

Introduction à la programmation en Python

Maria Zrikem

GE1, GI1, GS1 ENSA de Marrakech

2022 - 2023



1. Les bases de Python

Généralités

Python : Environnement de travail

Variables - Types - Expressions

Instructions de contrôle :
conditions et répétitives

2. les structures de données en python

Less listes

Les tuples

les dictionnaires

les ensembles

3. Manipulation des fichiers

Les fichiers texte

les fichiers CSV

Les exceptions

4. Les fonctions et modules

Les fonctions

les modules

5. La programmation objet en python

- Généralités - Concept Objet

 - Vocabulaire et principe d'encapsulation

- Définir des classes

 - Créer des objets : instanciation

 - Accès : attributs et méthodes

- Gérer des collections d'objets

6. Utilisation de fenêtres et de graphismes

Références

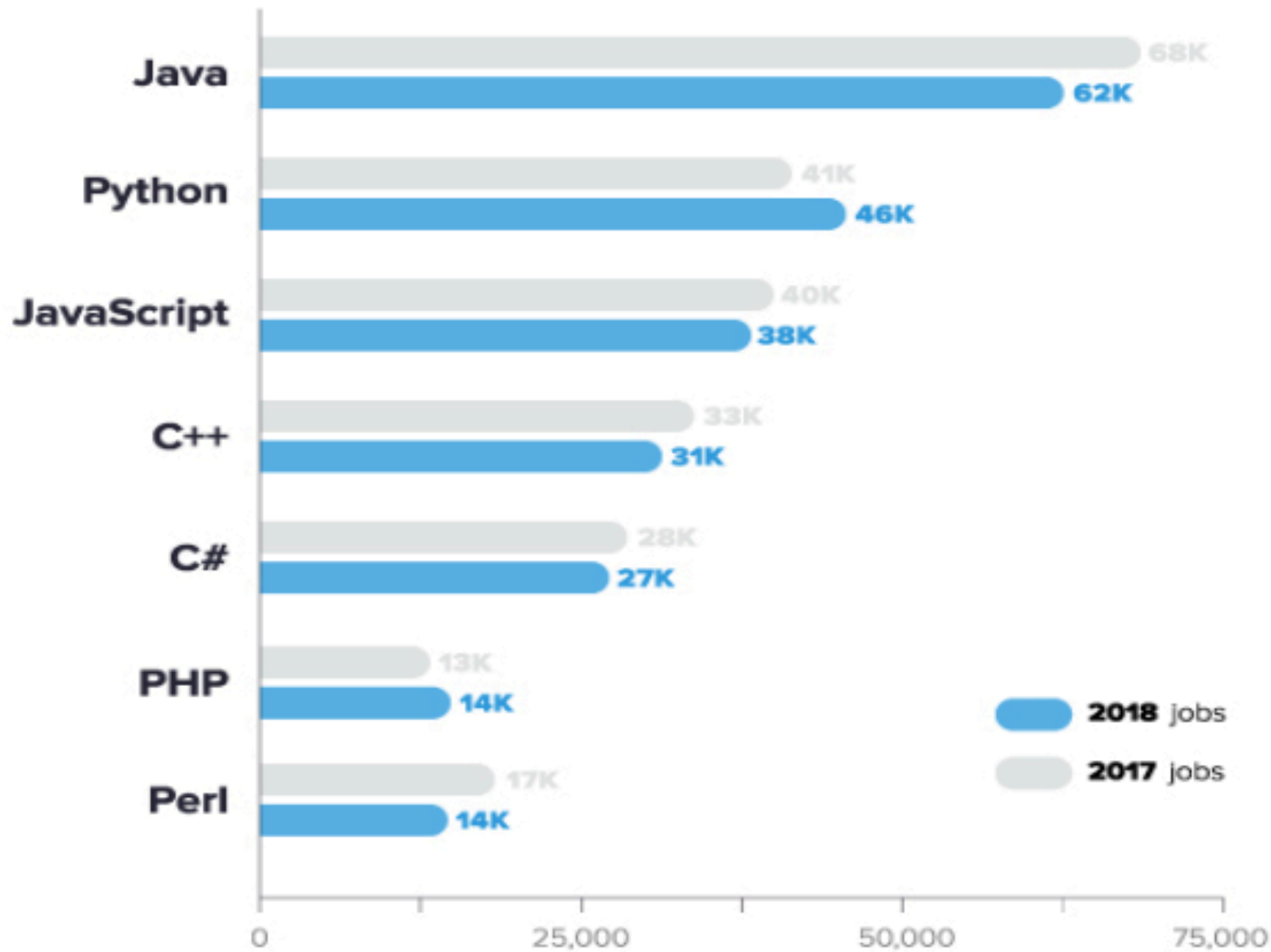
- Livre: apprendre à programmer avec Python 3, Gérard Swinnen
- Cours: Découverte de la Programmation Sous Python, Fouzia Moussouni
- Apprenez à programmer en Python, Prolixe



Introduction

Job postings containing top languages

Indeed.com - November, 17th 2017



IRNI - Python

Introduction

Top 8 Most Demanded Programming Languages in 2022

1. JavaScript / TypeScript. ...
2. Python. ...
3. Java. ...
4. C# ...
5. PHP. ...
6. C/C++ ...
7. Ruby. ...
8. GO.

11 déc. 2022

Programmes et langages

Dans un ordinateur, on distingue deux sortes de programmes :

- **Le système d'exploitation** : l'ensemble des programmes qui gèrent les ressources matérielles et logicielles. Il propose une aide au dialogue entre l'utilisateur et l'ordinateur : l'interface textuelle (interpréteur de commande) ou graphique (gestionnaire de fenêtres). Il est souvent multi-tâches et parfois multi-utilisateurs
- **Les programmes applicatifs** sont dédiés à des tâches particulières. Ils sont formés d'une série de commandes contenues dans un programme source qui est transformé pour être exécuté par l'ordinateur.

Programmes et langages

Il y a différents niveaux de langages :

- Chaque processeur possède un langage propre, directement exécutable : **le langage machine**. Il est formée d'une suite de 0 et de 1 appelés bits, souvent traités en groupes de 8 (octet), 16, 32 ou 64 bits. Il n'est pas portable, mais c'est le seul que l'ordinateur puisse utiliser. Il n'est pas facile de programmer en binaire.
- **Le langage d'assemblage** est un codage alphanumérique du langage machine. Ses instructions sont très élémentaires. Il est plus lisible que le langage machine, mais n'est toujours pas portable. On le traduit en langage machine par un assembleur.
- **les langages de haut niveau**. Souvent normalisés, ils permettent le portage d'une machine à l'autre. Ils sont composés d'instructions plus abstraites et plus puissantes. Ils sont traduits en langage machine par **un compilateur** ou **un interpréteur**.

- Language machine : Le langage machine est un langage binaire directement compréhensible par la machine.
- Language Assemblé : dit aussi Assembleur est un langage bas niveau très facilement traduisible pour être compris par la machine.
- Language de programmation : est un langage compréhensible par l'humain. Il doit être compilé ou interprété pour être compris par la machine.

Deux techniques : la compilation et l'interprétation

La compilation

- La compilation traduit le code source d'un programme dans son ensemble en langage objet. Cela consiste en une phase d'analyse lexicale, syntaxique et sémantique plus une phase de production de code objet. Pour générer le langage machine il faut encore une phase particulière : l'édition de liens.
- La compilation est contraignante mais offre au final une grande vitesse d'exécution.
- Un langage de programmation pour lequel un compilateur est disponible est appelé **un langage compilé**.



Deux techniques : la compilation et l'interprétation

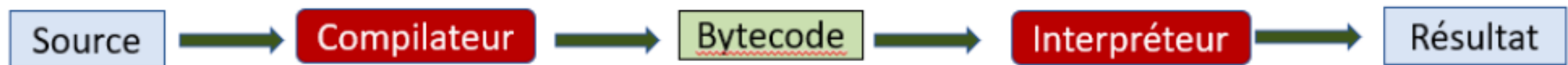
L'interprétation

- Chaque ligne du source analysée est traduite au fur et à mesure en instructions directement exécutées
- Aucun programme objet n'est généré.
- Cette technique est très souple mais les codes générés sont peu performants : l'interpréteur doit être utilisé à chaque exécution.
- Un langage de programmation pour lequel un interpréteur est disponible est appelé un **langage interprété**.



Production de programme : compilateur vs interpréteur

- **Techniques mixtes** : l'interprétation du bytecode compilé est un bon compromis entre la facilité de développement et la rapidité d'exécution. Exemple : le cas de Python.
- Le bytecode est une forme intermédiaire, portable sur tout ordinateur muni de la machine virtuelle Python.



- Interprétation pour exécuter un programme :
 - Python charge le fichier source .py en mémoire, en fait l'analyse syntaxique, produit le bytecode et enfin l'exécute.
 - Afin de ne pas refaire inutilement toute la phase d'analyse et de production, Python sauvegarde le bytecode produit (dans un fichier .pyo ou .pyc) et recharge simplement le fichier bytecode s'il est plus récent que le fichier source dont il est issu.
 - Pour chaque module importé par le programme, Python vérifie d'abord s'il existe une version précompilée du bytecode dont la date correspond au fichier .py.
 - S'il y en a un, Python l'utilise, sinon il fait une analyse syntaxique du module .py, et utilise le bytecode qu'il vient de générer.
- Pas besoin de compiler explicitement un module, Python gère ce mécanisme de façon transparente.

Plusieurs modèles sont possibles, entre autres :

- **la méthodologie procédurale** tente de structurer d'abord les traitements (actions). Elle consiste à diviser le problème en sous-problèmes plus simples (analyse descendante), et ensuite réutiliser au maximum les sous algorithmes (analyse remontante).
- **la méthodologie objet** décortique le problème sous un angle différent : partir de "Sur quoi porte le programme ?" et non plus de "Que doit faire le programme ?". Elle tente de structurer d'abord les données (dans des objets) auxquelles elle associe des traitements (actions). C'est une représentation qui se rapproche du monde réel. Le système est un ensemble d'objets qui interagissent entre eux et avec le monde extérieur (à l'aide de méthodes).

Python offre les deux techniques.

- **Un algorithme** : un ensemble d'instructions décrivant les étapes de résolution d'un problème.
- **Un programme** : est la traduction d'un algorithme en un langage de programmation (compilable ou interprétable).



- Un programme est souvent écrit en plusieurs parties dont une qui pilote les autres : le programme principal.

Développer : les étapes

- Ecrire le code dans un éditeur de texte
- Sauvegarder dans un fichier
- Compiler et interpréter le code
- Corriger les erreurs. On dit aussi **"debug"**
- Recommencer l'étape précédente jusqu'à ce qu'il n'y ait plus d'erreur
- Effectuer des tests et améliorer si besoin.

Présentation d'un programme : le code source est destiné à l'être humain. Pour en faciliter la lecture, il doit être judicieusement commenté. [En Python](#), un commentaire commence par le caractère `#` et s'étend jusqu'à la fin de la ligne :

```
#-----  
# Voici un commentaire  
#-----  
  
9 + 2 # En voici un autre
```


Trois types d'erreurs

- **Erreurs de syntaxe** : sont des erreurs dues au non respect des règles de syntaxe du langage de programmation utilisé. Elles sont détectées et provoquent l'arrêt du programme. L'utilisateur est alerté par un message.

Exemple

Syntaxe correcte : `Nom= "Dupont"`

Code écrit : `Nom = Dupont` l'oubli des guillemets est une erreur de syntaxe

- **Erreurs de sémantique** : sont des erreurs de logique. Elles ne sont pas détectées et ne gênent pas l'exécution du programme mais le résultat n'est pas celui attendu.

Exemple

La surface d'un rectangle `Surface = Largeur*Longueur`

Le calcul de la surface avec le code : `Surface = Largeur/Longueur` est une erreur sémantique

- **Erreurs d'exécution** : ce sont des erreurs qui apparaissent quand le programme est en cours d'exécution. Elles sont liées au contexte d'exécution. Elles sont détectées lorsque quelque chose d'inhabituel gêne l'exécution comme lire un fichier qui n'existe pas.

Exemple

Le code suivant génère une erreur à l'exécution :

$A=50$

$B=100-2*A$

$C=A/B$

Une division par 0 est une erreur qui se produit à l'exécution.

Le langage python?

- Python est un langage de programmation haut niveau,
- Il a été créé en 1989 par Guido van Rossum.
- Ses principales caractéristiques :
 - portable,
 - gratuit,
 - dynamique, Interprété
 - extensible,
 - modulaire
 - et orienté objet.
- Il a une syntaxe simple et intuitive et un niveau d'abstraction élevé
- Il offre un système de gestion d'exceptions
- Nombreux types d'utilisation / bibliothèques spécialisées
- La version actuelle est Python 3.7 (2018) et le langage continue à évoluer..



Installer python?

- L'installation de Python est facile, aussi bien sous Windows que sous les systèmes Unix.
- Quel que soit votre système d'exploitation, vous devez vous rendre sur le site officiel de Python : <https://www.python.org>

Sous Windows

- Cliquez sur le lien `Download` dans le menu principal de la page.
- Sélectionnez la version de Python que vous souhaitez utiliser.
- Enregistrez puis exécutez le fichier d'installation et suivez les étapes
- Une fois l'installation terminée, vous pouvez vous rendre dans le menu `Démarrer>Tous les programmes`.

Anaconda

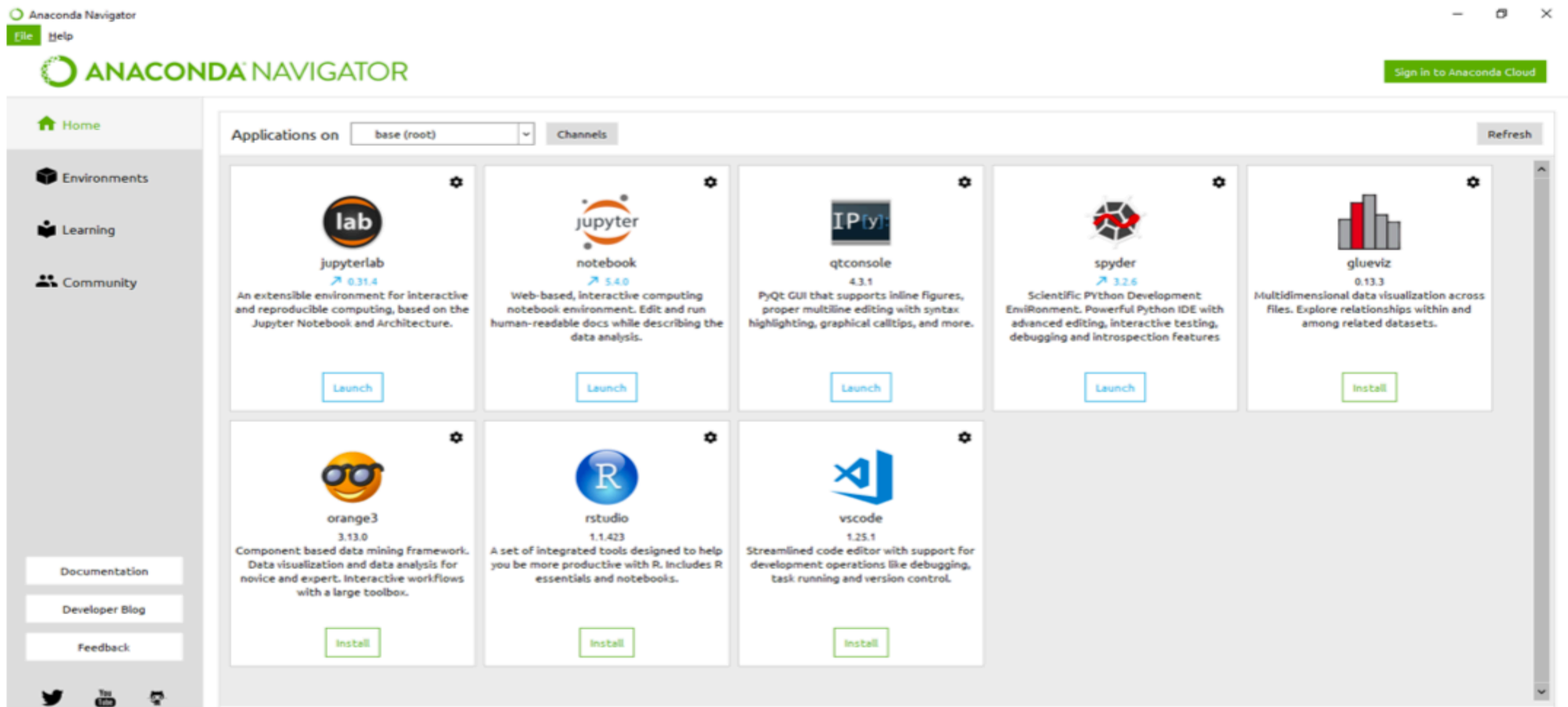
- Anaconda est une distribution libre et open source de gestion de paquets pour les langages Python et aussi R appliqué au développement d'applications dédiées à la science des données et à l'apprentissage automatique.
- Il permet de nombreuses fonctionnalités : la possibilité d'installer des librairies et de les utiliser dans les programmes, l'accès à plusieurs logiciels pour faciliter le développement et le test des programmes.
- L'accès à ses services via : [Anaconda Navigator](#) ou [Anaconda Prompt](#).
- **Spyder** est outil intégré dans Anaconda.



Environnement de travail : Programmation en Python

Anaconda Navigator

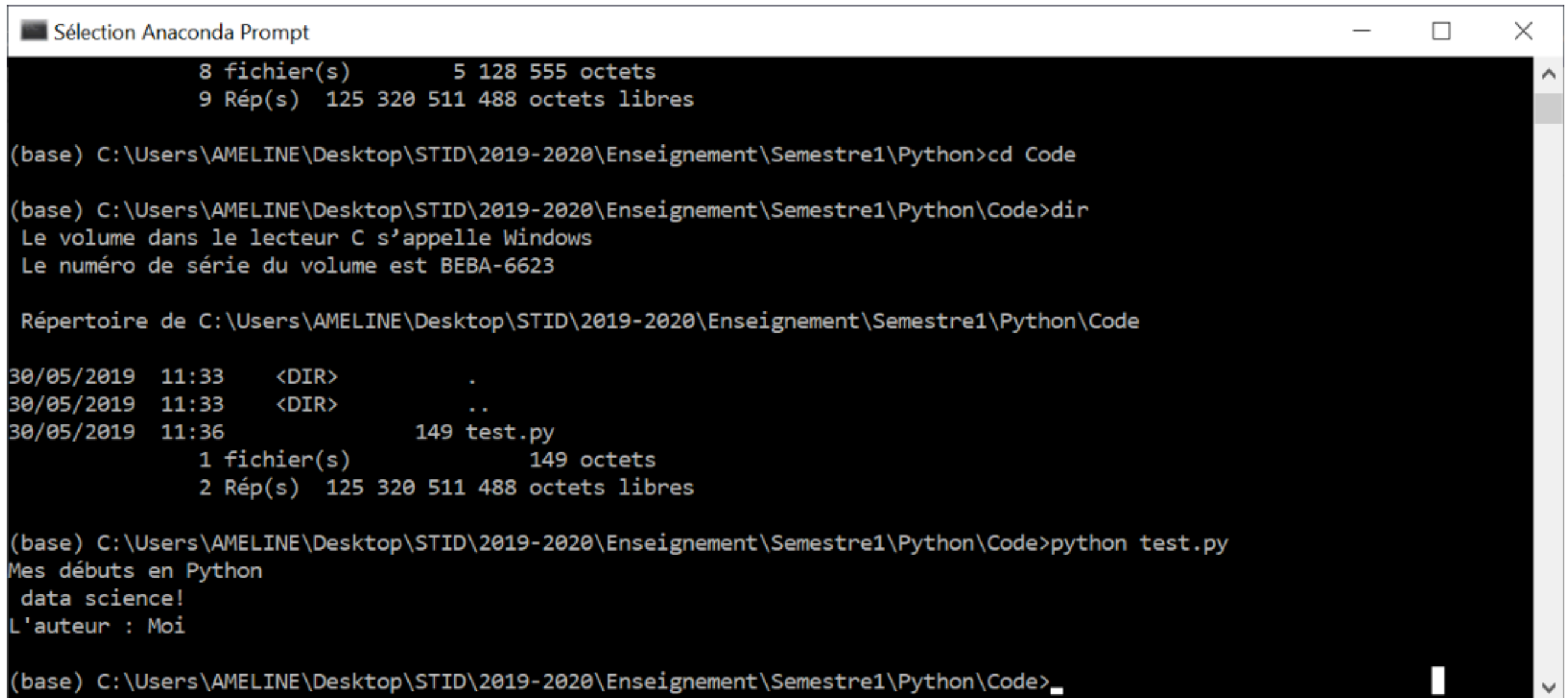
Une interface d'accès aux logiciels installés dans la suite Anaconda : permet de contrôler les librairies utilisées par le programme, vérifier si celles-ci sont bien à jour. Il offre de la documentation, un espace de partage communautaire, un service de Cloud...



Environnement de travail : Programmation en Python

Anaconda Prompt

C'est une console "interpréteur de commandes" permettant de taper des commandes comme par exemple l'exécution d'un programme python, ou encore l'installation d'une librairie... C'est l'équivalent d'un IDLE disponible initialement dans Python.



```
Sélection Anaconda Prompt

      8 fichier(s)          5 128 555 octets
      9 Rép(s)  125 320 511 488 octets libres

(base) C:\Users\AMELINE\Desktop\STID\2019-2020\Enseignement\Semestre1\Python>cd Code

(base) C:\Users\AMELINE\Desktop\STID\2019-2020\Enseignement\Semestre1\Python\Code>dir
Le volume dans le lecteur C s'appelle Windows
Le numéro de série du volume est BEBA-6623

Répertoire de C:\Users\AMELINE\Desktop\STID\2019-2020\Enseignement\Semestre1\Python\Code

30/05/2019  11:33    <DIR>          .
30/05/2019  11:33    <DIR>          ..
30/05/2019  11:36                149 test.py
      1 fichier(s)          149 octets
      2 Rép(s)  125 320 511 488 octets libres

(base) C:\Users\AMELINE\Desktop\STID\2019-2020\Enseignement\Semestre1\Python\Code>python test.py
Mes débuts en Python
data science!
L'auteur : Moi

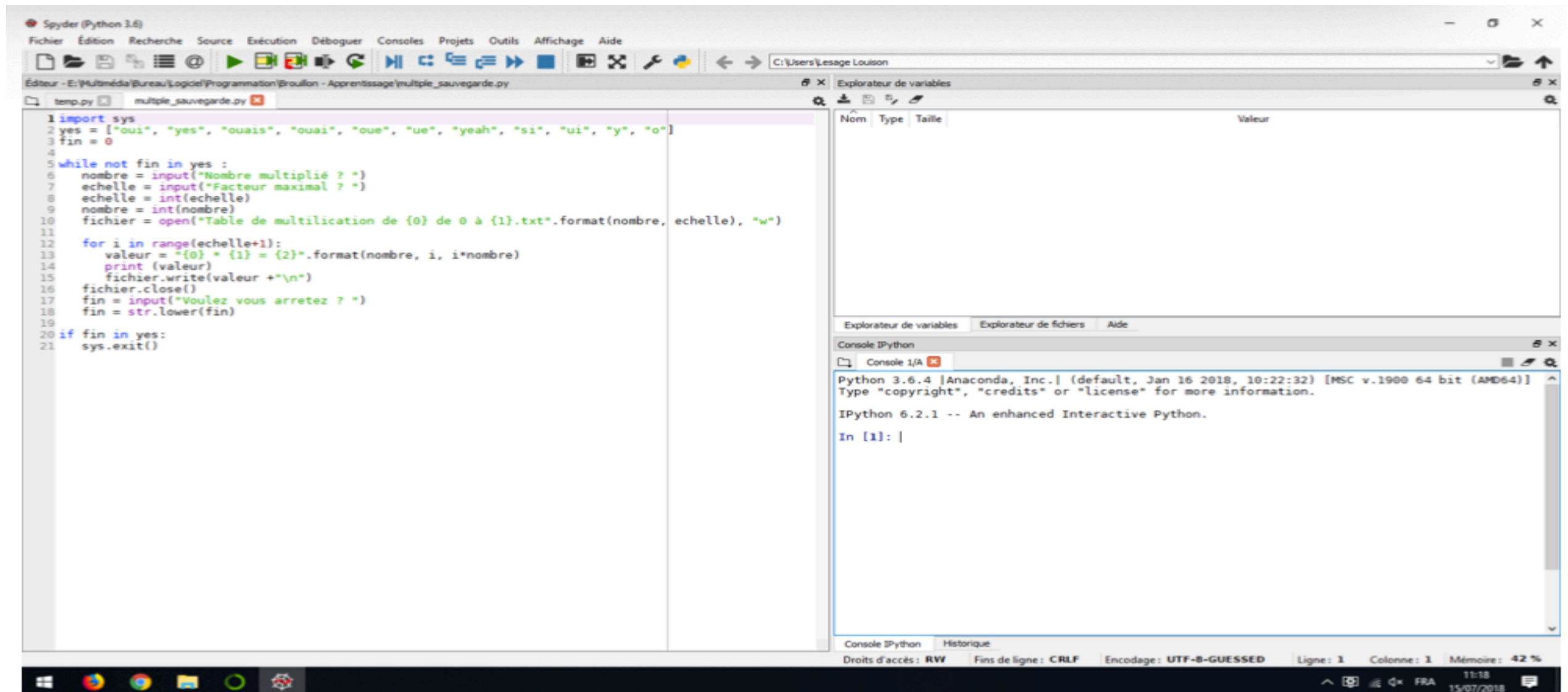
(base) C:\Users\AMELINE\Desktop\STID\2019-2020\Enseignement\Semestre1\Python\Code>_
```

Environnement de travail : Programmation en Python

Spyder

Spyder est un logiciel intégré dans Anaconda. Il permet de développer et tester les projets utilisateurs. Il est accessible via Anaconda Navigator. Il se divise en trois parties majeures, eux-mêmes divisibles.

Trois parties : Un éditeur de texte, un explorateur et une console.



Premier script en python

Exemple

```
print("Bonjour tout le monde !")
```

Exécution :

- **Mode interactif** : Taper cette instruction (commande) dans une console (celle de l'éditeur ou celle de l'IDE de Python) et valider. L'interpréteur Python s'exécute et le message s'affiche.

Remarque

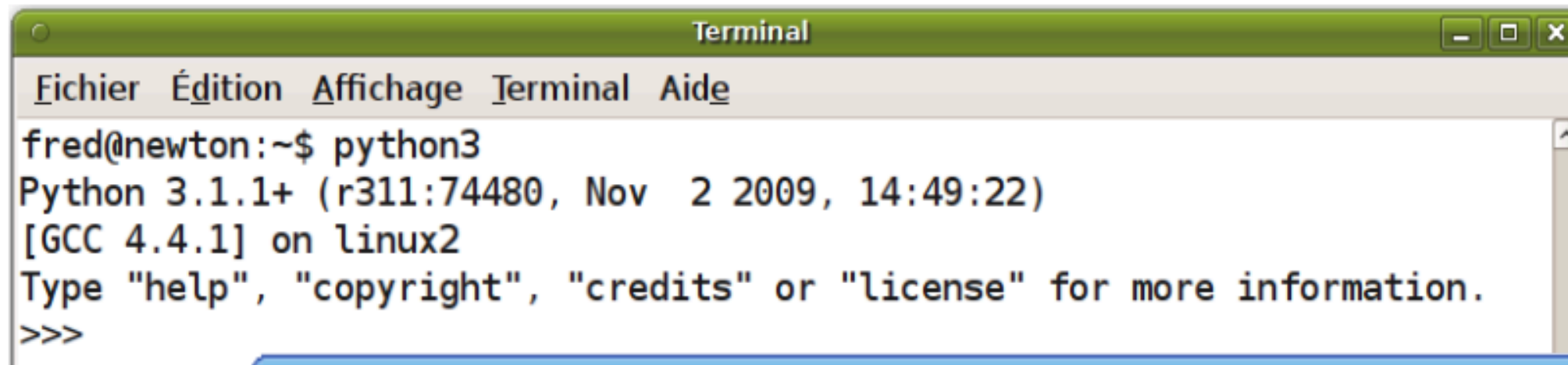
- Chaque ligne du source analysée est traduite au fur et à mesure en instructions directement exécutées
 - Aucun programme objet n'est généré.
- **Programme Python** : Saisir le code dans l'éditeur de texte et le sauvegarder dans un fichier avec l'extension .py (Ex test.py).

Exécuter :

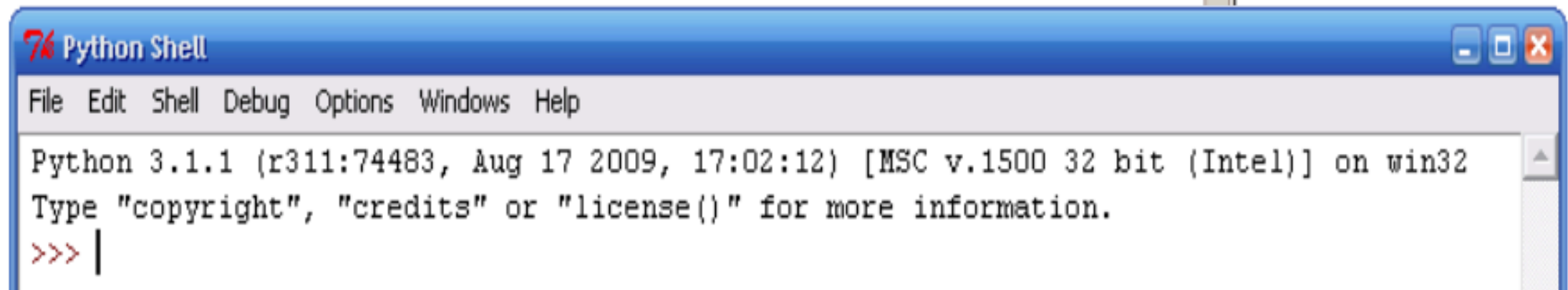
- via l'interface graphique de l'IDE utilisé (PyCham, Spyder, XEmacs)
- via la ligne de commande en tapant **python test.py** soit dans la console de l'IDE ou dans fenêtre de commandes.

Python en mode interactif

*L'interpréteur peut être lancé directement depuis la ligne de commande (dans un « shell » Linux, ou bien dans une fenêtre DOS sous Windows) : taper la commande **'python'** ou **'python2'** ou **'python3'***



```
Terminal
Fichier Édition Affichage Terminal Aide
fred@newton:~$ python3
Python 3.1.1+ (r311:74480, Nov  2 2009, 14:49:22)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.1.1 (r311:74483, Aug 17 2009, 17:02:12) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

Les trois caractères >>> constituent le **signal d'invite**, ou **prompt principal**, lequel vous indique que Python est prêt à exécuter une commande.

Python en mode interactif

- Utiliser l'interpréteur comme une simple calculatrice de bureau.
- Tester des commandes, comme :

```
>>> 5+3
```

```
>>> 2 - 9
```

```
# les espaces sont optionnels
```

```
>>> 7 + 3 * 4
```

```
# la hiérarchie des opérations mathématiques
```

```
# est-elle respectée ?
```

```
>>> (7+3)*4
```

```
# Les parenthèses sont fonctionnelles.
```

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

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

```
# une division entière
```

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

```
>>> 20.5 / 3
```

```
>>> 20,5 / 3
```

```
# Erreur
```

Structure d'un programme Python

Que contient un programme?

- l'encodage facultatif
- toutes sortes de commentaires
- l'importation des modules utilisés
- la définition des fonctions utilisateur
- et le programme principal

Commentaires

Une ligne : avec le caractère # au début

```
# commentaire
```

Plusieurs lignes : avec des trois quotes doubles " ou simples '

```
""" ligne 1  
   ligne 2  
"""
```

Structure d'un programme Python

```
# -*- coding: utf-8 -*-

'''Description de ce a quoi sert le module, ses fonctionnalites.'''

#Autres commentaires generaux (auteur, date d'ecriture, adresse e-mail, etc.)

# -----Imports: inclusion des modules dont vous avez besoin -----
#import os # exemple, a decommenter en cas de besoin
#import sys # exemple, a decommenter en cas de besoin

# ----- Fonctions et variables globales -----

def ma_fonction(param1, param2):
    '''Explication de ce que fait la fonction, des resultats renvoyes le cas
    echeant, et de ce a quoi correspondent ses parametres.'''
    pass # on ne fait rien (a remplacer par ce ce que fait la fonction)

# ----- Le corps du programme: ici commence l'exécution du programme
# ----- Le point de démarrage-----
if __name__ == '__main__':
    # on insère les tests des fonctions et les autres instructions à exécuter
    pass # on ne fait rien
```

Structure d'un programme Python

Exemple : Premier Programme (structure simplifiée)

```
1 # -*- coding: utf-8 -*-
2
3 """
4 Mon premier programme en Python
5 """
6 nom = input("Entrez votre nom : ")
7
8 print("Bonjour ", nom, "!")
9 print('--Fin--')
```

Règles de base

- En Python, il n'y a qu'une instruction par ligne.
- Les lignes ne commencent pas toutes au même endroit
 - Le positionnement (indentation) des lignes est important.
 - On appelle bloc d'instructions un ensemble d'instructions qui se suivent et avec la même indentation.
 - Modifier l'indentation modifie le comportement du programme.
- Les instructions sont exécutées dans l'ordre, de haut en bas

Variables - Types - Affectation

Variable

Une variable est un nom qui référence une valeur en mémoire ;

Le **type** d'une variable est le type de la valeur qu'elle référence.

Affectation

nom = **expression**, ou **expression** est une valeur ou un calcul qui produit une valeur.

Exemple

```
a=3
```

```
b = 'ENSA'
```

```
c=a+2
```

```
a=a+1
```

Variables - Types - Affectation

Type

- Les valeurs possèdent un type. Ce type va déterminer ce qui se passe quand on fait une opération sur des valeurs.
- Le type d'une variable est le type de la valeur qu'elle référence.

Les principaux types de base sont :

Type	En python	Exemples
entier	<code>int</code>	12, -4, 123545, . . .
flottant	<code>float</code>	3.14159, -1.5, 12., 4.56e12, . . .
booléen	<code>bool</code>	True (vrai) ou False (faux)
indéfini, rien	<code>None</code>	None
chaîne de caractères	<code>str</code>	'bonjour', 'IUT STID', . . .

Attention

Les majuscules-minuscules sont importantes : `True/= true`

Noms des Variables

- Les noms de variables sont des noms qu'on choisit assez librement.

De préférence assez courts, mais aussi explicites que possible, pour exprimer clairement ce que la variable est censée contenir :

altitude, *altit* ou *alt* (au lieu de *x*) pour exprimer une altitude
prix_unit pour exprimer un prix unitaire, etc

Nommage

Il y a certaines règles de nommage à respecter pour les variables (et fonctions introduites plus loin dans le cours) :

- Un nom de variable est une séquence de lettres (a à z , A à Z) et de chiffres (0 à 9), qui doit toujours commencer par une lettre.
- Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que \$, #, @, etc. sont interdits, à **l'exception** du caractère _ (underscore)
- La **casse est significative** (les caractères majuscules et minuscules sont distingués). *Attention : Joseph, joseph, JOSEPH sont donc des variables différentes*
- Le nom ne peut pas être un mot réservé de Python
- Prenez l'habitude d'écrire l'essentiel des noms de variables en caractères minuscules (y compris la première lettre). Il s'agit d'une simple convention, mais elle est largement respectée. N'utilisez les majuscules qu'à l'intérieur même du nom, pour en augmenter éventuellement la lisibilité, comme dans *TableDesMatières*.

Exemples

`a = 18`

`annee_nais = '12/10/2001'`

`3age = 18` `#faux`

Mots réservés

- Les mots suivants sont réservés pour le langage :

and	as	assert	break	class	continue
def	del	elif	else	except	finally
for	from	global	if	import	in
is	lambda	nonlocal	not	or	pass
raise	return	try	while	with	yield
True	False	None			

- Dans l'éditeur la coloration syntaxique d'un mot veut dire qu'il s'agit d'un mot réservé

Affectation

- En Python comme dans de nombreux autres langages, l'opération d'affectation (ou assignation) est représentée par le signe *égal*.

```
>>> n = 7
```

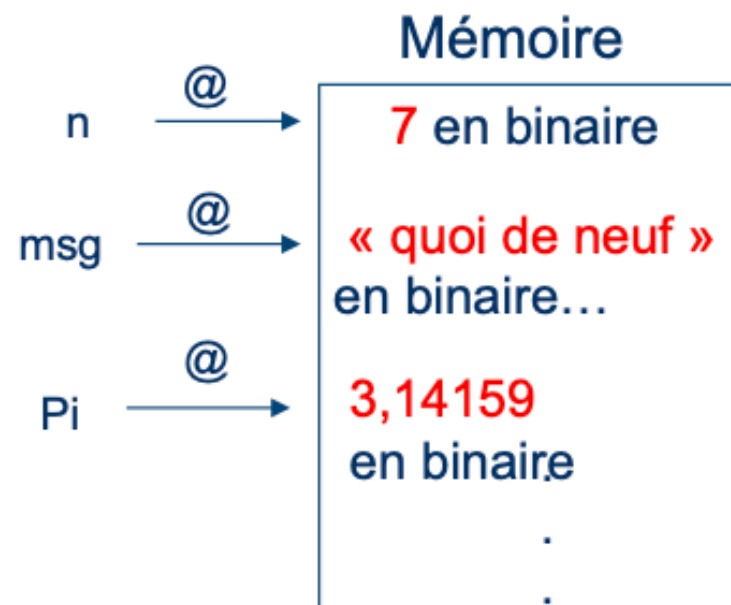
```
# donner à n la valeur 7
```

```
>>> msg = "Quoi de neuf ?"
```

```
# affecter la valeur "Quoi de neuf ?" à msg
```

```
>>> pi = 3.14159
```

```
# assigner sa valeur à la variable pi
```



Variables - Types - Affectation

Afficher la valeur d'une variable

En Python, pour afficher la valeur à l'écran, il existe deux possibilités :

- La première consiste à entrer au clavier le nom de la variable, puis <Enter>.

```
>>> n
7
>>> msg
"Quoi de neuf ? "          #affiche les guillemets, donc le type.
>>> pi
3.14159
```

- A l'intérieur d'un programme, vous utiliserez toujours l'instruction **print** :

```
>>> print (msg)
Quoi de neuf ?          #pas de guillemets.
>>> print(n)
7
```

Pas de typage explicite sous python

- Sous Python, il n'est pas nécessaire d'écrire des lignes de programme spécifiques pour définir le type des variables avant de pouvoir les utiliser.
- Il vous suffit en effet d'assigner une valeur à un nom de variable pour que celle-ci soit automatiquement créée avec le type qui correspond au mieux à la valeur fournie.
- Par exemple, les variables `n`, `msg` et `pi` ont été créées automatiquement chacune avec un type différent :
 - « nombre entier » pour `n`,
 - « chaîne de caractères » pour `msg`,
 - « nombre à virgule flottante » ou « float », pour `pi`.

Variables - Types - Affectation

Affectation multiple

- Sous Python, on peut assigner une valeur à plusieurs variables simultanément.

```
>>> x = y = 7
>>> x
7
>>> y
7
```

- On peut aussi effectuer des *affectations parallèles* à l'aide d'un seul opérateur :

```
>>> a, b = 4, 8.33
>>> a
4
>>> b
8.33
```

Remarque : $x, y = y, x$ échange la valeur de x et de y.

Variables - Types - Affectation

Transtypage

- La fonction `type()` permet de connaître le type d'une valeur.
- Il est possible de changer le type d'une valeur (le transtypage).
 - `str(?)` convertit en chaîne de caractère
 - `int(?)` convertit en entier, quand cela est possible
 - `float(?)` convertit en flottant, quand cela est possible
 - `bool(?)` convertit en booléen

Exemples

Instruction	Résultat	Instruction	Résultat	Instruction	Résultat
<code>type(10)</code>	?	<code>type(-10)</code>	?	<code>type('01')</code>	?
<code>type(0)</code>	?	<code>type(2.5)</code>	?	<code>type('1.0')</code>	?
<code>type('STID')</code>	str	<code>type(true)</code>	?	<code>type('false')</code>	str
<code>int(4.5)</code>	4	<code>int(-4.5)</code>	-4	<code>int('0345')</code>	345
<code>int('IUT')</code>	erreur	<code>float(4)</code>	4	<code>float('4.5')</code>	4.5
<code>str(4)</code>	'4'	<code>str(True)</code>	'True'	<code>str(-4.5)</code>	'-4.5'
<code>bool(4)</code>	True	<code>bool(0)</code>	False	<code>bool('IUT')</code>	True

Variables - Types - Affectation

Transtypage

En pratique, on se sert surtout de :

- `str` qui fonctionne tout le temps.
- `int` et `float` appliqués à une chaîne de caractères donne un nombre.
Attention : `int('1')` fonctionne, mais pas `int('1.2')` !
- `int` appliqué à un float pour tronquer les décimales
- `bool` qui donne `False` si son paramètre équivaut à 0, `True` sinon

A retenir pour les Booléens

Python attribue à une expression booléenne la valeur `False` si c'est :

- la constante `False`
- la constante `None`
- une séquence ou une collection vide
- une donnée numérique de valeur 0

Tout le reste vaut `True`.

Exercices :

1. Décrivez le plus clairement et le plus complètement possible ce qui se passe à chacune des trois lignes de l'exemple ci-dessous :
 >>> largeur = 20
 >>> hauteur = 5 * 9.3
 >>> largeur * hauteur
 930
2. Assignez les valeurs respectives 3, 5, 7 à trois variables a, b, c. Effectuez l'opération a - b/c.

Opérateurs et expressions

On manipule les valeurs et les variables qui les référencent, en les combinant avec des *opérateurs* pour former des *expressions*.

Exemple :

$a, b = 7.3, 12$

$y = 3*a + b/5$

Opérateurs et expressions

Opérateurs sur les nombres

Opérateur	Opération	Opérateur	Opération
+	somme	-	différence
*	produit	/	division numérique
//	division Euclidienne	%	reste de la division (modulo)
**	puissance		

Opérateurs logiques

Opérateur	Opération	Opérateur	Opération	Opérateur	Opération
<i>ET</i>	et	<i>OR</i>	ou	<i>NOT</i>	négation

Opérateurs sur les chaînes de caractères

Opérateur	Opération	Opérateur	Opération
+	concaténation	*	répétition

Opérateurs de comparaison

Opérateur	Opération	Opérateur	Opération
==	égal	!=	différent
>	strictement supérieur	<	strictement inférieur
>=	supérieur ou égal	<=	inférieur ou égal

Priorité des opérateurs (PAMDAS)

- **P** pour *parenthèses*. Ce sont elles qui ont la plus haute priorité. Elles vous permettent donc de « forcer » l'évaluation d'une expression dans l'ordre que vous voulez. Ainsi $2*(3-1) = 4$, et $(1+1)**(5-2) = 8$.
- **E** pour *exposants*. Les exposants sont évalués avant les autres opérations. Ainsi $2**1+1 = 3$ (et non 4), et $3*1**10 =$
- **M** et **D** pour *multiplication* et *division*, qui ont la même priorité. Elles sont évaluées avant *l'addition A* et la *soustraction S*, lesquelles sont donc effectuées en dernier lieu. Ainsi $2*3-1 = 5$ (plutôt que 4), et $2/2-1 = 0$.
- Si deux opérateurs ont la même priorité, l'évaluation est effectuée de gauche à droite. Ainsi dans l'expression $30*100/60$, la multiplication est effectuée en premier, et la machine doit donc ensuite effectuer $3000/60$, ce qui donne 50.

Exercices :

1. Décrivez le plus clairement et le plus complètement possible ce qui se passe à chacune des trois lignes de l'exemple ci-dessous :
 >>> largeur = 20
 >>> hauteur = 5 * 9.3
 >>> largeur * hauteur
 930
2. Assignez les valeurs respectives 3, 5, 7 à trois variables a, b, c. Effectuez l'opération a - b/c.

Les fonctions prédéfinies : print et input

print

- La fonction `print(ch)` permet d'afficher ce qui est entre parenthèse sur la sortie standard (l'écran).

Exemple : `print('Bonjour !')`

`age=18`

`print('Vous avez',age,'ans')`

input

- La fonction `input(str)` permet à l'utilisateur de saisir une valeur au clavier

Exemple : `valeur= input ('Entrez votre année de naissance : ')`

- la chaîne 'Entrez votre année de naissance :' est affichée à l'écran
- le programme attend que soit rentrée une chaîne
- le programme affecte cette valeur à la variable `valeur`
- la fonction `input` renvoie toujours une chaîne de caractères
- on a donc besoin de convertir en entier : `annee=int(valeur)`

L'une des grandes forces d'un langage de programmation de haut niveau est qu'il permet de **construire des instructions complexes par assemblage de fragments divers**. Ainsi par exemple, si vous savez comment additionner deux nombres et comment afficher une valeur, vous pouvez combiner ces deux instructions en une seule :

Exemple

Ecrire un script qui va saisir une date (heure, minutes et secondes) et qui va calculer le nombre de secondes écoulées depuis minuit

L'une des grandes forces d'un langage de programmation de haut niveau est qu'il permet de **construire des instructions complexes par assemblage de fragments divers**. Ainsi par exemple, si vous savez comment additionner deux nombres et comment afficher une valeur, vous pouvez combiner ces deux instructions en une seule :

Exemple

```
>>> h, m, s = 15, 27, 34
```

```
>>> print("nombre de secondes écoulées depuis minuit = ", h*3600 + m*60 + s)
```

```
nombre de secondes écoulées depuis minuit = 55654
```


Instructions conditionnelles

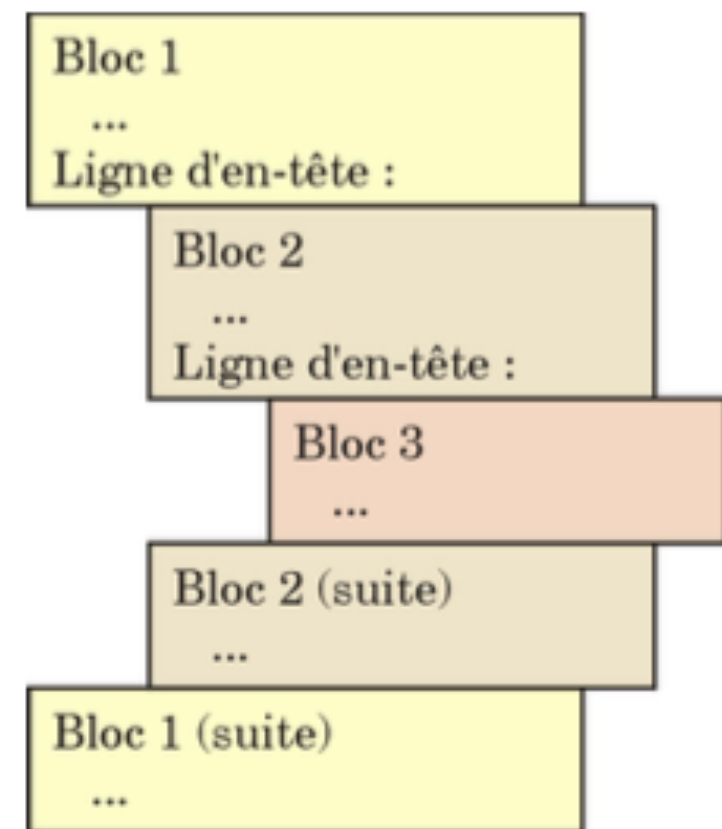
- Elles permettant d'aiguiller le déroulement du programme dans différentes directions, en fonction des circonstances rencontrées.
- Disposer d'instructions capables de tester une certaine condition et de modifier le comportement du programme en conséquence.
- Une instruction simple tient sur une seule ligne. Exemple : $x=10$
- Une instruction composée se compose d'une ligne d'en-tête terminée par : et d'un bloc d'instructions indenté par rapport à la ligne d'en-tête.
- L'instruction **if** (SI en Algorithmique) est une instruction composée.

Instructions conditionnelles

Quelques règles de syntaxe :

- Les limites des instructions et des blocs sont définies par la mise en page
Avec Python, vous devez utiliser les sauts à la ligne et l'indentation.
- Instruction composée = En-tête , double point , bloc d'instructions indenté.

Les blocs d'instructions sont toujours associés à une ligne d'en-tête contenant une instruction bien spécifique (if, elif, else, while, def, ...) se terminant par un double point



- Les espaces et les commentaires sont normalement ignorés

Instructions conditionnelles

Condition if

if **expression** :

instruction 1 du if

instruction 2 du if

...

- **expression** renvoie un booléen (True ou False)
- les instructions du bloc du if sont effectuées uniquement si l'expression est évaluée à True
- dans tous les cas, le programme reprend à l'instruction après if
- On peut mettre autant de conditions que l'on veut dans un if. Exemple :
if $a < b$ and $(b == 2 * c \text{ or } b == 3 * c)$:
- Ces conditions sont évaluées dans l'ordre où elles sont écrites
- Si la première condition d'un and est fausse, alors la suite n'est pas évaluée (le cas de $a = 2$ et $b = 1$ dans l'exemple).

Instructions conditionnelles

Condition if : exemple

```
1 # -*- coding: utf-8 -*-
2 nom = input("Entrez votre nom ?")
3 age = input("Entrez votre age ?")
4 age=int(age) #convertit la valeur saisie en entier
5 print("Bonjour", nom, "!")
6 if age>17:
7     print("Vous êtes majeur(e).")
8 print('Fin.')
```

En interactif

```
>>> a = 150
>>> if (a > 100):
...     print("a dépasse la centaine" )
...
.
```

Tabulation obligatoire



Frappez encore une fois <Enter>.
Le programme s'exécute, et vous obtenez :
a dépasse la centaine.

Instructions conditionnelles

Condition if - else

if **expression** :

instruction 1 du if

...

else :

instruction 1 du else

...

En interactif

```
>>> a = 20
```

```
>>> if (a > 100):
```

```
...     print("a dépasse la centaine")
```

```
... else:
```

```
...     print("a ne dépasse pas cent »)
```

```
...
```

Frappez <Enter> encore une fois.
Le programme s'exécute, et
affiche cette fois : **a ne dépasse
pas cent.**

Instructions conditionnelles

- Il est possible d'imbriquer plusieurs instructions conditionnelles
- L'imbrication nécessite une indentation de blocs

En interactif

```
>>> a = 0
>>> if (a > 0):
...     print("a est positif ")
... elif a<0 :
...     print("a est négatif " )
... else
...     print("a est nul" )
```

Condition if - else

```
if expression :
    instruction 1 du if
    ...
elif :
    instruction 1 du elif
    ...
else
    instruction 1 du else
    ...
```

Instructions répétitives

- `while` (TANT QUE) et `for` (POUR) permettent de boucler sur un traitement
- `break` sert à interrompre une boucle.
- `continue` sert à court-circuiter une boucle (passer directement à l'itération suivante).

Boucler avec : `while`

`While` `condition` :

instruction 1 de `while`

instruction 2 de `while`

...

- les instructions du bloc du `while` sont effectuées uniquement si la condition est évaluée à `True`.
- une fois sorti de la boucle, le programme reprend à l'instruction après `while` fin

Instructions répétitives

Parcourir avec : for

```
for i in range(1,10):           #pour i allant de 1 à 9 avec pas de 1
    instruction 1 de for
    instruction 2 de for
    ...
```

- `range(1,5)` : génère 4 entiers de 1 à 4
- `range(5)` : génère 5 entiers de 0 à 4 (idem que `range(0,5)`)
- `range(1,10,2)` : génère des entiers à partir de 1 avec un pas de 2
- `for` fonctionne en général avec les listes introduites dans la partie 2 du cours.

Exemple

```
print('Les 10 premiers multiple de 3 :')
```

```
for i in range(1,11):
```

```
    print(i * 3, end=" ") # nombres affichés espacés sur la même ligne
```

Exercices :

1. Écrivez un programme qui affiche les 20 premiers termes de la table de multiplication par 7
2. Écrivez un programme qui affiche une table de conversion de sommes d'argent exprimées en euros, en dollars canadiens. La progression des sommes de la table sera « géométrique », comme dans l'exemple ci-dessous :
 - 1 euro(s) = 1.65 dollar(s)
 - 2 euro(s) = 3.30 dollar(s)
 - 4 euro(s) = 6.60 dollar(s)
 - 8 euro(s) = 13.20 dollar(s)
 - etc. (S'arrêter à 16384 euros)
3. Écrivez un programme qui affiche une suite de 12 nombres dont chaque terme soit égal au triple du terme précédent..

Exercices

1. Ecrire un programme qui test si un nombre est paire.
2. Ecrire un programme qui calcule le volume d'un parallélépipède rectangle dont sont fournis au départ la largeur, la hauteur et la profondeur.
3. Ecrire un programme qui convertisse un nombre entier de secondes fourni au départ, en un nombre d'années, de mois, de jours, de minutes et de secondes. (Utilisez l'opérateur modulo : %).
4. écrire un programme qui affiche la suite de Fibonacci, i.e. une suite de nombre dont chaque terme et la somme des deux précédents. (le premier terme est 1)
5. Ecrire un programme qui affiche les 20 premiers termes de la table de multiplication par 7, en signalant au passage (à l'aide d'une astérisque) ceux qui sont des multiples de 3.
Exemple : 7 14 21 * 28 35 42 * 49
6. Ecrivez un programme qui calcule les 50 premiers termes de la table de multiplication par 13, mais n'affiche que ceux qui sont des multiples de 7.
7. 4 Ecrivez un programme qui affiche la suite de symboles suivante :

```
*  
**  
***  
****  
*****  
*****  
*****
```