
    - uniquement chaînes de caractères
  - Les opérateurs booléens sont entre booléens
    - `&, |` et `not` ont pour opérande des booléens

AGRO CAMPUS OUEST

Description du robot-ordinateur

1) Ordinateur gestionnaire de casiers
    1) Présentation de la mémoire
    2) Déclaration et affectation de variable
    3) Opérateurs et conditions
    4) Communication avec l'extérieur
2) Tableaux d'évolution de variables

Déclaration et affectation d'une variable

- Première instruction : **déclaration d'une variable**

Réserve moi un casier !

  - Nécessite l'information sur le **type** et le **nom du casier** !
- Syntaxe en pseudo-code

```
int v
double somme
string T
```

- Déclaration de variable : *à écrire à l'extérieur de l'organigramme*

Déclaration et affectation d'une variable

- Seconde instruction : **affectation d'une variable**

  Met une valeur dans le casier !

- Affectation par **une valeur**
  - Met une valeur explicite dans le casier d'une variable
    - Écrase tout ce qui s'y trouvait
  - Syntaxe en pseudo-code

```
int v
v <- 10
```

  - Attention : vérification du type, il faut que la valeur corresponde au type !

```
int v
v <- 'coucou'
```

Déclaration et affectation d'une variable

- Seconde instruction : **affectation d'une variable**

  Met une valeur dans le casier !

- Affectation par **une autre variable**
  - **Copie** le contenu d'un casier dans un autre
  - Syntaxe en pseudo-code

```
int foo
int bar
foo <- bar
```

  - Attention : vérification du type, il faut que la valeur corresponde au type !

```
int nombre
string texte
nombre <- texte
```

Déclaration et affectation d'une variable

- Seconde instruction : **affectation d'une variable**

  Met une valeur dans le casier !

- Affectation par **le résultat d'une opération**
  - Met le résultat d'une opération dans le casier d'une variable
  - Syntaxe en pseudo-code

```
int foo
int bar
foo <- 2 + bar * 10
```

  - Attention : vérification des types
    - pour les opérations (partie droite de la flèche)
    - pour l'affectation
  - **Les opérations ne sont que dans la partie droite !**

Variable affectée et opérande

- Syntaxe (exemple)

```
int X
X <- 5
X <- X + 3
```

- `X` sert deux fois
  - variable de destination de l'affectation
  - opérande d'une addition
  - cette opération est appelée une **incrémentation**
- L'ordinateur fait
  - **d'abord l'opération à droite d'une affectation,**
  - **puis l'affectation**

```
int X, Y
X <- 4
Y <- -6
X <- (X + 2) * X + Y
```

Avec Python

- L'affectation est faite à l'aide d'un opérateur « = »
- Les variables sont typées
- Mais
  - La déclaration d'une variable se fait lors de son initialisation
  - Le typage est dynamique :
    - il peut changer en fonction de ce qu'on met dans une variable
    - C'est dangereux ! Ne l'utilisez pas !!
- + variable explorer de Sypder

```
>>> x=1
>>> type(x)
int
>>> x='coucou'
>>> type(x)
str
```

Communication avec l'extérieur

- Troisième instruction : **affichage à l'écran**

Affiche à l'écran

- Affiche uniquement du texte *(et pas de dessins !)*
- Syntaxes possibles
  - Affichage d'une constante (nombre ou texte)

```
print(123.4)
print('Bonjour')
```

  - Affichage d'une variable

```
print(CC)
print(x)
```

  - Affichage du résultat d'une opération

```
print(CC1 + CC2)
print(2 * Total)
print(2*'coucou')
```

//Concaténation de deux chaînes

Communication avec l'extérieur

- Quatrième instruction : **lecture au clavier**

Attend une saisie clavier et place le résultat dans une variable

- Syntaxe en pseudo code

```
>>> N = input('donner une valeur')
donner une valeur
```

  - **Attente** : la saisie de l'utilisateur est validée par la touche « `Entrée` »
  - Quel est le type de la variable récupérée ??

Transtypage des variables

- Il est possible de « tenter » de transformer le contenue d'une expression en forcant son type
  - L'usage est assez fréquent en Python
- Par exemple
  - Chaîne de caractère en nombre
    - Utile pour `input()`
  - Nombre en chaîne de caractères
    - Utile pour `print()`
- Chaque type de base est associé à une fonction

```
>>> x=1
>>> type(x)
int
>>> c = str(x)
>>> type(c)
str
>>> type( str(x) )
str
>>> type( float(x) )
float
>>> x = 'coucou'
>>> int(x)
```

Petit résumé des instructions possibles pour un ordinateur

- les opérations entre variables et constantes
  - elles sont typées
- la déclaration de variables
- l'affectation de variables
  - par une constante, une variable ou une opération
  - typée également
- l'affichage
  - d'une constante, d'une variable ou d'une opération
- la lecture au clavier (affectée à une variable)

Mais qu'est ce qu'on fait ??

- Message 1 : il est important de savoir analyser les programme pour savoir quel est la nature des variables et des opérations
- Message 2 : Attention … le typage dynamique rend les choses parfois difficiles
  - Quel est la sémantique du `*` dans cette commande ?

```
>>> print( 2 * x )
```

Les conditions évaluables par l'ordinateur

- L'ordinateur est capable de **comparer une variable** avec
  - une constante
  - une autre variable
  - le résultat d'une opération
- Opérateurs de comparaison
  - Opérateurs : `<`, `>`, `<=`, `>=`, `==`, `!=`
  - Sémantique
    - entre nombres
    - entre chaînes de caractères
- Combinaison des opérateurs
  - and, or, not

```
CC=='quit'
```

```
X>Y
```

```
2*X < 3*Y+2
```

Opérateur de différence

Opérateur d'égalité

Les conditions évaluables par l'ordinateur

- Quelques exemples sur les opérateurs
- Quelques exemples sur les combinaisons d'opérateurs logiques

```
>>> x=1 ; y=4
>>> (x>2) or (y<5)
....
>>> (x>2) and (y<5)
....
>>> not ( (x>2) or (y<5) )
....
>>> not (x>2) and not (y<5)
....
>>> (not (x>2) ) and (not (y<5))
....
```

```
>>> x=1
>>> x>2
....
>>> x==1
....
>>> 4*x >= 2
....
>>> x == "toto"
....
>>> "toto" > "tutu"
....
```

L'ordinateur « comprend » les structures de contrôle

- Structures séquentielles, alternatives et itératives

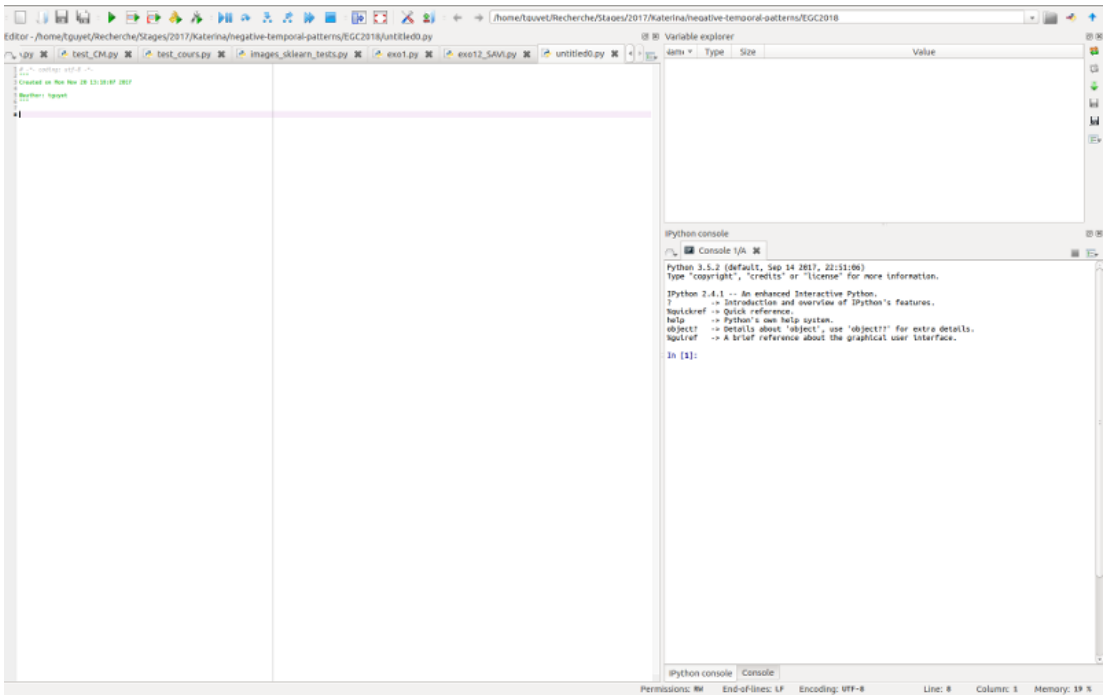
```
double T
print 'Quelle température fait-il ?'
T <- scan

SI T<4 ALORS
|  print 'Prends ton manteau'
SINON
|  print 'Prends ta veste'
└

print 'A plus tard ...'
```

L'ordinateur « comprend » les structures de contrôle

- Vers l'écriture d'un programme
  - La console est utile pour tester rapidement des commandes, explorer les variables
  - Un programme est, quant à lui, réalisé dans un fichier de « script Python »
    - Ecrire les instructions dans le fichier
    - Utiliser la touche F5 pour exécuter (ou icône consacré)



Les structures de contrôle Python

- Séquencement
  - Les lignes successives d'un programme sont exécutées les unes à la suite des autres

```
X = 1
Y = 4
Z = X * Y**2
```

Les structures de contrôle Python

- Alternatives
  - Mot clé : `if`
  - La condition est mise entre le « `if` » et les « : »
  - Le `else` est optionnel
    - Le bloc commence par « : »
    - Possible utilisation du `elif`
  - **L'indentation sert à identifier le bloc d'instructions conditionnel**
    - **Tabulations vs espaces !!**

```
X = float(input('x ??'))
if X >= 1 :
    Y = 4
    Z = X * Y**2
print(Z)
```

```
X = float(input('x ??'))
if X >= 1 :
    Y = 4
    Z = X * Y**2
else:
    Z = X**3
print(Z)
```

```
X = float(input('x ??'))
if X >= 1 :
    Y = 4
    Z = X * Y**2
elif X>=0:
    Z=0
else :
    Z = X**3
print(Z)
```

Les structures de contrôle Python

- Répétitions
  - Mot clé : `while`
  - La condition est mise entre le « `while` » et les « : »
  - **L'indentation sert à identifier le bloc d'instructions répété**
- Exemple
  - Tester différentes valeurs de X !!

```
X = float(input('x ??'))
I = 0
Y = 1
while I <= X :
    Y = 1/(Y+X)
    I += 1
    print( str(I) + ' :' +str(Y) )
print('fin')
```

Les structures de contrôle Python

- Répétitions
  - Mot clé : `for`
- La structure de contrôle `for` fait répéter un bloc d'instructions en faisant varier la valeur d'une variable à chaque « tour de boucle »
- Cette expression se montre très utile en pratique
  - Elle se généralise facilement
- `I` prend successivement toutes les valeurs possibles définies dans range

```
X = float(input('x ??'))
Y = 1
for I in range(0,X) :
    Y = 1/(Y+X)
    print( str(I) + ' :' +str(Y) )

print('fin')
```

```
t = "coucou"
for c in t:
    print(c)
```

Remarques sur les scripts Python

- Les commentaires sont ajoutés en utilisant le caractère « # »
- Il est recommandé d'inclure l'un des pseudo-commentaires suivants au début de vos scripts Python afin d'éviter les problèmes des accents et guillemets français.

```
# -*- coding:Latin-1 -*-
# -*- coding:Utf-8 -*-
```

- Attention à l'indentation
  - Vous ne pouvez pas ajouter des espaces en début de ligne.

    Les espaces en début de ligne sont importants dans la syntaxe de Python, car ils délimitent des blocs d'instructions

Description du robot-ordinateur

1) Ordinateur gestionnaire de casiers
    1) Présentation de la mémoire
    2) Déclaration et affectation de variable
    3) Opérateurs et conditions
    4) Communication avec l'extérieur
2) **Tableaux d'évolution de variables**

Tableaux d'évolution de variables

- Outil pour simuler le déroulement d'un programme informatique
- Utilités
  - compréhension d'un algorithme
  - vérification du bon comportement par rapport à la tâche qu'on demande

Exemple de tableau d'évolution de variables

N° de ligne

```
1. int X
2. int Y
3. X <- 1
4. Y <- 4
5. TANT QUE X != Y FAIRE
6.     print 'foo'
7.     Y <- Y-1
8. print 'bar'
```

Exemple de tableau d'évolution de variables

N° de ligne

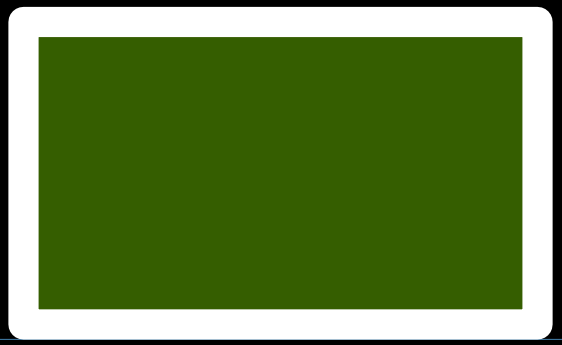
```
1. int X
2. int Y
3. X <- 1
4. Y <- 4
5. TANT QUE X != Y FAIRE
6.     print 'foo'
7.     Y <- Y-1
8. print 'bar'
```

N° de ligne	X	Y	X != Y

Exemple de tableau d'évolution de variables

N° de ligne

```
1. int X
2. int Y
3. X <- 1
4. Y <- 4
5. TANT QUE X != Y FAIRE
6.     print 'foo'
7.     Y <- Y-1
8. print 'bar'
```



N° de ligne	X	Y	X != Y

Conclusion sur l'algorithmique informatique

- Image du fonctionnement d'un ordinateur
  - Gestionnaire de casier
- Instructions élémentaires (et leurs variantes)
  - Déclaration et affectation de variables
  - Opérateurs et conditions
  - Communication avec l'extérieur : lecture/écriture
  - *Les procédures/fonctions restent à faire ...*
- Tableaux d'évolution de variables
  - Savoir construire et remplir un tableau d'évolution de variables
  - Savoir tirer des informations générales sur l'algorithme à partir du tableau

Conclusion sur l'algorithmique informatique