Alain Busser

Prépas
CAPES
AGREG

Python pour les (futurs) professeurs



ellipses

Références sciences

Python pour les (futurs) professeurs

Prépas
CAPES
AGREG

Alain Busser

ellipses

Collection Références sciences

dirigée par Paul de Laboulaye
paul.delaboulaye@editions-ellipses.fr

Retrouvez tous les livres de la collection et des extraits sur www.editions-ellipses.fr



Remerciements

L'auteur remercie Marion Le Gonidec, David Pouvreau, Bruno Pagani et Dominique Tournès pour la relecture de cet ouvrage, en espérant que cette relecture les a également amusé.e.s !

ISBN 9782340-110892
Dépôt légal : décembre 2025
©Ellipses Édition Marketing S.A.
8/10 rue la Quintinie 75015 Paris



Le Code de la propriété intellectuelle et artistique n'autorisant, aux termes des alinéas 2 et 3 de l'article L. 122-5, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1er de l'article L. 122-4).
Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.

www.editions-ellipses.fr

Avant-propos

Python

L'un des plus vieux langages de programmation connus est `Fortran`, développé par John Backus dans les années 1950. Comme on a rapidement trouvé que ce langage n'est pas parfait, d'autres langages ont été développés par la suite, et sont encore créés aujourd'hui. Parmi ces centaines de langages, Guido Van Rossum, qui a découvert la programmation lors de ses études de mathématiques, a créé en 1989 un langage de programmation (un de plus) qu'il a décidé de nommer `Python`, en hommage à l'acteur Eric Idle (membre de la troupe Monty Python).
Ce qui distingue Python des centaines d'autres langages de programmation existants, est le fait que

- le langage est concis (c'est une volonté de Guido Van Rossum),
- le langage est proche de la langue naturelle (mais de moins en moins, la concision étant considérée comme prioritaire sur le caractère naturel),
- Python fonctionne sur une machine virtuelle qui a été portée sur la plupart des plateformes connues (on peut programmer en Python sur un PC, un Mac, un ordinateur tournant sur Gnu/Linux, voire un smartphone),
- beaucoup de datalogues utilisent Python pour analyser les *Big Data*,
- beaucoup de chercheurs en intelligence artificielle (IA) utilisent Python pour programmer le *machine learning*,
- Python possède de nombreuses bibliothèques (entre autres pour les mathématiques),
- Python est très utilisé dans le domaine de l'éducation, pour l'apprentissage de la programmation,
- le projet `micropython` permet de programmer de toutes petites machines (entre autres les calculatrices graphiques) en Python.

Noms des langages

Depuis plusieurs années, il est d'usage de donner aux langages de programmation, des noms propres. On écrit donc le nom d'un langage de programmation avec une lettre majuscule suivie de lettres minuscules. Ainsi

- le mot `Python` (avec un P majuscule) désigne le ***langage*** Python,

- le mot `python` (en minuscules) désigne l'application permettant de programmer en Python.

En lançant l'application `python`, on ouvre une console permettant d'effectuer des calculs, comme ici :

```
>>> 6*7
42
```

Mais, souvent, on utilise un logiciel plus élaboré permettant d'écrire des scripts Python dans un éditeur de texte, de faire afficher les types des variables, les erreurs de syntaxe, parfois de dessiner la machine virtuelle Python (par exemple avec le site pythontutor [Guo09]).

Python et les mathématiques

Depuis 2009, le ministère de l'éducation nationale français injecte des activités de programmation dans les programmes de mathématiques, à partir de la Seconde (AlgoBox en 2010) puis à partir de la Sixième (Scratch en 2015). Entre-temps, Python a été choisi dans certaines formations post-bac (y compris en CPGE) et en enseignement de l'informatique (ISN en 2012) au lycée.
L'uniformisation des outils de programmation en cours de mathématiques est arrivée avec la mission de Laurent Chéno (Inspecteur Général), en 2019. Il a contribué à la mise au point des programmes d'informatique (SNT et NSI) avec Python comme langage central, et proposé Python dans les programmes de mathématiques du lycée.
Une expression qui courait alors chez certaines profs de mathématiques était : ***Python au service des mathématiques et non l'inverse***. En effet dès que Python est considéré comme une contrainte, une réaction de rejet s'ensuit, alors que si on voit ce que l'on peut gagner à utiliser Python, on est plus rapidement convaincu de son utilité.
Cet ouvrage est donc à considérer comme une liste (non nécessairement exhaustive) de services que Python peut rendre en cours de mathématiques.
On peut aussi voir cet ouvrage comme un livre de vulgarisation mathématique, utilisant Python comme vecteur. L'auteur espère que son lectorat s'amusera autant en lisant le livre, que l'auteur en l'écrivant.

Table des matières

1 La console **11**
1.1 Fonctionnement . . . 12
1.1.1 Expressions . . . 12
1.1.2 Instructions . . . 16
1.2 Affectation de variables . . . 16
1.2.1 Affectation directe . . . 17
1.2.2 Affectations en place . . . 18
1.2.3 Underscore . . . 19
1.3 Quelques singletons de Python . . . 19
1.3.1 Les petits entiers . . . 20
1.3.2 Les booléens . . . 21
1.3.3 None . . . 22
1.4 Les modules . . . 22

2 Ensembles et probabilités **23**
2.1 Les ensembles . . . 23
2.1.1 Ensembles . . . 23
2.1.2 Inclusion . . . 24
2.1.3 Intersection . . . 26
2.1.4 Réunion . . . 27
2.1.5 Complément . . . 28
2.1.6 Quantificateurs . . . 28
2.2 Algèbre de Boole . . . 29
2.2.1 Définition . . . 29
2.2.2 Logique propositionnelle . . . 29
2.2.3 Notion d'équation . . . 31
2.3 Mesure et probabilités . . . 32
2.3.1 Notion de borélien . . . 32
2.3.2 Théorie de Lebesgue . . . 33
2.3.3 Axiomatique de Kolmogorov . . . 33
2.3.4 Théorie de Bayes . . . 34

3 Couples, n-uplets et tuples **35**
3.1 Couples . . . 35
3.1.1 Par les ensembles . . . 35
3.1.2 Par les fonctions . . . 36
3.2 Graphes et relations . . . 37
3.2.1 Produit cartésien . . . 37
3.2.2 Variables . . . 39
3.3 Fonctions . . . 39
3.3.1 Définition . . . 39
3.3.2 Dictionnaires de Python . . . 40
3.3.3 Fonctions de Python . . . 42
3.4 Triplets et tuples . . . 42
3.4.1 Triplets . . . 42
3.4.2 Quadruplets . . . 42

4 Construction des nombres entiers **43**
4.1 Construction de Dedekind . . . 43
4.2 Axiomatique de Peano . . . 45
4.2.1 Relation d'ordre . . . 45
4.2.2 Addition . . . 45
4.2.3 Multiplication . . . 46
4.3 Construction de Von Neumann . . . 47
4.4 Construction de Church . . . 47
4.5 Théorie des catégories . . . 49
4.5.1 Objets et catégories . . . 49
4.5.2 Foncteurs . . . 53
4.5.3 Monades . . . 55
4.5.4 Construction des entiers naturels . . . 56
4.6 Entiers relatifs . . . 57

5 Arithmétique des nombres entiers **59**
5.1 Opération des entiers sur des additionnables . . . 59
5.1.1 Additions itérées . . . 59
5.1.2 Multiplication des entiers . . . 61
5.2 Division . . . 62
5.2.1 Propriété archimédienne . . . 62
5.2.2 Divisibilité . . . 63
5.3 Primalité . . . 67
5.3.1 Nombre de diviseurs . . . 67
5.3.2 Test de primalité . . . 67
5.3.3 Facteurs premiers . . . 69
5.3.4 Codage de Gödel . . . 71
5.3.5 une permutation des entiers . . . 73
5.3.6 Suite de Collatz . . . 74
5.4 Division euclidienne . . . 76
5.4.1 Division euclidienne en Python . . . 76

5.4.2 Arithmétique de Presburger 78
5.4.3 Les restes chinois 78

6 Dyadiques et décimaux **81**
6.1 Fractions dyadiques 81
6.1.1 Construction de Conway 82
6.1.2 Implémentation en Python 83
6.2 Écriture binaire 84
6.2.1 Pour les nombres entre 0 et 1 84
6.2.2 Suite logistique 86
6.2.3 Conjugaison 88
6.2.4 Des flottants non dyadiques 90
6.3 Nombres décimaux 93
6.3.1 Décimaux et flottants 93
6.3.2 Un module Python pour le calcul décimal 94

7 Fractions **97**
7.1 Construction des fractions 97
7.1.1 comme classes d'équivalence 97
7.1.2 comme opérateurs 101
7.1.3 comme quotients 102
7.2 Écriture mixte d'une fraction 103
7.3 Fractions de Farey 105
7.3.1 Approximations rationnelles d'un réel 105
7.3.2 L'arbre de Stern-Brocot 106
7.3.3 Les cercles de Ford 110
7.4 Fractions continuées 111
7.5 Les fractions égyptiennes 113
7.6 Les fractions selon Conway 114
7.6.1 Fractran 114
7.6.2 Les enchevêtrements de Conway 116

8 Structure de corps **119**
8.1 Corps à deux éléments 119
8.2 Corps des réels 120
8.2.1 Les réels en Python 120
8.2.2 Quelques réels particuliers 120
8.3 Corps finis 122
8.3.1 Corps ayant un nombre premier d'éléments 122
8.3.2 Autres corps finis 124
8.4 Corps des nimbers 125
8.4.1 Nim-addition 125
8.4.2 Nim-multiplication 127
8.5 Nombres algébriques 128
8.5.1 Triplets pythagoriciens 128
8.5.2 Nombres de Pell 130

8.5.3 Courbes elliptiques . . . 132
8.5.4 Extensions quadratiques . . . 134
8.6 Résolution d'équations . . . 135
8.6.1 Équations du second degré . . . 135
8.6.2 Équations du troisième degré . . . 135

9 Mathématiques des textes 137
9.1 Cryptographie . . . 137
9.1.1 Le chiffre de César . . . 137
9.1.2 Chiffre de Vigenère . . . 138
9.2 Parcours de graphes . . . 138
9.2.1 Chemins et cycles eulériens . . . 138
9.2.2 Chemins et cycles hamiltoniens . . . 140
9.3 Problème du mot . . . 141
9.3.1 Suite de Prouhet-Thue-Morse . . . 141
9.3.2 Générateurs et relations . . . 141
9.4 Travaux d'Emil Post . . . 142
9.4.1 Systèmes de tague . . . 142
9.4.2 Problème de correspondance . . . 143
9.4.3 Look and say . . . 145
9.5 Dessin de courbes fractales . . . 146
9.5.1 Courbe de Von Koch . . . 146
9.5.2 Courbe de Sierpinski . . . 147
9.5.3 Le mot de Fibonacci . . . 148
9.6 Automates . . . 149
9.6.1 Suites de Beatty . . . 149
9.6.2 Automate de division par 2 . . . 149
9.6.3 Automate de triplement . . . 151
9.7 Logique des combinateurs . . . 152
9.7.1 S, K et I . . . 152
9.7.2 Comme règles de réécriture . . . 153
9.7.3 B, C, K et W . . . 154
9.7.4 Le combinateur M . . . 154

10 Complexes 155
10.1 Complexes en Python . . . 155
10.1.1 Équations polynomiales . . . 156
10.1.2 Logarithme d'un nombre négatif . . . 157
10.1.3 Trigonométrie complexe . . . 158
10.1.4 Géométrie hyperbolique . . . 159
10.2 Entiers chez les complexes . . . 160
10.2.1 Entiers de Gauss . . . 160
10.2.2 Entiers d'Eisenstein . . . 163
10.3 Systèmes dynamiques complexes . . . 165
10.3.1 Ensembles de Julia . . . 165
10.3.2 Ensemble de Mandelbrot . . . 167

10.4 Quaternions . . . 169
10.5 Octonions . . . 172

11 Analyse **173**
11.1 Suites . . . 173
11.1.1 Algorithme de Héron . . . 173
11.1.2 Algorithme de Gauss-Salamin . . . 174
11.1.3 Le logarithme de 2 . . . 174
11.2 Fonctions . . . 175
11.2.1 Résolution d'équations . . . 175
11.2.2 Fonctions trigonométriques . . . 176
11.2.3 Limites . . . 178
11.2.4 Dérivée . . . 179
11.2.5 Intégrales . . . 182
11.3 Calcul formel . . . 185
11.4 Circulation d'un vecteur . . . 186

12 Algèbre linéaire **187**
12.1 Vecteurs du plan . . . 187
12.1.1 Vecteurs de la tortue . . . 187
12.1.2 Droites du plan . . . 189
12.1.3 Systèmes dynamiques dans le plan . . . 190
12.2 Vecteurs de l'espace . . . 192
12.2.1 Vecteurs 3D en Python . . . 192
12.2.2 Espace des cercles . . . 193

13 Statistiques **195**
13.1 Statistique en Python . . . 195
13.1.1 Les listes de Python . . . 195
13.1.2 Statistiques sur les listes . . . 196
13.1.3 Régression linéaire . . . 196
13.2 Variables aléatoires . . . 197
13.2.1 Variables aléatoires binomiales . . . 198
13.2.2 divhasard . . . 200

14 Annexes **203**
14.1 Interpréteur Sofus en Python . . . 203
14.2 Catégories et foncteurs en Python . . . 205
14.2.1 Catégories . . . 205
14.2.2 Foncteurs . . . 206
14.3 Union-Find . . . 206
14.4 Nombres algébriques . . . 207
14.5 Quaternions . . . 207
14.6 Octonions . . . 208
14.7 Vecteurs . . . 209
14.7.1 Dessin de vecteurs . . . 209
14.7.2 Barycentres . . . 209

14.7.3 vecteur en 3D 209
14.8 Ensemble de Cantor 210

Bibliographie **210**

Index **212**

Chapitre 1

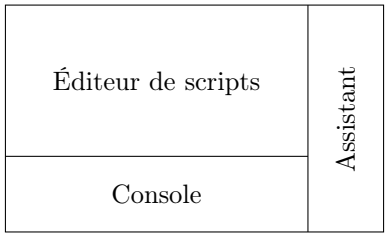
La console

Il y a basiquement deux façons d'utiliser l'application `python` :

- On peut créer un script (écrit en Python) qui est dans un fichier appelé par exemple ***module.py*** puis l'exécuter en entrant en ligne de commande `python module.py`. C'est de cette manière que fonctionnent les logiciels aidant à programmer en Python, comme Mu, Thonny, EduPython, etc., mais aussi ceux permettant d'installer des modules additionnels comme `pip3`.
- On peut aussi entrer directement la commande `python` (sans nom de fichier derrière) qui permet de programmer de façon interactive, ligne par ligne, et s'avère plus efficace, pédagogiquement, que se lancer directement dans le grand bain.

La console de Python n'existe pas seulement en ligne de commande, on la trouve également sur les calculatrices (parfois sous le nom de *shell*) mais aussi dans des logiciels de programmation Python (Anaconda, jupyter, etc.) ou pas ; par exemple le logiciel de topologie `regina-normal` qui permet de faire des calculs dans des espaces topologiques de dimensions 3 ou 4 dotés d'une structure de géométrie hyperbolique (difficile d'imaginer plus mathématique que ça !) est doté d'une console Python, ainsi que le logiciel de traitement d'images `Gimp` (sous la forme du filtre `Python-fu`) et le logiciel de modélisation 3D `Blender` (sous la forme d'une vue pouvant par exemple remplacer celle de la caméra). L'installation de Python depuis le site `https://www.python.org/` donne de toute façon une console appelée IDLE, elle-même programmée en Python.

Dans cet ouvrage, les extraits de dialogue avec la console proviendront en général de Thonny [Ann15], dont la fenêtre est organisée ainsi :



1.1 Fonctionnement

Dans une console Python, on voit régulièrement apparaître un ***prompt***, qui, dans le cas de Python, est représenté par trois caractères ***chevron*** (supérieur, en mathématiques) disposés ainsi :

```
>>>
```

Ce prompt est à considérer comme une parole du *djinn* du conte *al Ad'Din*. Dans le conte, le djinn demande sans cesse à celui qui possède la lampe merveilleuse, quel est le vœu à réaliser. La console Python, quant à elle, demande l'expression à évaluer.

1.1.1 Expressions

Une ***expression*** est une suite de caractères (lettres, chiffres, symboles opératoires, parenthèses) ayant une valeur. Par exemple `2+3` est une expression, tout comme `sin(pi)` ou `(x+1)*(3*x-1)`. Toute expression (bien formée tout au moins) a une valeur et le travail de la console python, c'est justement d'évaluer cette valeur. Pour cela, on va sur la console :

```
>>>
```

on entre l'expression :

```
>>> 2+3
```

puis, pour évaluer l'expression, on appuie sur le bouton Entrée :

```
>>> 2+3
5
```

Une expression (ou plutôt, sa valeur) a un type, qu'on peut connaître en lui appliquant la fonction `type` :

```
>>> type(2+3)
<class 'int'>
```

Le type d'une valeur (un objet Python) est en fait une fonction qui peut transformer un objet de type donné en un objet du nouveau type :

```
>>> int(3.14)
3
```

Ici un réel (le nombre 3,14) a été converti en un entier (celui qu'on obtient en supprimant tout ce qui est après la virgule). La conversion ne marche pas toujours :

```
>>> int(3.1.4)
  File "<stdin>", line 1
    int(3.1.4)
        ^^^^^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

Le type `type` est lui-même un type :

```
>>> type(type)
<class 'type'>
```

Mais ce n'est pas le plus utile en mathématiques. Un des types les plus importants est celui qui permet de faire de la logique, c'est le type `bool` qu'on verra dans le chapitre sur les ensembles :

```
>>> 1 < 2
True
>>> type(True)
<class 'bool'>
```

Pour vérifier que $2 \leqslant 2$ on écrit :

```
>>> 2 <= 2
True
```

Mais pour vérifier que $2 + 2 = 4$ on n'écrit pas le signe égal :

```
>>> 2+2 = 4
  File "<stdin>", line 1
    2+2 = 4
    ^^^
SyntaxError: cannot assign to expression here.
Maybe you meant '==' instead of '='?
```

En effet, le signe d'égalité est utilisé pour une instruction (voir plus bas). Pour tester une égalité on utilise le double égal :

```
>>> 2+2 == 4
True
```

Les types les plus utiles en mathématiques sont `bool` (propositions logiques), `int` (entiers relatifs) et `float` (réels).
En ce qui concerne les expressions algébriques, Python connaît les priorités opératoires :

```
>>> 2+3*4
14
```

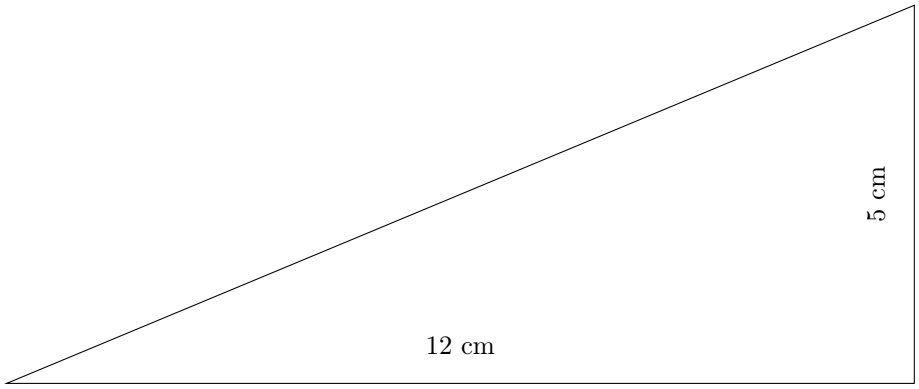
ainsi que la soustraction des relatifs :

```
>>> 2-(-3)
5
```

L'opération d'exponentiation se note par un double astérisque :

```
>>> 3**2
9
>>> 2**3
8
```

Par exemple, si on veut calculer l'hypoténuse de ce triangle rectangle :



On peut commencer par calculer le carré de cette hypoténuse, qui d'après Pythagore, est

```
>>> 12**2 + 5**2
169
```

puis chercher de qui 169 est le carré. Mais comment ***racine carrée*** se traduit en Python ? En anglais on dit *square root* qui s'abrège en `sqrt` :

```
>>> sqrt(169)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
```

En réalité, le mot `sqrt` n'est pas défini en Python. Il faut donc le définir (par exemple avec l'algorithme de Héron d'Alexandrie ; voir le chapitre sur l'analyse) ou, plus simple, l'importer (ici, depuis le module `math`) :

```
>>> from math import sqrt
>>> sqrt(169)
13.0
```

Remarque : il y a d'autres moyens de calculer l'hypoténuse, par exemple si on sait que $\sqrt{x} = x^{\frac{1}{2}}$:

```
>>> 169**0.5
13.0
```

ou, avec une meilleure connaissance du module `math` :

```
>>> from math import hypot
>>> hypot(12,5)
13.0
```

qui fait utiliser Pythagore par la pythie de Python plutôt que par l'utilisatrice humaine...
Les listes de Python peuvent aussi être considérées comme des expressions, ce qui permet parfois de se passer de boucles :

```
>>> [x**2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

De fait, il est bon d'attendre au moins quelques heures avant de programmer en Python (c'est-à-dire écrire un script) parce que la console est un outil d'apprentissage hors pair, ne serait-ce que pour réviser les priorités opératoires (et aussi, se familiariser avec la syntaxe d'un langage qui est, comme tout langage de programmation, assez long à apprendre).
Mais dès qu'on a appris à programmer en Python (définir des fonctions dans un script) il reste possible de continuer à utiliser la console. Par exemple si on veut calculer (en kg/m^2) un IMC connaissant la masse corporelle et la taille d'un individu, on constate que la fonction `imc` n'existe pas, pas même dans le module `math` :

```
>>> from math import imc
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: cannot import name 'imc' from 'math' (unknown location)
```

Qu'à cela ne tienne, définissons-la :

```
>>> def imc(masse,taille): return masse/taille**2

>>> imc(80,1.75)
26.122448979591837
```

La définition de la fonction `imc` peut éventuellement être stockée dans un fichier nommé `nutri.py` et ensuite, utilisée dans la console sous peine d'avoir exécuté l'instruction (voir plus bas) suivante :

```
>>> from nutri import imc
```

Même lorsqu'on maîtrise la programmation Python, l'usage de la console pour tester des hypothèses en évaluant des expressions (en particulier avec la fonction `type`) peut s'avérer utile pour la programmation. Ceci dit, si on définit, avec indentation [1] correcte, la fonction `imc` dans un fichier Python, cela donne

```
def imc(masse,taille):
    return masse/taille**2
```

1. L'indentation d'une ligne est le décalage de cette ligne vers la droite.

où l'indentation est bien visible, alors que dans la console on a

```
>>> def imc(masse,taille):
    return masse/taille**2
```

où le prompt gêne la vue. Dans ce livre il faudra donc, à chaque fois que l'indentation apparaît (`def`, `for` etc.) faire abstraction de ce prompt et imaginer que la ligne avec le prompt est reculée pour avoir une indentation correcte sur les autres lignes.

1.1.2 Instructions

Pour faire un peu de trigonométrie en Python, on peut tenter

```
>>> from math import sin,cos,pi
>>> sin(pi/2)
1.0
>>> cos(pi/2)
6.123233995736766e-17
```

qui montre que, pour Python, le cosinus d'un angle droit n'est pas tout-à-fait égal à zéro (voir le chapitre sur l'analyse pour en savoir plus) mais aussi que la première expression `from math import sin,cos,pi` n'a pas été évaluée. En fait ce n'est ***pas une expression***. C'est une instruction. Une instruction n'a pas de valeur (enfin, sa valeur est `None` pour être précis, voir plus bas à propos de `None`) mais par contre, elle a vocation à être exécutée. L'exécution d'une instruction a pour effet de modifier la mémoire de l'ordinateur (ou la position d'un robot). Par exemple

```
>>> from turtle import *
>>> left(45)
>>> forward(100)
```

ne donne aucune évaluation d'expression, mais a pour effet de

- créer un robot virtuel de type *tortue*,
- faire tourner ce robot d'un angle de 45° vers sa gauche,
- faire avancer le robot d'une longueur de 100 pixels,

ce qui produit un dessin (le robot est muni d'un crayon qui trace le parcours effectué, comme un escargot qui dépose sa bave) mais ne calcule rien dans la console.
Des instructions courantes dans la console sont `import`, `from ... import ...`, `assert ...`, `def ...: return ...` (`return` est en soi une instruction). La plus taoïste des instructions est `pass` : elle ne fait rien (*wu wei* en chinois).
Mais les instructions les plus fréquentes sont les affectations, qui servent à créer et modifier des variables.

1.2 Affectation de variables

L'expression `x+2` n'a de valeur que si `x` en a une, ce qui, au début d'une session Python, n'est pas le cas :

```
>>> x+2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Si `x` est égal à 3, l'expression a une valeur, à savoir 3+2 :

```
>>> x = 3
>>> x+2
5
```

L'instruction `x = 3` a eu pour effet de créer une variable, ayant

- un nom : `x`,
- une valeur : 3.

Une fois qu'une variable est créée, on ne peut plus modifier son nom, mais on peut modifier sa valeur (c'est pour ça qu'on l'appelle une variable) en lui imposant le résultat d'une évaluation d'expression. Par exemple si on veut que x soit égal à 1 on écrit :

```
>>> x = 1
>>> x+2
3
```

1.2.1 Affectation directe

Souvent, on a besoin d'incrémenter une variable (par exemple pour compter), c'est-à-dire augmenter sa valeur d'une unité (si elle était à 5, la faire passer à 6). Or ceci ne marche pas :

```
>>> compteur = 5
>>> compteur + 1
6
```

En effet, si l'instruction `compteur = 5` a bien eu pour effet que le compteur soit égal à 5, la ligne suivante `compteur + 1` est une expression, et n'a aucun effet :

```
>>> compteur
5
```

Pour que le compteur soit effectivement incrémenté, il faut

- évaluer l'expression `compteur + 1`,
- puis placer le résultat de cette évaluation dans la variable `compteur` :

```
>>> compteur = compteur + 1
>>> compteur
6
```

C'est là qu'un se rend compte que l'utilisation du signe d'égalité (depuis Fortran au milieu des années 1950) pour une affectation, est malheureuse, car il est évident que l'équation $x = x + 1$ n'a pas de solution :

```
>>> compteur == compteur + 1
False
```

En Python, le signe d'égalité ne sert pas à exprimer une égalité, mais une affectation. C'est le signe d'égalité dédoublé qui sert à exprimer (ou tester) une égalité.
Au fait, est-il si évident que cela, que l'équation $x = x + 1$ n'a pas de solution ?

```
>>> x = float('inf')
>>> x == x+1
True
>>> x + 1
inf
```

En fait, $\infty + 1 = \infty$ et $-\infty + 1 = -\infty$ donc l'équation $x = x + 1$ a deux solutions dans $\overline{\mathbf{R}} = [-\infty, +\infty]$ et une solution sur la droite projective puisque sur la droite projective $\infty = -\infty$ (point invariant par les translations).
Mais un compteur a rarement le temps de devenir infini, et on a besoin d'une façon plus intuitive que `compteur = compteur + 1` pour augmenter d'une unité un compteur. Or, tout comme Sofus [Bus18] qui a été créé pour éviter ce problème, Python permet de faire des affectations en place.

1.2.2 Affectations en place

Pour modifier une variable, en général

- on lit son contenu actuel,
- on évalue une expression où intervient ce contenu actuel,
- puis on écrase l'ancien contenu, en le remplaçant par le résultat de l'évaluation.

C'est compliqué. Et pas seulement pour les élèves. Python permet également de modifier directement des variables, *in situ* c'est-à-dire sans passer par l'extraction et la réinjection de données. Pour cela on écrit le symbole d'une opération (addition, soustraction, multiplication, division euclidienne, division exacte, nim-addition, opérations sur les booléens, etc.) entre le nom de la variable et la quantité dont elle doit changer, suivi par un signe d'égalité. Par exemple pour incrémenter une variable `compteur`, il suffit de faire

```
>>> compteur += 1
```

Pour décrémenter une variable, ce sera

```
>>> decompte -= 1
```

Pour doubler une variable, on fait

```
>>> variable *= 2
```

1.2.3 Underscore

Dans la console Python, il y a une variable dont le nom est formé d'une espace soulignée (appelée *underscore* en anglais) que les élèves appellent *tiret du 8* et qui contient le résultat de la dernière évaluation d'expression :

```
>>> 6*7
42
>>> _
42
>>> _+1
43
>>> _
43
```

Quand on a besoin d'une variable (par exemple une variable de boucle) et qu'on ne sait pas comment la nommer, on peut utiliser cette variable presque anonyme :

```
>>> [1 for _ in range(8)]
[1, 1, 1, 1, 1, 1, 1, 1]
```

Cette coutume vient du langage `Prolog` qui utilisait ce symbole pour une place vide c'est-à-dire une variable dont la valeur importe peu.

1.3 Quelques singletons de Python

Python est un langage de programmation objet, c'est-à-dire que les résultats des évaluations des expressions Python sont des objets, dotés de méthodes dépendant de leur type. On dit même que le type d'une variable est déterminé par le jeu de ses méthodes. Par exemple, on sait que 2+3 est un entier parce que si on demande quel est son type, Python répond que c'est un entier. Mais si on demande ses méthodes :

```
>>> type(2+3)
<class 'int'>
>>> dir(2+3)
['__abs__', '__add__', ..., 'real', 'to_bytes']
```

La liste des attributs (comme `denominator`) et des méthodes est très longue, cela veut dire que les entiers sont très riches et savent faire beaucoup de choses. Note : il n'est pas nécessaire d'appliquer la fonction `dir` à un entier particulier, on peut aussi l'appliquer au type correspondant :

```
>>> dir(int)
['__abs__', '__add__', ..., 'real', 'to_bytes']
```

En `Smalltalk` et en `Java`, un nombre est un objet doté de méthodes équivalant à ces méthodes de Python :

- `__lt__()` (*less than*) permettant la comparaison entre nombres,

- `__add__()` permettant d'additionner un nombre avec un autre nombre.

Mais comme il n'y a pas de relation d'ordre compatible avec l'addition sur les nombres complexes, ceux-ci ne sont pas des nombres en Java et en Smalltalk ! En revanche, les chaînes de caractères de Python peuvent être comparées (dans l'ordre alphabétique) entre elles :

```
>>> 'bon' < 'jour'
True
```

et être additionnées entre elles :

```
>>> 'bon' + 'jour'
'bonjour'
```

ce qui ferait d'elles des nombres. Cependant l'addition des chaînes de caractères n'est pas commutative :

```
>>> 'jour' + 'bon'
'jourbon'
```

et, de fait, on ne considère pas les chaînes de caractères comme des nombres, même en Python !

Revenons aux entiers. Un entier n'est pas tout à fait la même chose en Python et en mathématiques : en mathématiques, un entier est un objet ayant certaines propriétés qui font que, par exemple, on peut additionner les entiers entre eux. En Python, un entier est un objet qui possède une méthode `__add__()` qui ***lui permet*** de s'additionner avec un autre entier. Ce n'est pas la mathématicienne qui sait comment additionner des entiers, c'est l'entier qui sait comment s'additionner avec un autre entier.

Une autre différence est que, si on a besoin d'un grand entier (par exemple 2025) au cours d'un calcul, cet entier est créé (c'est-à-dire qu'on lui réserve une place dans la mémoire) pour le calcul, et ensuite, lorsqu'on n'a plus besoin de lui, il est effacé de la mémoire, par un algorithme appelé ***ramasse-miettes*** (`garbage collector` en anglais) et n'existe plus vraiment, jusqu'à ce qu'on ait besoin de lui à nouveau, auquel cas il sera recréé.

C'est comme la théorie de la subitisation, selon laquelle il n'existe pas de neurone reconnaissant le nombre 13, ce qui oblige à compter pour dénombrer 13 objets (et les neurologues mesurent que ça prend plus de temps pour dénombrer 13 objets, que pour en dénombrer 8). Par contre, le fait que les nombres 0, 1, 2 et 3 soient reconnus presque immédiatement laisse penser qu'il y a des neurones qui leurs sont dédiés. L'équivalent en Python est appelé singleton : c'est un objet qui existe en un seul exemplaire et qui a sa place réservée en mémoire, de façon à pouvoir s'y référer chaque fois qu'on en a besoin.

1.3.1 Les petits entiers

Les entiers (typiquement entre -255 et 255) les plus petits sont des singletons. Il ne serait pas utile (ni efficace) que Python recrée l'entier 1 chaque fois qu'il en a besoin,