Le langage de programmation Python permet d'interagir avec la machine à l'aide d'un programme appelé l'interprète Python. On peut utiliser l'interprète Python de deux manières différentes : le mode interactif consiste à dialoguer instruction par instruction avec l'interpréteur, le mode programme consiste à écrire dans un fichier une séquence d'instructions pour les faire exécuter d'un coup par l'interpréteur.

Arnaquer les ordinateurs du lycée pour utiliser IDLE : Sur les ordinateurs du lycée, on utilisera l'environnement IDLE pour travailler en Python. Pour lancer IDLE, créer un fichier texte dans un dossier de votre choix :

```
clic droit > Nouveau > Document texte > "nom quelconque.txt"
```

puis changer l'extension du fichier en `.py` :

```
clic droit sur le fichier > Renommer > "nom quelconque.py"
```

On peut maintenant lancer IDLE :

```
clic droit sur le fichier > Edit with IDLE > edit with IDLE 3.5 (32 bits)
```

La fenêtre ouverte représente votre fichier Python. Pour l'instant, nous n'allons pas y toucher et essayer le mode interactif. Appuyer sur F5.

1 Mode interactif

Nous sommes maintenant dans le mode interactif de Python. On peut voir ce mode comme une calculatrice : on donne des instructions à l'interpréteur et celui-ci les exécute. Les trois `> > >` indiquent que nous avons la main et que Python attend une instruction.

1.1 Arithmétique

Essayer les commandes suivantes (appuyer sur `ENTREE` pour lancer le calcul) :

- `1+2`
- `6**2`
- `7*6`
- `7//2`
- `10.2-23`
- `2**8`
- `7/2`
- `7%2`

On voit que dans le mode interactif, Python exécute les calculs demandés et affiche les résultats. On a accès à différentes opérations mathématiques :

Opérations arithmétiques en Python :
- Addition $a+b$: `a+b`
- Soustraction $a-b$: `a-b`
- Multiplication $a \times b$: `a*b`
- Puissance a^b : `a**b`
- Division $\frac{a}{b}$: `a/b`
- Quotient de la division euclidienne de a par b : `a//b`
- Reste de la division euclidienne de a par b : `a%b`

On peut bien sur combiner ces opérations et écrire par exemple :

```
((2+3)*7/3+5)**2
```

La priorité des calculs est usuelle (parenthèses > puissances > multiplications et divisions > additions et soustractions). Essayer les commandes suivantes :
- `1+*3`
- `1+)(2+3`
- `7/0`
- `2+" "`

Avec ces commandes, Python détecte des erreurs. Dans les deux premières, l'opération n'est pas bien posée et Python ne comprend pas ce qui est demandé : on parle d'erreur de syntaxe. La troisième erreur correspond à une valeur interdite (on ne peut pas diviser par 0). La quatrième est une erreur de type car le + suppose que `a` et `b` sont des entiers et on a ici `b=" "` un texte.

Attention ! Les nombres à virgule se représentent avec un point et non une virgule, la virgule sert en général en Python à séparer des éléments, ainsi `1.2` représente le nombre réel $1,2$ alors que `1,2` représente le couple d'entier $(1;2)$.

1.2 Variables

On peut voir les variables Python comme des boites mémorisant une certaine valeur. Elles permettent notamment de mémoriser un résultat en lui donnant un nom. On peut ainsi réutiliser ce résultat plus tard. L'ensemble des variables et leur valeurs respectives est appelé l'état de l'interprète. Cet état peut changer à chaque instruction exécutée. Essayer les commandes suivantes :

- `a = 3+4`
- `b = 5+1`
- `a*b`

On voit que les deux premières commandes n'affichent rien. Cela se produit car l'interpréteur affiche les calculs, or `a=3+4` et `b=5+1` sont des affectations (des remplissage de variables).

Variables : Une variable a un nom et une valeur. Pour déclarer une variable en python on utilise la syntaxe :

<nom de la variable> = <valeur ou calcul>

Une fois une variable déclarée, on peut l'utiliser dans un calcul en utilisant sont nom. L'interpréteur Python remplacera alors la variable par sa valeur actuelle.

Attention : l'ordre est primordial, on ne peut pas écrire `3=a`.

Il est possible de modifier la valeur d'une variable déclarée. Essayer les commandes suivantes :

- `a = 8`
- `a = 12`
- `a`

Ici, on a écrasé la valeur `8` présente dans `a` pour la remplacer par la valeur `12`. Tester les commandes suivantes :

- `a = 2`
- `a = a+3`
- `a`
- `a = a*4`
- `a`

Ici, la deuxième commande a ajouté 3 à la valeur (déjà renseignée) de `a`. Analysons son exécution :

1. `a = ...` : la valeur de `a` va être définie, il faut effectuer le calcul `...`.
2. `a+3` : pour l'instant `a` contient la valeur `2` donc `a+3` donne 5.
3. L'affectation redéfinit la valeur de `a` à `5`.

On dispose de raccourcis pour effectuer ce genre de modifications. Essayer les commandes suivantes :

- `a=2`
- `a+=3`
- `a`
- `a*=4`
- `a`

Nommer une variable : Le nom d'une variable peut contenir n'importe quelle lettre (minuscule ou majuscule), n'importe quel chiffre et des underscores (_) cependant Python fixe certaines règles :

- il ne peut pas commencer par un chiffre ;
- il ne peut contenir aucun autre caractère (`+ , - espace . > <` etc.)

De plus, il existe un certain nombre de conventions / de bonnes habitudes à prendre :

- donner des noms ayant du sens dans le contexte du programme (`longueur, aire, points` plutôt que `a,b,c`) ;
- en cas de mots multiples, séparer les mots par un `_` ou avec des majuscules après le premier mot (`nombre_joueurs` ou `nombreJoueurs` plutôt que `nombrejoueurs`) ;
- ne pas commencer par une majuscule (sauf pour les classes, cf cours de terminale)
- pour les nombres sans contexte particulier :
  - `i,j,k` pour des indices (des nombres entiers qui vont varier souvent de 1 en 1) ;
  - `n,m` pour des nombres entiers positifs constants ;
  - `x,y,z` pour des nombres réels.
  - `a,b,c` : pas de règle particulière.

2 Mode programme

Le mode programme de Python consiste à écrire une suite d'instruction dans un fichier. L'interpréteur exécutera alors toutes ces instructions dans l'ordre. Nous allons maintenant écrire nos instructions dans la partie texte de IDLE. Une fois que le programme est écrit, on lance son exécution en appuyant sur la touche F5.

2.1 Affichage

En mode programme, le résultat d'un calcul n'est pas affiché par défaut. Pour le faire, on utilise la commande `print`. Tester le programme suivant :

```
print(42)
print(21+2*8)
print("coucou")
print("21+2*8")
```

On voit avec ce programme que :
— les commandes sont exécutées dans l'ordre ;
— `print(21+2*8)` affiche le résultat du calcul ;
— les guillemets permettent d'afficher du texte non interprété comme un calcul ;
— chaque `print` effectue un retour à la ligne à la fin de l'affichage.

Pour afficher quelque chose en Python, on utilise la syntaxe :

`print(<ce qu'on veut afficher>)`

On peut afficher :
— des nombres ;
— des résultats de calculs / des variables ;
— des textes (entourés par des guillemets).

Tester le programme suivant :

```
a = 1
b = 2
s = a+b
print("la somme de", a, "et", b, "est", s)

print("pas de re", end="")
print("tour a la ligne")
```

Il est possible d'afficher plusieurs éléments sur la même ligne :

`print(<elmt1>, <elmt2>, <elmt3>, ...)`

On peut définir le comportement de la fin d'un print :

`print(<ce qu'on veut afficher>, end = <fin de print>)`

Par défaut, la fin d'un `print` produit un retour à la ligne. Rajouter `end = ""` permettra de continuer exactement au même endroit au prochain.

2.2 Interagir avec l'utilisateur

On distinguera souvent en programmation le programme en lui même et l'utilisateur de ce programme avec qui le programme peut éventuellement interagir. Prenons un exemple de programme :

```
a=1
b=-1
c=1
x=3
print(a*x**2+b*x+c)
```

Ce programme calcule ax^2+bx+c avec $a=1$, $b=-1$, $c=1$ et $x=3$. Il est aisé de modifier ce programme en modifiant les variables `a, b, c, x`, cependant :
— ce n'est pas pratique ;
— cela ne permet pas à quelqu'un ne sachant pas programmer d'utiliser le programme.

On peut améliorer ce programme en ajoutant un peu de texte et en permettant à l'utilisateur d'entrer lui même les valeurs voulues grâce à la commande `input`. Lorsqu'un `input()` est lu par l'interpréteur, l'utilisateur du programme est invité à saisir un texte dans la console Python.

```
print("Calcul de ax**2+b*b+x :")
a = int(input("   Valeur de a ?"))
b = int(input("   Valeur de b ?"))
c = int(input("   Valeur de c ?"))
x = int(input("   Valeur de x ?"))
print("Resultat :")
print(a*x**2+b*x+c)
```

Dans ce programme, on donne la main à l'utilisateur et on récupère le texte qu'il tape grâce à la commande `input`.

Pour demander à l'utilisateur de saisir un texte et le récupérer :

```
s = input(<message pour l'utilisateur>)
```

Ici, le texte de l'utilisateur est stocké dans la variable `a`. Si on doit utiliser ce texte comme un nombre entier ou à virgule, il est important de le convertir avec les commandes `int` ou `float` :

```
a = int(input(<message pour l'utilisateur>))

x = float(input(<message pour l'utilisateur>))
```

3 Bibliothèque Turtle

3.1 Import de bibliothèques

Python propose un certain nombre d'instructions couvrant les besoins les plus courant en programmation comme `print` et `input`. On peut également avoir recours à des bibliothèques (ou modules) qui sont des collections d'instructions plus spécialisées et permettant d'effectuer de nouvelles tâches.

Quelques exemples de bibliothèques que l'on sera amené à utiliser :
- `math` contient des fonctions mathématiques ;
- `random` permet de générer des nombres aléatoires ;
- `sys` permet d'interagir avec l'OS de l'ordinateur ;
- `tkinter` permet de créer des fenêtres graphiques pour intéragir avec l'utilisateur.

Importation des instructions d'une bibliothèque : Pour utiliser les instructions d'une bibliothèque, il y a plusieurs façons de faire possibles (à ne faire qu'une fois, au début du programme) :

- `import <bibliothèque>` ou `import <bibliothèque> as <raccourci>` va nous permettre d'accéder au instructions de la bibliothèque à condition de préciser la bibliothèque à chaque fois (ou le raccourci). Par exemple, dans la bibliothèque `random`, on a l'instruction `randint(a,b)` permettant de générer un entier aléatoire entre deux entiers a et b.

```
>>> import random
>>> random.randint(1,6)
5
```

ou :

```
>>> import random as r
>>> r.randint(1,6)
2
```

- Avec `from <bibliothèque> import <fct1>, <fct2>, <...>` On peut choisir d'importer uniquement quelques instructions de la bibliothèque et dans ce cas, plus besoin de rappeler le nom de la bibliothèque à chaque appel. Par exemple, on peut importer les fonctions `cos` et `sqrt` (racine carrée) de la bibliothèque `math`.

```
>>> from math import cos, sqrt
>>> sqrt(9)
3
```

- Avec `from <biliothèque> import *` on peut aussi importer l'intégralité des fonctions comme précédemment. Cette méthode peut paraître pratique mais elle comporte des risques : si deux bibliothèques chargées ainsi comportent des instructions du même nom, elles vont rentrer en conflit. Aussi, on n'utilisera cette méthode que sur une seule bibliothèque à la fois.

```
>>> from math import *
>>> exp(0)
1
```

3.2 Turtle

Le module Turtle permet de dessiner des figures avec un stylo virtuel. Elle contient en premier lieu des instructions pour déplacer le stylo :

Instruction	Description
`goto(x,y)`	aller aux coordonnées (x, y)
`forward(d)`	avancer d'une distance d
`backward(d)`	reculer d'une distance d
`left(a)`	tourner à gauche d'un angle a
`right(a)`	tourner à droite d'un angle a
`circle(r,a)`	tracer un arc de cercle de rayon r et d'angle a
`dot(r)`	tracer un point de rayon r

Remarque : Les angles sont par défaut en degrés et dans `circle`, on tourne dans le sens trigonométrique (pour tourner dans l'autre sens, donner des angles négatifs).

Essayer le programme suivant :

```
from turtle import *
forward(60)
left(120)
forward(60)
right(90)
circle(60,300)
right(90)
forward(60)
goto(0,0)
```

On a également des instructions servant à modifier le comportement du stylo :

Instruction	Description
`up()`	relève le stylo
`down()`	abaisse le stylo
`width(e)`	règle l'épaisseur du trait à e
`color(c)`	règle la couleur du trait à c
`begin_fill()`	active le mode remplissage
`end_fill()`	tdésactive le mode remplissage
`fillcolor(c)`	règle la couleur du remplissage à c

Remarque : Les couleurs peuvent être désignées par une chaîne de caractères (`"yellow"`, `"blue"`, etc.) ou avec un triplet de nombres `(r,g,b)` avec r, g et b compris entre 0 et 1 et représentant respectivement les niveaux de rouge, vert et bleu de la couleur souhaitée. Toute aire contenue à l'intérieur de la trajectoire du stylo pendant que le mode remplissage est actif prend la couleur du remplissage.

Essayer le programme suivant :

```
from turtle import *
width(6)
color(0.2,0.5,0.2)
goto(60,0)
goto(60,110)
goto(0,110)
goto(0,0)
up()
goto(5,5)
down()
width(1)
fillcolor("blue")
begin_fill()
goto(55,5)
goto(5,105)
goto(55,105)
goto(5,5)
end_fill()
```

Enfin, quelques instructions permettent de modifier le comportement de la bibliothèque en général :

Instruction	Description
`reset()`	tout effacer et recommencer à zéro
`speed(s)`	définit la vitesse de déplacement du stylo
`title(e)`	donner le titre t à la fenêtre du dessin
`ht()`	cache le stylo
`st()`	montre le stylo

Remarque : La vitesse peut être donnée en texte (`"slowest"`, `"slow"`, `"normal"`, `"fast"`, `"fastest"`) ou parmi les nombres entiers entre 0 et 10 (1 = déplacement lent, 10 = déplacement rapide, 0 = déplacement instantané).