Classe de seconde
Accompagnement personnalisé

Atelier informatique : programmation

Atelier informatique : programmation

Activité n°0 : Introduction

Un **système d'exploitation** (en anglais OS : Operating System) est l'ensemble des programmes d'un appareil informatique (ordinateur, tablette, smartphone...) qui sert d'interface entre le matériel (processeur, mémoire, disque dur, clavier...) et les logiciels d'application (tableur, traitement de texte, messagerie, navigateur web, lecteur multimédia, jeu vidéo...).

Citez 3 systèmes d'exploitation :

Un **programme informatique** est une séquence d'instructions (cela représente le **code source**).

C'est le processeur qui exécute les instructions du programme.

Un **logiciel** est un ensemble composé d'un ou plusieurs programmes.

Un **langage de programmation** sert à produire des programmes informatiques.

Citez 3 langages de programmation :

Une **licence de logiciel** est un contrat qui définit les conditions dans lesquelles ce programme peut être utilisé, diffusé ou modifié.

Pour simplifier, on peut classer les licences en deux catégories :

- les licences propriétaires : la duplication, la modification ou l'usage sont limités.

- les licences libres : l'utilisation, l'étude du code source, la modification et la duplication sont permises.

Cochez les bonnes réponses :

Logiciel	Licence propriétaire	Licence libre / Open Source	Logiciel	Licence propriétaire	Licence libre / Open Source
Windows 8			Mac OS		
Linux			Android		
Word			Excel		
LibreOffice			VLC		
Firefox			Internet Explorer		

Les logiciels propriétaires sont généralement payants, et les logiciels libres sont souvent gratuits.
Attention : gratuit n'est pas synonyme de libre !

Nous allons apprendre les bases de la programmation en utilisant un langage nommé Python (dans sa version 3).

Avant de commencer réellement notre apprentissage de Python (et donc de la programmation), nous allons devoir nous familiariser avec notre environnement de travail.
Nous allons utiliser l'environnement IDLE (environnement installé par défaut).
Lancez donc IDLE :

Démarrer → Programmes → Python → IDLE (Python GUI)

Dans le menu *File*, choisissez *New Window*, une fenêtre devrait alors apparaître.

Traditionnellement, les apprentis programmeurs commencent leur carrière en écrivant un programme qui permet d'afficher à l'écran le message *Hello World !*
Nous n'allons pas déroger à cette tradition.
En langage Python, il suffit d'une seule instruction pour afficher ce message (notez bien que selon le langage utilisé cela peut être plus complexe) :

```
print("Hello World !")
```

Recopier cette ligne dans la fenêtre d'IDLE et lancer l'exécution du programme en appuyant sur la touche F5 (Run → Run Module).
IDLE va vous demander d'enregistrer le programme :

- Répertoire : un dossier du bureau de l'ordinateur qui vous servira de dossier de travail
- Nom du fichier : **programme1.py** (n'oubliez pas l'extension .py)

En fin de séance, vous penserez à faire une sauvegarde sur votre clé USB !

Vous devriez voir le message *Hello World !* apparaître dans la seconde fenêtre.
Bravo, vous venez d'écrire votre premier programme !
Notez qu'il est possible d'utiliser des apostrophes à la place des guillemets :

```
print('Hello World !')
```

À faire vous même

Essayer d'écrire un programme qui affichera votre prénom à l'écran.
Après avoir testé votre programme avec IDLE, recopier votre code ci-dessous :

Annexe

Ce document est téléchargeable sur le web à l'adresse suivante :
http://fabrice.sincere.free.fr/ressources

L'environnement de développement IDLE de Python est un logiciel libre et gratuit que vous pouvez télécharger ici : **http://www.python.org/getit**

Définition du mot ordinateur d'après le dictionnaire *Le Petit Larousse* :
« Machine automatique de traitement de l'information, obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques. »

Qui dit traitement de l'information, dit donc données à manipuler. Un programme passe donc son temps à traiter des données. Pour pouvoir traiter ces données, l'ordinateur doit les ranger dans sa mémoire (RAM Random Access Memory). La RAM se compose de cases dans lesquelles nous allons ranger ces données (une donnée dans une case).

Alors, qu'est-ce qu'une variable ?
Eh bien, c'est une petite information (une donnée) temporaire que l'on stocke dans une case de la RAM. On dit qu'elle est "variable" car c'est une valeur qui peut changer pendant le déroulement du programme.
Une variable est constituée de 2 choses :

- Elle a une valeur : c'est la donnée qu'elle stocke (par exemple le nombre 12)
- Elle a un nom : c'est ce qui permet de la reconnaître :

```
i = 12
print(i)
```

Grâce à cette ligne, nous avons défini une variable qui porte le nom i et qui contient la valeur 12.

À faire vous même

À l'aide de IDLE, testez le code suivant :

```
point_de_vie = 15
print(point_de_vie)
```

Comme vous pouvez le constater, la fonction `print()` permet d'afficher la valeur contenue dans la variable `point_de_vie`.
J'attire votre attention sur l'absence de guillemets, vous pouvez retester cet exemple avec :
`print("point_de_vie")`

Alors, que se passe-t-il ?

À faire vous même

Écrire un programme dans lequel on attribut la valeur 18 à la variable `pointDeForce`.
La valeur de `pointDeForce` doit ensuite s'afficher à l'écran.

Après avoir testé votre programme avec IDLE, recopier votre code ci-dessous :

Un ordinateur est bien évidemment capable d'effectuer des opérations arithmétiques et mathématiques.
Les signes utilisés sont classiques : + (addition), - (soustraction), * (multiplication) et / (division).
Il est tout à fait possible d'effectuer des opérations directement avec des nombres, mais il est aussi possible d'utiliser des variables.

À faire vous même

D'après vous, que fait ce programme ?

```
a = 15
b = 4
somme = a + b
print(somme)
```

Réponse :

Vérifier votre réponse en l'exécutant à l'aide de IDLE.

Écrire un programme qui multiplie le contenu de 2 variables (nom des variables : `c` et `d`).
Le résultat de cette opération devra être rangé dans une troisième variable (`resultat`).
Votre programme devra afficher le contenu de la variable `resultat`.
Après avoir testé votre programme avec IDLE, recopier votre code ci-dessous :

À faire vous même

D'après vous, que fait ce programme ?

```
a = 11
print(a)
a = a + 1
print(a)
```

Réponse :

Vérifier votre réponse en l'exécutant à l'aide de IDLE.

Détaillons ce qui se passe :
Nous créons une variable `a` et nous lui attribuons la valeur 11.
Nous affichons à l'écran la valeur de `a` (c'est à dire 11).
La suite est un peu plus complexe, mais très importante à comprendre.
Il va falloir lire la ligne `a = a + 1` de droite à gauche, décortiquons cette ligne :
`a + 1` : nous prenons la valeur actuelle de `a` (c'est-à-dire 11) et nous ajoutons 1 à 11, à droite de l'égalité nous avons donc maintenant la valeur 12.
Nous attribuons la valeur qui vient d'être calculée à la variable `a` (donc maintenant `a` vaut `12`).
Nous affichons à l'écran la nouvelle valeur de `a`
Ce raisonnement peut être généralisé pour éviter des erreurs parfois difficiles à corriger :
Dans une égalité, commencer toujours par évaluer l'expression se trouvant à droite du signe égal.

Les variables ne contiennent pas forcément des nombres, elles peuvent aussi stocker des suites de caractères, on parle alors de chaîne de caractères.

À faire vous même

Tester le code suivant :

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

Les variables peuvent donc contenir des types de données différents, pour l'instant nous en avons vu deux :
- le type « nombre entier » (`int` de integer en anglais)
- le type « chaîne de caractères » (`str` de string en anglais)

Il existe d'autres types de variables :
- le type « nombre à virgule flottante » : `float`
- le type « booléen » : `bool`
- etc...

En Python les variables ont un type, mais le programmeur n'est pas obligé de préciser ce type.

Il existe beaucoup de langage (C++, Java....) où l'utilisateur doit absolument définir le type d'une variable avant de pouvoir l'utiliser, faute de quoi cela entraînera une erreur.

La fonction `type()` vous permet de connaître le type d'une variable.

À faire vous même

Tester ce programme :

```
a = "Salut !"
b = 36
c = 5.87
d = '42'
print(type(a))
print(type(b))
print(type(c))
print(type(d))
```

Quel est le résultat ?

L'utilisation du signe + ne se limite pas à l'addition. Il est aussi utilisé pour la concaténation.
Le terme *concaténation* désigne l'action de mettre bout à bout au moins deux chaînes.

Comme vous avez pu le deviner en lisant la définition ci-dessus, la concaténation va concerner les chaînes de caractères.

À faire vous même

```
a = "Hello"
b = " World !"
monExpression = a + b
print(monExpression)
```

Qu'est-ce que ce programme va permettre d'afficher à l'écran ?

Soit une variable `a` de type `str`, une variable `b` de type `int` et une variable `c = a + b`
Dans ce cas précis, le signe + est-il le signe de la concaténation ou de l'addition ?
La variable `c` est-elle de type `str`, de type `int` ou d'un autre type ?

À faire vous même

Écrire un programme permettant de répondre à ces questions.

Répondre aux questions en exécutant le programme.

Comme vous pouvez le constater votre programme renvoie une erreur.
En Python, il n'est pas toujours possible d'utiliser le signe + avec des variables de type différent.

Ne trouvez-vous pas que pour l'instant cela manque un peu d'interactivité ?
En effet, l'utilisateur de vos programmes est plutôt passif !
Heureusement la fonction `input()` va permettre aux utilisateurs de rentrer des données.

À faire vous même

Tester ce programme :

```
age = input("Quel est votre âge ?")
print(age)
```

La variable `age` va contenir la réponse entrée au clavier par l'utilisateur.

À faire vous même

Écrire un programme qui demande son prénom à l'utilisateur puis l'affiche sous la forme *Bonjour, Toto* (si Toto est le prénom entré par l'utilisateur !)

À faire vous même

Écrire un programme qui demande à l'utilisateur d'entrer 2 nombres et affiche le résultat de l'addition de ces 2 nombres.

Problème : si l'utilisateur entre *25* puis *12* le résultat affiché est *2512* !
Avez-vous une idée du problème ?
Pour vous aider dans votre recherche, compléter votre programme pour qu'il affiche le type des deux variables d'entrée.

Alors ?

Les valeurs obtenues grâce à la fonction `input()` sont forcément de type `str` (chaîne de caractères).
Nous avons donc ici une concaténation et non pas une addition : '25' + '12' donne '2512'

La fonction `int()` permet de convertir une variable de type `str` en une variable de type `int` :

```
a1 = int(a)
# avec a de type str et a1 de type int
```

Modifier votre programme pour qu'il fonctionne normalement (25 + 12 doit donner 37) :

Remarques

La fonction `float()` permet de convertir une variable de type `str` en une variable de type `float` (nombre à virgule) :

```
d1 = float(d)
# avec d de type str et d1 de type float
```

Si quelqu'un vous dit que « 4 est égal à 5 », vous lui répondez quoi ? C'est faux !
Si maintenant la même personne vous dit que « 7 est égal à 7 », vous lui répondrez bien évidemment que c'est vrai.
En Python, ces deux expressions (« 4 est égal à 5 » et « 7 est égal à 7 ») s'écriront `4==5` et `7==7` (notez bien le double signe égal).

À faire vous même

Tester le programme suivant :

```
print(4==5)
print(7==7)
```

Quel est le résultat attendu après l'exécution de ce programme ?

Le double égal (==) est l'opérateur d'égalité.
L'égalité est soit vraie (True) soit fausse (False).
L'utilisation de l'opérateur d'égalité va prendre tout son sens avec des variables.

À faire vous même

Soit le programme suivant :

```
a = "Paul"
b = "Pierre"
print(a==b)
a = "Pierre"
print(a==b)
```

Quel est le résultat attendu après l'exécution de ce programme ?

ATTENTION : Il ne faut pas confondre l'opérateur d'égalité (==) et l'opérateur d'affectation (=) utilisé pour attribuer une valeur aux variables. La confusion entre ces 2 opérateurs est une erreur classique qu'il est parfois très difficile à détecter !

Il est possible d'utiliser aussi l'opérateur « différent de » !=

À faire vous même

Soit le programme suivant :

```
a = "Paul"
b = "Pierre"
print(a!=b)
a = "Pierre"
print(a!=b)
```

Quel est le résultat attendu après l'exécution de ce programme ?

Notez aussi l'existence des opérateurs :
- « strictement inférieur à » <
- « strictement supérieur à » >
- « inférieur ou égal à » <=
- « supérieur ou égal à » >=

A chaque fois ces opérateurs retournent une valeur de type booléen : True (vrai) ou False (faux).

À faire vous même

Soit le programme suivant :

```
a = 4
b = 4
print(a<b)
a = 7
print(a<b)
```

Quel est le résultat attendu après l'exécution de ce programme ?

À faire vous même

Soit le programme suivant :

```
a = 14
b = 7
print(a>=b)
a = 7
print(a>=b)
```

Quel est le résultat attendu après l'exécution de ce programme ?

Pour terminer, notez qu'une variable qui ne peut contenir que True ou False est de type booléen.

Nous allons maintenant étudier une structure fondamentale en programmation, le « si … alors … sinon ... ».
L'idée de base est la suivante :

si condition :
    bloc d'instructions 1

sinon :
    bloc d'instructions 2

Comment cela fonctionne ?
Si la condition est vraie (True) alors le bloc d'instructions 1 est exécuté et le bloc d'instructions 2 est ignoré.
Sinon (sous-entendu que la condition est fausse) le bloc d'instructions 2 est exécuté et le bloc d'instructions 1 est ignoré.

Notez le décalage vers la droite du bloc d'instructions 1 et du bloc d'instructions 2.
Ce décalage est appelé **indentation** (touche Tabulation du clavier).
L'indentation est obligatoire en langage Python.

À faire vous même

Soit le programme suivant :

```
prenom = 'Toto'
if prenom == 'Toto':
     print('Je suis Toto.')
else:
     print('Il y a erreur.')
     print("Je ne m'appelle pas Toto !")     # ligne 6
print('Au revoir !')
```

Quel est le résultat attendu après l'exécution de ce programme ?

Notez la présence des 2 guillemets " à la place des apostrophes ' dans la ligne 6.
Pourquoi d'après vous ?

À faire vous même

Écrire un programme qui demande l'âge de l'utilisateur.
Si l'utilisateur a 18 ans ou plus, le programme devra afficher *Bonjour, vous êtes majeur*.
Si l'utilisateur a moins de 18 ans, le programme devra afficher *Bonjour, vous êtes mineur*.

Une instruction `if` peut contenir plusieurs conditions, nous aurons alors une structure de la forme :

si condition1 op_logique condition2 :
    bloc d'instructions 1
sinon:
    bloc d'instructions 2

« op_logique » étant un opérateur logique.

Nous allons étudier 2 opérateurs logiques : le « ou logique » (`or`) et le « et logique » (`and`).

Par exemple (condition1 or condition2) est vraie si la condition1 est vraie ou la condition2 est vraie.
Autre exemple (condition1 and condition2) est vraie si la condition1 est vraie et la condition2 est vraie.

À faire vous même

```
note = float(input("Note sur 20 : "))
if note>=10.0 and note<=20.0 :
     print("J'ai la moyenne")
else :
     print("Encore un effort !")
print("Fin du programme")
```

Quel est le résultat attendu après l'exécution de ce programme ?

Est-ce que le programme tient compte des notes supérieures à 20 ou inférieures à 0 ?

Nous allons compléter le programme précédent avec l'instruction `elif` (sinon si) :

```
note = float(input("Note sur 20 : "))
if note>=10.0 and note<=20.0 :
     print("J'ai la moyenne")
elif note<0.0 or note>20.0 :
     print("Note invalide !")
else :
     print("Encore un effort !")
print("Fin du programme")
```

Est-ce que maintenant le programme fonctionne si la note est supérieure à 20 ou inférieure à 0 ?

À faire vous même

Écrire un programme qui demande :
- le tarif normal d'une place de cinéma
- votre âge

Le programme doit ensuite afficher le prix à payer, sachant qu'une réduction de 50 % est accordée pour les plus de 65 ans, et que c'est 4 euros pour les moins de 14 ans.

La notion de boucle est fondamentale en informatique.
Une boucle permet d'exécuter plusieurs fois des instructions qui ne sont présentes qu'une seule fois dans le code.
La structure de la boucle `while` est la suivante :

while condition :
    bloc d'instructions

suite du programme

Tant que la condition reste vraie, les instructions à l'intérieur du bloc (partie indentée) seront exécutées.

À faire vous même

Soit le programme suivant :

```
i = 0
while i<=10:
     print("i vaut :",i)
     i = i + 1
print("C'est terminé")
```

NB : le `"i vaut :",i` permet d'afficher la chaine «`i vaut :` » et la valeur contenue dans la variable `i` sur la même ligne.
Quel est le résultat attendu après l'exécution de ce programme ?

À faire vous même

Écrire un programme permettant d'afficher une table de multiplication.
L'utilisateur entre la table qu'il désire (de 1 à 10), le programme permet alors d'afficher la table demandée.
Par exemple si l'utilisateur demande la table des 3, le programme devra afficher :
1 * 3 = 3
2 * 3 = 6
...
...
10 * 3 = 30

À faire vous même

Écrire le programme du jeu qui consiste à deviner un nombre entre 1 et 100, par exemple :

```
Entrer un nombre :   50
trop petit !
Entrer un nombre :   75
trop petit !
Entrer un nombre :   87
trop grand !
Entrer un nombre :   81
trop petit !
Entrer un nombre :   84
Gagné !
```

Remarque : pour créer un nombre entier aléatoire entre 1 et 100, il faut importer le module `random` puis utiliser la fonction `randint()` :

```
from random import *
nombre = randint(1,100)
```

On peut aussi faire des boucles avec l'instruction `for`
C'est d'ailleurs souvent plus simple qu'avec l'instruction `while`

À faire vous même

La boucle `for ... in` va nous permettre de parcourir les éléments d'une liste d'entiers :

```
for i in range(0,11) :
     print("i vaut :",i)
print("Fin de la boucle")
```

Que fait le programme ?

Soit le programme suivant :

```
for i in range(1,21,2) :
     print("i vaut :",i," i au carré vaut :",i*i)
print("Fin de la boucle")
```

Que fait le programme ?

À faire vous même

Écrire le programme qui affiche toutes les années bissextiles des cent ans à venir : 2016, 2020...
Attention, 2100 est une année commune !

À faire vous même

Écrire un programme qui simule le comportement d'un dé à 6 faces, et qui affiche le résultat de 10 lancers de dé.

L'un des concepts les plus importants en programmation est celui de fonction.
Les fonctions permettent en effet de décomposer un programme complexe en une série de sous-programmes plus simples.

Voici la syntaxe pour définir une fonction :

def NomDeMaFonction (paramètre) :
    bloc d'instructions
    return valeur

Codons par exemple la fonction mathématique $y = x^2$ en créant une fonction `FonctionCarre` :

```
def FonctionCarre(x):
    y = x*x
    return y
```

Pour utiliser la fonction `FonctionCarre`, il suffit d'écrire :
`FonctionCarre(4)` (dans ce cas précis, notre fonction renverra le nombre 16)

À faire vous même

Soit le programme suivant :

```
def FonctionCarre(x):
    y = x*x
    return y

print('5 au carré vaut :',FonctionCarre(5))
print('6 au carré vaut :',FonctionCarre(6))
```

Quel est le résultat attendu après l'exécution de ce programme ?

Une fonction peut ne pas avoir de paramètres.
Elle peut aussi ne rien retourner (pas d'instruction `return`) :

```
from time import *

def Maintenant() :
    print(strftime('%H:%M:%S'))

Maintenant()
```

Quel est le résultat attendu après l'exécution de ce programme ?

Le module *tkinter* de Python permet de créer des interfaces graphiques (GUI : graphical user interface).

De nombreux composants graphiques (ou widgets) sont disponibles : fenêtre (classe *Tk*), bouton (classe *Button*), case à cocher (classe *Checkbutton*), étiquette (classe *Label*), zone de texte simple (classe *Entry*), menu (classe *Menu*), zone graphique (classe *Canvas*)...

On peut également gérer de nombreux événements : clic sur la souris, déplacement de la souris, appui sur une touche du clavier, top d'horloge...

Programme GUI1 : création d'une fenêtre graphique (à tester)

```
from tkinter import *
mafenetre = Tk()
mafenetre.mainloop()
```

Programme GUI2 (à tester)

```
from tkinter import *
mafenetre = Tk()
mafenetre.title('GUI2')
mafenetre.geometry('480x320')
mafenetre.resizable(width = True, height = True)
mafenetre['bg'] ='khaki'
mafenetre.mainloop()
```

A quoi servent les attributs *title, geometry et resizable* ?
Que fait l'option *bg* ?

À faire vous même : programme GUI3

Modifier le programme GUI2 pour obtenir une fenêtre de largeur 640 pixels et de hauteur 360 pixels, non redimensionnable, avec un fond de couleur jaune.

Programme GUI4 : création d'un widget Label (à tester)

```
from tkinter import *
mafenetre = Tk()
mafenetre.title('GUI4')
mafenetre.geometry('480x320')
label1 = Label(mafenetre,text='Bonjour !',fg='red',bg='white')
label1['font']=('Arial',16)
label1.place(x=50,y=100)
mafenetre.mainloop()
```

À faire vous même : programme GUI5

Compléter le programme GUI4 de manière à ajouter le texte « Hello world ! » en haut à gauche de la fenêtre, avec une taille de police de 12.

Programme GUI6 : création d'un widget Button (à tester)

```
from tkinter import *
mafenetre = Tk()
mafenetre.title('GUI6')
mafenetre.geometry('480x320')
label1 = Label(mafenetre,text='Bonjour !',fg='blue')
label1['font']=('Arial',16)
label1.place(x=50,y=50)
bouton1 = Button(mafenetre,text='Clic',width=10,height=2)
bouton1.place(x=50,y=150)
mafenetre.mainloop()
```

À faire vous même : programme GUI7

Compléter le programme GUI6 afin de créer un second bouton « Quitter ».

Programme GUI8

Jusqu'à maintenant, il ne se passe rien quand on appuie sur les boutons.
Nous allons voir comment associer une action à un bouton :

Dans le programme GUI7, modifier le code :
`Button(mafenetre, text ='Quitter')`
par :
`Button(mafenetre, text ='Quitter', command = mafenetre.destroy)`

Maintenant, que se passe-t-il quand on clique sur le bouton « Quitter » ?

Programme GUI9

Nous allons maintenant associer une action au bouton « Clic » :

Commencer par modifier la ligne :

```
bouton1 = Button(mafenetre,text='Clic',width=10,height=2,command=Clic)
```

Puis ajouter en début de programme la fonction :

```
def Clic():
    label1['text'] = 'Vous avez cliqué !'
```

Maintenant, que se passe-t-il quand on clique sur le bouton « Clic » ?

À faire vous même : programme GUI10

Créer un troisième bouton « Effacer » qui permet d'effacer le texte du widget Label.

Programme GUI11 : compteur (à tester)

```
from tkinter import *

def Plus():
    labelCompteur['text'] = labelCompteur['text'] + 1

mafenetre = Tk()
mafenetre.title('Compteur')
mafenetre.geometry('480x320')
labelCompteur = Label(mafenetre,text=0,bg='yellow',font=('Arial',16))
labelCompteur.place(x=50,y=50)
boutonPlus = Button(mafenetre,text='+',command=Plus)
boutonPlus.place(x=50,y=100)
mafenetre.mainloop()
```

À faire vous même : programme GUI12 compteur/décompteur

Reprendre le programme GUI11 et ajouter un bouton « moins » et un bouton « Remise à zéro ».
Attention : le compteur ne doit pas aller dans les valeurs négatives.

Programme GUI13 : création d'un widget Entry (à tester)

```
from tkinter import *

def Clic():
    labelBonjour['text']= 'Bonjour '+ prenom.get()

def Effacer():
    labelBonjour['text']= 'Bonjour'
    prenom.set('')

mafenetre = Tk()
mafenetre.title('GUI13 Bonjour')
mafenetre.geometry('480x320')
labelPrenom = Label(mafenetre,text='Prénom :')
labelPrenom.place(x=50,y=100)
labelBonjour = Label(mafenetre,text='Bonjour',bg='green',font=('Arial',16))
labelBonjour.place(x=50,y=200)

prenom = StringVar()
entryPrenom = Entry(mafenetre,textvariable = prenom)
entryPrenom.place(x=120,y=100)

boutonGo = Button(mafenetre,text='Go',command=Clic)
boutonGo.place(x=50,y=150)

boutonEffacer = Button(mafenetre,text='Effacer',command=Effacer)
boutonEffacer.place(x=150,y=150)

mafenetre.mainloop()
```

À faire vous même : programme GUI14

Reprendre le programme GUI13 de manière à gérer le nom en plus du prénom.

Autres widgets

Des exemples avec d'autres widgets sont à tester ici :
http://fsincere.free.fr/isn/python/cours_python_tkinter.php