

## Sommaire

<b>Installation.....</b>	<b>1</b>
<b>Présentation d'un IDE.....</b>	<b>2</b>
<b>Importer des fonctions de modules.....</b>	<b>2</b>
<b>Affecter des valeurs à des variables.....</b>	<b>2</b>
<b>Récupérer une frappe au clavier.....</b>	<b>3</b>
<b>Opérations arithmétiques.....</b>	<b>3</b>
<b>Valeurs aléatoires.....</b>	<b>3</b>
<b>Afficher un message ou une valeur.....</b>	<b>3</b>
<b>Chaînes de caractère.....</b>	<b>4</b>
Concaténation de chaînes de caractères.....	4
<b>Graphiques avec Turtle.....</b>	<b>4</b>
<b>Instructions conditionnelles : if.....</b>	<b>5</b>
Structure.....	5
or, and et not.....	5
<b>Définir une fonction.....</b>	<b>5</b>
Fonction à plusieurs variables.....	5
<b>Boucle for (pour) : répéter une instruction un certain nombre de fois.....</b>	<b>6</b>
i est pris dans une liste.....	6
i est pris dans un intervalle.....	6
<b>Boucle while (tant que) : répéter tant qu'une condition reste vraie.....</b>	<b>6</b>
Et pour boucler tant qu'une condition est fausse ?.....	6
<b>Gestion de listes.....</b>	<b>7</b>
Récupérer un élément ou une partie de liste.....	7
Modifier une liste.....	7
Récupérer des informations sur une liste.....	7
<b>Graphiques avec Pylab.....</b>	<b>8</b>
Tracé de nuages de points ou de courbes.....	8

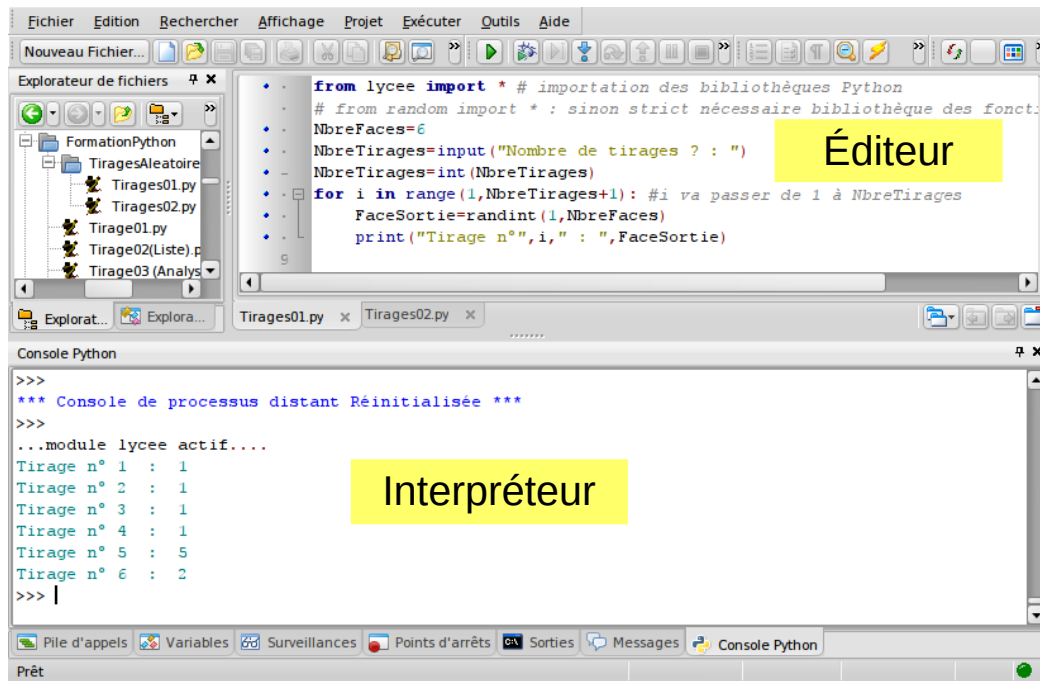
## Installation

Pour programmer en Python, on a besoin d'avoir Python installé sur sa machine, et un éditeur de textes.

On peut aussi installer un IDE (*Integrated Development Environment* ou environnement de développement intégré) tel que IDLE, PyScripter, Thonny ou EduPython.

La distribution EduPython est fournie avec un module nommé *Lycee* contenant lui-même de nombreux modules mathématiques, graphiques ou autres. Ce module *Lycee* peut être systématiquement appelé dans tous les programmes, ce qui évite de faire appel aux modules spécifiques nécessaires.

## Présentation d'un IDE



**Interpréteur** : affiche ">>>" : Pour tester en direct des commandes Python

**Editeur** : pour créer une liste de commandes

## Importer des fonctions de modules

*Ces instructions doivent apparaître en début de programme si on a besoin dans la suite du programme de fonctions particulières*

`from lycee import *` : importe toutes les fonctions du module lycee (dans EduPython)

`from math import *` : importe toutes les fonctions du module math

`from math import sqrt, cos` : importe les fonctions sqrt et cos du module math

`import math` : importe toutes les fonctions du module maths mais nécessite de des appeler sous la forme `math.fonction`

module math : documentation : <https://docs.python.org/3/library/math.html>

## Affecter des valeurs à des variables

`a ← quinze : a = "quinze"`

`b ← 5 × 3 : b = 5*3`

affectations à plusieurs variables

`a,b = "quinze", 5*3`

## Récupérer une frappe au clavier

---

```
Texte = input("votre texte ?")
```

```
Entier = int(input("votre nombre entier ?"))
```

```
Reel = float(input("votre nombre réel ?"))
```

## Opérations arithmétiques

---

addition	+	2 + 5 donne 7
soustraction	-	8 - 2 donne 6
multiplication	*	6 * 7 donne 42
puissance	**	5 ** 3 donne $5^3 = 125$
division	/	7 / 2 donne 3.5
reste de division entière	%	7 % 3 donne 1
quotient de division entière	//	7 // 3 donne 2

## Valeurs aléatoires

---

Nécessite le module random

```
from random import *
```

(voir <https://python-simple.com/python-modules-math/random.php> )

`random.choice(["a", "b", "c"])` : donne aléatoirement "a", "b" ou "c"

`randint(0, 3)` : donne un entier entre 0 et 3 inclus

`random()` : donne une valeur décimale de l'intervalle [0 ; 1[

## Afficher un message ou une valeur

---

```
a = 3
signe="multiplié par"
b = 5
egal="égale"
print(a, signe, b, egal, a*b)
```

Ce programme affiche : 3 multiplié par 5 égale 15

Pour ne pas passer à la ligne entre deux instructions print :

```
print (a, signe, b, egal, end=" ")
print(a*b)
```

Pour sauter une ligne dans une instruction print :

```
print(a, signe, b, egal, "\n", a*b)
```

## Chaînes de caractère

---

Le type de données non modifiable `str` représente une chaîne de caractère.

On délimite une telle suite de caractères soit par des apostrophes, soit par des guillemets.

```
phrase = "La patate est chaude"
```

```
print(len(phrase))
```

*affiche le nombre de caractères : 20*

```
phrase2 = phrase.replace("a", "o")
```

*va remplacer a par o → "Lo potote est choude"*

```
ListeMots = phrase.split()
```

*crée la liste des mots séparés par espace : ["La","patate","est","chaude"]*

```
phrase.index("a")
```

*renvoie la position de la lettre a (ici : 1 car la première lettre est numérotée à 0)*

### Concaténation de chaînes de caractères

La concaténation de chaînes de caractères consiste à mettre bout à bout des morceaux pour créer des chaînes de caractères complexes. On assemble les morceaux utilisant le signe "+". Le saut de ligne se code "\n"

Les nombres doivent être transformés en chaîne de caractère avec la fonction `str()` pour pouvoir être assemblés avec les chaînes de caractères.

Exemple : dans cet exemple, C est un nombre réel

```
Question="Le côté recherché est-il plus grand que "
```

```
Question=Question+"+str(C)+" ?\n 0 pour Oui, N pour Non :"
```

L'affichage de `Question` va donner :

```
Le côté recherché est-il plus grand que 15.0 ?
```

```
0 pour Oui, N pour Non :
```

## Graphiques avec Turtle

---

Importer le module : `from turtle import *`

Turtle est un module permettant de dessiner des formes graphiques :

`goto(x,y)` : va aux coordonnées (x ; y). *L'origine du repère est le centre de la fenêtre*

`forward(20)` ou `fd(20)` ; `backward(20)` ou `bd(20)` : avance ; recule de 20 pixels

`right(60)` ou `rt(60)` ; `left(60)` ou `lt(60)` : tourne à gauche ou à droite de 60°

`setheading(n)` : oriente la tortue (sens trigonométrique) : Est (0), Nord (90)

`print(heading())` : affiche l'orientation de la tortue

`penup()` ou `up()` ; `pendown()` ou `down()` : abaisse, lève le crayon

`showturtle()` ; `hideturtle()` : montre ; cache la tortue

`speed(n)` : change la vitesse de lent (1) à rapide (10) ; ou même super rapide (0)

`pensize(n)` : épaisseur du crayon en pixel

`circle(R)` : trace un cercle dans le sens trigonométrique

`circle(R,A, steps=S)` : trace un arc de rayon R, d'angle au centre A° et à S côtés

`circle(R,180)` : trace un demi-cercle ; `circle(R, steps = 8)` trace un octogone

## Instructions conditionnelles : if

---

La ligne if se termine par " : ", les instructions qui en dépendent sont indentées

Test d'égalité : on utilise `==`

Test d'inégalité : on utilise `!=` (non égal)

Autres opérateurs de comparaison : `<` ; `<=` ; `>` ; `>=`

Le résultat d'un test est `True` ou `False`

### Structure

```
if a == 3:
    instruction
else :
    instruction sinon
suite du programme
```

### or, and et not

Pour un dé à 6 faces les tests suivants sont équivalents :

```
if face==1 or face==3 or face==5:
if face!=2 and face!=4 and face!=6:
if not(face==2 or face==4 or face==6):
```

## Définir une fonction

---

La ligne def se termine par " : ", les instructions qui en dépendent sont indentées

```
# définition de la fonction calculant d en fonction de t
def DistanceChute(temps):
    d = 9.81/2*temps**2
    return d

# affichage de la distance parcourue en 5 s de chute libre
print("En 5 s de chute, on parcourt", DistanceChute(5),"m.")
```

### Fonction à plusieurs variables

```
# définition de fonction calculant D et V à partir de a et t
def DistanceVitesse(acceleration, temps):
    d = acceleration/2*temps**2
    v = acceleration*temps
    return d, v

# récupération simultanée de la distance et de la vitesse
Distance, Vitesse = DistanceVitesse(9.81, 5)
```

Remarques :

Une fonction peut ne pas renvoyer (return) de valeurs : elle sert alors à exécuter une routine (affichage ou autre ...)

Une fonction peut fonctionner sans lui passer de paramètre : dans ce cas, on la définit en ne mettant rien entre parenthèse : `def FonctionSansParametre()`

## Boucle for (pour) : répéter une instruction un certain nombre de fois

---

Boucle for : la ligne for se termine par " : " , les instructions de la boucle sont indentées

### i est pris dans une liste

```
for i in [1,2,3,5,7,9,11]:  
    print(i)  
    print (i**2)
```

pour chacune des valeurs de la liste, i prend les valeurs successives de la liste, on affiche i, on affiche son carré, puis une fois tous les nombres de la liste passés, on poursuit le programme

### i est pris dans un intervalle

range(b) : le nombre prendra les valeurs dans [0 ; b[ (b exclu de l'intervalle)

for i in range(5) : i prendra pour valeurs : 0 ; 1 ; 2 ; 3 et 4

range(a,b) : le nombre prendra les valeurs dans [a ; b[ (b exclu de l'intervalle)

for i in range(2,5) : i prendra pour valeurs : 2 ; 3 et 4

range(a,b,c) : le nombre prendra les valeurs de [a ; b[ par pas de c

for i in range(5,11,2) : i prendra les valeurs : 5 ; 7 ; 9

## Boucle while (tant que) : répéter tant qu'une condition reste vraie

---

Boucle while : la ligne for se termine par " : " , les instructions de la boucle sont indentées

```
i = 1  
while i <= 10 :  
    print(i)  
    i = i + 1  
suite du programme
```

Tant que  $i \leq 10$  on affiche i, on l'augmente de 1 et on recommence.

La suite du programme s'effectue quand  $i = 11$

La boucle tant que doit contenir une instruction susceptible de modifier sa condition, sinon cette boucle est infinie.

### Et pour boucler tant qu'une condition est fausse ?

Dans ce cas, on peut utiliser **not(condition)**

Exemple pour obliger l'utilisateur à répondre par oui ou par non :

Reponse="A" *#On initialise pour entrer dans la boucle*

while **not(Reponse in "OoNn")** :

Reponse = input("Répondre O pour Oui, N pour Non")

suite du programme

## Gestion de listes

---

( [http://python.lycee.free.fr/manipulations\\_listes\\_texte.html](http://python.lycee.free.fr/manipulations_listes_texte.html) )

Une liste se définit entre crochets.

```
maliste=[1, "deux", 2+1,"quatre",5, 6, 7,"huit"]
```

### Récupérer un élément ou une partie de liste

<code>maliste[1]</code>	renvoie "deux"	( $\Delta$ la numérotation dans la liste commence à 0)
<code>Maliste[1:2]</code>	renvoie la liste ["deux"]	(liste de l'élément 1 à 2 exclu)
<code>maliste[1:4]</code>	renvoie la liste ["deux", 3, "quatre"]	(liste de l'élément 1 à 4 exclu)
<code>maliste[0:7:2]</code>	renvoie [1, 3, 5, 7]	(de l'élément 0 à 7 exclu, tous les 2 éléments)
<code>maliste[3:]</code>	renvoie ["quatre", 5,6 ,7, "huit"]	(liste de l'élément 3 au dernier)
<code>maliste[:3]</code>	renvoie [1, "deux", 3]	(liste du début à l'élément 3 exclu)
<code>maliste[-1]</code>	renvoie "huit"	Le dernier élément
<code>maliste[-2]</code>	renvoie 7	L'avant dernier élément

### Modifier une liste

Tri de liste (sur éléments de même type): `maliste.sort()`

Tri décroissant : `maliste.sort(reverse=True)`

Ajouter un élément à la fin de la liste : `maliste.append(Élément)`

Ajouter les éléments de `maliste2` à `maliste` : `maliste.extend(maliste2)`

Insérer l'élément A devant le numéro i : `maliste.insert(i,A)`

Remplacer l'élément n°i par A : `maliste[i] = A`

Remplacer plusieurs éléments : `maliste[i:j]=[A, B, C]`

Remplacer la fin de la liste à partir de l'élément n°i : `maliste[i:]=[A,B,C]`

Supprimer l'élément numéro i : `maliste[i:i+1]=[]`

Supprimer le dernier élément : `maliste[len(maliste)-1:]=[]`

Mélanger une liste (nécessite le module random) : `shuffle(maliste)`

### Récupérer des informations sur une liste

Longueur de liste : `len(maliste)` renvoie 4

Rechercher si un élément est dans la liste : `7 in maliste` renvoie true ou false

Rechercher la position d'un élément dans la liste : `maliste.index(Element)`

Rechercher le nombre d'occurrences d'un élément : `maliste.count(Element)`

Rechercher le maximum d'une liste : `max(maliste)`

*Ces plupart de ces fonctions fonctionnent dans une chaîne de texte.*

## Graphiques avec PyLab

Importer le module : `from pylab import *`

Afficher le graphique : **à placer en dernier après les instructions de traçage** : `show()`

Régler les axes : `axis([xmin, xmax, ymin, ymax])`

Afficher la grille : `grid()`

Créer un point croix rouge: `plot(x, y, "rx")`

Le code règle la couleur, le tracé de la courbe ou des points.

Liste des codes →

Couleur		Style	
b	bleu	-	ligne continue
g	vert	--	tirets
r	rouge	:	pointillés
c	cyan	.	des points
m	magenta	o	des billes
y	jaune	x	des croix
k	noir	v	des triangles
w	blanc	-.	points-tirets

Titre de graphique :

`title("Hauteur en fonction du temps")`

Libellé abscisses : `xlabel("Temps (s)")`

Libellé ordonnées : `ylabel("Distance (m)")`

### Tracé de nuages de points ou de courbes

La commande plot accepte des listes de valeurs :

`plot([0,1,2,3,4],[0,1,4,9,16],"-g")` : Crée une série de segments verts

Définir des tableau de valeurs :

`x = arange(0, 5, 0.1)` : crée une liste de 50 nombres dans  $[0 ; 5[$  par pas de 0,1

`x = linspace(0, 5, 51)` : crée une liste de 51 nombres dans  $[0 ; 5]$

`y = -5 * x ** 2 + 20 * x + 10` :

crée une liste calculée sur chacun des éléments de x

### Exemple :

```
from pylab import *
title("Hauteur en fonction du temps")
xlabel("Temps (s)")
ylabel("Hauteur(m)")
x = linspace(0, 5, 51)
y = -5 * x ** 2 + 20 * x + 10
plot(x, y, "k-")
grid()
show()
```

