Introduction à l'informatique

Cours 3

kn@lmf.cnrs.fr
https://usr.lmf.cnrs.fr/~kn

université PARIS-SACLAY

Plan

Définitions

Un *langage* est un système de communication **structuré**. Il permet **d'exprimer une pensée** et de **communiquer** au moyen d'un système de signes (vocaux, gestuel, graphiques, ...) doté **d'une sémantique**, et le plus souvent **d'une syntaxe**.

Un *langage de programmation* est un système de communication **structuré**. Il permet **d'exprimer un algorithme** de façon à ce qu'il soit **réalisable par un ordinateur**. Il est doté **d'une sémantique**, et **d'une syntaxe**.

Syntaxe et sémantique

La *syntaxe* est l'ensemble des **règles de bonne formation du langage**.

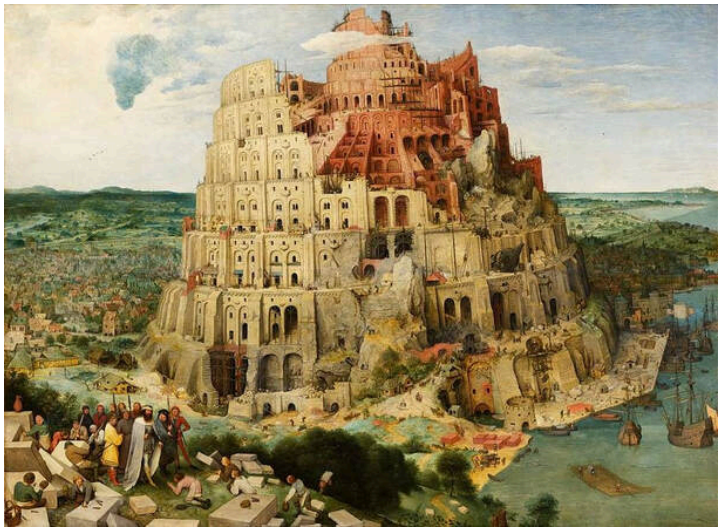
Exemple avec du code Python:

```
1 + 4          # est syntaxiquement correct

1 / 'Bonjour'  # est syntaxiquement correct

1 +            # est syntaxiquement incorrect
```

La *sémantique* est l'ensemble des règles qui donne le **sens** des programmes **bien formés**

```
1 + 4          # est sémantiquement correct

1 / "Bonjour"  # est sémantiquement incorrect
```

Quels langages de programmation ?



La tour de Babel, Pieter Brueghel l'Ancien, 156

Quels sont les caractéristiques des langages de programmation ?

◆ Généralistes ou dédiés : certains langages ont un but spécifique (par exemple SQL pour interroger les bases de données) d'autres sont généralistes (comme C++, Python ou Java)

◆ Compilés ou interprétés : des langages compilés sont traduits par un **compilateur** en instructions machines (C, C++, Java, ...).
Les langages interprétés disposent d'un **interprète** qui permet d'évaluer des phrases du langages (le Shell, Python, JavaScript, ...).

◆ Typés dynamiquement ou statiquement

◆ Impératifs, fonctionnels, orienté objets, logiques, ...

◆ ...

Est-ce une bonne chose ?

Oui !

◆ Il n'y a pas un unique langage de programmation qui soit le meilleur
◆ Un bon programmeur ou une bonne programmeuse se doit de connaître **plusieurs** langages
◆ Connaître plusieurs langages permet d'aborder des problèmes de façon différentes
◆ Cela permet aussi de choisir le meilleur outil pour résoudre son problème

Sur le cycle de Licence, au moins 5 langages

◆ C++ (programmation impérative)
◆ **Python** (programmation impérative et objet)
◆ Java (programmation Orientée Objet)
◆ OCaml (programmation fonctionnelle)
◆ SQL (interrogation de bases de données)

Plan

Description

- Créé par Guido van Rossum en 1991
- 1991 : Python 1.0
- 2000 : Python 2.0
- 2008 : Python 3.0
- 2020 : Python 2.0 n'est plus supporté, version actuelle 3.8

Dans ce cours, on utilisera exclusivement la version 3 du langage

C'est aussi cette version qui est maintenant au programme de 1ère et Tale

Caractéristiques du langage

- Langage généraliste : traitement de données, interfaces graphiques, réseau, jeux, calcul scientifique, intelligence artificielle, ...
- Langage interprété : la commande **python3** permet d'exécuter des scripts Python
- (et bien d'autres qu'on va découvrir au fur et à mesure)

Un premier programme

On considère le fichier **salut.py**

```
LIMIT=40                              #On définit une variable globale
entree = input("Quel est votre age ?") #On lit une entrée de l'utilisateur
age = int(entree)                     #On la convertit en nombre

if age >= LIMIT:                      #On teste la valeur et on affiche
    print ('Salut, vieux !')
else:
    print ('Salut, toi !')
```

On peut exécuter ce programme dans un terminal :

```
$ python3 salut.py
Quel est votre age ? 38
Salut, toi !
$
```


Qu'y a t'il dans ce programme ?

- Définition de variables
- Entrées (de l'utilisateur) et affichages (dans la console)
- Manipulation de constantes (nombres, textes, ...)
- Tests de valeurs

On va se familiariser avec ses aspects, ainsi qu'avec le programme **python3** et ses différents modes d'utilisation.

Comment programmer avec Python ?

On privilégie pour les TPs un mode minimal

1. Ouvrir un terminal
2. Créer et se placer dans un répertoire pour le TP (par exemple **`IntroInfo/TP3`**)
3. Lancer un éditeur de texte sur un fichier Python :

```
$ gedit exo1.py &
$
```

4. Éditer le code, sauver le fichier
5. Tester le programme :

```
$ python3 exo1.py
```


La boucle d'interaction

Le programme **`python3`** possède aussi un **mode interactif** qui permet d'évaluer des instructions, comme un shell.

Il suffit de lancer la commande **`python3`** sans argument.

```
$ python3
Python 3.6.9 (default, Apr 18 2020, 01:56:04)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 + 1
2
>>> 3 * 10
30
>>> x = 42
>>> x + 10
52
>>>
```

On peut quiter avec **`CTRL-d`**

Mode programme et mode intéractif

Le mode intéractif attend des instructions Python et les exécute au fur et à mesure.
Il peut être utilisé pour tester des petits morceaux de programmes.

Le mode programme : **`python3`** interprète les lignes du fichier passé en argument.

Plan

Les entiers

En Python, les entiers peuvent être de taille arbitraire. Ils sont signés (on peut représenter les nombres positifs et négatifs) :

```
>>> 1
1
>>> -149
-149
>>> 1111111333333333332999999999999013000000000000003333333333333333
1111111333333333332999999999999013000000000000003333333333333333
```


Opération sur les entiers

Symbole	Description
+	addition
-	soustraction
*	multiplication
/	division exacte (résultat à virgule)
//	division entière (résultat entier)
%	modulo
**	puissance

```
>>> 1 + 1
2
>>> 27 - 100
-73
>>> 3 * 4
12
>>> 4 / 3
1.33333333333333
>>> 4 // 3
1
```


Opération sur les entiers (suite)

```
>>> 10 % 4
2
>>> 2 ** 100
1267650600228229401496703205376
>>>  3 + 4 * 7
31
>>>  (3 + 4) * 7
49
>>> 1 / 0

Traceback (most recent call last):
  File "", line 1, in
ZeroDivisionError: division by zero
```


Conclusion sur les entiers

- Python supporte des entiers signés de taille arbitraire
- Les opérations arithmétiques **+**, **-**, *****, **%** et ****** sont naturelles
- Il existe deux version de la division : exacte (**/**) et entière (**//**)
- Les priorités sont aussi celles utilisées en mathématiques (*****, **/**, **//**, **%** sont plus prioritaires que **+**, **-**)
- On peut utiliser des parenthèses pour grouper les opérations
- Certaines opérations peuvent provoquer des erreurs (ex: division par 0)

Les nombres à virgule

En Python, les « nombres à virgule » ont une précision limitée. On les représente en utilisant la notation scientifique :

```
>>> 1.5
1.5
>>> -12.3423e13
-123423000000000.0
>>> 1.55555555555555555555555555
1.5555555555555556
```

Remarque : **-12.3423e13 = -12.3423 × 10^{13} = -123423000000000.0**

Attention : En Python, comme dans de nombreux langages, calculer avec des nombres à virgule (nombres *flottants*) peut provoquer des erreurs d'arrondi.

Opérations sur les nombres à virgule

On utilise les mêmes opérations que sur les entiers. Il n'y a qu'une seule division (**/**)

```
>>> 1.5 + 1.5
3.0
>>> 3.141592653589793 * 2
6.283185307179586
>>> 10.5 / 3
3.5
>>> 1.2 + 1.2 + 1.2
3.5999999999999996
>>> 4.5 ** 100
2.0953249170398634e+65
>>>  1.0 / 0
Traceback (most recent call last):
File "", line 1, in
ZeroDivisionError: float division by zero
```


Textes

On représente les « textes » par des **chaînes de caractères**.

```
>>> 'Bonjour, ça va bien ?'
'Bonjour, ça va bien ?'
```

On ne montre que quelques opérations sur les chaînes de caractères :

Symbole	Description
+	concaténation
*****	répétition
len(*s*)	nombre de caractères

Opérations sur les textes

```
>>> 'Bonjour' + ', ça va bien ?'
'Bonjour, ça va bien ?'
>>> 'Bonjour' * 3
'BonjourBonjourBonjour'
>>>len('Bonjour')
7
```

Attention : il existe des opérations plus complexes sur les chaînes, et plusieurs façons d'écrires celles ci. On verra cela au fur et à mesure.

Plan

Définition de variables

Une **variable** est un moyen de donner un **nom** au résultat d'un calcul.
En Python, une variable est une suite de caractères qui commence par une lettre ou un « _ » et contient des lettres, des chiffres ou des « _ ».
On définit une variable avec le symbole « = ».

```
>>> x = 314
>>> y = 2500 * 2
>>> x
314
>>> y
5000
>>> x + y
5314
>>> toto_A1 = x + y
>>> toto_A1 + 10
5324
```


Entrées et sorties

Le but d'un programme est de **lire des données**, calculer des résultats et les **afficher**. Les entrées peuvent êtres de plusieurs sortes (lecture de fichiers, connexions réseaux, clics de souris, ...). Les sorties peuvent être de plusieurs sortes (écriture dans des fichiers, connexions réseaux, affichage graphique, ...).

On commence par deux opérations simples

- Lire une chaîne de caractères au clavier (entrée)
- Afficher un message dans le terminal (sortie)

Pourquoi faire ?

En mode interactif, l'interpréteur affiche les résultats intermédiaires entre chaque instruction.

En mode programme (liste d'instructions dans un fichier `.py`) il fut utiliser des instructions pour effectuer des affichages et saisir des données.

input()

La fonction prédéfinie **input()** bloque le programme et attend que l'utilisateur saisisse du texte et valide avec la touche **entrée**.

La fonction renvoie le texte comme une chaîne de caractères.

On peut donner en argument à **input()** une chaîne de caractères qui sera affichée comme message.

```
nom = input('Entrez votre nom : ')
```

(suite sur le transparent suivant)

print()

La fonction prédéfinie **print(*s*)** affiche la chaîne de caractères *s* passée en paramètre.

```
nom = input('Entrez votre nom : ')
print ('Bonjour, ' + nom + ' !')
```

Les lignes ci-dessus sont écrites dans un fichier **bonjour.py**

```
$ python3 bonjour.py
Entrez votre nom : Kim
Bonjour, Kim !
$
```


Conversions

Considérons le programme suivant, écrit dans un fichier **age.py** :

```
annee = input('Entrez votre année de naissance : ')
age = 2020 - annee
print ('Vous avez ' + age + ' ans !')
```

```
$ python3 age.py
Entrez votre année de naissance : 1981
Traceback (most recent call last):
File "age.py", line 2, in
age = 2020 - annee
TypeError: unsupported operand type(s) for -: 'int' and 'str'
$
```

Ici l'opération **2020 - age** est incorrect : on essaye de soustraire une **chaîne de caractères** d'un nombre.

int()

La fonction prédéfinie **int(*s*)** permet de convertir une chaîne de caractères numériques en nombre. On essaye de corriger le programme **age.py** :

```
annee = input('Entrez votre année de naissance : ')
age = 2020 - int(annee)
print ('Vous avez ' + age + ' ans !')
```

```
$ python3 age.py
Entrez votre année de naissance : 1981
Traceback (most recent call last):
File "age.py", line 3, in
print ('Vous avez ' + age + ' ans !')
TypeError: can only concatenate str (not "int") to strTraceback (most recent ca
$
```

Il y a une autre erreur. On essaye de concaténer (« coller ») des chaînes de caractères et un nombre.

str()

La fonction prédéfinie **str(*v*)** permet de convertir une valeur *v* en chaîne de caractères.
On peut corriger complètement le programme **age.py** :

```
annee = input('Entrez votre année de naissance : ')
age = 2020 - int(annee)
print ('Vous avez ' + str(age) + ' ans !')
```

```
$ python3 age.py
Entrez votre année de naissance : 1981
Vous avez 39 ans !
```

Attention cependant, la fonction **int()** peut déclancher une erreur :

```
$ python3 age.py
Entrez votre année de naissance : Toto
Traceback (most recent call last):
File "age.py", line 2, in
age = 2020 - int(annee)
ValueError: invalid literal for int() with base 10: 'Toto'
```


Affichage des erreurs

Lorsqu'une erreur se produit en mode programme, le programe Python s'interrompt.
Le message d'erreur indique la ligne ou se situe l'erreur et la raison de cette dernière.

En mode interactif, la ligne n'est pas affichée (c'est forcément la dernière ligne saisie), et l'invite attend de nouvelles instructions Python.

Tests

Les tests sont des instructions fondamentales en programmation.
Ils permettent de donner au programme un **comportement différent** selon le résultat du test.
De manière simplifié, un test est composer de trois éléments :

1. Une expression **booléene** (la condition)
2. Une suite d'instructions à exécuter si la condition est vraie
3. Optionnellement, une suite d'instructions à exécuter si la condition est fausse.

Les booléens

L'algèbre de Boole (George Boole, 1847) est une branche de l'algèbre dans laquelle on ne considère que deux valeurs : **True** et **False**.
Les opérations sur ces valeurs sont la négation (**not**), le « ou logique » (**or**) et le « et logique » (**and**).

On peut manipuler ces objets en Python, comme on le fait avec des entiers, des nombres à virgule ou des chaînes de caractères.

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

Les comparaisons

Les booléens servent à exprimer le résultat d'un **test**. Un cas particulier de test sont les comparaisons. Les opérateurs de comparaisons en Python sont :

Symbole	Description
`==`	égal
`!=`	différent
`<`	inférieur
`>`	supérieur
`<=`	inférieur ou égal
`>=`	supérieur ou égal

Attention : dans les premiers cours on ne comparera que des nombres. Les comparaisons d'autres types (chaînes de caractères par exemple) seront expliquée plus tard.

Les comparaisons (exemples)

Le résultat d'une comparaison est toujours un booléens (`True` ou `False`) :

```
>>> 2 == 1 + 1
True
>>> 3 <= 10
True
>>> x = 4
>>>  x > 3 and x < 8
True
>>>  not (x == 4)
False
```

Attention : ne pas confondre « `=` » (affectation d'une variable) et « `==` » (égalité).

Instruction `if/else`

La syntaxe de l'instruction `if` est :

```
if e:
    i1
    i2
    ...
else:
    j1
    j2
    ...
suite
```

- `e` est une expression booléene (dont le résultat est `True` ou `False`)
- Les instructions i_n sont exécutée si `e` vaut `True`
- Les instructions j_n sont exécutée si `e` vaut `False`
- La partie `else:` est optionnelle
- Suite est exécuté après la dernière instruction i_n ou j_n
- Les instructions i_n et j_n **doivent être décalées de 4 espaces**.

Blocs d'instructions

Python est un langage dans lequel l'indentation est **significative**. C'est à dire qu'on ne peut pas mettre des retours à la ligne ou des espaces n'importe où.
L'indentation indique des **blocs** d'instructions qui appartiennent au même contexte. Exemple :

```
x = 45
if x < 10:
    print ("on a testé x")
    print ("x est plus petit que 10")
```

Le code ci-dessus n'affiche rien.

```
x = 45
if x < 10:
    print ("on a testé x")
print ("x est plus petit que 10")
```

Le code ci-dessus affiche « `x est plus petit que 10` ». L'absence d'indentation mets le deuxième `print` en dehors du `if`

Comparaison avec C++

En Python, l'indentation joue le même rôle que les accolades en C++

```
int x = 45
if (x < 10) {
    print ("on a testé x");
    print ("x est plus petit que 10");
}
```

```
int x = 45
if (x < 10) {
    print ("on a testé x");
}
print ("x est plus petit que 10");
```

(attention, il n'y a pas de fonction **print** prédéfinie en C++, c'est juste pour l'exemple)

Commentaires

On peut ajouter des commentaires dans un fichier Python.

Un commentaire est toute séquence de caractères qui commence par « **#** » jusqu'à la fin de la ligne.
Les commentaires sont **ignorés** par l'interpréteur Python.
Ils sont là pour aider les personnes qui écrivent le code ou qui le lisent.
Il est **très important de commenter judicieusement** son code.

```
if x > 10 and x < 100:
    # x est dans les bonnes bornes de température
    y = x * 10
else:
    # trop froid ou trop chaud, on réinitialise y
    y = 0
```

C'est particulièrement important pour faire le lien entre les objets manipulé par le langages et les concepts « humains » qu'ils représentent.

Retour sur notre exemple

```
LIMIT=40                                #On définit une variable globale
entree = input("Quel est votre age ?") #On lit une entrée de l'utilisateur
age = int(entree)                       #On la convertit en nombre

if age >= LIMIT:                        #On teste la valeur et on affiche
    print ('Salut, vieux !')
else:
    print ('Salut, toi !')
```


Après un survol de Python...

- Fonctionne en mode intéractif (« commandes ») ou programme (« dans un fichier »)
- Permet de manipuler des nombres (entiers, à virgule), du texte, des booléens
- La construction **if/else** permet d'exécuter du code en fonction du résultat d'un **test**
- Les instructions peuvent être regroupées en blocs.
- L'indentation (le décalage) marque les blocs.