Eléments de base du langage de programmation Python

Plan

1 Introduction

2 Installation et prise en main

3 Variables et types

4 Structures de Contrôle

5 Les fonctions

6 Les fichiers

7 Interfaces graphiques avec Tkinter

Introduction

Python est l'un des langages de programmation les plus utilisés dans le monde, il a été crée en 1991, ses dernières années ont vu son utilisation croitre de manière considérable.

- Il est simple, il met l'accent sur la lisibilité
- Facile à apprendre.
- Bibliothèque très riche permettant une meilleure productivité.

Il est utilisé dans divers domaines :

- Intelligence artificielle.
- Analyse de données.
- Développement web.
- Big Data, robotique, cybersécurité, ...

Installation et prise en main

On peut installer facilement python sur sa machine, il est recommandé d'installer la derniere version [1].

Il existe aussi plusieurs distributions de python :

- Anaconda, enrichi avec des bibliothèques pour l'IA et data science.
- WinPython : distribution scientifique de python.
- ...

1. La dernière version actuellement (2023) est la 3.10

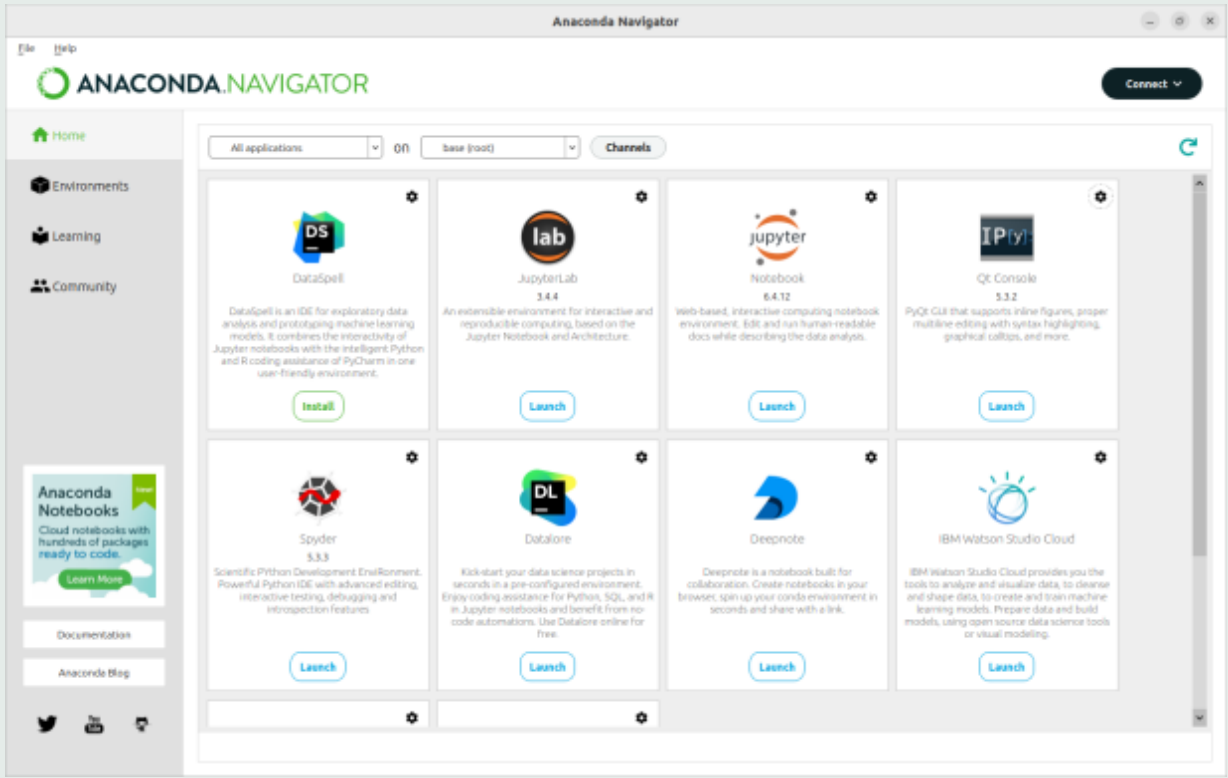
Installation et prise en main

Anaconda est une distribution permettant d'effectuer des traitements en science des données et de l'apprentissage automatique.

- Elle offre des milliers de packages open source pré-installés,
- Elle peut être installée sous Windows, macOS ou Linux.
- Offre un environement pour faciliter son utilisation (installtion de nouveaux packages, ...).
- Il y a une version gratuite.
- Offre un support communautaire large et gratuit.

1. Rendez-vous sur https ://www.anaconda.com/products/distribution,
2. Télécharger la version qui correspond à votre système d'exploitation (Mac/Windows/Linux).
3. Double-cliquez sur le fichier téléchargé pour installer anaconda sur votre ordinateur.
4. acceptez les propositions par défaut à chaque fois en cliquant sur **suivant** (Next).

1. Cliquez sur Start (dans le cas de windows),
2. cherchez Anaconda Navigator, puis cliquez dessus pour l'ouvrir
3. Anaconda s'ouvre est affiche toutes les options disponibles (voir la capture d'écran suivante)



1. Cliquez sur lancer (launch) dans la case correspondant à Spyder.
2. On peut aussi lancer Spyder à partir de la console avec la commaande spyder.

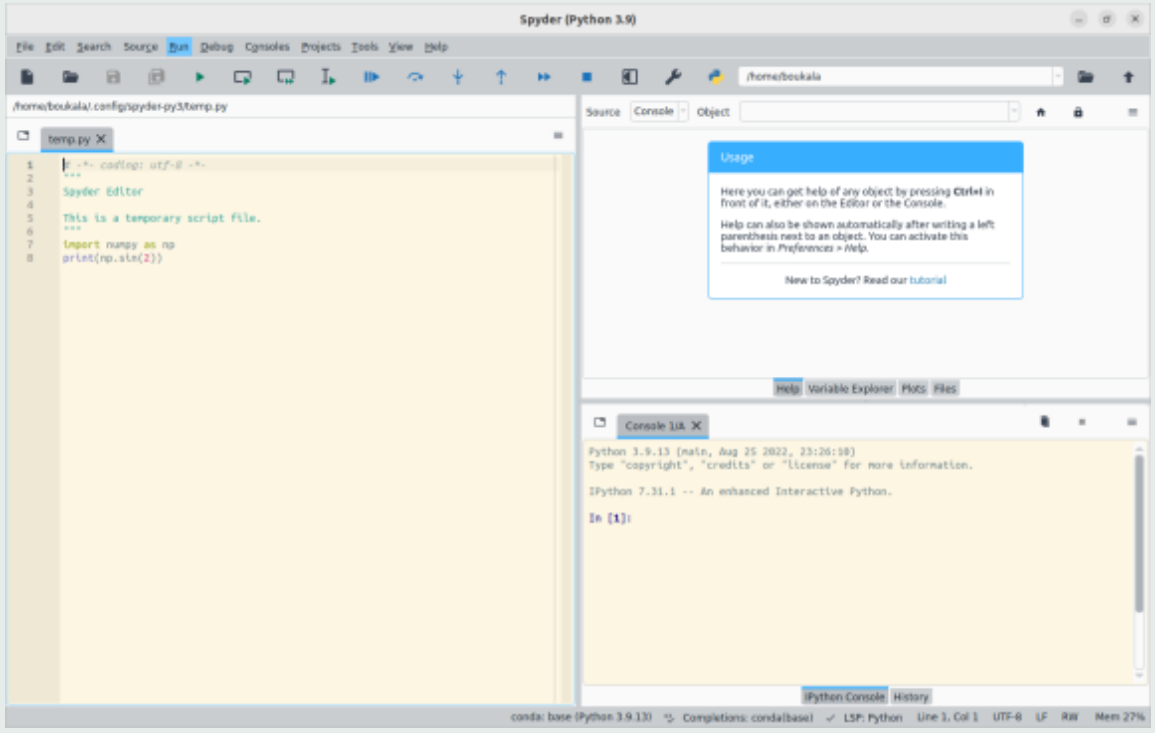


Figure – Fenêtre de Spyder

tapez **print('Bonjour')**, puis cliquez sur le bouton **Exécuter** ou dans le menu **Run**.

```
print('Bonjour')
```

```
Bonjour
```

tapez **print(11+15)**, puis cliquez sur le bouton **Exécuter**.

```
print(11+5)
```

```
16
```

Ce qui est entre quottes ou entre guillemets est affiché tel quel.
tapez **print('11+15')**, puis cliquez sur le bouton **Exécuter**.

```
print('11+5')
```

```
11+5
```

Remarque

Pour des raisons de clarté, les captures d'écran ont été prise sur l'outil jupyter notebook.

Tapez les lignes suivantes :

```
a = 12
b = 7
c = a + b
a += 1 # equivalent à a = a + 1
print(a)
print(c)
print(b/4)
```

Lancez l'exécution, vous obtiendrez les résultats suivants :

```
a = 12
b = 7
c = a + b
a += 1 # equivalent à a = a + 1
print(a)
print(c)
print(b/4)
```

```
13
19
1.75
```

On peut aussi afficher plusieurs éléments, séparés par des virgules, dans le print. Tapez les lignes suivantes :

```
a = 25
b = "Toto"
print('bonjour', b)
print('ton âge est :', a, 'ans')
```

Lancez l'exécution, vous obtiendrez les résultats suivants :

```
a = 25
b = "Toto"
print('bonjour', b)
print('ton âge est', a, 'ans')
```

```
bonjour Toto
ton âge est 25 ans
```

La fonction input() permet de lire des données saisie à partir du clavier.

Tapez les instructions suivantes :

```
nom = input('Donner votre nom : ')
age = input('Donner votre age : ')
print('Bonjour', nom, ' votre age est de', age, 'ans')
```

Lancez l'exécution.
Le programme va vous demander de donner votre nom. Introduisez votre nom et terminer par la touche Entrée.
Il va vous demander ensuite d'introduire votre age.
Lorsque vous aurez saisi votre age et appuyé sur la touche Entrée, le programme affiche le résultat.

Attention

les données récupérées par la fonction input() sont des chaînes de caractères. Les données numériques ne peuvent être utilisées telles quelles dans des calculs éventuels, elles doivent être convertis avant, sinon cela génère une erreur d'exécution (voir la capture suivante).

Nous reviendrons sur ce point plus loin.

```
a = input('a: ')
b = input('b: ')
print(a-b)
```

```
a: 12
b: 14

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_12379/3870947848.py in <module>
      3 a = input('a: ')
      4 b = input('b: ')
----> 5 print(a-b)

TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

Les commentaires peuvent servir à expliquer une partie d'un programme, ou à mettre des indications dans le code source, comme l'auteur ... .
Les commentaires sont ignorés lors de l'exécution.
Si on veut insérer un commentaire sur une seule ligne dans notre programme, on met un # au début du commentaire.
Si le commentaire est sur plusieurs lignes, on l'encadre par trois guillemets """.
Exemple :

```
# Mon programme
# lecture de données

nom = input('Veuillez introduire votre nom : ')
age = input('Votre age : ')

"""
Vérification des données introduites et
affichage des résultats
"""

print('Bonjour', nom, 'votre age est', age)
```

```
Veuillez introduire votre nom : Ali
Votre age : 15
Bonjour Ali votre age est 15
```

- variable permet de conserver une valeur (une donnée) dans un programme.
- La variable est définie par un nom, qui est composé de lettres et de chiffres, le premier caractère doit être une lettre.
- Le type d'une variable (nombre, chaîne de caractères, ...) est reconnu par python lorsqu'on affecte une donnée à cette variable.
- Une variable ne peut être utilisée si aucune donnée ne lui a été affectée.

Exemple :

```
nom = 'Omar'    #   La variable nom est une variable de type chaîne de caractère  (str)
                #   Une chaîne doit être entre deux apostrophes ou deux guillemets

age = 25        #   La variable age est de type entier  (int)

taille = 1.75   #   La variable taille est de type flotant (float)

print(nom, 'a', age, 'ans', 'sa taille est', taille, 'm')


c1 = "Ali est l'enfant de Omar"      # La vriable c1 est de type chaîne aussi.
print(c1)

a = 1.5         #   a est une variable de type flottant
b = a * 2       #   b est aussi une variable de type flottant

print('Le double de ', a, 'est', b)
```

```
Omar a 25 ans sa taille est 1.75 m
Ali est l'enfant de Omar
Le double de  1.5 est 3.0
```

- Python est un language à typage dynamique.
- Il n'est pas nécessaire de déclarer les variables avant de pouvoir leur affecter une valeur.
- La valeur que l'on affecte possède un type qui dépend de la nature des données (nombre entier, nombre à virgule, chaîne de caractères, etc).
- Le type d'une variable peut changer si on change sa valeur.
- Pour connaître le type d'une variable on utilise la fonction type()

Les principaux types dans python sont :

- Le type int (entier).
- Le type float (flottant).
- Le type str (chaîne de caractères).
- Le type bool (booléen).
- Le type complex (complexe).

Python dispose aussi de types plus élaborés, construits à partir des types de bases (int, float, str, ...) ou à partir de types construits.

- Les tuples
- Les listes
- Les dictionnaires
- Les ensembles

Les tuples

Le tuple (tuple) est une structure permetant de créer une collection ordonnée de plusieurs éléments, séparés par des virgule et entourés par des parenthèses.

- C'est une structure **indexée**, on peut accédé à ses éléments par leurs indexes. Le premier à pour index 0, le deuxième 1, ...
- On peut avoir la même valeur se répéter plusieurs fois dans un tuple.
- Un élément d'un tuple peut lui même être un tuple.
- On ne peut pas modifier un tuple une fois qu'il a été créé.
- La fonction len() permet d'avoir la taille (nombre d'éléments) d'un tuple.

Exemple de tuple

```
t = (3, 12, 1.4, 'pomme')         #  t est un tuple
print(t, type(t))

a = t[0]
print(a)

a = t[2]
print(a)

a, b, c, d = t
print(a,d)

jours = ('dim', 'lun', 'mar', 'mer', 'jeu', 'ven', 'sam')
print(jours)
print(jours[2], type(jours[2]))
```

```
(3, 12, 1.4, 'pomme') <class 'tuple'>
3
1.4
3 pomme
('dim', 'lun', 'mar', 'mer', 'jeu', 'ven', 'sam')
mar <class 'str'>
```

Les listes

En Python, une liste est définie comme une collection d'éléments séparés par des virgules, encadrée par des crochets.

- Les éléments sont modifiables.
- on peut ajouter d'autres éléments à une liste avec la fonction append().
- On peut avoir la même valeur se répéter plusieurs fois dans un liste.
- C'est une structure **indexée**, on peut accédé à ses éléments par leurs indexes. Le premier à pour index 0, le deuxième 1, ...
- Un élément d'une liste peut lui même être une liste.
- La fonction len() permet d'avoir la taille (nombre d'éléments) d'une liste.

Exemple de liste

```
l = [3, 12, 1.4, 'pomme']          #  t est une liste
print(l, type(l))

print(l[2])                        #  affiche le troisième élément

l[1] = 3.14                        #  modification du 2ème élément
print(l)


l.append('Terre')                  #  ajoout d'un élémént à la liste
l.append(5)                        #  ajoout d'un élémént à la liste
print(l)
```

```
[3, 12, 1.4, 'pomme'] <class 'list'>
1.4
[3, 3.14, 1.4, 'pomme']
[3, 3.14, 1.4, 'pomme', 'Terre', 5]
```

Les ensembles

Un ensemble est défini comme une collection d'éléments séparés par des virgules, encadrée par des accolades.

- Les éléments d'un ensemble ne se répètent pas.
- On ne peut pas accéder à un élément par son index, mais on peut parcourir un ensemble.
- Ils ne sont pas modifiables.
- On peut ajouter un élément à un ensemble avec la fonction add().
- On peut ajouter des éléments d'un autre ensemble, d'un tuple ou d'une liste à un ensemble avec la fonction update().
- On peut supprimer un éléments avec la fonction remove() ou discard().

Les ensembles, suite

- on peut tester si une donnée appartient à un ensemble avec in.
- La fonction clear() permet de vider un ensemble.
- La fonction len() permet d'avoir la taille (nombre d'éléments) d'un ensemble.
- Plusieurs autres fonctions sont définies pour réaliser des opérations sur les ensembles telles que : l'union, l'intersection, la différence, ...

Exemple d'ensembles à refaire pour inclure l'union, l'intersection, la différence, ...

```
e = {1, 3.5, 'toto'}              #  création d'un ensemble
print(e)

e.add(10)                         #  ajout d'un élément à l'ensemble e
print(e)

e.remove(3.5)                     #  suppression d'un élément de l'ensemble e,
print(e)

s = {2, 7}
e.update(s)                       #  ajout de l'ensemble s dans l'ensemble e
print(e)

print("La longueur de l'ensemble e est",len(e))
```

```
{1, 'toto', 3.5}
{10, 1, 'toto', 3.5}
{10, 1, 'toto'}
{1, 2, 'toto', 7, 10}
La longueur de l'ensemble e est 5
```

Les dictionnaires

Le dictionnaire permet de stocker des données sous forme de clé : valeur. les items clé : valeur sont séparés par des virgules, le tout est encadré par des accolades.

- Les clés sont mises entre guillemets.
- Les valeurs peuvent être de type simple (nombre, chaine de caractères, ...) ou de type construit (dictionnaire, liste, ...).
- Pour accéder à une valeur d'un item on utilise sa clé.
- De même pour modifier un item.
- On peut ajouter de nouveaux items à un dictionnaire
- Pour supprimer un item on utilise la fonction pop()
- Plusieurs autres fonctions existent pour manipuler les dictionnaires, on peut citer clear(), update(), keys(), values(), items(), ...

Exemple sur les dictionnaires

```
#  Création d'un dictionnaire affecté à la variable pers
pers = {"nom":'Ben', "prenom":'Ali', "age":28}
#  Ben, Ali et 28 sont les valeurs respectives des clés nom, prenom et age.

print('pers =', pers, type(pers))

print('Le nom :', pers["nom"])              #  Accès à l'item nom.

pers["taille"] = 1.75           #  Ajout d'un nouvel item.
print('pers avec le nouvel item taille :', pers)

pers.pop("age")                 #  Supression de l'item age.
print("pers après suppression de l'item age :",pers)

#  Dictionnaire contenant un élément dictionnaire
p = {"nom":'Ben', "prenom":'Ali', "Age":28, "adresse":{"rue":"Rue des jardins", "ville":"Alger"}}
print('p :', p)

print('La rue est:', p["adresse"]["rue"])      #  Accès à l'attribut rue de l'adresse.
```

```
pers = {'nom': 'Ben', 'prenom': 'Ali', 'age': 28} <class 'dict'>
Le nom : Ben
pers avec le nouvel item taille : {'nom': 'Ben', 'prenom': 'Ali', 'age': 28, 'taille': 1.75}
pers après suppression de l'item age : {'nom': 'Ben', 'prenom': 'Ali', 'taille': 1.75}
p : {'nom': 'Ben', 'prenom': 'Ali', 'Age': 28, 'adresse': {'rue': 'Rue des jardins', 'ville': 'Alger'}}
La rue est: Rue des jardins
```

Ecrire un programme python qui demande à saisir la longueur et la largeur d'un rectancle, calcule et affiche son périmètre et sa surface.

Ecrire un programme python qui demande à saisir le rayon d'un circle, calcule et affiche son périmètre et sa surface.

L'instruction if permet d'executer un bloc d'instruction ou un autre selon qu'une condition soit vérifiée ou pas.

Les conditions peuvent être des comparaîsons ou tout autre expression qui donne un résultat vrai ou faux.

Plusieurs conditions peuvent être combinées en utilisant des opérateurs logiques (and, or) pour donner une nouvelle condition.

Exemple sur les conditions

```
a = 5            #  on affecte 5 à a
b = 7            #  et 7 à b

print(a < b)                    #   a < b  est vraie
print(a == b)                   #   on utilise == pour tester l'égalité, le symbole = est utilisé pour l'affectation
print(b != 0)                   #   on teste si b est différent de 0
print((a <= b) and (a >= 0))    #   tester que a est compris entre 0 et b
print((a > b) or (a < 0))       #   tester que a n'est pas dans l'intervalle 0 et b
print(not(b > 0))               #   tester la négation
print((a-b) > 0)                #   tester le resultat d'une expression
```

```
True
False
True
True
False
False
False
```

La forme la plus simple de l'instruction if est :

if condition : instruction

Exemple :

```
age = 15
if age < 18 : print('Mineur')
print('Fin')
```

```
Mineur
Fin
```

```
age = 23
if age < 18 : print('Mineur')
print('Fin')
```

```
Fin
```

Si le nombre d'instructions à exécuter, lorsque la condition est vraie, est plus important on adopte alors la forme suivante :

```
if condition :
    instruction1
    instruction2
    ...
    instructionN
```

Les instructions instruction1, instruction2, ... , instructionN constituent un bloc, elles sont exécutées si la condition est vraie.

Le bloc

Un bloc est un ensemble d'instructions ayant la même indentation (décalage avec la touche d'indentation vers la droite).

Exemple sur le if

```
a = 5            #  on affecte 5 à a
b = 7            #  et 7 à b

print(a < b)                    #   a < b  est vraie
print(a == b)                   #   on utilise == pour tester l'égalité, le symbole = est utilisé pour l'affectation
print(b != 0)                   #   on teste si b est différent de 0
print((a <= b) and (a >= 0))    #   tester que a est compris entre 0 et b
print((a > b) or (a < 0))       #   tester que a n'est pas dans l'intervalle 0 et b
print(not(b > 0))               #   tester la négation
print((a-b) > 0)                #   tester le resultat d'une expression
```

```
True
False
True
True
False
False
False
```

L'instruction if else permet d'exécuter un **bloc** d'instruction si la condition est vraie, ou d'exécuter un autre bloc si la condition est fausse. Sa syntaxe est :

```
if condition :
        bloc1
else :
        bloc2
```

Exemple sur le if else

```
age = 16
if age < 18 :
    print('Mineur')
else :
    print('majeur')
print('Fin')
```

```
Mineur
Fin
```

```
age = 25
if age < 18 :
    print('Mineur')
else :
    print('majeur')
print('Fin')
```

```
majeur
Fin
```

On peut avoir des instructions if imbriquées Exemple

```
a = -4
if a > 0 :
    print ('a est positif')
else :
    if a < 0 :
        print ('a est négatif')
    else :
        print ('a est nul')

print('Voilà !')
```

```
a est négatif
Voilà !
```

Ecrire un programme python qui demande à saisir deux nombre et affiche le maximum des deux.

L'instruction for est une instruction composée, la première ligne se termine par deux-points :, suivie d'un **bloc** indenté qui constitue le corps de la boucle.

On appelle une itération de la boucle chaque fois que le corps de la boucle est exécuté.

On peut effectuer des itérations en utilisant un compteur

```
for i in range(4) :
    j = i * i
    print('le carré de', i, 'est :', j)
print('fin de la boucle')
```

```
le carré de 0 est : 0
le carré de 1 est : 1
le carré de 2 est : 4
le carré de 3 est : 9
fin de la boucle
```

On peut boucler sur une liste, un ensemble ou une dictionnaire.

```
l = ['ciel', 'nuage', 'soleil', 'planete']
for x in l :
    print(x)
print('---')
```

```
ciel
nuage
soleil
planete
---
```

```
d = {'marque':'BMW', 'puissance':200, 'couleur':'Noir'}
for elt in d.keys() :
    print(elt, '=', d[elt])
```

```
marque = BMW
puissance = 200
couleur = Noir
```

Dans le bloc d'instruction du for, on peut avoir des instructions simples (affectations, affichage, ...), des instructions if ou d'autres boucles for imbriquées

Exercice 1
Ecrire un programme python qui determine et affiche le minimum, le maximum, la somme et la moyenne des éléments de l'ensemble {3, 4, 1.5, 10, -2}.

Exercice 1
Ecrire un programme python qui construit une nouvelle liste l2 à partir de la liste l1, tel que chaque élément de l1 est transformé en sa valeur absolue dans l2.
Exp : l1 = [1, -3, -4, 6, 8], l2 = [1, 3, 4, 6, 8] **Exercice 2**

Ecrire un programme qui donne les nombres premiers compris entre 2 et 100.

On utilise la boucle while, pour répéter l'exécution d'un bloc, si on ne connait pas à l'avance la condition d'arrêt.
L'exécution continue tant que la condition n'est pas vérifiée.

Exemple

```
l = [3, 4, 7, 9, 10, 17, 30]
x = 10                                  # l'élément recherché
i = 0
while (i < len(l)) and (x != l[i]) :    # on s'arrête lorsqu'on trouve x ou si on a vérifié toute la liste
    i = i + 1

if i < len(l):                          # i < à la longueur de la liste, on a donc trouvé x avant la fin de la liste
    print(x, 'exist dans la liste, il est à la position', i+1)
else :
    print("x n'existe pas dans la liste")   # on a parcouru toute la liste sans trouver x
```

```
10 exist dans la liste, il est à la position 5
```

L'instruction break permet de d'interrompre l'exécution d'une boucle (while ou for), et passer à l'instruction qui suit le bloc de la boucle.

L'instruction continue permet d'arrêter l'itération en cour et de passer à l'itération suivante.

Exemple

```
l = [3, 4, 7, 9, 10, 17, 30]
x = 10                                  # l'élément recherché
i = 0
while i < len(l) :  # on s'arrête si on a vérifié toute la liste
    if x == l[i] :
        break       # on s'arrête si on trouve x
    i = i + 1

if i < len(l):                          # i < à la longueur de la liste, on a donc trouvé x avant la fin de la liste
    print(x, 'exist dans la liste, il est à la position', i+1)
else :
    print(x, "n'existe pas dans la liste")   # on a parcouru toute la liste sans trouver x
```

```
10 exist dans la liste, il est à la position 5
```

Exercice 1
Ecrire un programme python qui vérifie si un nombre donné (utilisez la fonction de lecture input()) appartient à l'ensemble {3, 4, 7, 10, 21, 24, 30}.

Exercice 2
Ecrire un programme python qui vérifie si dans une liste un même élément figure plus d'une fois.

Définition d'un fonction

Une fonction c'est une portion de code (de programme) identifié par un nom, on peut lui passer des paramètres et peut retourner un ou plusieurs résultats.

L'avantage des fonctions est :

- Elles sont écrites une seule fois, mais peuvent être appelées plusieurs fois.
- Permettent une meilleure structuration et lisibilité des programmes.

Nous avons vu et utilisé des fonction définies dans python telles que : input(), print(), len(), range(), ...

De base le nombre de fonctions disponibles en python est limité. Néanmoins, des fonctions peuvent être importées pour être utilisées, par exemple, si on veut utiliser des fonctions mathématiques telles que sinus, cosinus, racine carrée, logarithme, ..., il suffit d'incorporer la ligne suivante au début du programme :

from math import *

Exemple

```
from math import *
print(sqrt(16))          #  Calcul et affichage de la racine carrée de 16
print(log(5))            #  Calcul et affichage du logarithme de 5
```

```
4.0
1.6094379124341003
```

Pour importer des fonctions, on peut utiliser l'une des instructions suivantes (On reprend l'exemple de la bibliothèque math) :

from math import * pour importer toutes les fonctions de la bibliothèque math.

ou

from math import sqrt, sin qui permet d'utiliser uniquement les fonctions spécifiées.

ou

import math, Pour utiliser les fonctions, on doit les préfixer par le nom de la bibliothèque.
Exp. math.sqrt(3), math.sin(a)

ou

import math as mt, les fonctions doivent être préfixées par l'alias mt.
Exp. mt.sqrt(15)

On trouve des milliers de bibliothèques, couvrant presque tous les besoins et dans tous les domaines :

- Sciences des données et mathématiques
- Intelligence artificielle
- Accès et manipulation des bases de données
- Manipulation d'images et de vidéos
- Interface graphique
- Développement Web
- Développement de jeux
- ...

- On ne peut pas importer dans un programme des bibliothèques qui ne sont pas déjà installées.
- Le choix de la distribution **anaconda** était motivé par le fait que beaucoup de bibliothèques sont installées par défaut.
- Pour connaître les bibliothèques déja installées dans votre python, il suffit de taper la commande **pip list** ou **conda list** dans une console (terminal).
- Pour installer une nouvelle bibliothèque, dans la plupart des cas, il suffit de taper la commande **pip install nom_bibliothèque**

- Pour définir une fonction dans python, on utilise le mot-clé def, suivi du nom de la fonction.
- Si la fonction retourne des résultats, on utiliser le mot-clé return.
- La fonction peut être appelé dans le programme en utilisant simplement son nom.
- On peut transmettre un ou plusieurs paramètres à une fonction

Exemple :

```
def circle_surf(r):                 #  définition de la fonction circle_surf avec le paramètre r
    s = r * r * 3.14
    return s

s = circle_surf(2)                  #  Appel de la fonction
print('Surface du circle de rayon 2 est', s)

print('Surface du circle de rayon 3 est', circle_surf(3))            #  Un autre appel
```

```
Surface du circle de rayon 2 est 12.56
Surface du circle de rayon 3 est 28.26
```

Exemple d'une fonction à plusieurs paramètres et plusieurs résultats :

```
def rectangle(largeur, longueur):       #  fonction rectangle avec deux paramètres
    perimetre = 2*(largeur + longueur)
    surface = largeur * longueur
    return perimetre, surface           # La fonction retouurne 2 résultats

a = rectangle(2,3)                      # appel de la fonction et récupération des résultat dans la variable a
print(a, type(a))                       # la variable a sera de type tuple

p, s = rectangle(4,5)                   # on peut récuperer les résultat dans deux variables
print('le périmètre est égal à', p)
print('la surface est égale à', s)
```

```
(10, 6) <class 'tuple'>
le périmètre est égal à 18
la surface est égale à 20
```

Si le nombre de résultats est important ou s'il n'est pas connu à l'avance, on peut retourner une liste, un ensemble, ...

```
# définition d'une fonction qui construit la liste des nombres premiers inférieurs à limite

def premiers(limite):
    lp = []                             #  initialisation de la vraiable lp à une liste vide
    for i in range(2,limite):
        premier = True
        for j in range(2,int(i/2)+1):
            if i%j == 0 :               #  Vérifier que i n'est pas divisible par les nombres de 2 à i/2+1
                premier = False
        if premier :
            lp.append(i)                # Ajouter l'élément à la variable lp
    return lp

print(premiers(50))                     # afficher les nombres premiers inférieur à 50
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

La création d'un module en python est très simple. Il suffit d'écrire un ensemble de fonctions dans un fichier, et de l'enregistrer avec une extension .py dans le même dossier que le programme qui va l'utiliser.

Exemple : Nous allons créer un module que nous enregistrerons sous le nom circle.py, dans lequel on défini les fonctions surface() et perimetre().

```
# Cette fonction calcule la surface du circle dont le rayon est donnée en paramètre

def surface(rayon):
    s = rayon * rayon * 3.14
    return s

# Cette fonction calcule le perimetre du circle dont le rayon est donnée en paramètre

def perimetre(rayon):
    p = 2 * rayon * 3.14
    return p
```

L'importation se fait de manière similaire à l'importation des autres bibliothèques.

```
import circle
print('Le périmètre', circle.perimetre(3))
print('La surface', circle.surface(3))
```

```
Le périmètre 18.84
La surface 28.26
```

Ou

```
from circle import *
print('Le périmètre', perimetre(3))
print('La surface', surface(3))
```

```
Le périmètre 18.84
La surface 28.26
```

Les fichiers

- Les fichiers sont très utiles, en particulier lorsque le volume des données est important.
- Python permet d'enregistrer des données dans des fichiers, il peut aussi lire ou modifier une fichier.
- Pour utiliser un fichier, on commence par l'ouvrir en utilisant la fonction open(), cette fonction possède deux paramètres : Le premier pour donner le chemin d'accès au fichier, le second pour donner le mode d'accès (lecture, écriture, ...).
- Après avoir terminé d'utiliser un fichier dans un programme python, il faut le fermer, en utilisant la fonction close().

Les différents modes d'accès aux fichiers sont :

- **r**, accès en lecture.
- **w**, accès en écriture. Si le fichier existe déjà, le contenu est écrasé, si le fichier n'existe pas, il sera alors crée.
- **a**, ouverture en ajout à la fin du fichier. Si le fichier n'existe pas python le crée..
- **x**, créer un nouveau fichier et l'ouvre pour écriture. Si le fichier existe déjà python affiche une erreur.

Exemple : Lecture d'un fichier

```
f = open("fichierdemo.txt", "r")       # Ouverture du fichier fichierdemo.txt en lecture
print(f.read())                        # Lecture et affichage du contenu du fichier
f.close()
```

```
Ce texte est enregistré dans le fichier.
Ce texte va être ajouté à la fin du fichier.
```

Pour écrire du contenu dans un fichier on ouvre le fichier avec le mode "w".

```
f = open("fichierdemo.txt", "w")                    # Ouverture du fichier fichierdemo.txt en écriture
f.write("Ce texte est enregistré dans le fichier.")  # Ecriture du texte dans le fichier
f.close()
```

Pour ajouter du contenu dans un fichier on ouvre le fichier avec le mode append "a".

```
f = open("fichierdemo.txt", "a")                          # Ouverture du fichier fichierdemo.txt en ajout
f.write("\nCe texte va être ajouté à la fin du fichier.")  # Ajout du texte à la fin du fichier
                                                          # Le \n permet d'effectuer un saut de ligne
f.close()
```

Lecture du contenu du fichier

```
f = open("fichierdemo.txt", "r")       # Ouverture du fichier fichierdemo.txt en lecture
print(f.read())                        # Lecture et affichage du contenu du fichier
f.close()
```

```
Ce texte est enregistré dans le fichier.
Ce texte va être ajouté à la fin du fichier.
```

La fonction readlines() permet de lire le contenu d'un fichier et prépare le contenu dans une liste. Chaque élément de la liste contient une ligne.

Chaque élément de la liste se termine par le caractère \n correspondant au code du **saut de ligne**

Les éléments de la liste sont des chaînes de caractères.

```
f = open("fichierdemo.txt", "r")        # Ouverture du fichier fichierdemo.txt en lecture
l = f.readlines()                       # Lecture du contenu du fichier et affectation du contenu dans une liste
print(l)
f.close()
```

```
['Ce texte est enregistré dans le fichier.\n', 'Ce texte va être ajouté à la fin du fichier.']
```

Lorsqu'un fichier est très volumineux, il n'est pas forcément souhaitable d'en stocker tout le contenu en mémoire. Il est à cet effet possible de le parcourir ligne après ligne.

On peut lire une ligne à la fois avec la fonction readline(), ou directement comme le montre l'exemple suivant :

```
f = open("fichierdemo.txt", "r")       # Ouverture du fichier fichierdemo.txt en lecture
for ligne in f:
    print(ligne)
```

```
Ce texte est enregistré dans le fichier.

Ce texte va être ajouté à la fin du fichier.
```

Rmq : IL y a eu double saut de ligne, l'un correspondant au saut de ligne dans le fichier et l'autre au saut de ligne de la fonction print().

Exercice 1

Ecrire un programme python qui demande de manière répétitive à saisir des nombres positifs et les enregistre au fur et à mesure dans un fichier. Le programme s'arrête lorsqu'on saisie -1.

Exercice 1

Ecrire un programme python qui lit des valeurs du fichier précédent, calcule la moyenne, détermine le minimum et la maximun.

La bibliothèque **Tkinter** (Tk interface), installée par défaut dans python, permet de créer des interfaces graphiques.

- des fenêtres,
- des widgets : cadres, boutons, zones de saisie, ...

et permet aussi la gestion des évènements. liés aux clavier, à la souris, ...

Pour utiliser Tkinter on doit l'importer au début du script :

```
from tkinter import *
```

Remarque : Il est recommandé d'utiliser python à travers la ligne de commandes (Terminal ou console) que sous l'environnement jupiter.

La fenêtre est l'objet principal qui va contenir tous les autres objets nécessaires dans votre application.

On utilise la classe Tk du module Tkinter pour créer une fenêtre.

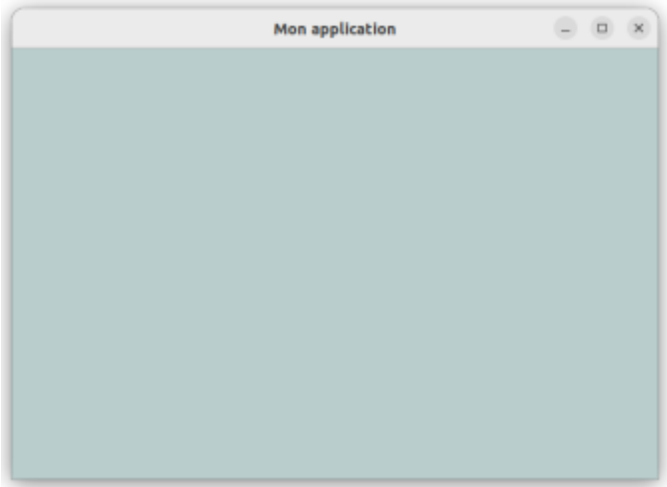
On peut personnaliser la fenêtre :

- en rajoutant un titre
- en précisant ses dimensions
- en changeant son arrière plan
- ...

Script pour générer une fenêtre :

```python
# Importation de l'ensemble des éléments du paquet tkinter
from tkinter import *

# Création d'une fenêtre avec la classe Tk
mafenetre = Tk()

# Ajout d'un titre à la fenêtre
mafenetre.title("Mon application")

# Personnaliser la couleur de l'arrière-plan
mafenetre.config(bg = "#B8CDCC")

# Définir les dimensions par défaut la fenêtre
mafenetre.geometry("600x400")

# Affichage de la fenêtre
mafenetre.mainloop()
```

et voici le resultat :



Tkinter comporte plusieurs widgets qui sont des composants graphiques qui peuvent être ajoutés dans une fenêtre. Les les plus utilisés sont :

- des textes ou images « Label » ,
- des boutons « Button »,
- des cases à cocher « Checkbutton » et des boutons radio « Radiobutton »,
- des listes « Listbox »,
- des entrées d'utilisateur « Entry »,
- des cadres « Frame »,
- une barre de menu « Menu » et un bouton de menu « Menubutton »,

Chacun de ces widgets comporte des paramètres permettant de personnaliser leurs couleurs de l'arrière-plan ou du texte, leurs dimensions, la police de leurs caractères, ...

Pour organiser les widgets et les placer, on utilise une des méthodes suivantes :

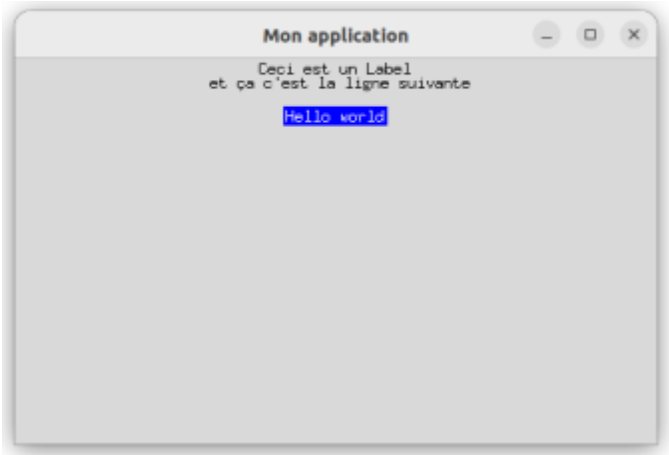
- pack() pour organiser et placer des widgets en blocs.
- grid() pour organiser et placer des widgets en étant comme des cases de tableau.
- place() pour organiser et placer des widgets dans un endroit précisé.

La fonction Label permet d'afficher un texte dans la fenêtre.

Script pour générer un label :

```
1   # L'importation de l'ensemble des éléments du paquet tkinter :
2   from tkinter import *
3
4   # Création d'une fenêtre avec la classe Tk :
5   fenetre = Tk()
6
7   # Ajout d'un titre à la fenêtre principale :
8   fenetre.title("Mon application")
9
10  # Définir les dimensions par défaut la fenêtre principale :
11  fenetre.geometry("500x300")
12
13  # Ajout d'un texte dans la fenêtre :
14  Label(fenetre, text = "Ceci est un Label\n et ça c'est la ligne suivante\n ").pack()
15
16  # Texte en blanc sur un font blue :
17  label = Label(fenetre, text="Hello world", foreground="white", background="blue").pack()
18
19  # Affichage de la fenêtre créée :
20  fenetre.mainloop()
```

et voici le resultat :



Remarque :

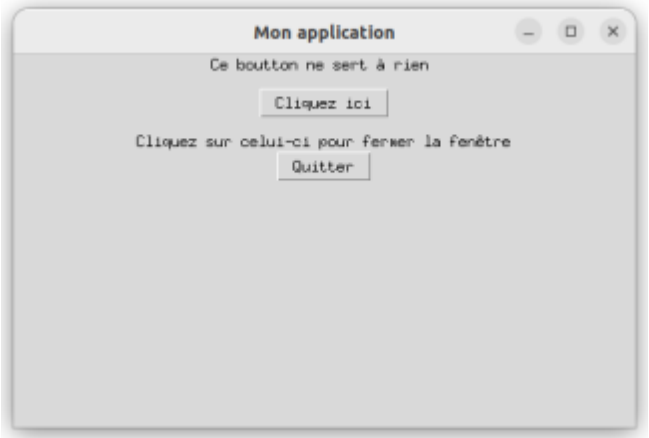
pack() permet d'empiler les widgets.

Le bouton peut servir à exécuter une action (par exemple fermer la fenêtre) ou appeler une fonction définie dans le programme.

Script pour générer un bouton :

```
1  # L'importation de l'ensemble des éléments du paquet tkinter :
2  from tkinter import *
3
4  # Création d'une fenêtre avec la classe Tk :
5  fenetre = Tk()
6  fenetre.title("Mon application")
7  fenetre.geometry("500x300")
8
9  # Ajout d'un texte dans la fenêtre :
10 Label(fenetre, text = "Ce boutton ne sert à rien \n ").pack()
11 # \n permet d'ajouter un saut de ligne.
12
13 # Ajout d'un bouton dans la fenêtre :
14 Button (fenetre, text = "Cliquez ici").pack()
15
16 # Texte en blanc sur un font blue :
17 Label(fenetre, text="\nCliquez sur celui-ci pour fermer la fenêtre").pack()
18
19 # Ajout d'un autre bouton dans la fenêtre, qui permet de fermer la fenêtre :
20 Button (fenetre, text = "Quitter", command=fenetre.destroy).pack()
21
22 # Affichage de la fenêtre créée :
23 fenetre.mainloop()
```

et voici le resultat :

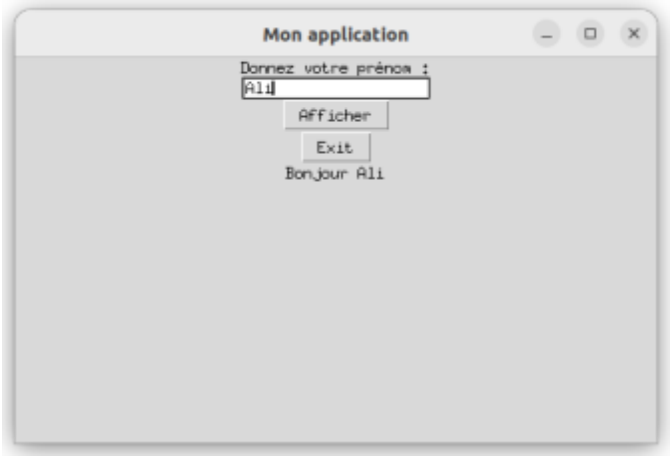


Le widget Entry permet d'afficher un champ de saisie.

Script pour générer un champ de saisie :

```
1  # L'importation de l'ensemble des éléments du paquet tkinter :
2  from tkinter import *
3
4  # Création d'une fenêtre avec la classe Tk :
5  fenetre = Tk()
6  fenetre.title("Mon application")
7  fenetre.geometry("500x300")
8
9  # Declaration du champs de saisie
10 myEntry = Entry(fenetre)
11
12 def afficher():
13     res = myEntry.get()
14     Label(fenetre, text = 'Bonjour '+res).pack()
15
16 Label(fenetre, text = "Donnez votre prénom :").pack()
17
18 # Affichage du champ de saisie
19 myEntry.pack()
20
21 # Lorsqu'on clique sur ce bouton, la fonction afficher est exécutée
22 Button(fenetre, text="Afficher", command=afficher).pack()
23
24 # Quitter :
25 Button(fenetre, text="Exit", command=fenetre.destroy).pack()
26
27 fenetre.mainloop()
```

et voici le resultat :



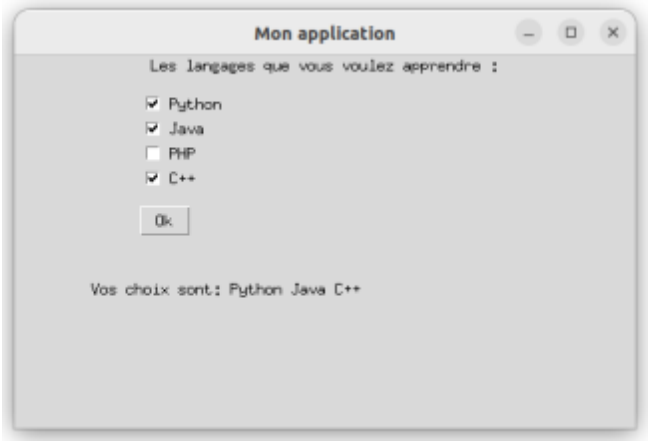
La fonction afficher est exécutée lorsqu'on clique sur le bouton afficher, elle récupère et affiche le contenu du champ de saisie.

Le widget Checkbutton permet d'afficher des options sous forme de cases à cocher, l'utilisateur peut en cocher une ou plusieurs.

Script pour générer des cases à cocher :

```
1   from tkinter import *
2
3   fenetre = Tk()
4   fenetre.title("Mon application")
5   fenetre.geometry("500x300")
6
7   var1 = IntVar()
8   var2 = IntVar()
9   var3 = IntVar()
10  var4 = IntVar()
11
12  # Definition de la fonction associé au clique du bouton
13  def afficher():
14      choix = ""
15      if var1.get() == 1: choix = choix+' Python'
16      if var2.get() == 1: choix = choix+' Java'
17      if var3.get() == 1: choix = choix+' PHP'
18      if var4.get() == 1: choix = choix+' C++'
19
20      Label(fenetre, text='Vos choix sont:'+choix).place(x=60,y=180)
21
22  # Donner un label au champ :
23  Label(fenetre, text="Les langages que vous voulez apprendre :").pack()
24
25  # Création des cases à cocher "Checkbutton" dans la fenêtre :
26  c1 = Checkbutton (fenetre, text = " Python", variable=var1).place(x=100,y=30)
27  c2 = Checkbutton (fenetre, text = " Java", variable=var2).place(x=100,y=50)
28  c3 = Checkbutton (fenetre, text = " PHP", variable=var3).place(x=100,y=70)
29  c4 = Checkbutton (fenetre, text = " C++", variable=var4).place(x=100,y=90)
30
31  bouton = Button (fenetre, text="Ok", command=afficher).place(x=100,y=120)
32
33  # Affichage de la fenêtre créée :
34  fenetre.mainloop()
```

et voici le resultat :

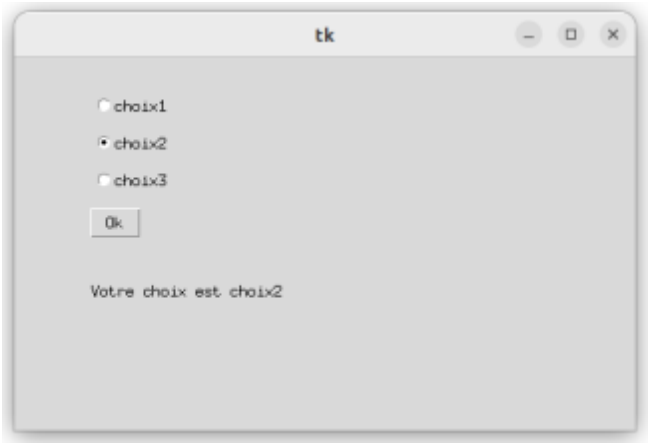


Le widget Radiobutton permet aussi d'afficher des options sous forme de cases à cocher, mais l'utilisateur ne peut en cocher qu'une seule.

Script pour générer des cases à cocher :

```
1   from tkinter import *
2
3   fenetre=Tk()
4   fenetre.geometry("500x300")
5
6   # Declaration de la variable associée aux Radiobuttons
7   var = IntVar()
8
9   # Definition de la fonction associé au clique du bouton
10  def afficher():
11      val = var.get()
12      Label(fenetre, text='Votre choix est choix'+str(val)).place(x=60,y=180)
13
14  # Création des cases à cocher "Radiobutton" dans la fenêtre :
15  choix1 = Radiobutton(fenetre, text="choix1", variable=var, value=1)
16  choix2 = Radiobutton(fenetre, text="choix2", variable=var, value=2)
17  choix3 = Radiobutton(fenetre, text="choix3", variable=var, value=3)
18
19  choix1.place(x=60,y=30)
20  choix2.place(x=60,y=60)
21  choix3.place(x=60,y=90)
22
23  # Création du bouton auquel on associe la fonction afficher :
24  bouton = Button (fenetre, text="Ok", command=afficher).place(x=60,y=120)
25
26  fenetre.mainloop()
```

et voici le resultat :



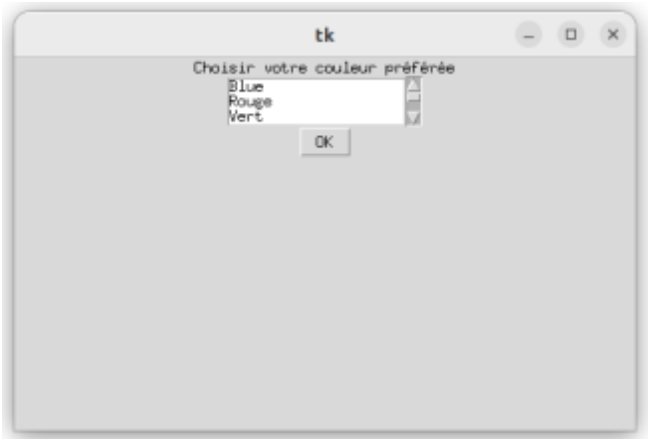
Le widget Listbox permet de créer des listes d'éléments, l'utilisateur peut en choisir un ou plusieurs éléments.

Le widget Scrollbar permet de créer une barre de défilement.

Script pour générer des menus :

```
1  from tkinter import *
2
3  gui = Tk()
4  gui.geometry("500x300")
5  Label(gui, text='Choisir votre couleur préférée').pack()
6
7  # Création d'un cadre frame
8  frame = Frame(gui)
9  # Création d'une barre de défilement qui va être palcée dans le cadre
10 scrollbar = Scrollbar(frame)
11 # Création d'une liste dans le cadre
12 liste = Listbox(frame, height=3, yscrollcommand = scrollbar.set )
13
14 # Définition de la fonction d'affichage
15 def afficher():
16     # Parcours des éléments séléctionnés
17     for i in liste.curselection():
18         Label(gui, text='Vous avez choisi le '+liste.get(i)).pack()
19
20 # Remplissage de la liste
21 liste.insert(1, "Blue")
22 liste.insert(2, "Rouge")
23 liste.insert(3, "Vert")
24 liste.insert(4, "Jaune")
25 liste.insert(5, "Orange")
26 liste.insert(6, "Rose")
27 # Affichage de la liste à gauche
28 liste.pack(side = LEFT, fill = BOTH )
29 # Affichage de la barre de défilement à droite
30 scrollbar.pack( side = RIGHT, fill = Y)
31 # Association de la barre de défilement à la liste
32 scrollbar.config(command = liste.yview )
33 frame.pack()
34 # Définition d'un boutton qu'on associe à la fonction affichage
35 Button(gui, text="OK", command=afficher).pack()
36 gui.mainloop()
```

et voici le resultat :



Le widget Menu permet de créer des menus de différents types. Ces menus serviront à exécuter des fonctions du programme.

Script pour générer des menus :

```
1   from tkinter import *
2
3   gui = Tk()
4   gui.geometry("500x300")
5
6   def hello():
7       print("hello world!")
8
9   def about():
10      Label(gui, text='Cette application ... ').place(x=50, y=50)
11
12  # créer un menu
13  menubar = Menu(gui)
14
15  # créer le sous-menu de File
16  filemenu = Menu(menubar, tearoff=0)
17  filemenu.add_command(label="New", command=hello)
18  filemenu.add_command(label="Open", command=hello)
19  filemenu.add_command(label="Save", command=hello)
20
21  # créer le sous-menu de Edit
22  editmenu = Menu(menubar, tearoff=0)
23  editmenu.add_command(label="Copy", command=hello)
24  editmenu.add_command(label="Paste", command=hello)
25
26  # Liaison des menus avec les sous-menus
27  menubar.add_cascade(label="File", menu=filemenu)
28  menubar.add_cascade(label="Edit", menu=editmenu)
29
30  # menus sans sous menu
31  menubar.add_command(label="About", command=about)
32  menubar.add_command(label="Quit!", command=gui.destroy)
33
34  # afficher le menu
35  gui.config(menu=menubar)
36  gui.mainloop()
```

et voici le resultat :

On peut utiliser l'une des trois méthodes pour positionner des widgets dans la fenêtre :

- pack
- grid
- place

On a déja vu, à travers les exemples précédents, l'utilisation de la fonction pack, qui ne fait qu'empiler, de haut en bas, les objets (widgets) dans l'ordre des appels de cette fonction.

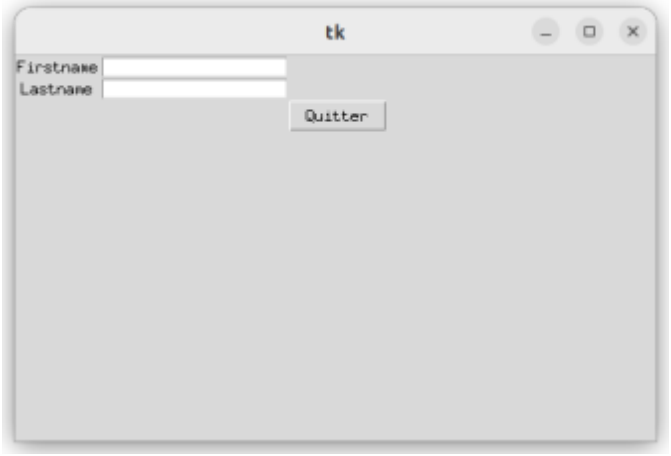
La méthode grid organise la fenêtre comme un tableau à deux dimension, on peut positionner un widget dans une cellule (case) de ce tableau en indiquant le numéro de la ligne et celui de la colonne.

ligne 0, colonne 0	ligne 0, colonne 1	ligne 0, colonne 2	...
ligne 1, colonne 0	ligne 1, colonne 1	ligne 1, colonne 2	...
ligne 2, colonne 0	ligne 2, colonne 1	ligne 2, colonne 2	...
...	...	...	...

Exemple :

```
1  from tkinter import *
2  gui = Tk()
3  gui.geometry("500x300")
4
5  # Affichage dans la cellule (0,0)
6  Label(gui, text="Firstname").grid(row=0, column=0)
7
8  # Affichage dans la cellule (1,0)
9  Label(gui, text="Lastname").grid(row=1, column=0)
10
11 # Affichage dans la cellule (0,1)
12 e1 = Entry(gui).grid(row=0, column=1)
13
14 # Affichage dans la cellule (0,1)
15 e2 = Entry(gui).grid(row=1, column=1)
16
17 # Affichage dans la cellule (2,2)
18 Button (gui, text = "Quitter", command=gui.destroy).grid(row=2, column=2)
19
20 gui.mainloop()
```

et voici le resultat :

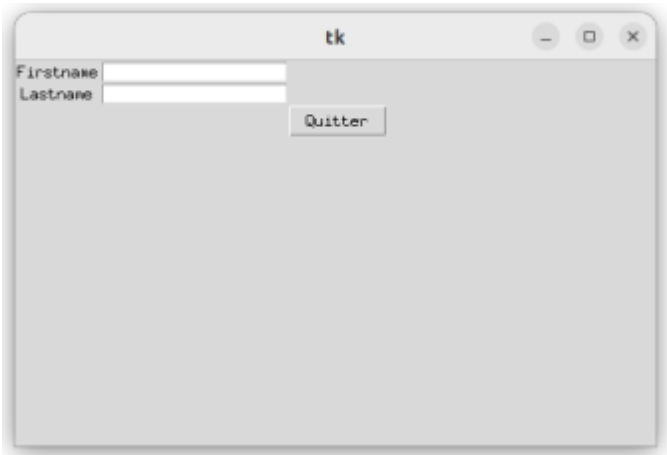


La méthode grid organise la fenêtre comme un tableau à deux dimension, on peut positionner un widget dans une cellule (case) de ce tableau en indiquant le numéro de la ligne et celui de la colonne.

Exemple :

```
1   from tkinter import *
2   gui = Tk()
3   gui.geometry("500x300")
4
5   # Affichage dans la cellule (0,0)
6   Label(gui, text="Firstname").grid(row=0, column=0)
7
8   # Affichage dans la cellule (1,0)
9   Label(gui, text="Lastname").grid(row=1, column=0)
10
11  # Affichage dans la cellule (0,1)
12  e1 = Entry(gui).grid(row=0, column=1)
13
14  # Affichage dans la cellule (0,1)
15  e2 = Entry(gui).grid(row=1, column=1)
16
17  # Affichage dans la cellule (2,2)
18  Button (gui, text = "Quitter", command=gui.destroy).grid(row=2, column=2)
19
20  gui.mainloop()
```

et voici le resultat :



Nous vous invitons à vous documenter sur les widgets :

- Text
- Message
- tkMessageBox
- LabelFrame
- Scale
- Spinbox
- ...

Exercice 1

Ecrire un programme python avec une interface graphique qui convertit la valeur de température du celsius au fahrenheit et réciproquement.

Exercice 1

Réaliser un calculatrice simple (addition, soustration, multiplication et division), l'utilisateur saisie deux nombres et clique sur le bouton correspondant à l'opération qu'il souhaite réaliser (+, -, x et /).

Liens utiles

Voici quelques liens pour des tutos sur python :

En anglais :

https://realpython.com/
https://www.pythontutorial.net/
https://waytolearnx.com/2020/06/tutoriels-python.html
https://www.w3schools.com/python/python_intro.asp

En Français :

https://docs.python.org/fr/3/
https://www.cours-gratuit.com/cours-python

En Vidéo :

APPRENDRE PYTHON (02 heures)
https://www.youtube.com/watch?v=oUJolR5bX6g

APPRENDRE PYTHON DE A à Z (07 heures)
https://www.youtube.com/watch?v=LamjAFnybo0