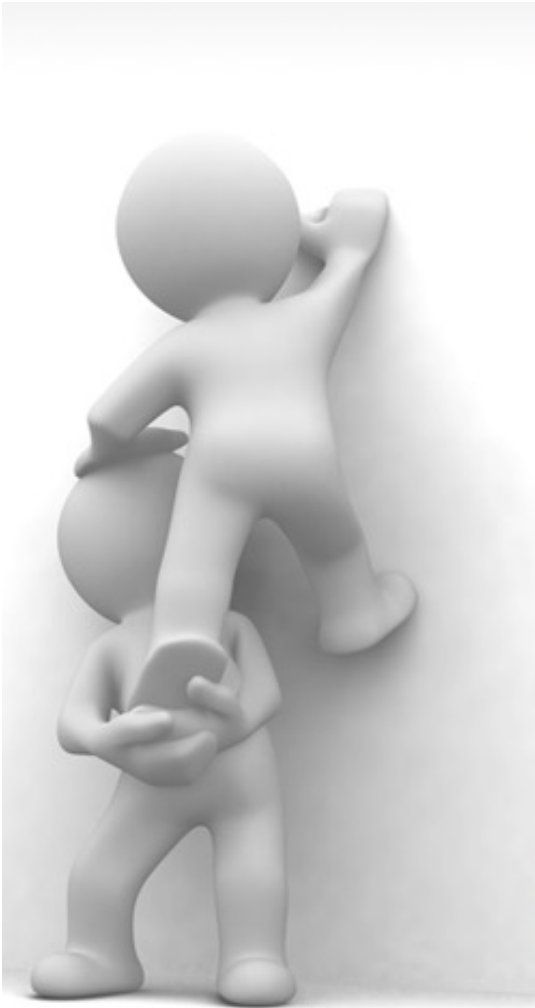


L3 Informatique - 2021-2022

Atelier 1 - Fondamentaux Python



Objectifs de ce cours



- Introduire le langage python et ses principales spécificités
- Vous donner les connaissances minimales pour réaliser les exercices de l'atelier 1
 - Variables et types de base
 - Instructions de saisie - affichage
 - Règles de présentation
 - Structures conditionnelles



Introduction au langage python



N'oubliez pas les futures "Battle" de vocabulaire...

Python: un langage multi-paradigme

Python (1990- Guido van Rossum)

- Programmation essentiellement impérative
- Mais aussi:
 - Programmation orientée objet
 - Définition de classes
 - Héritage
 - Programmation fonctionnelle
 - Listes en compréhension
 - Paramètres de type fonction



Objectifs et spécificités de Python

- Simplicité et puissance
- Lisibilité du code
- Développement rapide d'applications
: Langage interprété
- Typage dynamique



Présentation des programmes



- Une seule instruction par ligne
 - mais possibilité de poursuivre une instruction sur une nouvelle ligne en utilisant le caractère \ en fin de ligne
- Pas de parenthèses, l'**indentation** définit la structure des instructions
 - Utiliser des indentations de 4 espaces: vous pouvez utiliser la touche tabulation mais ce n'est pas obligatoire
- Les **commentaires** débutent par le caractère #



Variables et types en python

Notion de variable

- Une variable est caractérisée par :

- un **identificateur** (nom de la variable):

- Contient des lettres, des chiffres, des caractères spéciaux
 - ne peut pas commencer par un chiffre.
 - minuscules et majuscules sont différenciées.

Python est dit
« sensible à la
casse »

Pour les conventions de nommage cf. PEP 8

- un **type** qui caractérise:

- la nature des informations qui peuvent être stockées dans la variable
 - Les opérations que l'on peut faire sur la variable

Même si le type n'est pas défini explicitement!

Qu'est-ce qu'un type?

- Ensemble de **valeurs**

Ensemble des nombres entiers, réels, ensemble des caractères, {True, False} ...

- Ensemble **d'opérations** pouvant être appliquées sur ces valeurs

- **Format** de représentation des valeurs en mémoire

Exemple Type entier

Valeurs: ensemble des entiers relatifs

Opérations: opérations arithmétiques +, -, *, / ...

Format mémoire : 4 octets

Différents types de types...

■ Types primitifs

- Composés de valeurs primitives **ne pouvant être décomposées** en valeurs plus simples
 - booléen, entier, réel, caractère
- Tous les langages proposent un ensemble de types primitifs natifs (Built-in)

■ Types structurés

- Composés de plusieurs valeurs (de type primitif ou structuré)
- On parle aussi de types composites ou construits

Types en python

- Python est dit à **typage dynamique**
 - Les types des variables ne sont pas déclarés explicitement
 - Le type d'une variable est déduit (« inféré ») pas l'interpréteur selon les types de valeurs qui lui sont affectées
 - Le type d'une variable peut évoluer dynamiquement au cours de l'exécution

```
x=10          #x est de type int
y=10.5        #y est de type float
z=True        #z est de type bool
w="bonjour"   #w est de type str
x=10+1.5      #x est de type float
```

Principaux Types natifs python

- Type entier : **int**
- Type réel : **float**
- Type complexe : **complex**
- Type booléen : **boolean**
 - 2 valeurs possibles: True et False

La fonction `type(var)` renvoie le type de la variable `var`

```
x=10  
print(type(x)) #affiche <class 'int'>
```

Quelques fonctions et opérateurs mathématiques

- x / y : quotient ; $x // y$: quotient entier ; $x \% y$: reste de x / y
- `divmod(x,y)` : la paire $(x // y, x \% y)$
- `math.floor(-7.6)` partie entière, donne ici `-8.0` ;
- `int(math.floor(4.5))` pour avoir l'entier 4.
- `math.exp(2)` : exponentielle.
- `math.log(2)` : logarithme en base naturelle.
- `math.sqrt` : racine carrée.
- `math.pow(4, 5)` ; `4**5` ; `pow(4,5)` : 4 à la puissance 5.
- `math.fmod(4.7, 1.5)` : modulo, ici `0.2`. Préférer cette fonction à `%` pour les flottants.
- `math.isnan(x)` : teste si `x` est nan (Not a Number) et renvoie `True` si c'est le cas.
- `random.random()` : renvoie un nombre aléatoire entre 0 et 1 exclu
- `random.random(0,3)` : renvoie un nombre aléatoire entre 0 et 3 exclu

A placer au début de votre fichier .py

```
import math  
import random
```

Chaines de caractères (version ultra-simplifiée)

- type str
- Littéraux entre quotes ' ou " :

```
phrase= "Bonjour"  
print(type(phrase))  
#affiche <class 'str'>
```

- Opérateur de concaténation

```
phrase2= phrase + " Monsieur"  
#phrase2= "Bonjour Monsieur"
```

- Accès au ième caractère

```
print(phrase[0])  
#affiche j
```

Nous y reviendrons
en détail plus tard...

Fonctions associées au codage des caractères

- **ord**(c) renvoie le nombre entier représentant le code du caractère c.
 - ord('a') 97
 - ord('M') renvoie 77
- **chr**(i) renvoie la chaîne de caractères dont le code est le nombre entier i.
 - chr(97) renvoie 'a'
 - chr(77) renvoie 'M'

Table des codes ASCII (extrait)

Symbol	Decimal	Symbol	Decimal
A	65	a	97
B	66	b	98
C	67	c	99
D	68	d	100
E	69	e	101
F	70	f	102
G	71	g	103
H	72	h	104
I	73	i	105
J	74	j	106
K	75	k	107
L	76	l	108
M	77	m	109
N	78	n	110
O	79	o	111
P	80	p	112
Q	81	q	113
R	82	r	114
S	83	s	115
T	84	t	116
U	85	u	117
V	86	v	118
W	87	w	119
X	88	x	120
Y	89	y	121
Z	90	z	122

Pour plus de détails sur les fonctions natives python:
<https://docs.python.org/fr/3/library/functions.html>

Quelques méthodes de manipulation de chaînes de caractères

- Il s'agit de « méthodes » de la classe str. Pour les utiliser, il faut les appliquer sur une variable de type str.
- **isalpha()**: booléen
 - renvoie True si la chaîne n'est composée que de caractères alphabétiques (lettres)

```
phrase="Bonjour"  
if phrase.isalpha() :  
    print( "ok que des lettres!")
```

Quelques méthodes de manipulation de chaînes de caractères

- **isdecimal()**: booléen
 - renvoie True si la chaîne n'est composée que de caractères décimaux(chiffres)
- **isdigit()** et **isnumeric()** fonctionnent de manière similaire mais incluent également les nombres exprimés avec des caractères spéciaux
 - ex: '\u00BD' index caractère unicode ½

Quelques méthodes de manipulation de chaînes de caractères

- **isspace()** : booléen
 - renvoie True si la chaîne n'est composée que d'espaces
- **upper()** : str (**lower()** pour mettre en minuscules)
 - Renvoie une chaîne dans laquelle les caractères présents dans la chaîne apparaissent en majuscules

```
phrase="Bonjour"  
phraseMaj=phrase.upper()  
print(phraseMAJ) #affiche BONJOUR
```

Fonctions de conversions de type

- `int(valeur)`

Attention erreur si la
conversion est impossible

- Convertit la valeur (type chaîne de caractères, réel,...) transmise en paramètre en nombre entier

```
x= int("44") #x=44  
Y= int("bonjour ») #ERREURc
```

- `float(valeur)`

- Convertit la valeur (type chaîne de caractères, entier,...) transmise en paramètre en nombre réel

- `str(valeur)`

- Convertit la valeur (type chaîne de caractères, réel,...) transmise en paramètre en chaîne de caractères

Notion de littéral

- Un littéral est l'écriture d'une valeur d'un certain type
- Les littéraux peuvent être utilisés :
 - En partie droite d'une affectation
 - Dans des expressions (notamment des conditions)
- Littéraux numériques
 - Un littéral de type entier est une suite de chiffres
 - Un littéral de type réel est une suite de chiffres avec éventuellement la présence d'un .
- Un littéral de type chaîne de caractères est une suite de caractères **entre quotes**

Règles de nommage

Pour les conventions de nommage cf. PEP 8

■ Nommage des variables

- `minuscules_separees_par_des_unders
core`
- Doivent vouloir dire quelque chose sans être trop long
- Éviter le l (L minuscule), et O (O majuscule) et seuls, qui peuvent se confondre avec un 1 et un 0 selon les polices

■ Nommage des constantes

- `MAJUSCULES_SEPAREES_PAR_DES_UNDERS
CORE`

Constantes en python

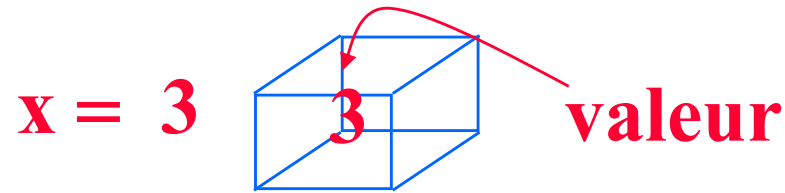
- Une constante est une variable qui a une valeur qui n'est jamais modifiée après son initialisation.
- Il n'y a **pas de mot clé spécifique** pour définir des constantes en python.
- Intérêt des constantes
 - Lisibilité
 - Facilité de maintenance



Instructions élémentaires

- Affectations
- Affichage
- Saisie clavier

Affectation (=)



Variable x de type ENTIER

L' **expression en partie droite** peut être:

- une **variable (ou constante) de même type** (ou type compatible) que la variable x
- une **valeur littérale de même type** (ou type compatible) que la variable x
- une **expression** dont l'évaluation produit un résultat de même type (ou type compatible) que la variable x.

Par exemple:

expression arithmétique si variable1 est de type réel

expression booléenne si variable1 est de type booléen

Affectations combinées

- L'affectation peut être combinée avec des opérateurs

- Addition puis affectation

$x += \text{valeur}$ est équivalent à $x = x + \text{valeur}$

- Autres opérations puis affectation

$x -= \text{valeur}$, $x *= \text{valeur}$, $x /= \text{valeur}$

Affichage: print

- La fonction print affiche une chaîne de caractère transmise en paramètre et passe à la ligne

```
print("Il fait beau")
print("Bonjour ", nom) #ou
    print("Bonjour", nom) + nom)
print("Votre age : ", age) #ou
    print("Votre age : " + str(age))
```

- Pour éviter le passage à la ligne:

```
print("Il fait beau", end="" )
```


Saisie clavier: input

- La fonction input demande à l'utilisateur de saisir au clavier une chaîne de caractères et renvoie la chaîne saisie

```
prenom = input("Entrez votre prénom : ")  
print("Bonjour", prenom)
```

- Attention!
 - La fonction renvoie toujours une chaîne de caractères si l'on doit saisir un nombre, il faut effectuer une conversion de type



```
age = int(input("Entrez votre age : "))  
note = float(input("Entrez votre note : "))
```



Structures conditionnelles

Schémas conditionnels

if *condition* :
 actions1

Si condition est évaluée à vrai, actions1 seront exécutées, si condition est évaluée à faux, aucune action n'est exécutée.

if *condition* :
 actions1
else :
 actions2

Si condition est évaluée à vrai, actions1 seront exécutées et actions2 seront ignorées, si condition est évaluée à faux, actions2 seront exécutées et actions1 seront ignorées.

Condition est un prédicat =
expression dont l'évaluation a pour
résultat la valeur True ou False

**PREDICATS
SIMPLES**


**PREDICATS
COMPOSES**

Prédicats simples

- variables booléennes
- comparaison entre 2 valeurs (constantes, variables ou expressions) de même type
- appel de fonction booléenne

```
def exemple_predicat():  
    #Exemples de schémas conditionnels avec prédicats simples  
    notemath = float(input("Note de maths: "))  
    noteinfo = float(input("Note d'informatique: "))  
    moyenne = (notemath + noteinfo) / 2  
    if notemath > 10 :  
        okmath = True  
    else:  
        pkmath = False  
    if moyenne < 10 :  
        print("Attention, Moyenne insuffisante")  
        if okmath:  
            print("Mais Bien en Maths! ")  
        else:  
            print("Moyenne supérieure à 10")
```

Test d'une variable booléenne
Equivalent à
if okmath==True :



Principaux opérateurs de comparaison

- ==
 - égalité (pour des nombres ou des chaînes)
- !=
 - inégalité (pour des nombres ou des chaînes).
- > >= < <=
 - comparaison

Il existe d'autres opérateurs de comparaison spécifiques des objets, nous y reviendrons...

Prédicats composés

- Prédicats définis par un ou plusieurs prédicats simples (et/ou composés) reliés entre eux par des connecteurs logiques.

Connecteur unaire NON (NOT)

P est un prédicat

Si P est vrai \implies NOT P est faux
Si P est faux \implies NOT P est vrai



Connecteurs binaires

- ET (AND)
- OU (OR)

Tables de Vérité

P	Q	P and Q	P or Q
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

Attention !!

La notation $X < J < P$ est autorisée en python mais interdite dans la plupart des langages

Il est préférable d'écrire
 $X < J$ and $J < P$

Prédicats composés

```
def exemple2_predicat():  
    """ Exemples de schémas conditionnels avec prédicats composés """  
    # note : réel 'moyenne générale d'un candidat au baccalauréat'  
    note = float( input("Taper la moyenne générale du candidat: "))  
    if note < 8 :  
        print("candidat refusé")  
    if note >= 8 and note < 10 :  
        print("candidat soumis à l'oral de rattrapage")  
    if note >= 10 and note < 12 :  
        print( "candidat admis mention passable")  
    if note >= 12 and note < 14 :  
        print( "candidat admis mention assez-bien")  
    if note >= 14 and note < 16 :  
        print( "candidat admis mention bien")  
    if note >= 16 :  
        print( "candidat admis mention très-bien")
```

Ce programme n'est pas du tout optimisé au niveau temps d'exécution !!!

Pourquoi ?

Comment peut-on l'améliorer ?

Prédicats composés

```
def exemple3_predicat():
#solution optimisée
    note = float( input("Taper la moyenne générale du candidat: "))
    if note < 8 :
        print("candidat refusé")
    else:#note >= 8
        if note<10 :
            #note >= 8 et note <10
            print("candidat soumis à l'oral de rattrapage")
        else: #note >=10
            if note < 12 :
                #note >= 10 et note < 12
                print( "candidat admis mention passable")
            else: #note>=12
                if note < 14 :
                    #note>=12 et note <14
                    print( "candidat admis mention assez-bien")
                else: #note >=14
                    if note < 16 :
                        #note>=14 et note <16
                        print( "candidat admis mention bien")
                    else: #note >=16
                        print( "candidat admis mention très-bien")
```

Prédicats composés

```
def exemple4_predicat():
#solution optimisée
    note = float( input("Taper la moyenne générale du candidat: "))
    if note < 8 :
        print("candidat refusé")
    elif note<10 :
        #note >= 8 et note <10
        print("candidat soumis à l'oral de rattrapage")
    elif note < 12 :
        #note >= 10 et note <12 Équivalent à la séquence
        print( "candidat admis mention passable")
    elif note < 14 :
        #note>=12 et note <14
        print( "candidat admis mention assez-bien")
    elif note < 16 :
        #note>=14 et note <16
        print( "candidat admis mention bien")
    else: #note >=16
        print( "candidat admis mention très-bien")
```

Structure
conditionnelle
if....:
elif....:

Variables booléennes et prédicats

- Un prédicat (simple ou composé) peut-être affecté à une variable booléenne

```
age=int(input("Entrez votre age : "))  
resultat=(age<15)  
if resultat :  
    print( "OK vous êtes majeur")
```

Équivalent à la séquence

```
if age<15:  
    resultat=True  
else:  
    resultat=False
```



Réalisation Atelier 1

Consignes Ateliers

- Ecrivez le code de l'atelier dans un fichier documenté
(Auteur/Date/Version/Description, cf. Vidéo IDLE)
- Une fonction par exercice
- Une ligne de commentaire expliquant le but de la fonction (docstring)
- Un commentaire par variable spécifiant en particulier le type de la variable

```
#x de type int  
x=10
```

Réalisation d'un exercice

```
#Auteur : Moi
```

```
#Date : 04/09/2020
```

```
#Version : 1
```

```
#Description : Codes de l'Atelier 1 L3
```

```
#Exercice 1
```

```
def ma_fonction():
```

```
    """ cette fonction
```

```
    effectue .... """
```

```
    instructions
```

```
#appel de la fonction
```

```
ma_fonction()
```

Commentaire spécial (docstrings)
décrivant le but de la fonction

Consignes atelier 1

- Eviter les évaluations de prédicats inutiles dans les schémas conditionnels
- Respecter les conventions de nommage
- Respecter les règles de présentation
- Définir des constantes dès que possible

Jeux de tests

- Pour chaque exercice, définissez des jeux de tests pertinents :
 - Exemple : fonction de saisie d'un âge et affichage d'un message selon que la personne est majeure ou non
 - Jeux de test
 - Saisie d'un âge <18
 - Saisie d'un âge ≤ 18
 - Bonus : Saisie d'un âge absurde (négatif ou supérieur à une valeur maximum)