

# Introduction à l'algorithmique et à la programmation en Python

# Informations générales

- **Objectifs :**
  - comprendre et utiliser des schémas algorithmique . . .
  - . . . en utilisant le langage Python (version 3)
- **Ressources :**
  - **Slides**
  - Un **TP de découverte** de Python et de l'environnement IDLE
  - **Python Tutor** : un excellent outil permettant d'exécuter un programme pas-à-pas, en visualisant le contenu de chaque variable. **A utiliser sans modération** pour bien comprendre ce qu'il se passe!  
<http://www.pythontutor.com> (et choisir Python 3.6)
  - **Exercices** : quizz et exercice de programmation
  - **Projet final**

# Sommaire (cliquable)

1. Variables, Affectations, Entrées/Sorties
2. Instruction If .... else
3. Booléens et chaînes de caractères
4. Boucles while
5. Fonctions (bases)
6. Fonctions (avancé)
7. Listes
8. Boucles For
9. Dictionnaires
10. Lecture/Écriture dans un fichier

# **THÈME 1: VARIABLES, AFFECTATIONS, ENTREES/SORTIES**

# Notions du thème:

- Introduction :
  - algorithme, programme
  - Interpréteur, compilateur
- Variables et expressions
- Entrées / Sorties

# Compilation et interprétation

- L'ordinateur permet d'**automatiser** des tâches
- Mais il faut utiliser un **programme**
  - syntaxe précise dans un langage donné lisible par l'humain
  - doit être transformé en un texte lisible par la machine (suite d'octets) → compilateur ou interpréteur
- **compilateur** : traduit une fois pour toute le code source en exécutable
- **interpréteur** : traduit au fur et à mesure à chaque lancement du programme interprété
- **Python** est un langage **interprété**

# Algorithme vs Programme

- Problème complexe: travail en 2 temps
  1. Résolution du problème  
en s'autorisant syntaxe approximative, fautes d'orthographe, abréviations (et en s'aidant avant avec des schémas) => **algorithme**
  2. Rédaction de la solution finale  
produire le **programme** qui permettra de faire faire ce qui est demandé à l'ordinateur
- Notion d'algorithme est plus générale :  
cuisine, protocole expérimental, indications routières...

# Exercice

- Donner l'algorithme pour faire une omelette (un seul œuf)
- Préciser
  - les ingrédients nécessaires,
  - les ustensiles nécessaires,
  - les actions à mener (dans l'ordre)





# Exercice 1 : faire une omelette

Algo Omelette\_Élémentaire

Début

*{ingrédients}*

1 œuf, sel poivre, beurre

*{ustensiles}*

1 saladier, une fourchette, une poêle, une spatule

*{procédure}*

Casser l'œuf dans un saladier

Saler et poivrer

Battre l'œuf à la fourchette

Dans une poêle, faire chauffer le beurre,

Verser l'œuf battu dans la poêle,

Cuire doucement jusqu'à l'obtention de la texture souhaitée

(baveuse à bien cuite) ←

Server

Fin

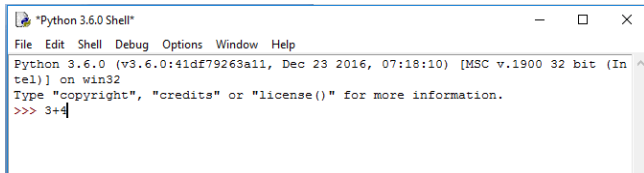
**déclarations  
variables**

**instructions**

**commentaires**

# Programmes en Python

- Deux modes d'exécution d'un code Python
  - utiliser l'interpréteur Python, instruction par instruction, un peu à la façon d'une calculatrice

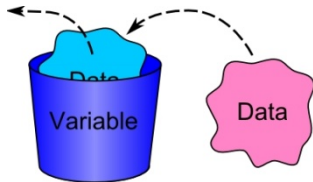


```
Python 3.6.0 Shell
File Edit Shell Debug Options Window Help
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 3+4
```

- écrire un ensemble d'instructions dans un fichier puis on l'exécute (via un interpréteur Python)

# Variables

- Conteneur d'information
- Identifié par un nom = un **identificateur**
- Ayant un « contenu »



# Identificateurs en Python

- **Suite non vide de caractères**
  - commençant par une lettre ou le caractère `_`
  - contenant seulement des lettres, des chiffres et/ou le caractère `_`
  - Ne peut pas être un mot réservé de Python
- **Exemples d'identificateurs**
  - **valides** : toto, proch\_val, max1, MA\_VALEUR, r2d2, bb8, \_mavar
  - **non valides** : 2be, C-3PO, ma var
- Les identificateurs sont **sensibles à la casse** : `ma_var` != `Ma_Var`
- **Conventions** pour les variables en Python :
  - utiliser des minuscules
  - pas d'accents

# Affectation

- Pour mémoriser une valeur dans une variable, on fait une affectation en utilisant le signe =
- **Exemples :**

```
n = 33
```

```
a = 42 + 25
```

```
ch = "bonjour"
```

```
euro = 6.55957
```

- L'identificateur (à gauche de =) reçoit la valeur (à droite du =; l'expression doit d'abord être évaluée).
- La première affectation d'une variable est aussi appelée **initialisation**

# Exemple

```
Python 3.6.0 Shell
File Edit Shell Debug Options Window Help
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:11)
Type "copyright", "credits" or "license()" for more in:
>>> a = 3 * 7
>>> a
21
>>> b = 7.3 # separateur decimal est un point
>>> b
7.3
>>> a = b + 5
>>> a
12.3
>>> a = a * 3
>>> a
36.900000000000006
>>>
```

La valeur d'une variable peut changer  
au cours de l'exécution d'un programme  
La valeur antérieure est perdue.

# Affectation vs Condition en Python

- Le signe “=” sert seulement à faire une affectation.
- Pour savoir si 2 nombres sont égaux, on utilise “==”

## Exemples :

```
>>> a = 6
```

```
>>> a
```

```
6
```

```
>>> b = 9
```

```
>>> a == b
```

```
False
```

# Notion de typage

- Les valeurs des variables sont de **nature** différente
  - entier
  - réel
  - chaîne de caractères

```
n = 33
a = 42 + 25
euro = 6.55957
ch = "bonjour"
```

- En programmation, on parle de **type**
- Selon les langages de programme, le type des variables
  - est déclaré dans le programme : typage **statique**
  - est déterminé par le compilateur/interprète : type **dynamique**
- En Python, le typage est **dynamique**
  - Pour connaître le type d'une variable: `type(ma_var)`



# Exemples

```
>>> a = 17
```

```
>>> type(a)
```

```
<class 'int'>
```

```
>>> a = "salut"
```

```
>>> type(a)
```

```
<class 'str'>
```

```
>>> a = 3.14
```

```
>>> type(a)
```

```
<class 'float'>
```

```
>>> type(21==7*3)
```

```
<class 'bool'>
```

# Expression

- C'est une formule qui peut être évaluée
- Exemples :

`42 + 2 * 5.3`

`3*2.0 - 5`

`"bonjour"`

`20 / 3`

- Expression : des **opérandes** et des **opérateurs**.
- Les opérateurs que l'on peut utiliser **dépendent du type** des valeurs qu'on manipule
- Une expression qui ne peut prendre que les valeurs True ou False est appelée **expression booléenne**

# Quelques opérateurs

- **arithmétiques** (sur des nombres) :  
+, -, \*, \*\*, /, %, //
- **de comparaison** (résultat booléen) :  
==, !=, <, >, <=, >=
- **logiques** (entre des booléens, résultat booléen) :  
or, and, not

# Exercice

Quelle est la réponse de l'interpréteur après chaque expression ?

```
>>> 2 + 3
```

```
>>> 2*3
```

```
>>> 2**3
```

```
>>> 20/3
```

```
>>> 20//3
```

```
>>> 20%3
```

```
>>> 2 > 8
```

```
>>> (2 <= 8) and (8 < 15)
```

```
>>> 2 <= 8 < 15
```

```
>>> (x % 2 == 0) or (x >= 0)
```

# Entrées / Sorties

- On a généralement besoin de pouvoir **interagir** avec un programme :
- pour lui fournir les données à traiter, par exemple au clavier : **entrées**
- pour pouvoir connaître le résultat d'exécution ou pour que le programme puisse écrire ce qu'il attend de l'utilisateur, par exemple, texte écrit à l'écran : **sorties**

# Les entrées : fonction **input()**

- A l'exécution, l'ordinateur :
  - interrompt l'exécution du programme
  - affiche éventuellement un message à l'écran
  - attend que l'utilisateur entre une donnée au clavier et appuie Entrée.
- C'est une saisie en **mode texte**
  - valeur saisie vue comme une **chaîne de caractères**
  - on peut ensuite changer le type

# Les entrées

```
>>> texte = input()
123
>>> texte + 1 # provoque une erreur
>>> val = int(texte)
>>> val + 1 # ok
124
>>> x = float(input("Entrez un nombre :"))
Entrez un nombre :
12.3
>>> x + 2
14.3
```

# Les sorties : la fonction print()

- affiche la **représentation textuelle** de n'importe quel nombre de valeurs fournies entre les parenthèses et séparées par des virgules
- à l'affichage, ces valeurs sont séparées par un **espace**
- l'ensemble se termine par un retour à la ligne
  - modifiable en utilisant sep et/ou end
- Possibilité d'insérer
  - des **sauts de ligne** en utilisant `\n` et
  - des **tabulations** avec `\t`



# Exemples de sorties

```
>>> a = 20
>>> b = 13
>>> print("La somme de", a, "et", b, "vaut",
          a+b, ".")
```

La somme de 20 et 13 vaut 33.

```
>>> print(a,b,sep= ";")
20;13
```

```
>>> print("a=",a, "b=",b, sep="\n")
a=
20
b=
13
```

- Ces slides ont été réalisés par:
  - Amir Charif
  - Lydie Du Bousquet
  - Aurélie Lagoutte
  - Julie Peyre
- Leur contenu est placé sous les termes de la licence **Creative Commons CC BY-NC-SA**

