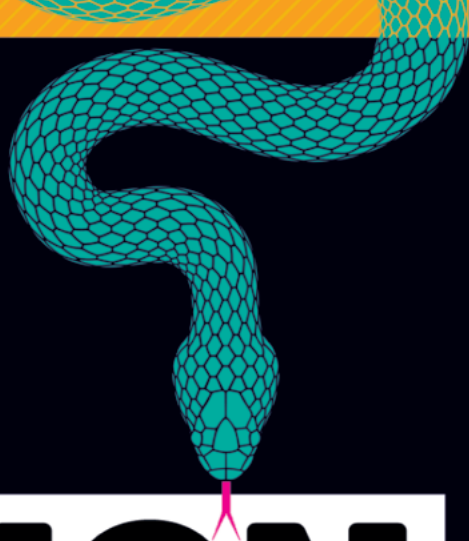


INF>RMATIQUE



PYTHON

POUR LA PHYSIQUE

Calcul, graphisme,
simulation

RICHARD TAILLET

INF&RMATIQUE

PYTHON

POUR LA PHYSIQUE

Calcul, graphisme, simulation

RICHARD TAILLET

deboeck **B**
SUPÉRIEUR

Pour toute information sur notre fonds et les nouveautés dans votre domaine de spécialisation, consultez notre site web : **www.deboecksuperieur.com**

© De Boeck Supérieur s.a., 2020
Rue du Bosquet, 7, B-1348 Louvain-la-Neuve

Tous droits réservés pour tous pays.

Il est interdit, sauf accord préalable et écrit de l'éditeur, de reproduire (notamment par photocopie) partiellement ou totalement le présent ouvrage, de le stocker dans une banque de données ou de le communiquer au public, sous quelque forme et de quelque manière que ce soit.

Dépôt légal :
Bibliothèque nationale, Paris : septembre 2020
Bibliothèque royale de Belgique, Bruxelles : 2020/13647/135

ISBN : 978-2-8073-2890-7

Table des matières

Table des matières	1
1 Introduction	11
1 Python	11
2 Aborder la physique autrement	11
3 Plan de l'ouvrage	13
4 Règles d'or	14
5 Remerciements	14
6 Télécharger les programmes	14
I Python	15
2 Fonctions de base	17
1 Démarrer et quitter	17
2 Calculatrice en mode interactif	18
3 Fonctions et paramètres	19
4 Bibliothèques	19
5 Écriture de programmes	22
6 Variables	23
7 Afficher	29
8 Erreurs	33
9 Précision	34
10 Tuples	35
11 Conditions <code>if - else</code>	37
12 Boucles <code>while</code> et <code>for</code>	41
13 Fonctions définies par l'utilisateur	46
13.1 Définir une fonction	46
13.2 Valeur par défaut des paramètres	46
13.3 Une fonction peut renvoyer plusieurs valeurs, ou aucune	47
13.4 Portée des variables : locale ou globale	48
14 Commentaires	50
15 IPython	51
3 Unités	53
1 Dimensions physiques	53
2 Unités de mesure	53
3 Constantes et unités	54
4 Modules dédiés	57

4.1	Les constantes physiques dans <code>scipy</code>	57
4.2	Le module <code>astropy</code>	59
4.3	Les valeurs du CODATA	61
4.4	Analyse dimensionnelle	62
4	Listes et tableaux	65
1	Listes	65
2	Méthodes et fonctions	66
3	La fonction <code>enumerate()</code>	67
4	Listes en compréhension	68
5	Listes de listes	69
6	La bibliothèque <code>numpy</code>	69
6.1	Tableaux <code>numpy</code>	69
6.2	Les commandes <code>arange()</code> et <code>linspace()</code>	71
6.3	Les fonctions mathématiques de <code>numpy</code>	72
7	Aparté technique	72
7.1	Affectation des variables	72
7.2	Copier une variable	74
7.3	Modifier une variable dans une fonction	75
8	Résumé de fonctions et méthodes utiles pour les listes	76
5	Représentation de données	77
1	Tracé des points	78
1.1	La fonction <code>scatter</code> de <code>matplotlib</code>	78
1.2	Sauvegarder la figure	80
1.3	Aspect des symboles	80
1.4	Barres d'erreur	81
1.5	Ajouter des titres, des légendes et du texte	83
1.6	Gestion des axes	86
1.7	Point de vocabulaire : figures et graphes	88
1.8	Afficher une grille de coordonnées	89
1.9	Enlever le cadre et positionner les axes : <code>spines</code>	90
1.10	Échelle logarithmique	92
2	Représentation graphique de fonctions	94
2.1	La fonction <code>plot()</code>	95
2.2	Les attributs de la ligne	96
2.3	Centrer les axes	98
3	Tracer plusieurs courbes	99
3.1	Sur le même graphe	99
3.2	Sur des graphes différents dans une même figure	100
3.3	Sur des figures différentes	101
4	Réglage fin des figures et des graphes	102
5	Histogrammes	104
5.1	Tracer des histogrammes	104

5.2	Les fonctions statistiques	107
6	La notion de backend	108
7	Les styles de matplotlib	109
8	Résumé des commandes utiles	112
9	Mini-projets	113
6	Dessin	115
1	Tracer des segments	115
2	Unités	116
3	Formes géométriques	117
4	Les couleurs	120
5	Les lignes, objets de type Line2D	120
6	Notion nouvelle : externalisation de fonctions	122
7	Mini-projets	123
II	Optique géométrique	125
7	Réfraction	127
1	Théorie	127
2	Illustration en python	127
2.1	Rappel de géométrie	127
2.2	Tracé des rayons lumineux	128
2.3	Exemples de mauvaises pratiques	129
2.4	Ajout des rapporteurs	130
3	Figures interactives : gestion des événements	131
4	Mini-projets	136
8	Le prisme	137
1	Théorie	137
2	Programmation	138
3	Déviation minimale	142
4	Courbe de déviation et minimum de déviation	144
4.1	Courbe de déviation	144
4.2	Détermination du minimum d'une fonction	145
5	Mini-projets	146
9	Sphère réfractante	147
1	Présentation	147
2	L'ordre zéro	147
3	L'ordre 1 : arc primaire de l'arc-en-ciel	149
4	L'ordre 2 : arc secondaire de l'arc-en-ciel	151
5	L'ordre N	153
6	Curseurs et boutons interactifs : les widgets	154

6.1	Slider : exemple minimal	154
6.2	Application à la sphère réfractante	156
6.3	Button : exemple minimal	157
6.4	RadioButtons et CheckButtons : exemples minimaux	159
6.5	Application à la sphère réfractante	160
7	Programmer en mode tortue	162
8	Mini-projet : les couleurs de l'arc-en-ciel	165
10	Miroirs et dioptres	167
1	Miroir sphérique	167
2	Dioptres sphériques	169
2.1	Faisceau incident parallèle	169
2.2	Source à distance finie	171
2.3	Points de Weierstrass	172
2.4	Obtention d'une lentille	173
3	Dioptre elliptique	174
4	Dioptre de profil quelconque	177
4.1	Détermination et dessin des rayons réfractés	177
4.2	Calcul numérique d'une dérivée	179
4.3	Calcul numérique de la dérivée par scipy	181
4.4	Application au dioptre de forme quelconque	182
4.5	Réfraction par des vagues	182
5	Caustiques : nappes tangentielles	184
5.1	Théorie	184
5.2	Exemple : réflexion sur un miroir sphérique concave	185
6	Mini-projets	186
6.1	Mini-projet 1 : caustique du miroir sphérique convexe	186
6.2	Mini-projet 2 : caustique du dioptre plan	187
6.3	Mini-projet 3 : réflexions multiples sur un miroir concave	187
11	Stigmatisme	189
1	Stigmatisme approché, conditions de Gaus	189
2	Images dans les conditions de Gauss	192
3	Aberrations du miroir sphérique	195
3.1	Cartes de densité : histogrammes bidimensionnels	198
3.2	Cercle de moindre confusion	201
4	Mini-projets : étude d'une lentille plan-convexe	203
12	Lentilles	205
1	Théorie	205
2	Application à une lentille	206
3	Définir de nouvelles classes d'objets	207
4	Position de l'image	213
5	Mini-projets	216

III Mécanique du point	217
13 Chute avec frottement	219
1 Relation fondamentale de la dynamique	219
2 Force de traînée	219
3 Chute dans l'air avec frottements	220
4 Résolution numérique : première approche	222
4.1 Utilisation de développements limités	223
4.2 Raffinement numérique	225
5 Utilisation de <code>scipy.integrate</code>	227
5.1 Équation différentielle du premier ordre	228
5.2 Équation différentielle du second ordre	230
5.3 Détecter des annulations	232
5.4 Solution dense	232
6 Application au problème de chute	233
7 Portée d'un lanceur de projectile	236
8 Mini-projets	238
14 Oscillateurs	239
1 Oscillateur harmonique	239
1.1 Définition	239
1.2 Équation différentielle du second ordre	239
1.3 Conservation de l'énergie mécanique	241
2 Pendule simple	242
2.1 Période du pendule	243
2.2 Animation des figures	249
15 Chaos	253
1 Pendule amorti et forcé	253
2 Résolution numérique	253
2.1 Mouvement	253
2.2 Doublement de période	255
2.3 Portrait de phase	256
2.4 Diagramme de bifurcation	258
3 Données persistantes	260
3.1 Lecture/écriture dans un fichier	261
3.2 Le module <code>json</code>	262
3.3 Application au diagramme de bifurcation	263
3.4 Le module <code>pickle</code>	264
16 Forces centrales	265
1 Résolution numérique en coordonnées cartésiennes	265
1.1 Force en r^n et oscillateur harmonique bidimensionnel	265
1.2 Force gravitationnelle	267

2	Coordonnées polaires	270
3	Loi des aires	272
4	Mini-projets	274
17	Chute dans un référentiel tournant	275
1	Déviati on vers l'est	275
2	Déviati on vers le sud	279
3	Calcul symbolique avec SymPy	281
3.1	Développements limités	281
3.2	Rendu LaTeX	281
3.3	Dérivées	283
3.4	Primitives	283
4	Mini-projets : approfondissement	285
4.1	Mini-projet 1 : force gravitationnelle plus réaliste	285
4.2	Mini-projet 2 : prise en compte des frottements	286
18	Points de Lagrange	287
1	Théorie	287
2	Dessiner une surface	289
3	Dessiner des contours 3D	292
4	Dessiner des contours plans	293
5	Position des points de Lagrange	295
5.1	Définition des points de Lagrange	295
5.2	Résolution d'équation par <code>fsolve()</code> et <code>roots()</code>	296
5.3	Retour aux points de Lagrange	297
6	Stabilité du mouvement au voisinage des points de Lagrange	299
7	Mini-projet : La bibliothèque PyQtGraph	301
19	Problème à N corps	303
1	Problème à 3 corps	303
1.1	Résolution sans subtilité	303
1.2	Conservation de l'énergie mécanique	305
2	Réécriture et généralisation au problème à N corps	306
2.1	Réécriture	306
2.2	Solutions chorégraphiques	308
2.3	Mesure du temps d'exécution d'un programme	308
2.4	Génération de nombres pseudo-aléatoires	309
2.5	Augmentation du nombre N de corps	310
3	Mini-projets : animation	311
20	Conclusion	313
21	Documentation officielle	314
	Index	315

Liste des programmes

Prog. 2.1: mon_programme1.py	22	Prog. 3.10: calcullette_astropy1d.py	60
Prog. 2.2: mon_programme2.py	22	Prog. 3.11: codata1.py	61
Prog. 2.3: notations1.py	24	Prog. 3.12: codata2a.py	61
Prog. 2.4: notations2.py	24	Prog. 3.13: codata2b.py	62
Prog. 2.5: notations3.py	25	Prog. 3.14: calcullette_astropy2a.py	62
Prog. 2.6: algebre_boole.py	27	Prog. 3.15: calcullette_astropy2b.py	63
Prog. 2.7: formatage1a.py	30	Prog. 3.16: calcullette_astropy2c.py	63
Prog. 2.8: formatage1b.py	30	Prog. 4.1: enumerate1.py	67
Prog. 2.9: formatage1c.py	30	Prog. 4.2: enumerate2.py	67
Prog. 2.10: formatage1d.py	30	Prog. 4.3: list_comprehension1.py	68
Prog. 2.11: formatage2a.py	31	Prog. 4.4: list_comprehension2.py	68
Prog. 2.12: formatage2b.py	31	Prog. 4.5: list_comprehension3.py	69
Prog. 2.13: formatage2c.py	31	Prog. 4.6: tableaux1c.py	70
Prog. 2.14: formatage2d.py	32	Prog. 4.7: fonction1.py	75
Prog. 2.15: formatage3a.py	32	Prog. 4.8: fonction2.py	75
Prog. 2.16: formatage3b.py	33	Prog. 5.1: graphes1a.py	78
Prog. 2.17: formatage3c.py	33	Prog. 5.2: graphes1a_sauvegarde.py	80
Prog. 2.18: tuples1c.py	36	Prog. 5.3: graphes1b.py	81
Prog. 2.19: tests1a.py	37	Prog. 5.4: graphes1c.py	82
Prog. 2.20: tests1b.py	38	Prog. 5.5: graphes1c2.py	82
Prog. 2.21: tests1c.py	38	Prog. 5.6: graphes1d.py	83
Prog. 2.22: tests1c2.py	39	Prog. 5.7: graphes1e.py	83
Prog. 2.23: tests1d.py	39	Prog. 5.8: graphes1f.py	84
Prog. 2.24: tests1e.py	39	Prog. 5.9: latex.py	85
Prog. 2.25: tests1f.py	40	Prog. 5.10: graphes1f_legende.py	85
Prog. 2.26: tests1a_condense.py	40	Prog. 5.11: graphes1f_legende2.py	86
Prog. 2.27: boucle_while1.py	41	Prog. 5.12: graphes1f2.py	86
Prog. 2.28: boucles1a.py	42	Prog. 5.13: graphes1f3.py	87
Prog. 2.29: boucles1b.py	43	Prog. 5.14: graphes1f4.py	87
Prog. 2.30: boucles1c.py	43	Prog. 5.15: graphes1f5.py	89
Prog. 2.31: boucles1d.py	44	Prog. 5.16: graphes1g.py	89
Prog. 2.32: tuples1f.py	45	Prog. 5.17: graphes1h.py	90
Prog. 2.33: tuples1g.py	45	Prog. 5.18: graphes1h_bis.py	90
Prog. 2.34: temps_chute1.py	46	Prog. 5.19: graphes_sans_cadre.py	91
Prog. 2.35: temps_chute2.py	47	Prog. 5.20: graphes_sans_cadre2.py	91
Prog. 2.36: discriminant2.py	47	Prog. 5.21: graphes_sans_cadre3.py	92
Prog. 2.37: ecrit_ligne.py	48	Prog. 5.22: graphes_echelle_log1.py	93
Prog. 2.38: discriminant1b.py	48	Prog. 5.23: graphes_echelle_log2.py	93
Prog. 2.39: discriminant1c.py	49	Prog. 5.24: graphes3a.py	94
Prog. 2.40: discriminant1d.py	49	Prog. 5.25: graphes3b.py	94
Prog. 2.41: discriminant1e.py	50	Prog. 5.26: graphes3b2.py	95
Prog. 2.42: commentaires.py	50	Prog. 5.27: graphes3c.py	95
Prog. 3.1: calcullette1.py	55	Prog. 5.28: graphes3e.py	96
Prog. 3.2: calcullette1b.py	55	Prog. 5.29: test_linestyle.py	97
Prog. 3.3: calcullette1c.py	56	Prog. 5.30: graphes3g.py	98
Prog. 3.4: calcullette1d.py	56	Prog. 5.31: graphes3c2.py	98
Prog. 3.5: calcullette1e.py	56	Prog. 5.32: graphes3c3.py	99
Prog. 3.6: calcullette1f_scipy.py	58	Prog. 5.33: graphes4a.py	99
Prog. 3.7: calcullette_astropy1a.py	59	Prog. 5.34: graphes4b.py	100
Prog. 3.8: calcullette_astropy1b.py	59	Prog. 5.35: graphes4b2.py	101
Prog. 3.9: calcullette_astropy1c.py	60	Prog. 5.36: graphes4c.py	101

Prog. 5.37:figure_reglage_fin.py	103
Prog. 5.38:figure_reglage_fin2.py	104
Prog. 5.39:histogramme1a.py	104
Prog. 5.40:histogramme1b.py	105
Prog. 5.41:histogramme1c.py	105
Prog. 5.42:histogramme1d.py	106
Prog. 5.43:histogramme1e.py	106
Prog. 5.44:histogramme1f.py	106
Prog. 5.45:test_backend.py	108
Prog. 5.46:style1.py	109
Prog. 5.47:monstyle.mplstyle	110
Prog. 5.48:style2.py	110
Prog. 5.49:style3.py	111
Prog. 5.50:style4.py	111
Prog. 5.51:test_symboles.py	114
Prog. 6.1:dessin1a.py	115
Prog. 6.2:dessin1b.py	115
Prog. 6.3:dessin1c.py	116
Prog. 6.4:dessin1d.py	116
Prog. 6.5:formes1.py	117
Prog. 6.6:formes2a.py	117
Prog. 6.7:formes2b.py	118
Prog. 6.8:pycasso.py	118
Prog. 6.9:formes3.py	119
Prog. 6.10:formes4.py	119
Prog. 6.11:Line2D.py	121
Prog. 6.12:rapporteur.py	122
Prog. 6.13:trace_rapporteur.py	122
Prog. 7.1:refraction1a.py	128
Prog. 7.2:refraction1c.py	129
Prog. 7.3:refraction1d.py	129
Prog. 7.4:refraction2.py	130
Prog. 7.5:evenements1a.py	132
Prog. 7.6:evenements1b.py	132
Prog. 7.7:evenements1c.py	133
Prog. 7.8:evenements1d.py	134
Prog. 7.9:evenements1e.py	134
Prog. 7.10:refraction_event.py	135
Prog. 8.1:raytracing_prisme.py	138
Prog. 8.2:raytracing_prisme2.py	140
Prog. 8.3:raytracing_prisme3.py	141
Prog. 8.4:prisme_deviation_min.py	142
Prog. 8.5:prisme_deviation_courbe.py	144
Prog. 8.6:prisme_deviation_min2.py	145
Prog. 9.1:arc_ordre_zero.py	148
Prog. 9.2:arc_ordre_zero_dense.py	148
Prog. 9.3:arc_primaire.py	149
Prog. 9.4:arc_secondaire.py	152
Prog. 9.5:arc_ordre_N.py	153
Prog. 9.6:curseur.py	155
Prog. 9.7:arc_ordre_zero_slider.py	156
Prog. 9.8:bouton.py	158
Prog. 9.9:bouton_radio.py	159
Prog. 9.10:bouton_check.py	160
Prog. 9.11:arc_exemple_complet.py	160
Prog. 9.12:tortue1.py	163
Prog. 9.13:tortue2.py	164
Prog. 9.14:arc_couleurs_slider.py	165
Prog. 9.15:arc_couleurs_slider.py	166
Prog. 10.1:miroir_spherique.py	168
Prog. 10.2:miroir_spherique2.py	169
Prog. 10.3:diopetre_spherique.py	169
Prog. 10.4:diopetre_spherique_point.py	171
Prog. 10.5:Weierstrass1.py	172
Prog. 10.6:Weierstrass2.py	173
Prog. 10.7:Weierstrass3.py	173
Prog. 10.8:diopetre_elliptique.py	176
Prog. 10.9:diopetre_elliptique_lent.py	177
Prog. 10.10:diopetre_profil.py	178
Prog. 10.11:derivee_numerique1.py	180
Prog. 10.12:derivee_numerique2.py	181
Prog. 10.13:derivee_numerique3.py	181
Prog. 10.14:diopetre_profil2.py	182
Prog. 10.15:profil_vagues.py	182
Prog. 10.16:profil_vagues3.py	183
Prog. 10.17:caustique_miroir_conc.py	185
Prog. 11.1:caustique_miroir_conc1a.py	189
Prog. 11.2:caustique_miroir_conc1c.py	190
Prog. 11.3:relation_conjugaison.py	192
Prog. 11.4:relation_conjugaison2.py	193
Prog. 11.5:miroir_concave_images.py	193
Prog. 11.6:miroir_convexe_images.py	194
Prog. 11.7:souris_miroir_concave.py	195
Prog. 11.8:caustique_miroir_conc1d.py	195
Prog. 11.9:aberrations_scatter1a.py	197
Prog. 11.10:aberrations_histo2d_1.py	199
Prog. 11.11:aberrations_histo2d_2.py	200
Prog. 11.12:aberrations_hexabin.py	201
Prog. 12.1:raytracing_lentille.py	206
Prog. 12.2:raytracing_lentille_div.py	207
Prog. 12.3:classe_exemple.py	208
Prog. 12.4:classes_optique.py	209
Prog. 12.5:lentilles_classe2.py	211
Prog. 12.6:lentilles_classe3.py	212
Prog. 12.7:lentilles_classe4.py	212
Prog. 12.8:lentilles_classe5.py	213
Prog. 12.9:lentilles_classe6.py	215
Prog. 13.1:frottements1.py	222
Prog. 13.2:frottements2a.py	223
Prog. 13.3:frottements2c.py	225
Prog. 13.4:frottements3.py	226
Prog. 13.5:tuto_solve_01.py	228
Prog. 13.6:tuto_solve_02.py	229
Prog. 13.7:tuto_solve_03.py	229
Prog. 13.8:tuto_solve_04.py	230
Prog. 13.9:tuto_solve_05.py	231
Prog. 13.10:tuto_solve_06.py	232
Prog. 13.11:tuto_solve_08.py	233
Prog. 13.12:frottements_solve_ivp1.py	234
Prog. 13.13:frottements_solve_ivp2.py	235

Table des matières / Table des matières

Prog. 13.14:frottements_solve_ivp3.py	235	Prog. 16.3:force_rn_polaire.py	271
Prog. 13.15:portee_frottements.py	237	Prog. 16.4:force_rn_polaire2.py	272
Prog. 14.1:scipy_ode2.py	240	Prog. 16.5:force_rn_lois_aies.py	272
Prog. 14.2:scipy_ode3.py	241	Prog. 17.1:deviation_est.py	277
Prog. 14.3:scipy_ode4.py	242	Prog. 17.2:deviation_est2.py	277
Prog. 14.4:pendule1a.py	244	Prog. 17.3:deviation_est_sud.py	280
Prog. 14.5:pendule1b.py	245	Prog. 18.1:lagrange_surfacela.py	290
Prog. 14.6:pendule2.py	246	Prog. 18.2:lagrange_surfacelb.py	291
Prog. 14.7:pendule3.py	248	Prog. 18.3:lagrange_surfacelc.py	291
Prog. 14.8:exemple_animation.py	250	Prog. 18.4:lagrange_surfaceld.py	292
Prog. 14.9:pendule_animation.py	251	Prog. 18.5:lagrange_surface2.py	292
Prog. 14.10:pendule_animation_save.py	252	Prog. 18.6:lagrange_contours1.py	293
Prog. 15.1:pendule_force1.py	254	Prog. 18.7:lagrange_contours2.py	294
Prog. 15.2:pendule_force_periodes.py	255	Prog. 18.8:lagrange_contours3.py	294
Prog. 15.3:portrait_phase1a.py	256	Prog. 18.9:lagrange_surface2b.py	295
Prog. 15.4:portrait_phase2.py	257	Prog. 18.10:lagrange_contours4.py	298
Prog. 15.5:bifurcation1a.py	258	Prog. 18.11:lagrange_traj.py	299
Prog. 15.6:bifurcation1final.py	260	Prog. 18.12:test_pyqtgraph.py	301
Prog. 15.7:json_write.py	262	Prog. 19.1:probleme_3_corps1.py	303
Prog. 15.8:json_read.py	262	Prog. 19.2:probleme_3_corps2.py	305
Prog. 15.9:bifurcation_write.py	263	Prog. 19.3:probleme_3_corps3.py	306
Prog. 15.10:bifurcation_read.py	263	Prog. 19.4:probleme_N_corps1a.py	306
Prog. 15.11:pickle_write.py	264	Prog. 19.5:choregraphie.py	308
Prog. 15.12:pickle_read.py	264	Prog. 19.6:time1.py	309
Prog. 16.1:force_rn.py	266	Prog. 19.7:probleme_N_corps1c.py	310
Prog. 16.2:force_rn_2b.py	269		



1 Introduction

Cet ouvrage a pour objectif d'accompagner l'apprentissage de Python pour écrire des programmes informatiques présentant un intérêt pour des physiciennes, qu'elles soient étudiantes, enseignantes, chercheuses ou simplement curieuses.¹ Il aborde trois grands champs d'application : le calcul, le graphisme et la simulation. Aucun prétexte pour revenir sur des points de physique ne sera négligé.

1 Python

Python est un langage de programmation possédant plusieurs qualités qui en font un excellent outil pour aborder la physique :

- ▶ Il s'installe facilement sur les systèmes d'exploitation les plus courants (Linux, MacOS, Windows) ;
- ▶ Sa syntaxe est assez naturelle ;
- ▶ Il donne accès à des bibliothèques permettant de tracer des courbes, calculer des intégrales, résoudre des équations différentielles, faire du calcul formel, etc. ;
- ▶ La communauté de développeurs est immense et on trouve beaucoup de réponses sur internet à des problèmes de programmation que l'on peut rencontrer.

L'ambition de cet ouvrage est de rendre cet outil accessible à des débutants en programmation. Ce n'est pas un cours d'informatique, et lorsque des compromis devront être trouvés entre la simplicité de lecture et la propreté du code, c'est le premier critère qui sera favorisé : dans l'idéal, le lecteur sera capable, au bout de quelques chapitres, de s'approprier le langage Python et de le considérer comme un outil puissant, qu'il pourra mettre en œuvre pour explorer des points de physique, que ce soit pour mieux les comprendre soi-même, pour les montrer à des élèves ou à des étudiants, pour réaliser des figures précises ou simplement pour le plaisir de programmer (car oui, ce plaisir existe!).

2 Aborder la physique autrement

Un cours de physique est généralement construit et présenté comme l'enchaînement d'énoncés de lois avec des applications de ces lois, pour comprendre des phénomènes observés ou prédire le résultat d'expériences. Imaginons par exemple qu'on se demande combien de temps met une bille lâchée d'une hauteur initiale donnée pour atteindre le sol. La démarche générale peut se résumer comme suit.

1. Dans la suite, j'alternerai entre le féminin et le masculin pour désigner la lectrice ou le lecteur, sans utiliser l'écriture inclusive, dont l'usage peut vite alourdir un texte utilisant déjà plusieurs typographies. L'ouvrage vise bien évidemment tous les publics de la même façon.

1 Introduction / Aborder la physique autrement

- ▶ On formalise le problème, en définissant le référentiel d'étude, le repère utilisé, le système physique étudié et les forces auxquelles celui-ci est soumis ;
- ▶ On écrit les lois qui gouvernent l'évolution du système, ici la relation fondamentale de la dynamique ;
- ▶ On résout les équations qui en résultent et qui donneraient ici l'altitude en fonction du temps ;
- ▶ On répond à la question que l'on se posait, ici on cherche la valeur du temps pour laquelle l'altitude est nulle.

On commence souvent par se concentrer sur des cas où les équations peuvent être résolues analytiquement, exactement. Ceci se fait au prix d'une simplification du problème initial, par exemple :

- ▶ On néglige les frottements, ou on considère que le système est parfaitement sphérique pour pouvoir appliquer une expression connue pour les forces de frottements ;
- ▶ On considère que l'accélération de la pesanteur est constante ;
- ▶ On considère que la Terre est un référentiel galiléen.

Ce faisant, dans certains cas (celui de la chute dans l'air est typique), on en vient à résoudre de manière exacte un problème qui ne correspond qu'approximativement au réel. Le paragraphe précédent s'appliquerait à des exemples issus d'autres branches de la physique que la mécanique. On réserve les cas moins idéalisés à « plus tard » et il n'est pas rare que ce « plus tard » ne vienne jamais, par manque de temps en cours et l'étudiant peut avoir l'impression que la physique se limite à l'étude de situations tellement idéalisées qu'elles perdent leur intérêt pour décrire le monde qui nous entoure.

Or, l'informatique permet aujourd'hui d'envisager une approche complémentaire, en étudiant des problèmes plus réalistes, en remplaçant l'étape de résolution analytique par une résolution numérique ou une modélisation numérique. Cette approche ne doit pas se substituer à la précédente mais la compléter. Suivant la sensibilité de l'enseignant et du public étudiant, on peut envisager de commencer par l'approche numérique.

La mécanique et l'optique géométrique se prêtent remarquablement bien à une approche numérique, pour des raisons différentes. Dans le cas de l'optique géométrique, la tâche purement graphique consistant à tracer des rayons lumineux est prise en charge de façon beaucoup plus efficace par un ordinateur que par une personne munie d'une règle et d'un rapporteur. L'informatique permet également une interactivité qui peut être précieuse d'un point de vue pédagogique : modifier de façon continue l'indice de réfraction d'un milieu, avec un curseur, pour voir l'effet sur les rayons réfractés, est beaucoup plus parlant que de faire à la main plusieurs dessins de suite. Dans le cas de la mécanique, l'informatique permet de mieux ancrer, dans l'esprit des étudiants, la distinction entre l'étape de formalisation et celle de résolution. Ce n'est pas parce qu'on ne sait pas résoudre mathématiquement une équation décrivant un système physique que l'on ne peut pas faire de physique du tout.

Il ne s'agit pas de remplacer toute réflexion analytique par de la programmation : nous verrons tout au long de cet ouvrage qu'au contraire, des raisonnements mathéma-

1 Introduction / Plan de l'ouvrage

tiques solides permettent de programmer de façon plus efficace et précise (voir la **règle 2** à la fin de cette introduction).

3 Plan de l'ouvrage

L'ouvrage est découpé en trois parties. La première est dédiée à la présentation générale de Python et à quelques premières applications : utilisation de Python comme calculatrice, gestion des unités, tracé de points de données sur un graphe, tracé de courbes. La seconde est consacrée à l'optique géométrique, en donnant des outils pour tracer des rayons de façon rapide et obtenir des graphes interactifs qui peuvent être utilisés aussi bien pour enseigner que pour apprendre la matière. La troisième est consacrée à la mécanique du point, en s'intéressant à des problèmes rarement abordés dans des introductions parce qu'ils conduisent à des équations non solubles analytiquement. La plupart des chapitres proposent des activités finales sous la forme de mini-projets. Il s'agit la plupart de temps d'adapter les programmes présentés dans le chapitre. Des « solutions » sont proposées, qu'il faut prendre comme des propositions pour répondre à la consigne, mais il existe souvent d'autres manières de le faire.

Des notions de Python sont introduites au fur et à mesure, pour les appliquer à des problématiques liées à la physique :

- ▶ La création et la gestion de styles matplotlib prédéfinis, au chapitre 5 ;
- ▶ La gestion des événements (clics de la souris, déplacement d'un objet à la souris), au chapitre 7 ;
- ▶ La gestion des widgets (éléments interactifs tels que les curseurs et les boutons cliquables), au chapitre 9 ;
- ▶ Les calculs numériques de dérivées, au chapitre 10 ;
- ▶ Le tracé de cartes de densité, au chapitre 11 ;
- ▶ La création de nouvelles classes, au chapitre 12 ;
- ▶ La résolution numérique des équations différentielle, introduite au chapitre 13 et réutilisée ensuite ;
- ▶ La gestion des animations, introduite au chapitre 14 et réutilisée ensuite ;
- ▶ La gestion des entrées/sorties (sauvegarde de données dans des fichiers, utilisation de json et pickle), au chapitre 14 ;
- ▶ Le calcul symbolique avec SymPy au chapitre 17 ;
- ▶ Le tracé de surfaces bidimensionnelles, au chapitre 18 ;
- ▶ La résolution numérique d'équations algébriques, au chapitre 18 ;
- ▶ La génération de nombres pseudo-aléatoires, au chapitre 19 ;
- ▶ La mesure du temps d'exécution des programmes, au chapitre 19.

1 Introduction / Règles d'or

4 Règles d'or

Règle 1 Écrivez les programmes de façon séquentielle, en commençant par écrire un squelette qui fait le minimum, puis en l'améliorant progressivement, en vous assurant à chaque étape que la version améliorée continue à faire correctement ce que faisaient les versions précédentes. Pour cela, faites de nombreux tests et surtout, pensez à sauvegarder les versions successives. Rien n'est plus frustrant que de chercher à améliorer un programme qui fonctionne, pour s'apercevoir que ça ne fonctionne plus et ne pas arriver à revenir à la version antérieure.

Règle 2 Modéliser et résoudre un problème de physique par des méthodes numériques ne dispense pas de l'aborder du point de vue analytique, bien au contraire : plus on est capable de mettre à jour des propriétés analytiques, plus la programmation sera simple et efficace. Dans l'idéal, on sait résoudre le problème de façon analytique dans certaines conditions, ce qui permet de vérifier que l'approche numérique redonne la bonne solution dans ces conditions.

Règle 3 Testez les programmes, en profondeur, en modifiant les paramètres qui interviennent dans le problème étudié. N'hésitez pas à écrire, avec des commandes `print()`, le contenu des variables au cours de l'exécution du programme, dans les phases de développement, quitte à les supprimer ou les commenter quand le programme est au point. Assurez-vous également que vous connaissez le type de toutes les variables que vous utilisez (la commande `type()` que nous introduirons plus loin peut vous aider).

Règle 4 Appuyez sur tous les boutons! N'hésitez pas à changer des commandes, des paramètres, dans les programmes présentés dans cet ouvrage, pour comprendre ce qu'ils font.

5 Remerciements

Pour terminer cette introduction, je voudrais rendre hommage aux personnes qui ont accepté de passer du temps et de l'énergie à relire cet ouvrage et à en tester les programmes. En particulier, un grand merci à Bertrand Aigouy, Pierre Noël, Éric Pagot et Laurence Perotto. Bien entendu, je porte l'entière responsabilité de toute erreur ou confusion qui pourrait subsister.

6 Télécharger les programmes

L'intégralité des programmes proposés dans cet ouvrage, ainsi que des propositions de solutions pour les mini-projets de fin de chapitre, sont téléchargeable à l'adresse www.deboecksuperieur.com/site/328907 sous la forme d'un fichier .zip.

Première partie

Python

La première partie de cet ouvrage présente les bases de Python, sa syntaxe, quelques-uns des types de variables qu'il propose (chapitres 2 et 4). Le chapitre 3 se présente comme un intermède, appliquant des notions vues au chapitre 2 pour gérer les unités et les dimensions physiques. Les chapitres 5 et 6 proposent une initiation détaillée à la production de graphes et de dessins, dont on aura besoin dans toute la suite.



2 Fonctions de base

1 Démarrer et quitter

Cet ouvrage s'appuie sur la version 3.7 de Python et englobe toutes les versions 3.X. Certaines syntaxes et certains comportements peuvent différer si vous utilisez une version 2.X. Nous vous laissons le soin d'installer le programme, par exemple sa distribution Anaconda,¹ complète et facile à installer. Nous avons choisi comme départ de cet ouvrage le point où Python est installé et où vous pouvez l'exécuter, depuis une fenêtre de type terminal accessible de la façon suivante :

- ▶ Sous Windows, le lancement du programme Anaconda propose un menu graphique, dans lequel `Command.exe` donne accès à ce terminal. On peut aussi y accéder en cherchant « invite de commande » dans le menu « Démarrer » ;
- ▶ Sous MacOS, le terminal est accessible dans « Applications/Utilitaires/ ».

Dans ce terminal, la commande `python` donne le résultat suivant sur l'ordinateur de l'auteur :

Console

```
$ python
Python 3.7.6 (default, Jan 8 2020, 13:42:34)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information
>>>
```

avec un curseur qui suit les caractères `>>>` , le programme attend une instruction à exécuter, après avoir donné quelques informations sur la version de Python utilisée (le message peut être différent, suivant la version de Python installée et le système d'exploitation que vous utilisez). Le caractère « \$ » qui précède `python` est le « prompt », le symbole du système d'exploitation qui invite à entrer une commande (il peut être différent sur le vôtre, en particulier « > » sous Windows), les caractères « `>>>` » constituent le prompt de Python.

La première commande qu'on peut avoir envie de lancer, pour se rassurer, est celle qui permet de quitter le programme :

Console

```
>>> exit()
```

à l'issue de laquelle l'utilisateur a quitté Python et retrouve le prompt de sa console. Notons que la commande `quit()` conduit au même résultat. On peut aussi utiliser « Control-D », c'est à dire appuyer sur la touche D pendant que la touche « control » (ou Ctrl) est enfoncée.

1. Disponible à l'adresse <https://www.anaconda.com/distribution/>.

2 Fonctions de base / Calculatrice en mode interactif

Environnements plus élaborés

Il est aussi possible d'installer IPython, une version plus interactive de Python que nous présenterons à la fin de ce chapitre. Par ailleurs, plusieurs environnements de développement en Python, comme IDLE ou Spyder, intègrent un éditeur, une fenêtre de terminal et d'autres outils de développement. Enfin, l'environnement Jupyter permet d'intégrer des programmes en Python, sur un navigateur internet, dans des bloc-notes qui mêlent texte et blocs de programmes pouvant être exécutés indépendamment. Nous allons rester, dans cet ouvrage, sur du Python pur lancé depuis la fenêtre de terminal. Nous vous invitons à créer un répertoire dédié à Python, dans lequel vous placerez les programmes proposés et ceux que vous écrirez. Ils devront être exécutés depuis ce répertoire.

2 Calculatrice en mode interactif

Relançons Python pour lancer une commande (marginale) plus intéressante, qui calcule le produit de deux nombres

Console

```
>>> 8*9
72
>>>
```

Python effectue le calcul demandé, puis redonne la main à l'utilisateur en attendant une nouvelle commande. À ce stade, vous voici en possession d'une calculatrice élémentaire, qui peut réaliser quelques opérations de base (addition +, soustraction -, multiplication * et division /, ainsi que la puissance **), avec les parenthèses pour indiquer les priorités usuelles, par exemple

Console

```
>>> 3*(5+2) - (0.5/2)**3
20.984375
>>>
```

où $3*(5+2)$ commence par calculer $5+2$ puis le multiplie par 3, et où $(0.5/2)**3$ calcule la quantité $(0.5/2)$ puis l'élève à la puissance 3. La puissance peut aussi être écrite avec la fonction `pow()` qui prend deux arguments séparés par une virgule, le premier indiquant le nombre à élever à la puissance voulue, le second la valeur de cette puissance (voir le paragraphe 3)

Console

```
>>> 3*(5+2) - pow(0.5/2, 3)
20.984375
>>>
```

Conformément aux conventions anglo-saxonnes, c'est un point décimal qui est utilisé dans les nombres dits « à virgule ». On peut utiliser la notation scientifique, par exemple le nombre $6,67 \times 10^{-10}$ peut être écrit sous les formes suivantes

2 Fonctions de base / Fonctions et paramètres

Console

```
>>> 6.67*pow(10,-10)
6.67e-10
>>> 6.67E-10
6.67e-10
>>> 0.000000000667
6.67e-10
>>> 6.67e-10
6.67e-10
```

Python propose aussi deux opérations moins courantes sur des calculettes : la division entière « // » et le reste par la division entière « % ». Ainsi,

Console

```
>>> 36//16
2
>>> 36%16
4
```

En effet, $36 = 2 \times 16 + 4$. Ces deux opérations se généralisent à des valeurs non entières,

Console

```
>>> 7//1.5
4.0
>>> 7%1.5
1.0
```

car $7 = 4 \times 1,5 + 1$.

3 Fonctions et paramètres

Le langage Python comporte de nombreuses **fonctions** : des objets qui effectuent une action ou qui renvoient une valeur. L'action ou la valeur renvoyée peut dépendre de paramètres que l'on spécifie dans la parenthèse (on les appelle les **arguments** de la fonction). Par exemple, la fonction `pow()` demande deux arguments et renvoie une valeur obtenue en mettant le premier à la puissance donnée par le second. Ainsi, la commande `pow(2, 3)` renvoie la valeur 8, égale à 2^3 . Nous verrons plus loin comment programmer de nouvelles fonctions.

4 Bibliothèques

Pour accéder à des fonctions mathématiques plus évoluées, il faut inclure une bibliothèque² mathématique, par exemple `math.py` (il y en a d'autres, nous y reviendrons), de la façon suivante

2. On utilise aussi parfois le terme de « librairie », mauvaise traduction de library, « package » ou « module ».

2 Fonctions de base / Bibliothèques

```
import math
```

ce qui donne accès à de nombreuses fonctions mathématiques,³ par exemple, en notant x la variable passée en argument⁴ :

- ▶ `sqrt(x)` : la racine carrée ;
- ▶ `exp(x)` : la fonction exponentielle ;
- ▶ `log(x)` et `log10(x)` : le logarithme népérien et le logarithme décimal. Noter que la fonction `log()` avec deux arguments permet d'accéder au logarithme dans n'importe quelle base : `log(7, 3)` renvoie le logarithme en base 3 de 7, c'est-à-dire $\ln(7)/\ln(3)$;
- ▶ `erf(x)` : la fonction erreur, définie comme l'intégrale d'une gaussienne ;
- ▶ `cos(x)`, `sin(x)`, `tan(x)` : les fonctions trigonométriques usuelles, dont l'argument x doit être exprimé en radians ;
- ▶ `acos(x)`, `asin(x)`, `atan(x)` : leurs fonctions réciproques, renvoyant des angles en radians (attention, ces fonctions portent des noms différents dans la bibliothèque `numpy` introduite plus loin) ;
- ▶ `cosh(x)`, `sinh(x)`, `tanh(x)` : les fonctions hyperboliques ;
- ▶ `acosh(x)`, `asinh(x)`, `atanh()` : leurs fonctions réciproques (attention, ces fonctions portent des noms différents dans la bibliothèque `numpy`) ;
- ▶ `degrees(x)` : convertit un angle des radians vers les degrés.

Elles s'utilisent de la façon suivante

Console

```
>>> import math
>>> math.sqrt(7)
2.6457513110645907
>>> 4*math.atan(1)
3.141592653589793
>>>
```

où la syntaxe `math.sqrt(7)` indique que Python doit aller chercher la fonction `sqrt` fournie par la bibliothèque `math`, celle-ci devant avoir été préalablement importée par `import math`. Cette syntaxe peut sembler lourde, on peut l'alléger légèrement par la commande

```
import math as m
```

qui définit un raccourci que l'on utilise ainsi :

3. La liste complète peut être consultée ici : <https://docs.python.org/3/library/math.html>

4. Dans toute la suite, nous dédierons une couleur aux fonctions fournies par des bibliothèques externes, `math`, `numpy`, `scipy` ou `matplotlib`.

2 Fonctions de base / Bibliothèques

Console

```
>>> import math as m
>>> m.sqrt(7)
2.6457513110645907
>>> 4*m.atan(1)
3.141592653589793
>>>
```

On peut alléger encore la notation en important explicitement les fonctions souhaitées par

```
from math import sqrt, atan
```

ce qui permet à l'interpréteur de prendre connaissance des fonctions `sqrt` et `atan` (et uniquement celles-ci),

Console

```
>>> from math import sqrt, atan
>>> sqrt(7)
2.6457513110645907
>>> 4*atan(1)
3.141592653589793
>>>
```

On peut importer d'un coup toutes les fonctions définies par le package `math` par

```
from math import *
```

ce qui permet de les utiliser comme suit

Console

```
>>> from math import *
>>> sqrt(7)
2.6457513110645907
>>> 4*atan(1)
3.141592653589793
>>>
```

La bibliothèque `math` fournit également les constantes mathématiques π et e :

Console

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>>
```

Attention, si vous importez la bibliothèque mathématique avec `from math import *`, il n'est plus possible de choisir `pi` ni `e` comme noms de variables, ce sont alors des noms réservés.

2 Fonctions de base / Écriture de programmes

Lorsque vous quittez Python avec `exit()`, celui-ci ne retient rien des opérations que vous avez effectuées et si vous le relancez, vous devrez réimporter la librairie `math` pour avoir de nouveau accès à ces fonctions mathématiques.

5 Écriture de programmes

Cette utilisation interactive de Python s'avère assez rapidement limitée et dans cet ouvrage nous préférons écrire les commandes à exécuter dans un fichier (on les appelle des **programmes** ou des **scripts** et on leur donne l'extension `.py` pour se rappeler qu'il s'agit de Python), grâce à un éditeur de texte. Attention, il est déconseillé d'utiliser un logiciel de traitement de texte, tel que Word, Libre Office ou Google Doc, car ceux-ci rajoutent des caractères de mise en page⁵ qui ne seront pas compris par Python : les fichiers de programme ne devront contenir que du texte pur. La plupart des éditeurs de texte⁶ permettent de coloriser automatiquement le code Python pour mieux faire apparaître leur structure, en affectant des couleurs aux mots-clés.

Commençons par un programme simple, un fichier `mon_programme.py` contenant une seule ligne de texte. Ce fichier doit se trouver dans le même répertoire que celui d'où est lancée la commande. L'idéal est de créer des sous-répertoires distincts pour chacun des projets que vous développerez.

Prog. 2.1 Fichier : `mon_programme1.py`

```
1 | 3*7
```

On lance l'exécution du programme par la commande

Console

```
$ python mon_programme1.py
$
```

ce qui ne fait... rien, à part rendre la main à l'utilisateur ! Il faut en effet demander explicitement à Python d'écrire le résultat du calcul qui nous intéresse, en utilisant la commande `print()` qui s'utilise comme suit

Prog. 2.2 Fichier : `mon_programme2.py`

```
1 | print(3*7)
```

ce qui donne bien le résultat escompté

Console

```
$ python mon_programme2.py
21
$
```

5. sauf si on prend soin d'enregistrer en format texte.

6. J'utilise Emacs, Nedit, Atom et Xcode, il en existe beaucoup d'autres.

2 Fonctions de base / Variables

L'utilisateur n'a pas la main pendant l'exécution, il écrit à l'avance les commandes à exécuter par Python, qui le fait puis rend la main à l'utilisateur (pas besoin d'écrire `exit()` à la fin). Il faut explicitement utiliser la commande `print()` pour faire écrire le résultat d'un calcul (ou plus souvent, le contenu d'une variable, nous y reviendrons en détail plus loin). Ce comportement peut sembler lourd au premier abord, mais vous verrez que lorsqu'on commence à écrire des programmes enchaînant plusieurs opérations, on souhaite précisément ne pas être submergé par les indications de ce que donnent les calculs intermédiaires !

Dans la suite de l'ouvrage, nous n'utiliserons que rarement les commandes en ligne, les réservant à de très courts enchaînements de commandes.

6 Variables

On est souvent amené à vouloir garder en mémoire le résultat de certaines opérations pour les réutiliser dans un autre calcul. C'est ce que permettent de faire les **variables**, des objets ayant des noms librement choisis par l'utilisateur (tant qu'il ne s'agit pas du nom d'une commande de Python), auxquels on affecte une valeur de la façon suivante :

Console

```
>>> toto=0.1
```

La commande `=` réalise ici une **affectation**, c'est-à-dire qu'elle attribue la valeur 0,1 à la variable `toto`. Dans les lignes qui suivent l'affectation, à chaque fois que `toto` est utilisé dans le programme, c'est sa valeur courante qui est utilisée, par exemple

Console

```
>>> toto=0.1
>>> print(toto)
0.1
>>> print(toto+1)
1.1
```

Le premier `print` écrit la valeur de la variable `toto`, le second écrit celle de `toto+1`. La valeur de la variable peut être changée en cours de programme. Par exemple

Console

```
>>> toto=0.1
>>> print(toto)
0.1
>>> toto = 1+toto
>>> print(toto)
1.1
```

À chaque fois que la variable `toto` est utilisée dans le programme, c'est sa valeur courante qui est utilisée. Au moment du premier `print(toto)`, `toto` vaut 0.1, puis au second il vaut 1.1.

2 Fonctions de base / Variables

Attention, l'affectation d'une valeur à une variable peut être perturbante, en particulier la ligne `toto = 1+toto` serait une aberration si le signe `=` était interprété comme une égalité mathématique. Or, ce signe `=` a une signification très différente en programmation, il provoque une action, en affectant une valeur à une variable (voir le paragraphe 7 pour une discussion plus poussée sur ce point) : ce qui se trouve après le signe `=` est calculé, puis affecté à la variable qui le précède. On obtiendrait une erreur en écrivant `1+toto = toto`, l'ordinateur ne sait pas affecter la valeur contenue dans `toto` à `1+toto` car `1+toto` n'est pas une variable.

Les noms de variables obéissent aux deux règles suivantes :

- ▶ ils ne peuvent contenir que les caractères suivants : lettres non accentuées, minuscules ou majuscules, et chiffres de 0 à 9, ainsi que le caractère `_` (« underscore » ou « tiret du 8 »). Les minuscules et les majuscules sont considérées comme des caractères différents : la variable `delta` est distincte de la variable `DeLta`.
- ▶ ils ne peuvent pas commencer par un chiffre.

Par exemple, Les noms `distance_AB` et `compteur1` sont valides, mais `distance-AB` ne l'est pas (le caractère `-` est réservé pour l'opérateur de soustraction), `distance AB` non plus (l'espace n'est pas un caractère autorisé), `1compteur` non plus (le nom ne doit pas commencer par un chiffre).

Toto et tatayoyo

Le choix du nom de variable dans l'exemple précédent illustre très bien une mauvaise pratique courante. Il est fortement conseillé de donner des noms explicites aux variables, pour rendre les programmes plus lisibles. Pour prendre un exemple élémentaire, pour un physicien le programme

Prog. 2.3

Fichier : `notations1.py`

```
1 g = 9.81
2 t = 2
3 d = 0.5*g*pow(t,2)
4 print(d)
```

est beaucoup plus lisible, pour calculer la distance parcourue pendant une chute libre de 2 secondes, que les programmes suivants, pourtant exactement équivalents du point de vue informatique

Prog. 2.4

Fichier : `notations2.py`

```
1 a = 9.81
2 b = 2
3 c = 0.5*a*pow(b,2)
4 print(c)
```

ou

2 Fonctions de base / Variables

Fichier : notations3.py

Prog. 2.5

```

1 toto1 = 9.81
2 tatayoyo = 2
3 bof = 0.5*toto1*pow(tatayoyo,2)
4 print(bof)

```

Notations raccourcies

Python propose des notations raccourcies pour certaines opérations mathématiques effectuées sur une variable, notée x dans la liste suivante :

- | | |
|--|---------------------------|
| ▶ $x += 3$ ajoute 3 à la variable x | Équivalent à $x = x+3$; |
| ▶ $x -= 3$ retranche 3 à la variable x . | Équivalent à $x = x-3$; |
| ▶ $x *= 3$ multiplie la variable x par 3. | Équivalent à $x = x*3$; |
| ▶ $x /= 3$ divise la variable x par 3. | Équivalent à $x = x/3$; |
| ▶ $x \% = 3$ renvoie le reste de la division de x par 3. | Équivalent à $x = x\%3$; |
| ▶ $x //= 3$ renvoie la division entière de x par 3. | Équivalent à $x = x//3$. |

Types

Il existe plusieurs types de variables.⁷ Dans les exemples précédents, nous avons rencontré des nombres à virgule (appelés des **float**, car la virgule est souvent qualifiée de « flottante ») et des entiers (des **int**). Lorsqu'on affecte une valeur numérique à une variable, il s'agit d'un **float** lorsque le nombre contient un point décimal (la virgule des nombres à virgules) et d'un **int** sinon. L'exemple suivant crée un **int** :

Console

```

>>>var = 4
>>>print(var*var)
16

```

Par défaut, Python écrit les **float** avec un point décimal et les **int** sans. On peut aussi indiquer à Python que la variable doit contenir un **float**, même si le nombre est mathématiquement un entier, en la définissant par l'une des trois commandes suivantes, avec un point décimal ou par la commande **float()** :

Console

```

>>>toto=4.0
>>>toto=4.
>>>toto=float(4)

```

La fonction **type()** permet de connaître le type d'une variable. Par exemple

⁷ La liste complète des types de base est disponible dans la documentation en ligne, à l'adresse <https://docs.python.org/3/library/datatypes.html>.

2 Fonctions de base / Variables

Console

```
>>>x = 1.  
>>>print(type(x))  
<class 'float'>  
>>> x=1  
>>> type(x)  
<class 'int'>
```

Les variables peuvent également être des chaînes de caractères⁸ (des « strings »), de type `str`, dont la valeur est comprise entre apostrophes simples (symbole `'`) ou doubles (symbole `"`)

Console

```
>>>machaine="bonjour et bienvenue"  
>>>print(machaine)  
bonjour et bienvenue
```

On aurait pu écrire de façon équivalente

Console

```
>>>machaine='bonjour et bienvenue'
```

Une chaîne de caractères définie entre des apostrophes doubles `"` ne doit pas elle-même contenir d'apostrophes doubles, et vice-versa pour les apostrophes simples. Dans cet ouvrage, nous utiliserons les apostrophes simples pour des chaînes de caractères représentant des paramètres bien définis en Python (par exemple `plt.axis('off')` pour supprimer les axes d'un graphe, voir le chapitre 5), et les apostrophes doubles pour du texte libre (par exemple `machaine="bonjour"`).

Notons quelques caractères spéciaux, `\n` pour le retour à la ligne, `\t` pour une tabulation.

Console

```
>>> machaine="bonjour\nà\ttous\net \tbienvenue"  
>>> print(machaine)  
bonjour  
à tous  
et bienvenue
```

Python sait faire de nombreuses opérations sur des chaînes de caractères, notons par exemple l'addition (qui réalise en fait une concaténation) et la multiplication par un entier (qui réalise une concaténation multiple)

Console

```
>>>chaine1 = "A"  
>>>chaine2 = "B"  
>>>print(chaine1+chaine2)  
AB  
>>>print(3*chaine1)  
AAA
```

8. Attention nous parlons ici du contenu des variables, pas du nom des variables.

2 Fonctions de base / Variables

Remarquez la différence entre le résultat des deux commandes `print` des commandes suivantes

Console

```
>>>a = 2
>>>b = 1
>>>print(10*"a"+3*"b")
aaaaaaaaabb
>>>print(10*a+3*b)
23
```

où `a` désigne une variable, alors que `"a"` désigne une chaîne de caractères contenant la lettre « a ». L'argument du premier `print()` est une chaîne de caractères tandis que celui du second est un `int`, résultat de l'opération $10*a+3*b$.

Citons enfin, parmi les autres types de variables disponibles, les booléens (`bool`) qui peuvent prendre deux valeurs, `True` ou `False`. Le programme suivant affecte une valeur `True` ou `False` à la variable booléenne `mon_test` puis affiche sa valeur.

Console

```
>>> mon_test = (3>2)
>>> print(mon_test)
True
```

Cet exemple permet d'introduire la notion de test : la variable `mon_test` contient ici le résultat du test $3>2$. L'inégalité étant vraie, la variable contient la valeur `True`. Vous pourrez vérifier qu'en changeant le signe de l'inégalité, on obtient la valeur `False`.

Les booléens sont ainsi nommés car ils obéissent à l'algèbre de Boole, avec les opérations `and`, `or` et `not` que montrent le programme suivant

Fichier : `algreboole.py`

Prog. 2.6

```
1 vrai = True
2 faux = False
3
4 print("True and True :", vrai and vrai)
5 print("True and False :", vrai and faux)
6 print("False and False :", faux and faux)
7 print("True or True :", vrai or vrai)
8 print("True or False :", vrai or faux)
9 print("False or False :", faux or faux)
10 print("not True :", not vrai)
11 print("not False :", not faux)
```

qui produit

Console

```
$ python algreboole.py
True and True : True
True and False : False
False and False : False
```

2 Fonctions de base / Variables

```
True or True : True
True or False : True
False or False : False
not True : False
not False : True
```

Ces opérations joueront un rôle important lorsque l'on en viendra aux branchements conditionnels ([if](#)).

Convertir d'un type à un autre

On peut réaliser des conversion d'un type de variable à un autre, par exemple à partir d'un [int](#), on peut obtenir un [float](#) ou un [str](#) :

Console

```
$ python
>>> a=2
>>> float(a)
2.0
>>> str(a)
'2'
```

À partir d'un [float](#), on peut obtenir un [int](#) ou un [str](#) :

Console

```
$ python
>>> a=2.3
>>> str(a)
'2.3'
>>> int(a)
2
```

À partir d'un [str](#), on peut obtenir un [int](#) ou un [float](#) :

Console

```
$ python
>>> a="32"
>>> int(a)
32
>>> float(a)
32.0
```

Bien sûr, ce mécanisme a ses limites, la conversion d'une chaîne de caractères alphanumériques en [int](#) peut produire une erreur.

Console

```
$ python
>>> a="bonjour"
>>> int(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'bonjour'
```

Combiner les types

Python permet d'effectuer des opérations entre certains objets de types différents. Par exemple, on peut ajouter, soustraire, multiplier ou diviser un `int` et un `float`, on obtient alors un `float`.

Console

```
>>> a = 1
>>> b = 0.5
>>> c = a+b
>>> type(a), type(b), type(c)
(<class 'int'>, <class 'float'>, <class 'float'>)
```

Ceci est très naturel, puisqu'en général, la multiplication d'un nombre à virgule par un entier donne un nombre à virgule. Vous pourrez tester d'autres combinaisons. Implicitement, Python a converti un type en un autre avant de faire l'opération, ici un `int` en `float`.

Certaines opérations ne sont pas possibles, par exemple ajouter un `int` ou un `float` à une chaîne de caractère `str`,

Console

```
>>> a = 1
>>> b = "bonjour"
>>> c = a+b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

l'opération génère un message d'erreur. En revanche, on peut multiplier un `str` par un `int`, avec un résultat plutôt naturel, comme on l'a déjà vu plus haut,

Console

```
>>> a = 4
>>> b = "bonjour"
>>> c = a*b
>>> print(c)
bonjourbonjourbonjourbonjour
```

7 Afficher

Python propose plusieurs solutions pour imprimer du texte et des variables. Elles utilisent toutes la fonction `print()`.

Formatage 1

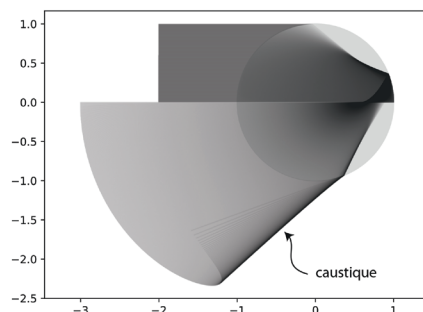
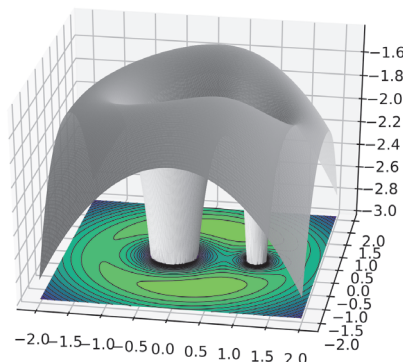
La commande `print()` peut contenir autant d'arguments que l'on veut, séparés par des virgules. Le code suivant affiche les valeurs de deux variables `res1` et `res2`, après leur avoir affecté des valeurs.

PYTHON POUR LA PHYSIQUE

Calcul, graphisme, simulation

Apprendre les bases de Python pour faire du calcul scientifique, des simulations numériques, produire des courbes de qualité, des graphes interactifs, des animations, voilà ce que propose cet ouvrage.

En 250 programmes courts, il s'appuie sur des exemples empruntés à la physique de première année de licence scientifique : l'optique géométrique et la mécanique du point. Sur ce premier thème, il montre comment tracer des rayons lumineux de façon rapide et obtenir des graphes interactifs qui peuvent être utilisés aussi bien pour enseigner que pour mieux comprendre la matière. En ce qui concerne le second thème, il aborde des problèmes rarement traités dans des cours d'introduction à la mécanique (chaos, problème à N corps) parce qu'ils conduisent à des équations non solubles analytiquement. Chaque chapitre se termine par des mini-projets corrigés.



Richard Taillet est enseignant-chercheur en physique à l'Université Savoie Mont Blanc et effectue son activité de recherche en astrophysique des particules au Laboratoire d'Annecy-le-Vieux de Physique Théorique. Il est impliqué dans de nombreuses activités de diffusion de la connaissance, au niveau universitaire et vers le grand public. Il a publié plusieurs ouvrages en physique chez De Boeck Supérieur.

ISBN : 978-2-8073-2890-7



9 782807 328907

deboeck
SUPÉRIEUR

www.deboecksuperieur.com