Numérique et sciences informatiques
Classe de première

A Programmation
6. Utiliser et créer un module

Lorsque l'on écrit un programme, on peut avoir besoin de fonctions ou de variables qui ne font pas partie du langage de programmation initial mais qui ont été créées ultérieurement.

• Il existe une multitude de fichiers python qui ont ainsi été créés et que l'on peut réutiliser. Certains fichiers, très souvent utiles, ont été téléchargés sur votre ordinateur au moment où vous y avez installé le langage Python. En Python de tels fichiers-ressouces s'appellent des **modules**.

• Nous allons découvrir quelques uns de ces modules et voir la syntaxe qu'il faut suivre pour pouvoir les utiliser.

I Importer et exploiter un module

I.1 Importer une partie ou la totalité d'un module

a) Importer la totalité d'un module

• Comme nous l'avons dit, un module est un fichier python. Il a donc un nom de la forme *monModule*`.py`.

• Si on souhaite avoir accès à la totalité des fonctions du fichier *monModule*`.py`, c'est-à-dire par exemple à toutes les fonctions et variables qu'il contient on doit écrire au début de son propre programme l'instruction suivante :

```
import monModule
```

Cette instruction peut aussi être tapée dans la console IDLE.

• Dans la suite du programme (ou dans IDLE), on pourra utiliser la fonction `maFonction(...)` en écrivant l'appel de fonction de la façon suivante :

```
monModule.maFonction(var1, ...)
```

Exemple :

```
>>> import math
>>> math.sqrt(5)
2.23606797749979
```

• On peut alléger un peu cette syntaxe en associant un nom plus court au module de la façon suivante :

```
import monModule as mod
```

Dans ce cas, les fonctions pourront être appelées de la façon suivante :

```
mod.maFonction(var1, ...)
```

Exemple :

```
>>> import math as m
>>> m.sqrt(5)
2.23606797749979
```

b) Importer une partie d'un module

• Pour éviter cette syntaxe qui reste lourde, on peut importer spécifiquement la fonction `maFonction` du module `monModule` de la façon suivante :

```
from monModule import maFonction
```

Dans ce cas, nous pouvons appeler la fonction comme si elle avait été définie dans notre propre fichier :

```
maFonction(var1, ...)
```

Exemple :

```
>>> from math import sqrt
>>> sqrt(5)
2.23606797749979
```

• On peut enfin importer *tous les éléments* d'un module (fonctions, variables, etc.) de la façon suivante :

```
from monModule import *
```

• Si on procède ainsi, on pourra appeler chaque fonction comme si elle avait été importée individuellement, c'est-à-dire sans faire précéder le nom de la fonction de "`monModule`" .
Exemple :

```
>>> from math import *
>>> sin(pi/6)
0.49999999999999994
```

I.2 Utiliser la console pour obtenir des informations sur un module

• Une fois que l'on a importé un module depuis la console IDLE, il est possible d'obtenir des informations sur ce module en tapant certaines commandes.

• Pour obtenir la liste des éléments d'un module, il suffit de taper :

```
>>> dir(monModule)
```

Exemple :

```
>>> import random
>>> dir(random)
['__call__',    '__class__',    '__delattr__',    '__dir__',
'__doc__',      '__eq__',       '__format__',     '__ge__',
'__getattribute__',   '__gt__',   '__hash__',   '__init__',
'__init_subclass__',  '__le__',   '__lt__',   '__module__',
'__name__',     '__ne__',       '__new__',      '__qualname__',
'__reduce__',   '__reduce_ex__',  '__repr__',   '__self__',
'__setattr__',          '__sizeof__',           '__str__',
'__subclasshook__', '__text_signature__']
```

• Pour avoir des informations sur un élement, on peut taper la commande suivante :

```
>>> help(monModule.unElement)
```

Exemple :

```
>>> help(random.randint)
Help on method randint in module random:
randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both
end points.
```

II Les modules `random` et `math`

II.1 Le module `random`

• Le module `random` contient de nombreuses fonctions permettant de générer des nombres de façon pseudo-aléatoire. Nous présenterons ici trois fonctions :

Nom de la fonction	Résultat obtenu
`random()`	renvoie « aléatoirement » un nombre de type **`float`** compris entre 0 inclus et 1 exclu.
`uniform(`*`a, b`***`)`**	renvoie « aléatoirement » un nombre de type **`float`** compris entre `a` inclus et `b` exclu.
`randint(`*`a, b`***`)`**	renvoie « aléatoirement » un nombre de type **`int`** compris entre `a` inclus et `b` inclus, où `a` et `b` sont de type **`int`.**

II.2 Le module `math`

• Le module `math` contient de nombreuses fonctions et constantes mathématiques.

Nom de la fonction	Résultat obtenu
`sqrt(`*`nb`***`)`**	renvoie la racine carrée de la variable `nb` de type **`int`** ou **`float`** sous la forme d'un nombre de type **`float`**
`floor(`*`nb`***`)`**	renvoie le plus grand entier inférieur à `nb` sous la forme d'un nombre de type **`float`**
`ceil(`*`nb`***`)`**	renvoie le plus petit entier supérieur à `nb` sous la forme d'un nombre de type **`float`**

Fonctions et constantes de trigonométrie

Nom	Résultat obtenu
`pi`	une constante qui contient une valeur approchée de la constante Pi avec un arrondi à 10^{-15}
`degrees(`*`x`***`)`**	convertit en degrés un angle `x` exprimé en radians
`radians(`*`x`***`)`**	convertit en radians un angle `x` exprimé en degrés
`cos(`*`x`***`)`**	renvoie le cosinus de `x` (exprimé en radians)
`sin(`*`x`***`)`**	renvoie le sinus de `x` (exprimé en radians)
`tan(`*`x`***`)`**	renvoie la tangente de `x` (exprimé en radians)

III Le module `turtle`

III.1 Présentation de la bibliothèque

Le module `turtle` permet de tracer des dessins dans une fenêtre créée spécifiquement pour cela. Le premier appel d'une fonction de cette bibliothèque va créer cette fenêtre graphique.

Essayer vous-même dans IDLE :

```
>>> from turtle import *
>>> title("ma zone de dessin")
```

Remarque : L'instruction `title(...)` crée une fenêtre et lui donne un nom correspondant à la chaîne de caractère passée en argument.

Le principe : on imagine une « tortue » que l'on déplace dans la fenêtre graphique et dont les déplacements peuvent tracer des lignes dans cette fenêtre lorsque le « crayon » qu'elle porte est abaissé.

• Pour repérer la position de la tortue, on dispose d'un repère orthornormé dont l'unité est le pixel, dont l'origine est au centre de la fenêtre et dont l'orientation des axes est horizontale vers la droite pour l'axe des abscisses et verticale vers le haut pour l'axe des ordonnées.

III.2 Gérer l'état du stylo

Appel de fonction	Résultat obtenu
up(), penup() ou **pu()**	on « lève le stylo » : les déplacements suivants sont effectués *sans trait* dans la fenêtre graphique
down(), pendown() ou **pd()**	on « abaisse le stylo » : les déplacements suivants seront effectués avec un trait dans la fenêtre graphique
isdown()	renbvoie `True` si le stylo est abaissé et `False` sinon

III.3 Gérer le déplacement de la tortue

Appel de fonction	Résultat obtenu
goto(x, y)	on place la tortue au point de coordonnées (x, y)
forward(*d*)	la tortue *avance* d'une distance d (en pixels) dans une direction qui dépend donc de son orientation au moment de l'instruction
backward(*d*)	la tortue *recule* d'une distance *d* (en pixels) vers l'arrière dans une direction qui dépend donc de son orientation
left(*a*)	la tortue pivote (et change donc d'orientation) d'un angle de *a* degrés « vers la gauche » (sens trigonométrique)
right(*a*)	La tortue pivote (et change donc d'orientation) d'un angle de *a* degrés « vers la droite » (sens des aiguilles d'une montre)
circle(*r*, *a*)	La tortue trace un arc de cercle de rayon *r* (pixels) et d'angle *a* (degrés)
dot(d)	La tortue trace un disque de diamètre *d*

III.4 Agir sur le trait

Appel de fonction	Résultat obtenu
width(*e*) ou **pensize**(e)	définit l'épaisseur du trait du crayon à *e* pixels.
color(*c*)	définit la couleur du trait. Cette couleur est indiquée par l'argument c, où c peut être : * un nom de couleur en anglais tel que "black", "blue", "red", etc. * un code de couleur formé de trois flottants compris entre 0 et 1
begin_fill()	on commence le remplissage
end_fill()	on achève le remplissage
fillcolor(*c***)**	on choisit la couleur de remplissage.

III.5 Autres fonctions

reset()	on efface tous les tracés dans la fenêtre et on place la tortue en (0 ; 0).
speed(*v***)**	on choisit la vitesse de déplacement de la tortue. v doit être un entier compris entre 0 et 10 où 1 est le plus lent, 10 est le plus rapide. 0 correspond à un déplacement « instantanné ».
ht()	on « cache la tortue » : la flèche qui la représente n'est plus affichée.