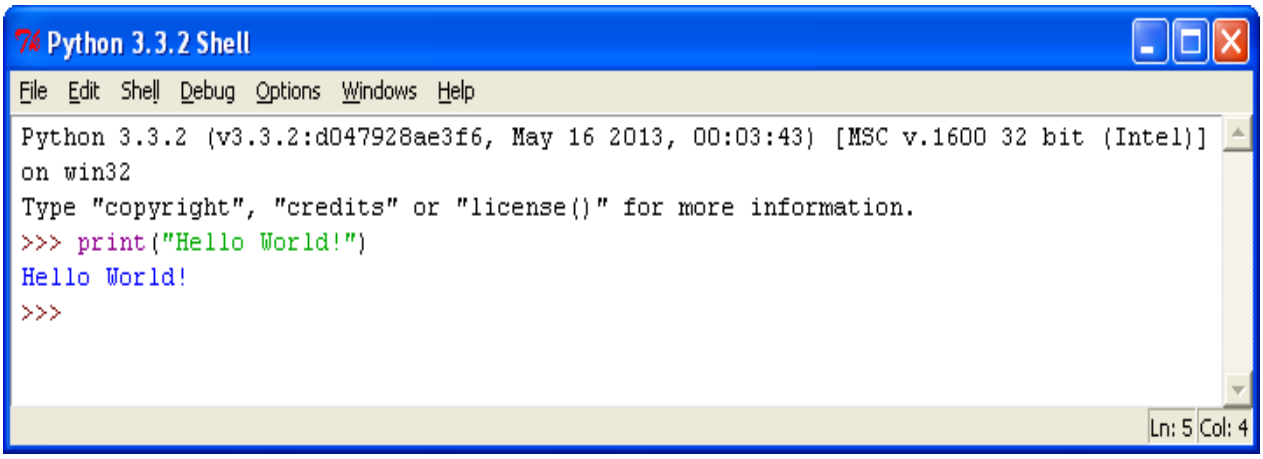
Chapitre 1 - Variables, types et opérateurs

Une variable est un espace mémoire dans lequel il est possible de stocker une valeur (une donnée).

Ouvrir IDLE :
Démarrer → Programmes → Python → IDLE (Python GUI)



0- Noms de variables

Le nom d'une variable s'écrit avec des lettres (non accentuées), des chiffres ou bien l'underscore _
Le nom d'une variable ne doit pas commencer par un chiffre.

En langage Python, l'usage est de ne pas utiliser de lettres majuscules pour nommer les variables (celles-ci sont réservées pour nommer les classes).

Exemple : `age, mon_age, temperature1`

> A éviter : ~~`Age, AGE, MonAge, monAge, Temperature1`~~

1- Le type `int` (integer : nombres entiers)

Pour affecter (on dit aussi assigner) la valeur 17 à la variable nommée `age` :

```
>>> age = 17
```

La fonction `print()` affiche la valeur de la variable :

```
>>> print(age)
17
```

La fonction `type()` retourne le type de la variable :

```
>>> print(type(age))
```

```
<class 'int'>
```

`int` est le type des nombres entiers.

```
>>> # ceci est un commentaire
>>> age = age + 1    # en plus court : age += 1
>>> print(age)
18
>>> age = age - 3    # en plus court : age -= 3
>>> print(age)
15
>>> age = age*2      # en plus court : age *= 2
>>> print(age)
30
>>> a = 6*3 - 20
>>> print(a)
-2
>>> b = 25
>>> c = a + 2*b
>>> print(b, c)          # ne pas oublier la virgule
25 48
```

L'opérateur // donne la division entière :

```
>>> tour = 450//360
>>> print(tour)
1
```

L'opérateur % donne le reste de la division (opération modulo) :

```
>>> angle = 450%360
>>> print(angle)
90
```

L'opérateur ** donne la puissance :

```
>>> mo = 2**20
```

```
>>> print(mo)
1048576
>>> racine2 = 2**0.5
>>> print(racine2)
1.41421356237
```

2- Le type `float` (nombres en virgule flottante)

```
>>> b = 17.0    # le séparateur décimal est un point (et non une virgule)
>>> print(b)
17.0
>>> print(type(b))
<class 'float'>
>>> c = 14.0/3.0
>>> print(c)
4.66666666667
>>> c = 14.0//3.0    # division entière
>>> print(c)
4.0
```

Attention : avec des nombres entiers, l'opérateur / fait une division classique et retourne un type `float` :

```
>>> c = 14/3
>>> print(c)
4.66666666667
```

Notation scientifique :

```
>>> a = -1.784892e4
>>> print(a)
-17848.92
```

Les fonctions mathématiques

Pour utiliser les fonctions mathématiques, il faut commencer par importer le module `math` :

```
>>> import math
```

La fonction `dir()` retourne la liste des fonctions et données d'un module :

```
>>> dir(math)
['__doc__', '__name__', '__package__', 'acos',
'acosh', 'asin', 'asinh', 'atan',
'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'erf',
'erfc', 'exp', 'expm1', 'fabs', 'factorial',
'floor', 'fmod', 'frexp', 'fsum',
'gamma', 'hypot', 'isinf', 'isnan', 'ldexp',
'lgamma', 'log', 'log10', 'log1p',
'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh', 'trunc']
```

Pour appeler une fonction d'un module, la syntaxe est la suivante :
module.fonction(arguments)

Pour accéder à une donnée d'un module :
module.data

```
>>> print(math.pi)         # donnée pi du module
math (nombre pi)
3.14159265359
>>> print(math.sin(math.pi/4.0)) # fonction sin()
du module math (sinus)
0.707106781187
>>> print(math.sqrt(2.0)) # fonction sqrt() du
module math (racine carrée)
1.41421356237
>>> print(math.sqrt(-5.0))
Traceback (most recent call last):
    print math.sqrt(-5.0)
ValueError: math domain error
>>> print(math.exp(-3.0)) # fonction exp() du
module math (exponentielle)
0.0497870683679
```

```
>>> print(math.log(math.e))    # fonction log() du module math (logarithme népérien)
1.0
```

3- Le type `str` (string : chaîne de caractères)

```
>>> nom = 'Dupont'   # entre apostrophes
>>> print(nom)
Dupont
>>> print(type(nom))
<class 'str'>
>>> prenom = "Pierre"     # on peut aussi utiliser les guillemets
>>> print(prenom)
Pierre
>>> print(nom, prenom)    # ne pas oublier la virgule
Dupont Pierre
```

La concaténation désigne la mise bout à bout de plusieurs chaînes de caractères.
La concaténation utilise l'opérateur +

```
>>> chaine = nom + prenom # concaténation de deux chaînes de caractères
>>> print(chaine)
DupontPierre
>>> chaine = prenom + nom # concaténation de deux chaînes de caractères
>>> print(chaine)
PierreDupont
>>> chaine = prenom + ' ' + nom
>>> print(chaine)
Pierre Dupont
>>> chaine = chaine + ' 18 ans' # en plus court : chaine += ' 18 ans'
```

```
>>> print(chaine)
Pierre Dupont 18 ans
```

La fonction `len()` retourne la longueur (length) de la chaîne de caractères :

```
>>> print(len(chaine))
20
```

Indexage et slicing :

```
>>> print(chaine[0])       # premier caractère (indice 0)
P
>>> print(chaine[1]) # deuxième caractère (indice 1)
i
>>> print(chaine[1:4])     # slicing
ier
>>> print(chaine[2:])      # slicing
erre Dupont 18 ans
>>> print(chaine[-1])      # dernier caractère (indice -1)
s
>>> print(chaine[-6:])     # slicing
18 ans
```

En résumé :

```
 +---+---+---+---+---+---+
 | M | u | r | i | e | l |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

```
>>> chaine = 'Aujourd'hui'
SyntaxError: invalid syntax
```

```
>>> chaine  = 'Aujourd\'hui'          # séquence d'échappement \'
>>> print(chaine)
Aujourd'hui
>>> chaine  = "Aujourd'hui"
>>> print(chaine)
Aujourd'hui
```

La séquence d'échappement `\n` représente un saut ligne :

```
>>> chaine = 'Première ligne\nDeuxième ligne'
>>> print(chaine)
Première ligne
Deuxième ligne
```

Plus simplement, on peut utiliser les triples guillemets (ou les triples apostrophes) pour encadrer une chaîne définie sur plusieurs lignes :

```
>>> chaine = """Première ligne
Deuxième ligne"""
>>> print(chaine)
Première ligne
Deuxième ligne
```

On ne peut pas mélanger les serviettes et les torchons (ici type `str` et type `int`) :

```
>>> chaine = '17.45'
>>> print(type(chaine))
<class 'str'>
>>> chaine = chaine + 2
TypeError: Can't convert 'int' object to str implicitly
```

La fonction `float()` permet de convertir un type `str` en type `float`

```
>>> nombre = float(chaine)
>>> print(nombre)
17.45
```

```
>>> print(type(nombre))
<class 'float'>
>>> nombre = nombre + 2          # en plus court : nombre += 2
>>> print(nombre)
19.45
```

La fonction `input()` lance une invite de commande (en anglais : prompt) pour saisir une chaîne de caractères.

```
>>> # saisir une chaîne de caractères et valider avec la touche Enter
>>> chaine = input("Entrer un nombre : ")
Entrer un nombre : 14.56
>>> print(chaine)
14.56
>>> print(type(chaine))
<class 'str'>
>>> nombre = float(chaine)# conversion de type
>>> print(nombre**2)
211.9936
```

4- Le type `list` (liste)

Une liste est une structure de données.
Le premier élément d'une liste possède l'indice (l'index) 0.
Dans une liste, on peut avoir des éléments de plusieurs types.

```
>>> infoperso = ['Pierre', 'Dupont', 17, 1.75, 72.5]
>>> # la liste infoperso contient 5 éléments de types str, str, int, float et float
>>> print(type(infoperso))
<class 'list'>
>>> print(infoperso)
['Pierre', 'Dupont', 17, 1.75, 72.5]
```

```
>>> print('Prénom : ', infoperso[0])            # premier élément (indice 0)
Prénom :  Pierre
>>> print('Age : ', infoperso[2])          # le troisième élément a l'indice 2
Age :  17
>>> print('Taille : ', infoperso[3])            # le quatrième élément a l'indice 3
Taille :  1.75
```

La fonction `range()` crée une liste d'entiers régulièrement espacés :

```
>>> maliste = range(10)
>>> print(list(maliste))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print(type(maliste))
<class 'range'>
>>> maliste = range(1,10,2)     # range(début,fin non comprise,intervalle)
>>> print(list(maliste))
[1, 3, 5, 7, 9]
>>> print(maliste[2])           # le troisième élément a l'indice 2
5
```

On peut créer une liste de listes, qui s'apparente à un tableau à 2 dimensions (ligne, colonne) :

```
 0  1  2
10 11 12
20 21 22
```

```
>>> maliste = [[0, 1, 2], [10, 11, 12], [20, 21, 22]]
>>> print(maliste[0])
[0, 1, 2]
>>> print(maliste[0][0])
0
```

```
>>> print(maliste[2][1])        # élément à la troisième ligne et deuxième colonne
21
>>> maliste[2][1] = 69          # nouvelle affectation
>>> print(maliste)
[[0, 1, 2], [10, 11, 12], [20, 69, 22]]
```

5- Le type bool (booléen)

Deux valeurs sont possibles : `True` et `False`

```
>>> choix = True
>>> print(type(choix))
<class 'bool'>
```

Les opérateurs de comparaison :

Opérateur	Signification	Remarques
<	strictement inférieur	
<=	inférieur ou égal	
>	strictement supérieur	
>=	supérieur ou égal	
==	égal	Attention : deux signes ==
!=	différent	

```
>>> b = 10
>>> print(b > 8)
True
>>> print(b == 5)
False
>>> print(b != 10)
False
>>> print(0 <= b <= 20)
True
```

Les opérateurs logiques : `and, or, not`

```
>>> note = 13.0
>>> mention_ab = note >= 12.0 and note < 14.0 # ou bien : mention_ab = 12.0 <= note < 14.0
>>> print(mention_ab)
True
>>> print(not mention_ab)
False
>>> print(note == 20.0 or note == 0.0)
False
```

L'opérateur `in` s'utilise avec des chaînes (type `str`) ou des listes (type `list`) :

```
>>> chaine = 'Bonsoir'
>>> # la sous-chaîne 'soir' fait-elle partie de la chaîne 'Bonsoir' ?
>>> resultat = 'soir' in chaine
>>> print(resultat)
True
>>> print('b' in chaine)
False
>>> maliste = [4, 8, 15]
>>> # le nombre entier 9 est-il dans la liste ?
>>> print(9 in maliste)
False
>>> print(8 in maliste)
True
>>> print(14 not in maliste)
True
```

6- Le type `dict` (dictionnaire)

Un dictionnaire stocke des données sous la forme **clé ⇒ valeur**
Une clé est unique et n'est pas nécessairement un entier (comme c'est le cas de l'indice d'une liste).

```
>>> moyennes = {'math': 12.5, 'anglais': 15.8} # entre accolades
>>> print(type(moyennes))
<class 'dict'>
>>> print(moyennes['anglais'])       # entre crochets
15.8
>>> moyennes['anglais'] = 14.3       # nouvelle affectation
>>> print(moyennes)
{'anglais': 14.3, 'math': 12.5}
>>> moyennes['sport'] = 11.0         # nouvelle entrée
>>> print(moyennes)
{'sport': 11.0, 'anglais': 14.3, 'math': 12.5}
```

7- Autres types

Nous avons vu les types les plus courants.
Il en existe bien d'autres :

- complex (nombres complexes, par exemple 1+2.5j)
- tuple (structure de données)
- set (structure de données)
- file (fichiers)
- ...

8- Programmation Orientée Objet (POO)

Python est un langage de programmation **orienté objet** (comme les langages C++, Java, PHP, Ruby...).
Une variable est en fait un **objet** d'une certaine **classe**.

Par exemple, la variable `amis` est un objet de la classe `list`.
On dit aussi que la variable `amis` est une **instance** de la classe `list`.
L'**instanciation** (action d'instancier) est la création d'un objet à partir d'une classe (syntaxe : **NouvelObjet = NomdelaClasse(arguments)**) :

```
>>> # instanciation de l'objet amis de la classe list
```

```
>>> amis = ['Nicolas', 'Julie'] # ou bien : amis = list(['Nicolas', 'Julie'])
>>> print(type(amis))
<class 'list'>
```

Une classe possède des fonctions que l'on appelle **méthodes** et des données que l'on appelle **attributs**.

La méthode `append()` de la classe `list` ajoute un nouvel élément en fin de liste :

```
>>> # instanciation d'une liste vide
>>> amis = []                  # ou bien : amis = list()
>>> amis.append('Nicolas')# synthase générale : objet.méthode(arguments)
>>> print(amis)
['Nicolas']
>>> amis.append('Julie')  # ou bien : amis = amis + ['Julie']
>>> print(amis)
['Nicolas', 'Julie']
>>> amis.append('Pauline')
>>> print(amis)
['Nicolas', 'Julie', 'Pauline']
>>> amis.sort()               # la méthode sort() trie les éléments
>>> print(amis)
['Julie', 'Nicolas', 'Pauline']
>>> amis.reverse()         # la méthode reverse() inverse la liste des éléments
>>> print(amis)
['Pauline', 'Nicolas', 'Julie']
```

La méthode `lower()` de la classe `str` retourne la chaîne de caractères en casse minuscule :

```
>>> # la variable chaine est une instance de la classe str
```

```
>>> chaine = "BONJOUR"          # ou bien : chaine = str("BONJOUR")
>>> chaine2 = chaine.lower()    # on applique la méthode lower() à l'objet chaine
>>> print(chaine2)
bonjour
>>> print(chaine)
BONJOUR
```

La méthode `pop()` de la classe `dict` supprime une clé :

```
>>> # instanciation de l'objet moyennes de la classe dict
>>> moyennes = {'sport': 11.0, 'anglais': 14.3, 'math': 12.5}
>>> # ou : moyennes = dict({'sport': 11.0, 'anglais': 14.3, 'math': 12.5})
>>> moyennes.pop('anglais')
14.3
>>> print(moyennes)
{'sport': 11.0, 'math': 12.5}
>>> print(moyennes.keys())      # la méthode keys() retourne la liste des clés
dict_keys(['sport', 'math'])
>>> print(moyennes.values())    # la méthode values() retourne la liste des valeurs
dict_values([11.0, 12.5])
```

Exercices

Exercice 1.1 ☆ Afficher la taille en octets et en bits d'un fichier de 536 ko.
On donne : 1 ko (1 kilooctet) = 2^{10} octets !!!
1 octet = 1 byte = 8 bits

Exercice 1.2 ★ Le numéro de sécurité sociale est constitué de 13 chiffres auquel s'ajoute la clé de contrôle (2 chiffres).
Exemple : 1 89 11 26 108 268 91
La clé de contrôle est calculée par la formule : 97 - (numéro de sécurité sociale modulo 97)

Retrouver la clé de contrôle de votre numéro de sécurité sociale.
Quel est l'intérêt de la clé de contrôle ?

Exercice 1.3 ★ Afficher la valeur numérique de $\sqrt{(4,6^3 - 15/16)}$
Comparer avec votre calculette.

Exercice 1.4 ★ A partir des deux variables `prenom` et `nom`, afficher les initiales (par exemple LM pour Léa Martin).

Exercice 1.5 ★☆ L'identifiant d'accès au réseau du lycée est construit de la manière suivante : initiale du prénom puis les 8 premiers caractères du nom (le tout en minuscule).
Exemple : Alexandre Lecouturier → alecoutur
A partir des deux variables `prenom` et `nom`, construire l'identifiant.

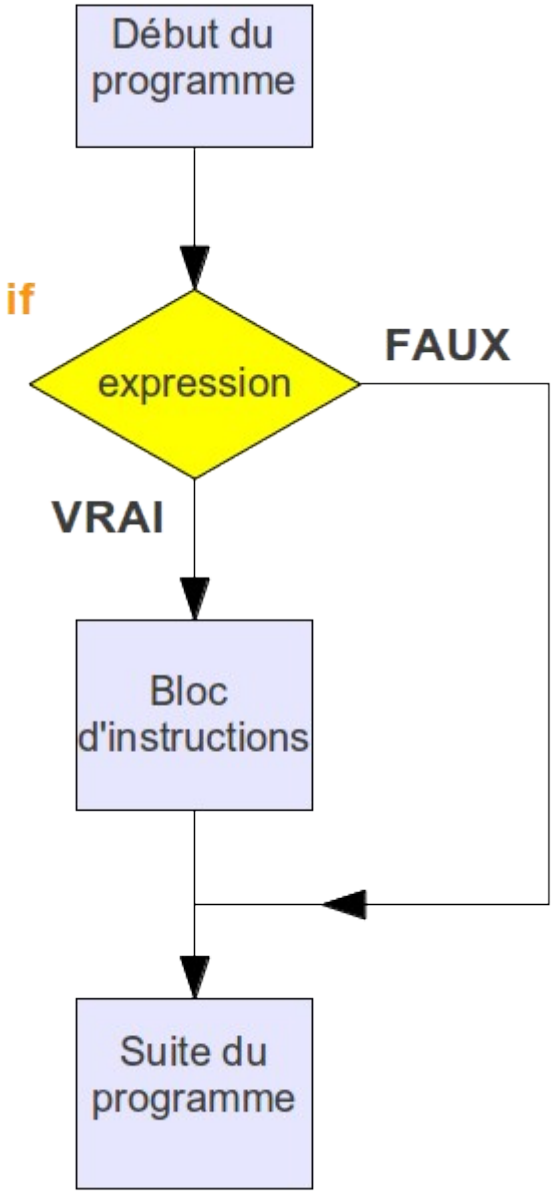
Exercice 1.6 ★★ On donne un extrait des logins d'accès au réseau du lycée :

```
alecoutur   Huz4
lmartin     monty
fsincere    gnugpl
```

1) Créer une variable de type `dict` qui contient les couples identifiant - mot de passe ci-dessus.
2) La saisie du login fournit deux variables `identifiant` et `motdepasse` : une pour l'identifiant et l'autre pour le mot de passe.
Construire une variable booléenne qui donne `True` en cas d'identification correcte, et `False` dans le cas contraire :
lmartin monty → True
alecoutur fqsdf → False
martin monty → False (ce cas est plus compliqué à traiter)

Chapitre 2 - Les conditions

L'instruction `if`



Syntaxe

```
if expression:                  # ne pas oublier le signe de ponctuation ':'
    bloc d'instructions         # attention à l'indentation
# suite du programme
```

Si l'expression est vraie (True) alors le bloc d'instructions est exécuté.
Si l'expression est fausse (False) on passe directement à la suite du programme.

Premier script

Nous allons commencer par créer le script `Condition1.py` :

Ouvrir IDLE :
Démarrer → Programmes → Python → IDLE (Python GUI)

File → New Window
Copier puis coller le code source ci-dessous :

```
# script Condition1.py


chaine = input("Note sur 20 : ")
note = float(chaine)
if note >= 10.0:
    # ce bloc est exécuté si l'expression (note >= 10.0) est vraie
    print("J'ai la moyenne")
print("Fin du programme")
```

File → Save As
Répertoire : C:\PythonXX
Nom du fichier : Condition1.py

Pour exécuter le script :
Run → Run Module (ou touche F5) :

```
>>>
Note sur 20 : 16
J'ai la moyenne
Fin du programme
>>>
Note sur 20 : 5
Fin du programme
```

L'instruction `else`

Une instruction `else` est toujours associée à une instruction `if`

Syntaxe

```
if expression:
    bloc d'instructions 1       # attention à l'indentation
```

```
else:                          # else est au même niveau que if
    bloc d'instructions 2      # attention à l'indentation
# suite du programme
```

Si l'expression est vraie (True) alors le bloc d'instructions 1 est exécuté.
Si l'expression est fausse (False) alors c'est le bloc d'instructions 2 qui est exécuté.

```
# script Condition2.py


chaine = input("Note sur 20 : ")
note = float(chaine)
if note >= 10.0:
    # ce bloc est exécuté si l'expression (note >= 10.0) est vraie
    print("J'ai la moyenne")
else:
    # ce bloc est exécuté si l'expression (note >= 10.0) est fausse
    print("C'est en dessous de la moyenne")
print("Fin du programme")
>>>
Note sur 20 : 15
J'ai la moyenne
Fin du programme
>>>
Note sur 20 : 8.5
C'est en dessous de la moyenne
Fin du programme
>>>
Note sur 20 : 56
J'ai la moyenne
```

```
Fin du programme
```

Pour traiter le cas des notes invalides (<0 ou >20), on peut imbriquer des instructions conditionnelles :

```
# script Condition3.py


chaine = input("Note sur 20 : ")
note = float(chaine)
if note > 20.0 or note < 0.0:
    # ce bloc est exécuté si l'expression (note > 20.0 or note < 0.0) est vraie
    print("Note invalide !")
else:
    # ce bloc est exécuté si l'expression (note > 20.0 or note < 0.0) est fausse
    if note >= 10.0:
        # ce bloc est exécuté si l'expression (note >= 10.0) est vraie
        print("J'ai la moyenne")
    else:
        # ce bloc est exécuté si l'expression (note >= 10.0) est fausse
        print("C'est en dessous de la moyenne")
print("Fin du programme")
>>>
Note sur 20 : 56
Note invalide !
Fin du programme
>>>
Note sur 20 : 14.6
J'ai la moyenne
Fin du programme
```

On ajoute encore un niveau d'imbrication pour traiter les cas particuliers 0 et 20 :

```
# script Condition4.py

chaine = input("Note sur 20 : ")
note = float(chaine)
if note > 20.0 or note < 0.0:
    print("Note invalide !")
else:
    if note >= 10.0:
        print("J'ai la moyenne")
        if note == 20.0:
            # ce bloc est exécuté si l'expression (note == 20.0) est vraie
            print("C'est même excellent !")
    else:
        print("C'est en dessous de la moyenne")
        if note == 0.0:
            # ce bloc est exécuté si l'expression (note == 0.0) est vraie
            print("... lamentable !")
print("Fin du programme")
>>>
Note sur 20 : 20
J'ai la moyenne
C'est même excellent !
Fin du programme
>>>
Note sur 20 : 3
C'est en dessous de la moyenne
Fin du programme
```

L'instruction `elif`

Une instruction `elif` (contraction de else if) est toujours associée à une instruction `if`

Syntaxe

```
if expression 1:
    bloc d'instructions 1
elif expression 2:
    bloc d'instructions 2
elif expression 3:
    bloc d'instructions 3 # ici deux instructions elif, mais il n'y a pas de limitation
else:
    bloc d'instructions 4
# suite du programme
```

Si l'expression 1 est vraie alors le bloc d'instructions 1 est exécuté, et on passe à la suite du programme.
Si l'expression 1 est fausse alors on teste l'expression 2 :

- si l'expression 2 est vraie on exécute le bloc d'instructions 2, et on passe à la suite du programme.
- si l'expression 2 est fausse alors on teste l'expression 3, etc.

Le bloc d'instructions 4 est donc exécuté si toutes les expressions sont fausses (c'est le bloc "par défaut").

Parfois il n'y a rien à faire.
Dans ce cas, on peut omettre l'instruction `else` :

```
if expression 1:
    bloc d'instructions 1
elif expression 2:
    bloc d'instructions 2
elif expression 3:
    bloc d'instructions 3
# suite du programme
```

L'instruction `elif` évite souvent l'utilisation de conditions imbriquées (et souvent compliquées).

Exemple

```
# script Condition5.py
# ce script fait la même chose que Condition4.py

note = float(input("Note sur 20 : "))
if note == 0.0:
    print("C'est en dessous de la moyenne")
    print("... lamentable !")
elif note == 20.0:
    print("J'ai la moyenne")
    print("C'est même excellent !")
elif note < 10.0 and note > 0.0:     # ou bien :
elif 0.0 < note < 10.0:
    print("C'est en dessous de la moyenne")
elif note >= 10.0 and note < 20.0:   # ou bien :
elif 10.0 <= note < 20.0:
    print("J'ai la moyenne")
else:
    print("Note invalide !")
print("Fin du programme")
>>>
Note sur 20 : 20
J'ai la moyenne
C'est même excellent !
Fin du programme
>>>
Note sur 20 : 3
C'est en dessous de la moyenne
Fin du programme
>>>
```

```
Note sur 20 : 77

Note invalide !

Fin du programme
```

Exercices

Exercice 2.1 ★ Le numéro de sécurité sociale est constitué de 13 chiffres auquel s'ajoute la clé de contrôle (2 chiffres).
La clé de contrôle est calculée par la formule : 97 - (numéro de sécurité sociale modulo 97)

Ecrire un script qui contrôle la validité d'un numéro de sécurité sociale.
On pourra utiliser la fonction `int()` pour convertir le type `str` en type `int`.
Exemple :

```
>>>

Entrer votre numéro de sécurité sociale (13
chiffres) --> 1891126108268

Entrer votre clé de contrôle (2 chiffres)
--------------> 91

Votre numéro de sécurité sociale est valide.

>>>

Entrer votre numéro de sécurité sociale (13
chiffres) --> 2891126108268

Entrer votre clé de contrôle (2 chiffres)
--------------> 91

Votre numéro de sécurité sociale est INVALIDE !

>>>
```

Exercice 2.2 ★ Nombre entier non signé et signé
Dans un octet, on peut stocker un nombre entier compris entre 0b00000000 = 0 et 0b11111111 = 255 (entier non signé, en numération binaire naturel).
On peut aussi stocker un entier compris entre -128 et +127 (entier signé, représentation dite en complément à deux).
En complément à deux, les nombres négatifs sont codés de la manière suivante :
-1 correspond à 255 en binaire naturel
-2 correspond à 254 en binaire naturel
...
-127 correspond à 129 en binaire naturel
-128 correspond à 128 en binaire naturel

1) Ecrire un script qui donne la correspondance entre entier signé et entier non signé.
Par exemple :

```
>>>
Entrer un entier signé en complément à deux (-128 à +127): 25
La représentation en binaire naturel est : 25
>>>
Entrer un entier signé en complément à deux (-128 à +127): -15
La représentation en binaire naturel est : 241
```

2) Ecrire un script qui donne la correspondance entre entier non signé et entier signé.
Par exemple :

```
>>>
Entrer un nombre entier (0 à 255): 250
Cela représente l'entier signé : -6
```

Exercice 2.3 ★ Ecrire un script qui demande la note au bac et qui affiche la mention correspondante.
Par exemple :

```
>>>
Note au bac (sur 20) : 13.5
Bac avec mention Assez Bien
>>>
Note au bac (sur 20) : 10.9
Bac avec mention Passable
>>>
Note au bac (sur 20) : 4
Recalé
>>>
```

Exercice 2.4 ★ Ecrire un script qui calcule l'indice de masse corporelle (IMC) d'un adulte et qui en donne l'interprétation (corpulence normale, surpoids...).
Par exemple :

```
>>>
Votre taille en cm ? 170
Votre masse en kg ? 68.5
IMC = 23.70 kg/m²
Interprétation : corpulence normale
>>>
```

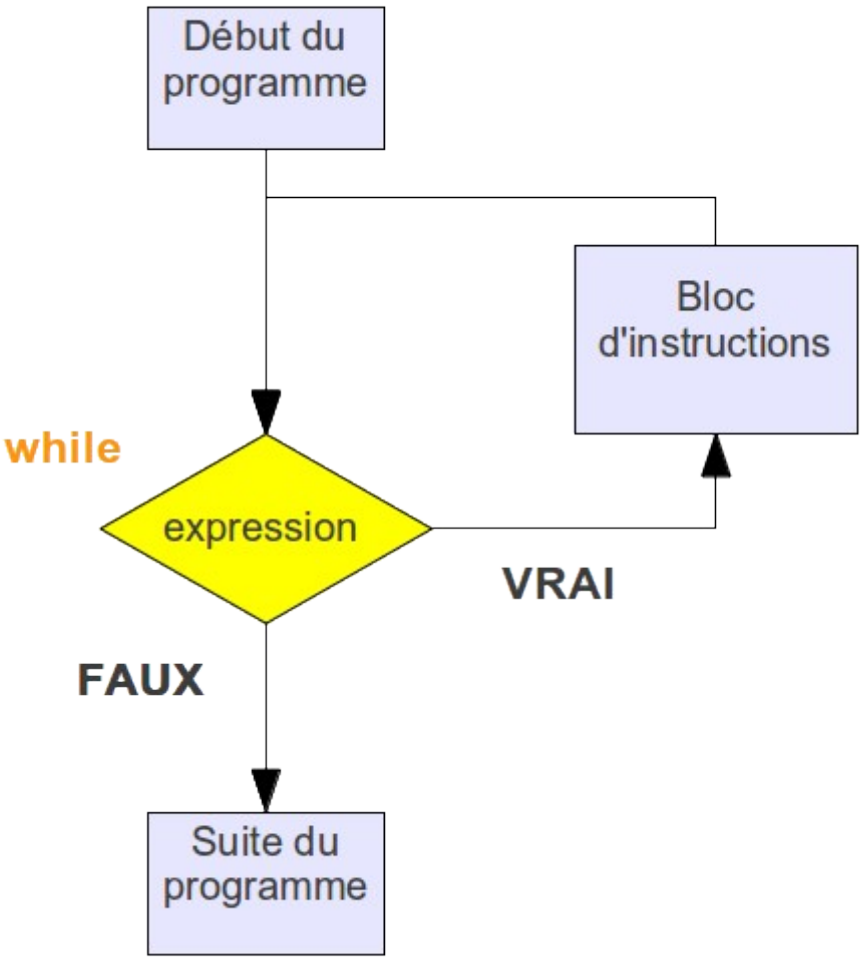
Exercice 2.5 ★★ Ecrire un script qui résout l'équation du second degré : $ax^2 + bx + c = 0$
Par exemple :

```
>>>
Résolution de l'équation du second degré : ax² + bx + c = 0
Coefficient a ? 1
Coefficient b ? -0.9
Coefficient c ? 0.056
Discriminant :  0.586
Deux solutions :
0.0672468158199
0.83275318418
>>>
Résolution de l'équation du second degré : ax² + bx + c = 0
Coefficient a ? 2
Coefficient b ? 1.5
Coefficient c ? 4
Discriminant :  -29.75
Il n'y a pas de solution.
```

Chapitre 3 - Les boucles

Une boucle permet d'exécuter une portion de code plusieurs fois de suite.

L'instruction `while`



Syntaxe

```
while expression:          # ne pas oublier le signe de ponctuation ':'
    bloc d'instructions        # attention à l'indentation
# suite du programme
```

Si l'expression est vraie (`True`) le bloc d'instructions est exécuté, puis l'expression est à nouveau évaluée.
Le cycle continue jusqu'à ce que l'expression soit fausse (`False`) : on passe alors à la suite du programme.

Exemple : un script qui compte de 1 à 4

```
# script Boucle1.py


# initialisation de la variable de comptage
```

```
compteur = 1
while compteur < 5:
    # ce bloc est exécuté tant que la condition (compteur < 5) est vraie
    print(compteur, compteur < 5)
    compteur += 1    # incrémentation du compteur, compteur = compteur + 1
print(compteur < 5)
print("Fin de la boucle")
>>>
1 True
2 True
3 True
4 True
False
Fin de la boucle
```

Table de multiplication par 8

```
# script Boucle2.py


compteur = 1    # initialisation de la variable de comptage
while compteur <= 10:
    # ce bloc est exécuté tant que la condition (compteur <= 10) est vraie
    print(compteur, '* 8 =', compteur*8)
    compteur += 1    # incrémentation du compteur, compteur = compteur + 1
print("Et voilà !")
>>>
1 * 8 = 8
2 * 8 = 16
```

```
3 * 8 = 24
4 * 8 = 32
5 * 8 = 40
6 * 8 = 48
7 * 8 = 56
8 * 8 = 64
9 * 8 = 72
10 * 8 = 80
Et voilà !
```

Affichage de l'heure courante

```
# script Boucle3.py

import time     # importation du module time
quitter = 'n'   # initialisation
while quitter != 'o':
    # ce bloc est exécuté tant que la condition est vraie
    # strftime() est une fonction du module time
    print('Heure courante ', time.strftime('%H:%M:%S'))
    quitter = input("Voulez-vous quitter le programme (o/n) ? ")
print("A bientôt")
>>>
Heure courante  09:48:54
Voulez-vous quitter le programme (o/n) ? n
Heure courante  09:48:58
Voulez-vous quitter le programme (o/n) ? o
A bientôt
```

L'instruction for

Syntaxe

```
for élément in séquence :
    bloc d'instructions
# suite du programme
```

Les éléments de la séquence sont issus d'une chaîne de caractères ou bien d'une liste.

Exemple avec une séquence de caractères

```
# script Boucle4.py


chaine = 'Bonsoir'
for lettre in chaine:           # lettre est la variable d'itération
    print(lettre)
print("Fin de la boucle")
```

La variable `lettre` est initialisée avec le premier élément de la séquence (`'B'`).
Le bloc d'instructions est alors exécuté.
Puis la variable `lettre` est mise à jour avec le second élément de la séquence (`'o'`) et le bloc d'instructions à nouveau exécuté...
Le bloc d'instructions est exécuté une dernière fois lorsqu'on arrive au dernier élément de la séquence (`'r'`) :

```
>>>
B
o
n
s
o
i
r
Fin de la boucle
```

Exemple avec les éléments d'une liste

```
# script Boucle5.py
```

```
maliste = ['Pierre', 67.5, 18]
for element in maliste:
    print(element)
print("Fin de la boucle")
```

Là, on affiche dans l'ordre les éléments de la liste :

```
>>>
Pierre
67.5
18
Fin de la boucle
```

Fonction `range()`

L'association avec la fonction `range()` est très utile pour créer des séquences automatiques de nombres entiers :

```
# script Boucle6.py
print(list(range(1,5)))

for i in range(1,5):
    print(i)
print("Fin de la boucle")
>>>
[1, 2, 3, 4]
1
2
3
4
Fin de la boucle
```

Table de multiplication

La création d'une table de multiplication paraît plus simple avec une boucle `for` qu'avec une boucle `while` :

```
# script Boucle7.py

for compteur in range(1,11):
    print(compteur, '* 9 =', compteur*9)
print("Et voilà !")
>>>
1 * 9 = 9
2 * 9 = 18
3 * 9 = 27
4 * 9 = 36
5 * 9 = 45
6 * 9 = 54
7 * 9 = 63
8 * 9 = 72
9 * 9 = 81
10 * 9 = 90
Et voilà !
```

L'instruction break

L'instruction `break` provoque une sortie immédiate d'une boucle `while` ou d'une boucle `for`.

Dans l'exemple suivant, l'expression `True` est toujours ... vraie : on a une boucle sans fin.
L'instruction `break` est donc le seul moyen de sortir de la boucle.

Affichage de l'heure courante

```
# script Boucle8.py

import time      # importation du module time
while True:
    # strftime() est une fonction du module time
    print('Heure courante ', time.strftime('%H:%M:
%S'))
```

```
    quitter = input('Voulez-vous quitter le
programme (o/n) ? ')
    if quitter == 'o':
        break
print("A bientôt")
>>>
Heure courante  14:25:12
Voulez-vous quitter le programme (o/n) ? n
Heure courante  14:25:20
Voulez-vous quitter le programme (o/n) ? o
A bientôt
```

Astuce

Si vous connaissez le nombre de boucles à effectuer, utiliser une boucle `for`.
Autrement, utiliser une boucle `while` (notamment pour faire des boucles sans fin).

Exercices

Exercice 3.1 ★ Ecrire un script qui affiche toutes les tables de multiplication (de 1 à 10).

Exercice 3.2 ★ Ecrire un script qui calcule la moyenne d'une série de notes.
On pourra utiliser une variable qui contient la somme intermédiaire des notes.

```
>>>
Nombre de notes ? 3
-->  15
-->  11.5
-->  14
Moyenne : 13.5
>>>
```

Exercice 3.3 ★
1) Avec une boucle `for`, écrire un script qui compte le nombre de lettres

z dans une chaîne de caractères.
Par exemple :

```
>>>
Entrer la chaîne : Zinedine Zidane
Résultat : 2
```

2) Faire la même chose, directement avec la méthode `count()` de la classe `str`.
Pour obtenir de l'aide sur cette méthode :

```
>>> help(str.count)
```

Exercice 3.4 ★ Avec une boucle `while`, écrire un script qui affiche l'heure courante avec une actualisation chaque seconde.
On utilisera la fonction `sleep()` du module `time`.
Pour obtenir de l'aide sur cette fonction :

```
>>> import time
>>> help(time.sleep)
```

Par exemple :

```
>>> # Taper CTRL + C pour arrêter le programme.
>>>
Heure courante  12:40:59
Heure courante  12:41:00
Heure courante  12:41:01
Heure courante  12:41:02
KeyboardInterrupt
>>>
```

Remarque : il est vilain de forcer l'arrêt d'un programme avec `CTRL + C`
Nous verrons comment interrompre proprement un programme dans le chapitre Gestion des exceptions.

Exercice 3.5 ★☆
1) Ecrire le script du jeu de devinette suivant :

```
>>>
Le jeu consiste à deviner un nombre entre 1 et
100 :
```

```
--->   50
trop petit !
--->   75
trop petit !
--->   87
trop grand !
--->   81
trop petit !
--->   84
trop petit !
--->   85
Gagné en 6 coups !
```

2) Quelle est la stratégie la plus efficace ?

3) Montrer que l'on peut deviner un nombre en 7 coups maximum.

Bibliographie : La dichotomie

Remarque : pour créer un nombre entier aléatoire entre 1 et 100 :

```
import random
nombre = random.randint(1,100)
```

Exercice 3.6 ★★ Code de César
En cryptographie, le code de César est une technique de chiffrement élémentaire qui consiste à décaler une lettre de 3 rangs vers la droite :
A → D
B → E
...
Z → C
1) Ecrire le script de ce codage.
Par exemple :

```
>>>
Message à coder ? abcdefghijklmnopqrstuvwxyz
defghijklmnopqrstuvwxyzabc
>>>
Message à coder ? Monty Python's Flying Circus
prqwb sbwkrq'v ioblqj flufxv
```

On pourra utiliser la chaîne `'abcdefghijklmnopqrstuvwxyz'` et la méthode `find()` de la classe `str`.
Pour obtenir de l'aide sur cette méthode :

```
>>> help(str.find)
```

2) Ecrire le script du décodage.
Par exemple :

```
>>>
Message à décoder ? prqwb sbwkrq'v ioblqj flufxv
monty python's flying circus
```

Exercice 3.7 ★★ Ecrire un script qui donne l'évolution de la suite convergente : $u_{n+1} = u_n/2 + 1/u_n$
Par exemple :

```
>>>
Valeur initiale ? 20
Jusqu'à quel rang ? 10
0 20.0
1 10.05
2 5.12450248756
3 2.75739213842
4 1.74135758045
5 1.44494338196
6 1.41454033013
7 1.41421360012
8 1.41421356237
9 1.41421356237
10 1.41421356237
>>>
```

Exercice 3.8 ★★ Fraction continue infinie

Estimer la valeur numérique de la fraction continue suivante :

$$1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \ldots}}}}$$

Comparer avec la valeur exacte : (1 + √5)/2

Exercice 3.9
1) ★ Ecrire un script qui détermine si un nombre entier est premier ou pas.
Par exemple :

```
>>>
Nombre ? 17
17 est un nombre premier
```

2) ★★ Ecrire un script qui décompose un nombre entier en un produit de facteurs premiers.

```
>>>
Nombre à décomposer ? 2142
```

Chapitre 4 - Les fonctions

Nous avons déjà vu beaucoup de fonctions : `print()`, `type()`, `len()`, `input()`, `range()`...
Ce sont des fonctions pré-définies (built-in functions).
Nous avons aussi la possibilité de créer nos propres fonctions !

Intérêt des fonctions

Une fonction est une portion de code que l'on peut appeler au besoin (c'est une sorte de sous-programme).

L'utilisation des fonctions évite des redondances dans le code : on obtient ainsi des programmes plus courts et plus lisibles.

Par exemple, nous avons besoin de convertir à plusieurs reprises des degrés Celsius en degrés Fahrenheit :

```
>>> print(100.0*9.0/5.0 + 32.0)
212.0
>>> print(37.0*9.0/5.0 + 32.0)
98.6
>>> print(233.0*9.0/5.0 + 32.0)
451.4
```

La même chose en utilisant une fonction :

```
>>> def fahrenheit(degre_celsius):
```

```
        """ Conversion degré Celsius en degré
Fahrenheit """
        print(degre_celsius*9.0/5.0 + 32.0)
>>> fahrenheit(100)
212.0
>>> fahrenheit(37)
98.6
>>> temperature = 233
>>> fahrenheit(temperature)
451.4
```

Rien ne vous oblige à définir des fonctions dans vos scripts, mais cela est tellement pratique qu'il serait improductif de s'en passer !

L'instruction def

Syntaxe

```
def nom_de_la_fonction(parametre1, parametre2,
parametre3, ...):
    """ Documentation
qu'on peut écrire
sur plusieurs lignes """  # docstring entouré de 3
guillemets (ou apostrophes)

    bloc d'instructions         # attention à
l'indentation

    return resultat         # la fonction retourne le
contenu de la variable resultat
```

Exemple n°1

```
# script Fonction1.py

def mapremierefonction(): # cette fonction n'a pas
de paramètre
    """ Cette fonction affiche 'Bonjour' """
```

```
    print("Bonjour")
    return              # cette fonction ne retourne
rien ('None')
                        # l'instruction return est ici
facultative
```

Une fois la fonction définie, nous pouvons l'appeler :

```
>>> mapremierefonction()  # ne pas oublier les
parenthèses ()
Bonjour
```

L'accès à la documentation se fait avec la fonction pré-définie `help()` :

```
>>> help(mapremierefonction)    # affichage de la
documentation
Help on function mapremierefonction in module
__main__:


mapremierefonction()
    Cette fonction affiche 'Bonjour'
```

Exemple n°2

La fonction suivante simule le comportement d'un dé à 6 faces.
Pour cela, on utilise la fonction `randint()` du module random.

```
# script Fonction2.py


def tirage_de():
    """ Retourne un nombre entier aléatoire entre 1
et 6 """
    import random
    valeur = random.randint(1, 6)
    return valeur
>>> print(tirage_de())
3
>>> print(tirage_de())
6
```

```
>>> resultat = tirage_de()
>>> print(resultat)
1
```

Exemple n°3

```
# script Fonction3.py

# définition des fonctions
def info():
    """ Informations """
    print("Touche q pour quitter")
    print("Touche Enter pour continuer")

def tirage_de():
    """ Retourne un nombre entier aléatoire entre 1
et 6 """
    import random
    valeur = random.randint(1, 6)
    return valeur

# début du programme
info()
while True:
    choix = input()
    if choix == 'q':
        break
    print("Tirage :", tirage_de())
>>>
Touche q pour quitter
Touche Enter pour continuer
```

```
Tirage : 5


Tirage : 6
q
>>>
```

Exemple n°4

Une fonction avec deux paramètres :

```
# script Fonction4.py


# définition de fonction
def tirage_de2(valeur_min, valeur_max):
    """ Retourne un nombre entier aléatoire entre
valeur_min et valeur_max """
    import random
    return random.randint(valeur_min, valeur_max)


# début du programme
for i in range(5):
    print(tirage_de2(1, 10))    # appel de la
fonction avec les arguments 1 et 10
>>>
6
7
1
10
2
>>>
```

Exemple n°5

Une fonction qui retourne une liste :

```
# script Fonction5.py
```

```
# définition de fonction
def tirage_multiple_de(nombretirage):
    """ Retourne une liste de nombres entiers aléatoires entre 1 et 6 """
    import random
    resultat = [random.randint(1, 6) for i in range(nombretirage)] # compréhension de listes (Cf. annexe)
    return resultat


# début du programme
print(tirage_multiple_de(10))
>>>
[4, 1, 3, 3, 2, 1, 6, 6, 2, 5]
>>> help(tirage_multiple_de)
Help on function tirage_multiple_de in module __main__:


tirage_multiple_de(nombretirage)
    Retourne une liste de nombres entiers aléatoires entre 1 et 6
```

Exemple n°6

Une fonction qui affiche la parité d'un nombre entier.
Il peut y avoir plusieurs instructions `return` dans une fonction.
L'instruction `return` provoque le retour immédiat de la fonction.

```
# script Fonction6.py


# définition de fonction
def parite(nombre):
    """ Affiche la parité d'un nombre entier """
    if nombre%2 == 1:   # L'opérateur % donne le reste d'une division
```

```
        print(nombre, 'est impair')
        return
    if nombre%2 == 0:
        print(nombre, 'est pair')
        return
>>> parite(13)
13 est impair
>>> parite(24)
24 est pair
```

Portée de variables : variables globales et locales

La *portée d'une variable* est l'endroit du programme où on peut accéder à la variable.

Observons le script suivant :

```
a = 10          # variable globale au programme


def mafonction():
    a = 20 # variable locale à la fonction
    print(a)
    return
>>> print(a)          # nous sommmes dans l'espace global du programme
10
>>> mafonction()      # nous sommes dans l'espace local de la fonction
20
>>> print(a)          # de retour dans l'espace global
10
```

Nous avons deux variables différentes qui portent le même nom a

Une variable a de valeur 20 est créée dans la fonction : c'est une *variable locale* à la fonction.
Elle est détruite dès que l'on sort de la fonction.

global

L'instruction `global` rend une variable globale :

```
a = 10          # variable globale

def mafonction():
    global a    # la variable est maintenant globale
    a = 20
    print(a)
    return
>>> print(a)
10
>>> mafonction()
20
>>> print(a)
20
```

Remarque : il est préférable d'éviter l'utilisation de l'instruction `global` car c'est une source d'erreurs (on peut ainsi modifier le contenu d'une variable globale en croyant agir sur une variable locale). La sagesse recommande donc de suivre la règle suivante :

- ne jamais affecter dans un bloc de code local une variable de même nom qu'une variable globale

Annexe : la compréhension de listes

La *compréhension de listes* est une structure syntaxique disponible dans un certain nombre de langages de programmation, dont Python.
C'est une manière de créer efficacement des listes.

Revenons sur l'exemple vu dans le script `Fonction5.py` :

```
resultat = [random.randint(1, 6) for i in range(10)
]
>>> print(resultat)
[3, 1, 5, 6, 4, 2, 1, 1, 3, 1]
```

Autre exemple : liste de carrés

```
carres = [i*i for i in range(11)]
```

```
>>> print(carres)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

La compréhension de listes évite donc d'écrire le code "classique" suivant :

```
carres = []
for i in range(11):
    carres.append(i*i)
```

Exercices

Exercice 4.1 ☆

1) Ecrire une fonction `carre()` qui retourne le carré d'un nombre :

```
>>> print(carre(11.11111))
123.4567654321
```

2) Avec une boucle `while` et la fonction `carre()`, écrire un script qui affiche le carré des nombres entiers de 1 à 100 :

```
>>>
1² = 1
2² = 4
3² = 9
...
99² = 9801
100² = 10000
Fin du programme
```

Exercice 4.2 ☆

1) Ecrire une fonction qui retourne l'aire de la surface d'un disque de rayon R.
Exemple :

```
>>> print(airedisque(2.5))
19.6349540849
```

2) Ajouter un paramètre qui précise l'unité de mesure :

```
>>> print(airedisque2(4.2, 'cm'))
55.4176944093 cm²
```

Exercice 4.3 ★

1) Ecrire une fonction qui retourne la factorielle d'un nombre entier N.

On rappelle que : N ! = 1×2×...×(N-1)×N
Exemple :

```
>>> print(factorielle(50))
30414093201713378043612608166064768844377641568960512000000000000
```

2) Comparez avec le résultat de la fonction factorial() du module math.

Exercice 4.4 ★
1) A l'aide de la fonction randint() du module random, écrire une fonction qui retourne un mot de passe de longueur N (chiffres, lettres minuscules ou majuscules).
On donne :

```
chaine = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
```

```
>>> print(password(10))
mHVeC5rs8P
>>> print(password(6))
PYthoN
```

2) Reprendre la question 1) avec la fonction choice() du module random.
Pour obtenir de l'aide sur cette fonction :

```
>>> import random
>>> help(random.choice)
```

3) Quel est le nombre de combinaisons possibles ?
4) Quelle durée faut-il pour casser le mot de passe avec un logiciel capable de générer 1 million de combinaisons par seconde ?

Lien utile : www.exhaustif.com/Generateur-de-mot-de-passe-en.html

Exercice 4.5 ★ Ecrire une fonction qui retourne une carte (au hasard) d'un jeu de Poker à 52 cartes.
On utilisera la fonction choice() ou randint() du module random.
On donne :

```
ListeCarte = ['2s','2h','2d','2c','3s','3h','3d','3c','4s','4h',
'4d','4c','5s','5h','5d','5c',
'6s','6h','6d','6c','7s','7h','7d','7c','8s','8h','8d','8c','9s','9h','9d','9c',
'Ts','Th','Td','Tc','Js','Jh','Jd','Jc','Qs','Qh','Qd','Qc','Ks','Kh','Kd','Kc','As','Ah','Ad','Ac']
```

```
>>> print(tiragecarte())
7s
>>> print(tiragecarte())
Kd
```

Exercice 4.6 ★★
1) Ecrire une fonction qui retourne une liste de N cartes **différentes** d'un jeu de Poker à 52 cartes.
Noter qu'une fonction peut appeler une fonction : on peut donc réutiliser la fonction `tiragecarte()` de l'exercice précédent.
Exemple :

```
>>> print(tirage_n_carte(2))
['As', 'Ah']
>>> print(tirage_n_carte(25))
['Jc', 'Jh', 'Tc', '2d', '3h', 'Qc', '8d', '7c',
'As', 'Td', '8h', '9c', 'Ad', 'Qh',
'Kc', '6s', '5h', 'Qd', 'Kh', '9h', '5d', 'Js',
'Ks', '5c', 'Th']
```

2) Simplifier le script avec la fonction `shuffle()` ou `sample()` du module `random`.

Exercice 4.7 ★ Ecrire une fonction qui retourne une grille de numéros du jeu Euro Millions.
On utilisera la fonction `sample()` du module `random`.

```
>>> print(euromillions())
[37, 23, 9, 11, 49, 2, 11]
>>> print(euromillions())
[16, 32, 8, 30, 40, 6, 4]
```

Exercice 4.8 ★★

1) Ecrire une fonction qui retourne la valeur de la fonction mathématique $f(x)= 27x^3 -27x^2 -18x +8$:

```
>>> print(f(0), f(1), f(0.5), f(0.25), f(0.375))
8.0 -10.0 -4.375 2.234375 -1.123046875
```

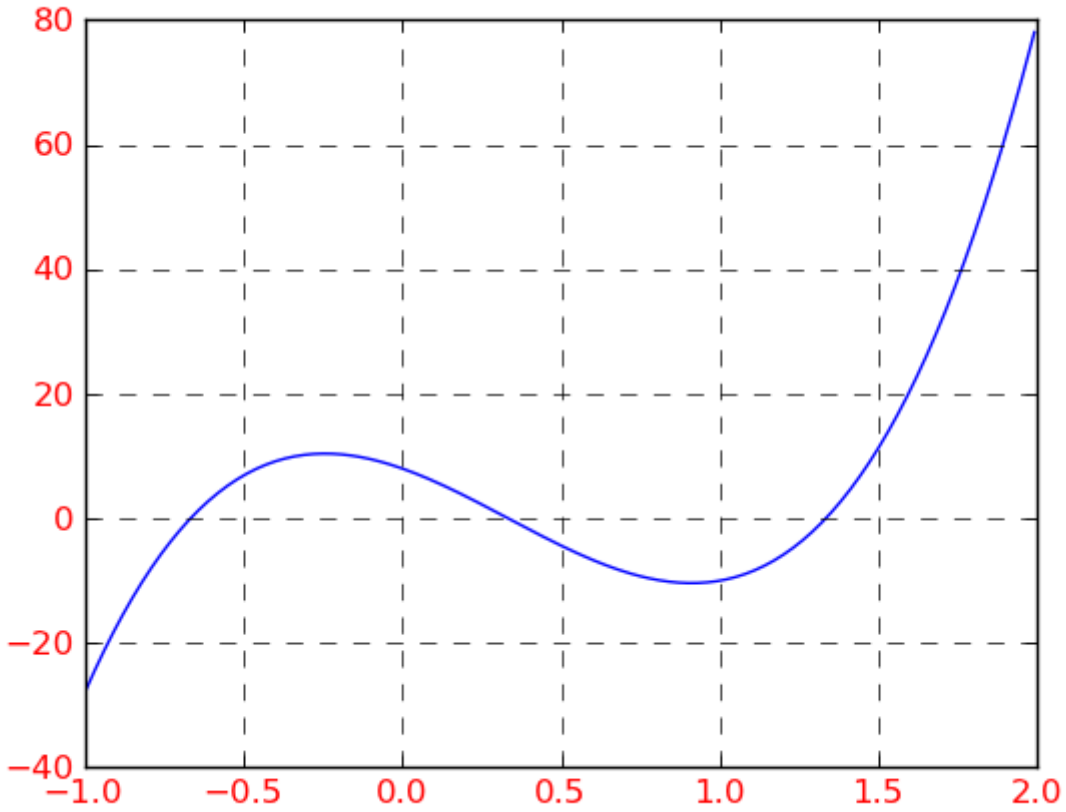
2) On se propose de chercher les zéros de cette fonction par la méthode de dichotomie.
Ecrire le script correspondant.

```
>>>
Recherche d'un zéro dans l'intervalle [a,b]
a? 0
b? 1
Précision ? 1e-12
0.5
0.25
0.375
0.3125
0.34375
0.328125
0.3359375
0.33203125
0.333984375
0.3330078125
0.33349609375
0.333251953125
...
...
```

```
0.333333333333
>>>
```

3) Chercher tous les zéros de cette fonction.

Annexe : représentation graphique de la fonction $f(x)= 27x^3 -27x^2 -18x +8$ (graphique réalisé avec la librairie matplotlib de Python)



Chapitre 5 - Gestion des exceptions

Intéressons-nous au script suivant :

```
# script inverse.py
chaine = input('Entrer un nombre : ')
nombre = float(chaine)
inverse = 1.0/nombre
print("L'inverse de", nombre, "est :", inverse)
```

Ce script vous demande de saisir un nombre, puis il calcule et affiche son inverse.

Les exceptions

Quand vous entrez un nombre, tout se déroule normalement :

```
>>>
Entrer un nombre : 10
L'inverse de 10.0 est : 0.1
>>>
```

Mais que se passe-t-il autrement ?

```
>>>
Entrer un nombre : bonjour
Traceback (most recent call last):
  File "inverse.py", line 3, in <module>
    nombre = float(chaine)
ValueError: could not convert string to float:
bonjour
>>>
>>>
Entrer un nombre : 0
Traceback (most recent call last):
  File "inverse.py", line 4, in <module>
    inverse = 1.0/nombre
ZeroDivisionError: float division by zero
>>>
```

Python a détecté une erreur : une **exception est levée**.
Ici nous avons une exception de type `ZeroDivisionError` (division par 0) et une exception de type `ValueError`.
Une exception arrête l'exécution normale d'un programme.

Gestion des exceptions

Heureusement, il est possible de gérer les exceptions pour éviter l'arrêt brutal du programme.
Par cela, on utilise conjointement les instructions `try` et `except`.
L'instruction `else` est optionnelle :

```
try:
```

```
    chaine = input('Entrer un nombre : ')
    nombre = float(chaine)
    inverse = 1.0/nombre
except:
    #ce bloc est exécuté si une exception est levée dans le bloc try
    print("Erreur !")
else:
    #on arrive ici si aucune exception n'est levée dans le bloc try
    print("L'inverse de", nombre, "est :", inverse)
>>>
Entrer un nombre : 56
L'inverse de 56.0 est : 0.0178571428571
>>>
Entrer un nombre : 0
Erreur !
>>>
```

On peut distinguer les différents types d'exceptions :

```
try:
    chaine = input('Entrer un nombre : ')
    nombre = float(chaine)
    inverse = 1.0/nombre
except ValueError:
    #ce bloc est exécuté si une exception de type ValueError est levée dans le bloc try
    print(chaine, "n'est pas un nombre !")
except ZeroDivisionError:
    #ce bloc est exécuté si une exception de type ZeroDivisionError est levée dans le bloc try
    print("Division par zéro !")
```

```
else:
    #on arrive ici si aucune exception n'est levée dans le bloc try
    print("L'inverse de", nombre, "est :", inverse)
>>>
Entrer un nombre : 0
Division par zéro !
>>>
Entrer un nombre : bonjour
bonjour n'est pas un nombre !
>>>
```

N'oubliez pas : un programme bien écrit doit gérer proprement les exceptions.

Exercices

Exercice 5.1
1) Compléter le script précédent de manière à ressaisir le nombre en cas d'erreur.
Par exemple :

```
>>>
Entrer un nombre : salut !
salut ! n'est pas un nombre !
Entrer un nombre : 2,3
2,3 n'est pas un nombre !
Entrer un nombre : 2.3
L'inverse de 2.3 est : 0.434782608696
>>>
```

2) Compléter le script de manière à accepter la virgule comme séparateur décimal.
Par exemple :

```
>>>
Entrer un nombre : 2,3
L'inverse de 2.3 est : 0.434782608696
```

```
>>>
```

On pourra utiliser la méthode `replace()` de la classe `str`

Exercice 5.2 Ecrire un script qui calcule la racine carrée d'un nombre, avec gestion des exceptions.
Par exemple :

```
>>>
Entrer un nombre : go
go n'est pas un nombre valide !
Entrer un nombre : -5.26
-5.26 n'est pas un nombre valide !
Entrer un nombre : 16
La racine carrée de 16.0 est : 4.0
>>>
```

Exercice 5.3 Soit le script :

```
i = 0
while True:
    i += 1
    print(i)
```

Il s'agit d'une boucle sans fin.
Pour arrêter ce programme, il faut appuyer sur les touches `CTRL + C`, ce qui lève une exception de type `KeyboardInterrupt` :

```
>>>
1
...
1216
1217
1218
1219
1220
Traceback (most recent call last):
    raise KeyboardInterrupt
```

```
KeyboardInterrupt
>>>
```

Compléter le script pour gérer proprement l'exception :

```
>>>
1
...
966
967
968
Fin du programme
>>>
```

Chapitre 6 - Classes et modules

La **classe** est un concept de base de la **programmation orientée objet**. En langage Python, vous pouvez très bien écrire un script sans définir de classes (c'est ce que nous avons fait jusqu'à présent).
Cependant, vous manipulez forcément des **objets (ou instances de classes)** : une variable entière est une instance de la classe `int`, une chaîne de caractères est une instance de la classe `str`...
Le module `Tkinter` (que nous aborderons dans le chapitre 7) sert à créer des interfaces graphiques : ce module fournit une bibliothèque de classes.
Il est donc important d'étudier les classes notamment pour bien comprendre comment les utiliser (pour un débutant, c'est déroutant !).

Ce chapitre est difficile : je vous conseille de relire ce que l'on a vu dans le chapitre 1.

Définition d'une classe

Voici un exemple de script qui utilise une classe définie par l'utilisateur.

Nous allons commencer par créer le fichier source `CompteBancaire.py` (on parlera par la suite du module `CompteBancaire`) :

Ouvrir IDLE :
Démarrer → Programmes → Python → IDLE (Python GUI)

File → New Window
Copier puis coller le code source ci-dessous :

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

```
# CompteBancaire.py

# définition de la classe Compte

class Compte:
    """Un exemple de classe :
    gestion d'un compte bancaire"""

    # définition de la méthode spéciale __init__
    def __init__(self, soldeInitial):
        """Initialisation du compte avec la valeur
soldeInitial."""
        # assignation de l'attribut d'instance solde
        self.solde = float(soldeInitial)

    # définition de la méthode NouveauSolde()
    def NouveauSolde(self, somme):
        """Nouveau solde de compte avec la valeur
somme."""
        self.solde = float(somme)

    # définition de la méthode Solde()
    def Solde(self):
        """Retourne le solde."""
        return self.solde

    # définition de la méthode Credit()
    def Credit(self, somme):
        """Crédite le compte de la valeur somme.
Retourne le solde."""
        self.solde += somme
```

```
        return self.solde

    # définition de la méthode Debit()
    def Debit(self, somme):
        """Débite le compte de la valeur somme.
Retourne le solde."""
        self.solde -= somme
        return self.solde

    # définition de la méthode spéciale __add__
(surcharge de l'opérateur +)
    def __add__(self, somme):
        """x.__add__(somme) <=> x+somme
        Crédite le compte de la valeur somme.
        Affiche 'Nouveau solde : somme'"""
        self.solde += somme
        print("Nouveau solde : {:+.2f}
€".format(self.solde))
        return self

    # définition de la méthode spéciale __sub__
(surcharge de l'opérateur -)
    def __sub__(self, somme):
        """x.__sub_(somme) <=> x-somme
        Débite le compte de la valeur somme.
        Affiche 'Nouveau solde : somme'"""
        self.solde -= somme
        print("Nouveau solde : {:+.2f}
€".format(self.solde))
        return self

if __name__ == '__main__':
```

```
    # Ce bloc d'instructions est exécuté si le
module est lancé en tant que programme autonome
    # Instanciation de l'objet cb1 de la classe
Compte
    cb1 = Compte(1000)
    # formatage des données pour afficher deux
chiffres après la virgule et le signe
    print("{:+.2f}".format(cb1.Solde()))
    print("{:+.2f}".format(cb1.Credit(200)))
    print("{:+.2f}".format(cb1.Debit(50.23)))
    print("{:+.2f}".format(cb1.Solde()))
    cb1.NouveauSolde(5100)
    print("{:+.2f}".format(cb1.Solde()))
    cb1+253.2
    cb1-1000+100
    cb1-cb1.Solde()
```

File → Save As
Répertoire : C:\PythonXX
Nom du fichier : CompteBancaire.py

Puis exécuter le module :
Run → Run Module (ou touche F5) :

```
>>>
+1000.00
+1200.00
+1149.77
+1149.77
+5100.00
Nouveau solde : +5353.20 €
Nouveau solde : +4353.20 €
Nouveau solde : +4453.20 €
Nouveau solde : +0.00 €
>>>
```

L'instruction `class`

Pour définir la nouvelle classe `Compte`, on utilise l'instruction `class`.
Une classe possède des fonctions que l'on appelle **méthodes** et des données que l'on appelle **attributs**.

Une méthode se définit de la même manière qu'une fonction (on commence par l'instruction `def`).
Une méthode possède au moins un paramètre qui s'appelle `self`.
Le paramètre `self` désigne toutes les instances qui seront créées par cette classe.

7 méthodes sont ainsi définies.

Un seul attribut d'instance est utilisé dans cette classe : `solde` (à ne pas confondre avec la méthode `Solde()`).

La méthode spéciale `__init__()`

La méthode spéciale `__init__()` est exécutée automatiquement lorsque l'on instancie (crée) un nouvel objet de la classe.
Cette méthode s'apparente à un constructeur.

Autres méthodes spéciales

Deux autres méthodes spéciales sont utilisées : `__add__()` et `__sub__()`.

L'écriture `cb1+253.2` est équivalente à `cb1.__add__(253.2)`
L'opérateur + représente ici une addition sur le solde.
L'écriture `cb1-1000` est équivalente à `cb1.__sub__(1000)`
L'opérateur - représente ici une soustraction sur le solde.

```
>>> cb1.NouveauSolde(500)
>>> print(cb1.Solde())
500.0
>>> cb1.__add__(1000)
Nouveau solde : +1500.00 €
>>> cb1+2000    # cette écriture est très pratique !
Nouveau solde : +3500.00 €
>>>
```

L'instruction `if __name__ == '__main__':`

Ici, le module est exécuté en tant que programme principal : les instructions qui suivent sont donc exécutées.

Dans le cas où ce module est importé dans un autre programme, cette partie du code est sans effet.

Documentation

La fonction `help()` est très utile.
On comprend ici l'intérêt de bien documenter ses programmes avec les `"""docstrings"""` :

```
>>> help(cb1)
Help on instance of Compte in module __main__:


class Compte
 |  Un exemple de classe :
 |  gestion d'un compte bancaire
 |
 |  Methods defined here:
 |
 |  Credit(self, somme)
 |      Crédite le compte de la valeur somme.
Retourne le solde.
 |
 |  Debit(self, somme)
 |      Débite le compte de la valeur somme.
Retourne le solde.
 |
 |  NouveauSolde(self, somme)
 |      Nouveau solde de compte avec la valeur
somme.
 |
 |  Solde(self)
 |      Retourne le solde.
 |
 |  __add__(self, somme)
 |      x.__add__(somme) <=> x+somme
```

```
 |      Crédite le compte de la valeur somme.
 |      Affiche 'Nouveau solde : somme'
 |
 |  __init__(self, soldeInitial)
 |      Initialisation du compte avec la valeur
soldeInitial.
 |
 |  __sub__(self, somme)
 |      x.__sub_(somme) <=> x-somme
 |      Débite le compte de la valeur somme.
 |      Affiche 'Nouveau solde : somme'
>>>
```

La fonction `dir()` retourne la liste des méthodes et attributs :

```
>>> dir(cb1)
['Credit', 'Debit', 'NouveauSolde', 'Solde',
'__add__',
'__doc__', '__init__', '__module__', '__sub__',
'solde']
```

Remarques

On peut instancier plusieurs objets d'une même classe :

```
>>> cb2 = Compte(10000)
>>> print(cb2.Credit(500))
10500.0
>>> cb3 = Compte(6000)    # et encore un !
>>> cb3-500
Nouveau solde : +5500.00 €
>>>
```

Même si cela n'est pas recommandable, vous pouvez accéder directement aux attributs d'instance :

```
>>> print(cb2.solde)
10500.0
```

```
>>> cb2.solde = 5000.0     # assignation de
l'attribut d'instance solde
>>> print(cb2.solde)
5000.0
>>> print(cb2.Solde())
5000.0
>>>
```

Notez que des mécanismes sont prévus pour personnaliser le mode d'accès aux attributs : fonction `property()` ou décorateur `@property`

Importation d'un module défini par l'utilisateur

L'importation de notre module personnel `CompteBancaire` se fait de la même façon que pour les modules de base (`math, random, time`...).

Redémarrer l'interpréteur interactif (Python Shell) :
Shell → Restart Shell

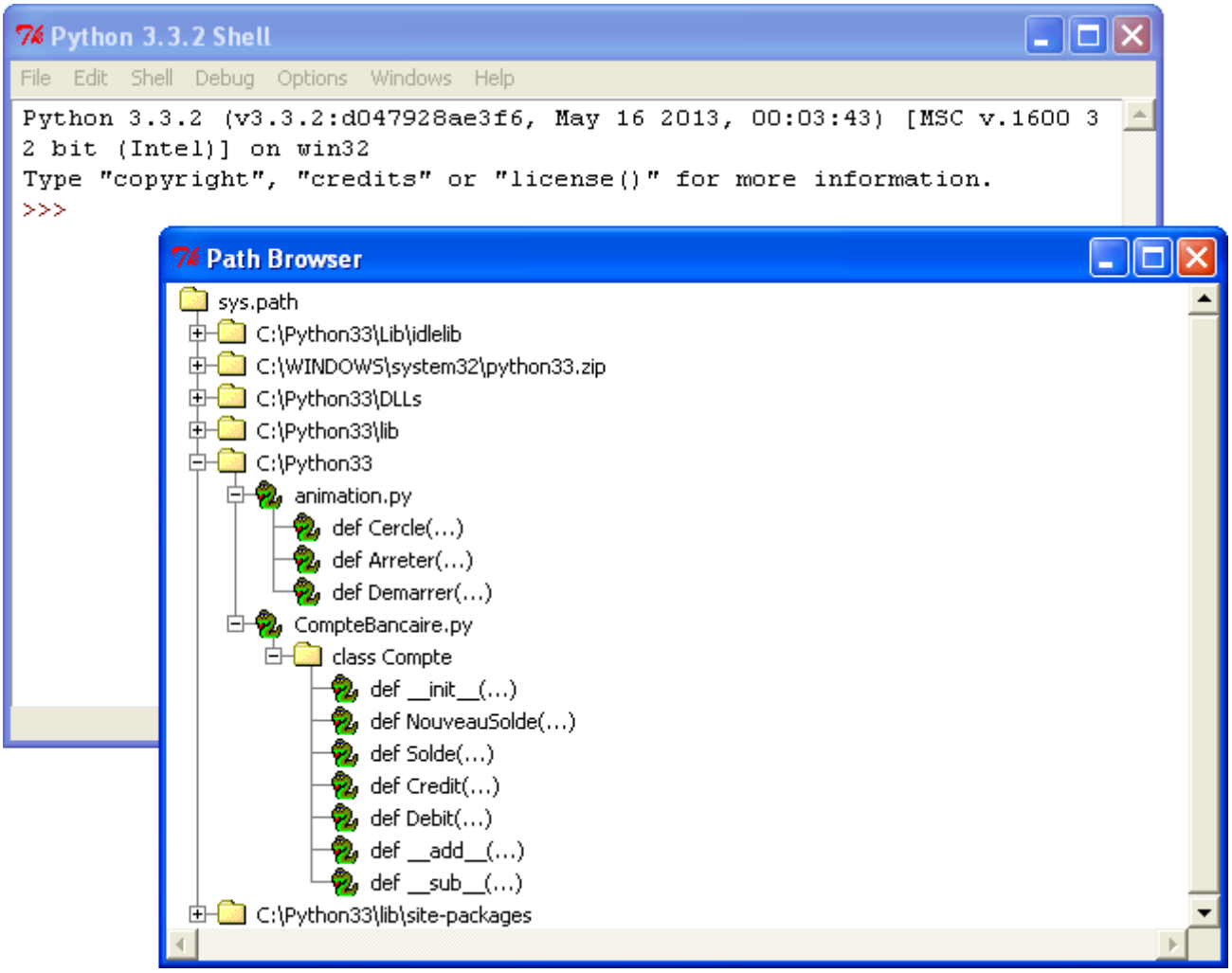
```
>>> import CompteBancaire
>>> cb = CompteBancaire.Compte(1500)
>>> print(cb.Debit(200))
1300.0
>>> print(cb.solde)
1300.0
>>> cb+2000
Nouveau solde : +3300.00 €
>>>
```

Le module `CompteBancaire` est importé par le programme principal (ici l'interpréteur interactif).
La partie du code qui suit l'instruction `if __name__ == '__main__':` est ignorée (il n'y a donc pas d'affichage après l'instruction `import CompteBancaire`).

Modules de classes, modules de fonctions

IDLE dispose de l'outil **Path Browser** pour connaître la structure d'un module.

File → Path Browser



Un module peut contenir des classes (`CompteBancaire`), des fonctions (`animation`) ou des fonctions et des classes.

Autre exemple

Le module standard `math` est un module de fonctions.
Pour s'en convaincre :

```
>>> import math
>>> help(math)
Help on built-in module math:

NAME
    math

FILE
    (built-in)
```

```
DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.

    acosh(...)
        acosh(x)

        Return the hyperbolic arc cosine (measured in radians) of x.
...
...

    trunc(...)
        trunc(x:Real) -> Integral

        Truncates x to the nearest Integral toward 0. Uses the __trunc__ magic method.

DATA
    e = 2.718281828459045
    pi = 3.141592653589793
```

Le module `math` a également deux données (`pi` et `e`) :

```
>>> print(math.e)          # donnée e du module math (nombre d'Euler)
2.71828182846
>>> print(math.pi)         # donnée pi du module math (nombre pi)
3.14159265359
>>> print(math.sin(math.pi/4.0))  # fonction sin() du module math (sinus)
0.707106781187
>>> print(math.sqrt(2.0)) # fonction sqrt() du module math (racine carrée)
1.41421356237
>>> print(math.exp(-3.0)) # fonction exp() du module math (exponentielle)
0.0497870683679
>>> print(math.log(math.e))    # fonction log() du module math (logarithme népérien)
1.0
```

Héritage de classes

Si vous êtes familiarisé avec le module `Tkinter`, voici un exemple qui montre toute la puissance des classes et de la notion d'héritage :


```
# -*- coding: utf-8 -*-
from tkinter import *

class EnterMessage(Frame):
    """ Classe EnterMessage (Frame de saisie du message)
```

```
    Cette classe dérive de la classe
Tkinter.Frame"""
    def __init__(self, master=None):
        """Initialisation : création d'un widget Fr
ame"""
        Frame.__init__(self, master, bg='navy')
        self.pack(padx=10, pady=10)
        self.CreateWidgets()

    def CreateWidgets(self):
        """ Création des widgets Entry et Button
dans le widget Frame """
        self.NouveauMessage = StringVar()
        Entry(master=self, textvariable=self.Nouvea
uMessage).pack(side=LEFT, padx=10, pady=10)
        Button(master=self, text="Nouveau", fg="nav
y", command=self.Nouveau).pack(padx=10, pady=10)

    def Nouveau(self):
        """ Création d'une instance de la classe Ne
wPostIt """
        if self.NouveauMessage.get() != "":
            NewPostIt(master=self.master, message=s
elf.NouveauMessage.get())
            self.NouveauMessage.set("")


class NewPostIt(Frame):
    """ Classe post-it (Frame Post-It)
    Cette classe dérive de la classe Tkinter.Frame"
""
    def __init__(self, master=None, message=None):
        """Initialisation : création d'un widget Fr
ame"""
        Frame.__init__(self, master, bg="maroon")
        self.pack(side=LEFT, padx=10, pady=10)
        self.CreateWidgets(message)

    def CreateWidgets(self, message):
        """ Création des widgets Label et Button da
ns le widget Frame"""
        Label(master=self, text=message, fg='maroon
', bg='white').pack(padx=10, pady=10)
        Button(master=self, text="Effacer", fg="nav
y", command=self.destroy).pack(padx=10, pady=10)


if __name__ == '__main__':
```

```
    # création de la fenêtre principale
    Mafenetre = Tk()
    Mafenetre.title('Post-it')
    Mafenetre['bg'] = 'bisque'

    # Création d'une instance de la classe EnterMes
sage
    EnterMessage(master=Mafenetre)

    Mafenetre.mainloop()
```

Exercices

Exercice 6.0 ★ On s'intéresse au module CompteBancaire.
Dans les méthodes `__add__()` et `__sub__()`, remplacer la ligne `return self` par simplement `return`
Tester ce nouveau code et expliquer la différence.

Exercice 6.1 ★ On s'intéresse au module CompteBancaire.
Définir une nouvelle méthode `Decouvert()` de la classe `Compte`, qui affiche selon le cas **Solde positif** ou **Solde négatif** :

```
>>> CompteFabrice = Compte(2000)
>>> CompteFabrice-3000
Nouveau solde : -1000.00 €
>>> CompteFabrice.Decouvert()
Solde négatif
>>> CompteFabrice+10000
Nouveau solde : +9000.00 €
>>> CompteFabrice.Decouvert()
Solde positif
>>>
```

Exercice 6.2 ★★ On s'intéresse au module CompteBancaire.
Définir une nouvelle méthode `Suivi()` qui affiche l'historique du solde.
On pourra définir un nouvel attribut de type `list` :

```
>>> CompteMuriel = Compte(5000)
>>> CompteMuriel-1000
Nouveau solde : +4000.00 €
>>> CompteMuriel+5000
```

```
Nouveau solde : +9000.00 €
>>> CompteMuriel.Suivi()
[5000.0, 4000.0, 9000.0]
>>>
```

Chapitre 7 - Interface graphique avec le module Tkinter

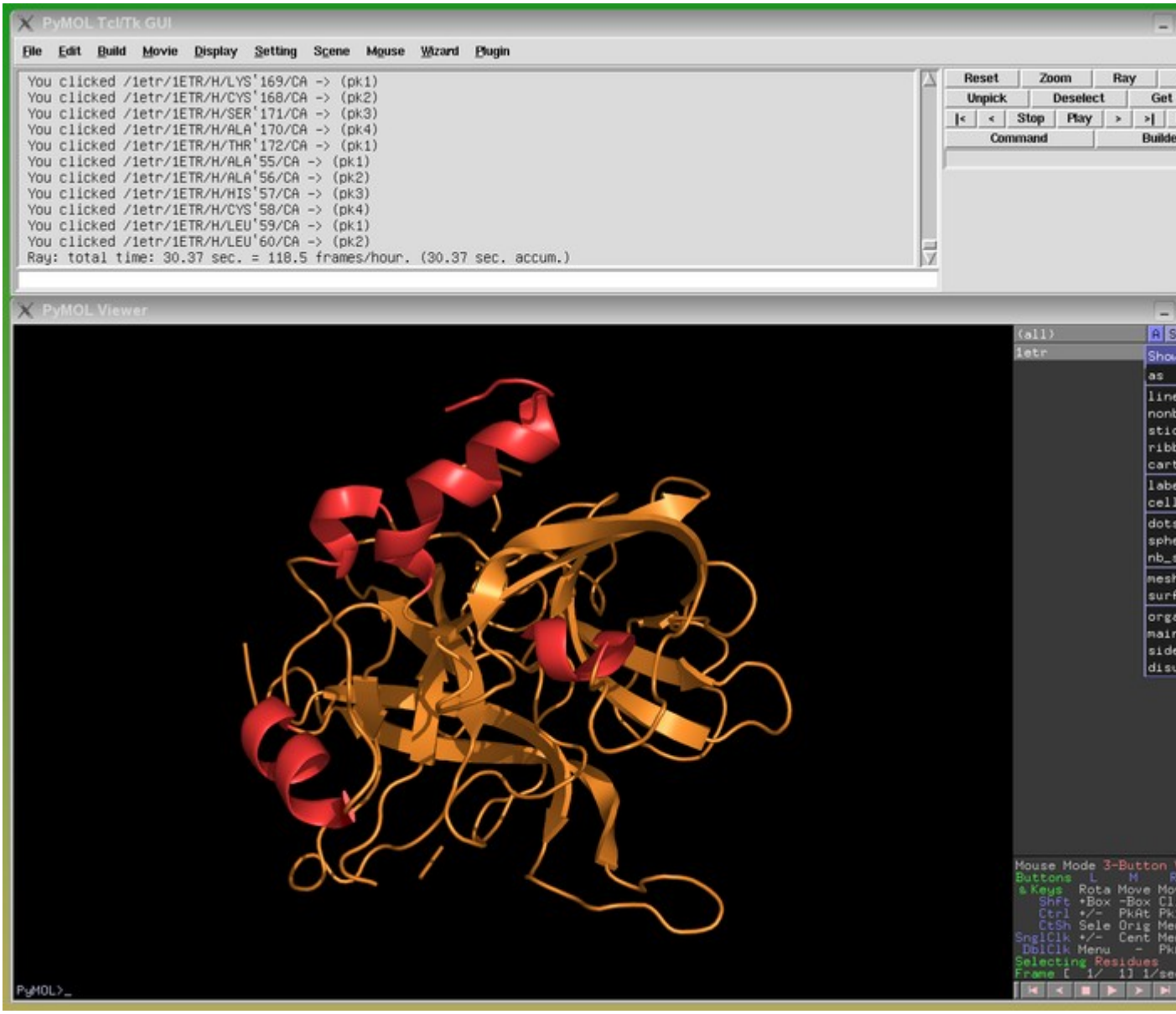
Le module `Tkinter` ("Tk interface") de Python permet de créer des interfaces graphiques (GUI : graphical user interface).

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`), cadre (classe `Frame`)...

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

Logiciels utilisant Python et sa bibliothèque graphique Tkinter

`Tkinter` est l'interface graphique des logiciels IDLE (environnement de développement intégré pour le langage Python) et PyMOL (logiciel libre de visualisation de structures chimiques en 3D) :

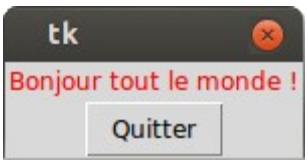


Des scripts pour débuter

Les exemples qui suivent ont été écrits avec Python version 3.

Exemple n°0 : widgets Button et Label

Commençons par le traditionnel Hello world !




```
# -*- coding: utf-8 -*-
# script bonjour.py
from tkinter import *
```

```
# Création de la fenêtre principale (main window)
Mafenetre = Tk()

# Création d'un widget Label (texte 'Bonjour tout
le monde !')
Label1 = Label(Mafenetre, text = 'Bonjour tout le
monde !', fg = 'red')
# Positionnement du widget avec la méthode pack()
Label1.pack()

# Création d'un widget Button (bouton Quitter)
Bouton1 = Button(Mafenetre, text = 'Quitter', comma
nd = Mafenetre.destroy)
Bouton1.pack()

# Lancement du gestionnaire d'événements
Mafenetre.mainloop()
```

Ce code est détaillé ici.

Exemple n°1 : widgets Button et Label

Ce script simule un dé à 6 faces :

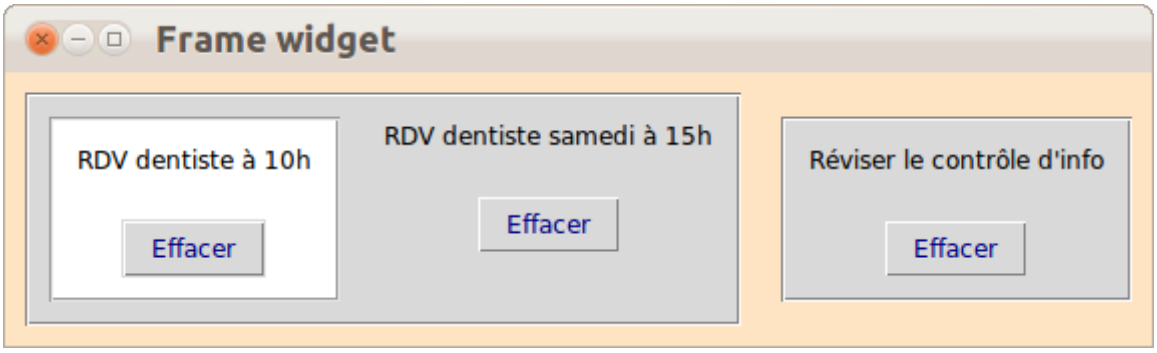

```
# script de.py
#(C) Fabrice Sincère


from tkinter import *
import random


def NouveauLance():
    nb = random.randint(1,6)
    Texte.set('Résultat -> ' + str(nb))


# Création de la fenêtre principale (main window)
```

```
Mafenetre = Tk()


Mafenetre.title('Dé à 6 faces')
Mafenetre.geometry('300x100+400+400')


# Création d'un widget Button (bouton Lancer)
BoutonLancer = Button(Mafenetre, text ='Lancer',
command = NouveauLance)
# Positionnement du widget avec la méthode pack()
BoutonLancer.pack(side = LEFT, padx = 5, pady = 5)


# Création d'un widget Button (bouton Quitter)
BoutonQuitter = Button(Mafenetre, text ='Quitter',
command = Mafenetre.destroy)
BoutonQuitter.pack(side = LEFT, padx = 5, pady = 5)


Texte = StringVar()
NouveauLance()


# Création d'un widget Label (texte 'Résultat ->
x')
LabelResultat = Label(Mafenetre, textvariable =
Texte, fg ='red', bg ='white')
LabelResultat.pack(side = LEFT, padx = 5, pady = 5)


Mafenetre.mainloop()
```

Ce code est détaillé ici.

Exemple n°2 : widgets Frame, Label et Button

Un widget Frame est une zone rectangulaire qui peut contenir d'autres widgets.

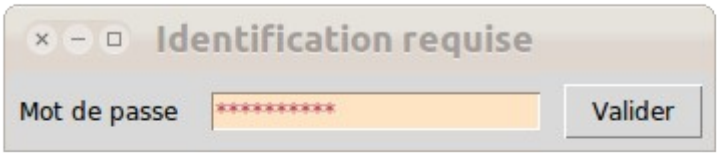


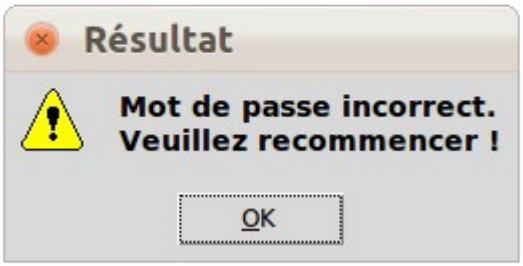

```
# script frames.py
#(C) Fabrice Sincère

from tkinter import *

# Création de la fenêtre principale
Mafenetre = Tk()
Mafenetre.title('Frame widget')
Mafenetre['bg']='bisque' # couleur de fond

# création d'un widget Frame dans la fenêtre princi
pale
Frame1 = Frame(Mafenetre,borderwidth=2,relief=GROOV
E)
Frame1.pack(side=LEFT,padx=10,pady=10)

# création d'un second widget Frame dans la fenêtre
principale
Frame2 = Frame(Mafenetre,borderwidth=2,relief=GROOV
E)
Frame2.pack(side=LEFT,padx=10,pady=10)

# création d'un widget Frame... dans un widget Fram
e
# le widget Frame1 est le parent du widget Frame3
# le parent du widget Frame1 est le widget Mafenetr
e (fenêtre principale)
Frame3 = Frame(Frame1,bg="white",borderwidth=2,reli
ef=GROOVE)
Frame3.pack(side=LEFT,padx=10,pady=10)

# création d'un widget Label et d'un widget Button
dans un widget Frame
Label(Frame1,text="RDV dentiste samedi à 15h").pack
(padx=10,pady=10)
Button(Frame1,text="Effacer",fg='navy',command=Fram
e1.destroy).pack(padx=10,pady=10)
```

```
Label(Frame2,text="Réviser le contrôle d'info").pac
k(padx=10,pady=10)
Button(Frame2,text="Effacer",fg='navy',command=Fram
e2.destroy).pack(padx=10,pady=10)

Label(Frame3,text="RDV dentiste à 10h",bg="white").
pack(padx=10,pady=10)
Button(Frame3,text="Effacer",fg='navy',command=Fram
e3.destroy).pack(padx=10,pady=10)

Mafenetre.mainloop()
```

Exemple n°3 : widgets Entry, Label, Button et boîte de dialogue MessageBox

Un script d'authentification :

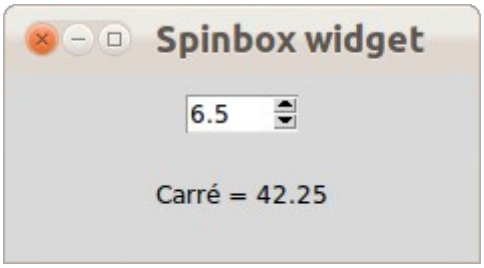





```
# script mot_de_passe.py
#(C) Fabrice Sincère
from tkinter import *
from tkinter.messagebox import * # boîte de
dialogue


def Verification():
    if Motdepasse.get() == 'python27':
        # le mot de passe est bon : on affiche une
boîte de dialogue puis on ferme la fenêtre
        showinfo('Résultat','Mot de passe
correct.\nAu revoir !')
        Mafenetre.destroy()
```

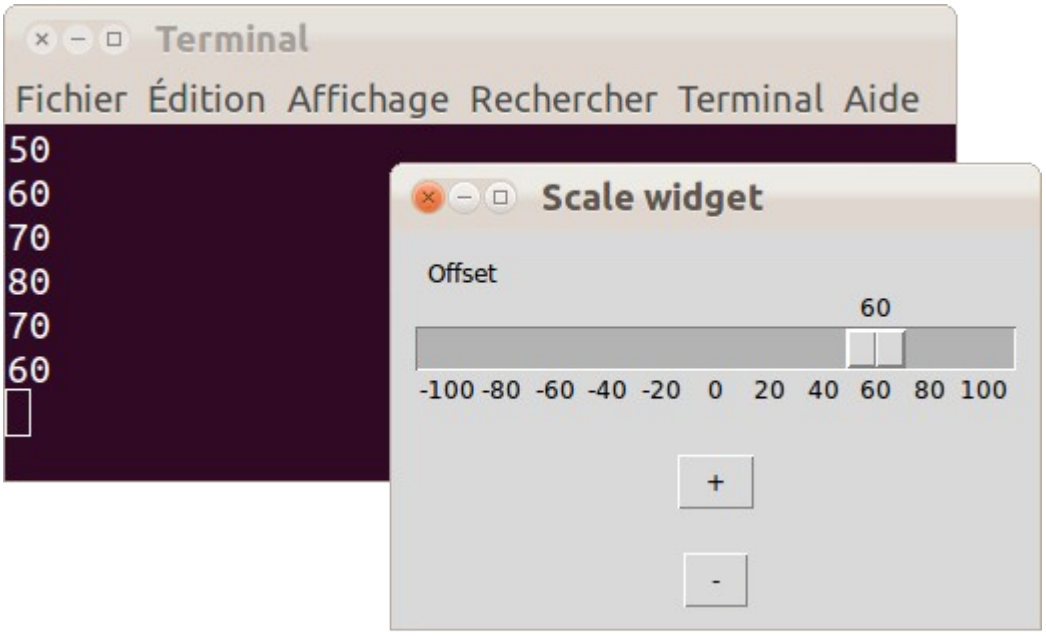
```
    else:
        # le mot de passe est incorrect : on
affiche une boîte de dialogue
        showwarning('Résultat','Mot de passe
incorrect.\nVeuillez recommencer !')
        Motdepasse.set('')


# Création de la fenêtre principale (main window)
Mafenetre = Tk()
Mafenetre.title('Identification requise')

# Création d'un widget Label (texte 'Mot de passe')
Label1 = Label(Mafenetre, text = 'Mot de passe ')
Label1.pack(side = LEFT, padx = 5, pady = 5)

# Création d'un widget Entry (champ de saisie)
Motdepasse= StringVar()
Champ = Entry(Mafenetre, textvariable= Motdepasse,
show='*', bg ='bisque', fg='maroon')
Champ.focus_set()
Champ.pack(side = LEFT, padx = 5, pady = 5)

# Création d'un widget Button (bouton Valider)
Bouton = Button(Mafenetre, text ='Valider', command
= Verification)
Bouton.pack(side = LEFT, padx = 5, pady = 5)


Mafenetre.mainloop()
```

Exemple n°4 : widgets Spinbox et Label




```
# script spinbox.py
#(C) Fabrice Sincère
from tkinter import *

def carre():
    """ Calcul du carré """
    Resultat.set("Carré = "+str(float(Valeur.get())
**2))

# Création de la fenêtre principale (main window)
Mafenetre = Tk()
Mafenetre.title("Spinbox widget")

Valeur = StringVar()
Valeur.set(2.0)
# Création d'un widget Spinbox
boite = Spinbox(Mafenetre,from_=0,to=10,increment=0
.5,textvariable=Valeur,width=5,command=carre)
boite.pack(padx=30,pady=10)

# Création d'un widget Label
Resultat = StringVar()
carre()
Label(Mafenetre,textvariable=Resultat).pack(padx=30
,pady=10)

Mafenetre.mainloop()
```

Exemple n°5 : widgets Scale et Button

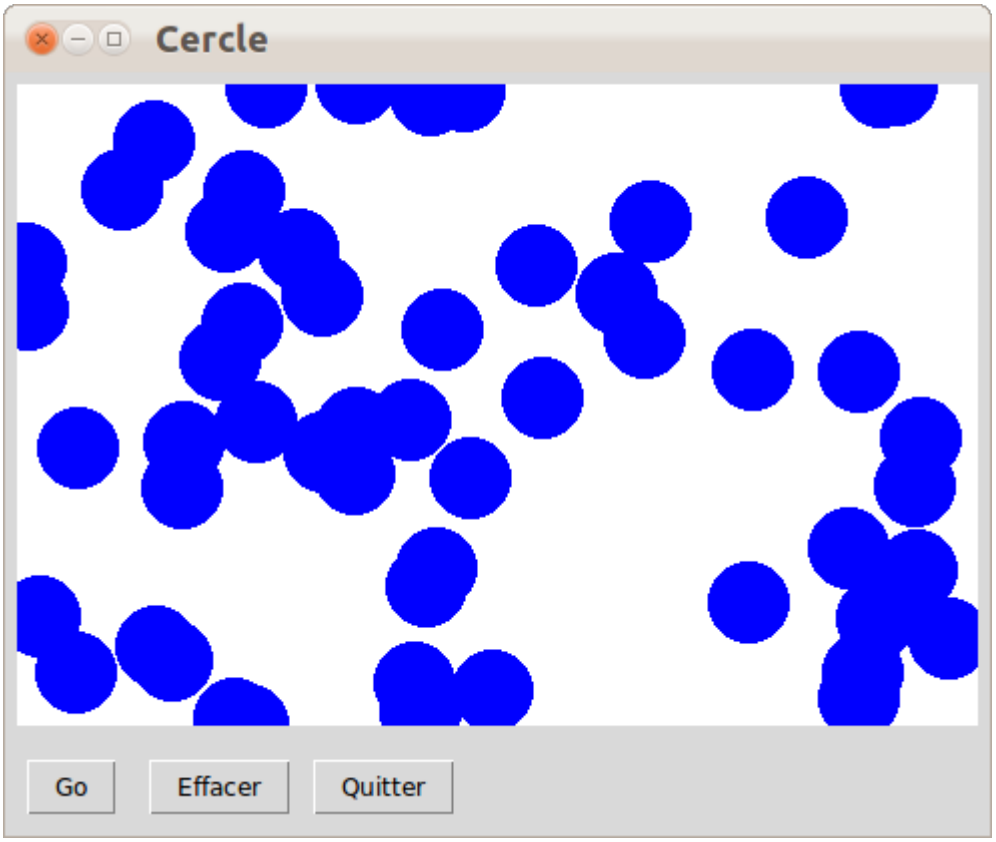



```
# script scale.py
#(C) Fabrice Sincère
from tkinter import *

def maj(nouvelleValeur):
    # nouvelle valeur en argument
    print(nouvelleValeur)
def plus():
    Valeur.set(str(int(Valeur.get())+10))
    print(Valeur.get())
def moins():
    Valeur.set(str(int(Valeur.get())-10))
    print(Valeur.get())

# Création de la fenêtre principale (main window)
Mafenetre = Tk()
Mafenetre.title("Scale widget")

Valeur = StringVar()
Valeur.set(50)
# Création d'un widget Scale
echelle = Scale(Mafenetre,from_=-
100,to=100,resolution=10,orient=HORIZONTAL,\
length=300,width=20,label="Offset",tickinterval=20,
variable=Valeur,command=maj)
echelle.pack(padx=10,pady=10)

# Création d'un widget Button (bouton +)
Button(Mafenetre,text="+",command=plus).pack(padx=1
0,pady=10)
```

```
# Création d'un widget Button (bouton -)
Button(Mafenetre,text="-",command=moins).pack(padx=
10,pady=10)

Mafenetre.mainloop()
```

Exemple n°6 : widgets Canvas et Button

Le script `cercle.py` dessine, à chaque clic sur le bouton `Go`, un disque de rayon 20 pixels à une position aléatoire :




```
# script cercle.py
#(C) Fabrice Sincère
from tkinter import *
import random

def Cercle():
    """ Dessine un cercle de centre (x,y) et de
rayon r """
    x = random.randint(0,Largeur)
    y = random.randint(0,Hauteur)
```

```
    r = 20
    Canevas.create_oval(x-r, y-r, x+r, y+r,
outline='blue', fill='blue')


def Effacer():
    """ Efface la zone graphique """
    Canevas.delete(ALL)


# Création de la fenêtre principale (main window)
Mafenetre = Tk()
Mafenetre.title('Cercle')


# Création d'un widget Canvas (zone graphique)
Largeur = 480
Hauteur = 320
Canevas = Canvas(Mafenetre, width = Largeur, height
=Hauteur, bg ='white')
Canevas.pack(padx =5, pady =5)


# Création d'un widget Button (bouton Go)
BoutonGo = Button(Mafenetre, text ='Go', command =
Cercle)
BoutonGo.pack(side = LEFT, padx = 10, pady = 10)


# Création d'un widget Button (bouton Effacer)
BoutonEffacer = Button(Mafenetre, text ='Effacer',
command = Effacer)
BoutonEffacer.pack(side = LEFT, padx = 5, pady = 5)


# Création d'un widget Button (bouton Quitter)
BoutonQuitter = Button(Mafenetre, text ='Quitter',
command = Mafenetre.destroy)
```

```
BoutonQuitter.pack(side = LEFT, padx = 5, pady = 5)


Mafenetre.mainloop()
```

Ce code est détaillé ici.

Exemple n°7 : widgets Canvas et Button ; gestion des images

Ce script reprend le script `cercle.py` avec une image de fond (méthode `create_image()` de la classe `Canvas`) et la possibilité d'effacer la dernière action (pour cela, on se sert du numéro identifiant de chaque item d'un widget Canvas) :


Cible
1
2
3
4
5
6
Tirer
Effacer le dernier tir
Effacer tout
Quitter

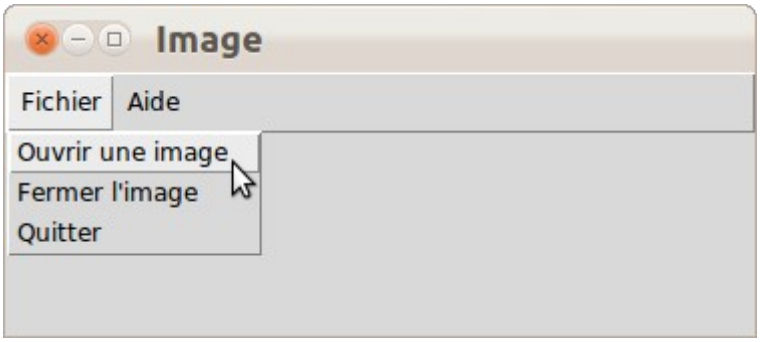
```
# script cible.py
#(C) Fabrice Sincère
from tkinter import *
import random

def Cercle():
    """ Dessine un cercle de centre (x,y) et de ray
on r """
    x = random.randint(0,Largeur)
    y = random.randint(0,Hauteur)
    r = 10

    # on dessine un cercle dans la zone graphique
    item = Canevas.create_oval(x-r, y-r, x+r, y+r,
outline='black', fill='black')

    print("Création du cercle (item" , item ,")")
    # affichage de tous les items de Canevas
    print(Canevas.find_all())

def Undo():
    """ Efface le dernier cercle"""
    if len(Canevas.find_all()) > 1:
        item = Canevas.find_all()[-1]
        # on efface le cercle
        Canevas.delete(item)

        print("Suppression du cercle
(item" , item ,")")
        # affichage de tous les items de Canevas
        print(Canevas.find_all())

def EffacerTout():
    """ Efface tous les cercles"""
    while len(Canevas.find_all()) > 1:
        Undo()


# Création de la fenêtre principale (main window)
Mafenetre = Tk()
Mafenetre.title('Cible')

# Image de fond
photo = PhotoImage(file="tk_cible.gif")

# Création d'un widget Canvas (zone graphique)
Largeur = 550
Hauteur = 550
```

```
Canevas = Canvas(Mafenetre,width = Largeur, height =Hauteur)
item = Canevas.create_image(0,0,anchor=NW, image=photo)
print("Image de fond (item",item,")")
Canevas.pack()

# Création d'un widget Button
BoutonGo = Button(Mafenetre, text ='Tirer', command = Cercle)
BoutonGo.pack(side = LEFT, padx = 10, pady = 10)

# Création d'un widget Button
BoutonEffacer = Button(Mafenetre, text ='Effacer le dernier tir', command = Undo)
BoutonEffacer.pack(side = LEFT, padx = 10, pady = 10)

# Création d'un widget Button
BoutonEffacerTout = Button(Mafenetre, text ='Effacer tout', command = EffacerTout)
BoutonEffacerTout.pack(side = LEFT, padx = 10, pady = 10)

# Création d'un widget Button (bouton Quitter)
BoutonQuitter = Button(Mafenetre, text ='Quitter', command = Mafenetre.destroy)
BoutonQuitter.pack(side = LEFT, padx = 10, pady = 10)

Mafenetre.mainloop()
```

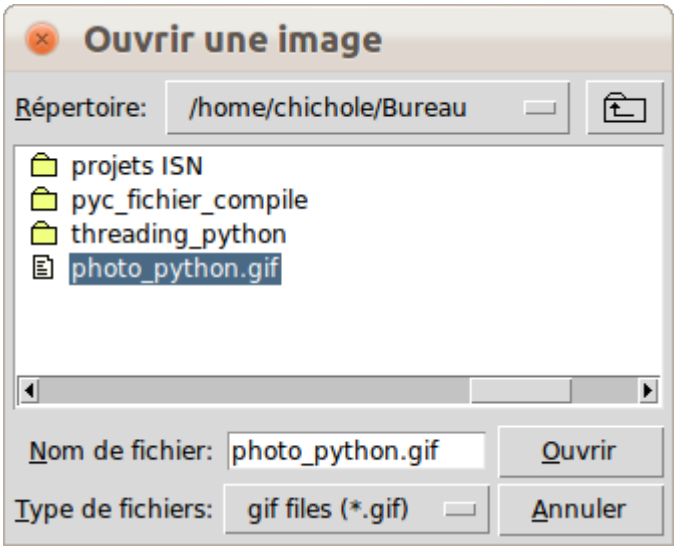
Télécharger l'image de fond tk_cible.gif

Exemple n°8 : widgets Menu et Canvas ; gestion des images ; boîtes de dialogue FileDialog et MessageBox

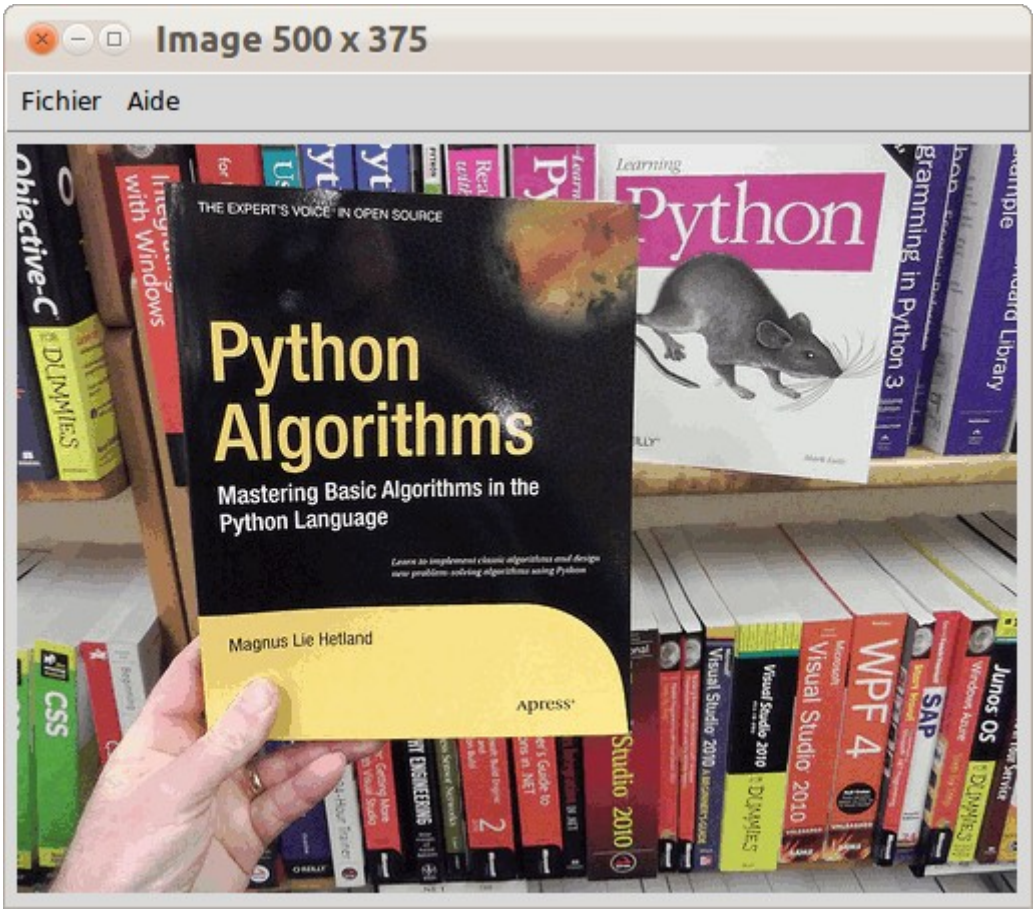
Le script suivant est un browser d'images (formats .gif .ppm .pgm), avec un widget Menu :



une boîte de dialogue `FileDialog` pour rechercher un fichier :



et un widget `Canvas` dans lequel sera affiché l'image :




```
# script lecture_gif.py
#(C) Fabrice Sincère
from tkinter import *
import tkinter.messagebox
import tkinter.filedialog

def Ouvrir():
    Canevas.delete(ALL) # on efface la zone graphiq
ue

    filename = tkinter.filedialog.askopenfilename(t
itle="Ouvrir une image",filetypes=[('gif files','.g
if'),('all files','.*')])
    print(filename)

    photo = PhotoImage(file=filename)
    gifdict[filename] = photo  # référence
    print(gifdict)

    Canevas.create_image(0,0,anchor=NW,image=photo)
    Canevas.config(height=photo.height(),width=phot
o.width())

    Mafenetre.title("Image "+str(photo.width())+" x
```

```
"+str(photo.height()))

def Fermer():
    Canevas.delete(ALL)
    Mafenetre.title("Image")

def Apropos():
    tkinter.messagebox.showinfo("A propos","Tutoria
l Python Tkinter\n(C) Fabrice Sincère")

# Main window
Mafenetre = Tk()
Mafenetre.title("Image")

# Création d'un widget Menu
menubar = Menu(Mafenetre)

menufichier = Menu(menubar,tearoff=0)
menufichier.add_command(label="Ouvrir une image",co
mmand=Ouvrir)
menufichier.add_command(label="Fermer l'image",comm
and=Fermer)
menufichier.add_command(label="Quitter",command=Maf
enetre.destroy)
menubar.add_cascade(label="Fichier", menu=menufichi
er)

menuaide = Menu(menubar,tearoff=0)
menuaide.add_command(label="A propos",command=Aprop
os)
menubar.add_cascade(label="Aide", menu=menuaide)

# Affichage du menu
Mafenetre.config(menu=menubar)

# Création d'un widget Canvas
Canevas = Canvas(Mafenetre)
Canevas.pack(padx=5,pady=5)

# Utilisation d'un dictionnaire pour conserver une
référence
gifdict={}

Mafenetre.mainloop()
```

Remarques

```
# dans le programme principal
gifdict = {}
# dans la fonction Ouvrir()
gifdict[filename] = photo
```

Si vous enlevez les lignes ci-dessus, l'ouverture des images ne fonctionne plus correctement.

En effet, `photo` est une variable locale à la fonction `Ouvrir()`. En dehors, elle n'existe plus et la référence à l'image est perdue.

Une manière de conserver la référence est d'utiliser un dictionnaire ou une liste qui ont "naturellement" une portée globale :

```
# dans le programme principal
giflist = []
# dans la fonction Ouvrir()
giflist.append(photo)
```

On peut aussi rendre globale la variable `photo` avec ce code au début de la fonction `Ouvrir()` :

```
global photo
```

Pour finir, la façon la plus élégante de résoudre ce genre de situation est de pratiquer la programmation orientée objet (encapsulation dans une classe) : voir le programme

Exemple n°9 : gestion du temps

L'heure courante est mise à jour toutes les secondes :

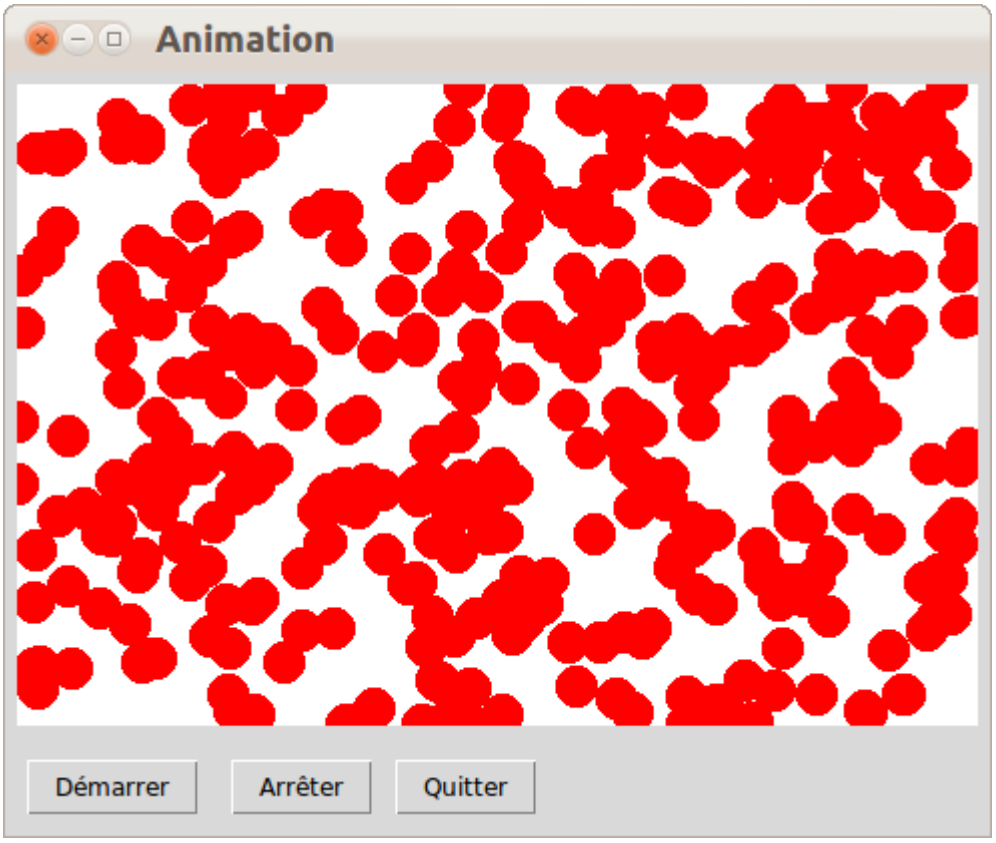
Pour cela, on utilise la méthode `after()` qui appelle une fonction après une durée donnée en millisecondes :

```
# script heure.py
#(C) Fabrice Sincère
from tkinter import *

import time

def maj():
    # on arrive ici toutes les 1000 ms
    heure.set(time.strftime('%H:%M:%S'))
    Mafenetre.after(1000,maj)
```

```
Mafenetre = Tk()
Mafenetre.title("Heure courante")

# Création d'un widget Label
heure = StringVar()
Label(Mafenetre,textvariable=heure).pack(padx=10,pa
dy=10)

maj()

Mafenetre.mainloop()
```

Exemple n°10 : widgets Canvas et Button ; gestion du temps

Le script `animation.py` est un exemple d'animation (affichage d'environ 20 disques par seconde) :



On se sert de la méthode `after()` pour actualiser la zone graphique toutes les 50 ms :

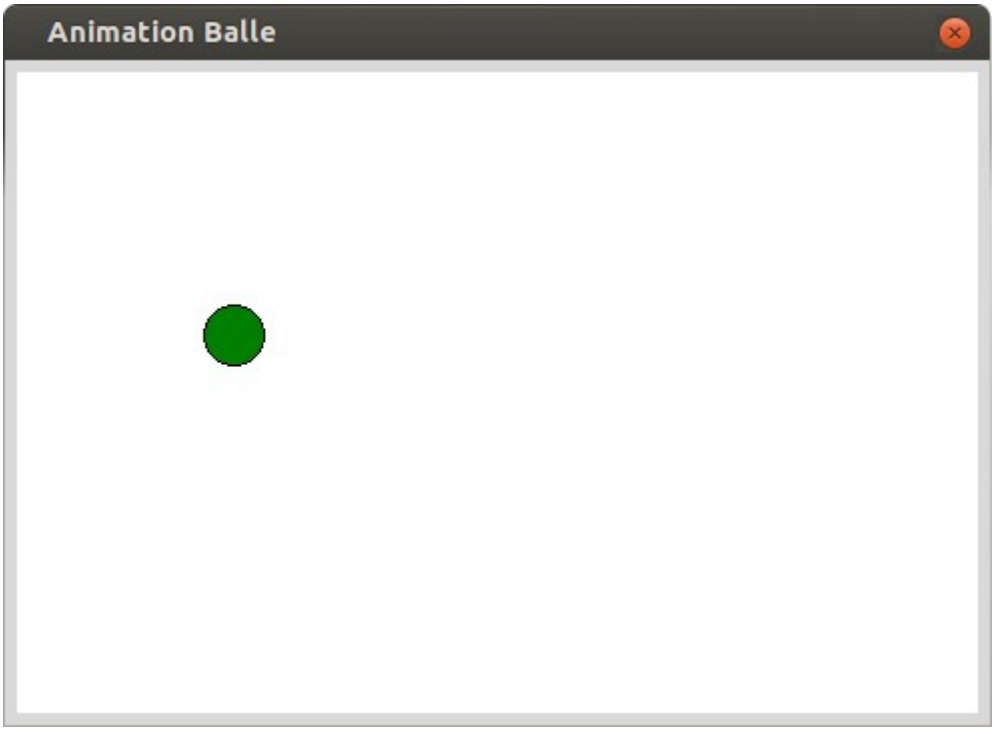
```
# script animation.py
#(C) Fabrice Sincère
from tkinter import *
```

```
import random

def Cercle():
    """ Dessine un cercle de centre (x,y) et de
rayon r """
    global Arret
    x = random.randint(0,Largeur)
    y = random.randint(0,Hauteur)
    r = 10
    Canevas.create_oval(x-r, y-r, x+r, y+r,
outline='red', fill='red')
    if Arret == False:
        # appel de la fonction Cercle() après une
pause de 50 millisecondes
        Mafenetre.after(50,Cercle)

def Arreter():
     """ Arrêt de l'animation """
     global Arret
     Arret = True

def Demarrer():
    """ Démarre l'animation """
    global Arret
    Canevas.delete(ALL)
    if Arret == True:
        Arret = False
        Cercle() # un seul appel à cette fonction

Arret = True
```

```
# Création de la fenêtre principale (main window)
Mafenetre = Tk()
Mafenetre.title('Animation')

# Création d'un widget Canvas
Largeur = 480
Hauteur = 320
Canevas = Canvas(Mafenetre, width = Largeur, height =Hauteur, bg ='white')
Canevas.pack(padx =5, pady =5)

# Création d'un widget Button (bouton Démarrer)
BoutonGo = Button(Mafenetre, text ='Démarrer', command = Demarrer)
BoutonGo.pack(side = LEFT, padx = 10, pady = 10)

# Création d'un widget Button (bouton Arrêter)
BoutonArreter = Button(Mafenetre, text ='Arrêter', command = Arreter)
BoutonArreter.pack(side = LEFT, padx = 5, pady = 5)

# Création d'un widget Button (bouton Quitter)
BoutonQuitter = Button(Mafenetre, text ='Quitter', command = Mafenetre.destroy)
BoutonQuitter.pack(side = LEFT, padx = 5, pady = 5)

Mafenetre.mainloop()
```

Exemple n°11 : widget Canvas ; gestion du temps

Le script `animation_balle.py` est une animation qui gère le déplacement d'une balle et de ses rebonds sur les bords :

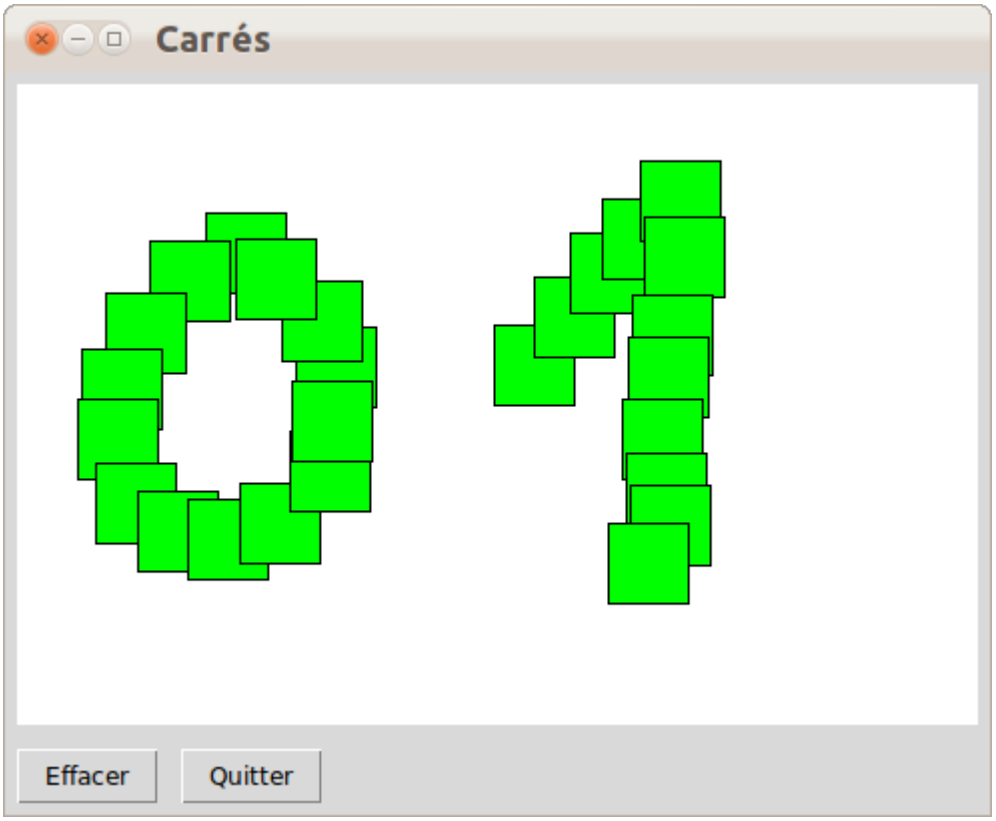



```
# script animation_balle.py
#(C) Fabrice Sincère

from tkinter import *
import math,random

LARGEUR = 480
HAUTEUR = 320
RAYON = 15 # rayon de la balle

# position initiale au milieu
X = LARGEUR/2
Y = HAUTEUR/2

# direction initiale aléatoire
vitesse = random.uniform(1.8,2)*5
angle = random.uniform(0,2*math.pi)
DX = vitesse*math.cos(angle)
DY = vitesse*math.sin(angle)

def deplacement():
    """ Déplacement de la balle """
    global X,Y,DX,DY,RAYON,LARGEUR,HAUTEUR

    # rebond à droite
    if X+RAYON+DX > LARGEUR:
        X = 2*(LARGEUR-RAYON)-X
        DX = -DX
```

```
    # rebond à gauche
    if X-RAYON+DX < 0:
        X = 2*RAYON-X
        DX = -DX

    # rebond en bas
    if Y+RAYON+DY > HAUTEUR:
        Y = 2*(HAUTEUR-RAYON)-Y
        DY = -DY

    # rebond en haut
    if Y-RAYON+DY < 0:
        Y = 2*RAYON-Y
        DY = -DY

    X = X+DX
    Y = Y+DY

    # affichage
    Canevas.coords(Balle,X-RAYON,Y-
RAYON,X+RAYON,Y+RAYON)

    # mise à jour toutes les 50 ms
    Mafenetre.after(50,deplacement)

# Création de la fenêtre principale
Mafenetre = Tk()
Mafenetre.title("Animation Balle")

# Création d'un widget Canvas
Canevas = Canvas(Mafenetre,height=HAUTEUR,width=LAR
GEUR,bg='white')
Canevas.pack(padx=5,pady=5)

# Création d'un objet graphique
Balle = Canevas.create_oval(X-RAYON,Y-
RAYON,X+RAYON,Y+RAYON,width=1,fill='green')

deplacement()
Mafenetre.mainloop()
```

Exemple n°12 : widgets Canvas et Button ; gestion de la souris

Le script `carre.py` dessine un carré à l'endroit du clic de la souris.
Pour cela, on utilise l'événement associé au clic gauche de la souris.



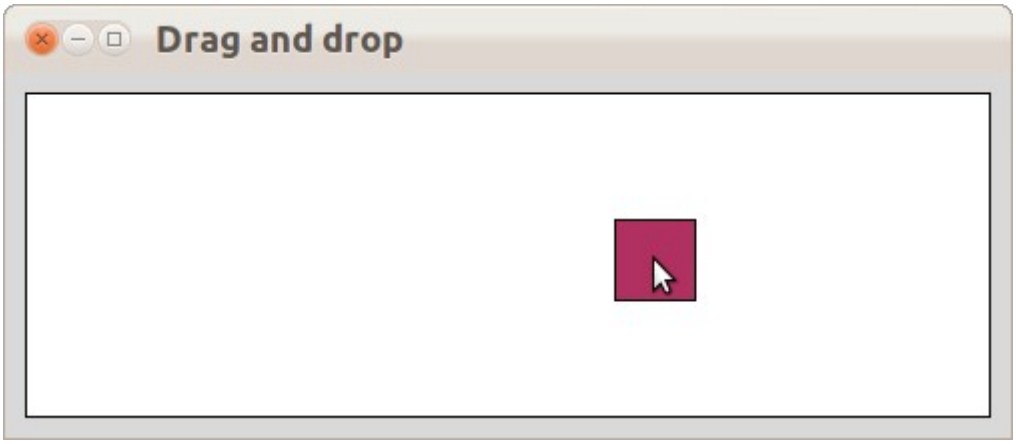

```
# script carre.py
#(C) Fabrice Sincère
from tkinter import *

def Clic(event):
    """ Gestion de l'événement Clic gauche sur la
zone graphique """
    # position du pointeur de la souris
    X = event.x
    Y = event.y
    # on dessine un carré
    r = 20
    Canevas.create_rectangle(X-r, Y-r, X+r, Y+r,
outline='black',fill='green')


def Effacer():
    """ Efface la zone graphique """
```

```
    Canevas.delete(ALL)


# Création de la fenêtre principale
Mafenetre = Tk()
Mafenetre.title('Carrés')


# Création d'un widget Canvas
Largeur = 480
Hauteur = 320
Canevas = Canvas(Mafenetre, width = Largeur, height
=Hauteur, bg ='white')
# La méthode bind() permet de lier un événement
avec une fonction :
# un clic gauche sur la zone graphique provoquera
l'appel de la fonction utilisateur Clic()
Canevas.bind('<Button-1>', Clic)
Canevas.pack(padx =5, pady =5)


# Création d'un widget Button (bouton Effacer)
Button(Mafenetre, text ='Effacer', command =
Effacer).pack(side=LEFT,padx = 5,pady = 5)


# Création d'un widget Button (bouton Quitter)
Button(Mafenetre, text ='Quitter', command =
Mafenetre.destroy).pack(side=LEFT,padx=5,pady=5)


Mafenetre.mainloop()
```

Ce code est détaillé ici.

Exemple n°13 : widget Canvas ; gestion de la souris

Nous allons voir comment déplacer un objet graphique avec la souris (clic, drag and drop) :



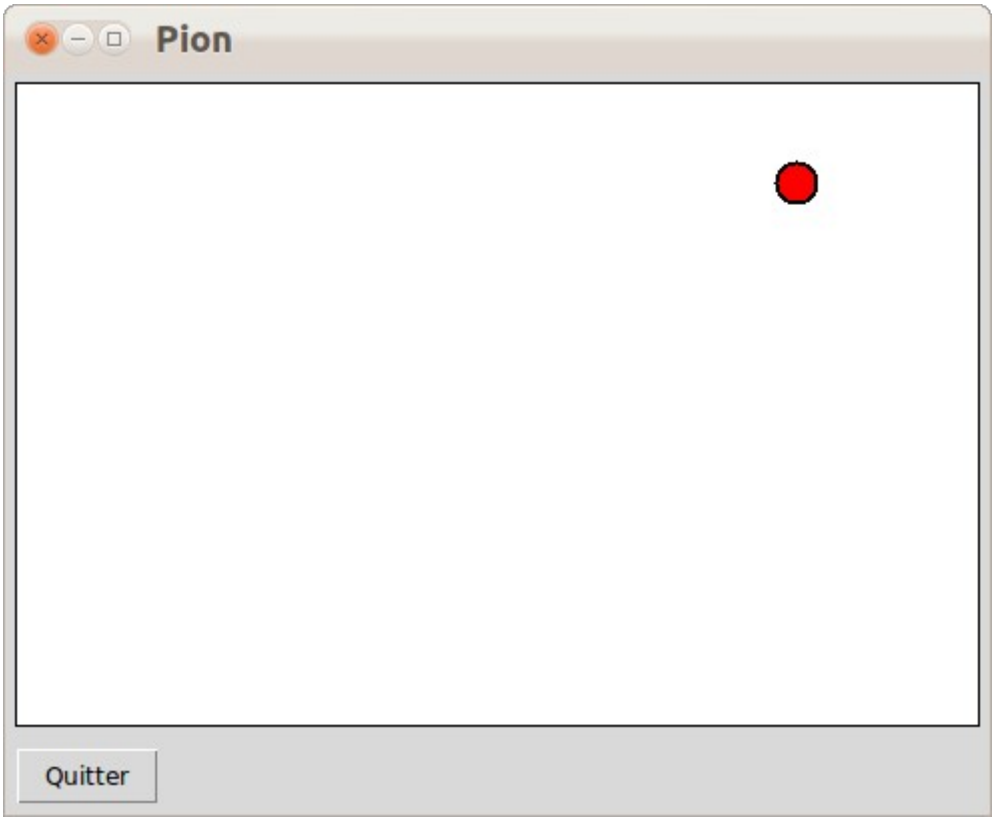

```
# script drag_and_drop.py
#(C) Fabrice Sincère
from tkinter import *

def Clic(event):
    """ Gestion de l'événement Clic gauche """
    global DETECTION_CLIC_SUR_OBJET

    # position du pointeur de la souris
    X = event.x
    Y = event.y
    print("Position du clic -> ",X,Y)

    # coordonnées de l'objet
    [xmin,ymin,xmax,ymax] = Canevas.coords(Carre)

    print("Position objet -> ",xmin,ymin,xmax,ymax)
    if xmin<=X<=xmax and ymin<=Y<=ymax: DETECTION_C
LIC_SUR_OBJET = True
    else: DETECTION_CLIC_SUR_OBJET = False
    print("DETECTION CLIC SUR OBJET ->
",DETECTION_CLIC_SUR_OBJET)

def Drag(event):
    """ Gestion de l'événement bouton gauche enfonc
é """
    X = event.x
    Y = event.y
    print("Position du pointeur -> ",X,Y)

    if DETECTION_CLIC_SUR_OBJET == True:
        # limite de l'objet dans la zone graphique
        if X<0: X=0
        if X>Largeur: X=Largeur
        if Y<0: Y=0
        if Y>Hauteur: Y=Hauteur
```

```
        # mise à jour de la position de l'objet (dr
ag)
        Canevas.coords(Carre,X-TailleCarre,Y-
TailleCarre,X+TailleCarre,Y+TailleCarre)

DETECTION_CLIC_SUR_OBJET = False

# Création de la fenêtre principale
Mafenetre = Tk()
Mafenetre.title("Drag and drop")

# Création d'un widget Canvas
Largeur = 480
Hauteur = 160
TailleCarre = 20
Canevas = Canvas(Mafenetre,width=Largeur,height=Hau
teur,bg ='white')
# Création d'un objet graphique
Carre = Canevas.create_rectangle(0,0,TailleCarre*2,
TailleCarre*2,fill='maroon')

# La méthode bind() permet de lier un événement ave
c une fonction
Canevas.bind('<Button-1>',Clic) # évévement clic ga
uche (press)
Canevas.bind('<B1-Motion>',Drag) # événement bouton
gauche enfoncé (hold down)

Canevas.focus_set()
Canevas.pack(padx=10,pady=10)

Mafenetre.mainloop()
```

Exemple n°14 : widgets Canvas et Button ; gestion du clavier

Le script `pion.py` gère le déplacement d'un pion avec le clavier.
Pour se faire, on utilise l'événement associé à l'appui d'une touche du clavier.

- touche déplacement vers le haut
- touche déplacement vers le bas
- touche déplacement vers la gauche
- touche déplacement vers la droite




```
# script pion.py
#(C) Fabrice Sincère
from tkinter import *

def Clavier(event):
    """ Gestion de l'événement Appui sur une touche
du clavier """
    global PosX,PosY
    touche = event.keysym
    print(touche)
    # déplacement vers le haut
    if touche == 'a':
        PosY -= 20
    # déplacement vers le bas
    if touche == 'q':
        PosY += 20
    # déplacement vers la droite
```

```
    if touche == 'm':
        PosX += 20
    # déplacement vers la gauche
    if touche == 'l':
        PosX -= 20
    # on dessine le pion à sa nouvelle position
    Canevas.coords(Pion,PosX -10, PosY -10, PosX
+10, PosY +10)


# Création de la fenêtre principale
Mafenetre = Tk()
Mafenetre.title('Pion')


# position initiale du pion
PosX = 230
PosY = 150


# Création d'un widget Canvas (zone graphique)
Largeur = 480
Hauteur = 320
Canevas = Canvas(Mafenetre, width = Largeur, height
=Hauteur, bg ='white')
Pion = Canevas.create_oval(PosX-10,PosY-
10,PosX+10,PosY+10,width=2,outline='black',fill='re
d')
Canevas.focus_set()
Canevas.bind('<Key>',Clavier)
Canevas.pack(padx =5, pady =5)


# Création d'un widget Button (bouton Quitter)
Button(Mafenetre, text ='Quitter', command =
Mafenetre.destroy).pack(side=LEFT,padx=5,pady=5)
```

```
Mafenetre.mainloop()
```

Ce code est détaillé ici.

Symboles des quelques touches spéciales

'Up', 'Down', 'Left', 'Right' (flèches directionnelles haut, bas, gauche, droite), 'Return' (touche), 'space' (barre Espace)...

Exemple n°15 : widgets Checkbutton et Button ; musiques et sons avec pygame

Le module pygame est un module externe de création de jeux vidéo en 2D.
pygame contient un sous module pygame.mixer qui permet de charger et de lire des musiques ou des sons dans plusieurs formats (mp3, ogg, wav...).
La procédure d'installation de pygame se trouve ici.

```
# script sons_pygame.py
#(C) Fabrice Sincère
# python version 3.2
# pygame version 1.9.2
from tkinter import *
import pygame

pygame.mixer.init()
pygame.mixer.music.load("chavmusic7.mp3")
# réglage volume
pygame.mixer.music.set_volume(0.3)

son1 = pygame.mixer.Sound("balla1.ogg")
son1.set_volume(0.5)
son2 = pygame.mixer.Sound("death1.wav")
son2.set_volume(1.0)

def PlaySon1():
    son1.play()
def PlaySon2():
    son2.play()
def Music():
```

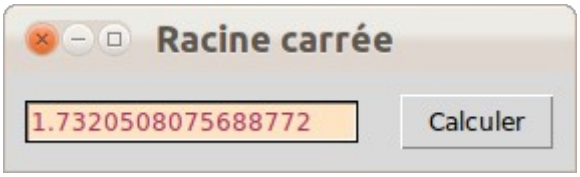
```
    print(musique.get())
    if musique.get() == 1:
        # 1 (ON)
        # joue en boucle
        pygame.mixer.music.play(-1)
    else:
        # 0 (OFF)
        pygame.mixer.music.stop()

# Création de la fenêtre principale (main window)
Mafenetre = Tk()
Mafenetre.title("Checkbutton widget + Pygame.mixer"
)

# Création d'un widget Button
Button(Mafenetre,text="son 1",command=PlaySon1).pac
k(side=LEFT,padx=10,pady=10)

# Création d'un widget Button
Button(Mafenetre,text="son 2",command=PlaySon2).pac
k(side=LEFT,padx=10,pady=10)

# Création d'un widget Checkbutton
musique = IntVar()
musique.set(1) # ON
Checkbutton(Mafenetre,text="Musique de fond",variab
le=musique,command=Music).pack(side=LEFT,padx=10,pa
dy=10)

Music()
Mafenetre.mainloop()
```

Télécharger le son death1.wav
Télécharger le son balla1.ogg
Télécharger la musique chavmusic7.mp3

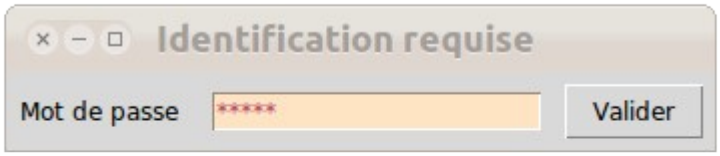
Exercices

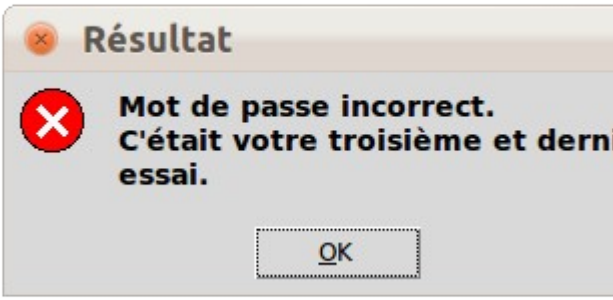
Exercice 7.1 ★ En s'inspirant des scripts `de.py` et `mot_de_passe.py`, écrire une application avec interface graphique qui calcule la racine carrée d'un nombre.

Par exemple, le calcul de √3 donne :

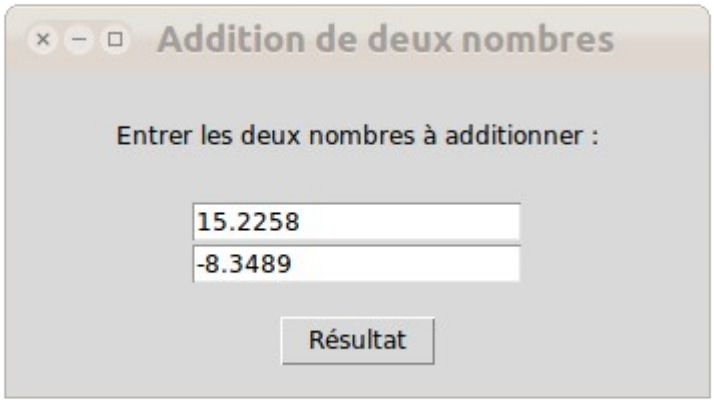


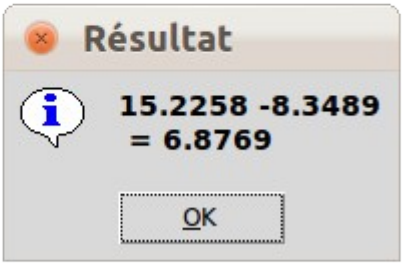
Exercice 7.2 ★ Reprendre le script `mot_de_passe.py` de manière à limiter le nombre d'essais à trois.



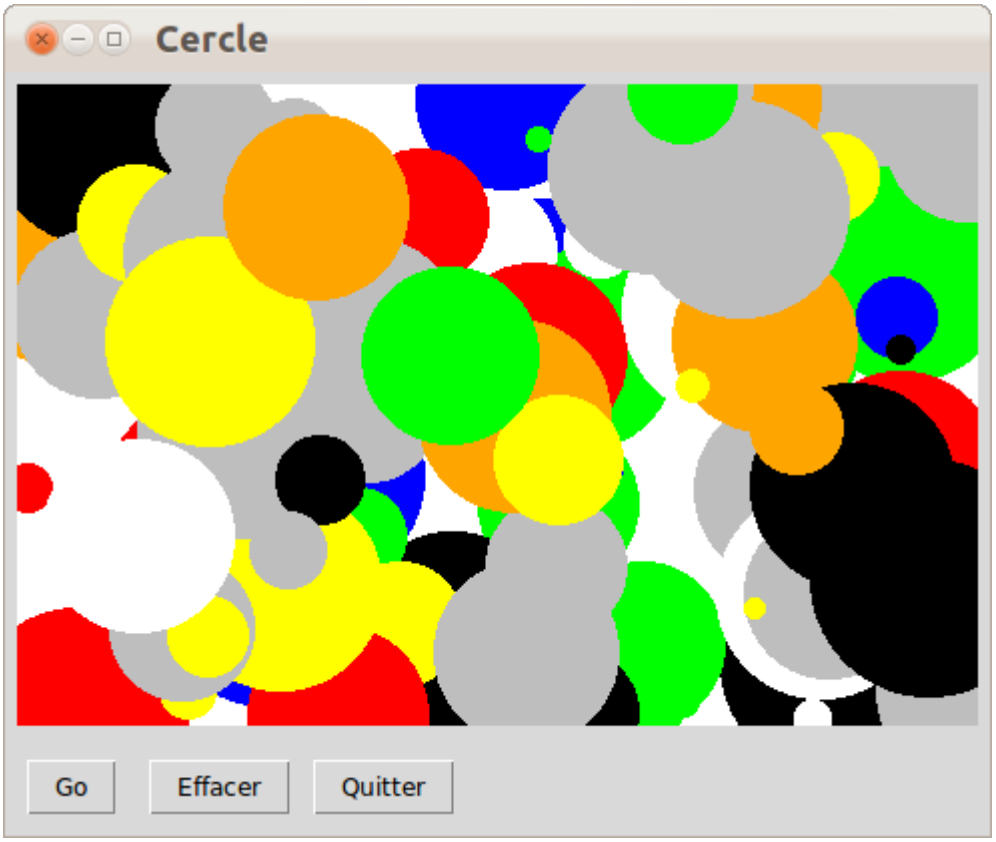


Exercice 7.3 ★ En s'inspirant des scripts `de.py` et `mot_de_passe.py`, écrire une application avec interface graphique qui calcule l'addition ou la soustraction de deux nombres :



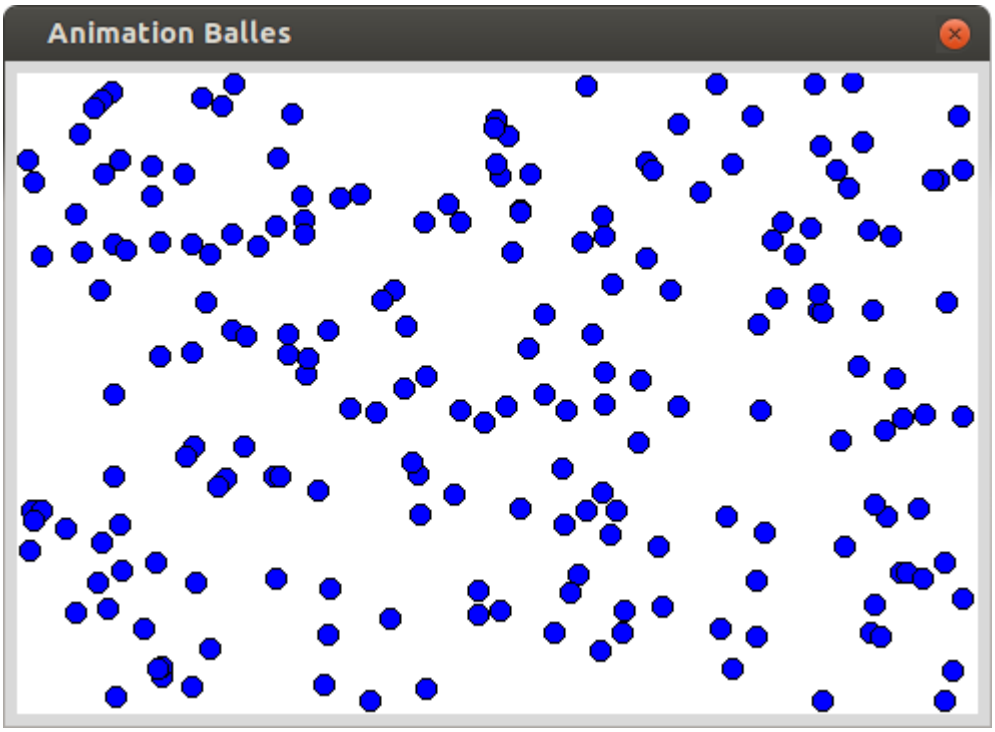


Exercice 7.4 ★ A partir du script `cercle.py`, dessiner des disques de positions, rayons et couleurs aléatoires :



Exercice 7.5 ★ A partir du script `animation.py`, faire une animation avec des disques de positions, rayons et couleurs aléatoires.

Exercice 7.6 ★★ A partir du script `animation_balle.py`, faire une animation qui gère la trajectoire d'un nombre quelconque de balles :



Remarque : on ne tiendra pas compte des chocs entre balles.

Exercice 7.7 ★★

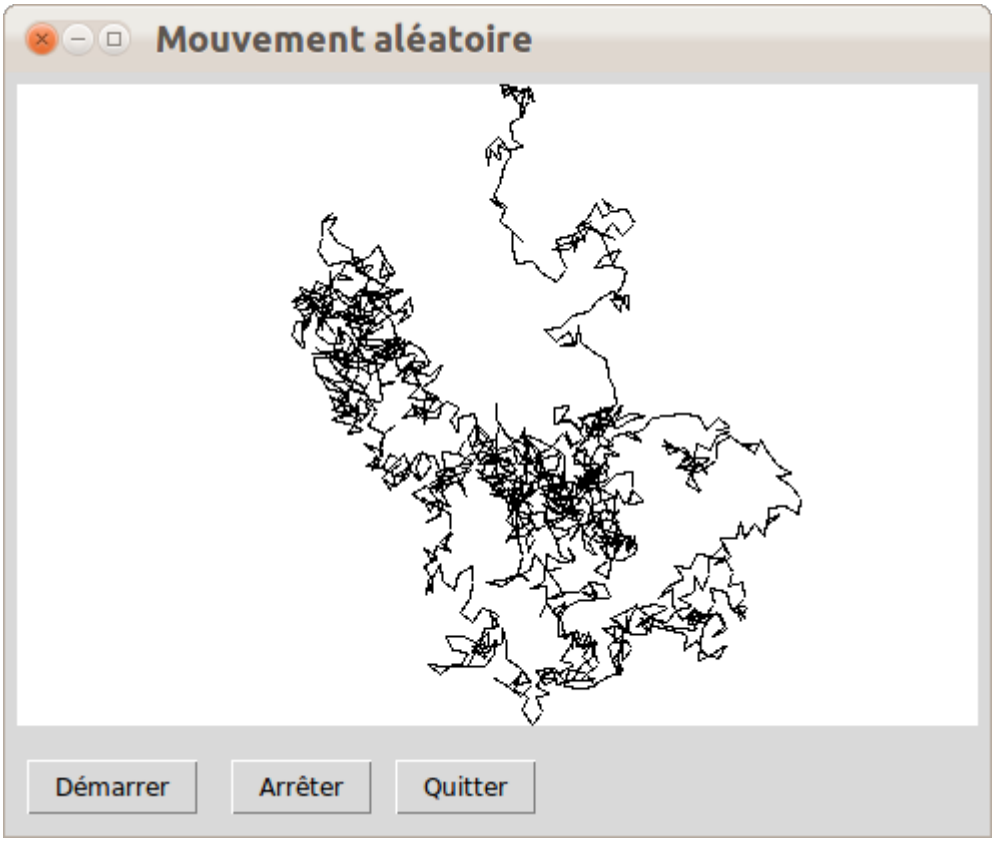
1) Reprendre le script `cible.py` et remplacer le disque noir par une image :

Télécharger l'image impact.gif
Remarque : l'image de l'impact doit avoir un fond transparent.

2) En s'inspirant du script `sons_pygame.py`, ajouter un effet sonore (tk_coup_fusil.wav).

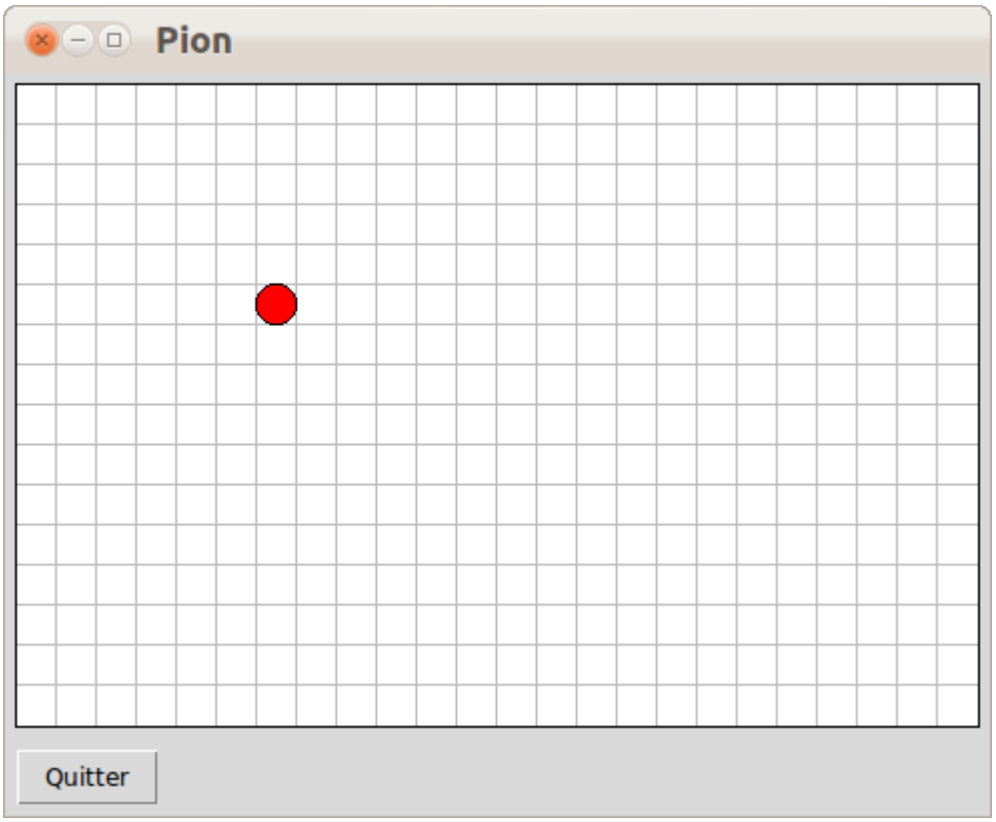
Exercice 7.8 ★★ En s'inspirant du script `animation.py`, faire l'animation d'un mouvement aléatoire brownien.

On utilisera la méthode `create_line()` de la classe `Canvas`.



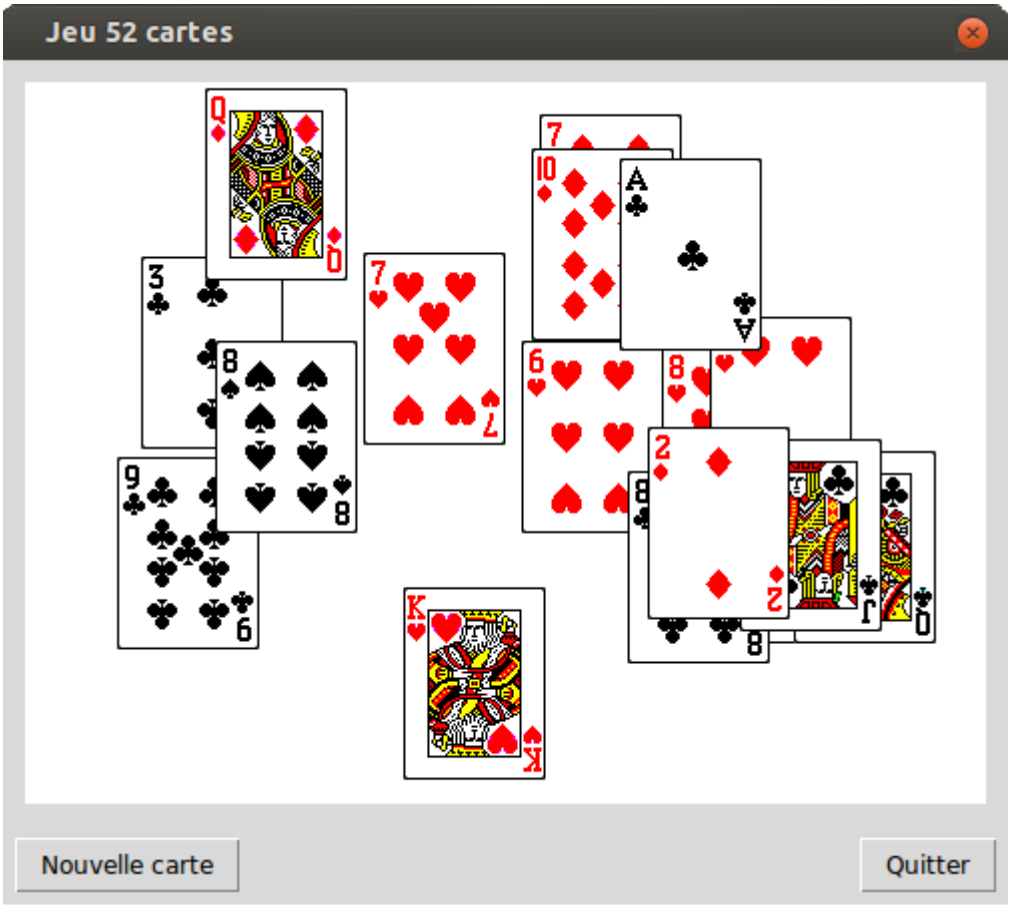
Exercice 7.9 ★ Compléter le script `pion.py` de manière à dessiner une grille.

On utilisera la méthode `create_line()` de la classe `Canvas`.



Exercice 7.10 ★★

Ecrire un script qui tire une carte et l'affiche à une position aléatoire.



Il faut se servir d'une référence comme dans le script `lecture_gif.py`

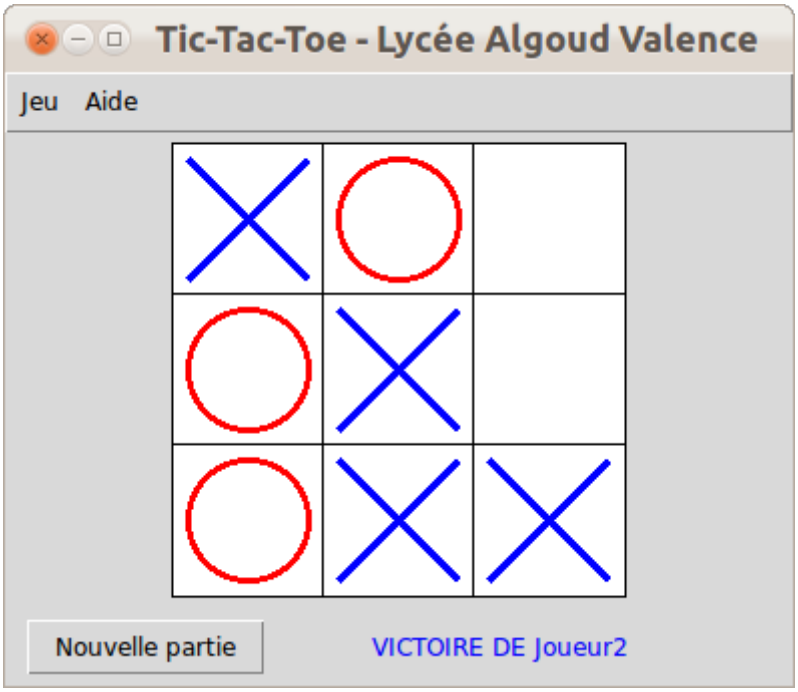
Télécharger les images des 52 cartes et un squelette du script ici.

Quelques idées de projets

Projet n°1 ★★★ Jeu Tic-Tac-Toe (jeu du morpion)

Un projet relativement simple pour un travail en binôme.

Le fichier exécutable est téléchargeable ici.



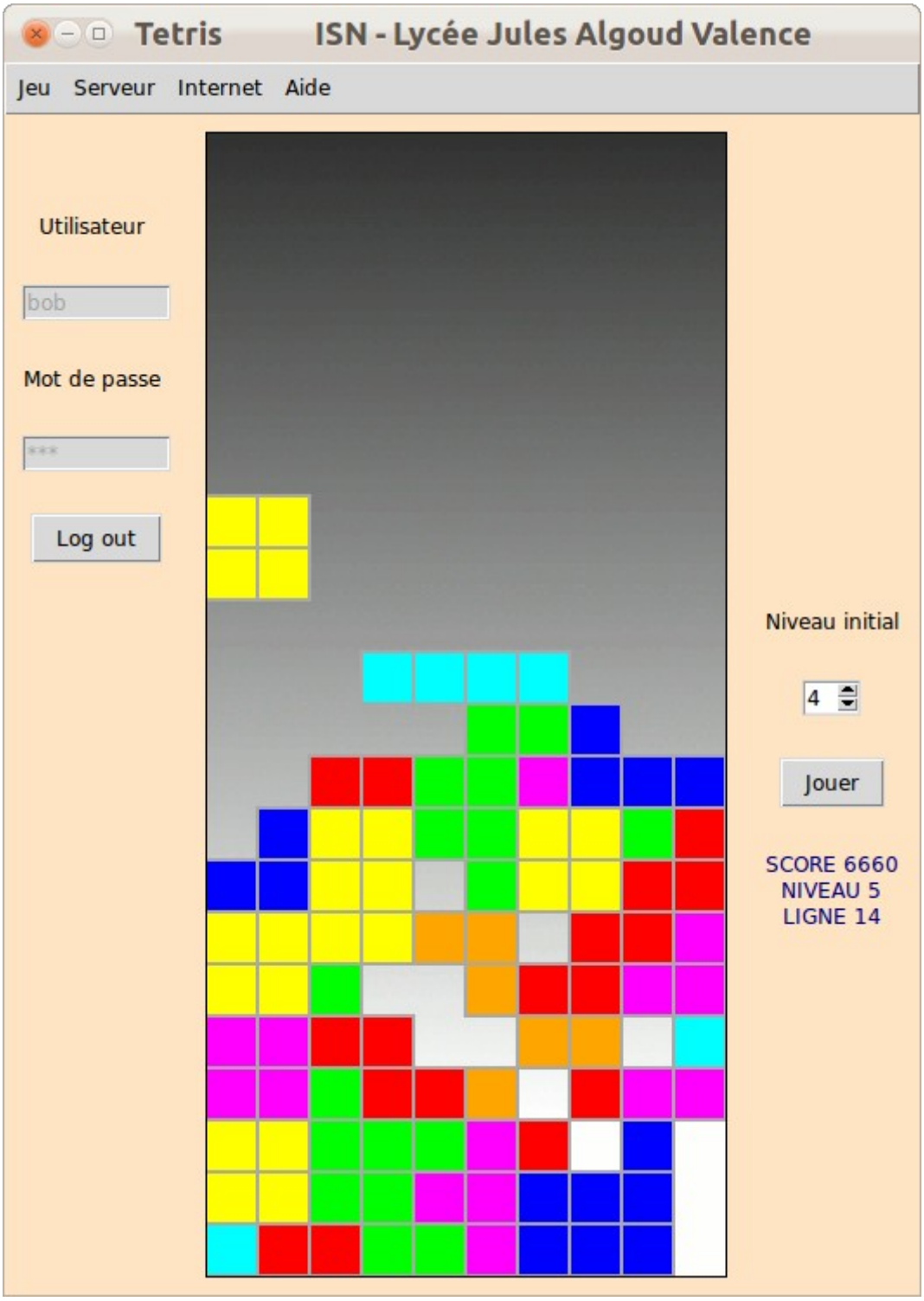
Remarque : dans ma version, le code source Python fait une bonne centaine de lignes.

On pourra compléter ce projet par une version en réseau (plus d'informations ici).

Projet n°2 ★★★★★ Jeu de Tetris avec classement en ligne

Un gros projet à décomposer en plusieurs tâches :

- jeu en local avec Python
- applications Web (en PHP ou CGI-Python, base de données MySQL)
  - nombre d'inscrits
  - inscription en ligne (essayez !)
  - classement en ligne
  - record
  - dernières parties
  - dernière version



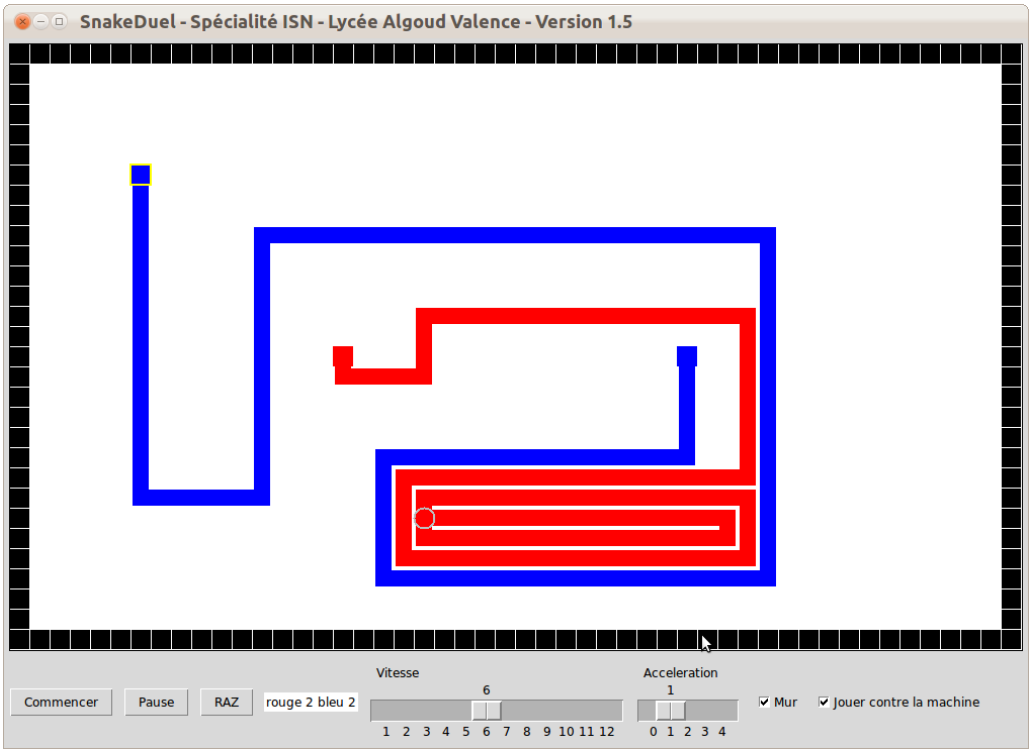
- Télécharger le fichier exécutable

Le fichier exécutable est téléchargeable ici.

Projet n°3 ★★★★ Jeu de SnakeDuel

Un jeu qui se joue à deux, ou seul contre l'ordinateur.

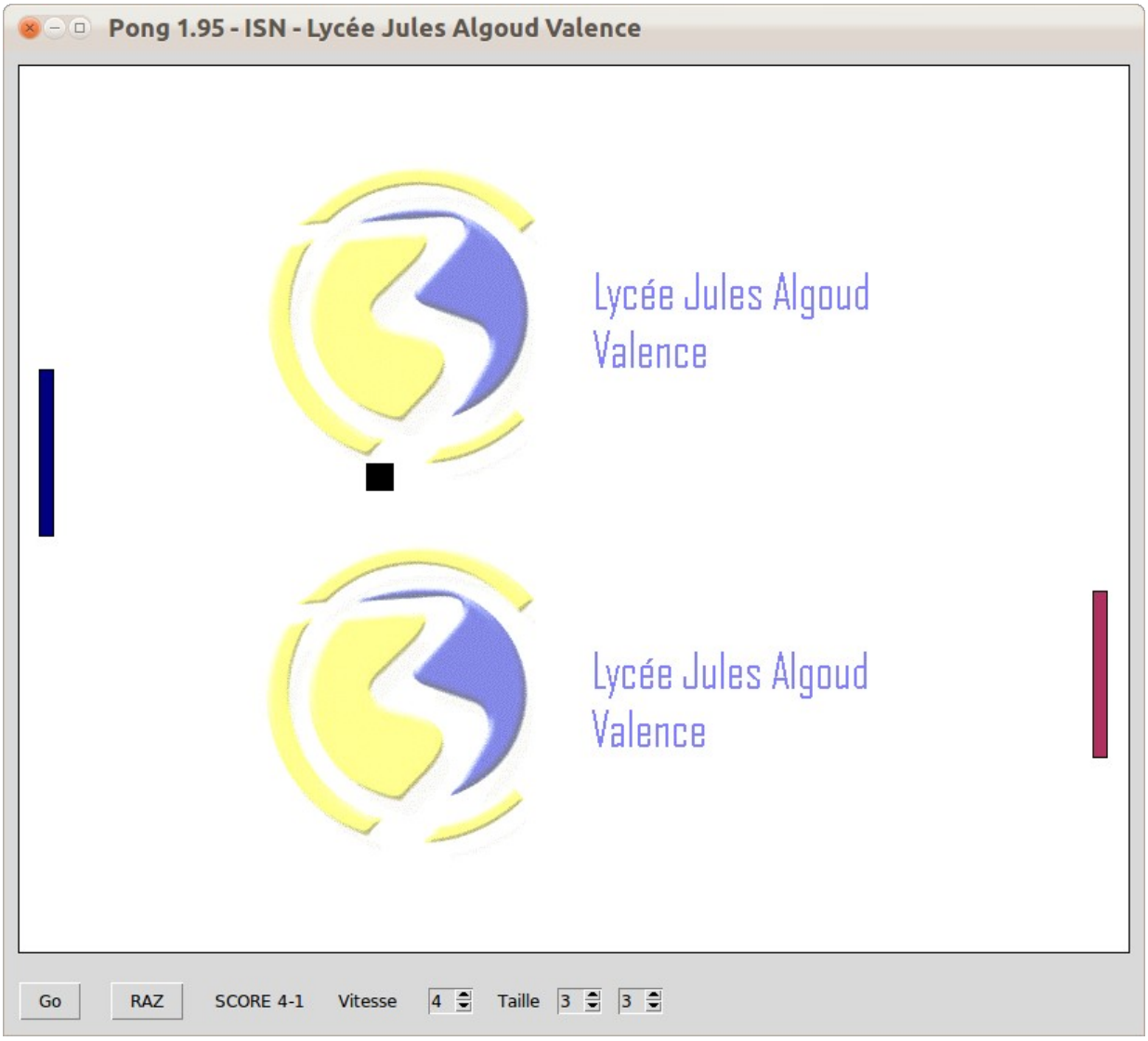
Le fichier exécutable est téléchargeable ici.



Projet n°4 ★★★★ Jeu de Pong

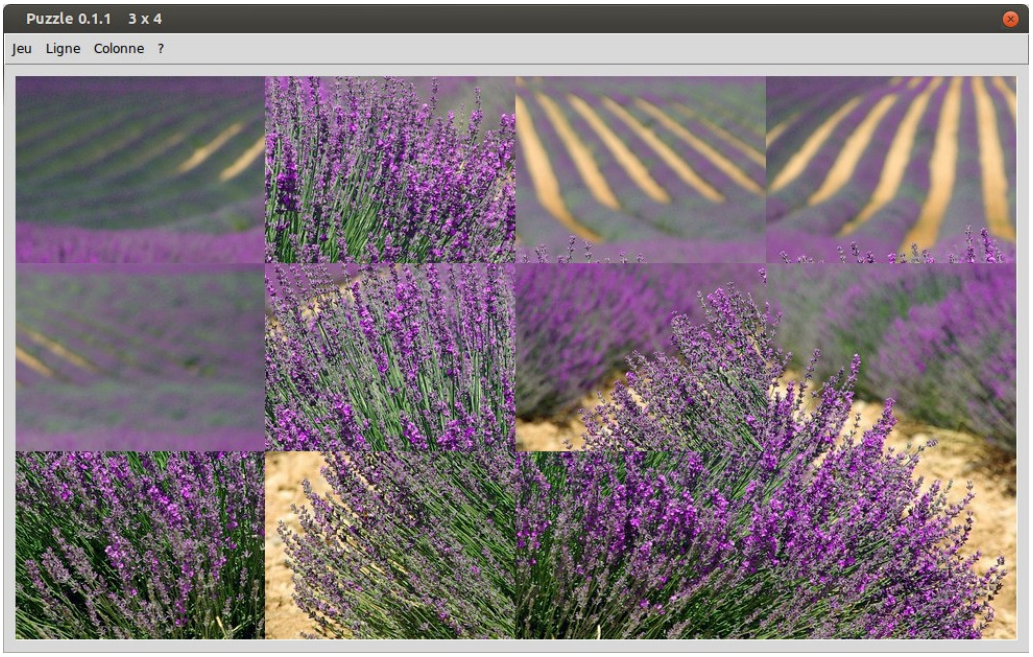
Un jeu qui se joue à deux.

Le fichier exécutable est téléchargeable ici.

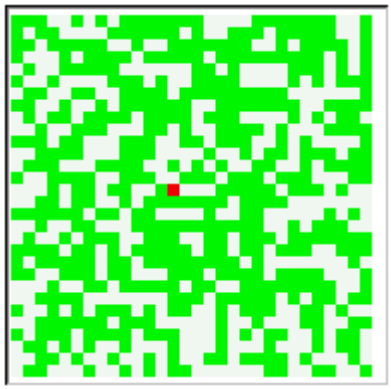


Projet n°5 ★★★★ Puzzle

Le fichier exécutable est téléchargeable ici.



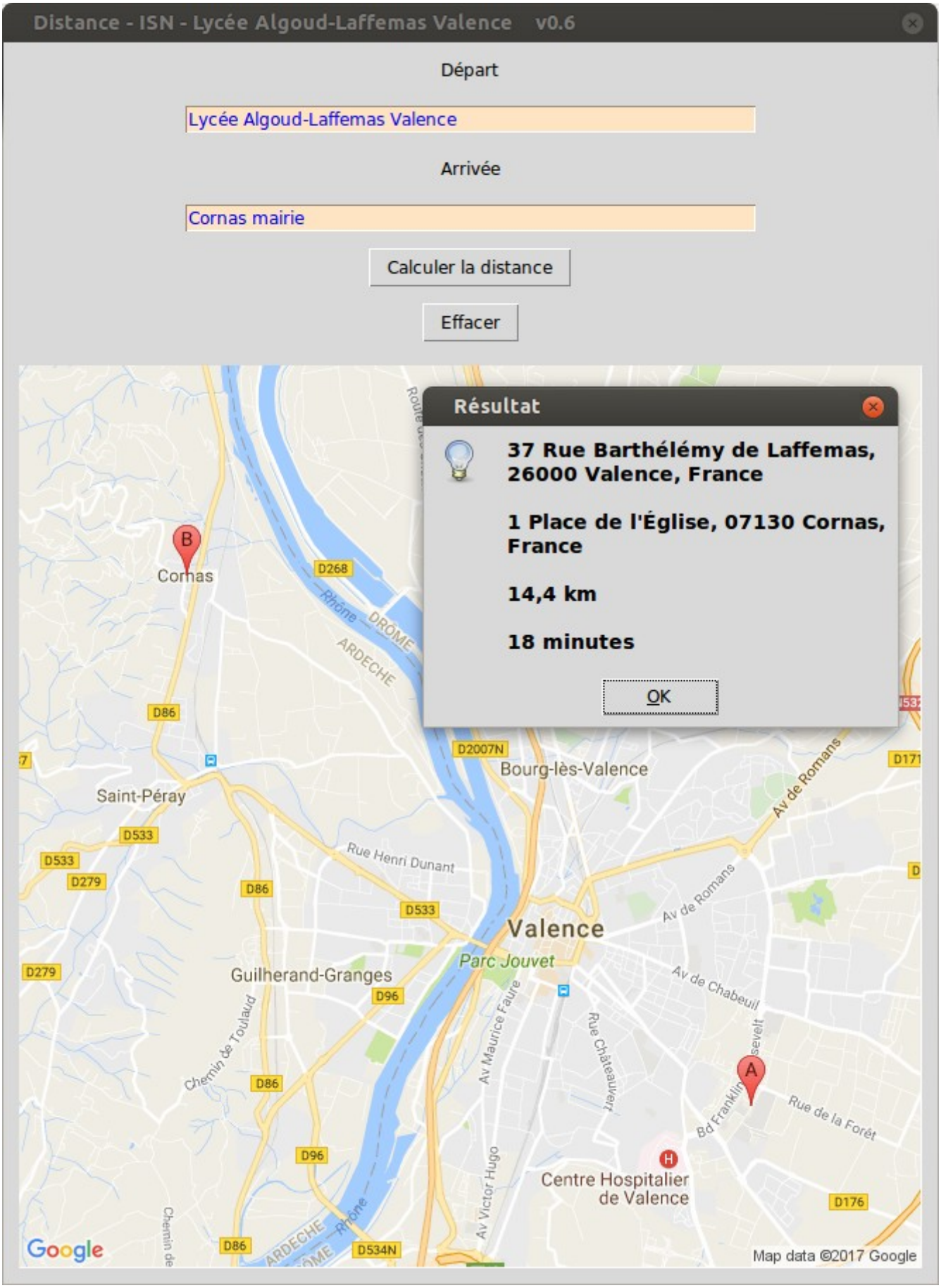
Projet n°6 ★★★ Simulation de feux de forêt



Liens utiles :

- cormas.cirad.fr
- deptinfo-ensip.univ-poitiers.fr

Projet n°7 ★★★ Distance entre deux villes


Distance - ISN - Lycée Algoud-Laffemas Valence v0.6
Départ
Lycée Algoud-Laffemas Valence
Arrivée
Cornas mairie
Calculer la distance
Effacer
Résultat
37 Rue Barthélémy de Laffemas, 26000 Valence, France
1 Place de l'Église, 07130 Cornas, France
14,4 km
18 minutes
OK
Cornas
Saint-Péray
Bourg-lès-Valence
Valence
Parc Jouvet
Guilherand-Granges
Centre Hospitalier de Valence
Rhône
DRÔME
ARDECHE
Av de Romans
Rue Henri Dunant
Av de Chabeuil
Av Maurice Faure
Rue Châteauvert
Av Victor Hugo
Rue de la Forêt
Chemin de Toulaud
D268
D86
D2007N
D533
D279
D96
D534N
D176
Google
Map data ©2017 Google

Vous aurez besoin des modules `urllib` et `json` ainsi que de l'API Google Maps.

Programmes exécutables pour Windows

Pas besoin d'avoir Python sur votre machine !

Les programmes exécutables (extension `.exe`) des exercices et de la plupart des projets sont téléchargeables ici (7 Mo).
Décompresser ensuite l'archive.

Pour jouer à Tetris (par exemple), lancer le programme `tk_Tetris.exe`
Plus d'informations sur les jeux ici.
Have fun !

Chapitre 8 - Lecture et écriture dans un fichier

Un fichier stocke des informations sur un support physique (disque dur, clé USB, CD, DVD, carte mémoire SD...).

Ouvrir un fichier consiste à le charger dans la mémoire vive (RAM) de l'ordinateur (c'est une mémoire volatile : elle s'efface quand on éteint l'ordinateur).
Enregistrer un fichier consiste à l'écrire sur un support physique de stockage (l'information est alors conservée de manière permanente).

Il existe deux types de fichiers :

- Les **fichiers textes** : l'information est stockée sous forme de caractères lisibles par un éditeur de texte (principalement des lettres et des chiffres).
- Les **fichiers binaires** : l'information est stockée en binaire (une suite d'octets dont la valeur est comprise entre 0x00 et 0xFF).

Ecriture dans un fichier

Le mode write

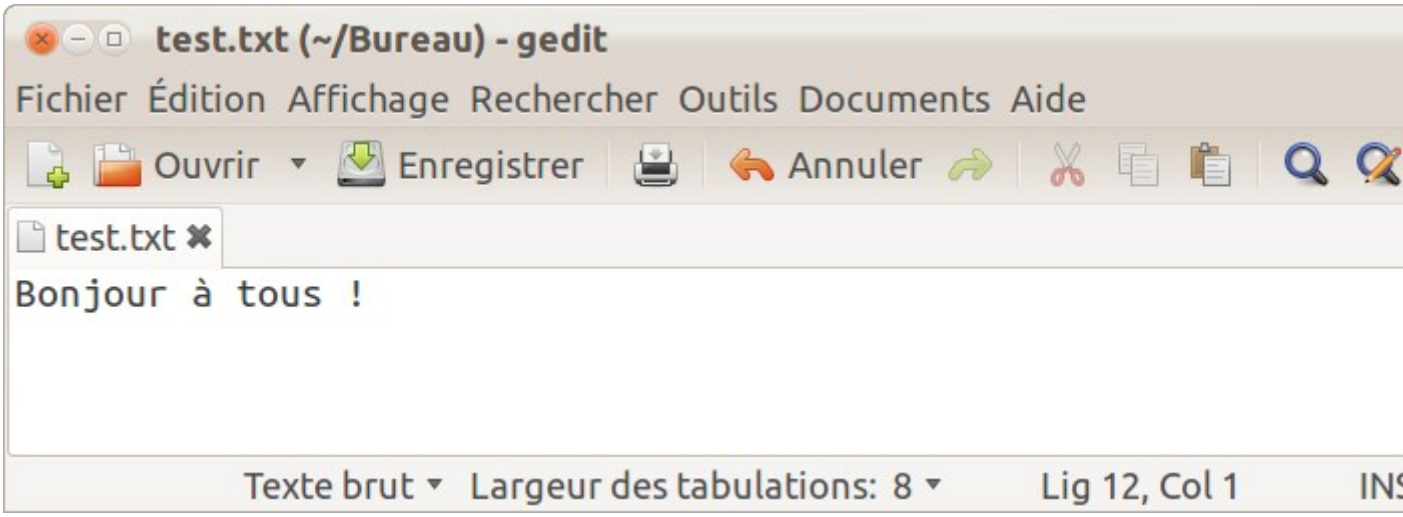
L'écriture dans un fichier se fait avec la fonction `open()` en mode écriture :

```
# script lecture.py


NomFichier = 'test.txt'
# création et ouverture du fichier test.txt en mode write 'w' (écriture)
# si le fichier test.txt existe déjà, il est écrasé
```

```
Fichier = open(NomFichier, 'w')      # instanciation de l'objet Fichier de la classe file

# écriture dans le fichier avec la méthode write()
Fichier.write('Bonjour à tous !')

# fermeture du fichier avec la méthode close()
Fichier.close()
```

Le fichier `test.txt` est créé dans le répertoire courant (sous Windows c'est normalement le répertoire `C:/PythonXX`, mais on peut aussi choisir un emplacement quelconque en spécifiant le chemin absolu, par exemple `NomFichier = 'F:/Mon dossier/test.txt'`).

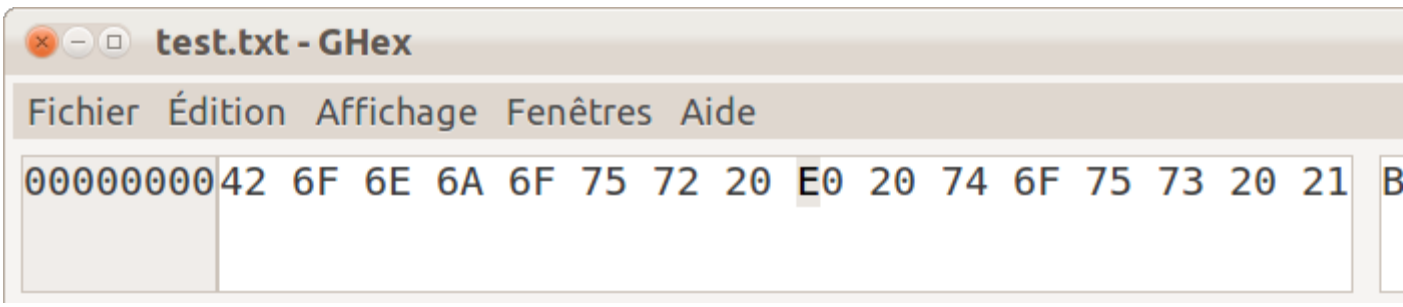
Vous pouvez vérifier son contenu en l'ouvrant avec un éditeur de texte (ou même avec un logiciel de traitement de texte) :



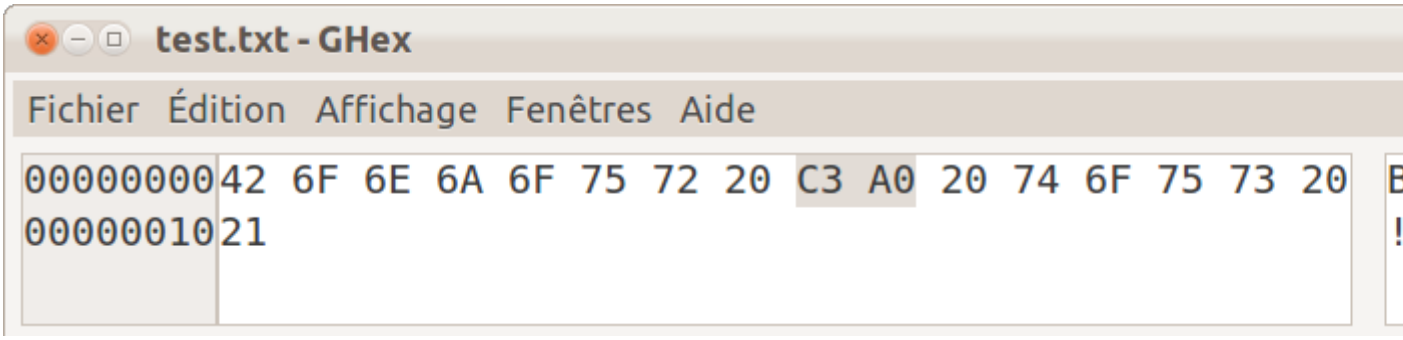
Remarques sur le codage des caractères

Pour Python, le codage par défaut des caractères est cp1252 (Windows-1252) sous Windows, et UTF-8 sous GNU/Linux.

Avec le codage cp1252, le fichier fait 16 octets :



Avec le codage UTF-8, le fichier fait 17 octets !



La différence est due au codage du caractère accentué **à** :
1 octet (0xE0) pour le codage cp1252 et 2 octets (0xC3 0xA0) pour le codage UTF-8.
Les autres caractères ont un codage commun sur un octet (codage ASCII).

Une bonne pratique est de spécifier l'encodage, par exemple :

```
Fichier = open(NomFichier, 'w', encoding='utf8')
```

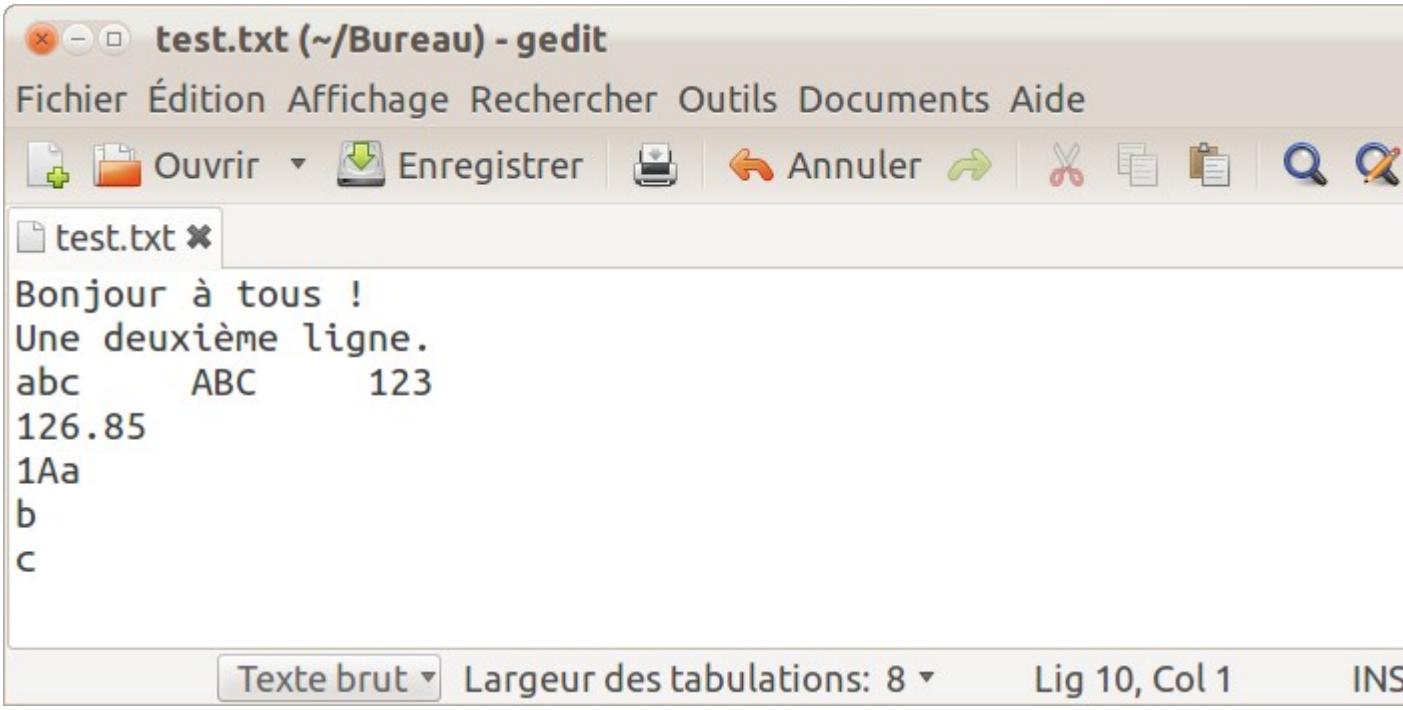
Le mode append

Pour écrire à la fin d'un fichier, on utilise la fonction `open()` en mode ajout.

Repartons du fichier précédent :

```
# ouverture du fichier test.txt en mode append 'a' (ajout)
Fichier = open('test.txt', 'a')      # instanciation de l'objet Fichier
Fichier.write('\nUne deuxième ligne.\n')# '\n' saut de ligne
Fichier.write('abc\tABC\t123\n')     # '\t' tabulation
Fichier.write(str(126.85)+'\n')      # str() transforme un nombre en chaîne de caractères
Fichier.write('\x31\x41\x61\n')      # écriture de '1Aa' en code ASCII
Fichier.write(chr(0x62)+'\n')        # écriture de 'b' en code ASCII
Fichier.write(chr(99))               # écriture de 'c' en code ASCII
Fichier.close()
```

Ce qui donne :



Lecture dans un fichier

Lecture en mode texte

La lecture dans un fichier texte se fait avec la fonction `open()` en mode ... lecture :

```
# ouverture du fichier test.txt en mode read 'r'
(lecture en mode texte)
Fichier = open('test.txt', 'r')      #
instanciation de l'objet Fichier de la classe file
# lecture dans le fichier avec la méthode read()
chaine = Fichier.read()
# affichage du contenu du fichier
print('Contenu du fichier :\n' + chaine)
# fermeture du fichier avec la méthode close()
Fichier.close()
>>>
Contenu du fichier :
Bonjour à tous !
Une deuxième ligne.
```

```
abc   ABC   123
126.85
1Aa
b
c
```

Lecture en mode binaire

En mode texte, un fichier est lu caractère par caractère.
En mode binaire, un fichier est lu octet par octet : c'est un peu différent !

Par exemple, si on veut la taille d'un fichier, il faut utiliser la lecture en mode binaire 'rb' (read binary) :

```
# ouverture du fichier test.txt en mode read binary 'rb' (lecture en mode binaire)
Fichier = open('test.txt', 'rb')
data = Fichier.read()
# affichage de la taille du fichier
print('Taille du fichier :', len(data), 'octets')
# affichage du contenu (en hexadécimal)
import binascii
print(binascii.hexlify(data))
# fermeture du fichier avec la méthode close()
Fichier.close()
>>>
Taille du fichier : 69 octets
b'426f6e6a6f757220e020746f757320210d0a556e652064657
57869e86d65206c69676e652e0d0a616263094142430931323
0d0a3132362e38350d0a3141610d0a620d0a63'
```

La taille du fichier est 69 octets avec le codage cp1252 (sous Windows) et 65 octets avec le codage UTF-8 (sous Linux).

Avec la lecture en mode texte 'r', on obtient 63 octets avec le codage cp1252 (sous Windows) et 65 octets avec le codage UTF-8 (sous Linux) !

Pourquoi ce décalage ?
Car sous Windows, un saut de ligne est codé sur 2 octets (0x0D 0x0A) et le caractère spécial d'une chaîne \n sur un seul octet !

Sous Linux, c'est plus simple avec un saut de ligne codé sur un seul octet (0x0A).

En définitive :
63 + 2 (2 caractères accentués **à** et **è**) = 65
63 + 6 (6 sauts de ligne) = 69

Sérialisation des données avec le module Pickle

Le module `Pickle` est extrêmement pratique pour sauvegarder dans un fichier des structures de données comme les listes (type `list`) ou les dictionnaires (type `dict`).

Pickling

Un exemple de sauvegarde d'un dictionnaire :

```
import pickle


# création d'un dictionnaire
departement = {36: 'Indre', 30: 'Gard', 75: 'Paris'}


# enregistrement du dictionnaire dans un fichier
Fichier = open('data.txt', 'wb')
pickle.dump(departement, Fichier)    # sérialisation
Fichier.close()
```

Unpickling

L'opération inverse est tout aussi simple :

```
import pickle


# récupération du dictionnaire
Fichier = open('data.txt', 'rb')
Dept = pickle.load(Fichier)    # désérialisation
Fichier.close()


print(Dept)
```

```
print(Dept[36])
>>>
{75: 'Paris', 36: 'Indre', 30: 'Gard'}
Indre
```

Exercices

Exercice 8.1 : fichiers binaires et fichiers textes

Les fichiers suivants sont-ils de type binaire ou de type texte ?

- Une photo au format JPEG
- Une image au format PNG
- Une image au format SVG
- Un fichier audio au format MP3
- Un document texte au format TXT
- Une page web au format HTML
- Un fichier exécutable (extension .exe)
- Un document au format PDF
- Un fichier compressé au format ZIP
- Un script en langage Python (extension .py)

Exercice 8.2

1) Créer un fichier texte contenant une suite aléatoire de chiffres.
On utilisera la fonction `randint()` du module `random`.
Exemple

2) Avec un logiciel tel que **7-Zip**, compresser le fichier avec différents algorithmes (zip, gzip, bzip2).
Le taux de compression (en %) est donné par la formule : 100×(taille initiale - taille après compression)/taille initiale
Comment expliquer ce taux de compression modeste (au mieux 57 %)?

Projet

Election des délégués de classe par un vote électronique

Ecrire un ensemble de scripts qui gère une élection par vote électronique. On utilisera des fichiers pour stocker la liste électorale, les mots de passe, la liste des candidats et le résultat des votes.

Pour une sécurité optimale, le vote se fera sur un ordinateur seul (sans connexion réseau, sans connexion à Internet...).

Exemple d'interface :

```
>>>
Election des délégués de classe
```

```
Actuellement, 18 élèves ont voté (sur 30)

Identifiant ? martin
Mot de passe ?

L'authentification a réussi.

Electeur : Martin Rémi

Liste des candidats :

Durand Yohan    --> 0
Barot Pauline   --> 1
Dupont Simon    --> 2

Vote blanc      --> 3

Quel est votre choix ? 1
Confirmer votre choix : 1

A voté !
Merci et bonne journée.
>>>
Résultat de l'élection des délégués de classe

Nombre de votants : 30 / 30

Durand Yohan    -->  5 voix (17.9 %)
Barot Pauline   -->  16 voix (57.1 %)
```

```
Dupont Simon    -->  7 voix (25.0 %)


Vote blanc      -->  2 voix
```

Chapitre 10 - Traitement des images

Les images au format PGM

Le format PGM (portable graymap file format) est utilisé pour des images en niveau de gris.
C'est un format d'image matricielle, sans compression, assez peu utilisé mais qui présente l'intérêt d'être facilement manipulable.

Conversion d'une image JPEG dans le format PGM avec le logiciel GIMP

Commencez par copier sur votre Bureau une image en couleur de format quelconque (JPEG, PNG, BMP, GIF, PPM...).
Par exemple :

Télécharger le fichier photo.jpg

Avec le logiciel de traitement d'images GIMP, nous allons convertir cette image dans le format PGM :
Ouvrir le fichier avec GIMP
Fichier → Enregistrer sous
Sélectionner le type de fichier (selon l'extension) → Image PGM
Exporter
Formatage des données → Brut
Enregistrer

Renommez ce fichier : `image.pgm`

Télécharger le fichier image.pgm

Un exemple de traitement : l'inversion de couleurs

Le script `inversion_image_pgm.py` permet d'inverser les couleurs d'une image.
On obtient ainsi le "négatif" :

```
# script inversion_image_pgm.py
import imghdr, struct
```

```
print("Inversion d'une image PGM en mode binaire à 256 niveaux de gris\n")


NomFichierSource = 'image.pgm'
NomFichierDestination = 'imageInverse.pgm'


print('Fichier source :',NomFichierSource)
print('Fichier destination :',NomFichierDestination)


def Inversion(octet):
    # cette fonction fait une inversion du niveau de gris
    # 0 (noir)    -> 255 (blanc)
    # 255 (blanc) -> 0 (noir)
    return 255-octet


if imghdr.what(NomFichierSource)=='pgm':  # test du format de l'image
    FichierSource = open(NomFichierSource,'rb')
    TailleFichier = len(FichierSource.read())
    print('\nTaille du fichier (en octets) :',TailleFichier)


    Largeur = int(input('Largeur de l\'image (en pixels) ? '))
    Hauteur = int(input('Hauteur de l\'image (en pixels) ? '))
    NbPixel = Largeur*Hauteur


    TailleEntete = TailleFichier - Largeur*Hauteur
```

```
    FichierSource.seek(0)


    # extraction de l'en-tête
    # la variable Entete est une chaîne d'octets
(type bytes)
    Entete = FichierSource.read(TailleEntete)


    # extraction des données
    # Data est une liste de nombre entiers (type
list)
    # la fonction int() retourne le contenu d'un
octet sous forme d'un entier


    Data = [int(i) for i in FichierSource.read()]    #
compréhension de listes


    FichierSource.close()


    if NbPixel == len(Data):
        print('Nombre de pixels :',Largeur*Hauteur)
        print("Nombre d'octets de
données :",len(Data))
        print("Taille de l'en-tête :",TailleEntete)


        FichierDestination =
open(NomFichierDestination,'wb')
        # écriture de l'en-tête du fichier
destination
        FichierDestination.write(Entete)


        # écriture des données du fichier
destination
        for i in Data:
```

```
            # conversion de type : int en bytes
FichierDestination.write(struct.pack('B',Inversion(
i)))

        FichierDestination.close()
        print('Travail terminé !')

    else:
        print('Erreur dans la saisie des
données !')

else:
    print("Ce n'est pas une image PGM !")
>>>
Inversion d'une image PGM en mode binaire à 256
niveaux de gris

Fichier source : image.pgm
Fichier destination : imageInverse.pgm

Taille du fichier (en octets) : 153654
Largeur de l'image (en pixels) ? 480
Hauteur de l'image (en pixels) ? 320
Nombre de pixels : 153600
Nombre d'octets de données : 153600
Taille de l'en-tête : 54
Travail terminé !
```

Le script crée le fichier `imageInverse.pgm` dans le répertoire courant.

Affichons l'en-tête des fichiers image PGM :

```
>>> import binascii
```

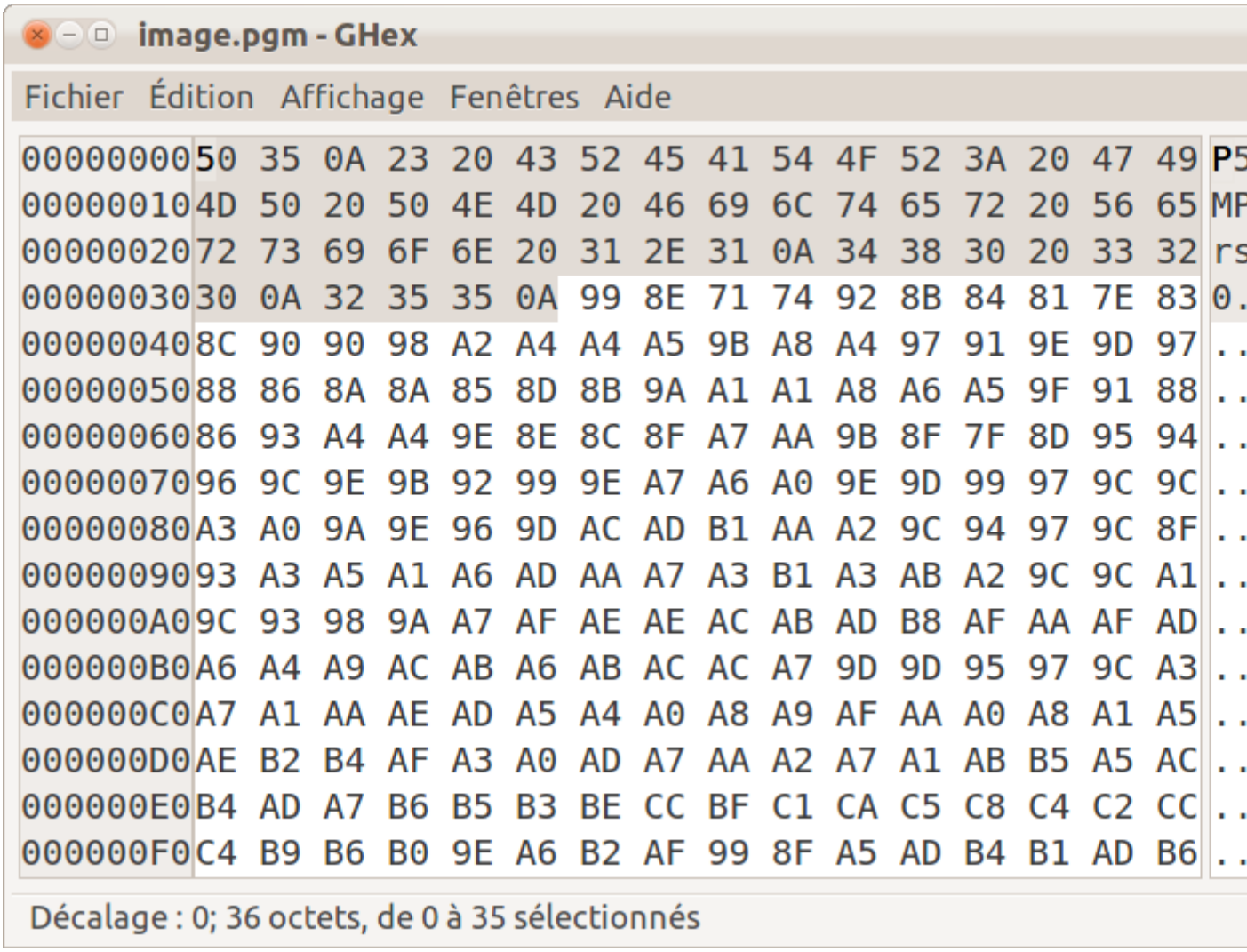
```
>>> print(binascii.hexlify(Entete))  # contenu de l'en-tête en hexadécimal

b'50350a2320435245415544f523a2047494d5020504e4d2046696c746572205665727369f6e20312e310a3438302033323300a3235350a'

>>> print(Entete)          # contenu de l'en-tête en chaîne de caractères

b'P5\n# CREATOR: GIMP PNM Filter Version 1.1\n480 320\n255\n'

>>>
```

L'en-tête contient en particulier la largeur et la hauteur de l'image (480 pixels x 320 pixels) et le nombre 255 (soit 256 niveaux de gris).

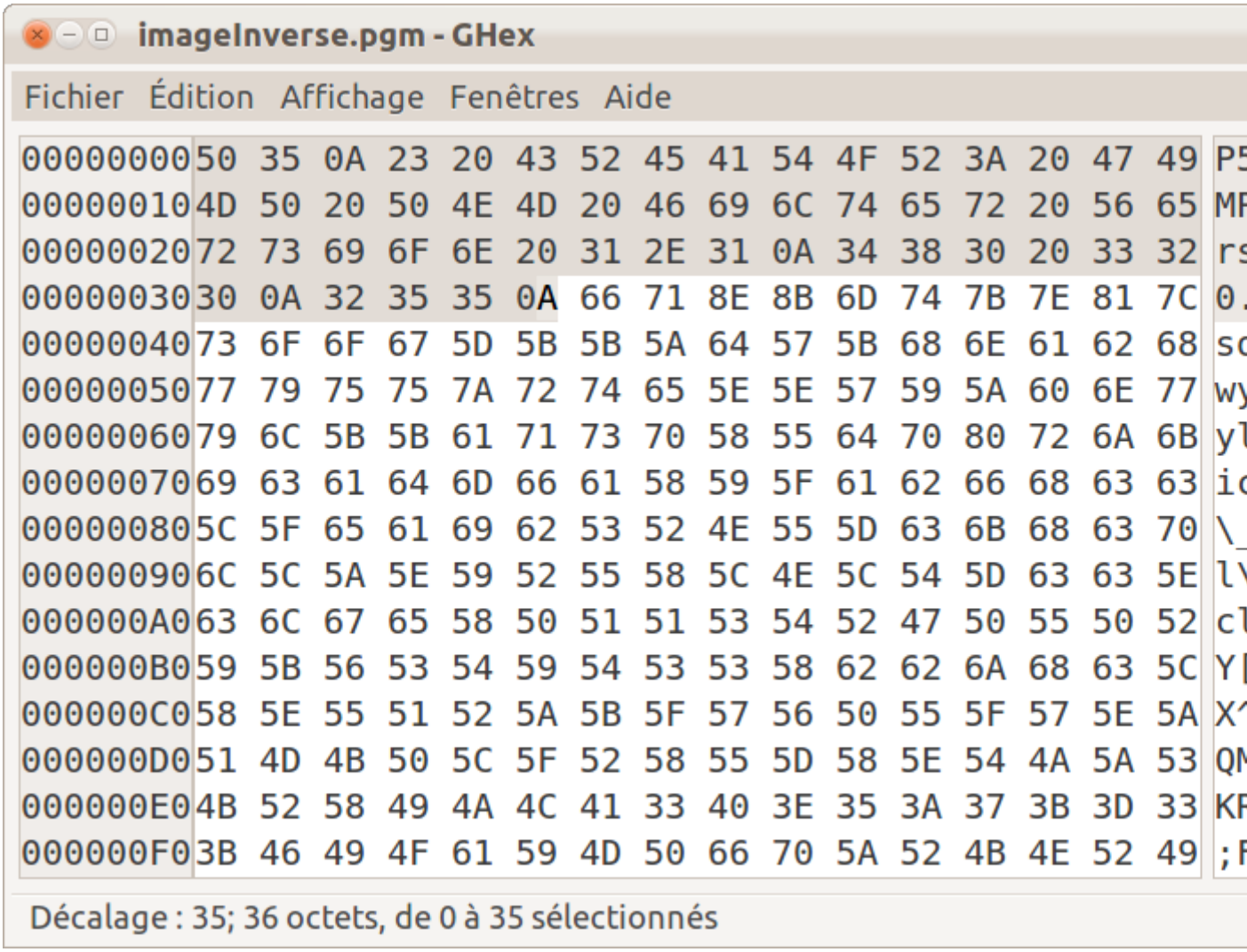
La couleur d'un pixel est codée sur un octet.
On va du noir (0x00) au blanc (0xff) en passant par tous les niveaux de gris.

```
>>> print(Data[0],hex(Data[0]))      # premier octet : pixel en haut à gauche de l'image

153 0x99

>>> print(Data[1],hex(Data[1]))      # deuxième octet : pixel à droite du précédent

142 0x8e

>>> print(hex(Data[479])) # 480ème octet : pixel en haut à droite

0xbf

>>> print(hex(Data[480])) # 481ème octet : pixel de la deuxième ligne à gauche

0x92

>>> print(hex(Data[153599]))    # dernier octet : pixel en bas à droite de l'image (320ème ligne)

0x25
```

Avec un éditeur hexadécimal, observons le contenu du fichier `image.pgm` :



et comparons avec le contenu du fichier `imageInverse.pgm` :



Inversion de couleurs avec le logiciel GIMP

Il suffit de faire :
Couleurs → Inverser

La librairie Pillow (Python Imaging Library)

Cette librairie spécialisée de Python fournit des outils de traitement d'images.
Pour la petite histoire, Pillow est un fork de l'ancienne librairie PIL.

Installation sur tablette ou smartphone sous Android

Dans Google Play, rechercher puis installer l'application QPython - Python for Android.

Installation sous Windows ou GNU/Linux

Pillow n'est pas présente par défaut.
Il faut donc la télécharger et l'installer.

Télécharger la librairie Pillow

Traitement d'images avec la librairie Pillow

Le script suivant permet de convertir une photo en couleur au format JPEG en une image en niveau de gris au format PGM, puis de la transposer (retournement, miroir).
Les résultats sont affichés dans des fenêtres graphiques indépendantes et enregistrés dans des fichiers images :

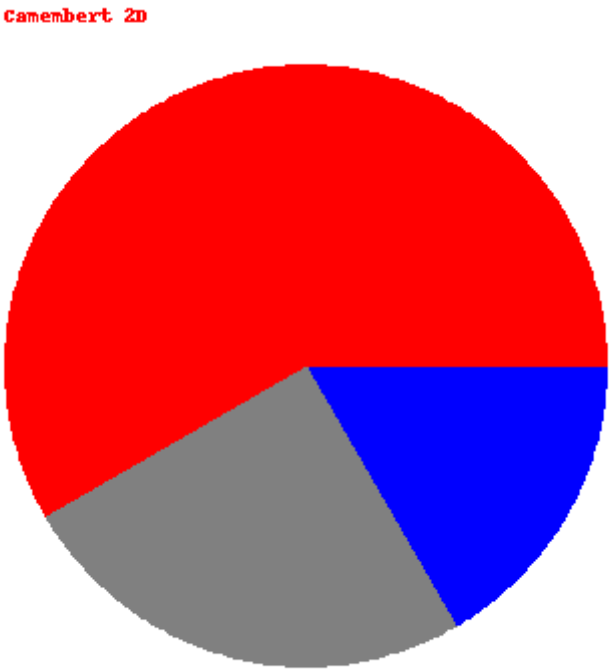
```
# Pillow 6.0.0
# importation du module Image de la librairie Pillow
from PIL import Image


# ouverture de l'image
img = Image.open('photo.jpg')
# affichage de l'image
img.show()
# affichage de la taille de l'image (en pixels)
print(img.size)
# conversion au format PPM (en couleur) et enregistrement de l'image
img.save('photo.ppm','PPM')
img.show()
# conversion en niveau de gris (pour obtenir le format PGM)
img0 = img.convert('L')
# enregistrement dans le fichier image.pgm
img0.save('image.pgm')
img0.show()


# retournement de l'image
img1 = img0.rotate(180)
# affichage et enregistrement de l'image retournée
```

```
img1.show()
img1.save('image_retourne.pgm')

# miroir horizontal
img2 = img0.transpose(Image.FLIP_LEFT_RIGHT)
img2.show()
img2.save('image_miroir_horizontal.pgm')

# miroir vertical
img3 = img0.transpose(Image.FLIP_TOP_BOTTOM)
img3.show()
img3.save('image_miroir_vertical.pgm')
>>>
(480, 320)
```

Dessiner avec la librairie Pillow

Le module ImageDraw de la librairie Pillow permet de créer des formes géométriques simples dans une image : ligne, cercle, rectangle, texte...

Voici un script qui fabrique l'image d'un camembert 2D (au format PNG) :




```
# script camembert.py
# Python 2.7 & Python 3
# Pillow 6.0.0
from PIL import Image,ImageDraw

# création d'une image 400x400 (fond blanc)
img = Image.new("RGB",(400,400),"#FFFFFF")

# création d'un objet Draw
dessin = ImageDraw.Draw(img)

# dessine un arc partiel et le remplit
dessin.pieslice((50,50,350,350),0,60,fill="blue")
dessin.pieslice((50,50,350,350),60,150,fill="gray")
dessin.pieslice((50,50,350,350),150,360,fill="red")

# dessine du texte
dessin.text((50,20),"camembert 2D",fill="red")

img.save("camembert.png","PNG")
img.show()
```

Exercices

Exercice 10.1 ★

1) Reprendre le script `inversion_image_pgm.py` de manière à réduire l'image initiale à deux couleurs (noir et blanc) en utilisant un seuil :

2) Faire la même chose avec le logiciel GIMP.

Exercice 10.2 ★★

1) Reprendre le script `inversion_image_pgm.py` de façon à retourner l'image (rotation à 180 degrés) :

2) Faire la même chose avec le logiciel GIMP.

Exercice 10.3 ★★

1) Reprendre le script `inversion_image_pgm.py` afin de faire une transformation miroir horizontal :

2) Faire la même chose avec le logiciel GIMP.

Exercice 10.4 ★★

1) Reprendre le script `inversion_image_pgm.py` afin de faire une transformation miroir vertical :

2) Faire la même chose avec le logiciel GIMP.

Exercice 10.5 ★ Redimensionnement d'une image JPEG

A l'aide de la méthode resize() du module Image de la librairie Pillow, redimensionner une image JPEG.

Par exemple :

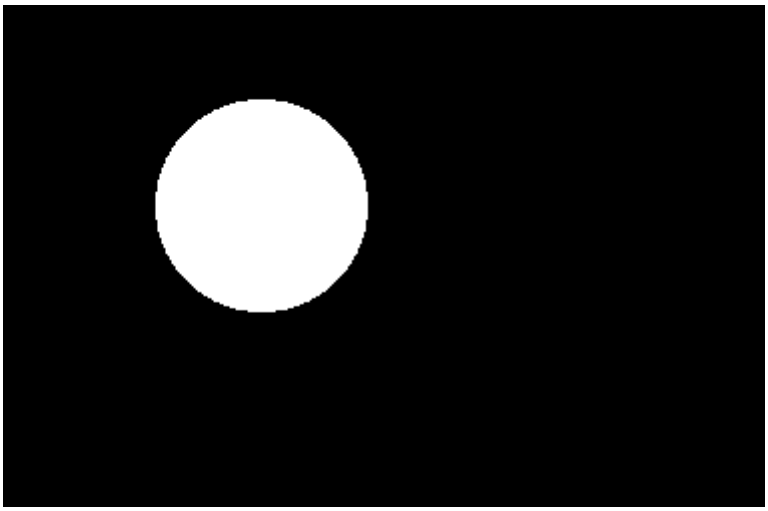
```
>>>
Redimensionnement d'une image JPEG


Nom de l'image originale ? photo.jpg
Taille de l'image originale : (largeur, hauteur) =
(1944, 2592)


Nouvelle taille :
Hauteur en pixels ? 800
Largeur en pixels : 600


Enregistrement de l'image redimensionnée
Nom de l'image redimensionnée ? photo_1.jpg
>>>
```

Exercice 10.6 ★★ Calcul du diamètre d'un disque

1) A l'aide de la méthode getpixel() du module Image de la librairie Pillow, estimer le diamètre d'un disque blanc d'une image à fond noir.

getpixel((x,y)) retourne la couleur du pixel de coordonnées (x,y).

Par exemple :



```
>>>
Nom du fichier ? disque.png

Largeur en pixels :  380
Hauteur en pixels :  250

Nombre de pixels (blanc) :  8820
Position du centre : x = 129.0 y = 100.0
Diamètre en pixels : 105.971565925
>>>
```

Chapitre 11 - Débogage

Le débogueur est un outil utile au débutant car il aide à comprendre le fonctionnement d'un script existant.
Pour un professionnel, il permet le test et la mise au point d'un script en cours de développement (détection et élimination des bugs).

Il existe plusieurs outils de déboguage sous Python, notamment le module standard pdb (en ligne de commande).
Nous ne nous intéresserons qu'au débogueur de l'environnement IDLE (en mode graphique).

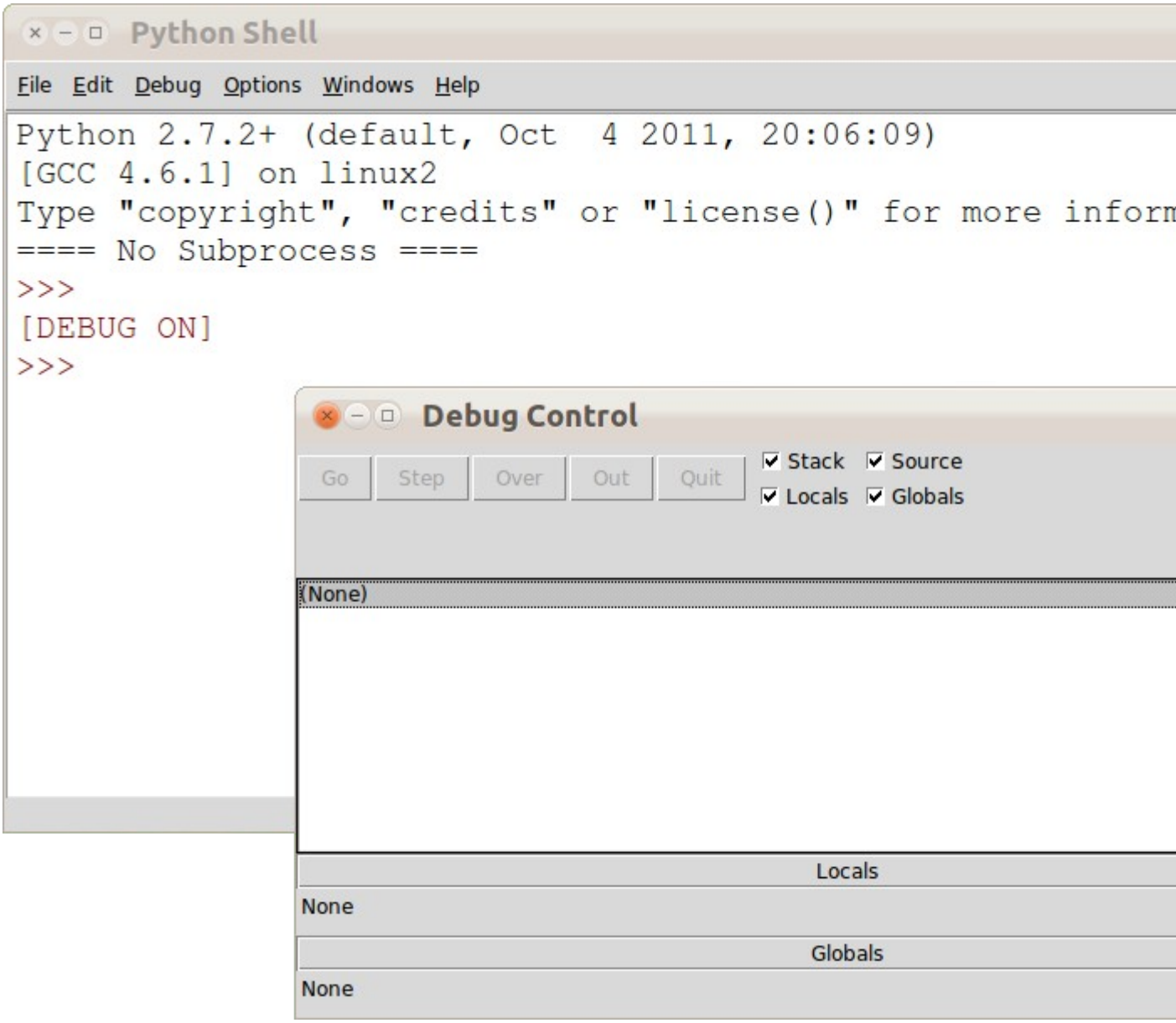
Exemple d'utilisation du débogueur

Commencer par télécharger le script test_debugger3.py

Ce script affiche le carré des nombres entiers de 1 à 5.
Nous allons tester son bon fonctionnement avec le débogueur de l'environnement IDLE.

Ouvrir IDLE :
Démarrer → Programmes → Python → IDLE (Python GUI)

Puis lancer le débogueur :
Debug → Debugger
Cocher les cases Source et Globals :



N.B. Les copies d'écran ont été réalisées avec Python 2.7, et cela reste valable avec Python 3.

Le débogueur possède 5 boutons de commande :

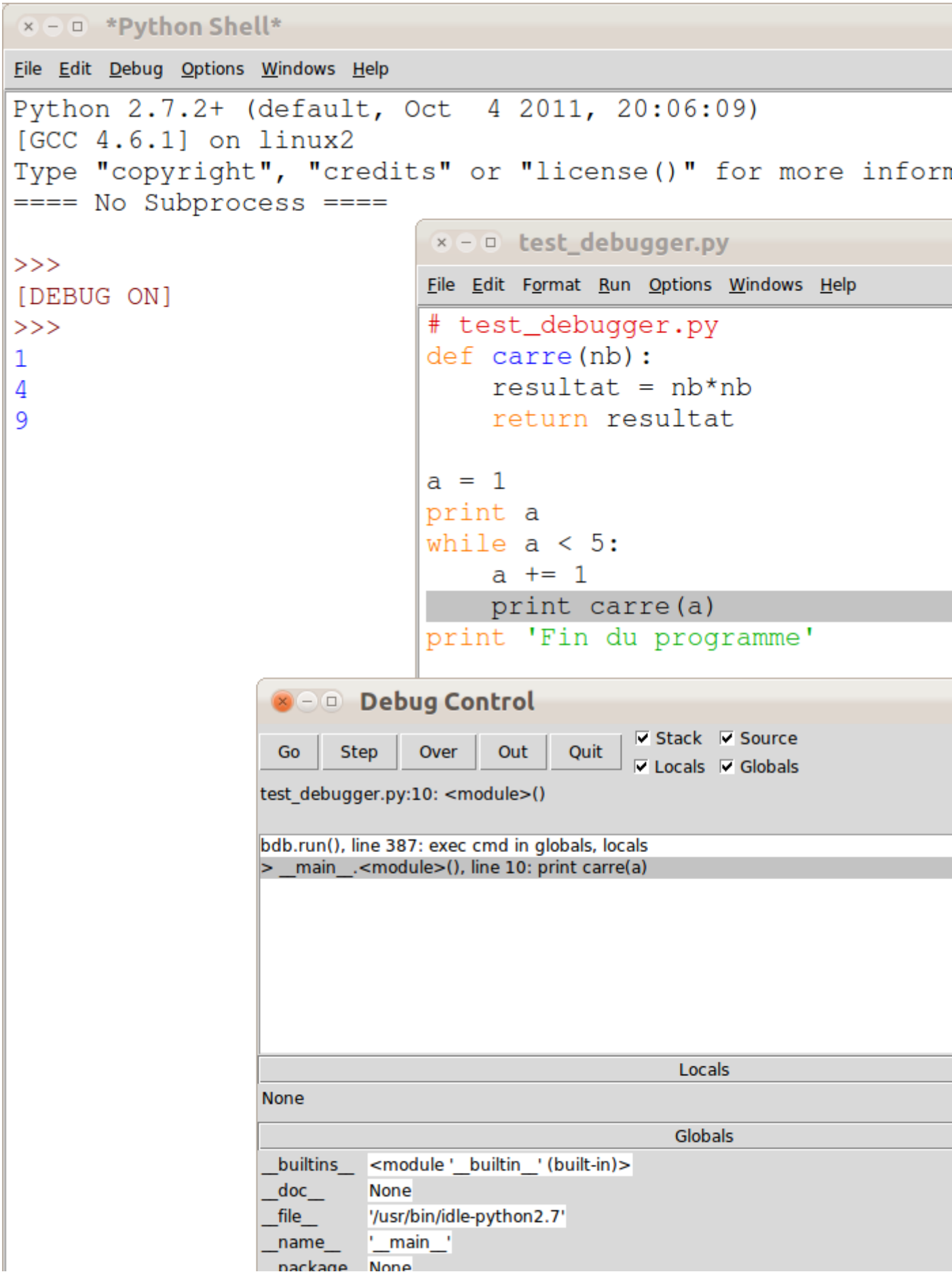
- `Go` : Exécution normale du programme jusqu'au prochain point d'arrêt.
- `Step` : Exécution pas-à-pas (instruction par instruction)
- `Over` : Exécution pas-à-pas du programme principal (le débogueur ne rentre pas dans les fonctions)
- `Out` : Pour sortir de la fonction en cours
- `Quit` : Termine le programme

Dans l'interpréteur intéractif (Python Shell), ouvrir le script `test_debugger3.py` :
File → Open → test_debugger3.py

La fenêtre du code source s'ouvre.
Dans cette fenêtre : Run → Run Module (ou touche)

Pas-à-pas grossier

Pour faire du pas-à-pas grossier, cliquer sur le bouton `Over` du débogueur :


Python Shell
File Edit Debug Options Windows Help
Python 2.7.2+ (default, Oct 4 2011, 20:06:09)
[GCC 4.6.1] on linux2
Type "copyright", "credits" or "license()" for more inform
==== No Subprocess ====
>>>
[DEBUG ON]
>>>
1
4
9
test_debugger.py
File Edit Format Run Options Windows Help
# test_debugger.py
def carre(nb):
resultat = nb*nb
return resultat
a = 1
print a
while a < 5:
a += 1
print carre(a)
print 'Fin du programme'
Debug Control
Go
Step
Over
Out
Quit
Stack
Source
Locals
Globals
test_debugger.py:10: <module>()
bdb.run(), line 387: exec cmd in globals, locals
> __main__.<module>(), line 10: print carre(a)
Locals
None
Globals
__builtins__ <module '__builtin__' (built-in)>
__doc__ None
__file__ '/usr/bin/idle-python2.7'
__name__ '__main__'
__package__ None

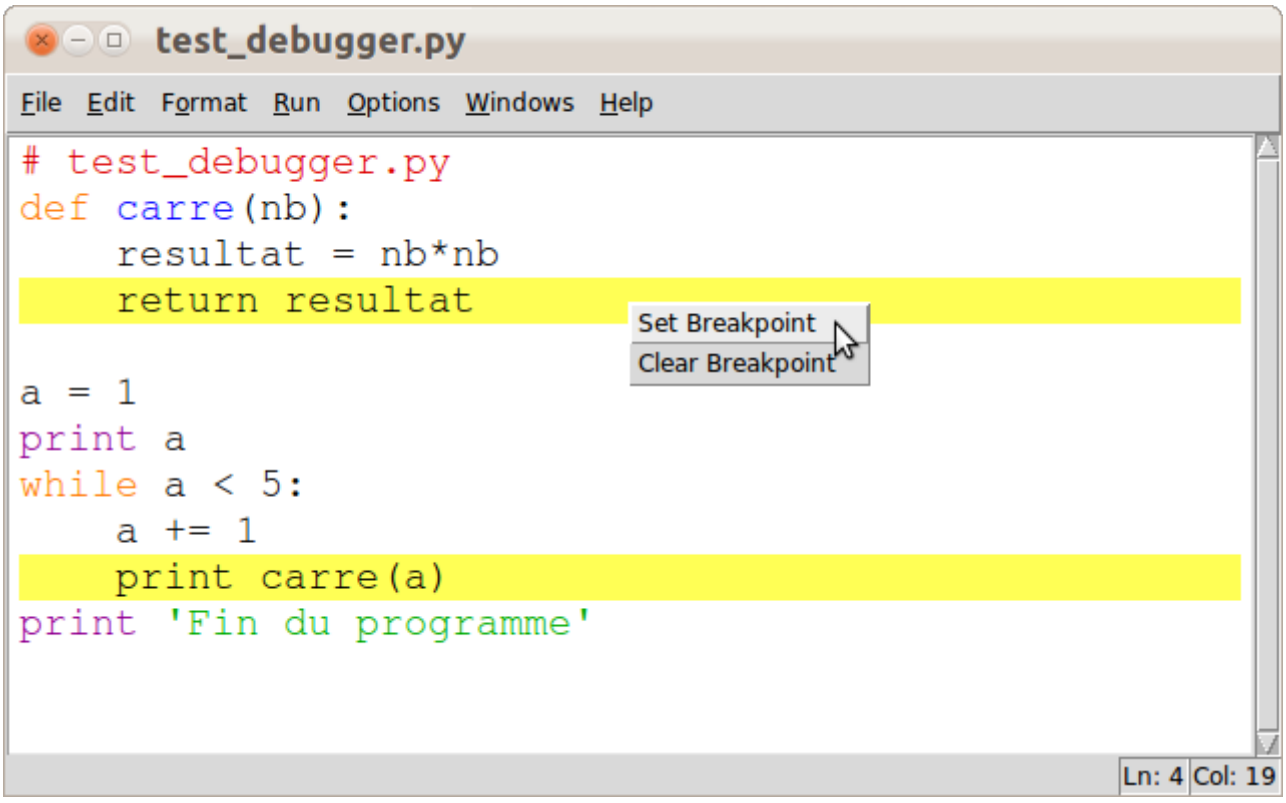
Noter que l'on peut observer le contenu des variables (actuellement `a` vaut 4).

Pas-à-pas détaillé

Pour faire du pas-à-pas détaillé, cliquer sur le bouton `Step` du débogueur. Pour sortir immédiatement d'une fonction, utiliser le bouton `Out` (en particulier pour sortir du script `PyShell.py` qui gère la fonction `print()`).

Point d'arrêt (Breakpoint)

Dans la fenêtre du code source, sur la ligne d'instruction considérée, faire un clic droit et choisir `Set Breakpoint` (la ligne est alors surlignée en jaune) :



Puis utiliser le bouton `Go`.

Exercices

Exercice 11.1 A l'aide du débogueur, étudier la fonction récursive `factorielle()` qui retourne la factorielle d'un nombre entier :

```
def factorielle(x):
    if x < 2:
        return 1
```

```
    else:
        result = x*factorielle(x-1)
        return result

print(factorielle(5))
```

N.B. Une fonction récursive est une fonction qui s'appelle elle-même !

Exercice 11.2 A l'aide du débogueur, étudier la suite de Conway dont le script est disponible ici :

conway3.py

```
>>>
0 1
1 11
2 21
3 1211
4 111221
5 312211
6 13112221
7 1113213211
8 31131211131221
9 13211311123113112211
10 11131221133112132113212221
...
```

réer un programme exécutable avec Py2exe

L'extension **Py2exe** permet de convertir un script Python en un programme exécutable (uniquement pour le système d'exploitation Windows).
Ce programme exécutable (fichier avec l'extension .exe) peut ensuite être lancé sur un ordinateur où Python n'est pas installé.

Téléchargement et installation de Py2exe

Il faut un ordinateur avec Windows et Python déjà installé (attention : Py2exe ne supporte pas la version 3 de Python).

Le programme d'installation de Py2exe est disponible ici :

sourceforge.net/projects/py2exe/files/py2exe/0.6.9/

Avec une machine 32 bits et Python version 2.7, vous devez utiliser le fichier suivant :

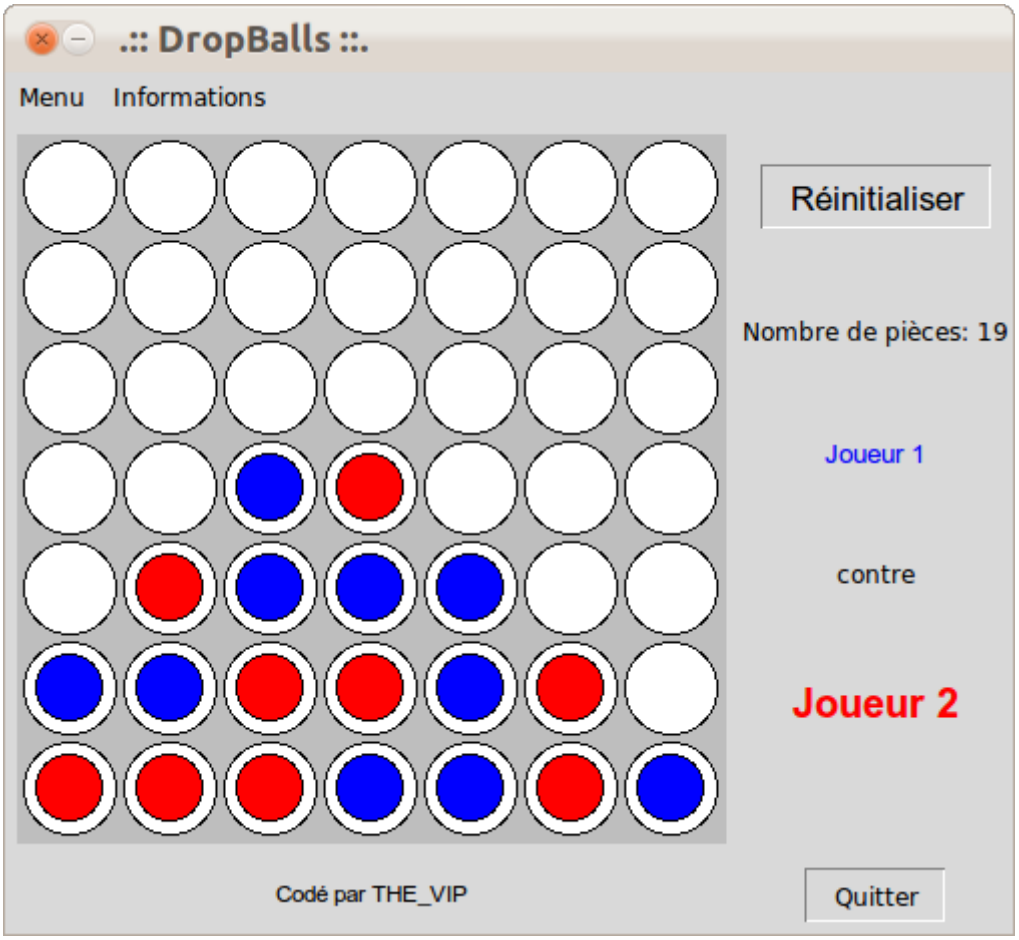
py2exe-0.6.9.win32-py2.7.exe

Une fois téléchargé, il suffit d'exécuter ce fichier pour installer Py2exe.

Utilisation de Py2exe

A titre d'exemple, voici le script d'un jeu de puissance 4 :

jeu_puissance4.py



Télécharger et enregistrer ce script dans le répertoire où est installé Python (chez moi, c'est le répertoire C:\Python27).

Créer le script `setup.py` :

```
# setup.py
from distutils.core import setup
import py2exe
```

```
setup(windows=["jeu_puissance4.py"])
```

Enregistrer le script `setup.py` dans le répertoire courant de Python.

Remarques

- si vous voulez disposer de la console en plus de l'interface graphique (GUI),
  remplacer `setup(windows=["jeu_puissance4.py"])` par `setup(console=["jeu_puissance4.py"])`
- pour traiter plusieurs scripts en même temps, il suffit de compléter la liste :
  `setup(windows=["jeu_puissance4.py","script2.py","script3.py"])`

Ouvrir l'interpréteur de commande de Windows (Démarrer → Exécuter → cmd)

Placez-vous dans le répertoire courant de Python (commande `cd` : change directory) :

```
> cd C:\Python27
```

puis lancer Py2exe :

```
> python setup.py py2exe
```

Chapitre 13 - Jeux vidéo avec le module Pygame

Dans le cadre du projet de la spécialité ISN (bac S), si vous êtes intéressés par la création de jeux vidéo en 2D, il existe le module `Pygame` de Python.
Attention : le choix de ce type de projet demande un investissement initial important, et cela ne doit pas empiéter sur les autres disciplines...

Du point de vue technique, `Pygame` se base sur la bibliothèque SDL (Simple DirectMedia Layer).

Aperçu des possibilités de Pygame

De nombreux jeux sont disponibles et téléchargeables sur le site officiel de Pygame :
www.pygame.org

Installation de Pygame (sous Windows)

Il faut d'abord installer Python version 3.2 :

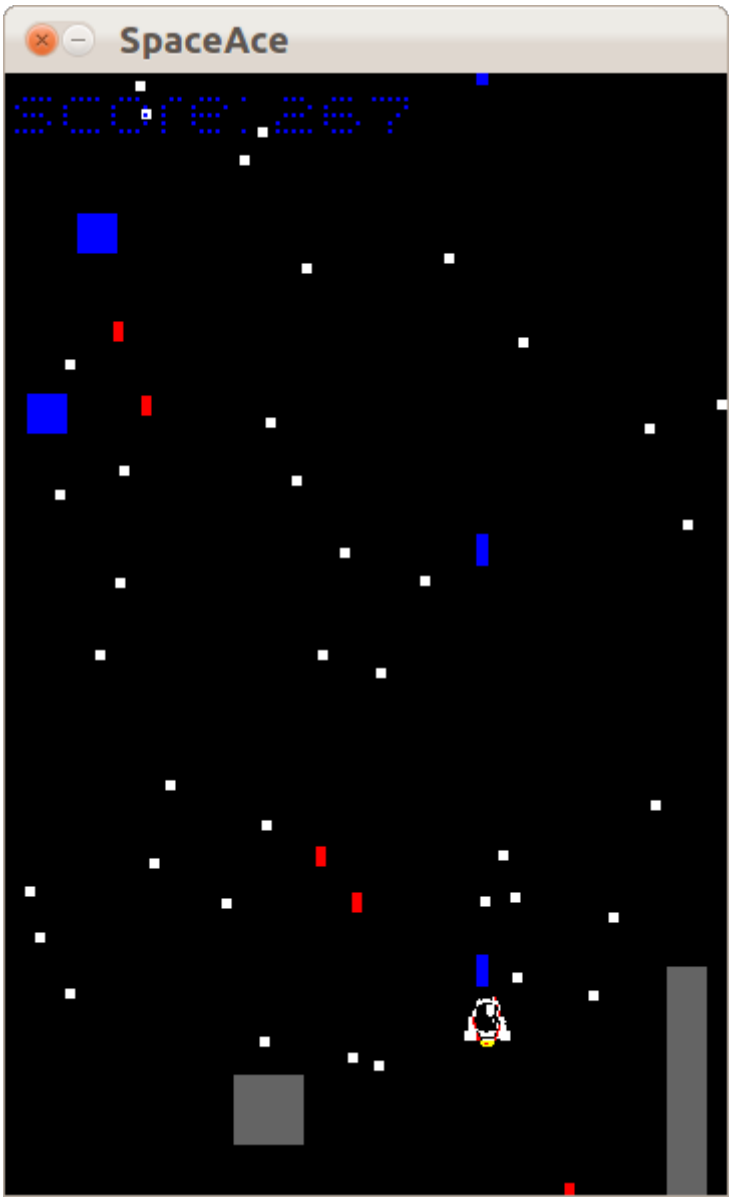
www.python.org/download/releases/3.2
www.python.org/ftp/python/3.2/python-3.2.msi

puis installer la version correspondante de Pygame :

www.pygame.org/download.shtml
pygame.org/ftp/pygame-1.9.2a0.win32-py3.2.msi

Test avec le jeu SpaceAce

Vous pouvez tester le bon fonctionnement en téléchargeant le jeu SpaceAce :



L'archive est ici (à extraire).

Un bug à corriger : il faut renommer le fichier `ship.PNG` en `ship.png`

Pour lancer le jeu, ouvrir et exécuter le script `MyApps.py` avec Python.

Vous noterez que le code source fait environ 300 lignes.

Règle du jeu : www.pygame.org/project-Spaceace-2595-.html

Chapitre 14 - Applications et jeux en réseau

Les applications et jeux en réseau se basent sur le modèle clients/serveur.

Au niveau de la programmation, la tâche est ardue car il faut avoir des connaissances en réseau (module `socket`) et en multithreading (module `threading`) :

- Un `socket` permet de connecter deux machines à travers un réseau.
Ainsi, pour un jeu en réseau avec 10 joueurs (soit 10 clients et 1 serveur), il faut créer 10 sockets (chaque client est connecté au serveur).
Dans le cas du réseau Internet, les sockets se servent du protocole IP (couche réseau) et du protocole TCP pour la couche transport (il existe aussi le protocole UDP qui est plus simple, plus rapide mais non fiable).

- Le `multithreading` est une technique qui permet d'exécuter plusieurs tâches (`thread`) en même temps et de manière indépendante dans un même processus.
Pour un jeu en réseau avec 10 joueurs, le serveur utilisera 10 threads pour communiquer individuellement avec chaque client.
Chaque thread gère donc le socket du client.
On retrouve le multithreading du côté de l'application cliente : pour traiter l'émission et la réception en parallèle, on utilise un thread pour parler au serveur, et un autre pour écouter le serveur.

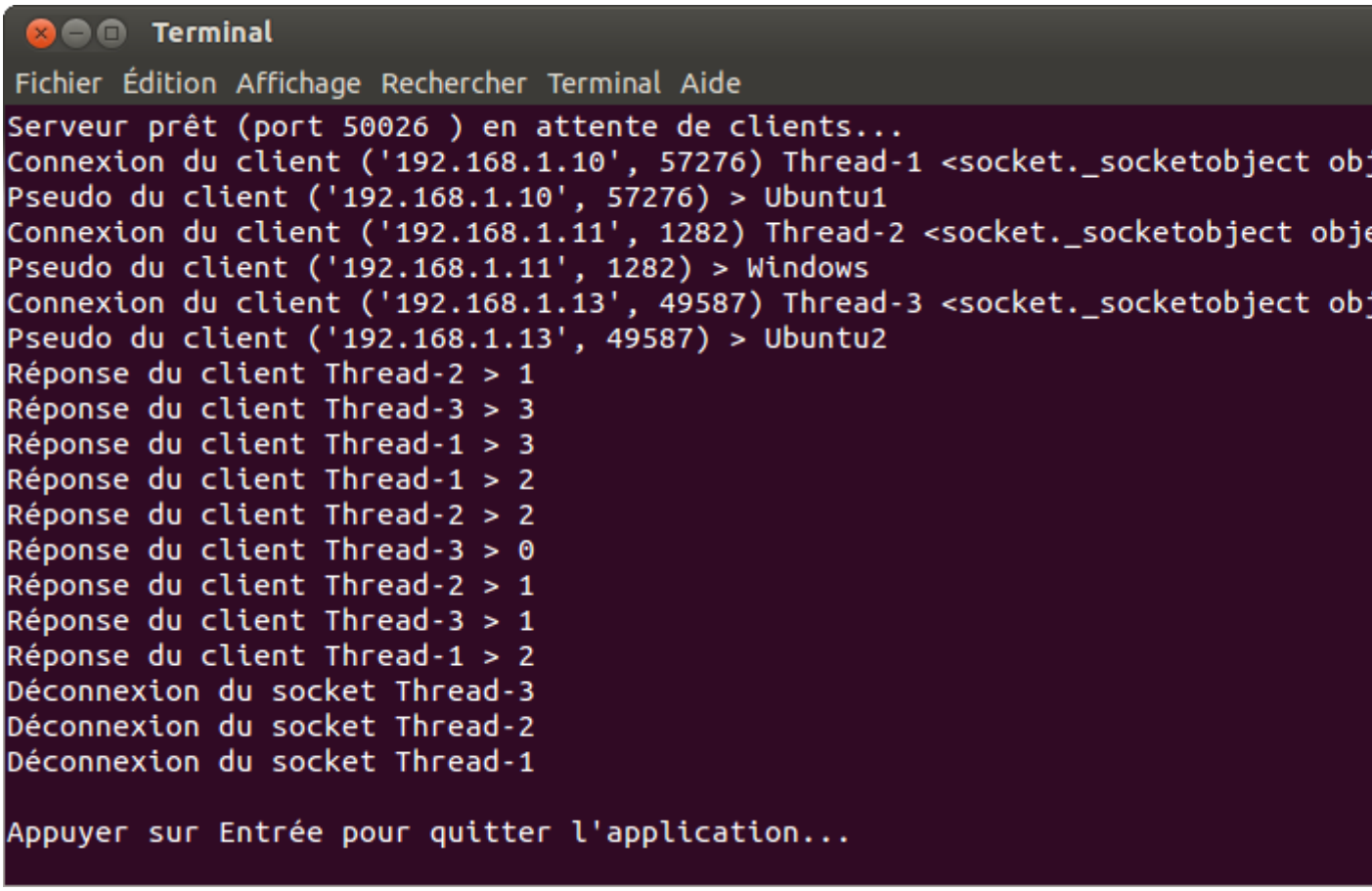
Exemple : un QCM en réseau

Il s'agit d'un jeu multijoueurs.
On peut y jouer en réseau local (LAN) ou sur Internet.

Chaque joueur lance son application cliente pour se connecter au serveur.
La première étape consiste à créer un pseudonyme, puis quand le nombre de joueurs est suffisant (3 dans l'exemple ci-dessous), la partie commence.
Le serveur envoie alors simultanément la première question à l'ensemble des joueurs : le but est d'y répondre correctement et le plus rapidement possible.
Quand tout le monde a répondu ou si la limite de temps est dépassée, on passe à la question suivante...
Un classement est établi à partir d'un système de points.

Le serveur

Avant toute chose, il faut lancer le serveur.
Le port par défaut est 50026 et le nombre de joueurs est 3.
Mais on peut choisir pour le numéro de port une valeur arbitraire comprise entre 49152 et 65535, ainsi qu'un nombre quelconque de joueurs : il suffit de modifier ces paramètres dans le script.

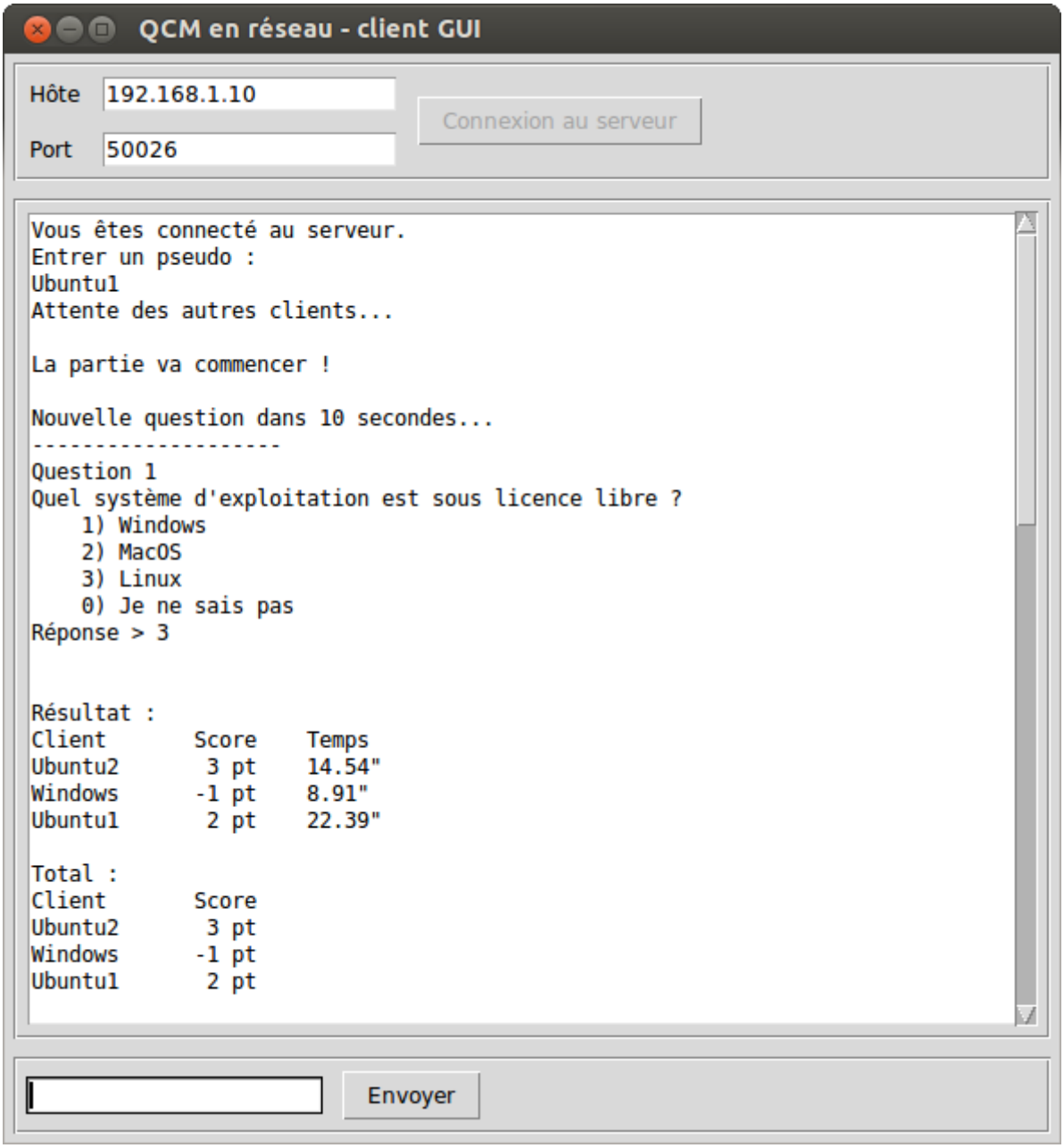


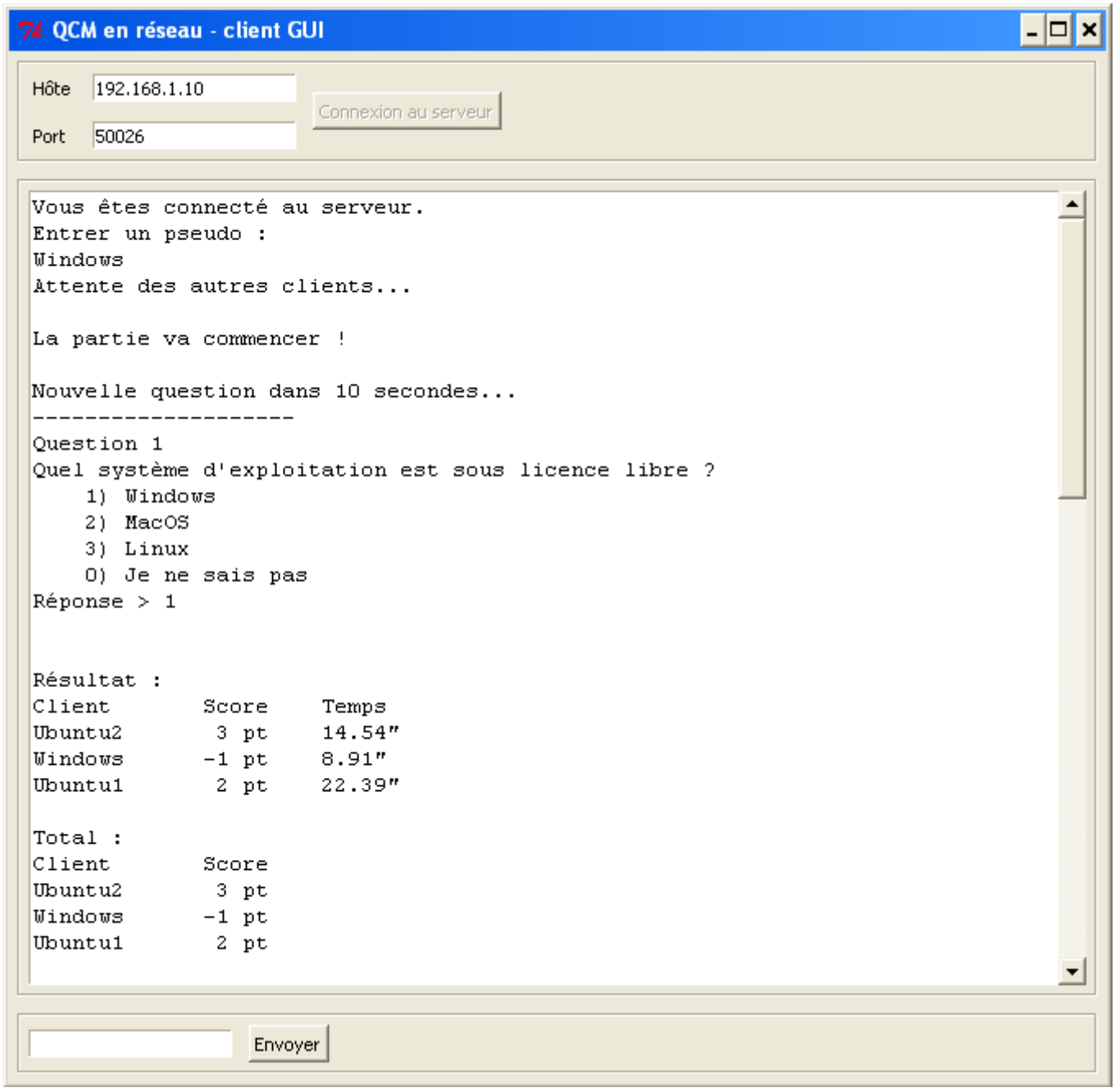
Les clients

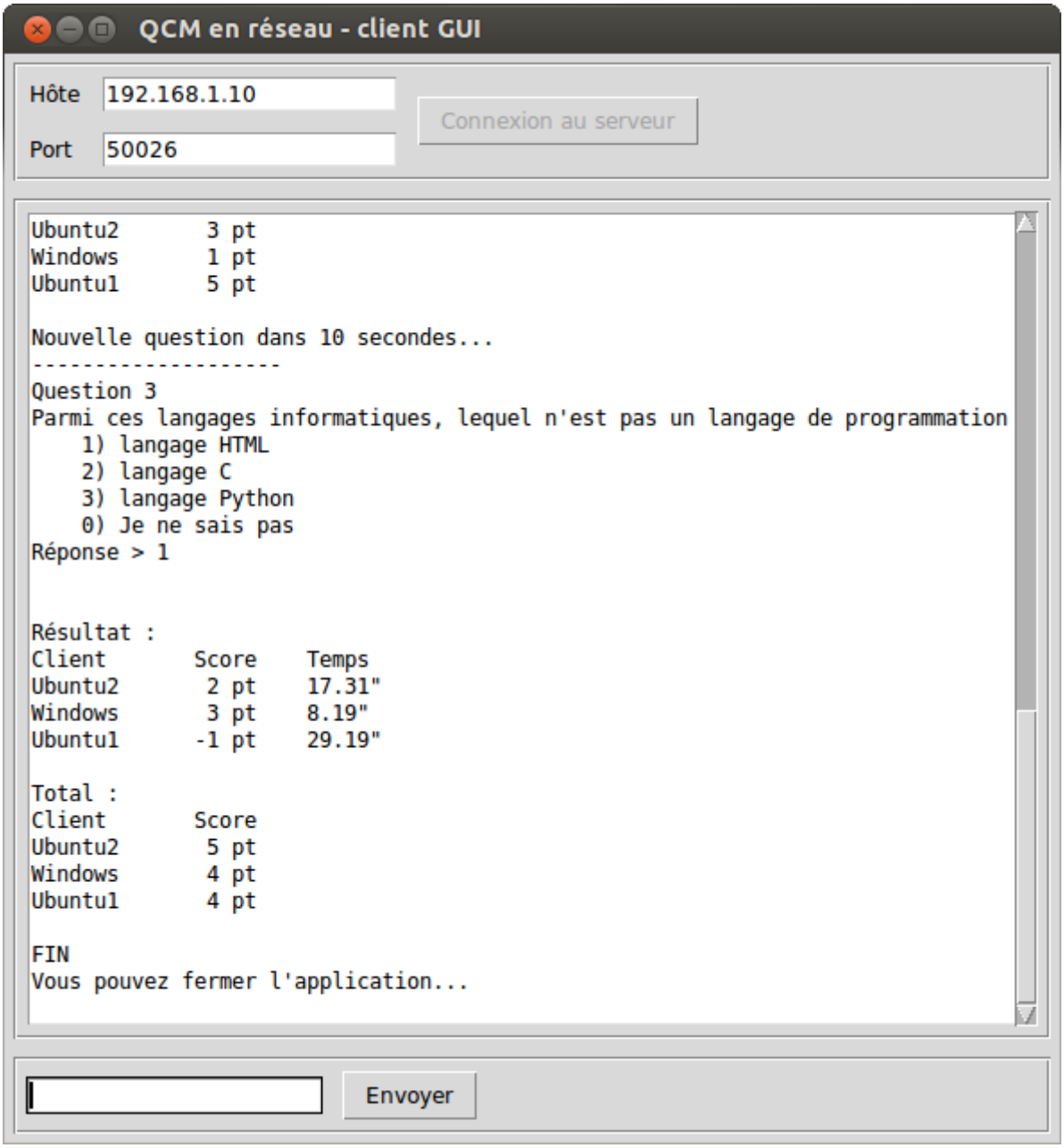
Ici, on joue à trois en réseau local (chez vous, derrière votre box, par exemple).
Pour se connecter au serveur, il faut connaître son adresse IP (192.168.1.10) et le port qu'il utilise (50026).

Python étant multiplateforme, on peut mélanger les systèmes d'exploitation : Windows, Linux, Mac...
Dans cet exemple :

- le serveur et un client tournent sur le même ordinateur sous Linux/Ubuntu (192.168.1.10)
- un deuxième client sur un ordinateur sous Windows (192.168.1.11)
- un troisième client sur un troisième ordinateur sous Linux/Ubuntu (192.168.1.13)


QCM en réseau - client GUI
Hôte
192.168.1.10
Port
50026
Connexion au serveur
Vous êtes connecté au serveur.
Entrer un pseudo :
Ubuntu1
Attente des autres clients...
La partie va commencer !
Nouvelle question dans 10 secondes...
Question 1
Quel système d'exploitation est sous licence libre ?
1) Windows
2) MacOS
3) Linux
0) Je ne sais pas
Réponse > 3
Résultat :
Client Score Temps
Ubuntu2 3 pt 14.54"
Windows -1 pt 8.91"
Ubuntu1 2 pt 22.39"
Total :
Client Score
Ubuntu2 3 pt
Windows -1 pt
Ubuntu1 2 pt
Envoyer


QCM en réseau - client GUI
Hôte
192.168.1.10
Port
50026
Connexion au serveur
Vous êtes connecté au serveur.
Entrer un pseudo :
Windows
Attente des autres clients...
La partie va commencer !
Nouvelle question dans 10 secondes...
Question 1
Quel système d'exploitation est sous licence libre ?
1) Windows
2) MacOS
3) Linux
0) Je ne sais pas
Réponse > 1
Résultat :
Client Score Temps
Ubuntu2 3 pt 14.54"
Windows -1 pt 8.91"
Ubuntu1 2 pt 22.39"
Total :
Client Score
Ubuntu2 3 pt
Windows -1 pt
Ubuntu1 2 pt
Envoyer



Finalement, c'est Ubuntu qui gagne devant Windows ;)

Remarques

Cette application fonctionne mais est loin d'être parfaite : les pertes de connexion ne sont pas gérées, etc...
Si vous avez testé cette application, cela m'intéresse d'avoir un retour de votre part.

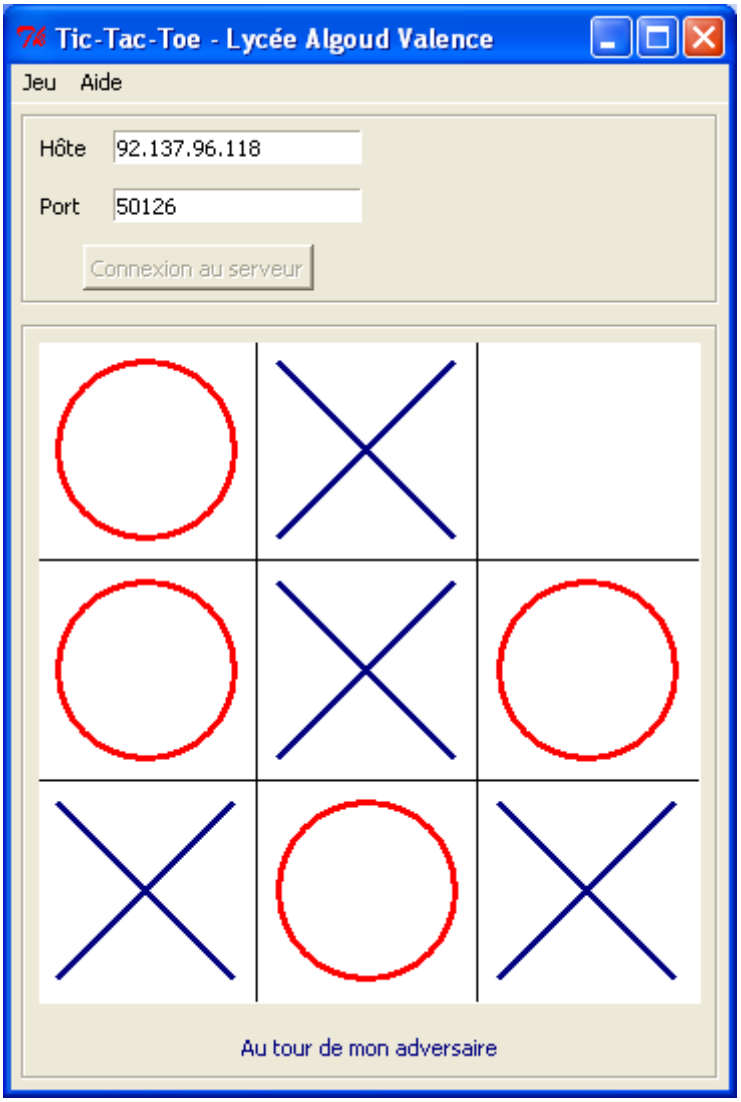
Télécharger les scripts

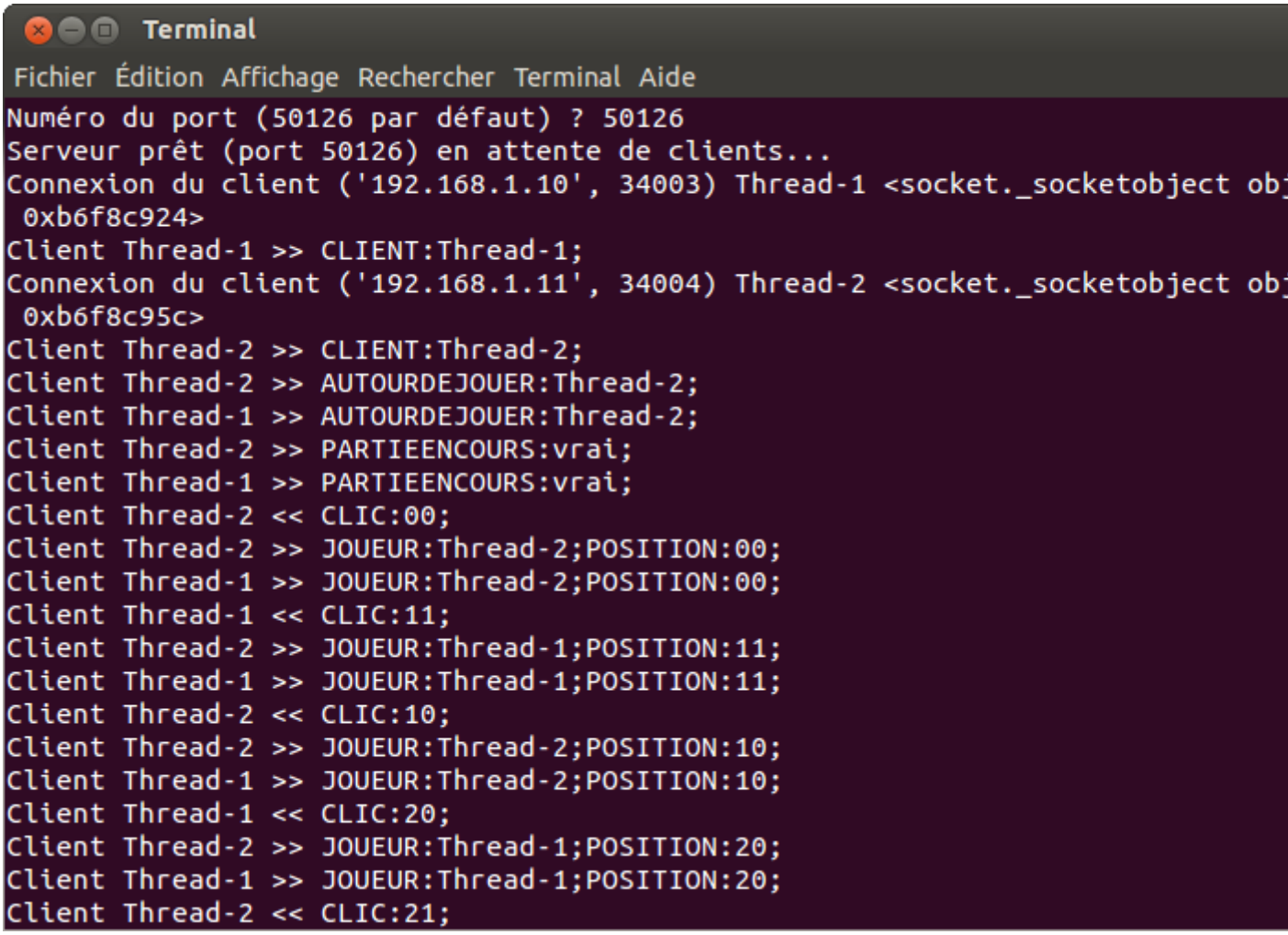
- serveur
- client (mode graphique avec Tkinter)
- client (mode console)

Projet

Jeu Tic-Tac-Toe en réseau (jeu du morpion)

Ecrire l'application serveur et l'application cliente d'un jeu de morpion en réseau local (LAN) ou sur Internet (plus d'informations ici) :





Bibliographie

Le cours de Gérard Swinnen, avec deux exemples concrets à essayer (un chat et un jeu de bombardes) :

- inforef.be/swi/python.htm
- python.developpez.com/cours/TutoSwinnen/?page=page_20#L18

Le livre de Brandon Rhodes et John Goerzen :

- Foundations of Python Network Programming (Apress)

La documentation officielle de Python :

- socket : low-level networking interface
- threading : higher-level threading interface

Un module Python pour l'étude des circuits électriques linéaires en régime continu

Ce module pédagogique s'appuie sur l'écriture des lois de l'électricité pour résoudre un problème donné.
Ce n'est donc pas un logiciel de simulation.

Installation

pypi.org/project/dc-electricity

```
pip install dc-electricity
```

Documentation

Version française
English version

Exemple 0 - Loi d'Ohm

Une résistance de 100 Ω est alimentée par une tension de 5 volts.
Que vaut le courant dans la résistance ?
Quelle est la puissance consommée ?

```
>>> from dcelectricity.dc_fr import *
>>> loi = Loi()
>>> U = Tension(5)
>>> R = Resistance(100)
>>> I = loi.Ohm(v=U, r=R)
>>> I
Intensité du courant : 0.05 A (50.000000 mA)
>>> P = loi.Joule(r=R, i=I)
>>> P
Puissance : 0.25 W (250.000000 mW)
```

On peut aussi utiliser des expressions littérales.
Notez que si l'expresssion est "inattendue", vous aurez droit à un message d'erreur :

```
>>> I = U/R
>>> I
Intensité du courant : 0.05 A (50.000000 mA)
>>> I = R/U
TypeError
```

Pour la puissance :

```
>>> P = R*I*I
>>> P = U*(U/R)
>>> P = U*I
```

Exemple 0b - Résistance équivalente

Que vaut la résistance équivalente à l'association en parallèle de 2 résistances 1 kΩ et 1,5 kΩ ?

```
>>> from dcelectricity.dc_fr import *
>>> loi = Loi()
>>> R1 = Resistance(1, 'k')
>>> R2 = Resistance(1.5, 'k')
>>> Req = loi.Rparallele(R1, R2)
>>> Req
Résistance : 600.000000 Ω
```

Voici une notation équivalente bien pratique :

```
>>> Req = R1//R2
>>> Req
Résistance : 600.000000 Ω
```

Avec les équations littérales :

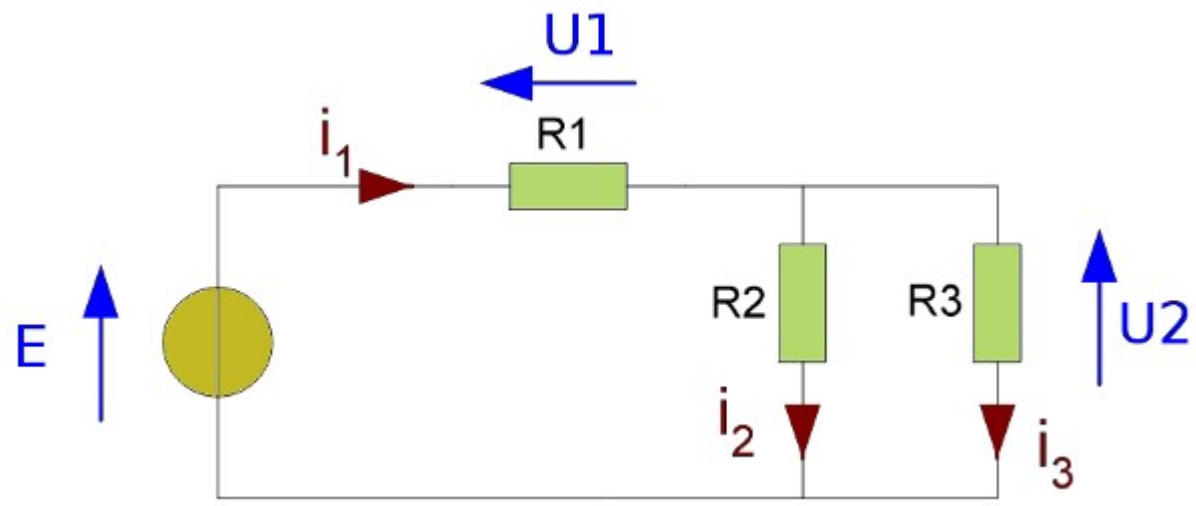
```
>>> Req = 1/(1/R1+1/R2)
>>> Req = R1*(R2/(R1+R2))
```

Enfin, en passant par les conductances :

```
>>> G1 = 1/R1
>>> G2 = 1/R2
>>> Geq = G1 +G2
>>> Geq
Conductance : 0.00166667 S (1.666667 mS)
>>> Req = 1/Geq
>>> Req
Résistance : 600.000000 Ω
```

Exemple 1

On s'intéresse au circuit électrique suivant :



Données :
E = 12 V ; R1 = 1 kΩ ; R2 = 2,7 kΩ et R3 = 1,8 kΩ

Télécharger les scripts exemple1.py et exemple1_bis.py

Ce script calcule les courants I1, I2 et I3, les tensions U1 et U2 et fait un bilan de puissances.

```
>>> from dcelectricity.dc_fr import *
>>> E = Tension(12)
>>> E
Tension : 12.000000 V
>>> R1 = Resistance(1, 'k')
>>> R1
Résistance : 1000 Ω (1.000000 kΩ)
>>> R2 = Resistance(2.7, 'k')
>>> R3 = Resistance(1.8, 'k')
>>> R23 = R2//R3  # résistances en parallèle
>>> R23
Résistance : 1080 Ω (1.080000 kΩ)
>>> Req = R1 +R23  # résistances en série
>>> Req
Résistance : 2080 Ω (2.080000 kΩ)
```

```
>>> I1 = E/Req  # Loi d'Ohm
>>> I1
Intensité du courant : 0.00576923 A (5.769231 mA)
>>> V1 = R1*I1  # Loi d'Ohm
>>> V1
Tension : 5.769231 V
>>> V2 = E -V1  # Loi des branches
>>> V2
Tension : 6.230769 V
>>> I2 = V2/R2  # Loi d'Ohm
>>> I2
Intensité du courant : 0.00230769 A (2.307692 mA)
>>> I3 = I1 -I2  # Loi des nœuds
>>> I3
Intensité du courant : 0.00346154 A (3.461538 mA)
```

Une autre approche :

```
>>> loi = Loi()
>>> # diviseur de tension
>>> V2 = loi.DiviseurTension(vtotal=E, r=R2//R3, r2=R1)
>>> V2
Tension : 6.230769 V
>>> # Théorème de Millman
>>> masse = Tension(0)
>>> V2 = loi.Millman(v_r=[(E, R1), (masse, R2), (masse, R3)])
>>> V2
Tension : 6.230769 V
>>> (E/R1)/(1/R1 +1/R2 +1/R3)
Tension : 6.230769 V
```

```
>>> V2/E
0.5192307692307693
>>> # diviseur de courant
>>> I3 = loi.DiviseurCourant(itotal=I1, r=R3,
r2=R2)
>>> I3
Intensité du courant : 0.00346154 A (3.461538 mA)
>>> I1*R2/(R2 +R3)
Intensité du courant : 0.00346154 A (3.461538 mA)
```

Puissances et loi de Joule

```
>>> P = E*I1  # puissance fournie par la source E
>>> P
Puissance : 0.0692308 W (69.230769 mW)
>>> P1 = loi.Joule(r=R1, i=I1)
>>> P1
Puissance : 0.033284 W (33.284024 mW)
>>> R1*I1*I1
Puissance : 0.033284 W (33.284024 mW)
>>> loi.Joule(r=R1, v=V1)
Puissance : 0.033284 W (33.284024 mW)
>>> V1*(V1/R1)
Puissance : 0.033284 W (33.284024 mW)
>>> P2 = loi.Joule(r=R2, i=I2)
>>> P2
Puissance : 0.0143787 W (14.378698 mW)
>>> P3 = loi.Joule(r=R3, i=I3)
Puissance : 0.021568 W (21.568047 mW)
>>> P1 +P2 +P3
Puissance : 0.0692308 W (69.230769 mW)
```

Exemple 2a

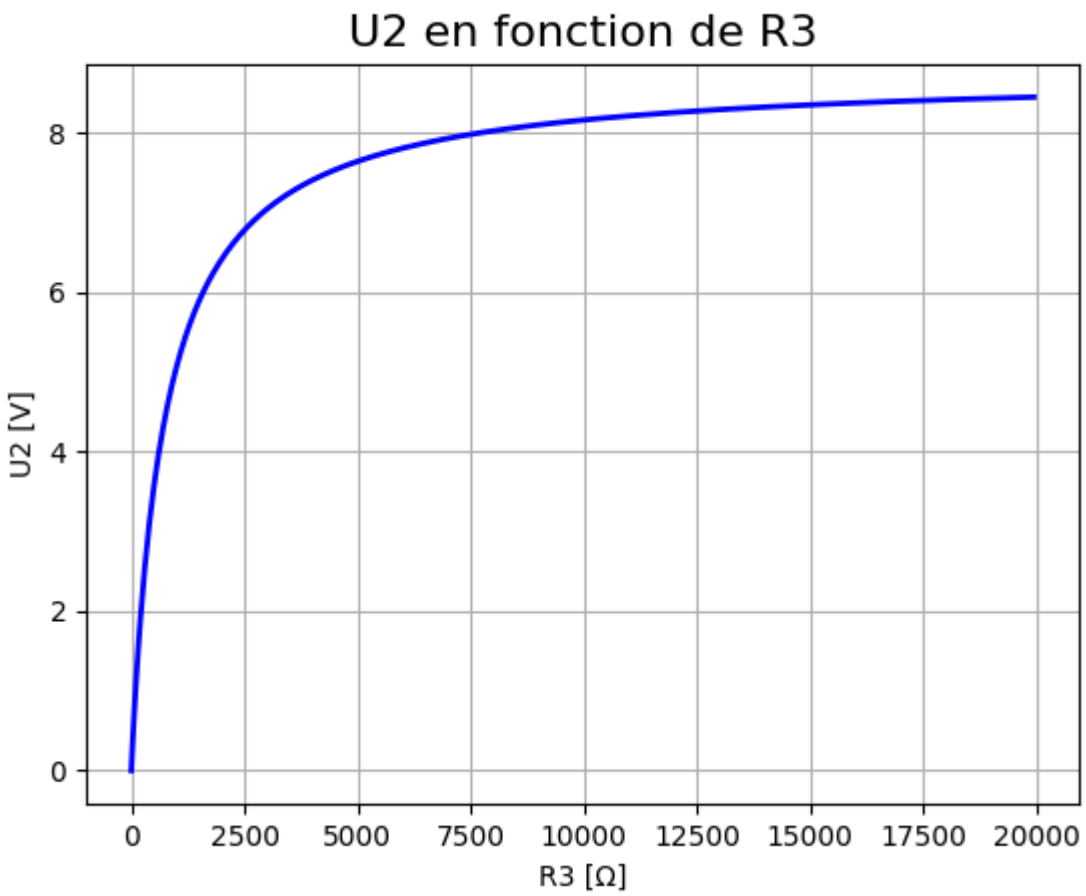
Le même circuit électrique mais avec une résistance R3 ajustable :

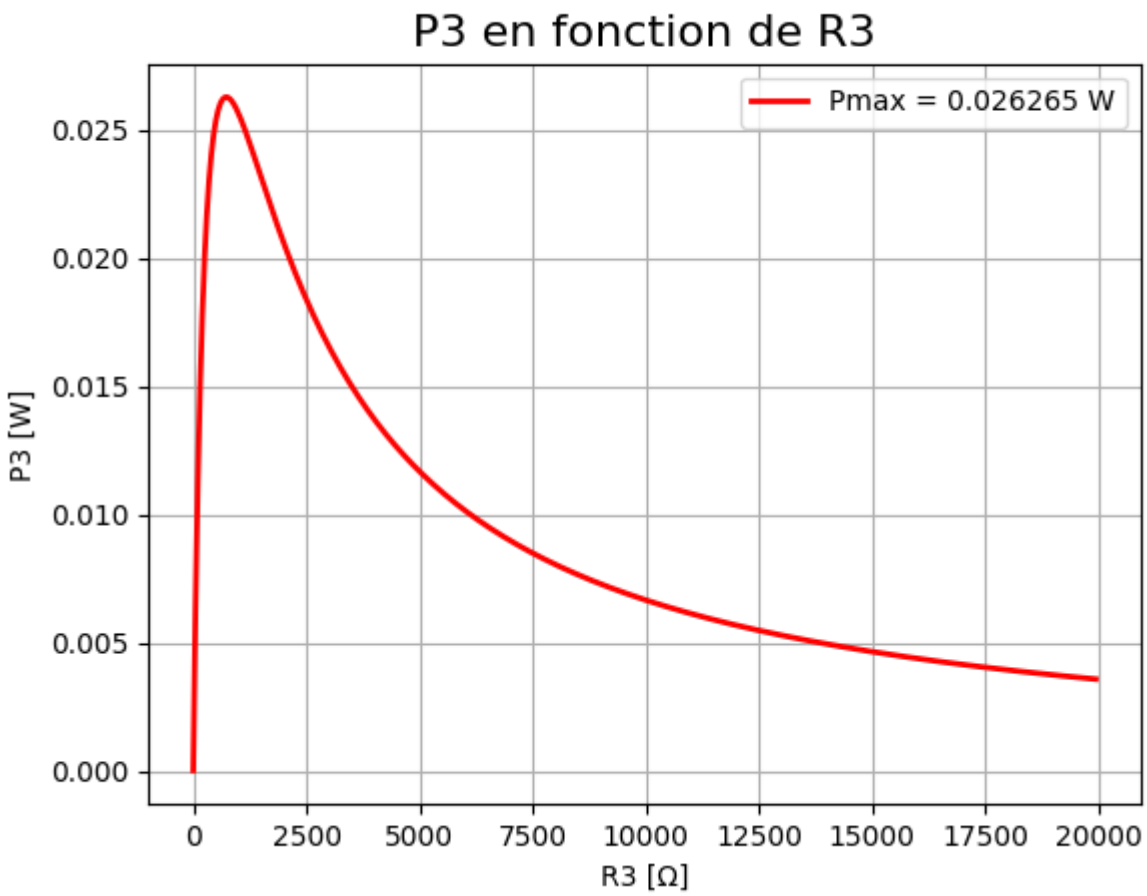
```
Etude de l'évolution de U2 en fonction de R3

R3 = ? 0

Propriétés de R3 :
Résistance : 0.000000 Ω
Propriétés de U2 :
Tension : 0.000000 V

R3 = ? 100

Propriétés de R3 :
Résistance : 100.000000 Ω
Propriétés de U2 :
Tension : 1.055375 V

R3 = ?
```

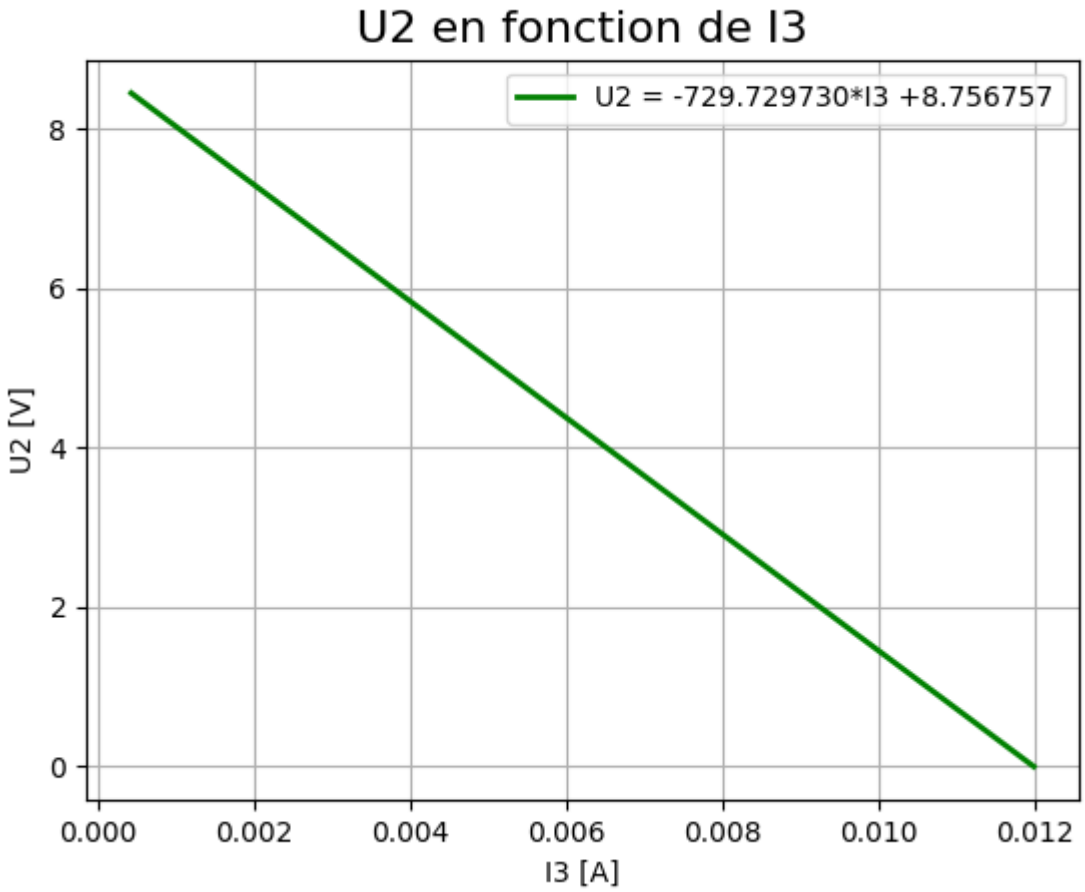
Télécharger les scripts exemple2a_analyse_parametrique.py et exemple2a_analyse_parametrique_bis.py

Exemple 2b

On utilise ici le module `matplotlib` pour faire de belles courbes :


U2 en fonction de R3
U2 [V]
0
2
4
6
8
0
2500
5000
7500
10000
12500
15000
17500
20000
R3 [Ω]


P3 en fonction de R3
Pmax = 0.026265 W
P3 [W]
0.025
0.020
0.015
0.010
0.005
0.000
0
2500
5000
7500
10000
12500
15000
17500
20000
R3 [Ω]



Télécharger les scripts exemple2b_analyse_parametrique.py et exemple2b_analyse_parametrique_bis.py

Exemple 3

On cherche la valeur de R3 qui donne U2 = 6,00 V.
On utilise une méthode dichotomique :

```
Itération  R3 (Ω)                 U2 (V)
         1                5000.0
7.641509433962264
         2                2500.0
6.7782426778242675
         3                1250.0
5.5290102389078495
         4                1875.0
6.303501945525291
         5                1562.5
5.969049373618275
```

```
         6               1718.75
6.146947223180407
         7              1640.625
6.060929983965794
         8             1601.5625
6.015759442079521 5
         9            1582.03125
5.992601726263872
        10           1591.796875
6.004229291252643
        11          1586.9140625
5.998427762523782
        12         1589.35546875
6.001331580654524
        13        1588.134765625
5.999880436236933
        14       1588.7451171875
6.000606199456697
        15     1588.43994140625
6.000243365618415
        16    1588.287353515625
6.000061912872932
        17   1588.2110595703125
5.999971177541544


Résultat : 1588.2110595703125 Ω
```

Télécharger les scripts exemple3_solution.py et exemple3_solution_bis.py

Avantages et limitations

Ce module gère les opérations arithmétiques de base (+, -, *, /) ainsi que // qui désigne deux résistances en parallèle.

La cohérence des unités est contrôlée :

```
>>> V3 = V1 -V2 +I3
TypeError
>>> I = I1 +0.5
```

```
TypeError
>>> I2 = Courant(0.5)
>>> I = I1 + I2
>>> I = 5*I2 -V1/R1 + I3
```

Le résultat d'une opération doit donner une grandeur dont l'unité est en volt, ampère, ohm, siemens, watt ou sans unité.
Autrement, vous aurez une erreur :

```
>>> R1/V1  # Ω/V -> Erreur
TypeError
>>> R2*(R3/(R2 +R3))  # Ω*(Ω/Ω) -> Ω*() -> Ω
>>> R2*R3/(R2 +R3)  # Ω*Ω -> Erreur
TypeError
>>> P = V1*(V1/R1)  # V*(V/Ω) -> V*A -> W
>>> P = V1*V1/R1  # V*V -> Erreur
TypeError
>>> V1()**2/R1()
```

Un module Python pour l'étude des circuits électriques linéaires en régime continu

Ce module pédagogique s'appuie sur l'écriture des lois de l'électricité pour résoudre un problème donné.
Ce n'est donc pas un logiciel de simulation.

Installation

pypi.org/project/dc-electricity

```
pip install dc-electricity
```

Documentation

Version française
English version

Exemple 0 - Loi d'Ohm

Une résistance de 100 Ω est alimentée par une tension de 5 volts.
Que vaut le courant dans la résistance ?
Quelle est la puissance consommée ?

```
>>> from dcelectricity.dc_fr import *
>>> loi = Loi()
>>> U = Tension(5)
>>> R = Resistance(100)
>>> I = loi.Ohm(v=U, r=R)
>>> I
Intensité du courant : 0.05 A (50.000000 mA)
>>> P = loi.Joule(r=R, i=I)
>>> P
Puissance : 0.25 W (250.000000 mW)
```

On peut aussi utiliser des expressions littérales.
Notez que si l'expresssion est "inattendue", vous aurez droit à un message d'erreur :

```
>>> I = U/R
>>> I
Intensité du courant : 0.05 A (50.000000 mA)
>>> I = R/U
TypeError
```

Pour la puissance :

```
>>> P = R*I*I
>>> P = U*(U/R)
>>> P = U*I
```

Exemple 0b - Résistance équivalente

Que vaut la résistance équivalente à l'association en parallèle de 2 résistances 1 kΩ et 1,5 kΩ ?

```
>>> from dcelectricity.dc_fr import *
>>> loi = Loi()
>>> R1 = Resistance(1, 'k')
>>> R2 = Resistance(1.5, 'k')
>>> Req = loi.Rparallele(R1, R2)
>>> Req
```

```
Résistance : 600.000000 Ω
```

Voici une notation équivalente bien pratique :

```
>>> Req = R1//R2
>>> Req
Résistance : 600.000000 Ω
```

Avec les équations littérales :

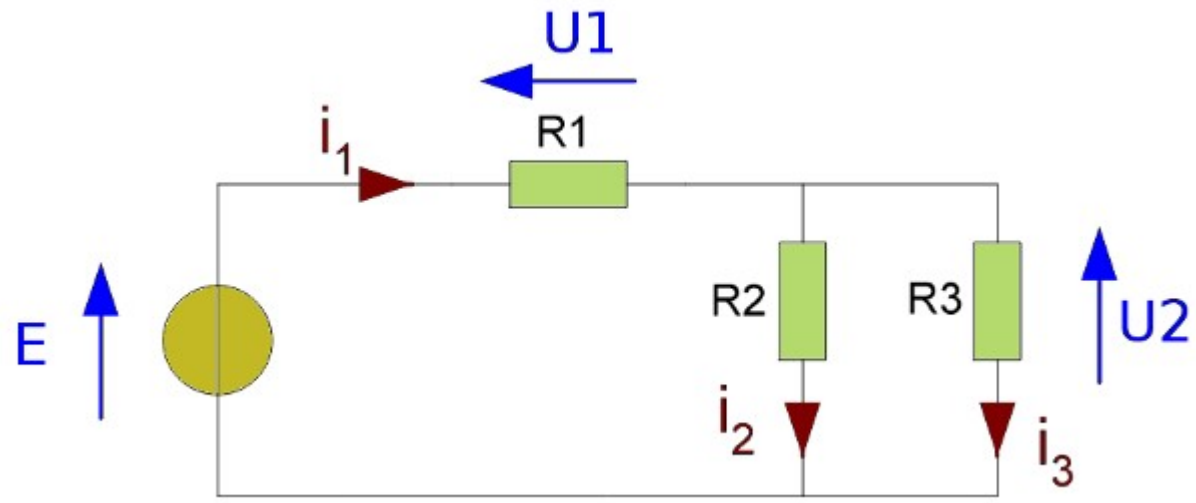
```
>>> Req = 1/(1/R1+1/R2)
>>> Req = R1*(R2/(R1+R2))
```

Enfin, en passant par les conductances :

```
>>> G1 = 1/R1
>>> G2 = 1/R2
>>> Geq = G1 +G2
>>> Geq
Conductance : 0.00166667 S (1.666667 mS)
>>> Req = 1/Geq
>>> Req
Résistance : 600.000000 Ω
```

Exemple 1

On s'intéresse au circuit électrique suivant :



Données :
E = 12 V ; R1 = 1 kΩ ; R2 = 2,7 kΩ et R3 = 1,8 kΩ

Télécharger les scripts exemple1.py et exemple1_bis.py

Ce script calcule les courants I1, I2 et I3, les tensions U1 et U2 et fait un bilan de puissances.

```
>>> from dcelectricity.dc_fr import *
>>> E = Tension(12)
>>> E
Tension : 12.000000 V
>>> R1 = Resistance(1, 'k')
>>> R1
Résistance : 1000 Ω (1.000000 kΩ)
>>> R2 = Resistance(2.7, 'k')
>>> R3 = Resistance(1.8, 'k')
>>> R23 = R2//R3  # résistances en parallèle
>>> R23
Résistance : 1080 Ω (1.080000 kΩ)
>>> Req = R1 +R23  # résistances en série
>>> Req
Résistance : 2080 Ω (2.080000 kΩ)
>>> I1 = E/Req  # Loi d'Ohm
>>> I1
Intensité du courant : 0.00576923 A (5.769231 mA)
>>> V1 = R1*I1  # Loi d'Ohm
>>> V1
Tension : 5.769231 V
>>> V2 = E -V1  # Loi des branches
>>> V2
Tension : 6.230769 V
>>> I2 = V2/R2  # Loi d'Ohm
>>> I2
Intensité du courant : 0.00230769 A (2.307692 mA)
```

```
>>> I3 = I1 -I2  # Loi des nœuds
>>> I3
Intensité du courant : 0.00346154 A (3.461538 mA)
```

Une autre approche :

```
>>> loi = Loi()
>>> # diviseur de tension
>>> V2 = loi.DiviseurTension(vtotal=E, r=R2//R3, r2=R1)
>>> V2
Tension : 6.230769 V
>>> # Théorème de Millman
>>> masse = Tension(0)
>>> V2 = loi.Millman(v_r=[(E, R1), (masse, R2), (masse, R3)])
>>> V2
Tension : 6.230769 V
>>> (E/R1)/(1/R1 +1/R2 +1/R3)
Tension : 6.230769 V
>>> V2/E
0.5192307692307693
>>> # diviseur de courant
>>> I3 = loi.DiviseurCourant(itotal=I1, r=R3, r2=R2)
>>> I3
Intensité du courant : 0.00346154 A (3.461538 mA)
>>> I1*R2/(R2 +R3)
Intensité du courant : 0.00346154 A (3.461538 mA)
```

Puissances et loi de Joule

```
>>> P = E*I1  # puissance fournie par la source E
>>> P
Puissance : 0.0692308 W (69.230769 mW)
```

```
>>> P1 = loi.Joule(r=R1, i=I1)
>>> P1
Puissance : 0.033284 W (33.284024 mW)
>>> R1*I1*I1
Puissance : 0.033284 W (33.284024 mW)
>>> loi.Joule(r=R1, v=V1)
Puissance : 0.033284 W (33.284024 mW)
>>> V1*(V1/R1)
Puissance : 0.033284 W (33.284024 mW)
>>> P2 = loi.Joule(r=R2, i=I2)
>>> P2
Puissance : 0.0143787 W (14.378698 mW)
>>> P3 = loi.Joule(r=R3, i=I3)
Puissance : 0.021568 W (21.568047 mW)
>>> P1 +P2 +P3
Puissance : 0.0692308 W (69.230769 mW)
```

Exemple 2a

Le même circuit électrique mais avec une résistance R3 ajustable :

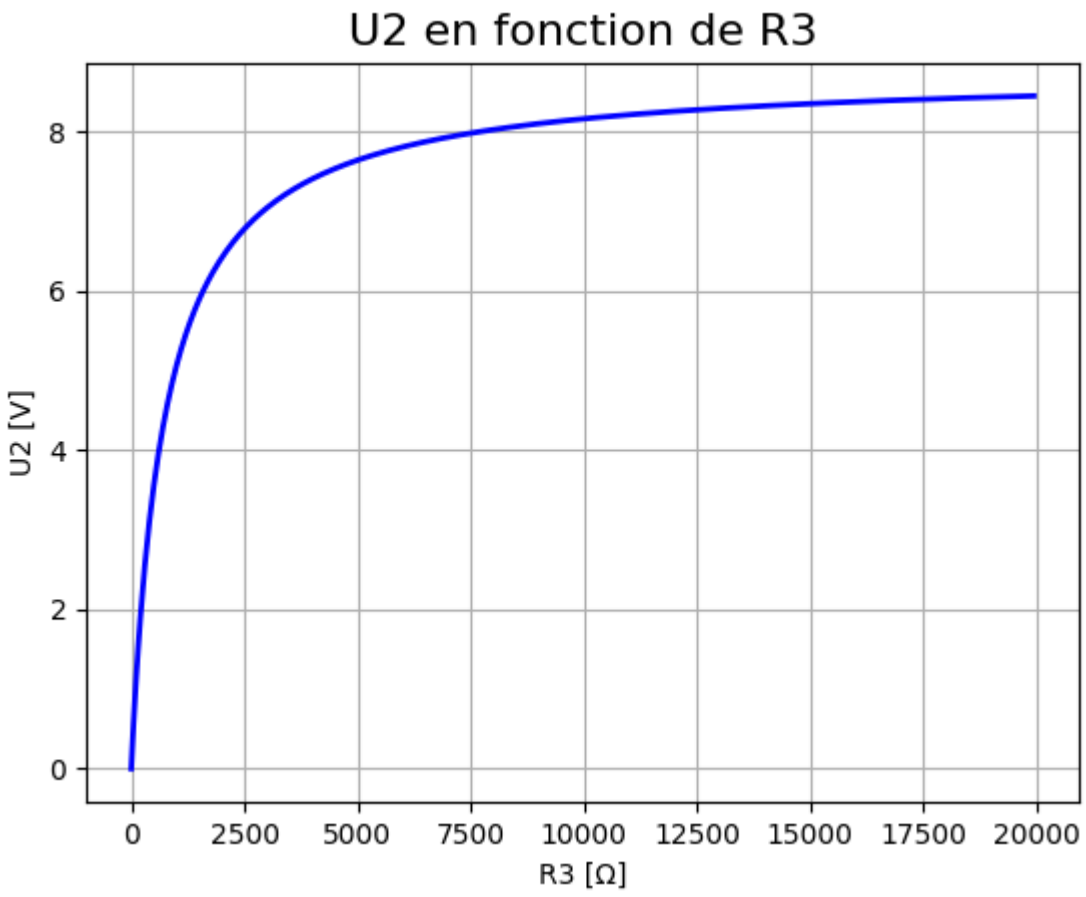
```
Etude de l'évolution de U2 en fonction de R3

R3 = ? 0

Propriétés de R3 :
Résistance : 0.000000 Ω
Propriétés de U2 :
Tension : 0.000000 V

R3 = ? 100
```

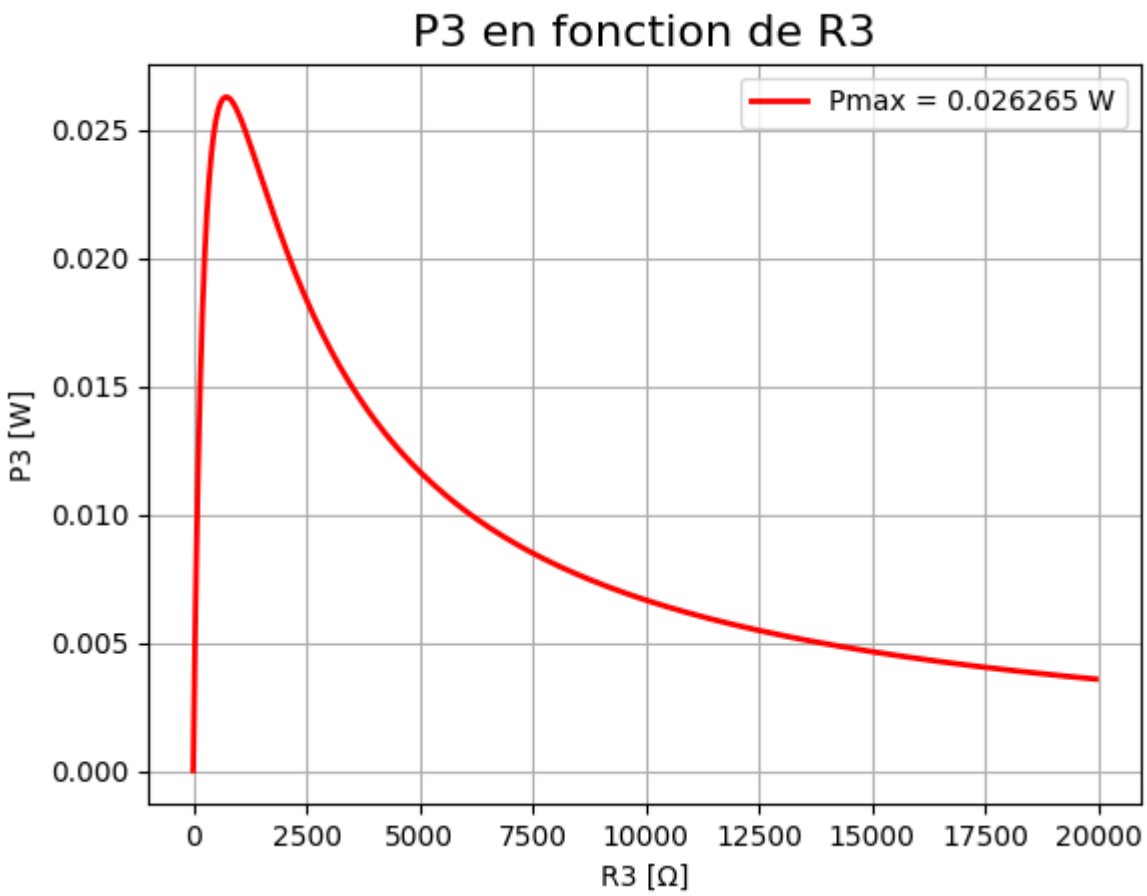
```
Propriétés de R3 :
Résistance : 100.000000 Ω
Propriétés de U2 :
Tension : 1.055375 V


R3 = ?
```

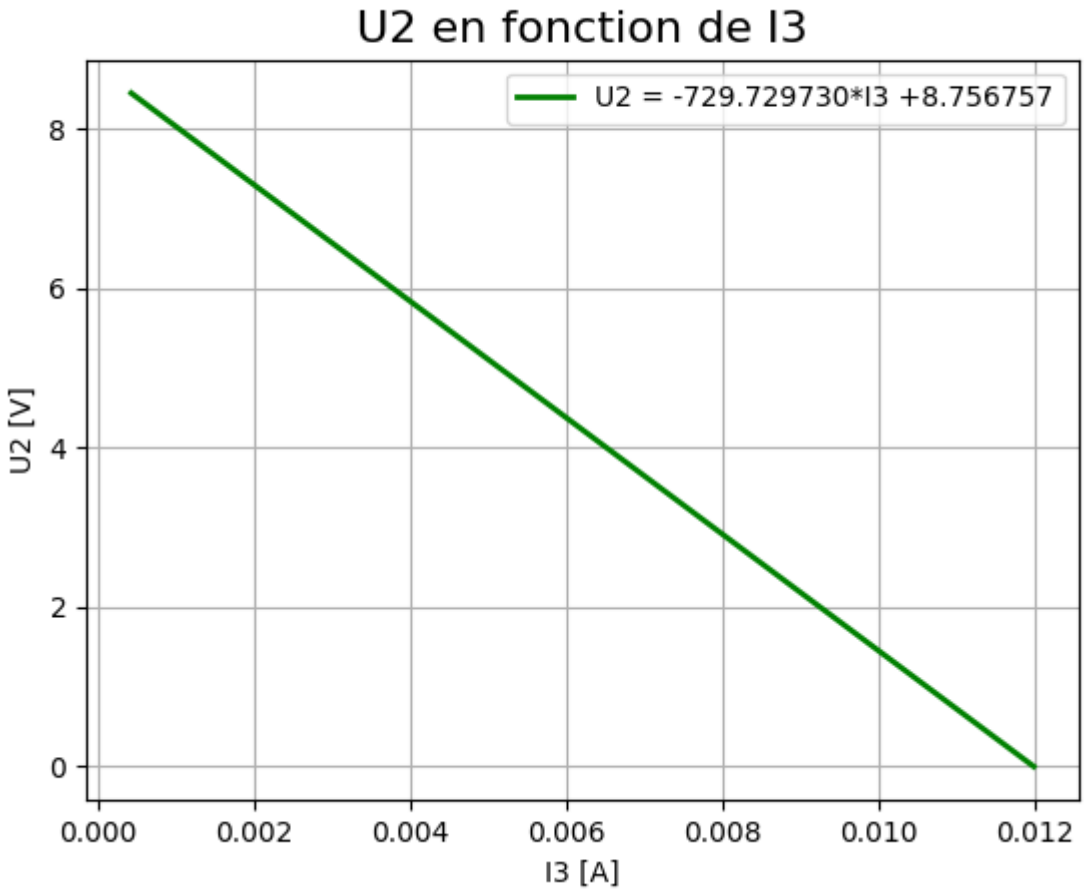
Télécharger les scripts exemple2a_analyse_parametrique.py et exemple2a_analyse_parametrique_bis.py

Exemple 2b

On utilise ici le module `matplotlib` pour faire de belles courbes :




P3 en fonction de R3
Pmax = 0.026265 W
P3 [W]
0.025
0.020
0.015
0.010
0.005
0.000
0
2500
5000
7500
10000
12500
15000
17500
20000
R3 [Ω]



Télécharger les scripts exemple2b_analyse_parametrique.py et exemple2b_analyse_parametrique_bis.py

Exemple 3

On cherche la valeur de R3 qui donne U2 = 6,00 V.
On utilise une méthode dichotomique :

```
Itération  R3 (Ω)                 U2 (V)
         1                5000.0
7.641509433962264
         2                2500.0
6.7782426778242675
         3                1250.0
5.5290102389078495
         4                1875.0
6.303501945525291
         5                1562.5
5.969049373618275
```

```
         6               1718.75
6.146947223180407
         7              1640.625
6.060929983965794
         8             1601.5625
6.015759442079521 5
         9            1582.03125
5.992601726263872
        10           1591.796875
6.004229291252643
        11          1586.9140625
5.998427762523782
        12         1589.35546875
6.001331580654524
        13        1588.134765625
5.999880436236933
        14       1588.7451171875
6.000606199456697
        15     1588.43994140625
6.000243365618415
        16    1588.287353515625
6.000061912872932
        17   1588.2110595703125
5.999971177541544


Résultat : 1588.2110595703125 Ω
```

Télécharger les scripts exemple3_solution.py et exemple3_solution_bis.py

Avantages et limitations

Ce module gère les opérations arithmétiques de base (+, -, *, /) ainsi que // qui désigne deux résistances en parallèle.

La cohérence des unités est contrôlée :

```
>>> V3 = V1 -V2 +I3
TypeError
>>> I = I1 +0.5
```

```
TypeError
>>> I2 = Courant(0.5)
>>> I = I1 + I2
>>> I = 5*I2 -V1/R1 + I3
```

Le résultat d'une opération doit donner une grandeur dont l'unité est en volt, ampère, ohm, siemens, watt ou sans unité.
Autrement, vous aurez une erreur :

```
>>> R1/V1  # Ω/V -> Erreur
TypeError
>>> R2*(R3/(R2 +R3))  # Ω*(Ω/Ω) -> Ω*() -> Ω
>>> R2*R3/(R2 +R3)  # Ω*Ω -> Erreur
TypeError
>>> P = V1*(V1/R1)  # V*(V/Ω) -> V*A -> W
>>> P = V1*V1/R1  # V*V -> Erreur
TypeError
>>> V1()**2/R1()
```