

TP n° 3

Consignes les exercices ou questions marqués d'un ★ devront être rédigés sur papier (afin de se préparer aux épreuves écrites du partiel et de l'examen). En particulier, il est recommandé d'être dans les mêmes conditions qu'en examen : pas de document ni de calculatrice. Tous les TPs se font sous Linux.

1 Travailler avec Python

Cette section décrit comment travailler avec Python dans les salles informatiques du SIF.

Mode commande

Pour lancer l'interpréteur Python et évaluer des instructions dans l'invite de commande, il suffit d'ouvrir un terminal et d'exécuter la commande `python3`. On peut quitter l'interpréteur avec le raccourci clavier `CTRL-d`.

Attention, il existe aussi une commande `python` qui lance la version 2.7 de l'interpréteur. Il ne faut pas utiliser cette commande dans le cadre de ce cours.

Mode programme

On recommande d'utiliser l'éditeur `geany` pour programmer avec Python. Ce dernier permet d'afficher du code Python de façon agréable, de tester la syntaxe du fichier et de lancer automatiquement un terminal pour tester le programme¹.

Configuration de geany

Cette étape n'est à faire qu'une seule fois lors de la première séance de Python. Elle permet de configurer `geany` correctement. Il faut exécuter les deux commandes ci-dessous dans un terminal :

```
mkdir -p ~/.config/geany/filedefs/
```

Cette commande crée un répertoire `filedefs` ainsi que tous les répertoires intermédiaires pour arriver à lui depuis le répertoire utilisateur.

```
cp /public/kn/filetypes.python ~/.config/geany/filedefs/
```

Cette commande copie un fichier de configuration dans le répertoire nouvellement créé.

Association du type de fichier

Cette étape n'est à faire qu'une seule fois lors de la première séance de Python. Ouvrir un terminal et créer un fichier portant l'extension `.py` dans le répertoire personnel :

```
touch ~/test.py
```

Ouvrir l'explorateur de fichiers (icône de dossier dans la barre des tâches). Naviguer jusqu'au fichier `test.py`. Cliquer dessus avec le bouton droit. Choisir **Properties** (ou **Propriétés** si l'interface est en français), puis aller dans l'onglet **Open With** (ou **Ouvrir Avec**). Sélectionner **Geany** dans la liste des choix proposés et fermer la fenêtre. L'éditeur sera alors lancé lorsque vous cliquerez sur un fichier portant l'extension `.py`.

1. Si vous avez déjà un éditeur de texte préféré et que vous savez vous en servir, vous pouvez le faire, mais vous devrez vous débrouiller seul si l'éditeur ne fonctionne pas comme prévu.

Travailler avec geany

Lors de l'édition de fichier Python avec **geany**, ce dernier utilise des couleurs pour différencier les mots clés et ainsi faciliter la lecture du code. La figure 1 indique les principales parties de l'éditeur.

Attention, lorsque **geany** est lancé sans argument et que vous créez un nouveau fichier, il faut commencer par **l'enregistrer avec l'extension .py**, sinon le mode Python ne sera pas activé pour ce fichier.

Le bouton de **Compilation** permet de vérifier la syntaxe du fichier Python sans avoir à l'exécuter. Les erreurs sont alors affichées dans la zone de message. Ce bouton est associé à la touche **F8** du clavier.

Le bouton d'**Exécution** ouvre un terminal et y exécute le code Python se trouvant dans le fichier. Ce bouton est associé à la touche **F5** du clavier. À la fin du programme, le terminal demande l'appui sur la touche **Entrée** afin de fermer la fenêtre.

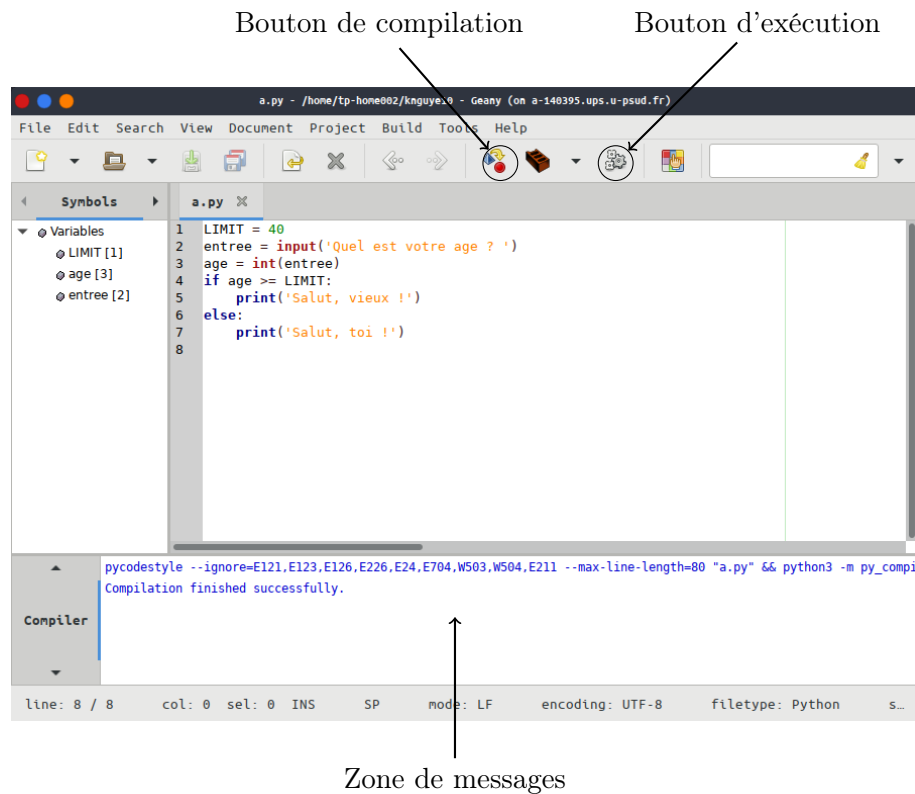


FIGURE 1 – Capture d'écran de l'éditeur **geany**

Exercices

Exercice 1

Ouvrir un terminal. Donner les commandes permettant de :

- créer un répertoire TP3 à l'intérieur du répertoire **IntroInfo**
- se placer à l'intérieur du répertoire TP3

Réponse:

```
$ mkdir IntroInfo/TP3
$ cd IntroInfo/TP3
```

On suppose pour les autres exercices que le répertoire TP3 est le répertoire courant.

Exercice 2

★ Pour chacune des expressions Python suivantes, dire si elle est syntaxiquement correcte. Si ce n'est pas le cas, indiquer où se trouve l'erreur de syntaxe. Si la syntaxe est correcte, indiquer si Python peut calculer une valeur et donner cette dernière, ou s'il se produit une erreur d'exécution. Vérifier les réponses à l'aide de l'interprète Python.

1. `42 + 43`
2. `2 + - 2`
3. `10 + * 3`
4. `'ABCD`
5. `'ABCD' + 'DEFG'`
6. `'ABCD' - 'DEFG'`
7. `100 / 3`
8. `100 // 3`
9. `'A' * 3`
10. `10 // 20 - 2 * 10`
11. `10 // (20 - 2 * 10)`
12. `'10' + str(10)`

Réponse:

- | | |
|---|--|
| 1. <code>42 + 43</code> , correct, calcule 85 | 7. <code>100 / 3</code> , correct, calcule 33.333333333333336 |
| 2. <code>2 + - 2</code> , correct, calcule 0 | 8. <code>100 // 3</code> , correct, calcule 33 |
| 3. <code>10 + * 3</code> , erreur de syntaxe | 9. <code>'A' * 3</code> , correct, calcule 'AAA' |
| 4. <code>'ABCD</code> , erreur de syntaxe (' manquant) | 10. <code>10 // 20 - 2 * 10</code> , correct, calcule -20 (car interprété comme, $(10 // 20) - (2 * 10)$) |
| 5. <code>'ABCD' + 'DEFG'</code> , correct, calcule 'ABCDEFG' | 11. <code>10 // (20 - 2 * 10)</code> , syntaxe correcte, erreur à l'évaluation car division par 0. |
| 6. <code>'ABCD' - 'DEFG'</code> , syntaxe correcte, erreur à l'évaluation, - n'est pas défini pour les chaînes de caractères. | 12. <code>'10' + str(10)</code> , correct, calcule la chaîne '1010'. |

Exercice 3

★ Soit les instructions Python suivantes. On suppose qu'elles sont saisies les unes à la suite des autres dans le même interprète Python, juste après le démarrage de ce dernier. Pour chaque instruction, dire ce que fait l'interprète et ce qu'il affiche.

- | | |
|---------------------------|--|
| 1. <code>x = 42</code> | 7. <code>z</code> |
| 2. <code>x - 8</code> | 8. <code>_VARIABLE = str(z) + '1'</code> |
| 3. <code>y = x - 8</code> | 9. <code>_VARIABLE</code> |
| 4. <code>z = y + z</code> | 10. <code>int(_VARIABLE)</code> |
| 5. <code>z = y + x</code> | 11. <code>_VARIABLE + 4</code> |
| 6. <code>str(z)</code> | 12. <code>int(_VARIABLE) + 4</code> |

Réponse:

1. `x = 42`, pas d’affichage, définit une variable `x` contenant l’entier 42.
2. `x - 8`, affiche 34
3. `y = x - 8`, pas d’affichage, définit une variable `y` contenant l’entier 34
4. `z = y + z`, affiche une erreur, la variable `z` n’a pas été définie.
5. `z = y + x`, pas d’affichage, définit une variable `z` contenant 76 (34 + 42).
6. `str(z)`, affiche '76' (chaîne de caractères).
7. `z`, affiche 76 (entier). Attention, appeler `str(z)` renvoie la conversion de 76 en chaîne, mais ne modifie pas le contenu de `z` qui reste un entier.
8. `_VARIABLE = str(z) + '1'`, pas d’affichage, définit une variable `_VARIABLE` contenant '761'.
9. `_VARIABLE`, affiche '761'.
10. `int(_VARIABLE)`, 761 (encore une fois sans modifier le contenu de `_VARIABLE`).
11. `_VARIABLE + 4`, provoque une erreur (addition entre la chaîne '761' contenue dans la variable `_VARIABLE` et l’entier 4).
12. `int(_VARIABLE) + 4`, affiche 765.

Exercice 4

Créer dans le répertoire `~/IntroInfo/TP3` un fichier `exo4.py`. Y écrire un programme qui demande le nom et le prénom d’une personne et la salue. Exemple (les saisies au clavier sont en italique) :

```
Quel est votre nom ? Lovelace
Quel est votre prénom ? Ada
Bonjour, Ada Lovelace !
```

Indiquer comment exécuter ce programme depuis la ligne de commande sans passer par le bouton « Exécuter » de Geany.

Réponse: Pour cet exercice et les suivants, voir les fichiers Python joints au corrigé.

Pour exécuter le programme, on peut lancer la commande :

```
python3 exo4.py
```

En se trouvant dans le répertoire TP3.

Exercice 5

Créer dans le répertoire `~/IntroInfo/TP3` un fichier `exo5.py`. Y écrire un programme qui demande une année et affiche si cette dernière est bissextile ou non. Exemple :

```
Saisir une année ? 1984
Bissextile
```

Autre exemple :

```
Saisir une année ? 2011
Non bissextile
```

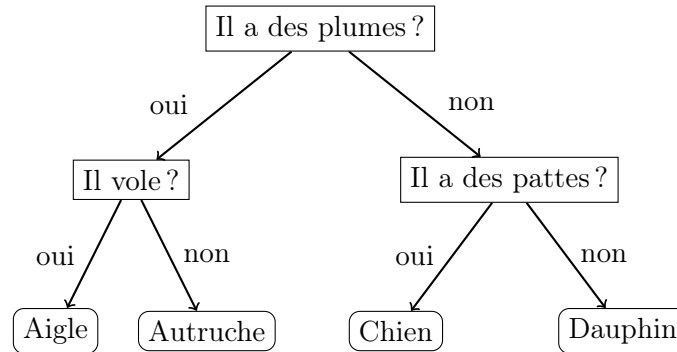
On rappelle qu’une année est bissextile si c’est un multiple de 4 qui doit aussi

- ne pas être un multiple de 100
- ou être un multiple de 400

Réponse: Attention, il convient d’observer que le `and` est plus prioritaire que le `or` de la même façon que le `*` est plus prioritaire que le `+`.

Exercice 6

Créer dans le répertoire `~/IntroInfo/TP3` un fichier `exo6.py`. Y écrire un programme qui demande à l'utilisateur de choisir dans sa tête parmi quatre animaux : Chien, Dauphin, Aigle, Autruche. Le programme pose ensuite des questions à l'utilisateur en suivant le diagramme de décision ci-dessous :



Précisions :

- On peut utiliser `input()` sans affecter son résultat à une variable simplement pour afficher un message et attendre que l'utilisateur appuie sur la touche **Entrée**.
- On peut comparer deux chaînes de caractères avec l'opérateur `==`.
- Si l'utilisateur ne répond pas **oui** on peut considérer qu'il a répondu **non**, même si la chaîne saisie est différente.

Exercice 7

Étant donnés deux points du plan (x_1, y_1) et (x_2, y_2) où les x_i et y_i sont des entiers, on souhaite calculer l'équation de la droite passant par ces deux points. Créer dans le répertoire `~/IntroInfo/TP3` un fichier `exo7.py`. Votre programme doit :

- demander les coordonnées des points à l'utilisateur
- si $x_1 = x_2$. Dans ce cas, la droite est verticale et le programme affiche $x=x_1$ (avec la bonne valeur de x_1).
- sinon, résoudre le système d'équations :

$$\begin{cases} y_1 &= ax_1 + b \\ y_2 &= ax_2 + b \end{cases}$$

d'inconnues a et b et afficher l'équation sous la forme : $y=a*x+b$ (ou $y=a*x-b$ si b est négatif) pour les bonnes valeurs de a et b . On demande à ce que ces valeurs soient des nombres à virgule si nécessaire. De plus, le programme devra être commenté. Exemple :

```
x1 ? 1
y1 ? 2
x2 ? 3
y2 ? 7

y=2.5*x-0.5
```

Exercice 8

Créer dans le répertoire `~/IntroInfo/TP3` un fichier `exo8.py`. Commencer ce fichier par la ligne :

```
from turtle import *
```

La signification exacte de ces instructions Python sera donnée plus tard dans le cours. On indique simplement qu'elles donnent accès à une fenêtre graphique et un curseur permettant de dessiner dans la fenêtre.

Les instructions de `turtle` comprennent en premier lieu des moyens d'orienter et déplacer le curseur (la « tortue ») dans le plan cartésien à deux dimensions (en utilisant le repère standard des mathématiques, avec abscisses sur un axe horizontal croissant vers la droite et ordonnées avec axe vertical croissant vers le haut).

instruction	description
<code>goto(<i>x</i>, <i>y</i>)</code>	aller au point de coordonnées (<i>x</i> , <i>y</i>)
<code>forward(<i>d</i>)</code>	avancer de la distance <i>d</i>
<code>backward(<i>d</i>)</code>	reculer de la distance <i>d</i>
<code>left(<i>a</i>)</code>	pivoter d'un angle <i>a</i> vers la gauche
<code>right(<i>a</i>)</code>	pivoter d'un angle <i>a</i> vers la droite
<code>circle(<i>r</i>, <i>a</i>)</code>	tracer un arc de cercle d'angle <i>a</i> et de rayon <i>r</i>
<code>dot(<i>r</i>)</code>	tracer un point de rayon <i>r</i>

Les coordonnées et distances sont mesurées en pixels et les angles en degrés. Les arcs de cercles sont parcourus dans le sens trigonométrique si le rayon est positif, et sens horaire si le rayon est négatif. S'ajoutent à cette base une série d'instructions permettant de modifier les dessins produits par chacun des déplacements.

instruction	description
<code>up()</code>	relever le crayon (et interrompre le dessin)
<code>down()</code>	redescendre le crayon (et reprendre le dessin)
<code>width(<i>e</i>)</code>	fixer à <i>e</i> l'épaisseur du trait
<code>color(<i>c</i>)</code>	sélectionner la couleur <i>c</i> pour les traits
<code>begin_fill()</code>	activer le mode remplissage
<code>end_fill()</code>	désactiver le mode remplissage
<code>fillcolor(<i>c</i>)</code>	sélectionner la couleur <i>c</i> pour le remplissage

Enfin, l'instruction `done()` permet de « terminer » un dessin. La fenêtre reste alors affichée jusqu'à ce qu'elle soit fermée par l'utilisateur. Cette instruction doit être la dernière.

1. Que dessine le programme ci-dessous (essayer de le déduire sans l'exécuter) ?

```
from turtle import *

color("black")
fillcolor("red")
x = 0
y = 200

up()
goto(x, y)
begin_fill()
down()

x = x + 50
y = y - 200
goto(x, y)

x = x - 50
y = y - 200
goto(x, y)

x = x - 50
y = y + 200
goto(x, y)

x = x + 50
y = y + 200
goto(x, y)

end_fill()
up()

done()
```

2. Compléter le fichier `exo8_2.py` ci-dessous, de façon à ce qu'il dessine un drapeau français, centré en $(0,0)$ et où chacune des trois bandes fait `hauteur` pixel de haut et `largeur` pixels de large. Le programme doit utiliser ces deux variables et permettre ainsi de changer la taille du drapeau en ne modifiant que les variables et pas la suite du code.

```
# fichier exo8_2.py

from turtle import *

hauteur = 400
largeur = 100

# Ci dessous le code à compléter.
```