Loric Informatique

Python

Version 1.2 (créé le 10/10/2022, modifié le 14/01/2024)

Python est un langage de programmation de haut niveau qui vous permet de travailler plus rapidement et d'intégrer plus efficacement vos systèmes. Python est puissant... et rapide, fonctionne partout, est convivial et facile à apprendre.

Outils nécessaires :

- Python ou Miniconda
- IDE Pyzo
- En complément : Visual Studio Code (facultatif, mais utile pour coder plus facilement)

Table des matières :

I. Bases

I.1. Syntaxe

```
instruction1
instruction2
instruction3
```

I.2. Types de variables

Fonction pour connaître le type : `type(variable)`

I.2.1. Types de base

Type	Description	Fonction pour le convertir
`int`	Nombre entier	`int()`
`float`	Nombre décimal (ex : 0.1)	`float()`
`str`	Chaîne de caractère entre '' ou ""	`str()`

I.2.2. Autres types

Type	Description
`bool`	Valeur pouvant être True ou False
`list`	Liste de valeurs sous forme de [*val1*, *val2*]
`dict`	Dictionnaire sous forme de {*cle1*: *val1*, *cle2*: *val2*}
`NoneType`	Variable qui a pour valeur None (qui est vide)

`tuple`	Tuplet sous forme de (*val1*, *val2*)
`set`	Tuplet sous forme de {*val1*, *val2*} sans ordre d'importance

I.3. Commentaires

```
#Commentaire tenant sur une ligne

##Commentaire souligné selon le logiciel tenant sur une ligne

"""
Commentaire pouvant être sur une ou plusieurs lignes
"""
```

I.4. Opérations

Instruction	Description
`1 + 2`	Renvoie 3
`3 - 1`	Renvoie 2
`6 * 4`	Renvoie 24
`5 / 2`	Renoie 2.5
`5 // 2`	Renvoie 2 (le quotient sans décimal)
`5 % 2`	Renvoie 1 (le reste de la division)
`5 ** 2`	Renvoie 25 (5 à la puissance 2)

I.5. Les variables

Instruction	Description
a `= 5`	Affecte 5 à une variable
a, b `= 0.5, 60`	Affecte des valeurs à de multiples variables
variable `+= 1`	Ajoute 1
variable `-= 1`	Enlève 1

I.6. Les listes

Instruction	Description
`liste = []`	Crée une liste vide
`liste = ["Loric", "Informatique", 69]`	Crée une liste avec des valeurs
`liste[i]`	Renvoie la valeur à la position *i* (l'indice de la première valeur est 0)
`liste[a:b]`	Renvoie les valeurs de la position *a* comprise jusqu'à la position *b* non comprise
`liste[a:]`	Renvoie les valeurs de la position *a* comprise jusqu'à la fin
`liste[:b]`	Renvoie les valeurs du début jusqu'à la position *b* non comprise
`liste.append(valeur)`	Ajoute une valeur dans la liste
`liste.pop(i)`	Enlève et renvoie la valeur de la liste à la position *i*
`liste.pop()`	Enlève et renvoie la dernière valeur de la liste
`len(liste)`	Renvoie la longueur de la liste
`liste = [i for i in range(n)]`	Crée une liste de 0 à *n*-1
`nouvelle_liste = liste.copy()`	Crée une copie de la liste

I.7. Les dictionnaires

Instruction	Description
`dict={}`	Crée un dictionnaire vide
`dict={"a":"Loric", "b":2004}`	Crée un dictionnaire avec des valeurs
`dict["a"]`	Récupère la valeur contenue dans une clé
`dict["c"] = 3`	Ajoute 3 au dictionnaire

I.8. Conditions

Les conditions renvoient True si elle est respectée et False sinon

I.8.1. Opérateurs de comparaison

Condition	Description de ce que vérifie la condition
`a == b`	*a* égal à *b*
`a < b`	*a* strictement inférieur à *b*
`a > b`	*a* strictement supérieur à *b*
`a <= b`	*a* inférieur ou égal à *b*
`a >= b`	*a* supérieur ou égal à *b*
`a != b`	*a* n'est pas égal à *b*
`a in b`	*a* est présent dans *b* (qui peut être une liste)
`or`	À mettre entre deux conditions, permet d'avoir une des deux condtions qui doit être vraie
`and`	À mettre entre deux conditions, permet d'avoir deux conditions qui doivent être vraie
`not condition`	La condition doit être fausse

`a is None`
ou
`a is not None`
	Autre manière de tester si une variable est nulle ou non (*a* == None fonctionne également)

I.8.2. Tests de conditions

Instruction	Description
`if condition1:` `    instruction1`	Si *condition1* est vraie, alors on exécute *instruction1*

`if condition1:`
`    instruction1`
`else:`
`    instruction2`
	Si *condition1* est vraie, alors on exécute *instruction1*, sinon, on exécute *instruction2*

`if condition1:`
`    instruction1`
`elif condition2:`
`    instruction2`
`else:`
`    instruction3`
	Si *condition1* est vraie, alors on exécute *instruction1*, sinon, si *condition2* est vraie, on exécute *instruction2*, sinon, on exécute *instruction3*

try:
    instruction1
except:
    instruction2
	Si *instruction1* provoque une erreur, on exécute *instruction2*

I.9. Boucles

Instruction	Description
for *i* in range(*n*): *instruction1*	On répète *n* fois *instruction1* pour *i* allant de 0 à *n* non compris
for *i* in range(*d*, *f*): *instruction1*	On répète *f-d* fois *instruction1* pour *i* allant de *d* compris à *f* non compris
for *i* in range(*d*, *f*, *p*): *instruction1*	On répète (*f-d*)//*n* fois *instruction1* pour *i* allant de *d* compris à *f* non compris avec pour pas égal à *p*
for *elt* in *liste*: *instruction1*	On parcours la liste (ou une chaine de caractère) pour *elt* prenant toutes les valeurs de *liste*
while *condition*: *instruction1*	On répète jusqu'à ce que *condition* soit fausse
break	Permet de sortir d'une boucle sans la terminer (fortement déconseillé)
pass	Passer un bloc s'il est vide, pour éviter des erreurs

II. Les fonctions et les classes

II.1. Les fonctions

II.1.1. Créer une fonction

```
def maFonction(variable1, variable2…):
    instructions
```

II.1.2. Retourner une valeur de la fonction (non obligatoire)

```
return variable
```

ou

```
return variable1, variable2…
```

II.1.3. Faire un appel à la fonction

```
variable = maFonction(valeur1, valeur2…)
```

ou

```
variable1, variable2… = maFonction(valeur1, valeur2…)
```

ou (s'il n'y a pas de variable de retour)

```
maFonction(valeur1, valeur2…)
```

Remarque : Il est possible d'affecter une valeur par défaut si aucune valeur n'est fournie aux paramètres de la fonction sous la forme : `variable = valeurParDefaut`

II.2. Les classes

II.2.1. Créer une classe

```
class MaClasse:

    variables_statiques = valeurs

    def __init__(self, variable1, variable2…):
        self.variable1 = variable1
        self.variable2 = variable2
```

II.2.2. Définir la classe dans un objet

```
monObjet = MaClasse(valeur1, valeur2…)
```

II.2.3. Ajouter un affichage d'un message de la part de la classe (lors de l'exécution de print(*monObjet*))

```
def __str__(self):
    return "message"
```

II.2.4. Créer une méthode (dans une classe)

```
def maMethode(self, variable1, variable2…):
    instructions
```

II.2.5. Faire un appel d'une méthode autre que __init__ et __str__ ou d'une variable dans une classe

```
self.maMethode(valeur1, valeur2…)
```

```
self.variable1 = valeur1
```

II.2.6. Faire un appel d'une méthode autre que __init__ et __str__ ou d'une variable en dehors d'une classe

```
monObjet.maMethode(variables)
```

```
monObjet.variable1 = valeur1
```

III. Les instructions

III.1. Instructions de bases

Instruction	Description
`print(valeur)`	Affiche un texte et/ou une variable avec un retour à la ligne
`print("Valeur : ", valeur, end = " ")`	Affiche un texte et/ou une variable sans retour à la ligne
`variable = input("Entrer une valeur : ")`	Demande une valeur, renvoyée dans la variable
`type(variable)`	Récupérer le type d'une variable
`len(variable)`	Obtenir la longueur d'une chaîne de caractères ou d'une liste
`liste = variable.split(valeur)`	Couper une chaîne de caractères en précisant le séparateur (valeur) et renvoie une liste
`assert condition, message`	Vérifie que condition est vraie, sinon retourne un message d'erreur
`del variable`	Supprimer variable de la mémoire (non obligatoire)
`round(n, m)`	Arrondit *n* de *m* chiffres après la virgule (m facultatif)
`global variable`	Permet de rappeler une variable du programme principale dans la fonction concernée (à éviter si possible)
`help(fonction)`	Obtenir les informations écrites dans les """commentaires""" d'une fonction (fonctionne aussi avec des classes)
`exit(code)`	Fermer immédiatement le programme (facultatif) avec un code d'erreur facultatif

III.2. Les fichiers

Instruction	Description

`fichier = open("nom_du_fichier", "mode", encoding = "utf-8")`	Ouvre un fichier selon le mode choisi (r : Ouvre en lecture; w : Crée un nouveau fichier et l'ouvre en mode écriture (écrase l'ancien fichier du même nom))
`variable = fichier.read()`	Utilisable 1 fois, permet de lire en entier le fichier et de le stocker dans une variable
`variable = fichier.readline()`	Lit une ligne dans le fichier, puis la suivante si l'instruction est réemployée
`fichier.write(variable)`	Écrit le contenu de la variable dans le fichier (à la suite)
`fichier.close()`	Ferme et enregistre le fichier

IV. Les bibliothèques

IV.0. Importer une bibliothèque (ou un fichier python)

`from bibliotheque import *`

ou

`from bibliotheque import fonctions`

(ou `import bibliotheque` mais nécessite d'ajouter *bibliotheque*. devant chaques fonctions ou variables... importées).

> Si la bibliothèque est introuvable, faire `pip install bibliotheque` dans le shell pyzo ou dans le cmd

IV.1. random (`from random import *`)

ramdom permet de générer des nombres aléatoires facilement.

Instruction	Utilité
`random()`	Créer un nombre aléatoire entre 0 compris et 1 exclu
`randint(a, b)`	Créer un nombre entier aléatoire entre *a* et *b* compris
`seed(nombre)`	Initialiser le générateur pseudo-aléatoire pour reproduire des suites de nombres aléatoires identiques (avec un nombre quelconque pour repérer la suite de nombres aléatoires)

IV.2. turtle (`from turtle import *`)

turtle permet de créer des formes avec des tortues et de les faire déplacer dans une fenêtre graphique.

Instruction	Utilité
`t = Turtle()`	Créer une tortue
`t.forward(n)`	Avancer de *n* pixels
`t.backward(n)`	Reculer de *n* pixels
`t.right(a)`	Tourner de *a* degrés dans le sens horaire
`t.left(a)`	Tourner de *a* degrés dans le sens anti-horaire
`t.goto(x, y)`	Aller vers les coordonnées *x* et *y*
`t.setx(x)`	Aller vers les coordonnées *x* et *y* non changé
`t.sety(y)`	Aller vers les coordonnées *x* non changé et *y*
`t.setheading(a)`	Régler l'orientation à *a* degrés
`t.home()`	Réinitialiser les coordonnées à (0;0)
`t.circle(r)`	Dessiner un cercle de rayon *r* pixels
`t.speed(v)`	Régler la vitesse de la tortue à *v*, compris entre 0 et 10 inclus (0 : aucune animation ; 1 : lent ; 10 : rapide)
`t.pos()`	Renvoyer la position de la tortue sous la forme (x,y)
`t.xcor()`	Renvoyer la coordonnée x de la tortue
`t.ycor()`	Renvoyer la coordonnée y de la tortue
`t.heading()`	Renvoyer le degré d'inclinaison de la tortue
`t.up()`	Lever la pointe du stylo — pas de dessin quand il se déplace.
`t.down()`	Baisser la pointe du stylo — dessine quand il se déplace.
`t.width(n)`	Régler l'épaisseur du stylo à *n* pixels

`t.pencolor(couleur)`	Régler la couleur du stylo
`t.fillcolor(couleur)`	Régler la couleur de remplissage
`t.color(couleur)`	Régler la couleur de la tortue
`t.begin_fill()`	À appeler juste avant de dessiner une forme à remplir
`t.end_fill()`	Remplir la forme dessinée après le dernier appel à begin_fill()
`t.clear()`	Supprimer les dessins de la tortue de l'écran
`t.reset()`	Supprimer les dessins de la tortue de l'écran, recentrer la tortue et assigner les variables aux valeurs par défaut.
`t.write(txt, align='left', font = ('Arial', 8, 'normal'))`	Écrire le texte txt sur l'écran avec des paramètres facultatifs
`stamp_id = t.stamp()`	Créer un clone d'une tortue et l'enregistrer dans *stamp_id*
`t.clearstamp(stamp_id)`	Supprimer le clone *stamp_id* de la tortue
`t.clearstamp()`	Supprimer tous les clones de la tortue
`tbis = t.clone()`	Créer une copie de la tortue
`t.undo()`	Annuler la ou les dernières (si répété) actions de la tortue
`t.showturtle()`	Afficher la tortue
`t.hideturtle()`	Cacher la tortue
`t.shape(forme)`	Changer la forme de la tortue (qui peut être "arrow", "turtle", "circle", "square", "triangle" ou "classic")
`t.shape(fichier)`	Remplacer la tortue par une image
`t.onclick(fonction)`	Exécuter une fonction au clic de la tortue
`t.onscreenclick(fonction)`	Exécuter une fonction au clic de la souris dans la fenêtre

`t.onkey(fonction,'touche')`	Exécuter une fonction à l'appui sur une touche du clavier (valeur possible : 'Left', 'Right', 'Top', 'Bottom', 'space'...)
`t.onrelease(fonction)`	Exécuter une fonction au relâchement du clic de la tortue
`t.textinput(soustitre, message)`	Créer une fenêtre demandant l'entrée d'un texte
`t.numinput(soustitre, message)`	Créer une fenêtre demandant l'entrée d'un nombre
`t.mainloop()`	À mettre à la fin du programme, permet le maintien de la fenêtre turtle
`exitonclick()`	À mettre à la fin, permet le maintien de la fenêtre et la sortie au clic dans la fenêtre
`bye()`	Fermer la fenêtre turtle

IV.3. numpy (`from numpy import *`)

numpy permet de créer des tableaux de valeurs d'une taille définie et de pouvoir les afficher de façon lisible son contenu dans une console (fonctionne de la même manière qu'une liste, mais avec des fonctionnalités supprimées).

Installation : `pip install numpy`

Instruction	Utilité
`tableau = array([val1, val2, val3])`	Créer un tableau d'éléments
`tableau = zeros(n, type)`	Créer un tableau de *n* éléments rempli de 0 avec pour type int ou float
`tableau = empty(n, type)`	Créer un tableau non initialisé de *n* éléments (avec des valeurs aléatoires) (fortement déconseillé)
`pi`	Obtenir la valeur de π
`tableau = arange(d, f, p)`	Créer une liste avec des valeurs de *d* à *f* non compris avec pour pas *p*

IV.4. math (`from math import *`)

math permet d'utiliser les fonctions de base de mathématiques.

Instruction	Utilité
`pi`	Obtenir la valeur de π
`sqrt(`*`nombre`*`)`	Utiliser la racine carrée
`log(`*`nombre`*`)` ou `log(`*`nombre, base`*`)`	Utiliser la fonction logarithme
`exp(`*`nombre`*`)`	Utiliser la fonction exponentielle
`degrees(`*`nombre`*`)`	Convertir des radians en degrés
`radians(`*`nombre`*`)`	Convertir des degrés en radians
`cos(`*`radians`*`)`	Utiliser la fonction cosinus
`sin(`*`radians`*`)`	Utiliser la fonction sinus
`tan(`*`radians`*`)`	Utiliser la fonction tangente
`acos(`*`nombre`*`)`	Utiliser la fonction cosinus^{-1}
`asin(`*`nombre`*`)`	Utiliser la fonction sinus^{-1}
`atan(`*`nombre`*`)`	Utiliser la fonction tangente^{-1}

IV.6. time (`from time import *`)

time permet de manipuler le temps.

Instruction	Utilité
`time()`	Retourner l'heure du système en secondes (la différence entre l'exécution de la fonction au début et à la fin peut donner le temps d'exécution du programme)
`time_ns()`	Retourner l'heure du système en nanosecondes (pour Python 3.7 ou supérieur)
`ctime()`	Retourner la date et l'heure sous la forme 'Tue Dec 10 16:07:12 2024'
`process_time()`	Retourner le temps CPU d'exécution du programme en secondes (sans prendre en compte les autres programmes)

`process_time_ns()`	Retourner le temps CPU d'exécution du programme en nanosecondes (pour Python 3.7 ou supérieur)
`sleep(`*`secondes`*`)`	Mettre en pause le programme temporairement

IV.10. tkinter (`from tkinter import Tk, Canvas`)

tkinter permet d'interagir avec une interface graphique.

Instruction	Utilité
`fenetre` `= Tk()`	Créer une fenêtre tkinter
`fenetre``.title("`*`nom de la fenêtre`*`")`	Changer le nom de la fenêtre (en haut à gauche)
`fenetre``.iconbitmap("`*`icone`*`.ico")`	Changer l'icône de l'application (en haut à gauche)
`fenetre``.after(`*`n, fonction`*`)`	Exécuter une fonction après *n* ms (utile pour faire des animations sans bloquer la fenêtre) (*fonction* peut être remplacé par lambda:*fonction*() afin d'ajouter des arguments dans la fonction)
`fenetre``.geometry('500x500+200+100')`	Redimensionner la fenêtre en 500x500 et placer la fenêtre à x = 200 et y = 100
`x` `=` *`fenetre`*`.winfo_rootx()`	Renvoie la position du bord haut de la fenêtre (sans prendre en compte le bandeau avec le soustitre et les boutons)
`y` `=` *`fenetre`*`.winfo_rooty()`	Renvoie la position du bord gauche de la fenêtre
`fenetre_bis` `= Toplevel(`*`fenetre`*`)`	Créer une fenêtre secondaire Tkinter (ne nécessite pas de mainloop pour maintenir cette fenêtre)
`menu` `= Menu(`*`fenetre`*`)`	Créer une barre de menus (situé en haut de la fenêtre)
`menu_A` `= Menu(`*`fenetre`*`)`	Créer un menu, qui sera placé dans la barre de menus
`menu_A``.add_command(label = "`*`Nom de la commande`*`", command =` *`fonction`*`)`	Ajouter une commande dans un menu
`menu``.add_cascade(label = "`*`nom_menu`*`", menu =` *`menu_A`*`)`	Ajouter le menu dans la barre de menus

menu.add_cascade(label = "*nom_menu*", command = *fonction*)	Ajouter une commande dans la barre de menus
fenetre["menu"] = *menu*	Appliquer le menu dans la fenêtre tkinter
zg = Canvas(*fenetre*, bg = *couleur*, width = *longueur*, height = *hauteur*, *parametres*) *zg*.grid(row = 0, column = 0, rowspan = *nblignes*, colomnspan = *nbcolonnes*)	Créer une zone graphique dans la fenêtre tkinter et le positionner avec un nombre de ligne ou de colonnes ou les deux (rowspan et colomnspan sont facultatifs). Les différents paramètres sont : bg = *couleur*, width = *longueur*, height = *hauteur*, anchor = *ancre* (avec *ancre* = NW, N, NE, E, SE, S, SW, W ou center), outline = *couleur_bordure*)
widget1 = Label(*fenetre*, text = "*message*", *parametres*)	Créer une zone de texte (paramètres : bg, fg, outline, width, height)
widget2 = Entry(*fenetre*, textvariable = StringVar(), *parametres*)	Créer une zone de saisie (paramètres : bg, fg, outline, width, height)
texte_saisi = *widget2*.get()	Récupérer le texte écrit dans la zone de saisie
widget3 = Button(*fenetre*, text = '*Texte*', *parametres*, command = *fonction*)	Créer un bouton (paramètres : bg, fg, outline, width, height)
widget.grid(row = *numligne*, column = *numcolonne*)	Placer un widget avec la méthode de positionnement dans une case (dépend de rowspan ou colomnspan ou les deux)
widget.place(x = *abscisse*, y = *ordonnee*)	Placer un widget avec la méthode de positionnement avec les coordonnées (à partir du bord de la zone graphique en haut à gauche)
widget.configure(*parametres*)	Modifier les paramètres d'un widget
widget.destroy()	Supprimer un widget
item1 = *zg*.create_line(*x_deb*, *y_deb*, *x_fin*, *y_fin*, width = *epaisseur*, fill = *couleur*)	Créer une ligne (paramètres : width, fill)
item2 = *zg*.create_rectangle(*x_deb*, *y_deb*, *x_fin*, *y_fin*, width = *epaisseur*, fill = *couleur*)	Créer un rectangle (paramètres : width, fill, outline)
item3 = *zg*.create_oval(*x_deb*, *y_deb*, *x_fin*, *y_fin*, width = *epaisseur*, fill = *couleur*)	Créer une ellipse (paramètres : width, fill, outline)
item4 = *zg*.create_text(*x*, *y*, text = "*message*", font = '*Police taille*')	Créer un texte avec un fond transparent (paramètres : fill, font, anchor)

fichier = ImageTk.Image.open("*nom_du_fichier*")	Ouvrir une image en format png ou jpg dans Python (étape 1 de l'insertion d'une image dans la fenêtre) (nécessite pillow)
fichier.resize((*x*, *y*), ImageTk.Image.ANTIALIAS)	Redimensionner la taille de l'image
image_tk = ImageTk.PhotoImage(image = *fichier*)	Ouvrir une image dans tkinter (étape 2 de l'insertion d'une image dans la fenêtre) (nécessite pillow)
item5 = *zg*.create_image(*x*, *y*, image = *image_tk*)	Insérer une image (dernière étape de l'insertion d'une image dans la fenêtre)
variable = *zg*.itemcget(*item*, '*parametre*')	Récupérer la valeur d'un paramètre pour un item déjà créé
zg.itemconfigure(*item*, *parametres*)	Modifier un ou plusieurs paramètres d'un item déjà créé
zg.delete(*item*)	Supprimer un item
zg.delete('all')	Supprimer tous les items
x, *y* = *zg*.coords(*item*)	Récupérer les coordonnées d'un item ayant en argument 2 coordonnées
x_deb, *y_deb*, *x_fin*, *y_fin* = *zg*.coords(*item*)	Récupérer les coordonnées d'un item ayant en argument 4 coordonnées
zg.coords(*item*, *x*, *y*)	Modifier les coordonnées d'un item ayant en argument 2 coordonnées
zg.coords(*item*, *x_deb*, *y_deb*, *x_fin*, *y_fin*)	Modifier les coordonnées d'un item ayant en argument 4 coordonnées
zg.bind('‹*evenement*›', *fonction*)	Créer un événement lié à une action sur la souris (la fonction doit obligatoirement contenir en paramètre event, et les coordonnées de la souris peuvent être récupérée avec event.x et event.y) (événements possibles : ‹ButtonPress-1› (Appui sur le bouton gauche), ‹ButtonRelease-2› (Relâchement du bouton gauche), ‹Double-Button-1› (Double clic sur le bouton gauche), ‹Motion› (Déplacement de la souris), ‹B1-Motion› (Déplacement avec bouton gauche appuyé), ‹Enter› (Entrée dans le widget de la souris), ‹Leave› (Sortie du widget de la souris))

fenetre.bind('‹*evenement*›', *fonction*)	Créer un événement lié à une action sur la souris (la fonction doit obligatoirement contenir en paramètre event) (événements possibles : ‹KeyPress› (Appui sur une touche quelconque), ‹KeyRelease› (Relâchement d'une touche quelconque), ‹a›, ‹A›, ‹1›, ‹2›… , ‹Right›, ‹Down›, ‹Up›, ‹Left›, ‹Alt›, ‹Shift›, ‹Control›… (les touches peuvent être combinées sous la forme : ‹Control-Up›))
fenetre.mainloop()	À mettre à la fin du programme, permet le maintien de la fenêtre tkinter