Programmer avec Dr.Geo 19.09

Faire des mathématiques en programmant

Hilaire Fernandes (hilaire@drgeo.eu)

Ce manuel est pour GNU Dr.Geo (version 19.09), un environnement de géométrie interactive et de programmation.

Compilation : 1 septembre 2019

Source de la documentation : `https://bazaar.launchpad.net/~drgeo-developers/drgeo`

Merci aux relecteurs de ce manuel : Alain Busser, Aubin Jarry, Georges Khaznadar.

Sommaire

1 Introduction 1

Partie I Pour bien démarrer

2 La syntaxe par l'exemple 5

Partie II Programmer au secondaire I

3 Nombres et opérations 27
4 Espace 38
5 Fonctions 76

Partie III Pour aller plus loin dans la programmation

6 Figure Pharo 83
7 Outils du développeur 100

Partie IV Annexes

A GNU Free Documentation License 113
B Copyright des documents 117
C Liste des exercices 118
D Solutions des exercices 121
E Liste des exemples 155
F Liste des figures 157
G Aides 159
H Index conceptuel 163
I Index des méthodes Dr.Geo 165

Table des matières

1 Introduction 1
1.1 À propos de Dr.Geo 1
1.2 À propos de ce livre 1
1.3 Contenu 1

Partie I Pour bien démarrer

2 La syntaxe par l'exemple 5
2.1 La syntaxe, c'est quoi ? 5
2.2 Où saisir le code ? 5
2.3 Hermès, messager des dieux 6
2.3.1 Message unaire 6
2.3.2 Message à mot clé 7
2.3.3 Message binaire 8
2.3.4 Priorités des messages 9
2.4 Cascade de messages 10
2.5 Les variables 12
2.6 Commentaire 15
2.7 Collection 15
2.8 Bloc de code 16
2.9 Structures de contrôle 19
2.9.1 Boucle 20
2.9.2 Test 22

Partie II Programmer au secondaire I

3 Nombres et opérations 27
3.1 Ensembles de nombres 27
3.1.1 Les entiers 27
3.1.2 Les décimaux 27
3.1.3 Les rationnels 28
3.2 Opérations 29
3.2.1 Priorité des opérations 29
3.2.2 Calcul fractionnaire 30
3.2.3 Division euclidienne 31
3.3 Nombres naturels 31
3.3.1 Multiples et diviseurs 31
3.3.2 Diviseurs d'un nombre naturel 33
3.3.3 Diviseurs communs 34
3.3.4 Plus grand diviseur commun 35
3.3.5 Nombre premier 35

4 Espace 38
4.1 Droites 38
4.1.1 Parallèles 38
4.1.2 Perpendiculaires 38

4.1.3 Distance . . . 40
4.1.3.1 Distance entre deux points . . . 40
4.1.3.2 Distance d'un point à une droite . . . 40
4.1.3.3 Distance entre deux droites parallèles . . . 41
4.2 Quadrilatères . . . 43
4.2.1 Parallélogramme . . . 43
Par les côtés opposés . . . 43
Par les diagonales . . . 45
4.2.2 Losange . . . 46
Troisième sommet . . . 46
Par les côtés opposés . . . 46
Par les diagonales . . . 47
4.2.3 Rectangle . . . 47
Troisième sommet . . . 47
Par les côtés opposés . . . 48
Par les diagonales . . . 48
4.2.4 Carré . . . 48
Troisième sommet . . . 49
Par les côtés opposés . . . 49
Par les diagonales . . . 49
4.3 Triangles . . . 50
4.3.1 Isocèle . . . 50
Côtés isométriques . . . 50
Axe de symétrie . . . 51
Angles isométriques . . . 51
4.3.2 Équilatéral . . . 52
Côtés isométriques . . . 52
Axes de symétrie . . . 53
4.3.3 Triangle rectangle . . . 53
4.3.4 Triangle rectangle isocèle . . . 53
4.4 Droites remarquables du triangle . . . 53
4.4.1 Médiatrices . . . 54
4.4.2 Bissectrices . . . 54
4.4.3 Hauteurs . . . 55
4.4.4 Médianes . . . 55
4.5 Angles . . . 55
4.5.1 Angles correspondants . . . 56
4.5.2 Angles alternes-internes . . . 57
4.5.3 Somme des angles d'un triangle . . . 57
4.5.4 Somme des angles d'un quadrilatère . . . 59
4.6 Transformations géométriques . . . 61
4.6.1 Symétrie centrale . . . 61
4.6.2 Symétrie axiale . . . 62
4.6.3 Translation . . . 62
4.6.4 Rotation . . . 63
4.6.5 Homothétie . . . 64
4.6.6 Transformer une collection d'objets . . . 65
4.6.7 Transformations en cascade . . . 68
4.7 Frise . . . 69
4.7.1 Translation . . . 70
4.7.2 Symétrie axiale . . . 72
4.7.3 Symétrie centrale . . . 74

4.7.4 Symétrie centrale et symétrie axiale 74

5 Fonctions 76
5.1 Fonction linéaire 76
5.2 Pente d'une droite 76
5.3 Fonction affine 78
5.4 Fonction quadratique 78
5.5 Fonction puissance n-ième 78
5.6 Fonction homographique 79

Partie III Pour aller plus loin dans la programmation

6 Figure Pharo 83
6.1 Exemples de figure Pharo 83
6.2 Méthodes de référence pour les Figures Pharo 84
6.2.1 Commandes générales 85
6.2.2 Point 86
6.2.3 Droite 87
6.2.4 Demi-droite 88
6.2.5 Segment 88
6.2.6 Cercle 89
6.2.7 Arc de cercle 89
6.2.8 Polygone 90
6.2.9 Les transformations géométriques 90
6.2.10 Lieu d'un point 91
6.2.11 Vecteur 91
6.2.12 Valeur numérique 92
6.2.13 Angle 93
6.2.14 Équation 93
6.2.15 Texte 93
6.2.16 Style et attribut 94
6.2.17 Autres méthodes 97
6.3 Galerie d'exemples de figures Pharo 98

7 Outils du développeur 100
7.1 Espace de travail 100
7.2 Profileur 103
7.3 Débogueur 104
7.4 Inspecteur 105
7.5 Chercheur 107
7.6 Spotter 108

Partie IV Annexes

Annexe A GNU Free Documentation License 113

Annexe B Copyright des documents 117

Annexe C Liste des exercices **118**

Annexe D Solutions des exercices **121**

D.1 La syntaxe par l'exemple 121
D.2 Nombres et opérations 125
D.3 Espace 129
D.4 Fonctions 153

Annexe E Liste des exemples **155**

Annexe F Liste des figures **157**

Annexe G Aides **159**

Annexe H Index conceptuel **163**

Annexe I Index des méthodes Dr.Geo **165**

1 Introduction

1.1 À propos de Dr.Geo

Dr.Geo[1] est l'environnement de géométrie interactive et de programmation du projet libre GNU[2]. Outre son interface graphique classique de géométrie interactive, il permet de concevoir des figures interactives par l'écriture d'un code informatique. Les figures géométriques sont ainsi programmées et non plus conçues à la souris. Cette approche s'appuie sur les spécificités du code telles que les structures de contrôle, les boucles, les collections, etc. pour produire un résultat visuel et interactif.

Programmer de la géométrie interactive est aussi une autre façon de faire des mathématiques : les concepts mathématiques sous-jacents sont réinvestis dans le contexte de la programmation, la figure interactive résultante donne un feedback immédiat quant à la validité du code écrit.

Le domaine des mathématiques couvert par cette approche ne se limite pas à la seule géométrie, il est ouvert aux domaines où les représentations visuelles, graphiques et dynamiques sont pertinentes, et ils sont nombreux.

1.2 À propos de ce livre

Le guide utilisateur de Dr.Geo[3] présente déjà en détail, de façon assez technique, toutes les possibilités offertes en termes de programmation. Mais ce guide est essentiellement techinque et n'a que faiblement une visée pédagogique. Ici nous proposons une approche progressive s'appuyant sur de nombreux exemples et exercices directement liés aux contenus mathématiques enseignés dans le secondaire. Les exercices sont corrigés en annexe.

Ce guide est donc destiné aux enseignants de mathématiques ou d'informatique, aux animateurs d'activités extra-scolaires, aux parents et aux jeunes souhaitant découvrir la programmation d'une façon originale et attrayante.

La programmation avec Dr.Geo se pratique avec le langage informatique Pharo[4], c'est le même langage avec lequel Dr.Geo est développé. Pharo est une implémentation et une extension moderne de Smalltalk qui fut mis au point au PARC[5], au cours d'un lent processus de raffinement d'une dizaine d'années, de 1972 à 1983. Ce langage fut conçu pour inventer l'informatique personnelle telle qu'elle est connue aujourd'hui. Avec en plus une vision romanesque propre à cette période de notre histoire, à savoir une dimension programmation par l'utilisateur accessible à tous, y compris aux plus jeunes.

1.3 Contenu

La partie I présente la programmation à travers une série d'exemples et d'exercices progressifs, leur correction se trouve en annexe. L'écriture du code informatique est en français, sans caractère accentué, le langage de programmation est Pharo. L'environnement de programmation est Dr.Geo basé sur celui de Pharo.

La partie II propose une série d'activités adaptés aux enseignements mathématiques en secondaire 1. Selon le système éducatif, le secondaire 1 correspond à des tranches d'âges de 11 à 14 ans. Les thèmes mathématiques abordés couvrent des éléments de la géométrie euclidienne, le repère cartésien, les fonctions affines, quadratiques, puissances n-ième, les équations et systèmes d'équations, transformations géométriques, etc.

---

[1] `http://drgeo.eu`

[2] `http://gnu.org`

[3] `https://www.gnu.org/software/dr-geo/doc/fr`

[4] `http://pharo.org`

[5] `https://fr.wikipedia.org/wiki/Palo_Alto_Research_Center`

Les activités proposées puisent dans ce vivier mathématiques, sans aucune réelle volonté d'exhaustivité. Le fil conducteur est plus l'intérêt à transposer une connaissance, un concept ou une résolution d'un problème mathématique sous le forme d'un petit programme informatique.

Parfois l'objectif recherché sera la simple illustration sous la forme d'une figure géométrique interactive, produit du programme informatique.

La partie III présente plus précisément plusieurs aspects techniques de Dr.Geo : un description détaillée des méthodes pour programmer et les outils pour développer du code informatique

En annexe, partie IV du document, le lecteur trouvera plusieurs indexes dont un sur les méthodes, des aides pour résoudre les exercices, les solutions des exercices, etc.

Partie I
Pour bien démarrer

2 La syntaxe par l'exemple

2.1 La syntaxe, c'est quoi ?

Tout langage de programmation s'appuie sur un ensemble de mots clés et de règles. Pharo est très simple, il en impose une vingtaine, et en ce qui nous concerne moins d'une quinzaine seront utilisés dans ce guide.

Pour donner une image parlante, la syntaxe est au code source, ce que la grammaire et la conjugaison sont au français. Mais pour un code source la conséquence à ne pas respecter la syntaxe est plus embêtante : le programme refusera de fonctionner. Un texte en français avec une grammaire ou une conjugaison approximative pourra toujours être compris par le lecteur humain, c'est la différence entre nous et les machines. Mais pas d'inquiétude, l'environnement Pharo de Dr.Geo offre plusieurs solutions lorsque le code comporte des erreurs de syntaxe.

Dans les sections suivantes nous présentons des exemples illustrant chaque élément de syntaxe. Vous êtes invités à recopier leur code source. C'est formateur et ils sont toujours courts. En prolongement des exemples, des exercices accessibles sont proposés, faites-les !

2.2 Où saisir le code ?

Le code source est saisi au clavier dans un "espace de travail" – nommé "Playground" en anglais qui veut dire espace de jeu, programmer est donc une activité ludique !

Cet outil ressemble à s'y méprendre à un éditeur de texte. Mais c'est en fait une console d'édition de code Pharo : pour écrire, compiler et exécuter du code source. Il est bien sûr possible d'y coller un code source copié ailleurs, par exemple depuis une version web de ce guide. Mais saisir le code source au clavier offre l'agréable sentiment de composer son programme et de réfléchir au rôle de chacune des instructions saisies. Les exemples sont de toute façon très courts.

Revenons à notre "espace de travail", pour l'afficher faire ...Clic arrière-plan → Outils → Espace de travail... ou cliquer sur l'arrière-plan et faire `Alt-k`[1].

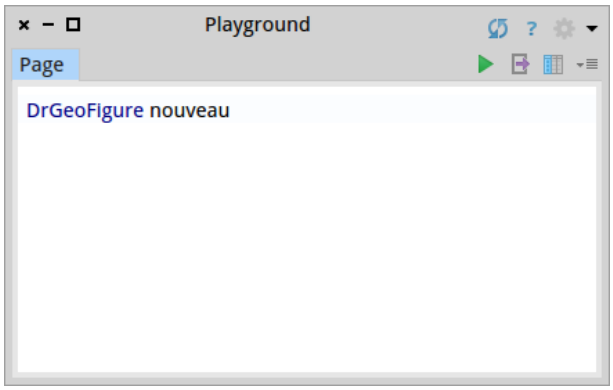


Figure 2.1: Notre "Espace de travail" ou "Playground" avec un code source d'une ligne !

Consigne Importante. Pour s'y retrouver dans les exemples et les exercices recopiés ou écrits, il faut **d'abord** renommer à chaque fois la page de l'espace de travail. Par exemple, lors de la recopie d'un exemple nommé *Exemple 2.1* : double cliquer sur l'onglet *Page* en haut à gauche puis écrire "Exemple 2.1" et valider avec le touche clavier `Enter`.

[1] Selon votre système d'exploitation, remplacer `Alt` par `Ctrl`.

2.3 Hermès, messager des dieux

Avec le langage Pharo tout est histoire de *demander* quelque chose à *quelqu'un* pour obtenir ce que nous souhaitons : afficher une nouvelle figure, faire un calcul, afficher un point dans une figure, afficher une phrase. Pour demander il suffit d'envoyer un **message** à ce quelqu'un et d'attendre sa réponse.

Maintenant, écrivons notre premier programme, comme dans l'exemple ci-dessous. Ensuite pour lancer le programme – le terme "exécuter" est souvent utilisé aussi – il suffit de sélectionner le code à la souris puis de faire ...Clic bouton droit → *Do it(d)* dans le menu contextuel. Ces deux opérations se font également avec le raccourci clavier `Ctrl-a` suivi de `Ctrl-d`. Vous obtenez alors immédiatement le résultat de ce programme, à savoir une figure vide !

```
DrGeoFigure nouveau
```

Exemple 2.1: Premier programme

Comment comprendre ce programme ?

C'est en fait une histoire d'objet à qui nous demandons quelque chose. Ici à l'objet `DrGeoFigure` nous demandons poliment une nouvelle figure en lui envoyant le message `nouveau`. Cela signifie que cet objet est capable de comprendre ce message ! Il nous répond une nouvelle figure qu'il a également le bon goût d'afficher. `DrGeoFigure` est en quelque sorte le patron des figures, le big boss à qui il faut demander des choses, le terme informatique c'est `Classe` – il doit avoir une certaine classe, voire une classe certaine mais rien à voir avec une classe d'école ! Et comme il est important son nom commence toujours par une majuscule.

Ainsi, `DrGeoFigure` est le receveur du message `nouveau` – il est toujours à gauche du message – il nous répond *une figure* qui s'appelle une instance de `DrGeoFigure` dans le jargon informatique.

2.3.1 Message unaire

Ce qui est bien avec les messages c'est que nous pouvons les envoyer les uns après les autres, si l'objet est capable de les comprendre.

Dans cet autre exemple, nous demandons à la figure nouvellement créée d'afficher le repère cartésien – les axes des abscisses et des ordonnées :

```
DrGeoFigure nouveau afficherAxes
```

Exemple 2.2: Figure avec les axes des abscisses et des ordonnées

Traduisons ce qui se passe : le message `nouveau` est envoyé à la classe `DrGeoFigure`, elle répond alors une nouvelle figure à qui nous demandons ensuite d'afficher les axes avec le message `afficherAxes`. Ainsi la lecture du code se fait de la gauche vers la droite, comme une phrase en français, mais en plus simple car il n'y pas vraiment de conjugaison ou grammaire bizarre.

Et donc nous envoyons des messages comme nous enfilons des perles dans un collier. La figure affichée est trop petite ! On envoie le message `pleinEcran`. Pas de grille ! On envoie le message `afficherGrille`. Il suffit donc d'envoyer les messages appropriés :

```
DrGeoFigure nouveau afficherAxes afficherGrille pleinEcran
```

Exemple 2.3: Des messages comme des perles sur un collier

La phrase commence à être longue ! Ces messages que nous envoyons en enfilade sont des messages *unaire* que la figure est capable de comprendre bien sûr. Chacun de ces messages demande juste *une* chose spécifique, sans autre précision supplémentaire, puis répond la même figure, ce qui permet l'enfilade de messages.

2.3.2 Message à mot clé

Supposons que la graduation des axes de notre figure ne nous convient pas, nous souhaitons grossir la figure. Il nous suffit d'envoyer le message approprié pour demander un changement d'échelle, mais comment préciser de combien ? Et bien il suffit de l'écrire avec le message :

```
DrGeoFigure nouveau afficherAxes pleinEcran echelle: 100
```

Exemple 2.4: Changement d'échelle

La phrase est tellement longue que le message `afficherGrille` a été enlevé, mais vous pouvez le laisser.

Ici le message est `echelle:` avec en plus la valeur `100`, cela s'appelle un *paramètre*. Remarquer que le message se termine par "**:**", cela veut dire qu'un paramètre est attendu tout de suite après. Un tel message est dit à *mot clé*, et il est possible d'avoir plusieurs paramètres, donc plusieurs mots clés, nous le verrons plus tard.

Pour résumer, nous avons demandé avec le message à mot clé `echelle: 100` un changement d'échelle à 100, cela veut dire "100 points écran = 1 unité".

Modifier l'Exemple 2.4 pour une échelle égale à 1. Qu'observe-t-on ?

Exercice 2.1: **Echelle 1**

Que se passe-t-il avec une échelle égale à 0 ?

Exercice 2.2: **Echelle 0**

Voici un autre exemple de message à mot clé pour afficher un texte dans une figure.

```
DrGeoFigure nouveau pleinEcran
   texte: 'Bonjour. Je suis Eric Dupont'
```

Exemple 2.5: Bonjour tout le monde

Cette-fois ci le message à mot clé est `texte:` et son paramètre est la phrase `'Bonjour. Je suis Eric Dupont'`. Une phrase est toujours entourée d'apostrophes `'` – dans le jargon informatique cela s'appelle une chaîne de caractères.

Modifier l'Exemple 2.5 pour afficher comme message 'Vive la classe XXX !!'

Exercice 2.3: **Ma classe**

2.3.3 Message binaire

Jusqu'à présent nous avons uniquement affiché une figure vide, il serait temps d'y ajouter un objet géométrique, par exemple un point. Si vous avez bien suivi jusqu'à présent, vous devinez qu'il suffira d'envoyer un message `point` à la figure pour demander de créer un point.

Mais où placer le point dans la figure ? Et bien nous devons aussi indiquer les coordonnées avec le message, et donc utiliser un message à mot clé `point:` où "les coordonnées" représente son paramètre. Cela nous donne donc le code ci-dessous.

```
DrGeoFigure nouveau afficherAxes point: 0 @ 1
```

Exemple 2.6: Figure avec un point

La figure ci-dessous est alors obtenue avec un point rouge de coordonnées (0;1). Comme c'est une figure interactive vous pouvez même attraper le point à la souris et le déplacer ci et là.

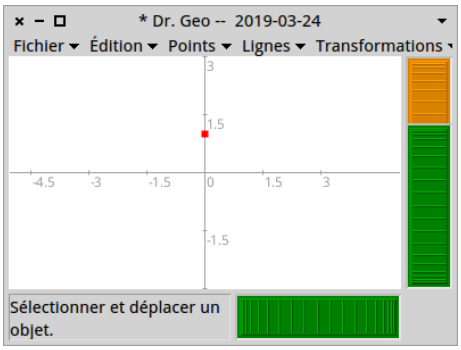


Figure 2.2: Figure Dr.Geo avec le point (0;1)

Vous avez sans doute deviné que le paramètre `0 @ 1` du message `point:` représente les coordonnées du point. Et bien ce code `0 @ 1` est encore une histoire de message. Cette fois-ci, cela s'appelle un message *binaire*, c'est le troisième et dernier type de message.

Un message *binaire* – comme son nom l'indique – se compose de deux objets et d'un message. Le message ici c'est `@`, il est envoyé au nombre `0` – zéro[2] avec comme paramètre le nombre `1`. L'objet `0` répond alors à ce message par un objet de type coordonnées de point[3]. Pour le dire d'une autre façon, nous demandons au nombre 0 de s'associer avec le nombre 1 pour former un couple de coordonnées.

[2] Souvenez-vous, le receveur du message est toujours à gauche du message.

[3] C'est une instance de la classe `Point`, une sorte de couple de coordonnées.

Outre représenter des coordonnées de points, les messages binaires sont aussi utiles pour écrire les quatre opérations arithmétiques :

1. `1 + 2`. Nous demandons à 1 de s'ajouter avec 2 et de nous répondre le résultat.
2. `8 * 3`. Nous demandons à 8 de se multiplier avec 3 et de nous répondre le résultat.

Modifiez Exemple 2.6 pour placer les points de coordonnées (0;0), (1;0), (0;-1) ou (-1;-1).

Exercice 2.4: **Variez les coordonnées**

Voici un autre exemple avec un vecteur. Attrapez-le à la souris.

```
DrGeoFigure nouveau afficherAxes vecteur: 3 @ 1.5
```

Exemple 2.7: Un vecteur, c'est un déplacement

Résumons !

Pharo propose donc 3 types différents de message pour écrire notre code informatique.

- Le message *unaire*, par exemple `afficherAxes` ou `pleinEcran`. La priorité d'envoi des messages est de la gauche vers la droite.
- Le message à *mot clé*, par exemple `echelle:` ou `point:` Il se termine par un ":" et attend à sa droite un paramètre.
- Le message *binaire*, par exemple `@`. Il a un seul paramètre, l'objet à sa droite.

2.3.4 Priorités des messages

Mais à l'identique des priorités des opérations arithmétiques (+, -, *, /), quel est l'ordre d'envoi de ces trois messages ? Et bien c'est :

Parenthèses >> Unaire >> Binaire >> Mot clé >> De gauche à droite

Dans Exemple 2.6, l'ordre d'envoi est donc :

1. Message unaire `nouveau` envoyé à `DrGeoFigure`, une nouvelle figure est retournée, notre figure.
2. Message unaire `afficherAxes` envoyé à notre figure.
3. Message binaire `@` pour obtenir un objet de coordonnées (0;1).
4. Message à mot clé `point:` envoyé à notre figure avec le paramètre coordonnées de point obtenu en retour du message précédent.

Dans ce code, quel est l'ordre d'envoi des messages ?
`DrGeoFigure nouveau point: 2 @ (2 + 2)`

Exercice 2.5: **Ordre d'envoi**

Que se passe-t-il lorsque les parenthèses de Exercice 2.5 sont supprimées ?
`DrGeoFigure nouveau point: 2 @ 2 + 2`

Exercice 2.6: **Ordre d'envoi**

Maintenant nous souhaitons afficher une droite passant par les *deux* points (0;0) et (0;1). Nous utilisons donc le message à mot clé `droitePassantPar:et:` avec *deux* paramètres. Le code s'écrit comme suit.

```
DrGeoFigure nouveau droitePassantPar: 0 @ 0 et: 1 @ 1
```

Exemple 2.8: Droite passant par (0;0) et (1;1)

Un message à mot clé peut avoir autant de paramètres que nécessaire : 1, 2, 3, etc. Bien sûr le receveur doit comprendre un tel message. Mais ici, notre figure est capable de comprendre le message `droitePassantPar:et:` et bien d'autres encore.

Un peu de pratique !

A toi de jouer maintenant avec les exercices suivants.

Construire un segment passant par les points (2;1) et (0;0).
Utiliser le message `segmentDe:a:`

Exercice 2.7: **Mon premier segment**

Construire une demi-droite d'origine (-2;-1) et passant par (0;0).
Utiliser le message `demiDroiteOrigine:passantPar:`

Exercice 2.8: **Ma première demi-droite**

Construire un cercle de centre (0;0) et de rayon 3.
Utiliser le message `cercleCentre:rayon:`

Exercice 2.9: **Mon premier cercle**

2.4 Cascade de messages

Tous ces messages envoyés les uns après les autres, ce n'est pas toujours très clair. Lorsque nous envoyons nos messages à un même objet – ici la figure créée par le code `DrGeoFigure nouveau` – il est préférable d'utiliser *une cascade de messages* en les séparant par un "**;**". Exemple 2.3 se réécrit alors comme suit.

```
DrGeoFigure nouveau afficherAxes; afficherGrille; pleinEcran
```

Exemple 2.9: Des messages en cascade

Pour rendre le code plus lisible, nous l'écrivons sur plusieurs lignes en insérant quelques espaces en début de lignes.

```
DrGeoFigure nouveau
   afficherAxes;
   afficherGrille;
   pleinEcran
```

Exemple 2.10: Des messages en cascade

Lorsque les messages sont envoyés de cette façon à un même objet, cela s'appelle une cascade de messages. Comme une cascade avec de l'eau qui ruisselle. Bon c'est vrai il faut beaucoup d'imagination, mais les informaticiens en ont beaucoup.
Que se passe-t-il dans cette cascade de messages ?

1. `DrGeoFigure nouveau` ⇒ objet *une figure* créé
2. `afficherAxes` est envoyé à *la figure*
3. `afficherGrille` est envoyé à *la figure*
4. `pleinEcran` est envoyé à *la figure*

Pourquoi n'y a-t-il pas de ";" avant le message `afficherAxes` *?*

Exercice 2.10: **Premier de cascade**

Avec les cascades, nous pouvons écrire plusieurs messages à mot-clé. Cela n'est pas possible sans.

```
DrGeoFigure nouveau pleinEcran;
   afficherAxes;
   afficherGrille;
   segmentDe: 0 @ 0 a: 1 @ 1;
   segmentDe: 0 @ 0 a: -1 @ 1
```

Exemple 2.11: Cascade avec toutes sortes de messages

Construire une figure avec ses axes, sa grille, en plein écran et avec un carré de coté 4 unités.

Exercice 2.11: **Un carré**

Dans ce carré construire un cercle inscrit à l'intérieur.

Exercice 2.12: **Un carré et un cercle**

Nous savons maintenant construire une figure, lui demander d'afficher ses axes ou sa grille. Dans cette figure nous savons y construire des lignes, des carrés ou des cercles. Comme nous utilisons souvent notre figure pour y construire des objets géométriques, nous pourrions lui donner un petit nom. Dans le jargon informatique cela s'appelle une *variable*.

2.5 Les variables

En programmation une variable permet de se souvenir d'un objet pour le réutiliser aussi souvent que souhaité. Le nom d'une variable est libre. Avant son utilisation, une variable nommée `maFigure` doit être déclarée en début de code de cette façon : `| maFigure |`.

```
| maFigure |
maFigure := DrGeoFigure nouveau.
maFigure point: 0 @ 0
```

Exemple 2.12: Une variable pour notre figure

1. A la première ligne, la variable `maFigure` est déclarée.
2. A la deuxième ligne, le résultat du message `DrGeoFigure nouveau` est placé dans la variable `maFigure` grâce au symbole d'affectation "**:=**". Observer le point à la fin de la ligne, c'est un séparateur avec la ligne de code suivante.
3. A la troisième ligne, nous envoyons un message à la variable `maFigure` pour créer un point dans la figure.

En résumé. Une variable est constituée d'un nom et d'un objet. Donner un nom à un objet – pour en faire une variable – s'appelle une affectation, elle se fait avec le symbole "**:=**". Pour déclarer une variable, il faut la placer entre `| |` au début du programme.

L'intérêt d'une variable est son utilisation multiple dans le code.

```
| maFigure |
maFigure := DrGeoFigure nouveau.
maFigure afficherGrille.
maFigure segmentDe: 0 @ 0 a: 4 @ 4.
maFigure segmentDe: 4 @ 0 a: 0 @ 4
```

Exemple 2.13: Une variable utilisée plusieurs fois

Remarques. Le point en fin de chaque ligne est un séparateur d'instructions. C'est pour cette raison que dans l'Exemple 2.13 la dernière ligne ne se termine pas par un point. Aussi la déclaration de variable `| maFigure |` ne se termine pas par un point car ce n'est pas une instruction.

Écrire le programme sur plusieurs lignes est donc optionnel et l'Exemple 2.13 se réécrit en une seule ligne comme suit, mais ce n'est pas clair du tout et à ne pas faire !

```
| maFigure | maFigure := DrGeoFigure nouveau. maFigure afficherGrille. maFigure segmentDe: [../..]
```

Modifier Exemple 2.13 pour construire un triangle de sommets les points de coordonnées (0;0) (4;0) et (1;3)

Exercice 2.13: **Triangle et variable**

Outre la figure, une variable peut référencer tout objet géométrique créé dans la figure. Dans l'exemple suivant, nous nous souvenons du segment créé dans la variable `segment`. Celle-ci est ensuite utilisée pour créer le milieu du segment.

```
| maFigure segment |
maFigure := DrGeoFigure nouveau.
maFigure afficherGrille.
segment := maFigure segmentDe: 0 @ 0 a: 4 @ 4.
maFigure milieuDe: segment
```

Exemple 2.14: Deux variables

Voici un autre exemple avec trois variables pour construire l'intersection de deux segments. Sur la figure construite, déplace les segments et observe.

```
| maFigure segment1 segment2 |
maFigure := DrGeoFigure nouveau.
maFigure afficherGrille.
segment1 := maFigure segmentDe: 0 @ 0 a: 4 @ 4.
segment2 := maFigure segmentDe: 2 @ 3 a: 4 @ 0.
maFigure intersectionDe: segment1 et: segment2
```

Exemple 2.15: Trois variables

Compléter Exemple 2.14 avec un deuxième segment d'extrémités les points (1;2) et (5;6). Construire son milieu et relier les deux milieux par un segment.

Exercice 2.14: **Segments liés par leur milieu**

Utiliser, modifier des objets

Une variable sert aussi à se souvenir de l'objet pour le modifier plus tard dans le code. Pour un objet géométrique cela signifie modifier un de ses attributs comme son nom (dans la figure), son aspect, sa couleur, etc.

Voici quelques possibilités pour modifier l'aspect d'un segment :

```
| segment |
segment := DrGeoFigure nouveau segmentDe: 0 @ 0 a: 5 @ 5.
segment nommer: 'S'.
segment couleur: Color blue.
segment epais.
segment marquerAvecDisque
```

Exemple 2.16: Variable et attributs

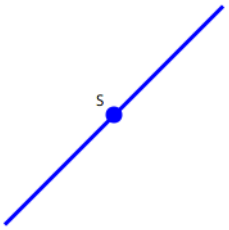


Figure 2.3: Attributs d'un segment

Quelques observations concernant cet exemple. Nous n'avons pas de variable pour nous souvenir de la figure, en effet dans la ligne de code qui crée la figure nous créons en suivant le segment, et par la suite nous ne manipulons plus la figure. En revanche nous avons une variable `segment` afin de nous souvenir du segment et modifier ses attributs. Le message à mot clé `couleur:` attend comme paramètre un objet couleur, un tel objet s'obtient en envoyant à la classe `Color` un message de type `blue`, `yellow`, `pink`, etc.[4]

Pour modifier l'aspect du segment, nous envoyons toujours les messages à la même variable `segment`. Nous pouvons alors réécrire le code avec une cascade de messages. Cela ne change rien au fonctionnement du programme, mais il est plus court et plus clair. Exemple 2.16 devient alors :

```
| segment |
segment := DrGeoFigure nouveau segmentDe: 0 @ 0 a: 5 @ 5.
segment
   nommer: 'S';
   couleur: Color blue;
   epais;
   marquerAvecDisque
```

Voici une liste de messages pour modifier l'aspect d'un segment et d'un point :

Forme de point.
`croix carre rond`

Taille de point.
`petit moyen large`

Épaisseur de ligne.
`fin normal epais`

Style de ligne.
`plein tiret pointille`

[4] Ces messages n'ont pas été traduits en français, d'où leur nom en anglais.

Codage de segment

`marquerAvecCercle` `marquerAvecDisque` `marquerAvecSimpleTrait` `marquerAvecDoubleTrait` `marquerAvecTripleTrait`

Nom d'un objet.

`nommer:`. Le paramètre est une objet de type chaîne de caractères. Par exemple `'Toto'`.

Couleur d'un objet.

`couleur:`. Le paramètre est un objet de type `Color`. Par exemple `Color black` ou `Color orange`.

Reprendre Exemple 2.15 et modifier à sa guise l'aspect des deux segments et du point d'intersection.

Exercice 2.15: **Attributs d'un point**

2.6 Commentaire

Lorsque le code d'une figure s'étale sur plusieurs lignes, il est parfois utile d'ajouter des commentaires pour expliquer ce que fait le code. Un commentaire est une phrase qui n'intervient pas dans le programme, elle est uniquement destinée au lecteur du code source, c'est-à-dire TOI ! Un commentaire s'écrit entre guillemets droits "". Il est placé n'importe où et il peut y en avoir plusieurs.

```
| maFigure segment1 segment2 |
"ceci est une nouvelle figure"
maFigure := DrGeoFigure nouveau.
maFigure afficherGrille.
segment1 := maFigure segmentDe: 0 @ 0 a: 4 @ 4.
segment2 := maFigure segmentDe: 2 @ 3 a: 4 @ 0.
"Construire l'intersection:"
maFigure intersectionDe: segment1 et: segment2
```

Exemple 2.17: Commentaire

2.7 Collection

A l'Exercice 2.13, l'objectif était de créer un triangle. Nous allons découvrir une méthode bien plus efficace pour créer un triangle comme un polygone à trois sommets.

```
DrGeoFigure nouveau polygone: {0 @ 0 . 4 @ 0 . 1 @ 3}
```

Exemple 2.18: Triangle facile !

Ici le message à mot clé `polygone:` a comme paramètre une collection contenant les coordonnées des sommets du triangle. Elle s'écrit `{0 @ 0 . 4 @ 0 . 1 @ 3}`. Une collection contient une série d'objets séparés par des points "." Une telle collection est dite dynamique car son contenu est évalué – calculé – lors de l'exécution du programme. Dans notre exemple, le contenu est une série de coordonnées créées par les quatre messages binaires `@`.

Une collection c'est un peu comme les ensembles en mathématiques : elle commence par une accolade ouvrante { et se termine par une accolade fermante }. L'ensemble des diviseurs de 12 est {1 ; 2 ; 3 ; 4 ; 6 ; 12} en écriture mathématiques et {1 . 2 . 3 . 4 . 6 . 12} en Pharo.

A l'aide d'une collection et du message `polygone:` *code un parallélogramme de sommets (0;0) (4;0) (5;3) (1;3).*

Exercice 2.16: **Parallélogramme facile**

2.8 Bloc de code

Jusqu'à présent, nous produisons des figures qui sont toujours identiques lors de l'exécution de son code source. Nous pouvons toutefois introduire du hasard en demandant de tirer au hasard un nombre entier qui servira d'abscisse ou d'ordonnée. Pour ce faire il suffit d'envoyer le message unaire `auHasard` à un nombre entier. Pour obtenir un nombre entier au hasard entre 1 et 5, écrire `5 auHasard`.

```
DrGeoFigure nouveau
   afficherAxes
   point: 5 auHasard @ 5 auHasard
```

Exemple 2.19: Point au hasard

Là pour comprendre, il faut se souvenir de l'ordre d'envoi des messages. Le paramètre du message `point:` est `5 auHasard @ 5 auHasard`. Les deux messages unaires `auHasard` sont envoyés avant le message `@`, ce qui donne un couple d'abscisse et d'ordonnée tirées au hasard pour les coordonnées du point.

Voici une variante de l'exemple précédent qui donne un résultat un peu différent :

```
DrGeoFigure nouveau
   afficherAxes;
   point: (11 auHasard - 6) @ (11 auHasard - 6)
```

Exemple 2.20: Point au hasard, les négatifs aussi !

Dans l'Exemple 2.20, quelles sont les valeurs possibles pour les abscisses et les ordonnées ?

Exercice 2.17: **Intervalle de valeur**

Le point farceur

Ce qui serait amusant c'est de créer une figure avec un point farceur qui se déplace au hasard lorsque nous essayons de l'attraper. Pour ce faire, les coordonnées du point doivent être tirées au hasard à chaque instant. Il suffit que le code `5 auHasard @ 5 auHasard` calculant les coordonnées soit exécuté à chaque instant, pour cela nous le transformons en un objet **bloc de code**.

Un bloc de code est un objet de type mini-programme qui s'exécute dans notre programme autant de fois que nous le souhaitons. Pour obtenir un bloc de code, il suffit de placer son code source entre crochets [ ]. Ainsi le paramètre du message à mot clé `point:` devient un bloc de code – bout de programme – qui sera exécuté à chaque fois que nous essayons d'attraper le point à la souris :

```
DrGeoFigure nouveau
   afficherAxes;
   point: [5 auHasard @ 5 auHasard]
```

Exemple 2.21: Point farceur

Modifier l'Exemple 2.20 pour en faire un point farceur.

Exercice 2.18: **Point farceur, les négatifs aussi !**

Dans l'Exemple 2.21, le point farceur a des coordonnées qui sont toujours des nombres entiers. Nous aimerions avoir des valeurs décimales, par exemple au dixième près comme 2,4 ou 4,8. Pour y arriver, une astuce classique en informatique est de tirer au hasard un nombre entier plus grand de 1 à 50 – et non plus de 1 à 5 – et de diviser par 10. Le code devient alors `50 auHasard / 10`.

```
DrGeoFigure nouveau
   afficherAxes;
   point: [(50 auHasard / 10) @ (50 auHasard / 10)]
```

Exemple 2.22: Point farceur au dixième

Modifier l'Exemple 2.22 pour que les coordonnées du point farceur soient des nombres décimaux négatifs comme {-4 ; -3,5 ; -3 ; -2,5... ; -1,5 ; -1}.

Exercice 2.19: **Point farceur négatif à 0,5 près**

Voici deux questions un peu difficiles sur le code de l'Exemple 2.22 :

Pourquoi est-ce que des parenthèses ont été ajoutées au receveur et au paramètre du message `@` ?

Exercice 2.20: **Pourquoi des parenthèses ?***

Est-ce que les parenthèses sont nécessaires autour du receveur du message `@` ? Pourquoi ?

Exercice 2.21: **Parenthèses, toutes nécessaires ?***

Point farceur sur diagonale

Une chose encore plus forte est de forcer notre point farceur à rester sur la diagonale des axes des abscisses et des ordonnées. C'est-à-dire sur la droite qui passe par l'origine (0;0) du repère et le point (1;1).

Dans la Figure 2.4, si nous observons bien le point M situé sur cette diagonale, son abscisse et son ordonnée sont les mêmes.

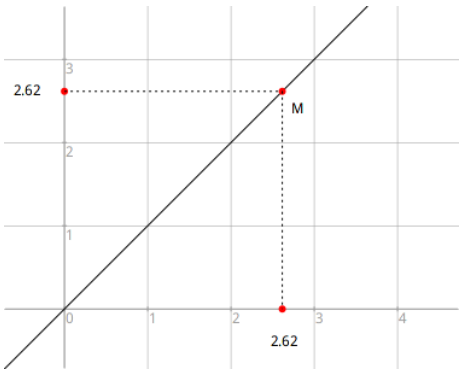


Figure 2.4: Point sur diagonale

> *Ajouter au crayon gris un autre point sur cette diagonale et mesurer le plus précisément ses abscisse et ordonnée. Quelle constatation faire ?*

Exercice 2.22: **Un autre point sur diagonale**

Pour revenir à notre point farceur, pour le forcer à rester sur la diagonale il suffit que ses abscisse et ordonnée soient égales. Pour cela, nous utilisons une variable qui est locale au bloc de code et que nous déclarons au début du bloc avec le code `| coordonnee |`. Cette variable contient alors l'unique calcul d'une coordonnée tirée au hasard.

```
DrGeoFigure nouveau
   afficherAxes;
   afficherGrille;
   point: [
      | coordonnee |
      coordonnee := 50 auHasard / 10.
      coordonnee @ coordonnee
   ];
   droitePassantPar: 0@0 et: 1@1
```

Exemple 2.23: Point farceur sur diagonale

A remarquer comment le bloc de code s'est étoffé sur plusieurs lignes. C'est en fait souvent le cas. La dernière ligne du bloc – `coordonnee @ coordonnee` – est la valeur retournée et qui sert à fixer les coordonnées du point farceur.

Pour plus de clarté, nous introduisons une variable `figure` pour lui envoyer des messages et éviter cette longue cascade de messages. Cela doit être la règle pour toute figure un temps soit peu complexe.

```
| figure |
figure := DrGeoFigure nouveau.
figure
   afficherAxes;
   afficherGrille.
figure point: [
   | coordonnee |
   coordonnee := 50 auHasard / 10.
   coordonnee @ coordonnee].
figure droitePassantPar: 0@0 et: 1@1
```

Exemple 2.24: Point farceur sur diagonale avec variable

Modifier l'Exemple 2.24 afin que l'ordonnée du point farceur soit le double de son abscisse. Que se passe-t-il ? Construire alors la ligne.

Exercice 2.23: **Point farceur, autre droite**

Et que se passe-t-il lorsque son ordonnée est le carré de son absicsse ?

Exercice 2.24: **Point farceur, autre droite ?**

Modifier le code d'un des programmes de point farceur pour nommer le point 'Je suis un farceur'.

Exercice 2.25: **Je suis un point farceur**

2.9 Structures de contrôle

Supposons que nous souhaitions placer des points sur l'axe des abscisses. Un point sur l'axe des abscisses a comme coordonnées (1;0), (2;0), (-3;0), (2,5;0), etc. Sa deuxième coordonnée, l'ordonnée, est toujours 0 car le point est collé sur l'axe des abscisses !

Pour créer des points d'abscisse respective 1, 2, 3 nous utilisons donc le code `figure point: 1 @ 0`, `figure point: 2 @ 0`, `figure point: 3 @ 0`.

Écrire le code d'une figure comprenant 10 points placés sur l'axe des abscisses dont les abscisses sont {1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ; 10}

Exercice 2.26: **Points sur axe des abscisses**

La résolution de l'Exercice 2.26 n'est pas compliqué, mais c'est répétitif d'écrire 10 fois le même code pour créer un point. Et bien cela tombe bien car les programmes sont excellents pour répéter des instructions.

2.9.1 Boucle

Pour répéter un ensemble d'instructions de code, nous utilisons une **boucle**. Pour résoudre l'Exercice 2.26 intelligemment, nous utilisons donc une boucle pour les valeurs d'abscisse de 1 à 10. Cela donne le code suivant :

```
| figure |
figure := DrGeoFigure nouveau afficherAxes.
1 a: 10 faire: [:abscisse |
   figure point: abscisse @ 0]
```

Exemple 2.25: Boucle de 1 à 10

Ici, nous introduisons un nouveau message à mot clé `a:faire:` avec deux paramètres. Ce message permet d'exécuter un code pour un intervalle de valeur, ici de 1 à 10, de 1 en 1.

Le receveur du message est le nombre 1, la valeur initiale des abscisses. Son premier paramètre est le nombre 10, valeur finale des abscisses.

Le deuxième paramètre, le bloc de code, est le code à exécuter pour chacune des valeurs d'abscisse. Ainsi le bloc `[:abscisse | figure point: abscisse @ 0]` est répété 10 fois et le paramètre `abscisse` prend successivement les valeurs entières de 1 à 10.

Dans le bloc de code la première ligne `[:abscisse|` déclare son paramètre. Si le bloc avait nécessité deux paramètres, nous écririons cette première ligne `[:abscisse :ordonnees|`. Le nom des paramètres est libre.

Écrire une boucle pour placer sur l'axe des abscisses les points d'abscisse de -10 à -1

Exercice 2.27: **Sur l'axe des abscisses, les négatifs aussi**

Écrire une boucle pour placer sur l'axe des ordonnées les points d'ordonnée de 1 à 10

Exercice 2.28: **Sur l'axe des ordonnées**

Écrire une boucle pour placer sur la diagonale des axes des abscisses et des ordonnées les points de coordonnée de 1 à 10

Exercice 2.29: **Sur la diagonale**

Nous constatons l'efficacité des boucles pour répéter un grand nombre de fois du code. Nous avons fait une boucle sur les valeurs entières de 1 à 10. Serait-il possible de le faire avec un pas de 0,5, comme 1 ; 1,5 ; 2 ; 2,5 etc. ?

Oui, il suffit d'utiliser le message `a:par:faire:` pour indiquer dans le 3ème paramètre le pas de progression dans la boucle. Notre code devient alors :

```
| figure |
figure := DrGeoFigure nouveau afficherAxes.
1 a: 10 par: 0.5 faire: [:abscisse |
   figure point: abscisse @ 0]
```

Exemple 2.26: Boucle de 1 à 10 à demi-pas

Modifier l'Exemple 2.26 pour afficher les points sur l'axes des abscisses de -5 à 5, tous les 2 dixièmes

Exercice 2.30: **Des pas de lilliputiens**

Encore plus fort, comment faire une boucle sur des abscisses hétéroclites, non régulières cette fois-ci. Par exemple des abscisses comme {-2 ; 4 ; 1/3 ; 3,14 ; -1/5} ?

Et bien nous utilisons un autre message `faire:` que nous envoyons cette fois-ci à une collection. Dans Exemple 2.18 nous avons fait connaissance avec une collection de coordonnées de points, cette fois-ci nous utilisons une collection de valeurs hétéroclites.

```
| figure |
figure := DrGeoFigure nouveau afficherAxes.
{-2 . 4 . 1/3 . 3.14 . -1/5} faire: [:abscisse |
   figure point: abscisse @ 0]
```

Exemple 2.27: Une boucle sur des valeurs en vrac

A l'aide d'une boucle sur une collection de nombres, placer les points de l'axe des ordonnées {-1 ; 5,2 ; -3,14 ; 2,6}

Exercice 2.31: **Nuage de points**

Lorsque nous plaçons les points, il serait judicieux de nommer les points avec leur abscisse.

Modifier l'Exemple 2.27 afin que les points aient comme nom leur abscisse. Indice : envoyer le message `nommer:` *à chaque point créé.*

Exercice 2.32: **Points nommés avec leur abscisse**

Une collection peut contenir n'importe quelle sorte d'objet : des valeurs ou des coordonnées de point comme déjà vu précédemment, et bien d'autres. Il est aussi possible de faire une boucle sur une collection hétéroclite de nombres – entiers, décimaux, fractionnaires.

À l'aide d'une boucle `faire:` *sur une collection, placer les points de coordonnées (1;1), (-1;1), (3;-1) et (2/3;-1/2)*

Exercice 2.33: **Points en vrac**

2.9.2 Test

Une chose intéressante à faire est de placer sur une droite les points dont les abscisses sont des nombres pairs, c'est à dire divisible par deux. Par exemple afficher les points dont les abscisses entre 1 et 100 sont des nombres pairs.

Pour cela nous devons *tester* si les abscisses sont des nombres pairs. **Cela s'appelle tester si une condition est vraie ou fausse, c'est fondamental dans l'écriture d'un programme informatique.** Voici l'exemple complet. Ne pas hésiter à grossir dans la figure pour mieux voir :

```
| figure |
figure := DrGeoFigure nouveau afficherAxes.
figure echelle: 5.
1 a: 100 faire: [:abscisse |
   abscisse pair siVrai: [figure point: abscisse @ 0]
]
```

Exemple 2.28: Abscisse pair

Nous introduisons ici un nouveau message à mot clé `siVrai:`. Il fonctionne de cette façon :

```
(condition) siVrai: [bloc de code à exécuter si condition vraie]
```

Le code de notre (condition) est ici `abscisse pair`. Le message unaire `pair` est envoyé à un nombre et ce dernier nous répond par un objet `"vrai"` ou `"faux"` (`true` ou `false` en anglais) selon que le nombre est pair ou non. Lorsque la condition est vraie alors le bloc en paramètre du message `siVrai:` est exécuté et le point est créé. Sinon il ne se passe rien et la boucle reprend avec la valeur suivante de l'abscisse.

Il existe de nombreux messages pour tester des conditions. En voici quelques uns à envoyer à un nombre : `impair`, `estPremier`, `estEntier`, `estDecimal`, `positif`, `strictementPositif`.

Tester chacun des messages ci-dessus en l'envoyant aux nombres 0 ; 1 ; 2 ; -5 ; 3,4. Pour afficher le résultat de chaque test, lancer le code avec la commande `"Print it"` *du menu de l'espace de travail ou le raccourci clavier* `Ctrl-p`.

Exercice 2.34: **Conditions sur nombre**

Afficher des nombres pairs n'est pas très intéressant, mais finalement avec peu de modifications nous pouvons afficher les nombres premiers.

Modifier l'Exemple 2.28 afin d'afficher uniquement les nombres dont l'abscisse est un nombre premier. Nommer les points avec leur abscisse.

Exercice 2.35: **Nombres premiers**

Partie II
Programmer au secondaire I

3 Nombres et opérations

Poser et résoudre des problèmes pour construire et structurer des représentations des nombres réels
Résoudre des problèmes numériques
Résolution de problèmes numériques en lien avec les ensembles de nombres travaillés, l'écriture de ces nombres et les opérations étudiées.
—*Plan d'Études Romand*

Dans ce chapitre nous ne construisons pas de figure interactive. Le code écrit, lorsqu'exécuté, affichera un résultat numérique. Ainsi depuis l'espace de travail le code édité est exécuté avec la commande *Print it* : ...Clic bouton droit → *Print it* (`Cltrl-p` au clavier). Le résultat de l'exécution de la dernière ligne du programme est imprimé.

3.1 Ensembles de nombres

3.1.1 Les entiers

Il existe différents ensembles de nombres dont les nombres naturels et les nombres entiers relatifs.

Ecrire l'exemple ci-dessous, puis le sélectionner à la souris et l'exécuter par un *Print it* (`Ctrl-p`).

```
(0 a: 100) commeEnsemble
```

Exemple 3.1: Les nombres naturels de 0 à 100

Le résultat est un ensemble des nombres naturels de 0 à 100. `Set` en anglais se traduit par Ensemble.

```
(0 a: 100) commeEnsemble
⇒ "a Set(0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99 100)"
```

Le message à mot clé `a:` crée un intervalle de valeurs numériques. Il est envoyé à un nombre, valeur de départ de l'intervalle, ici 0, avec comme argument la fin de l'intervalle, ici 100. Le message unaire `commeEnsemble` est uniquement là comme commodité d'affichage de la réponse.

Créer l'ensemble de nombres entiers relatifs de -80 à 50.

Exercice 3.1: **Nombres entiers relatifs**

3.1.2 Les décimaux

Les nombres décimaux s'écrivent avec un point "." comme séparateur de la partie entière et de la partie décimale : `1.5`, `1235.021` ou `0.5`. Le nombre `0.00004` s'écrit plus simplement `4e-5` ; cela veut dire 4 précédé de 5 zéros ou 4 comme 5ième chiffre après la virgule.

Attention. Les ordinateurs se représentent les nombres décimaux d'une façon légèrement imprécise. Vous devez savoir que contrairement à vous, ils ne savent pas toujours faire des calculs exacts avec les nombres décimaux. La plupart des systèmes informatiques masquent ces erreurs car elles sont très faibles. Le langage Pharo ne cachent pas cette imprécision.

```
0.1 + 0.2 - 0.3
⇒  5.551115123125783e-17
```

Exemple 3.2: Ordinateur dyscalculique !

Dans l'Exemple 3.2, la valeur retournée devrait être zéro mais ce n'est pas le cas. L'ordinateur retourne `5.55e-17`, soit 0.0000000000000000555, c'est très proche de zéro, mais il y a tout de même une petite erreur.

Donner 3 calculs montrant des erreurs sur les résultats attendus.

Exercice 3.2: **D'autres erreurs de calculs avec des décimaux**

Lorsque la précision doit être absolue pour un programme, il faut utiliser une autre représentation des nombres décimaux : les fractions ou nombres rationnels.

3.1.3 Les rationnels

Les nombres fractionnaires s'écrivent avec une barre de division : en exécutant et en affichant – par *print it* – le code `5/2` ⇒ `(5/2)`. Le système a retourné une fraction entre parenthèses, il n'a pas calculé la forme décimale.

Que se passe-t-il lors de l'exécution du code `5/0` *?*

Exercice 3.3: **Vers l'infini et au delà**

Revenons au problème de dyscalculie de l'ordinateur avec les nombres décimaux. En utilisant l'écriture fractionnaire des nombres décimaux, l'Exemple 3.2 devient alors comme ci-dessous.

```
(1/10) + (2/10) - (3/10)
⇒  0
```

Exemple 3.3: Enfin juste avec les fractions !

Cette fois-ci l'ordinateur ne souffre plus de dyscalculie.

Reprendre l'Exercice 3.2 en écrivant les décimaux avec leur écriture fractionnaire. Les erreurs constatées disparaissent.

Exercice 3.4: **Corriger les erreurs**

L'écriture fractionnaire est donc très utile lorsque la précision est nécessaire. Pour obtenir l'écriture décimale d'une fraction, il suffit de lui demander en lui envoyant le message unaire `commeDecimal`.

Par exemple `(5/2) commeDecimal` ⇒ `2.5`. Attention les parenthèses sont importantes pour l'ordre d'envoi des messages.

Une autre possibilité est d'écrire la fraction avec un numérateur ou un dénominateur sous forme décimale. `5.0/2` ou `5/2.0` ⇒ `2.5`.

Demander à Dr.Geo, de deux façons différentes, l'écriture décimale des fractions suivantes :
- `15/7`
- `535/17`

Exercice 3.5: **Fraction et écriture décimale**

Dr.Geo sait aussi donner l'écriture fractionnaire d'un nombre décimal. Encore une fois, il suffit de demander par l'envoi d'un message `commeFractionApprochee`. Nous savons déjà que l'ordinateur est dyscalculique lorsqu'il manipule des nombres décimaux. C'est pour cette raison que lorsque nous demandons une fraction d'un nombre décimal le message s'appelle `commeFractionApprochee` et non pas `commeFraction` car nous devons tenir compte de son trouble.

Observer les deux exemples ci-dessous.

```
1.3 commeFraction
⇒  (5854679515581645/4503599627370496)

1.3 commeFractionApprochee
⇒  (13/10)
```

Exemple 3.4: Encore la dyscalculie

Le premier montre le nombre rationnel tel que l'ordinateur se représente 1.3. Si c'est juste à l'affichage– la division donne bien 1.3 – c'est loin d'être la réponse attendue. Le deuxième exemple donne la réponse attendue. Il faut donc utiliser le message `commeFractionApprochee` pour obtenir une fraction "raisonnable".

Déterminer avec Dr.Geo les écritures fractionnaires des nombres suivants : 1.2 ; 17.3 ; 0.00175 et 9542.25

Exercice 3.6: **Conversion en fraction**

3.2 Opérations

3.2.1 Priorité des opérations

Le langage informatique Pharo ne connaît pas la priorité des opérateurs. L'ordre de calcul des quatre opérations arithmétiques $(+; -; *; /)$ est toujours de la gauche vers la droite. Les opérateurs sont en fait des messages binaires comme expliqué dans le chapitre sur la syntaxe du langage : l'ordre d'envoi d'une série de messages binaires est de la gauche vers la droite.

Pour changer l'ordre de calculs, il faut donc utiliser des parenthèses.

```
5 * 3 + 2
⇒  17

5 + 3 * 2  C'est équivalent en mathématiques à  (5 + 3) * 2
⇒  16 !

5 + (3 * 2)   C'est équivalent en mathématiques à  5 + 3 * 2
⇒  11
```

Exemple 3.5: Ordre de calculs

Pour résumer. Lorsque nous écrivons un calcul avec le langage Pharo, il faut donc garder en tête que l'ordre de calculs des opérations arithmétiques est toujours de la gauche vers la droite. Pour changer l'ordre, utiliser des parenthèses.

Écrire le code informatique de calculs pour obtenir les résultats suivants (à droite de ⇒). Lorsque nécessaire, utiliser des parenthèses pour suivre l'ordre des priorités mathématiques des opérateurs arithmétiques :
$10/5 + 2 \Rightarrow 4$
$2 + 10/5 \Rightarrow 4$
$10 + 7 * 2 + 4 \Rightarrow 28$
$(6 + 4) * 2 \Rightarrow 20$
$4 * 5 + 7 * 2 \Rightarrow 34$

Exercice 3.7: **Calculs arithmétiques**

3.2.2 Calcul fractionnaire

Dr.Geo sait calculer avec des nombres fractionnaires, il faut juste bien veiller à **placer les fractions entre parenthèses**.

```
Mathématiques :  1 / 5 + 3 / 5 ⇒   4 / 5
Code : (1/5) + (3/5) ⇒  (4/5)
```

Exemple 3.6: Calculs fractionnaires

S'entraîner avec l'exercice suivant :

Écire le code pour effectuer les calculs suivants :
$2/9 + 3/9 \Rightarrow 5/9$
$5/7 - 2/7 \Rightarrow 3/7$
$(2/3) * (4/5) \Rightarrow 8/15$

Exercice 3.8: **Calculs fractionnaires**

Résultat mathématique. L'inverse d'une nombre – et d'une fraction – s'obtient en divisant 1 par ce nombre.

Par exemple, $1/(4/5) = 5/4$. Cela s'écrit de la même façon en code informatique.

```
  Code :
1/(3/5) ⇒  (5/3)
```

Exemple 3.7: Inverse d'une fraction

Écrire le code pour calculer les inverses des fractions (7/4) et (98/99).

Exercice 3.9: **Inverses de fractions**

3.2.3 Division euclidienne

Résultat mathématique. La division euclidienne ou division entière de deux nombres entiers où 423 est le dividende et 15 le diviseur donne un quotient de 28 et un reste de 3.

Ces résultats se calculent avec les messages binaires `//` pour le quotient et `\\` le reste.

```
423 // 15 ⇒  28 "le quotient"
423 \\ 15 ⇒  3  "le reste"
```

Exemple 3.8: Division euclidienne

Écrire le code pour calculer le quotient et le reste des divisions suivantes :
- 65 par 7
- 732 par 13
- 5241 par 29

Exercice 3.10: **Divisions euclidiennes**

3.3 Nombres naturels

Résultat mathématique. Un nombre naturel est un nombre entier et positif, 0 compris. L'ensemble des nombres naturels est infini.

Pour obtenir le 1er million de ces nombres, nous le codons `0 a: 999999`. Pour vérifier son cardinal – nombre d'éléments – nous codons `(0 a: 999999) taille` et nous affichons le résultat par *Print it* (`Ctrl-p`).

3.3.1 Multiples et diviseurs

Résultat mathématique. Lorsque le **reste** d'une division euclidienne est égale à zéro cela signifie que le **dividende** est un multiple du **diviseur**. Cela revient à dire que le **diviseur** est un diviseur du **dividende**.

Avec le division euclidienne nous disposons donc d'une méthode pour tester si un nombre est multiple d'une autre nombre ou si un nombre est diviseur d'un autre nombre. Exécuter l'exemple ci-dessous avec la commande *Print it* – `Ctrl-p` – et observer la réponse affichée.

```
85214 \\ 24 = 0
   siVrai: ['C''est un multiple !']
   siFaux: ['Ce n''est pas un multiple.']
```

Exemple 3.9: Tester un multiple

Modifier l'Exemple 3.9 avec les valeurs 85200 et 24. Quelle réponse est obtenue ?

Exercice 3.11: **Test de multiple**

Adapter l'Exemple 3.9 pour tester si 24 est un diviseur de 85200.

Exercice 3.12: **Test de diviseur**

L'exemple suivant montre comment créer un programme interactif pour tester si un nombre est multiple d'un autre nombre nombre.

```
| a b |
a := (UIManager default request: 'Un premier nombre') commeNombre.
b := (UIManager default request: 'Un deuxième nombre') commeNombre.
a \\ b = 0
   siVrai: [ UIManager default alert:
      a asString,  ' est un mulitple de ',  b asString ]
   siFaux: [ UIManager default alert:
      a asString,  ' n''est pas un mulitple de ',  b asString ]
```

Exemple 3.10: Programme interactif avec multiple

Dans cet exemple, `a` et `b` sont deux variables pour représenter nos deux nombres. Pour demander à l'utilisateur la valeur de ces nombres, nous utilisons l'objet `UIManager default`. C'est le gestionnaire par défaut – manager – de l'Interface Utilisateur – UI veut dire *User Interface* en anglais, il s'occupe des fenêtres à l'écran.

Pour afficher une petite fenêtre où saisir le nombre, nous le demandons au gestionnaire avec le message `request:` – cela veut dire requête/demande. Le gestionnaire nous retourne alors la réponse de l'utilisateur. Cette réponse est une phrase que nous convertissons en nombre avec le message `commeNombre`. Pour afficher la réponse, nous utilisons le message `alert:` envoyé au manager. Cette réponse est construite à partir de portions de phrase concaténées – collées – avec le message binaire virgule :“`,`”.

Modifier l'Exemple 3.10 afin de tester si un nombre est diviseur d'un autre nombre. La réponse est donnée sous la forme d'une phrase affichée dans une fenêtre.

Exercice 3.13: **Programme interactive avec diviseur**

3.3.2 Diviseurs d'un nombre naturel

Pour construire l'ensemble des diviseurs de 100, nous choisissons chacun des nombres naturels de 1 à 100 qui divise 100. Le code ci-dessous lancé par un *Print it* (`Ctrl-p`) affiche une collection des diviseurs de 100.

```
(1 a: 100) choisir: [ :n |  100 \\ n = 0]
```

Exemple 3.11: Diviseurs de 100

Le message à mot clé `choisir:` est envoyé à l'ensemble {1;...;100}. Dans le bloc de code `[]`, l'argument `n` prend chacune des valeurs des nombres de l'ensemble. Nous testons alors si `n` divise 100, c'est vrai lorsque le reste de la division est égale à 0, en code cela donne `100 \\ n = 0`. Dans ce cas, la méthode `choisir:` retient la valeur de `n`.

Adapter l'Exemple 3.11 pour obtenir les diviseurs de 155.

Exercice 3.14: **Diviseurs de 155**

Une dernière chose à faire est de transformer ce code en un bloc de code. De cette façon nous pouvons demander les diviseurs de plusieurs nombres naturels sans répéter tout le code.

```
| diviseurs |
diviseurs := [:nombre |
   (1 a: nombre) choisir: [ :n |  nombre \\ n = 0]].
diviseurs valeur: 100.
diviseurs valeur: 155
```

Exemple 3.12: Diviseurs de 155 et 100

Le bloc de code `[]` est affecté à la variable `diviseurs`. Dans ce bloc, `:nombre` est un paramètre qui prend la valeur de 100 lorsque nous codons `diviseurs valeur: 100` ou qui prend la valeur de 155 lorsque nous codons `diviseurs valeur: 155`.

Attention, dans cet exemple seule la dernière ligne est affichée lors d'une exécution par un *P rint it*. Pour afficher à la fois les diviseurs de 100 et 155, nous concaténons les réponses en remplaçant les deux dernières lignes de code par `(diviseurs valeur: 100), (diviseurs valeur: 155)`.

3.3.3 Diviseurs communs

Résultat mathématique. Un diviseur est commun à deux nombres naturels lorsqu'il divise ces deux nombres.

Examinons en détail l'Exemple 3.12 avec les diviseurs communs de 100 et 155.

Pour 100, le code nous retourne cette liste de diviseurs :

```
| diviseurs |
diviseurs := [:nombre |
   (1 a: nombre) choisir: [ :n |  nombre \\ n = 0]].
diviseurs valeur: 100.
⇒  #(1 2 4 5 10 20 25 50 100)
```

Pour 155, le code nous retourne cette liste de diviseurs :

```
| diviseurs |
diviseurs := [:nombre |
   (1 a: nombre) choisir: [ :n |  nombre \\ n = 0]].
diviseurs valeur: 155.
⇒  #(1 5 31 155)
```

Nous observons que 1 et 5 sont les diviseurs communs de 100 et 155 car ils sont présents dans les collections `#(1 2 4 5 10 20 25 50 100)` **et** `#(1 5 31 155)`. Nous le traduisons en code en demandant l'intersection de ces deux collections, c'est-à-dire ne retenir que leurs nombres communs, cela se code :

```
#(1 2 4 5 10 20 25 50 100) & #(1 5 31 155)
⇒ #(5 1)
```

Modifier l'Exemple 3.12 pour afficher la collection des diviseurs communs de 100 et 155

Exercice 3.15: **Diviseurs communs de 100 et 155**

Maintenant, écrivons un programme interactif pour demander et calculer les diviseurs communs de deux nombres.

Adapter la solution de l'Exercice 3.15 pour écrire un programme interactif comme l'Exemple 3.10 calculant et affichant les diviseurs de deux nombres.

Exercice 3.16: **Programme interactif de diviseurs communs**

Nous pouvons aussi définir un nouveau bloc de code pour calculer les diviseurs communs de deux nombres.

```
| a b diviseurs divCommuns |
diviseurs := [:nombre |
   (1 a: nombre) choisir: [ :n |  nombre \\ n = 0]].
divCommuns := [:x :y | (diviseurs valeur: x) & (diviseurs valeur: y)].
""
divCommuns valeur: 100 valeur: 155
```

Exemple 3.13: Bloc de code diviseurs communs

3.3.4 Plus grand diviseur commun

Résultat mathématique. Lorsque nous nous intéressons aux diviseurs communs de deux nombres naturels, il y en a un de particulier, c'est le **P**lus **G**rand **D**iviseur **C**ommun. Dans l'ensemble des diviseurs communs c'est le plus grand, le maximum.

Toujours avec l'exemple des diviseurs communs de 100 et 155, nous avions trouvé la collection `#(5 1)`, pour obtenir le PGDC, nous demandons le maximum de cette collection en lui envoyant le message `max` , cela donne le code :

```
#(5 1) max
⇒ 5
```

Le PGDC de 100 et 155 est 5 !

Écrire un bloc de code affecté à une variable `pgdc` *et qui retourne le PGDC de deux nombres naturels. Le bloc de code reprendra la solution de l'Exercice 3.15. Comme l'Exemple 3.13, ce bloc aura deux arguments.*

Exercice 3.17: **PGDC**

Écrire un programme interactif qui demande deux nombres naturels à l'utilisateur et qui répond d'une phrase le PGDC affiché dans une fenêtre. S'inspirer de l'Exemple 3.10 et de la solution de Exercice 3.17

Exercice 3.18: **Programme interactif de PGDC**

3.3.5 Nombre premier

Résultat mathématique. Un nombre premier est un nombre naturel qui a exactement deux diviseurs : 1 et lui-même. Lorsque nous connaissons la collection des diviseurs d'un nombre, il est premier si elle contient exactement deux nombres : 1 et le nombre lui-même.

Par exemple 155 n'est pas premier car sa collection de diviseurs `#(1 5 31 155)` contient 4 nombres. Pour le coder il suffit donc de demander la taille de la collection des diviseurs en lui envoyant le message `taille` :

```
#(1 5 31 155) taille
⇒ 4
"155 n'est pas premier"
```

En posant la question *155 est-il premier ?*, nous attendons une réponse qui est soit Vrai, soit Faux. En code informatique cela se traduit en `true` ou `false`. Pour obtenir cette réponse nous comparons la taille de la collection des diviseurs à 2. Si la taille est égale à 2 nous obtenons la réponse `true`, sinon `false`.

```
| diviseurs |
diviseurs := [:nombre |
   (1 a: nombre) choisir: [ :n |  nombre \\ n = 0]].
""
"Tester chacune des lignes ci-dessous"
(diviseurs valeur: 155) taille = 2.
"⇒  false"
(diviseurs valeur: 29) taille = 2.
"⇒  true"
(diviseurs valeur: 100) taille = 2.
"⇒  false"
(diviseurs valeur: 1) taille = 2.
"⇒  false"
(diviseurs valeur: 2) taille = 2.
"⇒  true"
```

Exemple 3.14: Nombre premier ?

Écrire un bloc de code affecté à une variable `premier` *qui retourne une réponse* `true` *ou* `false` *selon que le nombre passé en paramètre est premier ou non.*

Exercice 3.19: **Nombre premier**

Nous disposons maintenant d'un bloc de code pour tester si un nombre est premier ou non. Il serait intéressant de construire la collection des nombres premiers de 1 à 1000. Une ligne de code supplémentaire est nécessaire par rapport à la solution de l'Exercice 3.19.

Compléter la solution de l'Exercice 3.19 avec une ligne de code supplémentaire pour déterminer la collection des nombres premiers entre 1 et 1000. **Indice :** *utiliser les messages à mot clé* `1 a: 1000` *et* `choisir:`.

Exercice 3.20: **Nombres premiers entre 1 et 1000**

Résultat mathématique. Deux nombres sont premiers entre eux lorsque leur seul diviseur commun est 1. Cela signifie que leur PGDC est 1.

Par exemple 21 et 25 sont premiers entre eux car 1 est leur seul diviseur commun. Noter que 21 et 25 pris individuellement ne sont pas des nombres premiers.

Écrire un programme interactif qui demande à l'utilisateur deux nombres naturels et qui répond par une phrase si les nombres sont premiers entre eux ou pas. S'inspirer de la solution de l'Exercice 3.18.

Exercice 3.21: **Nombres premiers entre eux ?**

4 Espace

Poser et résoudre des problèmes pour modéliser le plan et l'espace
Résolution de problèmes géométriques en lien avec les figures et les transformations étudiées
—*Plan d'Études Romand*

4.1 Droites

4.1.1 Parallèles

Résultat mathématique. Deux droites d1 et d2 parallèles à une même troisième droite d3 sont parallèles entre elles. d1 // d3 et d2 // d3 ⇒ d1 // d2.

Cette proposition des éléments d'Euclide se traduit en un code simple :

```
| figure d1 d2 d3 |
figure := DrGeoFigure nouveau.
d1 := figure droitePassantPar: 0 @ 5 et: 2 @ 0.
d1 nommer: 'd1'.
(figure point: 0 @ 5) montrer.
d2 := figure paralleleA: d1 passantPar: 5 @ 0.
d2 nommer: 'd2'.
d3 := figure paralleleA: d1 passantPar: 6 @ 0.
d3 nommer: 'd3'
```

Exemple 4.1: Trois droites parallèles

Dans la figure, déplacer le point rouge et observer.

La 5ème ligne du code source force à rendre visible le point de coordonnées (0;5). Celui-ci est en effet ajouté à la figure en même temps que la droite d1, mais par défaut il est masqué à ce moment là pour ne pas surcharger la figure. Pour interagir avec la figure, il est intéressant de déplacer ce point là, nous le rendons donc visible.

Cette proposition est vraie pour 3 droites, écrire le programme pour construire les droites parallèles à d1 et passant par les points de l'axes des abscisses {3 ; 3.5 ; 4 ; 4.5 ;...; 12}

Exercice 4.1: **Droites parallèles à la folie**

4.1.2 Perpendiculaires

Résultat mathématique. Deux droites d1 et d2 perpendiculaires à une même troisième droite d3 sont parallèles entre elles.

Cette proposition des éléments d'Euclide se traduit également en un code simple :

```
| figure d1 d2 d3 |
figure := DrGeoFigure nouveau.
d1 := figure droitePassantPar: 0 @ 5 et: 2 @ 0.
d1 nommer: 'd1'.
(figure point: 0 @ 5) montrer.
d2 := figure perpendiculaireA: d1 passantPar: 0 @ 0.
d2 nommer: 'd2'.
d3 := figure perpendiculaireA: d1 passantPar: 6 @ 0.
d3 nommer: 'd3'
```

Exemple 4.2: Deux droites perpendiculaires

Dans la figure, déplacer le point rouge et observer.

Ecrire le programme pour construire les droites perpendiculaires à d1 et passant par les points de l'axes des abscisses {3 ; 3.5 ; 4 ; 4.5 ;...; 12}

Exercice 4.2: **Droites perpendiculaires à la folie**

Dans ce monde de grisaille – fond de figure blanc/noir et droite noire/blanche – ce serait revigorant de colorer les droites, par exemple :

```
| figure d1 droite |
figure := DrGeoFigure nouveau.
d1 := figure droitePassantPar: 0 @ 5 et: 2 @ 0.
d1 nommer: 'd1'.
(figure point: 0 @ 5) montrer.
3 a: 12 faire: [:abscisse |
   droite := figure perpendiculaireA: d1 passantPar: abscisse @ 0.
   abscisse pair
      siVrai: [droite couleur: Color red]
      siFaux: [droite couleur: Color blue]
]
```

Exemple 4.3: Droites paires ou impaires colorées

La perpendiculaire créée est affectée à une variable `droite`. La boucle `a:faire:` parcourt les abscisses entières de 3 à 12. Dans cette boucle, la condition `abscisse pair` teste avec le message à mot clé `siVrai:siFaux:` si l'abscisse est paire ou impaire. Selon le cas, la droite est colorée en rouge ou en bleu.

Voici un exercice plus intéressant à faire :

Modifier l'Exemple 4.3 pour construire les perpendiculaires avec les abscisses entières de 1 à 500. Colorer en rouge les abscisses pairs, en orange les impairs et en bleu les nombres premiers. Modifier l'échelle de la figure à 3 pour une meilleure vue d'ensemble.

Exercice 4.3: **Pair, impair, premier**

Dans la figure produite, il est alors possible de reconnaître la séquence des nombres premiers parmi les nombres pairs et impairs.

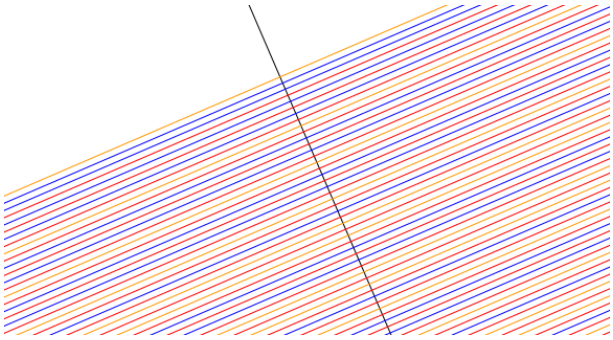


Figure 4.1: Les nombres premiers en bleu, parmi les autres nombres pairs et impairs non premiers, respectivement rouge et orange

4.1.3 Distance

4.1.3.1 Distance entre deux points

Résultat mathématique. La distance entre deux points est simplement la mesure du segment d'extrémités ces deux points.

```
| figure ptA ptB |
figure := DrGeoFigure nouveau afficherAxes.
ptA := figure point: 2 @ 3.
ptA nommer: 'A'.
ptB := (figure point: 3 @ -2) nommer: 'B'.
figure segmentDe: ptA a: ptB.
(figure distanceDe: ptA a: ptB) montrer
```

Exemple 4.4: Distance entre deux points

Dans la figure, en déplaçant à la souris les points A ou B, la distance est automatiquement actualisée. En attrapant la distance, des tirets rouges montrent qu'elle se rapporte aux points A et B.

4.1.3.2 Distance d'un point à une droite

Résultat mathématique. La distance entre un point et une droite est obtenue en construisant la droite perpendiculaire à la première droite et passant par le point. Ensuite mesurer la distance entre le point et l'intersection des deux droites. *C'est le plus court chemin du point à la droite.*

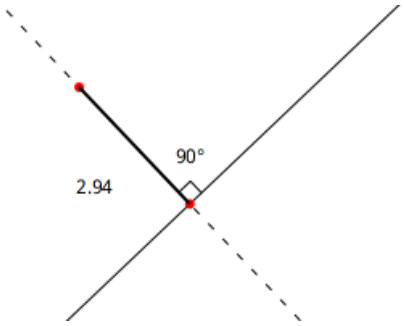


Figure 4.2: Distance d'un point à une droite

Pour réaliser la figure, nous introduisons un nouveau message à mot clé `intersectionDe:et:`. Ses paramètres sont les deux lignes dont nous souhaitons l'intersection.

```
| figure droite pointA perp intersection |
figure := DrGeoFigure nouveau.
droite := figure droitePassantPar: 5 @ 5 et: 7 @ -2.
pointA := figure point: -5 @ -5.
perp := figure perpendiculaireA: droite passantPar: pointA.
intersection := figure intersectionDe: droite et: perp.
(figure distanceDe: pointA a: intersection) montrer
```

Exemple 4.5: Distance d'un point à une droite

4.1.3.3 Distance entre deux droites parallèles

Résultat mathématique. La distance entre deux droites parallèles est obtenue en construisant une perpendiculaire à ces deux droites et en mesurant la distance entre leur intersection. *C'est la longueur d'un plus court chemin entre ces deux droites.*

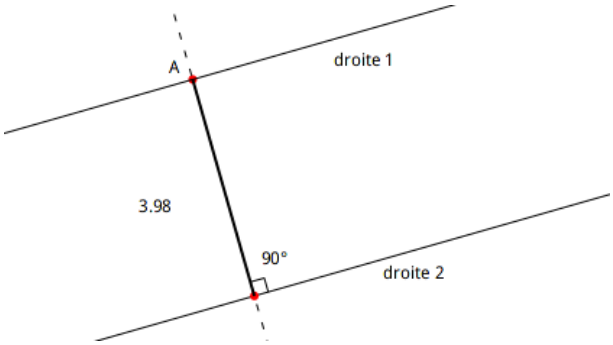


Figure 4.3: Distance entre deux droites parallèles

Dans l'exercice suivant, le programme pour produire une telle figure est partiellement écrit, il est à terminer.

Terminer l'écriture du programme ci-dessous pour construire la distance entre deux droites parallèles.

```
| figure droite1 droite2 perp pointA pointB |
figure := DrGeoFigure nouveau afficherAxes.
droite1 := figure droitePassantPar: 5 @ 5 et: 7 @ -2.
droite2 := figure paralleleA: droite1 passantPar: 0 @ 0.
perp := figure perpendiculaireA: droite2 passantPar: -5 @ 0.
pointA := figure intersectionDe: droite1 et: perp.
[...]
```

Exercice 4.4: **Distance entre deux droites parallèles**

Dans la figure produite à l'Exercice 4.4, que se passe-t-il lorsque la perpendiculaire est déplacée à la souris ?

Exercice 4.5: **Déplacer la perpendiculaire**

La distance entre deux droites parallèles s'obtient donc par la construction d'un chemin perpendiculaire entre celles-ci. Pour comparer avec d'autres chemins allant d'une droite à l'autre – et non perpendiculaires – il serait intéressant de construire un grand nombre de segments entre `pointA` et d'autres points sur `droite2`.

Mais avant cela, voici un exemple avec un autre chemin pour aller de `droite1` à `droite2`. La longueur de ce chemin est plus grande que la distance !

```
| figure droite1 droite2 perp pointA autrePoint|
figure := DrGeoFigure nouveau afficherAxes.
droite1 := figure droitePassantPar: 5 @ 5 et: 7 @ -2.
droite2 := figure paralleleA: droite1 passantPar: 0 @ 0.
perp := figure perpendiculaireA: droite2 passantPar: -5 @ 0.
pointA := figure intersectionDe: droite1 et: perp.
autrePoint := figure pointSurLigne: droite2 a: 0.8.
(figure segmentDe: autrePoint a: pointA) pointille
```

Exemple 4.6: Un autre chemin

Cet exemple introduit un nouveau message à mot clé `pointSurLigne:a:`. Il place un point sur une ligne. Son premier paramètre est une ligne – ici `droite2` – et le deuxième paramètre est l'abscisse du point sur la ligne. Cette abscisse est une valeur décimale entre 0 et 1.

En adaptant l'Exemple 4.6, construire une série de segments d'extrémités `pointA` *et des points sur* `droite2`*. Ces derniers sont construits à l'aide d'une boucle* `a:par:faire:` *de 0 à 1 par un pas de 0.01.*

Exercice 4.6: **D'autres chemins**

Dans la figure déplacer la perpendiculaire et observer l'effet graphique !

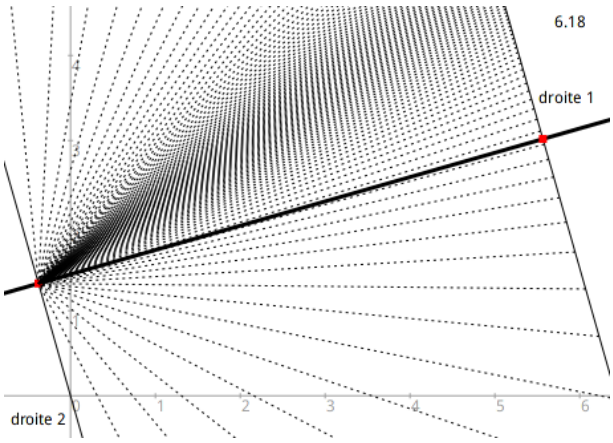


Figure 4.4: Que de chemins !

4.2 Quadrilatères

4.2.1 Parallélogramme

Un parallélogramme possède plusieurs propriétés caractéristiques que nous utilisons pour le construire de différentes façons. Nous explorons celles-ci dans les sections suivantes.

Par les côtés opposés

Résultat mathématique. Un parallélogramme est un quadrilatère dont les côtés opposés sont parallèles.

Étant donnés trois points A, B, C, le quatrième sommet D est construit comme l'intersection de **la parallèle à AB** passant par C et de **la parallèle à BC** passant par A.

```
| figure a b c d ab bc |
figure := DrGeoFigure nouveau.
a := (figure point: 1 @ 1) nommer: 'A'.
b := (figure point: 5 @ 2) nommer: 'B'.
c := (figure point: 6 @ 6) nommer: 'C'.
ab := figure segmentDe: a a: b.
bc := figure segmentDe: b a: c.
d := figure
  intersectionDe: (figure paralleleA: ab passantPar: c) cacher
  et: (figure paralleleA: bc passantPar: a) cacher.
d nommer: 'D'.
figure segmentDe: a a: d.
figure segmentDe: c a: d
```

Exemple 4.7: Parallélogramme et parallèles

Qu'observe-t-on lorsque les points A, B ou C sont déplacés ? Pourquoi ?

Exercice 4.7: **Toujours parallélogramme**

En suivant l'Exemple 4.7, construire le parallélogramme OMNP connaissant ses sommets M(-5;2) N(3;2) et P(1;-5). Il est conseillé de faire un croquis.

Exercice 4.8: **Un autre parallélogramme**

Résultat mathématique. Les cotés opposés d'un parallélogramme sont **isométriques** et parallèles.

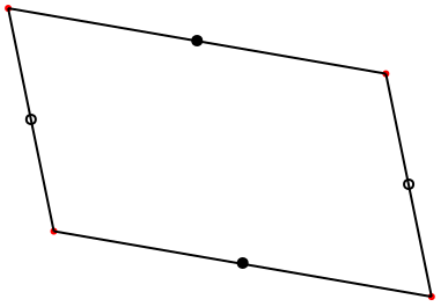


Figure 4.5: Côtés opposés du parallélogramme

C'est une propriété utilisée pour construire un parallélogramme au compas. Ici dans le code d'une figure programmée, nous utilisons des cercles dont les rayons sont les longueurs des côtés.

```
| figure o m n p mn pm cercle1 cercle2 |
figure := DrGeoFigure nouveau.
m := (figure point: -5 @ 2) nommer: 'M'.
n := (figure point: 3 @ 2) nommer: 'N'.
p := (figure point: 1 @ -5) nommer: 'P'.
mn := figure segmentDe: m a: n.
pm := figure segmentDe: p a: m.
cercle1 := figure cercleCentre: p segment: mn.
cercle1 tiret.
cercle2 := figure cercleCentre: n segment: pm.
cercle2 tiret.
o := figure intersectionDe: cercle1 et: cercle2.
o nommer: 'O'.
figure segmentDe: o a: n.
figure segmentDe: o a: p
```

Exemple 4.8: Parallélogramme et côtés isométriques

Une fois le code exécutée, dans la figure attraper et déplacer les points M, N ou P et observer la figure pour bien la comprendre.

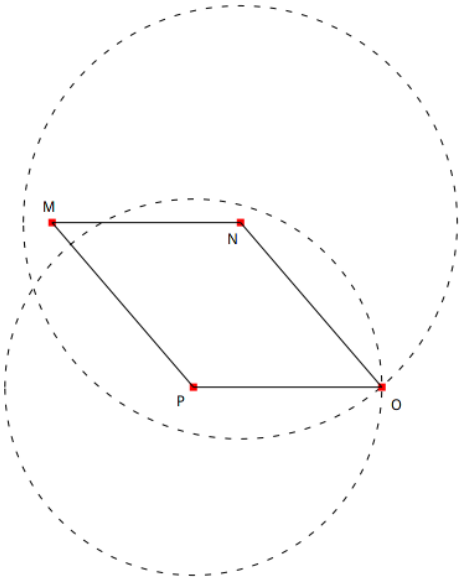


Figure 4.6: Parallélogramme et cercles

Les deux cercles de la Figure 4.6 se coupent en deux points dont l'un est le point O. Est-ce que le deuxième point d'intersection des deux cercles convient pour former un parallélogramme ? Pourquoi ?

Exercice 4.9: **Deux points d'intersection**

Par les diagonales

Résultat mathématique. Les diagonales d'un parallélogramme se coupent en leur milieu. Ce milieu est donc un centre de symétrie.

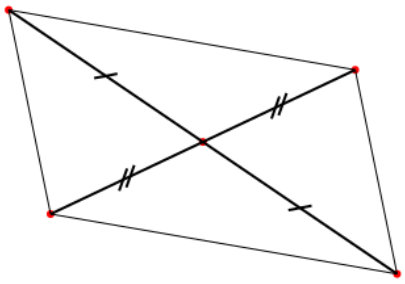


Figure 4.7: Diagonales du parallélogramme

Pour un parallélogramme ABCD dont I est le milieu de ses diagonales alors le point D est le symétrique de B par rapport à I. Cela nous fournit un programme de construction de ABCD que nous développons dans les deux exercices suivants.

Connaissant les points A(-5;2) B(3;2) et C(1;-5), construire le point I milieu de AC. Utiliser le message à mot clé `milieuDe:et:` pour construire le milieu des points A et C.

Exercice 4.10: **Parallélogramme et centre**

L'Exercice 4.10 construit le point I, centre du parallélogramme. Pour construire le parallélogramme ABCD, le point D est construit comme symétrique de B par rapport à I avec le message `symetriqueDe:selonCentre:`.

A partir de l'Exercice 4.10 construire le parallélogramme ABCD. Tracer ABCD avec un polygone.

Exercice 4.11: **Parallélogramme et symétrie**

4.2.2 Losange

Résultat mathématique. Un losange est un parallélogramme dont les côtés sont isométriques.

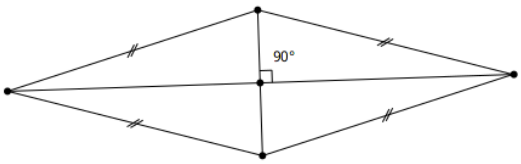


Figure 4.8: Losange

Troisième sommet

Ainsi pour un losange ABCD, connaissant A et B, le point C est tel que BA=BC. Pour placer correctement un point C, nous construisons le cercle de centre B et rayon BA, puis nous plaçons sur ce cercle un point C.

Programmer une figure avec les points A(0;0) et B(5;1), puis construire le cercle de centre B et passant par A. Sur ce cercle placer le point C d'abscisse 0.2. Utiliser les messages à mots clés `cercleCentre:passantPar:` et `pointSurLigne:a:`

Exercice 4.12: **Côtés adjacents isométriques**

Par les côtés opposés

Une fois que les points A, B et C sont placés tel que BA=BC, il suffit de reprendre le programme de construction pour un parallélogramme : *étant donnés trois points A, B, C, le quatrième sommet D est construit comme l'intersection de* **la parallèle à AB** *passant par C et de* **la parallèle à BC** *passant par A.*

Compléter le code de l'Exercice 4.12 pour construire le point D afin que ABCD soit un parallélogramme. S'inspirer également du code de l'Exemple 4.7 pour cette construction.

Exercice 4.13: **Losange comme un parallélogramme**

Pour vérifier que le programme est valide, déplacer A, B et C afin d'observer ce qui se passe.

Par les diagonales

Résultat mathématique. Les diagonales d'un losange – en plus de se couper en leur milieu comme pour tout parallélogramme – sont **perpendiculaires**.

Lorsque les points A, B et C d'un losange ABCD sont placés sur la figure il est alors possible de construire le sommet D comme symétrique de B par rapport à I, milieu de AC.

Compléter le code de l'Exercice 4.12 pour construire le point D et le parallélogramme ABCD avec la méthode vue à l'Exercice 4.11.

Exercice 4.14: **Losange et centre**

4.2.3 Rectangle

Résultat mathématique. Un rectangle est un parallélogramme dont les côtés sont perpendiculaires.

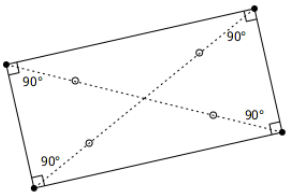


Figure 4.9: Rectangle

Troisième sommet

Ainsi pour un rectangle ABCD, connaissant les points A et B, le point C est tel que la droite BA est perpendiculaire à la droite BC. Pour placer correctement un point C, nous donc construisons la droite passant par B et perpendiculaire à la droite AB, puis nous plaçons sur cette droite un point C.

Programmer une figure avec les points A(0;0) et B(5;1), puis construire la droite passant par B et perpendiculaire à la droite AB. Sur cette droite placer le point C d'abscisse 0.1 Utiliser les messages à mots clés `perpendiculaireA:passantPar:` et `pointSurLigne:a:`

Exercice 4.15: **Côtés adjacents perpendiculaires**

Par les côtés opposés

Une fois les trois points A, B et C placés comme dans l'Exercice 4.15, il suffit de reprendre le programme de construction pour un parallélogramme : *étant donnés trois points A, B, C, le quatrième sommet D est construit comme l'intersection de* **la parallèle à AB** *passant par C et de* **la parallèle à BC** *passant par A.*

Compléter le code de l'Exercice 4.15 pour construire le point D afin que ABCD soit un parallélogramme. S'inspirer également du code de l'Exemple 4.7 pour cette construction.

Exercice 4.16: **Rectangle comme un parallélogramme**

Par les diagonales

Résultat mathématique. Les diagonales d'un rectangle – en plus de se couper en leur milieu comme pour tout parallélogramme – sont **isométriques**.

Lorsque les points A, B et C d'un rectangle ABCD sont placés sur la figure il est alors possible de construire le sommet D comme symétrique de B par rapport à I, milieu de AC.

Compléter le code de l'Exercice 4.15 pour construire le point D et le parallélogramme ABCD avec la méthode vue à l'Exercice 4.11.

Exercice 4.17: **Rectangle et centre**

Autre approche. Puisque les diagonales du rectangle sont isométriques, nous commençons par la construction d'une diagonale, par exemple AC. La deuxième diagonale BD sera construite telle que BD=AC. Dans ce programme de construction nous utiliserons un cercle de diamètre AC.

Programmer une figure avec le segment d'extrémités les points A(0;0) et C(5;2). Construire le milieu I du segment AC puis le cercle de diamètre AC.

Exercice 4.18: **Rectangle et cercle**

Maintenant nous plaçons un point B où nous le souhaitons sur le cercle. La droite IB est construite puis le point D comme intersection entre la droite IB et le cercle.

Placer le point B d'abscisse 0.4 sur le cercle. Construire la droite IB et le point D comme d'intersection de IB et du cercle. Construire le polygone ABCD. **Indices :** *utiliser les messages à mots clés* `pointSurLigne:a:`, `droitePassantPar:et:` *et* `intersectionDe:et:`

Exercice 4.19: **Rectangle et diagonale**

4.2.4 Carré

Résultat mathématique. Un carré est un parallélogramme dont les côtés sont isométriques et perpendiculaires. Le carré cumule les propriétés du losange et du rectangle, il est à la fois un losange et un rectangle.

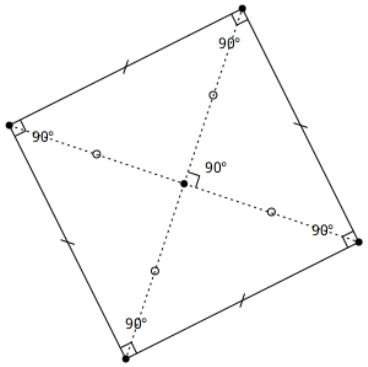


Figure 4.10: Carré

Troisième sommet

Ainsi pour un carré ABCD, connaissant A et B, le point C est tel que BA=BC et la droite BA est perpendiculaire à la droite BC.

Pour placer correctement un point C, nous construisons :

1. la droite passant par B et perpendiculaire à la droite AB ;
2. le cercle de centre B et passant par A ;
3. le point C intersection de cette droite perpendiculaire avec ce cercle.

Programmer une figure avec les points A(0;0) et B(5;1). Construire la droite passant par B perpendiculaire à la droite AB puis le cercle de centre B passant par A. Enfin, construire le point C comme intersection de cette droite et de ce cercle. **Indices :** utiliser les messages à mots clés `perpendiculaireA:passantPar:`, `cercleCentre:passantPar:` et `intersectionDe:et:`

Exercice 4.20: **Côtés adjacents isométriques et perpendiculaires**

Par les côtés opposés

Une fois les points A, B et C placés, il suffit de reprendre un programme de construction pour un parallélogramme déjà répété plusieurs fois.

Compléter le code de l'Exercice 4.20 pour construire le point D afin que ABCD soit un parallélogramme. S'inspirer également du code de l'Exemple 4.7 pour cette construction. Pour plus de clarté dans la figure finale, cacher les droites et cercle intermédiaires.

Exercice 4.21: **Carré comme un parallélogramme**

Par les diagonales

Résultat mathématique. Les diagonales d'un carré – en plus de se couper en leur milieu comme pour tout parallélogramme – sont isométriques et perpendiculaires.

Lorsque les points A, B et C d'un carré ABCD sont placés sur la figure il est alors possible de construire le sommet D comme symétrique de B par rapport à I, milieu de AC.

Compléter le code de l'Exercice 4.20 pour construire le point D et le carré ABCD avec la méthode vue à l'Exercice 4.11.

Exercice 4.22: **Carré et centre**

4.3 Triangles

Construire un triangle quelconque ne comporte pas de difficulté, il suffit d'indiquer les coordonnées des trois sommets du polygone.

```
| figure |
figure := DrGeoFigure nouveau.
figure polygone: {0@0 . 5@0 . 2@3}
```

Exemple 4.9: Triangle quelconque

4.3.1 Isocèle

Un triangle isocèle possède plusieurs propriétés offrant des approches différentes dans sa construction. Nous les explorons dans les sections suivantes.

Côtés isométriques

Résultat mathématique. Un triangle ABC isocèle en A a ses côtés AB et AC isométriques.

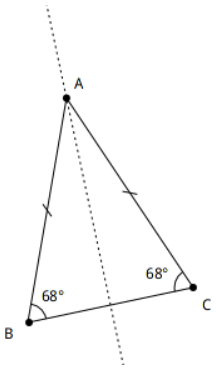


Figure 4.11: Triangle isocèle et côtés isométriques

Ainsi, étant donné un segment BC, le point A est construit au compas. Il est le point d'intersection de deux cercles de même rayon et de centre respectif B et C. Le rayon du cercle est la longueur des côtés AB et AC. Le triangle ABC est alors isocèle en A.

```
| figure b c cercle1 |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
cercle1 := figure cercleCentre: b rayon: 4
```

Exemple 4.10: Triangle isocèle et cercle

Compléter l'Exemple 4.10 pour construire le triangle ABC isocèle en A. Le point A sera construit comme l'intersection de deux cercles de rayon 4. Cacher les objets géométriques intermédiaires pour plus de clarté dans la figure finale.

Exercice 4.23: **Triangle isocèle, côtés isométriques**

Axe de symétrie

Résultat mathématique. Un triangle ABC isocèle en A admet comme axe de symétrie la médiatrice du segment BC.
Ou dit autrement, le point A appartient à la médiatrice du segment BC, cette médiatrice est également la hauteur issue de A du triangle ABC.

Connaissant B(5;1) et C(0;0), construire un point A sur la médiatrice du segment BC puis le triangle ABC. Utiliser les messages à mots-clés `mediatrice:` et `pointSurLigne:a:`

Exercice 4.24: **Triangle isocèle, axe de symétrie**

Angles isométriques

Résultat mathématique. Un triangle ABC isocèle en A a ses angles B et C isométriques.

Ainsi, étant donnés un segment BC et une mesure d'angle choisie – par exemple 50 degrés – nous construisons au rapporteur deux demi-droites d'origine B et C et formant un angle de 50 degrés avec BC. Ces deux droites sont sécantes en A et le triangle ABC ainsi formé est isocèle en A.

Pour résumer, dans cette construction les étapes sont les suivantes :

1. Choisir une mesure d'angle nommée *alpha* ;
2. Construire les deux demi-droites d'origine B et C faisant un angle *alpha* avec la droite BC ;
3. Construire A le point d'intersection de ces deux demi-droites.

Dr.Geo n'a pas d'outil de type rapporteur, la construction des demi-droites est donc délicate. Elle nécessite de s'appuyer sur la transformation géométrique rotation. Dans l'exemple suivant nous montrons le début du programme pour construire la demi-droite d'origine C format un angle donné avec BC.

```
| figure alpha1 b c b1 demiDroite1 |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
alpha1 := figure angleCentre: 10@10 de: 12@10 a: 12@13.
b1 := figure rotationDe: b parCentre: c etAngle: alpha1.
demiDroite1 := figure demiDroiteOrigine: c passantPar: b1
```

Exemple 4.11: Angle et rotation

Compléter l'Exemple 4.11 en construisant la demi-droite d'origine B et faisant un même angle avec BC, puis le point d'intersection A et le triangle ABC. Faire un croquis montrant la deuxième rotation nécessaire peut aider.

Exercice 4.25: **Triangle isocèle et angle***

Dans la figure finale, ce qui serait amusant c'est de modifier l'angle à la souris. Pour cela il suffit de rendre visible un des points de l'angle, par exemple celui de coordonnées (12;10).

A la fin du programme de l'Exercice 4.25, rendre visible le point de coordonnées (12;10). Cacher tous les objets géométriques intermédiaires pour plus de clarté dans la figure finale.

Exercice 4.26: **Triangle isocèle et angle variable**

Dans les sections précédentes nous avons écrit trois programmes différents pour la construction d'un triangle isocèle. Selon la propriété du triangle isocèle utilisée, le programme était plus ou moins simple à concevoir. Souvent en informatique, il existe différentes approches pour résoudre un même problème. Une bonne connaissance du domaine – ici le triangle isocèle – permet de choisir la méthode la plus simple dans la conception du programme informatique. Ici s'appuyer sur l'axe de symétrie du triangle produisait clairement le programme le plus simple.

4.3.2 Équilatéral

Côtés isométriques

Résultat mathématique. Un triangle ABC équilatéral a ses côtés AB, AC et BC isométriques.

Ainsi étant donné un segment BC, le point A est construit au compas tel que BA=CA=BC. A est un point d'intersection des deux cercles de **même rayon BC** et de centre respectif B et C.

En complétant l'Exemple 4.10, écrire un programme pour construire le triangle ABC équilatéral. Le point A sera construit comme l'intersection de deux cercles de rayon BC. Cacher les objets géométriques intermédiaires pour plus de clarté dans la figure finale.

Exercice 4.27: **Triangle équilatéral, côtés isométriques**

Axes de symétrie

Résultat mathématique. Un triangle ABC équilatéral admet trois axes de symétrie, les médiatrices de ses trois côtés.

Ainsi étant donné un segment BC, le point A se construit comme une intersection de la médiatrice de BC et un cercle de centre B ou C et de rayon BC.

En complétant l'Exemple 4.10, écrire un programme pour construire le triangle ABC équilatéral. Le point A sera construit comme l'intersection de la médiatrice de BC et d'un cercle de rayon BC. Cacher les objets géométriques intermédiaires pour plus de clarté dans la figure finale.

Exercice 4.28: **Triangle équilatéral, axes de symétrie**

4.3.3 Triangle rectangle

Résultat mathématique. Un triangle ABC rectangle en B admet un angle droit ABC.

Connaissant B(5;1) et C(0;0), construire un point A sur la droite perpendiculaire au segment BC en B. Construire ensuite le triangle ABC. Utiliser les messages à mots-clés `perpendiculaireA:passantPar:` et `pointSurLigne:a:`

Exercice 4.29: **Triangle rectangle**

4.3.4 Triangle rectangle isocèle

Résultat mathématique. Un triangle ABC isocèle rectangle en B admet un angle droit ABC et deux côtés isométriques BA et BC.

Connaissant B(5;1) et C(0;0), construire un point A sur la droite perpendiculaire au segment BC en B tel que BA=BC. Construire ensuite le triangle ABC. Utiliser les messages à mots-clés `perpendiculaireA:passantPar:` et `cercleCentre:passantPar:`

Exercice 4.30: **Triangle rectangle isocèle**

Le codage permet de rendre visible des propriétés sur des angles ou des segments isométriques.

Reprendre la solution de l'Exercice 4.30 pour cacher les constructions intermédiaires, marquer les segments isométriques et afficher la valeur de l'angle droit. Utiliser les messages à mots-clés `angleGeometriqueCentre:de:a:` et `marquerAvecSimpleTrait`

Exercice 4.31: **Triangle rectangle isocèle codé**

4.4 Droites remarquables du triangle

Dans le triangle, nous connaissons 4 droites remarquables : les médiatrices, les bissectrices, les hauteurs et les médianes.

4.4.1 Médiatrices

Résultat mathématique. Dans un triangle, les trois médiatrices construites à partir des côtés du triangle se coupent un même point M. Les médiatrices sont dites *concourantes* en ce point M, c'est le point d'intersection des trois médiatrices.
Pour rappel, la médiatrice d'un segment est une droite perpendiculaire à ce segment en son milieu.

Une figure Dr.Geo comprend les messages `mediatriceDe:` et `meditraiceDe:a:` pour construire une médiatrice à partir d'un segment ou de deux points – voir l'annexe des méthodes pour leur utilisation.

Construire un triangle de sommets A(2;1), B(7;2) et C(4;7) puis les 3 médiatrices de chacun des 3 côtés AB, BC et AC. Nommer M le point d'intersection de ces médiatrices.

Exercice 4.32: **Médiatrices du triangle**

Résultat mathématique. Le point M, intersection des médiatrices, est le centre d'un cercle qui passe par les trois sommets A, B et C. C'est le *cercle circonscrit* du triangle ABC.

Compléter l'Exercice 4.32 en construisant le cercle circonscrit au triangle ABC.

Exercice 4.33: **Cercle circonscrit d'un triangle**

4.4.2 Bissectrices

Résultat mathématique. Dans un triangle, les trois bissectrices des trois angles du triangle sont concourantes en un point O.
Pour rappel, la bissectrice d'un angle coupe celui-ci en deux angles isométriques.

Une figure Dr.Geo comprend les messages `bissectriceDe:` et `bissectriceSommet:cote1:cote2:` pour construire une bissectrice à partir d'un angle géométrique ou de trois points.

Construire un triangle de sommets A(2;1), B(7;2) et C(4;7) puis les 3 bissectrices de chacun des ses 3 angles. Nommer O le point d'intersection de ces bissectrices.

Exercice 4.34: **Bissectrices du triangle**

Résultat mathématique. Le point O est le centre du *cercle inscrit* dans le triangle ABC. Le rayon de ce cercle est la distance du point O à n'importe quel côté du triangle : la distance de O à AB, de O à BC et de O à AC est la même. Ainsi pour tracer ce cercle inscrit, il faut d'abord construire et mesurer une des distances de O à AB, BC ou AC.

Compléter l'Exercice 4.34 en construisant le cercle inscrit dans le triangle ABC. Le rayon du cercle sera d'abord construit comme la distance de O à un des côtés du triangle.

Exercice 4.35: **Cercle inscrit d'un triangle***

4.4.3 Hauteurs

Résultat mathématique. Dans un triangle, les trois hauteurs issues des trois sommets sont concourantes en un point H appelé orthocentre du triangle.
Pour rappel, une hauteur d'un triangle est une droite passant par un sommet et perpendiculaire au côté opposé à ce sommet.

Construire un triangle de sommets A(2;1), B(7;7) et C(4;5) puis les 3 hauteurs issues de chacun de ses 3 sommets. Nommer H le point d'intersection de ces hauteurs.

Exercice 4.36: **Hauteurs du triangle**

4.4.4 Médianes

Résultat mathématique. Dans un triangle, les trois médianes issues des trois sommets sont concourante en un point G appelé centre de gravité du triangle.
Pour rappel, une médiane d'un triangle est une droite passant par un sommet et le milieu du côté opposé à ce sommet.

Construire un triangle de sommets A(2;1), B(7;7) et C(4;5) puis les 3 médianes issues de chacun de ses 3 sommets. Nommer G le point d'intersection de ces médianes.

Exercice 4.37: **Médianes du triangle**

4.5 Angles

Depuis Dr.Geo, il est possible de construire deux types d'angle à partir de trois points :

- un angle géométrique dont la mesure est comprise entre 0° et 180°. Le message à envoyer à la figure est `angleGeometriqueCentre:de:a:`.
- un angle orienté dont la mesure est comprise entre 0° et 360°. L'angle est orienté dans le sens contraire des aiguilles d'une montre. Le message à envoyer à la figure est `angleCentre:de:a:`.

Observer comment les angles ci-dessous sont différents alors qu'ils sont construits à partir des mêmes points.

```
| figure |
figure := DrGeoFigure nouveau.
figure demiDroiteOrigine: 0@0 passantPar: -2@2.
figure demiDroiteOrigine: 0@0 passantPar: 3@1.
(figure angleGeometriqueCentre: 0@0 de: -2@2 a: 3@1) couleur: Color blue.
(figure angleCentre: 0@0 de: -2@2 a: 3@1) couleur: Color brown
```

Exemple 4.12: Angles géométrique et orienté

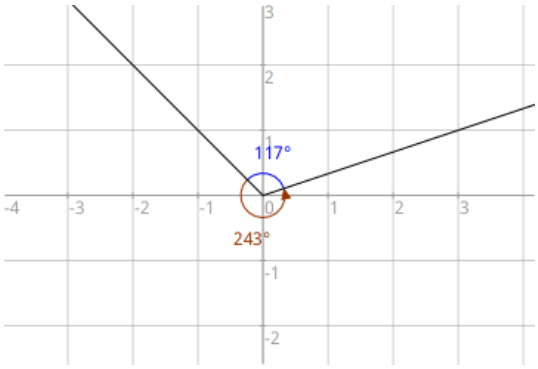


Figure 4.12: Angles géométrique (bleu) et orienté (marron avec flèche)

4.5.1 Angles correspondants

Résultat mathématique. Deux angles correspondants portés par deux droites parallèles sont isométriques.

Les angles sont placés du même côté des droites parallèles. Dans la figure ci-dessous, les droites AC et BD sont parallèles ; les angles BAC et EBD sont correspondants et isométriques.

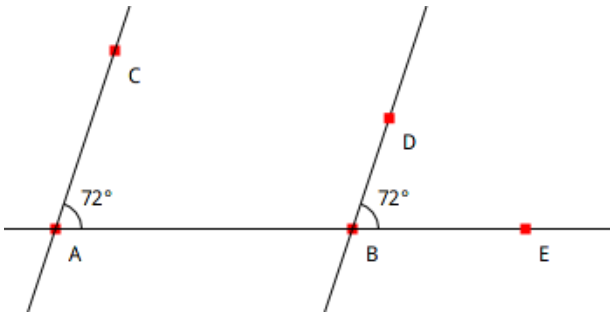


Figure 4.13: Angles correspondants portés par deux droites parallèles

Dans la figure interactive produite par le code ci-dessous, déplacer les points D et E de part et d'autre du point B pour observer le changement de mesure de l'angle EBD.

```
| figure a b c d e d1 d2 d3 |
figure := DrGeoFigure nouveau.
a:= (figure point: 0@0) nommer: 'A'.
b:= (figure point: 5@0) nommer: 'B'.
c:= (figure point: 1@3) nommer: 'C'.
d1 := figure droitePassantPar: a et: b.
d2 := figure droitePassantPar: a et: c.
d3 := figure paralleleA: d2 passantPar: b.
d:= (figure pointSurLigne: d3 a: 0.85) nommer: 'D'.
e:= (figure pointSurLigne: d1 a: 0.96) nommer: 'E'.
figure angleGeometriqueCentre: a de: b a: c.
figure angleGeometriqueCentre: b de: e a: d
```

Exemple 4.13: Angles correspondants

4.5.2 Angles alternes-internes

Résultat mathématique. Deux angles alternes-internes portés par deux droites parallèles sont isométriques.

Les angles sont placés de part et d'autre de la troisième droite et entre les deux droites parallèles. Dans la figure ci-dessous, les droites AC et BD sont parallèles ; les angles BAC et EBD sont alternes-internes et isométriques.

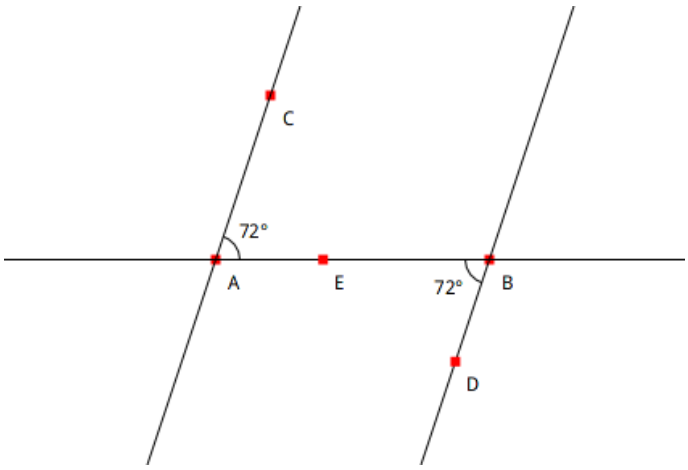


Figure 4.14: Angles alternes-internes portés par deux droites parallèles

Dans la figure interactive produite par le code ci-dessous, déplacer les points D et E de part et d'autre du point B pour observer le changement de mesure de l'angle EBD.

```
| figure a b c d e d1 d2 d3 |
figure := DrGeoFigure nouveau.
a:= (figure point: 0@0) nommer: 'A'.
b:= (figure point: 5@0) nommer: 'B'.
c:= (figure point: 1@3) nommer: 'C'.
d1 := figure droitePassantPar: a et: b.
d2 := figure droitePassantPar: a et: c.
d3 := figure paralleleA: d2 passantPar: b.
d:= (figure pointSurLigne: d3 a: 0.15) nommer: 'D'.
e:= (figure pointSurLigne: d1 a: 0.85) nommer: 'E'.
figure angleGeometriqueCentre: a de: b a: c.
figure angleGeometriqueCentre: b de: e a: d
```

Exemple 4.14: Angles alternes-internes

4.5.3 Somme des angles d'un triangle

Résultat mathématique. La somme des angles d'un triangle est toujours égale à 180°.

Nous allons construire une figure expliquant cette propriété. Nous écrivons d'abord le code d'une figure d'un triangle avec ses sommets, ses côtés et ses angles.

```
| figure a b c ab bc |
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
b := (figure point: 6@0) nommer: 'B'.
c := (figure point: 4@9) nommer: 'C'.
(figure segmentDe: a a: b) normal.
ab := figure droitePassantPar: a et: b.
bc := (figure segmentDe: b a: c) normal.
(figure segmentDe: a a: c) normal.
(figure angleGeometriqueCentre: b de: a a: c) couleur: Color red.
(figure angleGeometriqueCentre: a de: b a: c) couleur: Color blue.
(figure angleGeometriqueCentre: c de: a a: b) couleur: Color brown
```

Exemple 4.15: Triangle et angles

Complétons maintenant ce code pour mettre en évidence deux angles particuliers de cette figure.

Compléter l'Exemple 4.15 en traçant une droite d1 passant par A et parallèle à BC. Placer sur d1 un point M d'abscisse 0.9 puis construire l'angle géométrique MAC. Que dire des angles ACB et MAC ?

Exercice 4.38: **Triangle et angles**

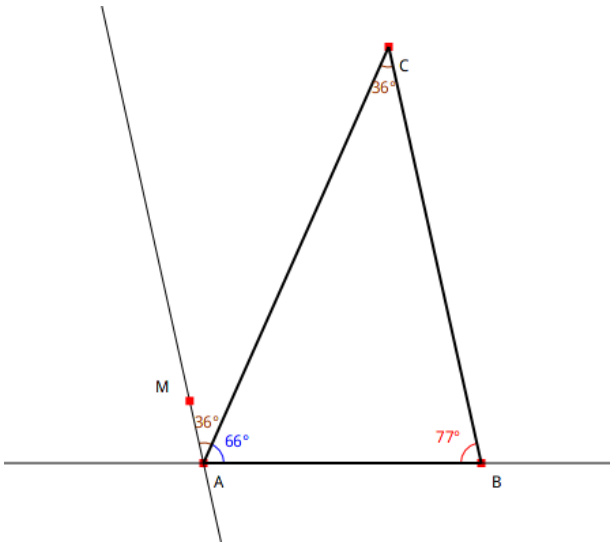


Figure 4.15: Triangle et angles alternes-internes

Nous complétons à nouveau le code de la figure pour mettre en évidence une autre paire d'angles particuliers.

Compléter le code de l'Exercice 4.38 en plaçant sur la droite AB un point N d'abscisse 0.2 puis construire l'angle géométrique MAN. Que dire des angles ABC et MAN ?

Exercice 4.39: **Triangle et angles encore**

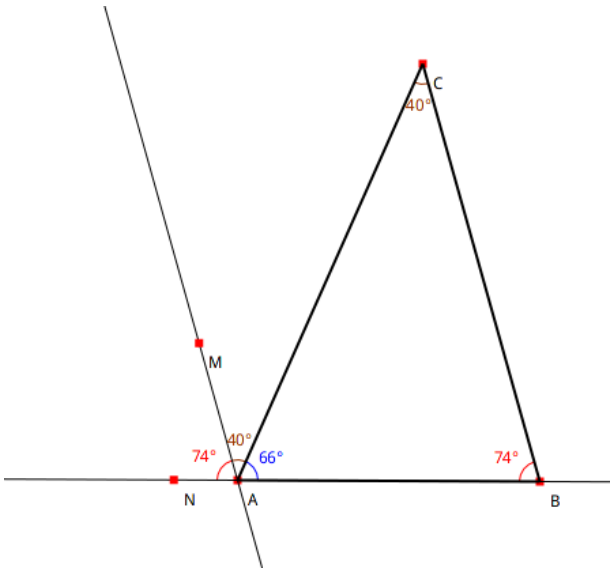


Figure 4.16: Triangle et angles correspondants

Que conclure sur la somme des angles d'un triangle ? Lorsque les points A, B ou C sont déplacés, la conclusion précédente est-elle toujours vraie ?

4.5.4 Somme des angles d'un quadrilatère

Résultat mathématique. Un polygone – quadrilatère – est convexe lorsqu'il contient chaque segment joignant deux de ses points.

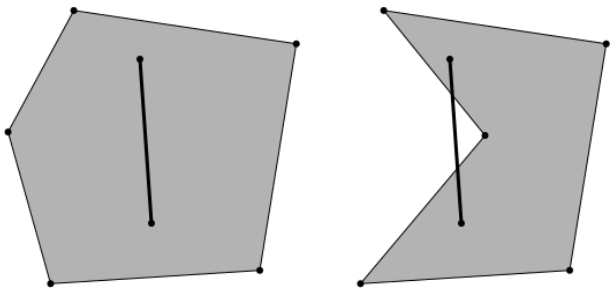


Figure 4.17: Polygones convexe et non convexe

En construisant une diagonale d'un quadrilatère convexe, cela revient à découper celui-ci en deux triangles. Il devient alors évident que la somme de ses angles est de 2 * 180°, soit 360°.

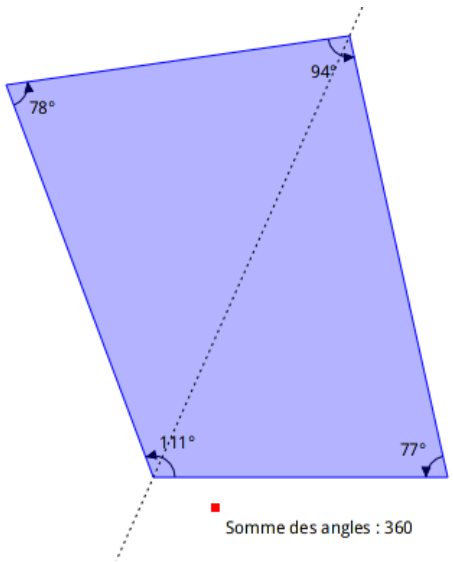


Figure 4.18: Somme des angles d'un quadrilatère convexe

Cette figure est produite avec le code suivant :

```
| figure ancre a b c d |
figure := DrGeoFigure nouveau.
figure polygone: { 0@0. 6@0. 4@9. -3@8 }.
(figure droitePassantPar: 0@0 et: 4@9) pointille.
a := figure angleCentre: 0@0 de: 6@0 a: -3@8.
b := figure angleCentre: 6@0 de: 4@9 a: 0@0.
c := figure angleCentre: 4@9 de: -3@8 a: 6@0.
d := figure angleCentre: -3@8 de: 0@0 a: 4@9.
ancre := figure point: -2 @ -2.
figure point: [
   ancre nommer: 'Somme des angles : ',
      (a mathItem  degreeAngle
      + b mathItem degreeAngle
      + c mathItem degreeAngle
      + d mathItem degreeAngle) rounded asString]
```

Exemple 4.16: Angles d'un quadrilatère convexe

Observer comment la somme des angles est calculée : à chaque angle `a`, `b`, `c` et `d` il est demandé son modèle mathématique en lui envoyant le message `mathItem`. A celui-ci, il est demandé la mesure de l'angle en degrés par le message `degreeAngle`[1].

Lorsque le quadrilatère est non-convexe et pour peu qu'il ne soit pas croisé – ses diagonales sont à l'intérieur du quadrilatère – le code de cette figure reste valide et la somme des angles est toujours de 360°.

[1] Un angle se mesure en degrés, mais aussi en radians et grades

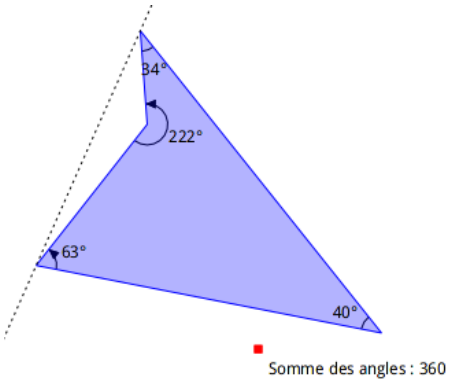


Figure 4.19: Somme des angles d'un quadrilatère non convexe, non croisé

En revanche, le code de cette figure ne convient pas pour afficher la somme des angles lorsque le quadrilatère est croisé. Certains angles se retrouvent à l'extérieur du quadrilatère. L'utilisation des angles orientés dans le code source ne convient pas dans ce cas.

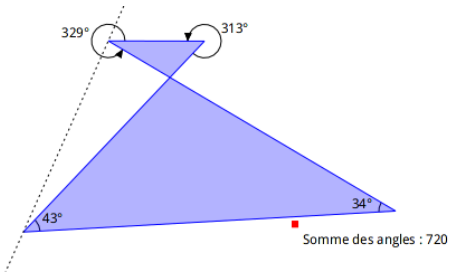


Figure 4.20: Somme des angles d'un quadrilatère non convexe, et **croisé**

Modifier le code de l'Exemple 4.16 pour afficher correctement les angles d'un quadrilatère croisé et ainsi avoir la somme correcte de ses angles. Indice : construire des angles géométriques avec le message `angleGeometriqueCentre:de:a:`.

Exercice 4.40: **Somme des angles d'un quadrilatère croisé**

Dans la figure interactive alors produite, et en manipulant la diagonale, que conclure sur la somme des angles d'un quadrilatère croisé ?

4.6 Transformations géométriques

Cinq transformations géométriques sont disponibles pour la programmation : symétrie centrale, symétrie axiale, translation, rotation et homothétie (changement d'échelle). Nous allons découvrir l'utilisation de ces transformations : un exemple montre comment coder une transformation puis un exercice à faire est proposé.

4.6.1 Symétrie centrale

Résultat mathématique. Cette transformation est définie par un point O appelé centre de la symétrie. Déplacer une figure par une symétrie de centre O, c'est faire tourner cette figure d'un demi-tour autour du point O. La symétrie centrale est une isométrie : la figure et sa transformée ont la même forme et les mêmes dimensions, les figures sont superposables.

Dans l'Exemple 4.17 ci-dessous, le triangle est transformé par la symétrie de centre O(-1;-1).

```
| figure triangle o |
figure := DrGeoFigure nouveau.
o := figure point: -1 @ -1.
triangle := figure polygone: { 0@0. 3@4. 5@2 }.
figure symetriqueDe: triangle selonCentre: o
```

Exemple 4.17: Image d'un triangle par une symétrie centrale

Maintenant s'entraîner avec l'exercice suivant.

Construire l'image d'un carré de côté 4 unités par une symétrie centrale de centre O(3;-2).

Exercice 4.41: **Image d'un carré par une symétrie centrale**

4.6.2 Symétrie axiale

Résultat mathématique. Cette transformation est définie par une droite d appelée axe de la symétrie. En pliant la feuille suivant le droite d, la figure et sa transformée se superposent. Cette transformation est également une isométrie.

Dans l'Exemple 4.18 ci-dessous, le triangle est transformé par la symétrie d'axe la droite d passant par les points (-3;0) et (5;-5).

```
| figure triangle d |
figure := DrGeoFigure nouveau.
d := figure droitePassantPar: -3 @ 0 et: 5 @ -5.
triangle := figure polygone: { 0@0. 3@4. 5@2 }.
figure symetriqueDe: triangle selonAxe: d
```

Exemple 4.18: Image d'un triangle par une symétrie axiale

Maintenant s'entraîner avec l'exercice suivant.

Construire l'image d'un carré de côté 4 unités par une symétrie axiale d'axe la droite d passant par les points (-3;3) et (-8;0).

Exercice 4.42: **Image d'un carré par une symétrie axiale**

4.6.3 Translation

Résultat mathématique. Cette transformation est définie par un vecteur v appelé vecteur de translation. Déplacer une figure par une translation, c'est faire glisser cette figure selon la direction, le sens et la longueur du vecteur v, sans la faire tourner. Cette transformation est une isométrie.

Dans l'Exemple 4.19 ci-dessous, le triangle est translaté selon le vecteur v d'origine le point A(1;1) et d'extrémité B(6;5).

```
| figure triangle a b v |
figure := DrGeoFigure nouveau.
a := figure point: 1 @ 1.
b := figure point: 6 @ 5.
v := figure vecteurOrigine: a extremite: b.
triangle := figure polygone: { 1@3. 3@7. 5@5 }.
figure translationDe: triangle parVecteur: v
```

Exemple 4.19: Image d'un triangle par une translation

Maintenant s'entraîner avec l'exercice suivant.

Construire l'image d'un carré de côté 4 unités par une translation de vecteur v d'origine le point A(-1;-1) et d'extrémité le point B(-4;-3).

Exercice 4.43: **Image d'un carré par une translation**

4.6.4 Rotation

Résultat mathématique. Cette transformation est définie par un point O appelé centre de la rotation, et un angle a appelé angle de la rotation. Déplacer une figure par une rotation de centre O et d'angle a, c'est faire tourner cette figure autour du point O d'un angle de a. La mesure de l'angle a est positive ou négative – sens contraire des aiguilles d'une montre (sens anti-horaire) ou sens des aiguilles d'une montre (sens horaire).

L'exemple ci-dessous montre les rotations d'un triangle avec une mesure d'angle positive puis une mesure d'angle négative.

```
| figure triangle o a1 a2 |
figure := DrGeoFigure nouveau.
o := figure point: 1 @ 1.
a1 := 70 degreesToRadians.
a2 := -70 degreesToRadians.
triangle := figure polygone: { 1@3. 3@7. 5@5 }.
figure rotationDe: triangle parCentre: o etAngle: a1.
figure rotationDe: triangle parCentre: o etAngle: a2
```

Exemple 4.20: Images d'un triangle par deux rotations

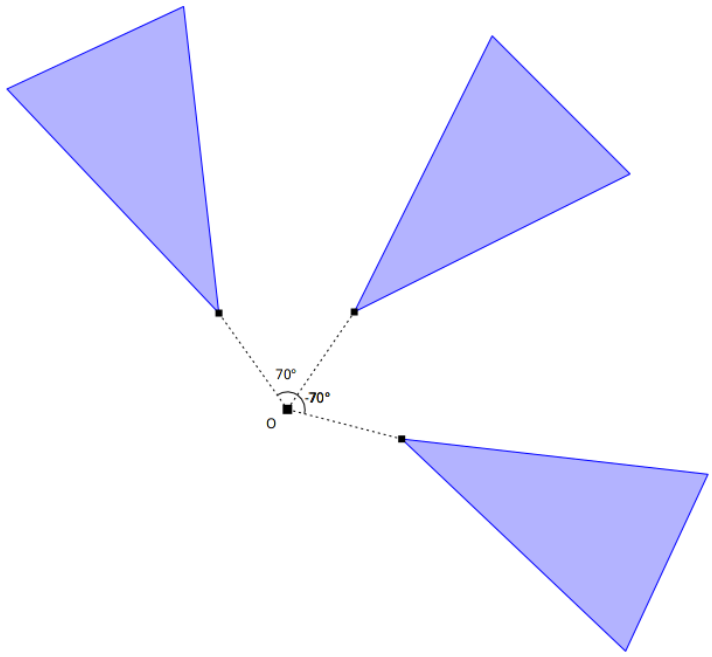


Figure 4.21: Deux rotations avec des angles de 70° et -70°

Maintenant s'entraîner avec l'exercice suivant.

Construire l'image d'un carré de côté 4 unités par une rotation de centre O(0;0) et d'angle 90° et par une deuxième rotation de centre O(0;0) et d'angle -90°.

Exercice 4.44: **Image d'un carré par deux rotations**

4.6.5 Homothétie

Résultat mathématique. Cette transformation géométrique est définie par un point O appelé centre de l'homothétie, et une valeur numérique k appelée rapport de l'homothétie. Elle agrandit ou réduit proportionnellement une figure selon le rapport k
Lorsque le rapport k est entre -1 et 1, la figure transformée est réduite.
Cette transformation n'est pas une isométrie, elle ne conserve par les longueurs.

```
| figure triangle o k1 k2 |
figure := DrGeoFigure nouveau.
o := figure point: 8 @ 7.
k1 := 1/4.
k2 := -1/3.
triangle := figure polygone: { 1@3. 3@7. 5@5 }.
(figure homothetieDe: triangle parCentre: o etFacteur: k1)
   nommer: Réduite et dans le même sens'.
(figure homothetieDe: triangle parCentre: o etFacteur: k2)
   nommer: 'Réduite et inversée'
```

Exemple 4.21: Images réduites d'un triangle par une homothétie

Lorsque le rapport k est supérieur à 1 ou inférieur à -1, la figure est agrandie par rapport à la figure originale.

```
| figure triangle o k1 k2 |
figure := DrGeoFigure nouveau.
o := figure point: 8 @ 7.
k1 := 3.
k2 := -2.
triangle := figure polygone: { 1@3. 3@7. 5@5 }.
(figure homothetieDe: triangle parCentre: o etFacteur: k1)
   nommer: 'Agrandie et dans le même sens'.
(figure homothetieDe: triangle parCentre: o etFacteur: k2)
   nommer: 'Agrandie et inversée'
```

Exemple 4.22: Images agrandies d'un triangle par une homothétie

Maintenant s'entraîner avec l'exercice suivant.

Construire l'image d'un carré de côté 4 unités par une homothétie de centre A(-8;5) et de rapport -1/2 et par une deuxième homothétie de centre B(4;-7) et de rapport 5/2.

Exercice 4.45: **Image d'un carré par deux homothéties**

4.6.6 Transformer une collection d'objets

Dans les exemples et les exercices précédents nous transformions des objets simples tels que des triangles et des carrés. Parfois nous souhaitons transformer un groupe d'objets, par exemple un carré avec un cercle inscrit à l'intérieur comme dans l'Exercice 2.12. L'exemple suivant montre qu'il est possible de transformer un à un les quatre côtés du segment et le cercle.

```
| figure c1 c2 c3 c4 cercle d |
figure := DrGeoFigure nouveau.
c1 := figure segmentDe: -2 @ 2 a: 2 @ 2.
c2 := figure segmentDe: 2 @ 2 a: 2 @ -2.
c3 := figure segmentDe: 2 @ -2 a: -2 @ -2.
c4 := figure segmentDe: -2 @ -2 a: -2 @ 2.
cercle := figure cercleCentre: 0 @ 0 rayon: 2.
d := figure droitePassantPar: -7 @ 0 et: 0 @ -8.
figure symetriqueDe: c1 selonAxe: d.
figure symetriqueDe: c2 selonAxe: d.
figure symetriqueDe: c3 selonAxe: d.
figure symetriqueDe: c4 selonAxe: d.
figure symetriqueDe: cercle selonAxe: d
```

Exemple 4.23: Symétrique d'un carré et d'un cercle inscrit

Nous remarquons que les 5 dernières lignes du code sont quasiment identiques. Cela indique souvent que ce code peut-être modifié afin d'être moins répétitif. Une première chose que nous pouvons faire c'est de grouper toutes les parties de notre figure (les 4 côtés du carré et le cercle) dans une collection d'objet. Dans le langage Pharo il existe des objets de type `Collection` pour y mettre toutes sortes d'objets comme des constructions géométriques.

Nous avons déjà rencontré des collections de type tableau avec le code `figure polygone: { 1@0. 5@0. 5@4 . 1@4 }`. Dans cet exemple, une collection avec 4 objets de type coordonnées de points est créé. A la place de ces coordonnées nous pouvons placer les 5 parties géométriques de notre figure à transformer – les côtés du carré et le cercle inscrit à l'intérieur. Observer alors dans l'Exemple 4.23 ci-dessous comment le code a été réécrit.

```
| figure collection d |
figure := DrGeoFigure nouveau.
d := figure droitePassantPar: -7 @ 0 et: 0 @ -8.
collection := {figure segmentDe: -2 @ 2 a: 2 @ 2 .
   figure segmentDe: 2 @ 2 a: 2 @ -2 .
   figure segmentDe: 2 @ -2 a: -2 @ -2 .
   figure segmentDe: -2 @ -2 a: -2 @ 2 .
   figure cercleCentre: 0 @ 0 rayon: 2}.
collection faire: [:forme | figure symetriqueDe: forme selonAxe: d]
```

Exemple 4.24: Symétrique d'un groupe d'objets

Lors de la création de la collection, les instructions pour créer les parties constituantes de la figure sont séparées par des ".". Ensuite l'envoi du message `faire:` à cette collection crée le symétrique de chacune de ses formes. Le nombre de lignes du code source du programme est ainsi presque divisé par 2.

Faire les exercices suivants pour s'entraîner.

Compléter le code de l'Exemple 4.24 pour y ajouter les diagonales du carré et le centre du cercle.

Exercice 4.46: **Symétrique d'une collection plus complexe**

Modifier la transformation du code de l'Exemple 4.24: utiliser une homothétie de centre (-10;-10) et rapport 1/4.

Exercice 4.47: **Homothétie d'une collection**

Observer les deux figures ci-dessous pour comprendre avec quels segments, cercles et polygone elles sont construites :

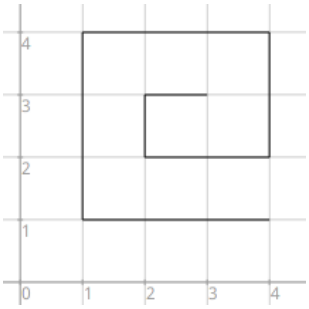


Figure 4.22: Spirale

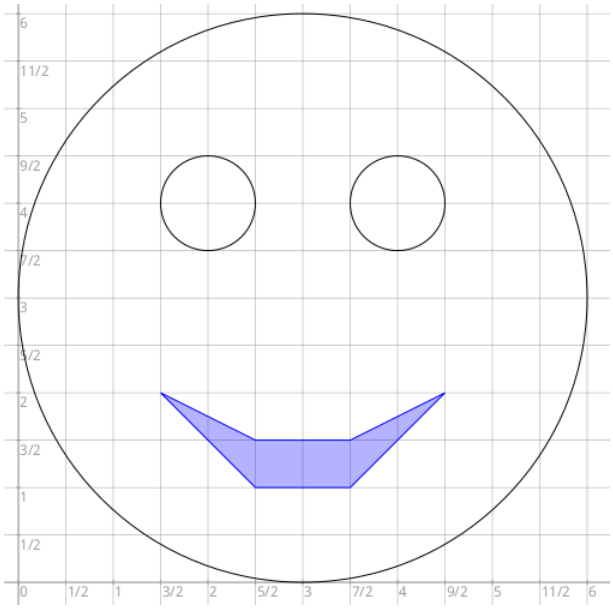


Figure 4.23: Smiley

Écrire le code pour construire la Figure 4.22. Placer les formes de cette construction – segments – dans une collection comme dans l'Exemple 4.24. Appliquer à cette collection une symétrie axiale d'axe la droite verticale passant par le point (4;0). Pour un meilleur effet esthétique cacher la droite.

Exercice 4.48: **Spirale**

Écrire le code pour construire la Figure 4.23. Placer les formes de cette construction – cercles et polygone – dans une collection comme dans l'Exemple 4.24. Appliquer à cette collection une symétrie central de centre le point (6;6).

Exercice 4.49: **Smiley**

Pour Résumer. Placer des objets géométriques dans une collection permet d'appliquer alors une transformation sur chacun de ces objets. Le code informatique écrit est alors plus court, cela s'appelle parcourir une collection pour y faire un traitement informatique – ici appliquer une transformation géométrique. Cette pratique est très importante en programmation.

4.6.7 Transformations en cascade

Pour prolonger l'étude précédente, il serait intéressant d'appliquer à nouveau une transformation géométrique sur les objets déjà transformés. Recopier le code de l'Exemple 4.25.

```
| figure collection v |
figure := DrGeoFigure nouveau.
v := figure vecteur: 2 @ 0.
collection := {figure segmentDe: 0 @ 0 a: 1 @ 3.
   figure segmentDe: 1 @ 3 a: 1 @ 3.
   figure segmentDe: 1 @ 3 a: 2 @ 0}.
5 foisRepete: [
   collection := collection collecter: [ :forme |
      figure translationDe: forme parVecteur:  v ] ]
```

Exemple 4.25: Triangles à gogo

Dans cet exemple, deux boucles imbriquées – l'une dans l'autre – sont utilisées. La première boucle est `foisRepete:`, ici elle **répète 5 fois** son bloc de code en paramètre. Dans le bloc de code de cette première boucle se trouve la deuxième boucle `collecter:`.

Comme avec le message `faire:`, `collecter:` translate chacun des objets de la collection mais, **en plus**, il collecte ces objets translatés dans une nouvelle collection. Celle-ci est ensuite affectée à la variable `collection`. Enfin la boucle `foisRepete:` est reprise et répétée encore quatre fois.

Voici un autre exemple au code plus complexe mais plus intéressant esthétiquement.

```
| figure collection d1 d2 d3 d4 d5|
figure := DrGeoFigure nouveau.
d1 := (figure droitePassantPar: 4@0 et: 4@5) pointille.
d2 := (figure droitePassantPar: 1@0 et: 1@5) pointille.
d3 := (figure droitePassantPar: -2@0 et: -2@5) pointille.
d4 := (figure droitePassantPar: -5@0 et: -5@5) pointille.
d5 := (figure droitePassantPar: -8@0 et: -8@5) pointille.
collection := {(figure segmentDe: 7@1 a: 4@1) normal.
   (figure segmentDe: 4@1 a: 4@4) normal.
   (figure segmentDe: 4@4 a: 7@4) normal.
   (figure segmentDe: 7@4 a: 7@2) normal.
   (figure segmentDe: 7@2 a: 5@2) normal.
   (figure segmentDe: 5@2 a: 5@3) normal.
   (figure segmentDe: 5@3 a: 6@3) normal}.
{d1 . d2 . d3 . d4 . d5} faire: [:axe |
   collection := collection collecter: [:forme |
      figure symetriqueDe: forme selonAxe: axe] ]
```

Exemple 4.26: Spirales à gogo

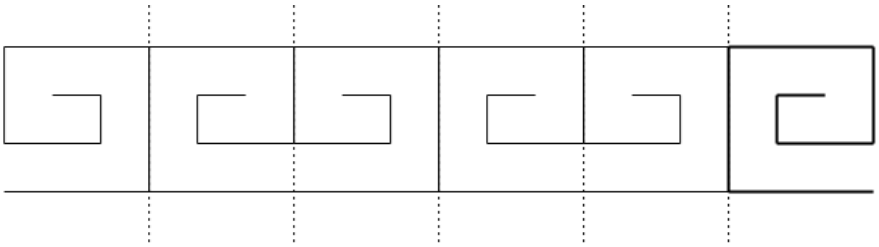


Figure 4.24: Frise de spirales

Ce code est plus technique, il utilise deux collections : une collection de droites `{d1 . d2 . d3 . d4 . d5}`, puis le message `collecter:` pour rassembler les objets transformés par les symétries axiales dans une nouvelle collection. Cette collection est alors affectée à la variable `collection`.

Une telle figure où un motif géométrique est transformé horizontalement à plusieurs reprises s'appelle une frise. Les symétries, translation et rotation sont utilisées pour produire des frises à partir d'un motif de base.

Modifier l'Exemple 4.26 afin d'appliquer une série de 5 symétries centrales. Le centre de ces symétries est à déterminer.

Exercice 4.50: **Une première frise**

4.7 Frise

Une frise est une bande, souvent horizontale, dont la vocation est de recevoir un décor, généralement constitué par la répétition d'un motif ornemental.

—*Wikipedia, Frise_(architecture), 5 octobre 2017*

4.7.1 Translation

La frise ci-dessous montre un motif déplacé successivement deux fois par une translation de vecteur v. Noter que le motif lui-même n'admet pas d'axe de symétrie.

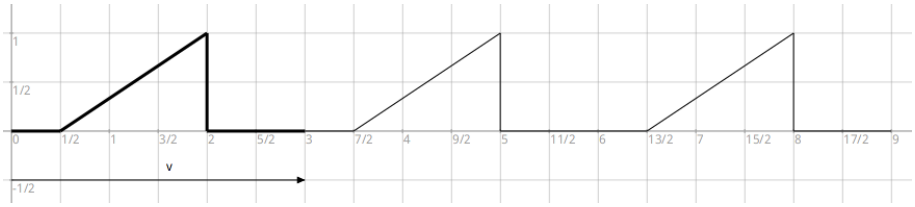


Figure 4.25: Une frise par une translation

Écrire le code informatique pour construire le motif de base – tracé épais – de la Figure 4.25. Placer le motif dans une collection, comme dans Exemple 4.24.

Exercice 4.51: **Motif pour translation**

À partir de ce motif de base, il est maintenant possible de construire une frise. Pour ce faire, le motif est glissé horizontalement de 3 unités vers la droite ou la gauche, c'est en fait la largeur du motif. Le motif est répété 5 fois à l'aide d'une boucle avec le message `foisRepete:`.

Compléter le code la solution de l'Exercice 4.51 pour construire une frise où le motif est répété 5 fois à droite. Indice : s'aider de l'Exemple 4.25 si nécessaire.

Exercice 4.52: **Translations du motif**

Voici un autre exemple avec une frise plus élaborée telle que rencontrée dans les rues de l'île de Rhodes Grecque.

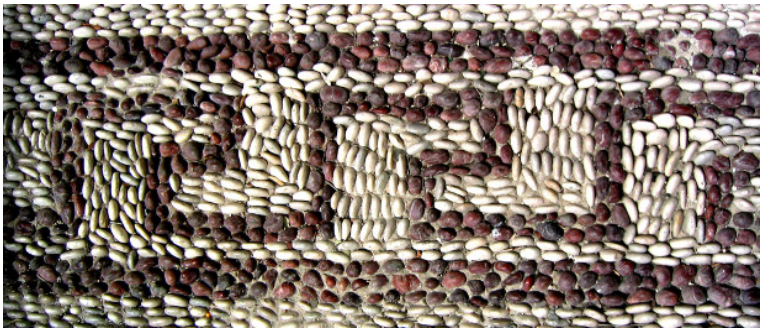


Figure 4.26: Frise vue dans une rue de Rhodes (galets de plage)

Cette frise est construite à partir d'un motif élémentaire répété horizontalement à l'aide d'une transformation géométrique, ici une translation. Ce motif élémentaire est une représentation symbolique des vagues de la mer, après tout Rhodes est une île Grecque de la méditerranée.

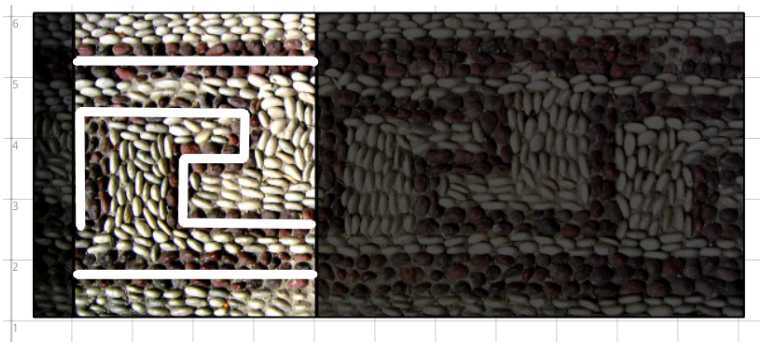


Figure 4.27: Mise en évidence du motif élémentaire d'une frise

Écrire le code informatique pour construire le motif de la Figure 4.27. Placer le motif dans une collection, comme dans l'Exemple 4.24. Si nécessaire s'aider d'une grille fournie en annexe. Il faudra alors bien repérer les coordonnées des sommets des segments. Voir Annexe G [Aides], page 159

Exercice 4.53: **Motif élémentaire**

Maintenant que le motif de base est codé et groupé dans une collection, il est possible de construire une frise. Pour ce faire, le motif est glissé horizontalement de 4 unités vers la droite ou la gauche, c'est en fait la largeur du motif (adapter selon la largeur du motif créé par chacun à l'Exercice 4.53).

Ce glissement est une transformation géométrique : une translation de vecteur v(4;0) par exemple.

A partir de la solution de l'Exercice 4.53 et en s'inspirant de l'Exemple 4.25, construire une frise avec 5 motifs de base.

Exercice 4.54: **Construction des vagues**

Dans la frise résultante, il est intéressant de mettre en évidence le motif de base.

Compléter la solution de l'Exercice 4.54 afin que le motif de base soit en trait épais.

Exercice 4.55: **Motif de base d'une frise**

Voici une autre frise construite à partir d'une spirale au tracé plus élaboré.

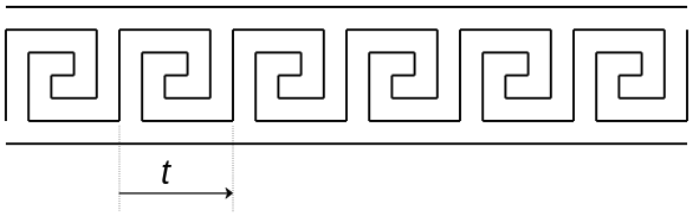


Figure 4.28: Encore une spirale

Ecrire le code pour construire cette frise. Prévoir 5 répétitions du motif de base.

Exercice 4.56: **Translation de spirale**

Pour construire cette frise, nous utilisons la translation. Dans la section suivante nous montrons que d'autres transformations géométriques sont utilisables pour construire une frise

4.7.2 Symétrie axiale

Observer la frise ci-dessous. Elle est construite par la translation de vecteur v du motif dessiné en trait épais.

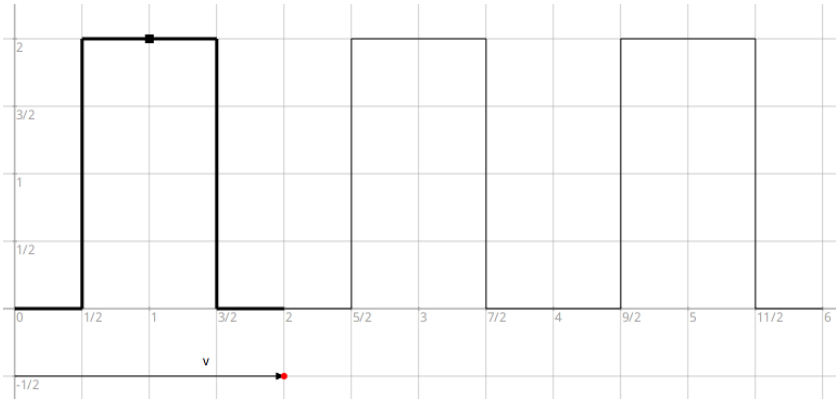


Figure 4.29: Une frise par translation mais...

Toutefois le motif admet un axe de symétrie comme montré dans la figure ci-dessous. Cela signifie que le motif de base de la frise se simplifie en un **motif élémentaire** composé d'uniquement 3 segments.

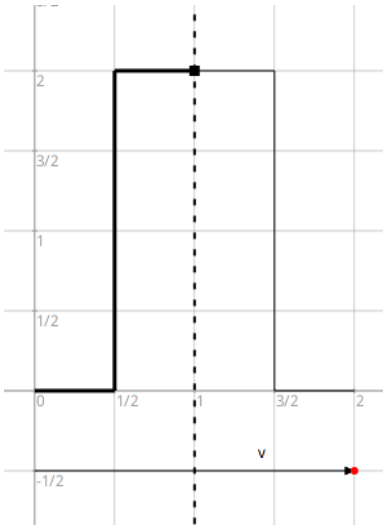


Figure 4.30: ...aussi une symétrie axiale

Écrire le code informatique pour construire le motif élémentaire de la Figure 4.30. Placer le motif dans une collection, comme dans l'Exemple 4.24. Si nécessaire s'aider d'une grille fournie en annexe. Voir Annexe G [Aides], page 159

Exercice 4.57: **Motif élémentaire**

Maintenant pour construire la frise, nous procédons en deux étapes avec deux transformations :

1. Une symétrie axiale pour compléter le motif élémentaire et obtenir le motif complet tel que montré dans la Figure 4.30. L'axe de la symétrie passe par les points (1;0) et (1;3).
2. Une translation de vecteur v pour déplacer le motif complété et construire l'ensemble de la frise montrée dans Figure 4.29

Aussi à l'étape 1, le symétrique du motif élémentaire doit être ajouté à la collection du motif. Cela se fait avec un code informatique nouveau, nous montrons comment le faire dans l'exemple ci-dessous.

```
| figure collection symetriques axe |
figure := DrGeoFigure nouveau.
axe := figure droitePassantPar: 1@0 et: 1@3.
collection := {figure segmentDe: 0@0 a: (1/2)@0.
   figure segmentDe: (1/2)@0 a: (1/2)@2.
   figure segmentDe: (1/2)@2 a: 1@2} commeCollectionOrdonnee.
symetriques := collection collecter: [:forme |
   figure symetriqueDe: forme selonAxe: axe].
collection ajouterTout: symetriques
```

Exemple 4.27: Compléter le motif élémentaire

Il y a deux nouveaux messages dans ce code informatique.

- Le message unaire `commeCollectionOrdonnee` envoyé à la collection créée avec le motif élémentaire. Ce message transforme la collection en une collection ordonnée où il est possible d'y ajouter des objets ultérieurement.

- Le message à mot clé `ajouterTout:` pour ajouter le contenue d'une collection à une autre. Ici les segments symétriques sont ajoutés à la collection du motif élémentaire. Après l'envoi de ce message, `collection` contient 6 segments et non plus 3 comme initialement.

Compléter l'Exemple 4.27 pour construire l'ensemble de la frise avec 5 répétitions du motif complété.

Exercice 4.58: **Translation du motif complété**

4.7.3 Symétrie centrale

Voici une frise qui se construit à partir d'un motif élémentaire qui est un losange – couleur foncée dans la figure ci-dessous. Ce motif est ensuite complété à l'aide d'une symétrie centrale. Puis, à l'aide d'une translation de vecteur V, la frise est construite dans son ensemble.

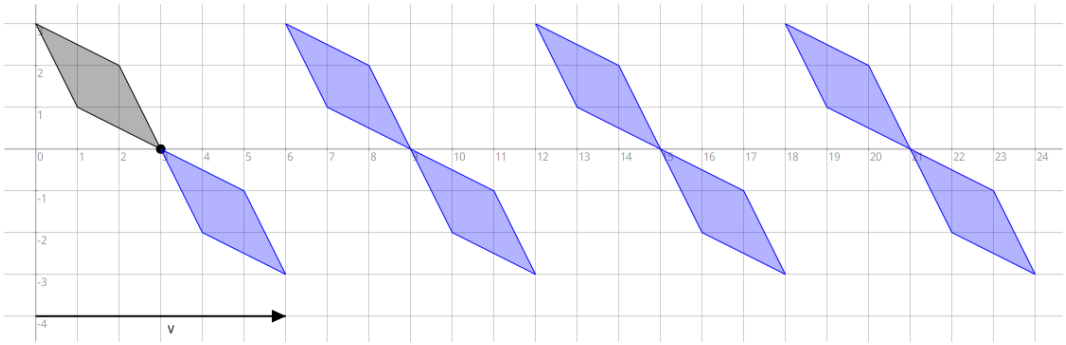


Figure 4.31: Frise et symétrie

Ecrire le code pour construire la frise telle que montrée dans la Figure 4.31. Adapter le code des exemples et des exercices de la section précédente **Symétrie axiale**. *Voici les étapes à suivre :*
- Coder le motif élémentaire, le losange. Voir Exercice 4.57.
- Coder la symétrie centrale. Voir Exemple 4.27.
- Translater les motifs plusieurs fois pour construire la frise. Voir Exercice 4.58.

Exercice 4.59: **Symétrie centrale puis translations**

4.7.4 Symétrie centrale et symétrie axiale

Voici une autre frise dont le motif élémentaire est toujours un losange – nommé *Motif 1* dans la figure ci-dessous. Ce motif est complété par une symétrie centrale comme dans l'exemple précédent, cela donne *Motif 2*. Ensuite, et c'est nouveau, une symétrie axiale d'axe la droite d est utilisée : *Motif 1* se transforme en *Motif 3* et *Motif 2* se transforme en *Motif 4*.

L'utilisation de la symétrie centrale puis de la symétrie axiale nous donne ce motif en forme d'étoile. Il est alors translaté autant de fois que souhaité pour obtenir une frise.

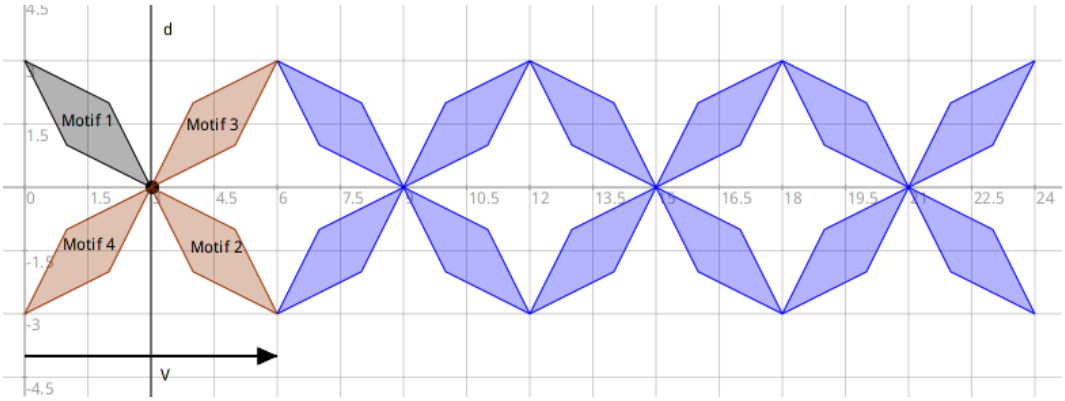


Figure 4.32: Frise et symétries

Ecrire le code pour construire la frise telle que montrée dans la Figure 4.32. Pour ce faire compléter le code de l'Exercice 4.59 pour y ajouter l'utilisation de la symétrie axiale.

Exercice 4.60: **Symétrie centrale, symétrie axiale puis translations**

Ce chapitre sur les frises géométriques est maintenant terminé. Il est possible d'en inventer et programmer beaucoup d'autres. Avec un peu d'observation, tu découvriras d'autres frises dans les monuments architecturaux, historiques ou récents.

5 Fonctions

Résoudre des problèmes numériques et algébriques
Résolution de problèmes en lien avec les notions étudiées (fonctions, diagrammes, expressions algébriques et équations).
Résolution de problèmes de proportionnalité.
—*Plan d'Études Romand*

5.1 Fonction linéaire

Résultat mathématique. Une fonction linéaire est de la forme $x \longmapsto ax$.
Le nombre a s'appelle la pente (ou facteur de linéarité, coefficient de linéarité, coefficient directeur). Sa représentation graphique est une droite passant par l'origine des axes.

Dr.Geo sait afficher les représentations graphiques – courbes – des fonctions. Il suffit de définir une fonction sous la forme d'un bloc de code ayant un argument, par exemple `[:x | 3 * x]` et d'utiliser le message à mot clé `courbeDe:de:a` envoyé à une figure.

```
| figure f |
figure := DrGeoFigure nouveau afficherAxes afficherGrille.
f := [:x | 3 * x].
figure courbeDe: f de: -5 a: 5
```

Exemple 5.1: Fonction linéaire de pente 3

Ici la droite est tracée pour x variant de -5 à 5.

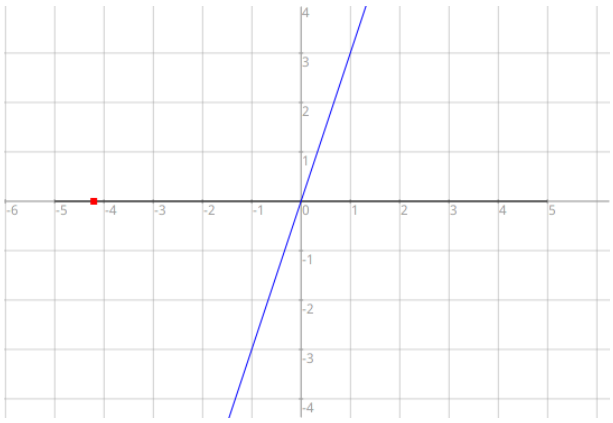


Figure 5.1: Représentation graphique d'une fonction linéaire de pente 3

Afficher la représentation graphique d'une fonction linéaire de pente -2.

Exercice 5.1: **Fonction linéaire de pente -2**

5.2 Pente d'une droite

Comment la pente a d'une fonction linéaire x $\longmapsto$ ax influence sa représentation graphique, la droite ?

Pour observer cette influence, dans Dr.Geo nous faisons varier la valeur a de la pente. Nous utilisons pour cela une réglette comme montré dans l'exemple ci-dessous.

```
| figure f a |
figure := DrGeoFigure nouveau afficherAxes afficherGrille echelle: 50.
a := figure entier: 1 a: 5 @ -5 min: -8 max: 8 nom: 'a' afficherValeur: true.
f := [:x | a valeur * x].
figure courbeDe: f de: -10 a: 10
```

Exemple 5.2: Fonction linéaire à pente un nombre entier variable

La réglette est créée avec le message à mot clé `entier:a:min:max:nom:afficherValeur:`, elle est affectée à la variable `a` :

- La valeur initiale de la réglette est 1 ;
- Elle est affichée à la coordonnées (5;-5) ;
- Sa valeur minimale est -10 ;
- Sa valeur maximale 10.

Pour obtenir la valeur actuelle de la réglette, lui demander avec le message `valeur` : cela se code `a valeur`.

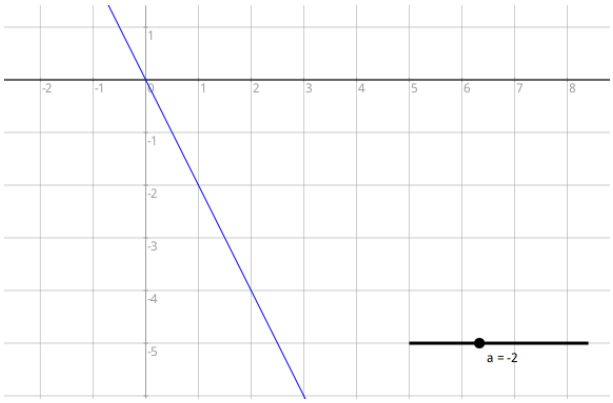


Figure 5.2: Représentation graphique d'une fonction linéaire de pente un entier variable

Faire varier la valeur de **a** *à l'aide de la réglette. Qu'observons nous lorsque la valeur de a est positive ? Négative ? Que se passe-t-il lorsque a est égale à zéro ?*

Exercice 5.2: **Observations fonction linéaire**

Modifier l'Exemple 5.2 afin d'utiliser une réglette avec un nombre **décimal**. *Chercher dans l'index des méthodes le message à mot clé à utiliser.*

Exercice 5.3: **Fonction linéaire à pente un nombre décimal variable**

5.3 Fonction affine

Résultat mathématique. Une fonction affine est de la forme $x \longmapsto ax + b$.
Sa représentation graphique est une droite.
Le nombre a est la pente de la droite.
Le nombre b est l'ordonnée à l'origine. C'est l'ordonnée – 2ème coordonnée – du point d'intersection de la droite avec l'axe vertical (ordonnées).

Pour comprendre le comportement de la droite d'une fonction affine selon les valeurs des nombres a et b, nous construisons une figure interactive où les nombres a et b sont variables.

En s'inspirant de l'Exemple 5.2, construire la droite d'une fonction affine. Les valeurs des nombres a et b seront obtenues par deux réglettes sur des nombres entiers.

Exercice 5.4: **Fonction affine dynamique**

Faire varier la valeur de **b** *à l'aide de la réglette. Qu'observons nous lorsque la valeur de* **b** *augmente ? Diminue ? Que se passe-t-il lorsque* **b** *est égale à zéro ? Que se passe-t-il lorsque* **a** *est égale à zéro ?*

Exercice 5.5: **Observations fonction affine**

5.4 Fonction quadratique

Résultat mathématique. Une fonction quadratique est de la forme $x \longmapsto ax^2 + bx + c$
La représentation graphique d'une fonction quadratique est une *parabole*. C'est une ligne qui n'est pas droite.

Pour comprendre le comportement de cette parabole selon les valeurs des nombres a, b et c, nous programmons une figure interactive où ces trois valeurs peuvent être modifiées par l'utilisateur.

En s'inspirant de l'Exemple 5.2, construire la parabole d'une fonction quadratique. Les valeurs des nombres a, b et c seront obtenues par trois réglettes sur des nombres décimaux puis entiers.

Exercice 5.6: **Fonction quadratique dynamique**

Faire varier la valeur de **a** *à l'aide de la réglette. Qu'observons nous selon le signe de a ? Que se passe-t-il lorsque a est égale à zéro ?*

Exercice 5.7: **Observations fonction quadratique**

5.5 Fonction puissance n-ième

Résultat mathématique. Une fonction puissance n-ième est une fonction de la forme $x \longmapsto x^n$ (n est un nombre naturel différent de zéro)

En s'inspirant de l'Exemple 5.2, construire la courbe d'une fonction puissance n-ième. La valeur du nombre n sera obtenue par une réglette de nombre entier compris entre 1 et 7.
Indice : utiliser le message `puissance:` pour calculer une puissance n-ième. Ce message est envoyé à la base (x) et son argument est l'exposant (n).

Exercice 5.8: **Fonction puissance n-ième dynamique**

5.6 Fonction homographique

Résultat mathématique. Une fonction homographique est, par exemple, une fonction de la forme $\mathrm{x} \longmapsto a/x$ (a et x différents de zéro)

En s'inspirant de l'Exemple 5.2, construire la courbe d'une fonction homographique de la forme a/x. La valeur du nombre a sera obtenue par une réglette de nombre entier compris entre -10 et 10.

Exercice 5.9: **Fonction homographique dynamique**

Faire varier la valeur de **a** *à l'aide de la réglette. Qu'observons nous selon le signe de a ?*

Exercice 5.10: **Observation fonction homographique**

Partie III
Pour aller plus loin dans la programmation

6 Figure Pharo

Les *Figures Pharo de Dr.Geo* – ou plus simplement figure dans la suite de ce texte – sont des figures écrites en langage Pharo. Il ne s'agit donc plus de construire une figure à l'aide de l'interface graphique de Dr.Geo mais plutôt de décrire une figure avec le langage Pharo et une API (interface de programmation dédiée)[1].

6.1 Exemples de figure Pharo

En lui-même Pharo est un langage de très haut niveau. Lorsqu'une figure est définie dans ce langage, nous disposons également de toute sa puissance pour par exemple définir récursivement telle partie de la figure, ou bien pour placer aléatoirement certains objets de telle sorte qu'à chaque ouverture de la figure, celle-ci soit légèrement différente. Bref, les FSD sont libérées du carcan de l'interface graphique tout en étant renforcées par le langage Pharo.

Une FSD est un code source Pharo à exécuter dans un espace de travail ...Clic arrière-plan → Outils → Espace de travail...[2] C'est une fenêtre texte depuis laquelle du code Pharo est écrit et exécuté. On peut aussi y coller du texte par *Ctrl-v*. Voir [workspace], page 100, pour plus d'information sur cet outil.

Nous allons étudier plusieurs exemples, chacun d'eux sera écrit dans un espace de travail et exécuté en sélectionnant le code puis la séquence de touches *Ctrl-d* pour *Do-it!*[3].

Commençons par étudier un exemple simple :

```
DrGeoFigure nouveau
```

C'est la plus petite description produisant une figure. Lors de son exécution, celle-ci va simplement créer une nouvelle figure vide. La figure Dr.Geo est affichée dans une fenêtre simplifiée puisqu'elle ne comporte pas la barre d'outils, seulement la barre des menus et les molettes.

Abordons un deuxième exemple :

```
| c item |
c := DrGeoFigure nouveau.
item := c point: 1.2 @ -2.
item nommer: 'A'.
```

Cette description définit une figure avec un point libre A de coordonnées initiales (1,2 ; -2). Quelques explications sur ce code :

- Deux variables temporaires *c* et *item* sont déclarées entre deux barres verticales `| |`. Il n'y pas de type, les variables sont toujours des références vers des objets.[4]
- Les objets sont ajoutés un à un à la figure en envoyant un message approprié à celle-ci. Ici le message à mot clé `#point:`, avec comme argument deux coordonnées, crée un point libre.

  Le résultat est une capsule[5] sur un objet géométrique "point" de Dr.Geo qu'il est possible de modifier par l'envoi de messages, ici `#nommer:` pour le renommer 'A'.

Poursuivons avec un troisième exemple :

```
| c triangle hasard m n p |
triangle := [:p1 :p2 :p3 |
```

[1] `https://fr.wikipedia.org/wiki/Interface_de_programmation`

[2] Le raccourci *Alt-k* fonctionne aussi lorsqu'aucune fenêtre est sélectionnée.

[3] Alternativement, c'est l'entrée à choisir dans le menu contextuel de l'espace de travail.

[4] Des instances de classes qui représentent des objets géométriques.

[5] Pour être précis, la capsule est un objet `DrGWrappedPoint` dont l'objectif est de simplifier la manipulation des objets géométriques de type point, pour aussi bien obtenir ses attributs ou modifier son style. Il est toujours possible d'accéder à l'objet point sous-jacent en envoyant le message `#mathItem` à la capsule. Il est alors possible d'utiliser les méthodes décrites dans le chapitre sur les scripts.

```
c segmentDe: p1 a: p2.
c segmentDe: p2 a: p3.
c segmentDe: p3 a: p1].
hasard := [5 - 10 auHasard].
c := DrGeoFigure nouveau.
m := c point: hasard valeur @ 0.
n := c point: 5 @ 0.
p := c point: hasard valeur @ 3.
triangle valeur: m valeur: n valeur: p.
```

Cet exemple est particulièrement intéressant, il nous montre trois choses importantes :

1. L'introduction d'une construction de plus haut niveau, non prévue au départ par Dr.Geo. Ici nous avons défini le bloc de code *triangle* qui, à partir de trois points, construit le triangle passant par ces trois points. Nous pouvons comparer ceci avec les macro-constructions mais avec une approche différente, orientée programmation.
2. La définition d'un bloc de code associé, ici nous avons défini *hasard* qui retourne un nombre entier compris entre -5 et 5. Nous utilisons ce bloc pour placer au hasard certains points de notre figure, ainsi à chaque exécution la figure est légèrement différente.
3. L'affectation du résultat d'une construction à une variable n'est pas obligatoire, nous l'utilisons lorsque nous souhaitons garder une référence de l'objet créé. Par exemple dans le bloc de code *triangle*, nous ne gardons pas de référence des segments créés, en revanche lorsque nous définissons nos trois points nous avons besoin de garder une référence dans des variables temporaires. Ainsi, lors de l'utilisation du bloc de code *triangle*, nous passons en paramètre les variables *m*, *n* et *p*.

Pour clore cette section, voici un dernier exemple :

```
| c a b d |
c := DrGeoFigure nouveau.
a := c point: 1@0.
b := c point: 5@0.
d := c line: a to: b.
a couleur: Color yellow;
   rond;
   large.
b cacher.
d tiret.
```

Deux points et une droite sont créés. Ensuite des commandes sont utilisées pour modifier l'aspect des objets, voire pour en cacher.

Nous avons terminé notre petite visite guidée des *Figures Pharo*. Dans les sections suivantes nous exposons l'ensemble des commandes disponibles.

6.2 Méthodes de référence pour les Figures Pharo

Pour construire un objet vous envoyez un message à la figure, l'objet construit est édité en lui envoyant des messages spécifiques.

Cependant, avant tout ajout d'un objet à une figure, cette dernière doit être créée avec la commande `DrGeoFigure nouveau`.

Dans la description de l'API des sections suivantes, pour plus de concision, l'objet capsule `DrGWrappedPoint` est noté `WrpPoint`, de même pour les autres capsules.

6.2.1 Commandes générales

`<une DrGeoFigure>` *nouveau* [Méthode sur `DrGeoFigure`]
⇒ Figure et affiche celle-ci dans une fenêtre. Le résultat est nécessaire pour créer des objets dans cette figure, il est donc important de la placer dans une variable.

```
| figure |
figure := DrGeoFigure nouveau.
```

`<une DrGeoFigure>` *minimal* [Méthode sur `DrGeoFigure`]
⇒ Figure et affiche celle-ci dans une fenêtre **sans décoration**.

C'est une alternative à la méthode `nouveau` pour créer une figure.

`supprimer` [Méthode sur `DrGeoFigure`]
Supprime la figure et ferme sa fenêtre

```
| figure |
figure := DrGeoFigure nouveau.
figure supprimer
```

`faire:` *bloc* [Méthode sur `DrGeoFigure`]
bloc, bloc de code Pharo contenant des instructions de construction et/ou d'animation de la figure interactive.

Exécute le bloc de code dans un processus en tâche de fond. A utiliser lorsque la construction doit se faire sous les yeux de l'utilisateur ou bien lorsque la figure est animée.

```
| figure point |
figure := DrGeoFigure nouveau.
point := figure point: 0@0.
figure do: [
   -5 a: 5 par: 0.1 faire: [:x |
      point deplacerA: x@(x cos * 3).
      (Delay forMilliseconds: 100) wait.
      figure actualiser]
]
```

`actualiser` [Méthode sur `DrGeoFigure`]
Mise à jour de la figure après modification des attributs de quelques items. La plupart du temps ce n'est pas nécessaire.

`pleinEcran` [Méthode sur `DrGeoFigure`]
La fenêtre de la figure est étendue pour couvrir tout l'écran.

`afficherGrille` [Méthode sur `DrGeoFigure`]
Affiche la grille de la figure.

`centrerVueEn:` *unPoint* [Méthode sur `DrGeoFigure`]
unPoint, coordonnées d'un point.

La figure est décalée afin d'afficher le point donné en argument au centre de la fenêtre.

```
figure centrerVueEn: 5@0
```

`echelle:` *unEntier* [Méthode sur `DrGeoFigure`]
unEntier, échelle de la figure. Une unité représente approximativement 1 pixel.

Modifie l'échelle de la figure.

```
figure echelle: 10
```

6.2.2 Point

`<WrpPoint> point:` *pointOuBloc* [Méthode sur `DrGeoFigure`]

pointOuBloc, un couple de coordonnées (x,y) ou un bloc de code retournant un couple de coordonnées. Dans le deuxième cas, le bloc est exécuté à chaque fois que les coordonnées du point sont demandées.

⇒ référence d'un point libre du plan de coordonnées *pointOuBloc*.

```
| fig |
fig := DrGeoFigure nouveau.
fig point: 5@2.
fig point: [ 5 auHasard @ 5 auHasard ].
```

`<WrpPoint> pointX:Y:` *v1 v2* [Méthode sur `DrGeoFigure`]

v1, un objet valeur

v2, une objet valeur

⇒ référence d'un point contraint par ses coordonnées

```
figure
   pointX: (figure valeurLibre: 2) cacher
   Y: (fiugre valeurLibre: 5) cacher.
```

`<WrpPoint> pointSurLigne:a:` *curve a* [Méthode sur `DrGeoFigure`]

curve, référence d'une ligne (droite, demi-droite, segment, etc.)

a, abscisse curviligne du point libre, la valeur est normalisée sur [0 ; 1].

⇒ référence d'un point libre sur une ligne

```
myPoint := figure pointSurLigne: s1 at: 0.5.
```

`<WrpPoint> milieuDe:et:` *p1 p2* [Méthode sur `DrGeoFigure`]

p1, référence d'un point ou d'un couple de coordonnées

p2, référence d'un point ou d'un couple de coordonnées

⇒ référence du milieu des deux points

```
| a i |
a := figure point: 1@1.
i := figure milieuDe: a et: 4@4.
```

`<WrpPoint> milieuDe:` *s* [Méthode sur `DrGeoFigure`]

s, référence d'un segment

⇒ référence du milieu du segment

```
figure milieuDe: s.
```

`<WrpPoint> intersectionDe:et:` *l1 l2* [Méthode sur `DrGeoFigure`]

l1, référence d'une ligne

l2, référence d'une ligne

⇒ référence du point d'intersection des deux lignes

```
figure intersectionDe: droite et: segment
```

`<WrpPoint> autreIntersectionDe:et:` *l1 l2* [Méthode sur `DrGeoFigure`]

l1, référence d'une ligne

l2, référence d'une ligne

⇒ référence de l'autre point d'intersection des deux lignes, lorsqu'il existe.

```
figure autreIntersectionDe: droite et: circle.
```

`<WrpPoint> point:parent:` *bloc item* [Méthode sur `DrGeoFigure`]

bloc, bloc de code retournant un point

item, référence d'un item géométrique

⇒ référence d'un point dont les coordonnées sont calculées avec le bloc de code ayant comme argument `item`.

```
| figure s mobile c block |
figure := DrGeoFigure nouveau.
s := figure segment: -5@0 to: 5@0.
mobile := figure pointSurLigne: s a: 0.1.
block := [:mathItem | |x|
   x := mathItem point x.
   x @ (x * x * x / 25 - x)].
c := figure point: block parent: mobile.
figure lieuDe: c lorsqueBouge: mobile.
```

`<WrpPoint> point:parents` *bloc liste* [Méthode sur `DrGeoFigure`]

bloc, bloc de code retournant un point

liste, une collection d'items géométriques

⇒ référence d'un point dont les coordonnées sont calculées avec le bloc de code ayant comme argument *liste*.

```
| figure d m p |
figure := DrGeoFigure nouveau.
d := figure droitePassantPar: -2 @ 1 et: 3 @ 3.
d couleur: Color blue.
m := figure point: 1 @ -1.
p := figure
   point: [:parents |
      parents first closestPointTo: parents second point]
   parents: {d . m}.
```

6.2.3 Droite

`<WrpCurve> droitePassantPar:et:` *p1 p2* [Méthode sur `DrGeoFigure`]

p1, référence d'un point ou d'un couple de coordonnées

p2, référence d'un point ou d'un couple de coordonnées

⇒ référence d'une droite passant par deux points

```
| p1 |
p1 := figure point: 0@0.
figure droitePassantPar: p1 et: 1@2.
```

`<WrpCurve> paralleleA:passantPar:` *d p* [Méthode sur `DrGeoFigure`]

d, référence d'une direction (droite, segment, vecteur,...)

p, référence d'un point ou d'un couple de coordonnées

⇒ référence d'une droite parallèle à la direction de *d* et passant par *p*

```
| a |
a := figure point: 1@5.
figure paralleleA: d passantPar: a.
```

`<WrpCurve> perpendiculaireA:passantPar:` *d p* [Méthode sur `DrGeoFigure`]

d, référence d'une direction (droite, segment, vecteur,...)

p, référence d'un point ou d'un couple de coordonnées

⇒ référence d'une droite perpendiculaire à la direction de *d* et passant par *p*

```
figure perpendiculaireA: d passantPar: 1@5.
```

`<WrpCurve> mediatriceDe:` *s* [Méthode sur `DrGeoFigure`]

s, référence d'un segment

⇒ référence de la médiatrice du segment *s*

```
c mediatriceDe: (c segmentDe: 0@0 a: 4@4)
```

`<WrpCurve> mediatriceDe:a:` *a b* [Méthode sur `DrGeoFigure`]

a, référence d'un point ou d'un couple de coordonnées

b, référence d'un point ou d'un couple de coordonnées

⇒ référence de la médiatrice du segment *ab*

```
figure mediatriceDe: 0@0 a: 4@4
```

`<WrpCurve> bissectriceDe:` *a* [Méthode sur `DrGeoFigure`]

a, référence d'un angle géométrique défini par **trois points**

⇒ référence de la bissectrice de l'angle *a*

```
figure bissectrice: angle
```

`<WrpCurve> bissectriceSommet:cote1:cote2` *a b c* [Méthode sur `DrGeoFigure`]

a,*b*,*c*, points définissant l'angle géométrique bac

⇒ référence de la bissectrice de l'angle bac

```
figure bissectriceSommet: 0@0 cote1: 1@0 cote2: 0@1
```

6.2.4 Demi-droite

`<WrpCurve> demiDroiteOrigine:passantPar:` *o p* [Méthode sur `DrGeoFigure`]

o, référence d'un point ou d'un couple de coordonnées, origine de la demi-droite

p, référence d'un point ou d'un couple de coordonnées, point de la demi-droite

⇒ référence d'une demi-droite définie par son origine et un point

```
| a |
a := figure point: 1@5.
figure demiDroiteOrigine: 0@0 passantPar: a.
```

6.2.5 Segment

`<WrpSegment> segmentDe:a:` *p1 p2* [Méthode sur `DrGeoFigure`]

p1, référence d'un point ou d'un couple de coordonnées

p2, référence d'un point ou d'un couple de coordonnées

⇒ référence d'un segment défini par ses extrémités

```
| a |
a := figure point: 5@5.
figure segmentDe: 10@10 a: a.
```

6.2.6 Cercle

`<WrpFilledCircle> cercleCentre:passantPar:` *c p* [Méthode sur `DrGeoFigure`]

c, référence d'un point ou d'un couple de coordonnées, centre du cercle

p, référence d'un point ou d'un couple de coordonnées, point du cercle

⇒ référence d'un cercle défini par son centre et un point

```
| a |
a := figure point: 1@5.
figure cercleCentre: a passantPar: 10@4.
```

`<WrpFilledCircle> cercleCentre:rayon:` *c r* [Méthode sur `DrGeoFigure`]

c, référence d'un point ou d'un couple de coordonnées, centre du cercle

r, référence d'un item numérique ou d'une valeur numérique, rayon du cercle

⇒ référence d'un cercle défini par son centre et son rayon

```
| a r |
a := figure point: 1@5.
r := figure valeurLibre: 4.
figure cercleCentre: a rayon: r.
figure cercleCentre: 4@4 rayon: 5
```

`<WrpFilledCircle> cercleCentre:segment:` *c s* [Méthode sur `DrGeoFigure`]

c, référence d'un point ou d'un couple de coordonnées, centre du cercle

r, référence d'un segment, rayon du cercle

⇒ référence d'un cercle défini par son centre et de rayon la longueur du segment

```
| a s |
a := figure point: 1@5.
s := figure segmentDe: 0@0 a: 3@0.
figure cercleCentre: a segment: s.
```

6.2.7 Arc de cercle

`<WrpFinitCurve> arcDe:a:passantPar:` *p1 p2 p3* [Méthode sur `DrGeoFigure`]

p1, référence d'un point ou d'un couple de coordonnées, 1ère extrémité de l'arc

p2, référence d'un point ou d'un couple de coordonnées, 2ème extrémité de l'arc

p3, référence d'un point ou d'un couple de coordonnées de l'arc

⇒ référence d'un arc de cercle défini par ses extrémités et un point

```
| a b |
a := figure point: 1@5.
b := figure point: 0@5.
figure arcDe: a a: -1 @ -2 passantPar: b.
```

`<WrpFinitCurve> arcCentre:de:a:` *O A B* [Méthode sur `DrGeoFigure`]

O, référence d'un point ou d'un couple de coordonnées, centre de l'arc

A, référence d'un point ou d'un couple de coordonnées, origine de l'arc

B, référence d'un point ou d'un couple de coordonnées, tel que l'angle de l'arc est AOB

⇒ référence d'un arc de cercle défini par son centre et l'angle AOB

```
| a b |
a := figure point: 1@5.
b := figure point: 0@5.
figure arcCentre: a de: b a: -1 @ -2.
```

6.2.8 Polygone

`<WrpFilledCurve> polygone:` *collection* [Méthode sur `DrGeoFigure`]

collection, une liste de références de points ou de couples de coordonnées ; sommets du polygone

⇒ référence d'un polygone défini par ses sommets

```
| b |
b := figure point: 1@3.
figure polygone: {1@2. b. 0@0. d}
```

`<WrpFilledCurve>` [Méthode sur `DrGeoFigure`]
`polygoneRegulierCentre:sommet:cotes:` *c s n*

c, référence d'un point ou d'un couple de coordonnées, centre du polygone

s, référence d'un point ou d'un couple de coordonnées, sommet du polygone

n, référence d'une valeur ou valeur, nombre de sommets du polygone

⇒ référence d'un polygone régulier défini par son centre, un de ses sommets, et son nombre de sommets

```
| b |
b := figure point: 1@3.
figure polygoneRegulierCentre: b sommet: 1@1 cotes: 7.
```

6.2.9 Les transformations géométriques

Les transformations géométriques permettent la construction des tranformés d'objets. Elles s'appliquent à des références d'objets de type point, segment, droite, demi-droite, vecteur, cercle, arc de cercle, polygone et lieu de point.

`<WrpCurve> rotationDe:parCentre:etAangle:` *i c a* [Méthode sur `DrGeoFigure`]

i, référence de l'objet à transformer (point, segment, droite, demi-droite, vecteur, cercle, arc-cercle, polygone)

c, référence d'un point ou d'un couple de coordonnées, centre de la rotation

a, référence d'un item valeur ou d'une valeur, angle de la rotation

⇒ référence de l'objet transformé

```
| c k l |
c := figure point: 5@5.
k := 3.1415.
l := figure droitePassantPar: 0@0 et: 5@5.
figure rotationDe: l parCentre: c etAngle: k.
figure rotationDe: l parCentre: 0@0 etAngle: Float pi / 3.
```

`<WrpCurve> homothetieDe:parCentre:etFacteur:` *i c k* [Méthode sur `DrGeoFigure`]

i, référence de l'objet à transformer (point, segment, droite, demi-droite, vecteur, cercle, arc-cercle, polygone)

c, référence d'un point ou d'un couple de coordonnées, centre de l'homothétie

k, référence d'un item valeur ou d'une valeur, facteur de l'homothétie

⇒ référence de l'objet transformé

```
| c k l |
c := figure point: 5@5.
k := -3.
l := figure droitePassantPar: 0@0 et: 5@5.
figure homothetieDe: l parCentre: c etFacteur: k.
figure homothetieDe: l parCentre: 0@0 etFacteur: 5.
```

`<WrpCurve> symetriqueDe:selonCentre` *i c* [Méthode sur `DrGeoFigure`]

i, référence de l'objet à transformer (point, segment, droite, demi-droite, vecteur, cercle, arc-cercle, polygone)

c, référence d'un point ou d'un couple de coordonnées, centre de la symétrie

⇒ référence de l'objet transformé

```
| a |
a := figure point: 4@2.
figure symetriqueDe: a selonCentre: 0@0
```

`<WrpCurve> symetriqueDe:selonAxe:` *i axe* [Méthode sur `DrGeoFigure`]

i, référence de l'objet à transformer (point, segment, droite, demi-droite, vecteur, cercle, arc-cercle, polygone)

axe, référence d'une droite, axe de la réflexion

⇒ référence de l'objet transformé

```
symetriqueDe: polygone selonAxe: d1
```

`<WrpCurve> translationDe:parVecteur:` *i v* [Méthode sur `DrGeoFigure`]

i, référence de l'objet à transformer (point, segment, droite, demi-droite, vecteur, cercle, arc-cercle, polygone)

v, référence d'un item vecteur ou d'un couple de coordonnées

⇒ référence de l'objet transformé

```
| u a|
u := figure vecteurOrigine: 1@1 extremite: 3@2.
a := figure translationDe: (figure point: 2@1) parVecteur: u
| u a|
a := figure translationDe: (figure point: 2@1) parVecteur: 2@1
```

6.2.10 Lieu d'un point

`<WrpCurve> lieuDe:lorsqueBouge:` *c m* [Méthode sur `DrGeoFigure`]

m, référence d'un point mobile sur une ligne

c, référence d'un point fixe dépendant du point *m*

⇒ référence d'un lieu

```
figure lieuDe: p lorsqueBouge: mobile
```

6.2.11 Vecteur

`<WrpCurve> vecteurOrigine:extremite:` *o e* [Méthode sur `DrGeoFigure`]

o, référence d'un point ou d'un couple de coordonnées, origine du vecteur

e, référence d'un point ou d'un couple de coordonnées, extrémité du vecteur

⇒ référence d'un vecteur

```
| b |
b := figure point: 0@5.
figure vecteurOrigine: b extremite: -1 @ -2
```

`<WrpCurve> vecteur:` *p* [Méthode sur `DrGeoFigure`]

p, référence d'un point ou d'un couple de coordonnées, coordonnées du vecteur

⇒ référence d'un vecteur

```
| p |
p := figure point: 5@5.
figure vecteur: p.
figure vecteur: -5 @ -5
```

6.2.12 Valeur numérique

`<Point> coordonnees` [Méthode sur `WrpPoint`]

⇒ coordonnées (statiques) d'un point

```
| c p |
p := pointSurLigne: segment a: 0.5.
c := p coordonnees.
c x
```

`<WrpValue> abscisseDe:` *item* [Méthode sur `DrGeoFigure`]

item, un point ou un vecteur

⇒ abscisse (dynamique) de *item*

```
| m x |
m := figure milieuDe: 10@5 et: 7@8.
x := figure abscisseDe: m
```

`<WrpValue> ordonneeDe:` *item* [Méthode sur `DrGeoFigure`]

item, un point ou un vecteur

⇒ ordonnée (dynamique) de *item*

```
| m x |
m := figure milieuDe: 10@5 et: 7@8.
x := figure ordonneesDe: m
```

`<WrpValue> valeurLibre:` *v* [Méthode sur `DrGeoFigure`]

v, la valeur initiale du nombre

⇒ référence d'un nombre libre

```
v := figure valeurLibre: (-1 arcCos)
```

`valeur:` *unNumbre* [Méthode sur `WrpValue`]

aNumber, un nombre

Modifie la valeur d'un objet nombre libre

```
v := figure valeurLibre: 3.
v valeur: Float pi
```

`<WrpValue> longueurDe:` *item* [Méthode sur `DrGeoFigure`]

item, référence d'un segment, cercle, arc ou vecteur

⇒ référence d'un nombre, longueur de l'item

```
figure longueurDe: v1
```

`<WrpValue> distanceDe:a:` *item point* [Méthode sur `DrGeoFigure`]

item, référence d'une droite ou d'un point

point, référence d'un point

⇒ référence d'un nombre, distance entre deux points ou un point et une droite

```
d := figure droitePassantPar: 0@0 et: 1@1.
figure distanceDe: d a: 12@2.
figure distanceDe: 0@0 a: 12@2
```

`<WrpValue> penteDe:` *droite* [Méthode sur `DrGeoFigure`]

droite, référence d'une droite

⇒ référence d'un nombre, pente de la droite

```
| p |
d := figure droitePassantPar: 0@0 et: 3@8.
p := figure pendteDe: d
```

6.2.13 Angle

`<WrpValue> angleCentre:de:a` *a b c* [Méthode sur `DrGeoFigure`]

a, référence d'un point, sommet de l'angle

b, référence d'un point

c, référence d'un point

⇒ référence d'un angle orienté bac dont la mesure appartient à [0 ; 360[.

```
figure angleCentre: 0@0 de: 3@1 a: -2@3
```

`<WrpValue> angleVecteur:et:` *v1 v2* [Méthode sur `DrGeoFigure`]

v1, référence d'un vecteur

v2, référence d'un vecteur

⇒ référence d'un angle orienté, formé par les deux vecteurs, dont la mesure appartient à [-180 ; 180[.

```
| v1 v2 a |
v1 := figure vecteurOrigine: a extremite: b.
v2 := figure vecteurOrigine: a extremite: c.
a := figure angleVecteur: v1 et: v2
```

`<WrpValue> angleGeometriqueCentre:de:a` *a b c* [Méthode sur `DrGeoFigure`]

a, référence d'un point, sommet de l'angle

b, référence d'un point

c, référence d'un point

⇒ référence d'un angle géométrique bac dont la mesure appartient à [0 ; 180[.

```
figure angleGeometriqueCentre: a de: b a: c
```

6.2.14 Équation

`<WrpValue> equationDe:` *item* [Méthode sur `DrGeoFigure`]

item, référence d'une droite ou d'un cercle

⇒ référence d'une équation de la droite ou du cercle

```
| e d |
d := figure droitePassantPar: 0@0 et: 15@13.
e := figure equationDe: e
```

6.2.15 Texte

`<WrpText> texte:` *string* [Méthode sur `DrGeoFigure`]

string, une chaîne de caractères

⇒ référence d'un objet texte positionné arbitrairement

```
figure texte: 'Hello'
```

`<WrpText> texte:a` *string point* [Méthode sur `DrGeoFigure`]

string, une chaîne de caractères

point, un couple de coordonnées

⇒ référence d'un objet texte positionné à aPoint

```
figure texte: 'Hello,
  je suis heureux !' a: 0@0
```

`texte:` *string* [Méthode sur `WrpText`]
string, une chaîne de caractères
Modifie le texte d'un objet texte

```
monTexte := figure texte: 'Hello'.
monTexte texte: 'Au revoir'
```

6.2.16 Style et attribut

Lecture des attributs

`<Point> coordonnees` [Méthode sur `WrpPoint`]
⇒ instance de `Point`, coordonnées du point

`<Number> x` [Méthode sur `WrpPoint`]
⇒ instance de `Number`, abscisse du point

`<Number> y` [Méthode sur `WrpPoint`]
⇒ instance de `Number`, ordonnée du point

```
a := figure point: 0@10.
figure assert: [a y = 10].
figure assert: [a coordonnees = (0@10)]
```

Modification des attributs

Pour modifier les attributs d'un objet déjà créé, nous lui envoyons le message approprié. La modification des attributs se fait donc toujours à posteriori.

`couleur:` *uneCouleur* [Méthode sur `WrpItem`]
uneCouleur, une instance de `Color`, voir ses méthodes de classe pour des définitions existantes : `Color black, Color red, Color blue, Color orange, Color yellow,...`
Modifie la couleur d'un item

```
pointA couleur: Color green
```

`couleurFond:` *uneCouleur* [Méthode sur `WrpText`]
unCouleur, une instance de `Color`
Modifie la couleur d'arrière plan d'un texte

```
monTexte couleurFond: Color green
```

`nommer:` *string* [Méthode sur `WrpItem`]
string, une chaîne de caractères
Renomme un item

```
segment nommer: '[AB]'
```

`textPositionDelta:` *vecteur* [Méthode sur `MathItemCostume`]
vecteur, une instance de `Point`
Modifie la position de l'étiquette d' un item relativement à sa position de référence.

```
pointA nommer: 'A'.
point costume textPositionDelta: -20 @ -20.
```

`cacher` [Méthode sur `WrpItem`]
Masque un item.

`montrer` [Méthode sur `WrpItem`]
Montre un item.

`fin` [Méthode sur `WrpCurve`]
Donne une épaisseur fine à une ligne (droite, demi-droite, cercle, lieu, etc.).

```
circle fin
```

`normal` [Méthode sur `WrpCurve`]
Donne une épaisseur normale à une ligne (droite, demi-droite, cercle, lieu, etc.). C'est l'épaisseur par défaut.

```
arc normal
```

`epais` [Méthode sur `WrpCurve`]
Donne une épaisseur large à une ligne (droite, demi-droite, cercle, lieu, etc.).

```
polygon epais
```

`plein` [Méthode sur `WrpCurve`]
Donne un style de trait continue, plein, à une ligne (droite, demi-droite, cercle, lieu, etc.).

```
polygon plein
```

`tiret` [Méthode sur `WrpCurve`]
Donne un style de trait en tirets à une ligne (droite, demi-droite, cercle, lieu, etc.).

```
polygon tiret
```

`pointille` [Méthode sur `WrpCurve`]
Donne un style de trait en pointillés à une ligne (droite, demi-droite, cercle, lieu, etc.).

```
arc pointille
```

`flecheDebut` [Méthode sur `wrpFinitCurve`]
Ajoute à un **arc** ou un **segment** une flèche en début de ligne.

```
| figure segment |
figure := DrGeoFigure nouveau.
segment := figure segmentDe: 0@0 a: 5@1.
segment flecheDebut
```

`flecheFin` [Méthode sur `wrpFinitCurve`]
Ajoute à un **arc** ou un **segment** une flèche en fin ligne.

```
segment flechefin
```

`fleches` [Méthode sur `wrpFinitCurve`]
Ajoute à un **arc** ou un **segment** des flèches en début et en fin de ligne.

```
| figure arc |
figure := DrGeoFigure nouveau.
arc := figure arcDe: 0@0 a: 5@3 passantPar: 2@1.
arc fleches
```

`marquerAvecCercle` [Méthode sur `wrpSegment`]
Marque – codage – un segment avec un cercle.

```
segment marquerAvecCercle
```

`marquerAvecDisque` [Méthode sur `wrpSegment`]
Marque – codage – un segment avec un Disque.

```
segment marquerAvecDisque
```

`marquerAvecSimpleTrait` [Méthode sur `wrpSegment`]
Marque – codage – un segment avec un trait.

```
segment marquerAvecSimpleTrait
```

`marquerAvecDoubleTrait` [Méthode sur `wrpSegment`]
Marque – codage – un segment avec un double trait.

```
segment marquerAvecDoubleTrait
```

`marquerAvecTripleTrait` [Méthode sur `wrpSegment`]
Marque – codage – un segment avec un triple trait.

```
segment marquerAvecTripleTrait
```

`marquerAucun` [Méthode sur `wrpSegment`]
Supprime toute marque d'un segment. Cette fonctionnalité sera rarement nécessaire car les segments nouvellement créés ne sont pas marqués.

```
segment marquerAucun
```

`croix` [Méthode sur `WrpPoint`]
Donne une forme en croix à un point.

```
a := figure point: 0@0.
a croix
```

`rond` [Méthode sur `WrpPoint`]
Donne une forme en rond à un point.

```
a rond
```

`carre` [Méthode sur `WrpPoint`]
Donne une forme carrée à un point.

```
a carre
```

`small` [Méthode sur `WrpPoint`]
Donne une petite taille à un point.

```
a small
```

`large` [Méthode sur `WrpPoint`]
Donne une taille large à un point.

```
a large
```

`bloquer` [Méthode sur `WrpItem`]
Bloque un item à sa position actuelle, pour peu que cela ait un sens.

```
| figure cercle |
figure := DrGeoFigure nouveau.
cercle := figure cercleCentre: 0@0 passantPar: 5@0.
figure := segmentDe: 0@0 a: (figure pointSurLigne: cercle a: 0.2).
(figure point: 0@0) bloquer
```

`debloquer` [Méthode sur `WrpItem`]
Débloque un item de sa position actuelle, pour peu que cela ait un sens. Cette fonctionnalité est rarement nécessaire car les items nouvellement créés sont débloqués par défaut.

```
| figure  |
(figure point: 0@0) debloquer
```

`deplacerA:` *point* [Méthode sur `WrpItem`]
point, couple de coordonnées
Déplace un point ou une valeur à la position donnée, pour peu que cela ait un sens.

```
| a |
a := figure point: 0@0.
a deplacerA: 5@5.
figure actualiser
```

6.2.17 Autres méthodes

La classe `DrGeoFigure` propose dans la catégorie `helpers` des méthodes supplémentaires pour faciliter la réalisation de figures interactives complexes.

`courbeDe:de:a:` *block x0 x1* [Méthode sur `DrGeoFigure`]

block, un bloc de code à un argument décrivant une fonction

x0, nombre, abscisse inférieure de la courbe

x1, nombre, abscisse supérieure de la courbe

Affiche la courbe représentative de la fonction décrite par le bloc de code de x0 à x1.

```
| figure |
figure courbeDe: [:x| x * x] de: -3 a: 3
```

`<BlockClosure> decimal:a:min:max` *f1 p f2 f3* [Méthode sur `DrGeoFigure`]

`<BlockClosure> decimal:a:min:max:nom:` *f1 p f2 f3 s* [Méthode sur `DrGeoFigure`]

`<BlockClosure>` [Méthode sur `DrGeoFigure`]
`decimal:a:min:max:nom:affciherValeur:` *v1 p v2 v3 s b*

`<BlockClosure> entier:a:min:max` *v1 p v2 v3* [Méthode sur `DrGeoFigure`]

`<BlockClosure> eniter:a:min:max:nom:` *v1 p v2 v3 s* [Méthode sur `DrGeoFigure`]

`<BlockClosure>` [Méthode sur `DrGeoFigure`]
`entier:a:min:max:nom:affciherValeur:` *v1 p v2 v3 s b*

v1, valeur initiale

p, position du bord gauche de la réglette

v2, valeur minimum

v3, valeur maximum

s, nom de la valeur

b, booléen (true|false), affiche ou non la valeur numérique avec le nom sous la forme 'a = 1.2'

⇒ un bloc de code retournant la valeur courante, décimale ou entière, de la réglette

Construis une réglette à la position indiquée avec une plage de valeur dans [f2 ; f3].

```
A := figure decimal: 1 a: -10@4 min: 0 max: 10 nom: 'A'.
F := figure entier: 3 a: -10@3 min: 0 max: 10 nom: 'F' afficherValeur: true.
A value + F value
```

Il existe d'autres variantes, dont certaines pour des nombres entiers.

`exporterVersImage:` *nomFichier* [Méthode sur `DrGeoFigure`]

nomFichier, une chaîne de caractère représentant le chemin et le nom du fichier où exporter la figure

Exporte la figure dans un fichier bitmap au format PNG.

```
| figure |
figure := DrGeoFigure minimal.
figure point: 0@0.
figure afficherAxes.
figure exporterVersImage: '/tmp/Toto.png'.
figure supprimer
```

6.3 Galerie d'exemples de figures Pharo

Pour illustrer l'utilisation des Figures Pharo Dr.Geo, nous vous proposons une petite série d'exemples. Ceux-ci vous montrent leurs importantes possibilités et nous espérons qu'ils seront également une source d'inspiration. Pour chacun de ces exemples, nous donnons le code source Pharo de la figure puis son résultat. Le code source doit être copié dans un espace de travail (...Clic arrière-plan → Outils → Espace de travail...) puis exécuté.

Animer une figure

Ces exemples s'appuient sur la gestion du temps et celle des processus programmés en Pharo.

Un premier exemple simple pour comprendre le principe :

```
| figure p pause |
figure:=DrGeoFigure nouveau.
p := figure point: 0@0.
pause := Delay forSeconds: 0.2.
figure faire: [
   100 foisRepete: [
      p deplacerA: (p coordonnees + (0.1@0)).
      figure actualiser.
      pause wait]]
```

Un deuxième exemple avec une figure plus élaborée :

```
| figure s r u pause |
figure := DrGeoFigure nouveau pleinEcran.
s := figure segmentDe: 0@ -1 a: 4@ -1.
r := figure pointSurLigne: s a: 0.8.
s := figure segmentDe: 0@0 a: 0@1.
u := figure pointSurLigne: s a: 0.7.
u rond petit; couleurr: Color blue.
1 a: 100 faire: [:n|
   u := figure
      point: [:parents| |y t|
         y := parents first y.
         t := parents second x.
         (n / 5) @ t * y * (1 - y)]
      parents:  {u . r}.
   u rond petit; couleur: Color blue].
pause := Delay forSeconds: 0.1.
figure faire: [
   0 a: 1 par: 0.05 faire: [:x |
      r mathItem setCurveAbscissa: x.
      figure actualiser.
      pause wait]]
```

Triangle de Sierpinski

Cet exemple s'appuie largement sur un bloc de code récursif.

```
| triangle c |
c := DrGeoFigure nouveau.
triangle := [].

triangle := [:s1 :s2 :s3 :n |
    c segmentDe: s1 a: s2;
```

```
        segmentDe: s2 a: s3;
        segmentDe: s3 a: s1.
    n >0 ifTrue:
        [triangle
            valeur: s1
            valeur: (c milieuDe: s1 et: s2) cacher
            valeur: (c milieuDe: s1 et: s3) cacher
            valeur: n-1.
        triangle
            valeur: (c milieuDe: s1 et: s2) cacher
            valeur: s2
            valeur: (c milieuDe: s2 et: s3) cacher
            valeur: n-1.
        triangle
            value: (c milieuDe: s1 et: s3) cacher
            value: (c milieuDe: s2 et: s3) cacher
            value: s3
            value: n-1.]].
triangle valeur: 0@3 valeur:  4@ -3 valeur: -4@ -3 valeur: 3.
(c point: 0@3) montrer
```

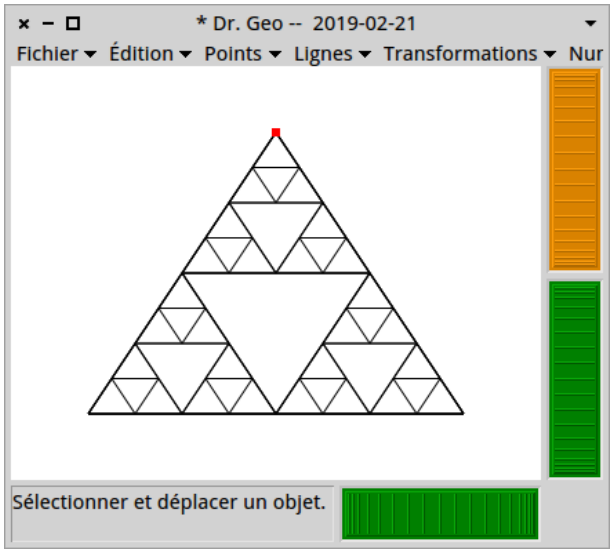


Figure 6.1: Triangle de Sierpinski

7 Outils du développeur

Du fait de son intégration dans l'environnement Pharo, Dr.Geo recèle quelques génies, nombres de ceux-ci sont cachés à la vue de l'utilisateur. Ce n'est pas tant le souhait d'en restreindre l'accès mais plutôt le souci d'alléger la charge cognitive avec une interface simple. Comme nous allons le voir dans les sections suivantes, ces génies s'invoquent par raccourcis clavier, commandes de menu ou codes Pharo.

Dans cette section nous présentons quelques outils à utiliser lors de l'écriture de scripts ou de figures programmées : l'espace de travail, le débogueur, l'inspecteur, etc.

7.1 Espace de travail

Exécuter le code d'une figure

Pour l'afficher ...Clic arrière-plan → Outils → Espace de travail... ou cliquer sur l'arrière-plan et faire *Alt-k*[1].

Un espace de travail – *Playground* – ressemble à s'y méprendre à un éditeur de texte. Mais c'est en fait une console d'édition de code Pharo : pour écrire, compiler et exécuter du code source, il est bien sûr possible d'y coller un code copié ailleurs. Après l'invocation d'un espace de travail[2], y coller le code source de la figure programmée ci-dessous[3] :

```
| figure fonction p integrale sommets |
fonction := [:x | x * x ].
sommets := OrderedCollection new.
figure := DrGeoSketch new.
p := figure point: -1 @ 0.
p hide.
sommets add: p.
-1 to: 1 by: 0.1 do: [:x |
   p := figure point: x @ (fonction value: x).
   sommets add: p hide].
p := figure point: 1 @ 0.
sommets add: p hide.
integrale := figure polygon: sommets.
integrale color: Color blue.
```

[1] Selon votre système d'exploitation, remplacer *Alt* par *Ctrl*.

[2] Contrairement aux génies, vous pouvez invoquer plusieurs espaces de travail.

[3] Pour coller un texte, essayer avec le raccourci clavier *Ctrl-v* ou depuis le menu contextuel de l'espace de travail (clic droit de souris).

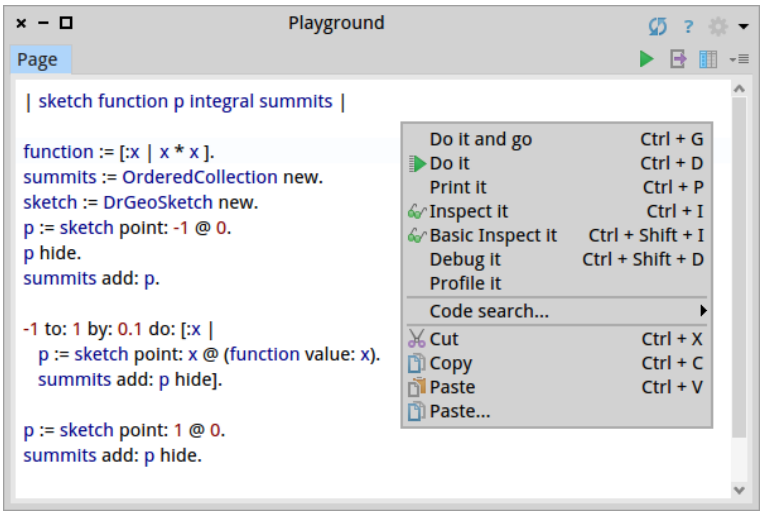


Figure 7.1: Votre espace de travail avec le code source collé et son menu contextuel

Pour compiler et exécuter ce code source, il suffit de le sélectionner à la souris et d'invoquer *Do it(d)* dans le menu contextuel. Ces deux opérations se font également au clavier par un `Ctrl-a` suivi d'un `Ctrl-d`. Vous obtenez alors immédiatement le résultat de cette figure programmée, sous la forme d'une figure interactive dans un canevas Dr.Geo.

Figure 7.2: Résultat de l'exécution du code source : intégrale de la fonction sur [-1 ; 1]

Exécuter et inspecter une figure

Depuis l'espace de travail, il existe une autre façon d'exécuter le code d'une figure et de compléter celle-ci pas à pas. Dans un espace de travail vide saisir ce code :

```
| figure |
figure := DrGeoSketch new.
figure point: 0@0.
figure
```

Cette fois-ci, pour l'exécuter utiliser le bouton vert de lecture, en haut à droite, ou son raccourci `Ctrl-Shift-g`. Cette commande s'appelle *Do it and go*, pour aller où ? Elle exécute le code de la figure et en plus ouvre un inspecteur sur le dernier objet du code, ici `figure`. Un inspecteur est un outil pour scruter les attributs d'un objet – ses variables – et naviguer dans ceux-ci ; l'inspecteur comprend un mini éditeur de code, en bas, pour exécuter des instructions sur l'objet inspecter.

Dans l'inspecteur à droite, en bas, il est alors possible de saisir et d'exécuter du code pour compléter la figure. Dans ce code, la figure est désignée par `self`, littéralement elle-même. Par exemple, pour afficher la grille, créer une droite et un point supplémentaire, saisir[4] :

```
"a DrGeoSketch"
self axesOn.
self line: 0@0 to: 1@0.
self point: 1@1
```

Ce code s'exécute par la combinaison de touches ***Ctrl-a*** puis ***Ctrl-d*** ou avec la souris comme expliqué précédemment.

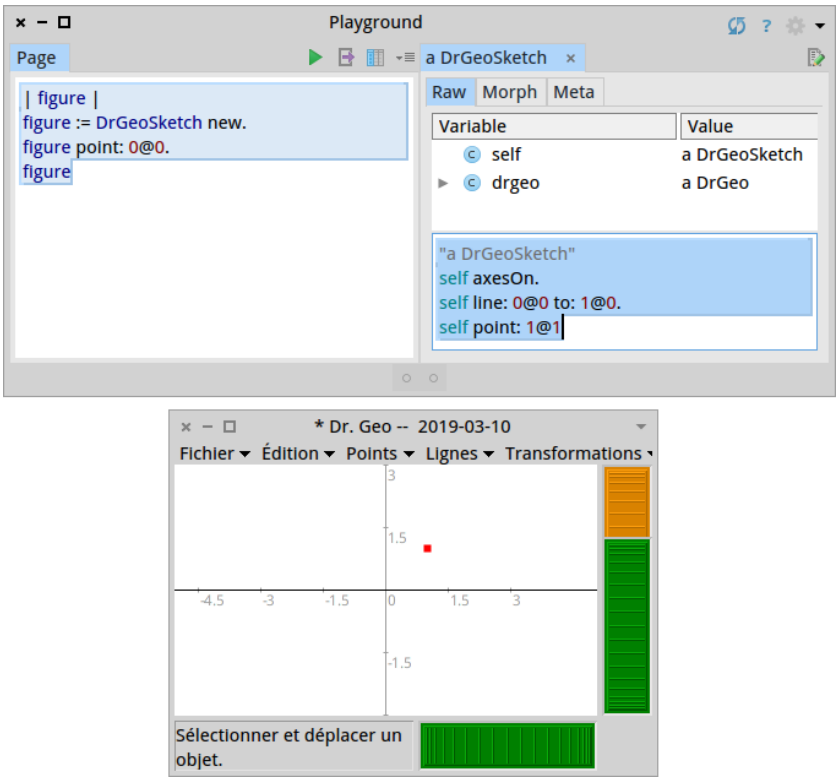


Figure 7.3: Exécution du code, inspection de l'objet `figure` et exécution d'instructions supplémentaires depuis l'inspecteur

Sauver et charger

L'espace de travail sauvegarde automatiquement le code du script édité. Il se recharge avec le bouton *Play pages* à droite de la fenêtre. Pour information, le code est sauvé dans un sous dossier de `DrGeo/Contents/Resources`.

Il est possible de nommer un script de l'espace de travail : double cliquer sur l'onglet *Page* à gauche en haut de l'espace de travail, éditer le nom du script puis valider par ***Enter***. Pour retrouver ultérieurement le script, écrire son nom dans l'outil *Spotter* : ***Shift-Enter*** pour invoquer *Spotter*, écrire le nom du script, il apparaît alors, le sélectionner et valider par ***Enter***.

Le code d'une figure se partage également sur Internet, de façon confidentielle. Sven Van Caekenberghe, un développeur de la communauté Pharo, propose un service de partage sur le *cloud*. Depuis l'espace de travail cliquer sur le bouton *Remote publish* à droite, une URL unique est alors

[4] `"a DrGeoSketch"` est un commentaire, inutile de le saisir ; il est dans l'exemple car l'inspecteur l'affiche dans son panneau en bas, cela permet de vous repérer.

copiée dans l'espace tampon du système d'exploitation hôte. Coller celle-ci dans tout document pour partager le code.

Pour charger dans Dr.Geo un code publié sur le *cloud*, le plus simple est d'utiliser l'outil de recherche *Spotter* : ***Shift-Enter*** puis ***Ctrl-v*** pour coller l'URL, puis ***Enter*** ouvrira un espace de travail avec ce code. L'essayer avec cette ressource `http://ws.stfx.eu/8I4GUN1B5W7K`.

7.2 Profileur

Lors de l'exécution d'un code source complexe, exécuter celui-ci par l'intermédiaire du *Profileur* aide à trouver les portions de code consommatrice en temps de calcul. Pour ce faire, dans le menu contextuel de l'espace de travail exécuter le code en invoquant *Profile it*. Le code source est exécuté, le canevas Dr.Geo affiché et en plus la fenêtre du profileur informe l'utilisateur sur le temps d'exécution du code et des méthodes invoquées. C'est un outil remarquable pour naviguer dans l'arbre d'exécution du code et afficher les méthodes consommatrice en temps machine.

Testé avec la code du début de section, le profileur montre que Dr.Geo prend plus de temps à construire la fenêtre que la figure :

```
86.3% {109ms} UndefinedObject>>DoIt
  71.0% {89ms} DrGeoSketch class(Behavior)>>new
  6.5% {8ms} DrGWrappedPoint(DrGWrappedItem)>>hide
  5.6% {7ms} DrGeoSketch>>point:
  1.6% {2ms} DrGeoSketch>>polygon:
  1.6% {2ms} primitives
```

En revanche sur une figure plus complexe, récursive par exemple, le profileur montre que le code consomme davantage de temps lors de la construction elle-même.

Copier-coller dans un espace de travail ce code qui construit la courbe de Sierpinski :

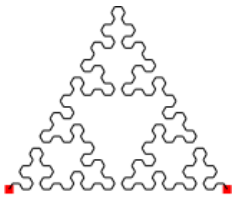


Figure 7.4: Courbe interactive de Sierpinski

```
| canvas dragon |
canvas := DrGeoSketch new.
dragon := [  ].
dragon := [ :a :b :k | | c m n |
   k > 0 ifTrue: [
      c := canvas
         altIntersectionOf: (canvas circleCenter: a to: b) hide
         and: (canvas circleCenter: b to: a) hide.
      c hide.
      m := (canvas middleOf: a and: c) hide.
      n := (canvas middleOf: b and: c) hide.
      dragon value: m value: a value: k - 1.
      dragon value: m value: n value: k - 1.
      dragon value: b value: n value: k - 1].
   k = 0 ifTrue: [ canvas segment: a to: b ].
].
canvas text: 'Sierpinski Dragon' at: 0@0.
dragon
   value: (canvas point: -4@1.5)
   value: (canvas point: 4@1.5)
   value: 5
```

Puis le sélectionner afin de l'exécuter avec le profileur. Davantage de temps est utilisé pour la construction des côtés de la courbe – méthode `segment:to:` – mais aussi d'autres éléments non visibles comme des intersections entre lignes.

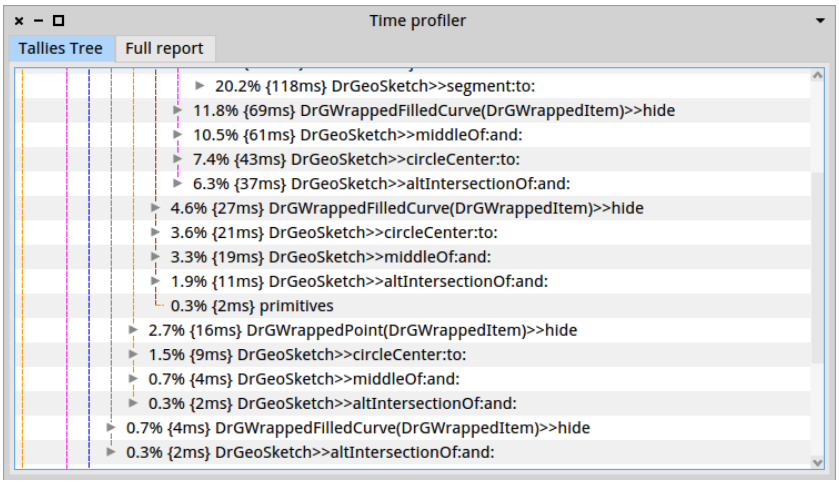


Figure 7.5: Le profileur Dr.Geo à l'issue de la construction de la courbe de Sierpinski

7.3 Débogueur

Dernier raffinement : l'exécution en mode pas à pas du code source. Cela se fait en l'exécutant avec le débogueur, dans le menu contextuel choisir la commande *Debug it*. Le débogueur est invoqué sur la première ligne, le code s'exécute pas à pas avec le bouton *Over*. Dans la partie basse à droite, les variables locales et leur contenu est consultable et modifiable.

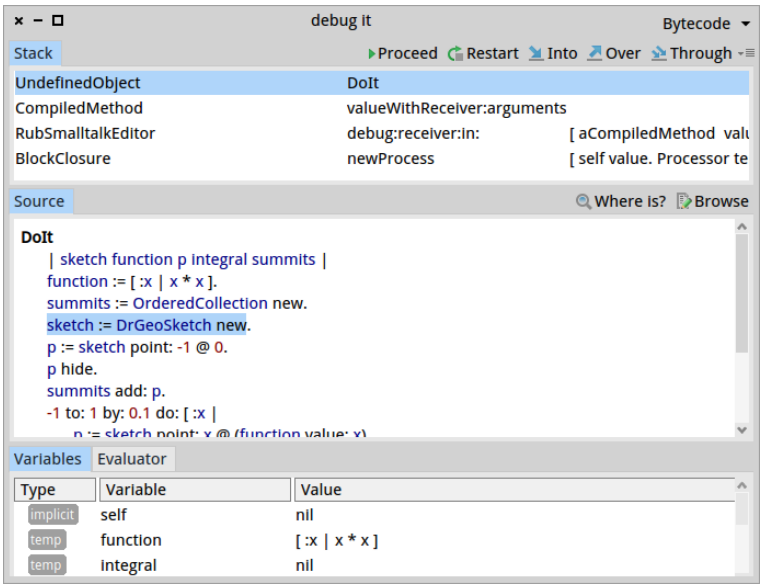


Figure 7.6: Le débogueur Dr.Geo

Quelques précisions concernant l'interface du débogueur s'imposent :

- **Stack.** Ce panneau indique la pile d'exécution des objets invoqués. Ce n'est pas utile lors de l'écriture du code source d'une figure depuis un espace de travail.
- **Proceed.** Ce bouton permet de reprendre l'exécution normale du code.

- **Restart.** Pour reprendre l'exécution pas à pas du programme depuis le début de la méthode, c'est à dire le début du code source de la figure.
- **Into.** Exécuter l'instruction mise en évidence, et suivre le message envoyé dans la méthode invoquée.
- **Over.** Exécuter l'instruction mise en évidence, et reprendre le contrôle après le message envoyé, c'est à dire à l'instruction suivante.
- **Through.** Exécuter l'instruction suivante, ne pas suivre le message envoyé mais exécuter pas à pas les éventuels blocs de code en arguments du message. Utile par suivre l'exécution dans un bloc de code.
- **Source.** Ce panneau contient le code source exécuté avec mise en évidence de l'instruction suivante à exécuter. Il offre également un menu contextuel avec quelques fonctions intéressantes comme "Run to here" pour exécuter le programme jusqu'à la ligne où aura été placé le curseur d'insertion textuelle.

Comme montré dans une section précédente, le débogueur permet l'exécution en mode pas à pas. Il s'invoque aussi à n'importe quel moment avec le raccourci clavier `Alt-.` (Alt + point).

En outre, le débogueur s'enclenche également par programmation, directement dans le code source en ajoutant une ligne `self halt.` Dans notre exemple précédent, nous pouvons modifier le code source comme suit :

```
...
p:=figure point: -1 @ 0.
p hide.
sommets add: p.
self halt.
-1 to: 1 by: 0.1 do: [ :x |
   p:=figure point: x @ (fonction value: x).
   sommets add: p hide].
...
```

Lorsque ce code est compilé et exécuté, le programme est stoppé. Dans la fenêtre qui s'affiche alors, il est demandé de faire un choix :

- **Proceed.** Reprendre l'exécution du programme à partir de l'instruction suivante
- **Abandon.** Stopper l'exécution du programme
- **Debug.** Invoquer le débogueur, une fenêtre identique à la figure précédente s'affiche alors pour exécuter pas à pas le programme et examiner les objets du programme.

7.4 Inspecteur

Avec l'inspecteur, l'utilisateur consulte les attributs d'une instance ou bien le contenu d'une variable.

Dans notre exemple, supposons que nous souhaitions voir le contenu de la collection *sommets*. Dans ce cas rien de plus simple, nous ajoutons une ligne de code où nous envoyons le message `inspect` à *sommets*, l'emplacement où se fait cette invocation n'est pas très important car nous n'avons ni point d'arrêt, ni exécution en mode pas à pas :

```
...
p:=figure point: -1 @ 0.
p hide.
sommets add: p.
sommets inspect.
-1 to: 1 by: 0.1 do: [ :x |
   p:=figure point: x @ (fonction value: x).
```

```
    sommets add: p hide].
...
```

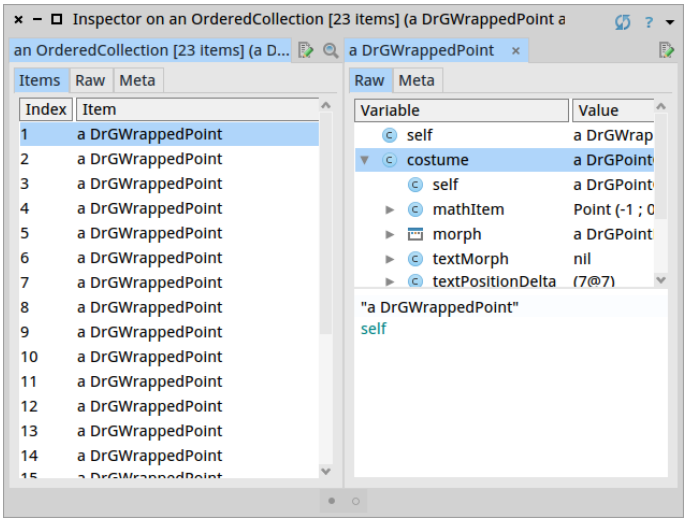


Figure 7.7: L'inspecteur sur la variable *sommets*

Un inspecteur est capable d'examiner autre chose que des attributs d'un objet, par exemple le contenu d'un dossier. Dr.Geo utilise cette fonctionnalité pour parcourir, exécuter, modifier les codes des figures du dossier `DrGeo/SmalltalkSketches`. Faire ...Clic arrière-plan → Outils → Travailler sur un script... un inspecteur s'affiche alors avec une liste de figures programmées. En choisissant une figure, son code source s'affiche dans un panneau à droite.

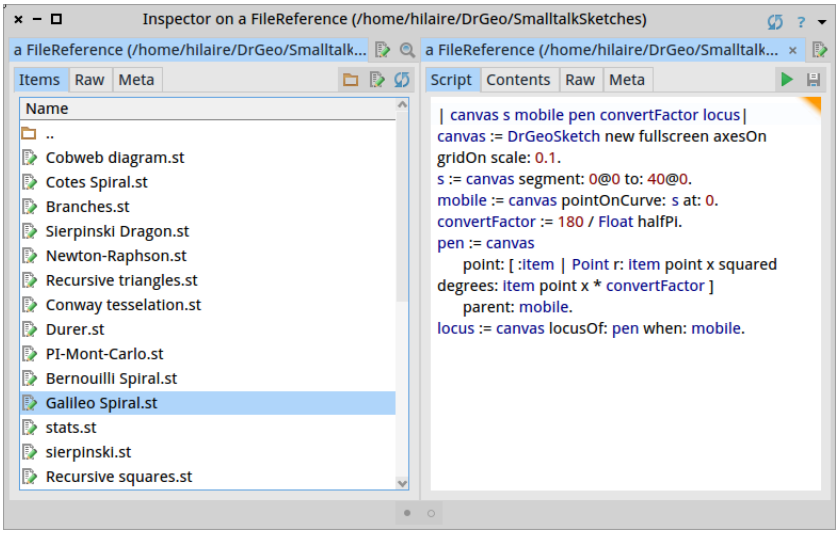


Figure 7.8: Inspecteur et codes source des figures

Choisir *la spirale de Galilée*, puis l'exécuter avec les raccourcis clavier ou les boutons, à l'identique de l'espace de travail.

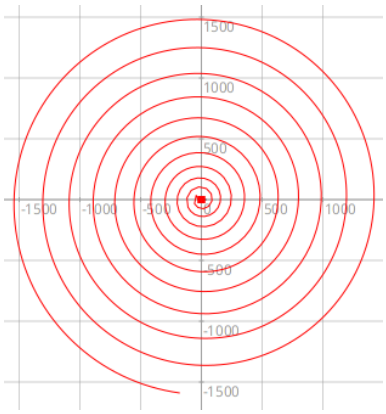


Figure 7.9: Spirale de Galilée

Lorsque le script est modifié dans le panneau à droite, cliquer sur le bouton “Save” à droite ou utiliser le raccourci clavier `Alt-s`. Pour créer un nouveau script, dans le panneau à gauche utiliser le bouton “Create File”. Par convention les codes sources Pharo de figure portent l’extension `.st`.

7.5 Chercheur

Le Chercheur est un outil de recherche de sélecteurs (nom d’une méthode), de classes, d’expressions dans le code source de Dr.Geo et de Pharo plus généralement. Il s’appuie sur les capacités réflexives de l’environnement Pharo. Pour ouvrir l’outil faire ...Clic arrière-plan → Outils → Chercheur de méthode...

La recherche se fait par un nom partiel ou encore plus intéressant par un exemple de calcul et de résultat souhaité. L’outil vous affiche alors l’ensemble des réponses correspondantes avec la possibilité de parcourir le code source de chacune d’elles in situ ; en effet il existe souvent plusieurs réponses valides, il convient de choisir celle souhaitée.

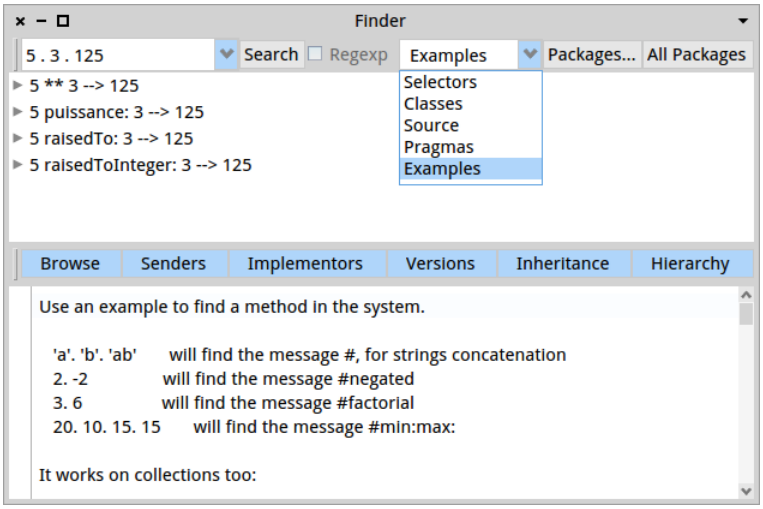


Figure 7.10: Réponse du Chercheur sur un motif de calcul et sa réponse souhaitée

Quelques exemples de recherches sur des éléments de natures différentes :

- **Sélecteur.** Saisir `^square` comme nom de sélecteur, cocher “Regexp” et choisir “Selectors” dans la liste. Ces réponses sont retournées (en cliquant sur l’une d’elle son code source est affiché) :

  `square`

```
squareNorm: (DrGLocus2ptsItem)
squared
squaredDistanceTo: (Point)
```

- **Classe.** Pour trouver les classes modélisant les demi-droites, saisir `DrGRay(.*)Item` comme nom de classe avec "Regex" coché et choisir "Class" dans la liste, le Chercheur retourne cette liste de classe :

```
DrGRay2Item
DrGRayHomothetyItem
DrGRayItem
DrGRayReflexionItem
DrGRayRotationItem
DrGRaySymmetryItem
DrGRayTranslationItem
```

- **Motif de calcul.** Pour trouver comment calculer la puissance 3 d'un nombre, saisir la séquence `5 . 3 . 125`. Elle indique qu'à partir de 5 et 3 nous souhaitons obtenir 125. Choisir "Examples" dans la liste, voici les possibilités trouvées :

```
5 ** 3 --> 125
5 puissance: 3 --> 125
5 raisedTo: 3 --> 125
5 raisedToInteger: 3 --> 125
```

Noter le sélecteur `puissance:` en *Français* propre à Dr.Geo.

Un autre exemple amusant sur l'addition des couleurs, saisir le motif `Color red . Color green . Color yellow`, le Chercheur retourne :

```
Color green + Color red --> Color yellow
Color red + Color green --> Color yellow
```

Il suffit d'utiliser le sélecteur `+` pour additionner des couleurs !

7.6 Spotter

Spotter est un outil de recherche de contenu dans l'environnement Dr.Geo. Il s'active par le raccourci clavier ***`Shift-Enter`*** et il suffit alors de saisir un mot de recherche. Il est capable de rechercher dans les menus, le code source, la documentation intégrée, voire des scripts de figure Pharo sur internet comme montré précédemment.

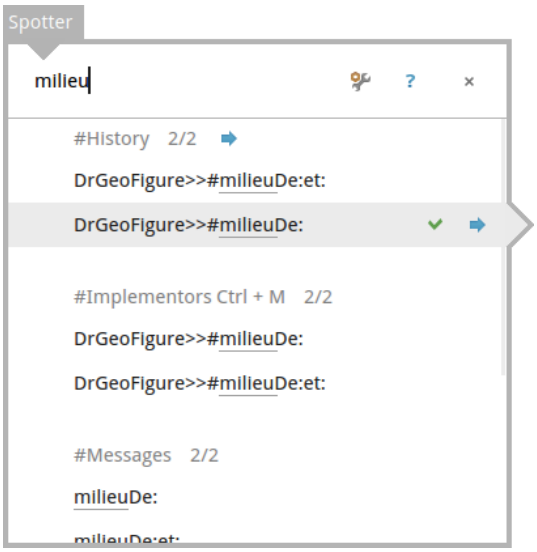


Figure 7.11: Spotter l'outil de recherche

Partie IV
Annexes

Annexe A GNU Free Documentation License

GNU Free Documentation License

Version 1.2, November 2002

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Annexe B Copyright des documents

Figure 4.26

Hannes Grobe 19:09, 3 September 2006 (UTC)
(https://commons.wikimedia.org/wiki/File:Rhodes_meander_hg.jpg), "Rhodes meander hg",
Editing, https://creativecommons.org/licenses/by-sa/2.5/legalcode

Figure 4.28

Theon (https://commons.wikimedia.org/wiki/File:Frise_exemple.svg),
https://creativecommons.org/licenses/by-sa/4.0/legalcode

Annexe C Liste des exercices

Exercice 2.1: Echelle 1 7
Exercice 2.2: Echelle 0 7
Exercice 2.3: Ma classe 8
Exercice 2.4: Variez les coordonnées 9
Exercice 2.5: Ordre d'envoi 9
Exercice 2.6: Ordre d'envoi 10
Exercice 2.7: Mon premier segment 10
Exercice 2.8: Ma première demi-droite 10
Exercice 2.9: Mon premier cercle 10
Exercice 2.10: Premier de cascade 11
Exercice 2.11: Un carré 11
Exercice 2.12: Un carré et un cercle 12
Exercice 2.13: Triangle et variable 13
Exercice 2.14: Segments liés par leur milieu 13
Exercice 2.15: Attributs d'un point 15
Exercice 2.16: Parallélogramme facile 16
Exercice 2.17: Intervalle de valeur 16
Exercice 2.18: Point farceur, les négatifs aussi ! 17
Exercice 2.19: Point farceur négatif à 0,5 près 17
Exercice 2.20: Pourquoi des parenthèses ?* 17
Exercice 2.21: Parenthèses, toutes nécessaires ?* 17
Exercice 2.22: Un autre point sur diagonale 18
Exercice 2.23: Point farceur, autre droite 19
Exercice 2.24: Point farceur, autre droite ? 19
Exercice 2.25: Je suis un point farceur 19
Exercice 2.26: Points sur axe des abscisses 19
Exercice 2.27: Sur l'axe des abscisses, les négatifs aussi 20
Exercice 2.28: Sur l'axe des ordonnées 20
Exercice 2.29: Sur la diagonale 20
Exercice 2.30: Des pas de lilliputiens 21
Exercice 2.31: Nuage de points 21
Exercice 2.32: Points nommés avec leur abscisse 21
Exercice 2.33: Points en vrac 22
Exercice 2.34: Conditions sur nombre 22
Exercice 2.35: Nombres premiers 23
Exercice 3.1: Nombres entiers relatifs 27
Exercice 3.2: D'autres erreurs de calculs avec des décimaux 28
Exercice 3.3: Vers l'infini et au delà 28
Exercice 3.4: Corriger les erreurs 28
Exercice 3.5: Fraction et écriture décimale 29
Exercice 3.6: Conversion en fraction 29
Exercice 3.7: Calculs arithmétiques 30
Exercice 3.8: Calculs fractionnaires 30
Exercice 3.9: Inverses de fractions 31
Exercice 3.10: Divisions euclidiennes 31
Exercice 3.11: Test de multiple 32
Exercice 3.12: Test de diviseur 32
Exercice 3.13: Programme interactive avec diviseur 33
Exercice 3.14: Diviseurs de 155 33
Exercice 3.15: Diviseurs communs de 100 et 155 34

Exercice 3.16: Programme interactif de diviseurs communs . . . 34
Exercice 3.17: PGDC . . . 35
Exercice 3.18: Programme interactif de PGDC . . . 35
Exercice 3.19: Nombre premier . . . 36
Exercice 3.20: Nombres premiers entre 1 et 1000 . . . 36
Exercice 3.21: Nombres premiers entre eux ? . . . 37
Exercice 4.1: Droites parallèles à la folie . . . 38
Exercice 4.2: Droites perpendiculaires à la folie . . . 39
Exercice 4.3: Pair, impair, premier . . . 40
Exercice 4.4: Distance entre deux droites parallèles . . . 42
Exercice 4.5: Déplacer la perpendiculaire . . . 42
Exercice 4.6: D'autres chemins . . . 42
Exercice 4.7: Toujours parallélogramme . . . 44
Exercice 4.8: Un autre parallélogramme . . . 44
Exercice 4.9: Deux points d'intersection . . . 45
Exercice 4.10: Parallélogramme et centre . . . 46
Exercice 4.11: Parallélogramme et symétrie . . . 46
Exercice 4.12: Côtés adjacents isométriques . . . 46
Exercice 4.13: Losange comme un parallélogramme . . . 47
Exercice 4.14: Losange et centre . . . 47
Exercice 4.15: Côtés adjacents perpendiculaires . . . 47
Exercice 4.16: Rectangle comme un parallélogramme . . . 48
Exercice 4.17: Rectangle et centre . . . 48
Exercice 4.18: Rectangle et cercle . . . 48
Exercice 4.19: Rectangle et diagonale . . . 48
Exercice 4.20: Côtés adjacents isométriques et perpendiculaires . . . 49
Exercice 4.21: Carré comme un parallélogramme . . . 49
Exercice 4.22: Carré et centre . . . 50
Exercice 4.23: Triangle isocèle, côtés isométriques . . . 51
Exercice 4.24: Triangle isocèle, axe de symétrie . . . 51
Exercice 4.25: Triangle isocèle et angle* . . . 52
Exercice 4.26: Triangle isocèle et angle variable . . . 52
Exercice 4.27: Triangle équilatéral, côtés isométriques . . . 52
Exercice 4.28: Triangle équilatéral, axes de symétrie . . . 53
Exercice 4.29: Triangle rectangle . . . 53
Exercice 4.30: Triangle rectangle isocèle . . . 53
Exercice 4.31: Triangle rectangle isocèle codé . . . 53
Exercice 4.32: Médiatrices du triangle . . . 54
Exercice 4.33: Cercle circonscrit d'un triangle . . . 54
Exercice 4.34: Bissectrices du triangle . . . 54
Exercice 4.35: Cercle inscrit d'un triangle* . . . 54
Exercice 4.36: Hauteurs du triangle . . . 55
Exercice 4.37: Médianes du triangle . . . 55
Exercice 4.38: Triangle et angles . . . 58
Exercice 4.39: Triangle et angles encore . . . 59
Exercice 4.40: Somme des angles d'un quadrilatère croisé . . . 61
Exercice 4.41: Image d'un carré par une symétrie centrale . . . 62
Exercice 4.42: Image d'un carré par une symétrie axiale . . . 62
Exercice 4.43: Image d'un carré par une translation . . . 63
Exercice 4.44: Image d'un carré par deux rotations . . . 64
Exercice 4.45: Image d'un carré par deux homothéties . . . 65
Exercice 4.46: Symétrique d'une collection plus complexe . . . 67

Exercice 4.47: Homothétie d'une collection 67
Exercice 4.48: Spirale 68
Exercice 4.49: Smiley 68
Exercice 4.50: Une première frise 69
Exercice 4.51: Motif pour translation 70
Exercice 4.52: Translations du motif 70
Exercice 4.53: Motif élémentaire 71
Exercice 4.54: Construction des vagues 71
Exercice 4.55: Motif de base d'une frise 71
Exercice 4.56: Translation de spirale 72
Exercice 4.57: Motif élémentaire 73
Exercice 4.58: Translation du motif complété 74
Exercice 4.59: Symétrie centrale puis translations 74
Exercice 4.60: Symétrie centrale, symétrie axiale puis translations 75
Exercice 5.1: Fonction linéaire de pente -2 76
Exercice 5.2: Observations fonction linéaire 77
Exercice 5.3: Fonction linéaire à pente un nombre décimal variable 77
Exercice 5.4: Fonction affine dynamique 78
Exercice 5.5: Observations fonction affine 78
Exercice 5.6: Fonction quadratique dynamique 78
Exercice 5.7: Observations fonction quadratique 78
Exercice 5.8: Fonction puissance n-ième dynamique 79
Exercice 5.9: Fonction homographique dynamique 79
Exercice 5.10: Observation fonction homographique 79

Annexe D Solutions des exercices

D.1 La syntaxe par l'exemple

Exercice 2.1

```
DrGeoFigure nouveau afficherAxes afficherGrille pleinEcran echelle: 1
```

Les graduations des axes sont de 50 en 50.

Exercice 2.2

error ZeroDivide, *erreur de division par zéro*. Il suffit de fermer les fenêtres.

Exercice 2.3

Un élève de la classe 932 écrira :

```
DrGeoFigure nouveau pleinEcran texte: 'Vive la 933 !!'
```

Exercice 2.4

```
DrGeoFigure nouveau afficherAxes point: 0 @ 0.
DrGeoFigure nouveau afficherAxes point: 1 @ 0.
DrGeoFigure nouveau afficherAxes point: 0 @ -1.
DrGeoFigure nouveau afficherAxes point: -1 @ -1.
```

Exercice 2.5

L'ordre d'envoi des messages :

1. `2 + 2` ⇒ 4
2. `nouveau` envoyé à `DrGeoFigure` ⇒ une figure
3. `pleinEcran` envoyé à la figure ⇒ la figure
4. `2 @ 4` ⇒ un objet coordonnées de point (2;4)
5. `point: 2 @ 4` envoyé à la figure

Exercice 2.6

L'ordre d'envoi des messages et les objets sont modifiés. En l'absence des parenthèses, les messages binaire `@` et `+` sont exécutés de la gauche vers la droite :

1. `nouveau` envoyé à `DrGeoFigure` ⇒ une figure
2. `pleinEcran` envoyé à la figure ⇒ la figure
3. `2 @ 2` ⇒ un objet coordonnées de point (2;2)
4. `2 @ 2 + 2` ⇒ un objet de coordonnées de point (4;4)
5. `point: 4 @ 4` envoyé à la figure

Exercice 2.7

```
DrGeoFigure nouveau segmentDe: 2 @ 1 a: 0 @ 0
```

Exercice 2.8

```
DrGeoFigure nouveau demiDroiteOrigine: -2 @ -1 passantPar: 0 @ 0
```

Exercice 2.9

```
DrGeoFigure nouveau cercleCentre: 0 @ 0 rayon: 3
```

Exercice 2.10

Si le message `afficherAxes` était précédé de ";" cela signifierait que le destinataire du message serait `DrGeoFigure`. Or celui-ci ne comprend pas ce message. Par ailleurs c'est à la nouvelle figure créée que nous demandons d'afficher ses axes, à savoir retourné par `DrGeoFigure nouveau`, donc pas de ";" pour envoyer le message à la nouvelle figure.

Exercice 2.11

```
DrGeoFigure nouveau pleinEcran;
   afficherAxes;
   afficherGrille;
   segmentDe: -2 @ 2 a: 2 @ 2;
   segmentDe: 2 @ 2 a: 2 @ -2;
   segmentDe: 2 @ -2 a: -2 @ -2;
   segmentDe: -2 @ -2 a: -2 @ 2
```

Exercice 2.12

```
DrGeoFigure nouveau pleinEcran;
   afficherAxes;
   afficherGrille;
   segmentDe: -2 @ 2 a: 2 @ 2;
   segmentDe: 2 @ 2 a: 2 @ -2;
   segmentDe: 2 @ -2 a: -2 @ -2;
   segmentDe: -2 @ -2 a: -2 @ 2;
   cercleCentre: 0 @ 0 rayon: 2
```

Exercice 2.13

```
| maFigure |
maFigure := DrGeoFigure nouveau.
maFigure afficherGrille.
maFigure segmentDe: 0 @ 0 a: 4 @ 0.
maFigure segmentDe: 4 @ 0 a: 1 @ 3.
maFigure segmentDe: 1 @ 3 a: 0 @ 0.
```

Exercice 2.14

```
| maFigure segment1 segment2 milieu1 milieu2|
maFigure := DrGeoFigure nouveau.
maFigure afficherGrille.
segment1 := maFigure segmentDe: 0 @ 0 a: 4 @ 4.
milieu1 := maFigure milieuDe: segment1.
segment2 := maFigure segmentDe: 1 @ 2 a: 5 @ 6.
milieu2 := maFigure milieuDe: segment2.
maFigure segmentDe: milieu1 a: milieu2
```

Exercice 2.15

```
| maFigure segment1 segment2 |
maFigure := DrGeoFigure nouveau.
maFigure afficherGrille.
segment1 := maFigure segmentDe: 0  0 a: 4  4.
segment1 couleur: Color pink;
   tiret;
   nommer: 'S1'.
```

```
segment2 := maFigure segmentDe: 2  3 a: 4  0.
segment2 couleur: Color orange;
   tiret;
   nommer: 'S2'.
(maFigure intersectionDe: segment1 et: segment2)
   nommer: 'I';
   large;
   croix.
```

Exercice 2.16

```
DrGeoFigure nouveau polygone: {0 @ 0 . 4 @ 0 . 5 @ 3 . 1 @ 3}
```

Exercice 2.17

1. `11 auHasard` donne une valeur entière au hasard entre 1 et 11 compris.
2. `11 auHasard - 6` donne donc une valeur entière comprise entre 1-6 et 11-6, à savoir entre -5 et 5.
3. Donc les valeurs possibles pour l'abscisse et l'ordonnée sont {-5 ; -4 ; -3 ; -2 ; -1 ; 0 ; 1 ; 2 ; 3 ; 4 ; 5}

Exercice 2.18

```
DrGeoFigure nouveau afficherAxes;
   point: [(11 auHasard - 6) @ (11 auHasard - 6)]
```

Exercice 2.19

```
DrGeoFigure nouveau afficherAxes;
   afficherGrille;
   echelle: 100;
   point: [(-8 auHasard / 2) @ (-8 auHasard / 2)]
```

Avec cette échelle de 100, la graduation des axes est à 0,5 près. Vous remarquez alors que le point farceur est toujours sur la grille.

Exercice 2.20

Il est nécessaire que l'abscisse – receveur à gauche du message `@` – et l'ordonnée – paramètre à droite du message `@` – soient calculées avant de constuire l'objet coordonnées, résultat de l'envoi du message `@`.

Exercice 2.21

Le receveur du message `@` est le résultat de `(5 auHasard / 10)` à sa gauche. Ce code comporte le message unaire `auHasard` qui est prioritaire sur le message `@` et le message binaire `/` qui est évalué avant le message `@` – ordre d'envoi des messages de la gauche vers la droite pour les messages de priorités identiques.

Les parenthèses ne sont donc pas nécessaires pour le receveur du message `@`.

Toutefois, les mettre facilite la compréhension du code par le lecteur humain.

Exercice 2.23

```
| figure |
figure := DrGeoFigure nouveau.
figure afficherAxes;
   afficherGrille.
figure point: [
```

```
    | coordonnee |
    coordonnee := 50 auHasard / 10.
    coordonnee @ (2 * coordonnee)].
figure droitePassantPar: 0@0 et: 1@2
```

Exercice 2.24

Le point farceur n'est plus sur une ligne droite. Il suit une ligne courbe qui s'appelle une parabole.

Exercice 2.25

Il faut introduire une variable `farceur` pour nommer ensuite le point.

```
| figure farceur |
figure := DrGeoFigure nouveau.
figure
   afficherAxes;
   afficherGrille.
farceur := figure point: [(50 auHasard / 10) @ (50 auHasard / 10)].
farceur nommer: 'Je suis un farceur'
```

Exercice 2.26

```
| figure |
figure := DrGeoFigure nouveau afficherAxes.
figure point: 1 @ 0.
figure point: 2 @ 0.
figure point: 3 @ 0.
figure point: 4 @ 0.
figure point: 5 @ 0.
figure point: 6 @ 0.
figure point: 7 @ 0.
figure point: 8 @ 0.
figure point: 9 @ 0.
figure point: 10 @ 0
```

Exercice 2.27

```
| figure |
figure := DrGeoFigure nouveau afficherAxes.
-10 a: -1 faire: [:abscisse |
   figure point: abscisse @ 0]
```

Exercice 2.28

```
| figure |
figure := DrGeoFigure nouveau afficherAxes.
1 a: 10 faire: [:ordonnee |
   figure point: 0 @ ordonnee]
```

Exercice 2.29

Le nom du paramètre du bloc est modifié pour plus de cohérence, car il représente à la fois une abscisse et une ordonnée.

```
| figure |
figure := DrGeoFigure nouveau afficherAxes.
1 a: 10 faire: [:coordonnee |
   figure point: coordonnee @ coordonnee]
```

Exercice 2.30

```
| figure |
figure := DrGeoFigure nouveau afficherAxes.
-5 a: 5 par: 0.2 faire: [:abscisse |
   figure point: abscisse @ 0]
```

Exercice 2.31

```
| figure |
figure := DrGeoFigure nouveau afficherAxes.
{-1 . 5.2 . -3.14 . 2.6} faire: [:ordonnee |
   figure point: 0 @ ordonnee]
```

Exercice 2.32

```
| figure |
figure := DrGeoFigure nouveau afficherAxes.
{-2 . 4 . 1/3 . 3.14 . -1/5} faire: [:abscisse |
   | point |
   point := figure point: abscisse @ 0.
   point nommer: abscisse]
```

Exercice 2.33

Observez les parenthèses autour de l'ordonnée du dernier point.

```
| figure |
figure := DrGeoFigure nouveau afficherAxes.
{1 @ 1 . -1 @ 1 . 3 @ -1 . 2/3 @ (-1/2)} faire: [:coordonnees |
   figure point: coordonnees]
```

Exercice 2.34

Pour le nombre 1, dans le code ci-dessous, placer le curseur clavier sur la ligne souhaitée et faire *Ctrl-P* au clavier pour afficher la condition retournée :

```
1 impair.
1 pair.
1 estPremier.
1 estEntier.
1 estDecimal.
1 positif.
1 strictementPositif.
```

Exercice 2.35

```
| figure |
figure := DrGeoFigure nouveau afficherAxes.
figure echelle: 5.
1 a: 100 faire: [:abscisse |
   abscisse estPremier siVrai: [
      | point |
      point := figure point: abscisse @ 0.
      point nommer: abscisse]
]
```

D.2 Nombres et opérations

Exercice 3.1

```
(-80 a: 50) commeCollectionOrdonnee
```

Exercice 3.2

```
5.2 + 0.9  - 6.1
⇒ 8.881784197001252e-16

5.2 + 0.7 + 0.11
⇒ 6.010000000000001

1.2 * 3  - 3.6
⇒ -4.440892098500626e-16
```

Exercice 3.3

Le système retourne une erreur `ZeroDivide`, division par zéro.

Exercice 3.4

```
(52/10) + (9/10)  - (61/10)
⇒ 0

(52/10) + (7/10)  + (11/100)
⇒ 601/100 soit 6.01

(12/10) * 3  - (36/10)
⇒ 0
```

Exercice 3.5

```
15.0/7
⇒ 2.142857142857143

(15/7) commeDecimal
⇒ 2.142857142857143

535/17.0
⇒ 31.470588235294116

(535/17) commeDecimal
⇒ 31.470588235294116
```

Exercice 3.6

```
1.2 commeFractionApprochee
⇒ (6/5)

17.3 commeFractionApprochee
⇒ (173/10)

0.00175 commeFractionApprochee
⇒ (7/4000)

9542.25 commeFractionApprochee
⇒ (38169/4)
```

Exercice 3.7

```
10 / 5 + 2
2 + (10 / 5)
10 + (7 * 2) + 4
(6 + 4) * 2
4 * 5 + (7 * 2)
```

Exercice 3.8

```
(2/9) + (3/9)
(5/7) - (2/7)
(2/3) * (4/5)
```

Exercice 3.9

```
1/(7/4) ⇒ (4/7)
1/(98/99) ⇒ (99/98)
```

Exercice 3.10

```
65 // 7 ⇒ 9
65 \\ 7 ⇒ 2
732 // 13 ⇒ 56
732 \\ 13 ⇒ 4
5241 // 29 ⇒ 180
5241 \\ 29 ⇒ 21
```

Exercice 3.11

Le programme retourne comme réponse *C'est un multiple.*

```
85200 \\ 24 = 0
   siVrai: ['C''est un multiple !']
   siFaux: ['Ce n''est pas un multiple.']
```

Exercice 3.12

Le programme retourne comme réponse *24 est un diviseur de 85200 !.*

```
85200 \\ 24 = 0
   siVrai: ['24 est un diviseur de 85200 !']
   siFaux: ['24 n''est pas un diviseir de 85200.']
```

Exercice 3.13

```
| a b reponse |
a := (UIManager default request: 'Un premier nombre') commeNombre.
b := (UIManager default request: 'Un deuxième nombre') commeNombre.
a \\ b = 0
   siVrai: [reponse := b asString, ' est un diviseur de ', a asString]
   siFaux: [reponse := b asString, ' n''est pas un diviseur de ', a asString].
UIManager default alert: reponse
```

Exercice 3.14

```
(1 a: 155) choisir: [ :n |  155 \\ n = 0]
```

Exercice 3.15

Observer l'utilisation des parenthèses () autour des deux appels du bloc de code `diviseurs` avec 100 et 155 comme arguments. C'est pour des raisons de priorités.

```
| diviseurs |
diviseurs := [:nombre |
   (1 a: nombre) choisir: [ :n |  nombre \\ n = 0]].
(diviseurs valeur: 100) & (diviseurs valeur: 155)
```

Exercice 3.16

```
| a b commun diviseurs |
diviseurs := [:nombre |
   (1 a: nombre) choisir: [ :n |  nombre \\ n = 0]].
""
a := (UIManager default request: 'Un premier nombre naturel') commeNombre.
b := (UIManager default request: 'Un deuxième nombre naturel') commeNombre.
commun := (diviseurs valeur: a) & (diviseurs valeur: b).
UIManager default alert: 'Les diviseurs communs de ',
   a asString, ' et ', b asString,
   ' sont : ', commun asString
```

Exercice 3.17

```
| diviseurs pgdc |
diviseurs := [:nombre |
   (1 a: nombre) choisir: [ :n |  nombre \\ n = 0]].
pgdc := [:a :b | ((diviseurs valeur: a) & (diviseurs valeur: b)) max].
""
pgdc valeur: 100 valeur: 155
```

Exercice 3.18

```
| a b diviseurs pgdc |
diviseurs := [:nombre |
   (1 a: nombre) choisir: [ :n |  nombre \\ n = 0]].
pgdc := [:x :y | ((diviseurs valeur: x) & (diviseurs valeur: y)) max].
""
a := (UIManager default request: 'Un premier nombre naturel') commeNombre.
b := (UIManager default request: 'Un deuxième nombre naturel') commeNombre.
UIManager default alert: 'Le PGDC de ',
   a asString, ' et ', b asString,
   ' est : ', (pgdc valeur: a valeur: b) asString
```

Exercice 3.19

```
| diviseurs premier |
diviseurs := [:nombre |
   (1 a: nombre) choisir: [ :n |  nombre \\ n = 0]].
premier := [: n | (diviseurs valeur: n) taille = 2].
""
premier valeur: 155
```

Exercice 3.20

```
| diviseurs premier |
diviseurs := [:nombre |
   (1 a: nombre) choisir: [ :n |  nombre \\ n = 0]].
premier := [: n | (diviseurs valeur: n) taille = 2].
""
(1 a: 1000) choisir: [:n | premier valeur: n]
```

Exercice 3.21

```
| a b diviseurs pgdc reponse |
diviseurs := [:nombre |
   (1 a: nombre) choisir: [ :n |  nombre \\ n = 0]].
pgdc := [:x :y | ((diviseurs valeur: x) & (diviseurs valeur: y)) max].
""
a := (UIManager default request: 'Un premier nombre naturel') commeNombre.
b := (UIManager default request: 'Un deuxième nombre naturel') commeNombre.
(pgdc valeur: a valeur: b) = 1
   siVrai:[reponse := a asString, ' et ', b asString, ' sont premiers entre eux.']
   siFaux: [reponse := a asString, ' et ', b asString, ' ne sont pas premiers entre eux.'].
UIManager default alert: reponse
```

D.3 Espace

Exercice 4.1

```
| figure d1|
figure := DrGeoFigure nouveau.
d1 := figure droitePassantPar: 0 @ 5 et: 2 @ 0.
d1 nommer: 'd1'.
(figure point: 0 @ 5) montrer.
3 a: 12 par: 0.5 faire: [:abscisse |
   figure paralleleA: d1 passantPar: abscisse @ 0]
```

Exercice 4.2

```
| figure d1|
figure := DrGeoFigure nouveau.
d1 := figure droitePassantPar: 0 @ 5 et: 2 @ 0.
d1 nommer: 'd1'.
(figure point: 0 @ 5) montrer.
3 a: 12 par: 0.5 faire: [:abscisse |
   figure perpendiculaireA: d1 passantPar: abscisse @ 0]
```

Exercice 4.3

```
| figure d1 droite |
figure := DrGeoFigure nouveau echelle: 3.
d1 := figure droitePassantPar: 0 @ 5 et: 2 @ 0.
d1 nommer: 'd1'.
(figure point: 0 @ 5) montrer.
1 a: 500 faire: [:abscisse |
   | couleur |
   droite := figure perpendiculaireA: d1 passantPar: abscisse @ 0.
   abscisse pair
      siVrai: [couleur := Color red]
      siFaux: [couleur := Color orange].
   abscisse estPremier siVrai: [couleur := Color blue].
   droite couleur: couleur]
```

Exercice 4.4

```
| figure droite1 droite2 perp pointA pointB |
figure := DrGeoFigure nouveau afficherAxes.
```

```
droite1 := figure droitePassantPar: 5 @ 5 et: 7 @ -2.
droite2 := figure paralleleA: droite1 passantPar: 0 @ 0.
perp := figure perpendiculaireA: droite2 passantPar: -5 @ 0.
droite1 nommer: 'droite 1'.
droite2 nommer: 'droite 2'.
pointA := figure intersectionDe: droite1 et: perp.
pointB := figure intersectionDe: droite2 et: perp.
(figure distanceDe: pointA a: pointB) montrer
```

Exercice 4.5

La distance entre les deux droites parallèles est inchangée.

Exercice 4.6

```
| figure droite1 droite2 perp ptA ptB |
figure := DrGeoFigure nouveau afficherAxes.
droite1 := figure droitePassantPar: 5 @ 5 et: 7 @ -2.
droite2 := figure paralleleA: droite1 passantPar: 0 @ 0.
perp := figure perpendiculaireA: droite2 passantPar: -5 @ 0.
perp epais.
droite1 nommer: 'droite 1'.
droite2 nommer: 'droite 2'.
ptA := figure intersectionDe: droite1 et: perp.
ptB := figure intersectionDe: droite2 et: perp.
(figure distanceDe: ptA a: ptB) montrer.
0 a: 1 par: 0.01 faire: [:valeur |
   | point |
   point := figure pointSurLigne: droite1 a: valeur.
   point cacher.
   (figure segmentDe: point a: ptB) pointille]
```

Exercice 4.7

Lorsque les points A, B ou C sont déplacés, le point D se déplace *automatiquement* afin que ABCD reste un parallélogramme. Cela vient du fait que le point D a été construit à partir de la propriété des côtés opposés parallèles du parallélogramme.

Exercice 4.8

```
| figure o m n p mn pm |
figure := DrGeoFigure nouveau.
m := (figure point: -5 @ 2) nommer: 'M'.
n := (figure point: 3 @ 2) nommer: 'N'.
p := (figure point: 1 @ -5) nommer: 'P'.
mn := figure segmentDe: m a: n.
pm := figure segmentDe: p a: m.
o := figure
   intersectionDe: (figure paralleleA: pm passantPar: n) cacher
   et: (figure paralleleA: mn passantPar: p) cacher.
o nommer: 'O'.
figure segmentDe: o a: n.
figure segmentDe: o a: p
```

Exercice 4.10

```
| figure a b c i |
```

```
figure := DrGeoFigure nouveau.
a := (figure point: -5 @ 2) nommer: 'A'.
b := (figure point: 3 @ 2) nommer: 'B'.
c := (figure point: 1 @ -5) nommer: 'C'.
i := (figure milieuDe: a et: c) nommer: 'I'
```

Exercice 4.11

```
| figure a b c i d |
figure := DrGeoFigure nouveau.
a := (figure point: -5 @ 2) nommer: 'A'.
b := (figure point: 3 @ 2) nommer: 'B'.
c := (figure point: 1 @ -5) nommer: 'C'.
i := (figure milieuDe: a et: c) nommer: 'I'.
d := (figure symetriqueDe: b selonCentre: i) nommer: 'D'.
figure polygone: {a . b . c . d}
```

Exercice 4.9

Le deuxième point d'intersection des deux cercles permet de former un quadrilatère dont les côtés opposés sont parallèles. Toutefois les côtés opposés ne sont alors pas parallèles. Le quadrilatère est dit croisé, les côtés opposés sont sécants. Ce n'est donc pas un parallélogramme.

Exercice 4.12

```
| figure a b c cercle |
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
b := (figure point: 5@1) nommer: 'B'.
cercle := figure cercleCentre: b passantPar: a.
c := figure pointSurLigne: cercle a: 0.2.
c nommer: 'C'
```

Exercice 4.13

```
| figure a b c d ab bc cercle |
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
b := (figure point: 5@1) nommer: 'B'.
cercle := figure cercleCentre: b passantPar: a.
c := figure pointSurLigne: cercle a: 0.2.
c nommer: 'C'.
ab := figure segmentDe: a a: b.
bc := figure segmentDe: b a: c.
d := figure
  intersectionDe: (figure paralleleA: ab passantPar: c) cacher
  et: (figure paralleleA: bc passantPar: a) cacher.
d nommer: 'D'.
figure segmentDe: a a: d.
figure segmentDe: c a: d
```

Exercice 4.14

```
| figure a b c d i cercle |
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
```

```
b := (figure point: 5@1) nommer: 'B'.
cercle := figure cercleCentre: b passantPar: a.
c := figure pointSurLigne: cercle a: 0.2.
c nommer: 'C'.
i := (figure milieuDe: a et: c) nommer: 'I'.
d := (figure symetriqueDe: b selonCentre: i) nommer: 'D'.
figure polygone: {a . b . c . d}
```

Exercice 4.15

```
| figure a b c ab droite |
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
b := (figure point: 5@1) nommer: 'B'.
ab := figure droitePassantPar: a et: b.
droite := figure perpendiculaireA:  ab passantPar: b.
c := figure pointSurLigne: droite a: 0.1.
c nommer: 'C'
```

Exercice 4.16

```
| figure a b c d ab bc droite |
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
b := (figure point: 5@1) nommer: 'B'.
ab := figure droitePassantPar: a et: b.
droite := figure perpendiculaireA:  ab passantPar: b.
c := figure pointSurLigne: droite a: 0.1.
c nommer: 'C'.
bc := figure segmentDe: b a: c.
d := figure
  intersectionDe: (figure paralleleA: ab passantPar: c) cacher
  et: (figure paralleleA: bc passantPar: a) cacher.
d nommer: 'D'.
figure segmentDe: a a: d.
figure segmentDe: c a: d
```

Exercice 4.17

```
| figure a b c d i ab droite|
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
b := (figure point: 5@1) nommer: 'B'.
ab := figure droitePassantPar: a et: b.
droite := figure perpendiculaireA:  ab passantPar: b.
c := figure pointSurLigne: droite a: 0.1.
c nommer: 'C'.
i := (figure milieuDe: a et: c) nommer: 'I'.
d := (figure symetriqueDe: b selonCentre: i) nommer: 'D'.
figure polygone: {a . b . c . d}
```

Exercice 4.18

```
| figure a c i cercle |
figure := DrGeoFigure new.
```

```
a := figure point: 0 @ 0.
c := figure point: 5 @ 2.
figure segmentDe: a a: c.
i := figure milieuDe: a et: c.
cercle := figure cercleCentre: i passantPar: a.
```

Exercice 4.19

```
| figure a c i cercle ib b d|
figure := DrGeoFigure new.
a := figure point: 0 @ 0.
c := figure point: 5 @ 2.
figure segmentDe: a a: c.
i := figure milieuDe: a et: c.
cercle := figure cercleCentre: i passantPar: a.
b := figure pointSurLigne: cercle a: 0.4.
ib := figure droitePassantPar: i et: b.
d := figure intersectionDe: ib et: cercle.
figure polygone: { a . b . c . d  }
```

Exercice 4.20

```
| figure a b c ab droite cercle |
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
b := (figure point: 5@1) nommer: 'B'.
ab := figure droitePassantPar: a et: b.
droite := figure perpendiculaireA:  ab passantPar: b.
cercle := figure cercleCentre: b passantPar: a.
c := figure intersectionDe: droite et: cercle.
c nommer: 'C'
```

Exercice 4.21

```
| figure a b c ab bc droite cercle |
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
b := (figure point: 5@1) nommer: 'B'.
ab := figure droitePassantPar: a et: b.
droite := figure perpendiculaireA:  ab passantPar: b.
cercle := figure cercleCentre: b passantPar: a.
c := figure intersectionDe: droite et: cercle.
c nommer: 'C'.
bc := figure segmentDe: b a: c.
d := figure
  intersectionDe: (figure paralleleA: ab passantPar: c) cacher
  et: (figure paralleleA: bc passantPar: a) cacher.
d nommer: 'D'.
figure segmentDe: a a: d.
figure segmentDe: c a: d.
figure segmentDe: a a: b.
ab cacher.
cercle cacher.
droite cacher.
```

Exercice 4.22

```
| figure a b c d i ab droite cercle |
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
b := (figure point: 5@1) nommer: 'B'.
ab := figure droitePassantPar: a et: b.
droite := figure perpendiculaireA:  ab passantPar: b.
cercle := figure cercleCentre: b passantPar: a.
c := figure intersectionDe: droite et: cercle.
c nommer: 'C'.
i := (figure milieuDe: a et: c) nommer: 'I'.
d := (figure symetriqueDe: b selonCentre: i) nommer: 'D'.
figure polygone: {a . b . c . d}
```

Exercice 4.23

```
| figure a b c cercle1 cercle2 |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
cercle1 := figure cercleCentre: b rayon: 4.
cercle2 := figure cercleCentre: c rayon: 4.
a := (figure intersectionDe: cercle1 et: cercle2) nommer: 'A'.
cercle1 cacher.
cercle2 cacher.
figure polygone: {a . b . c}
```

Exercice 4.24

```
| figure a b c bc mediatrice |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
bc := figure segmentDe: b a: c.
mediatrice := figure mediatrice: bc.
a := (figure pointSurLigne: mediatrice a: 0.2) nommer: 'A'.
figure polygone: {a . b . c}
```

Exercice 4.25

La difficulté pour terminer l'exercice vient de l'angle nécessaire pour la 2e rotation, ce n'est pas le même. C'est le complémentaire à 360 degrés du premier angle. Le plus simple étant définir un deuxième angle *alpha2* en inversant les deux extrémités du premier angle.

```
| figure alpha1 alpha2 b c b1 c1 demiDroite1 demiDroite2|
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
alpha1 := figure angleCentre: 10@10 de: 12@10 a: 12@13.
b1 := figure rotationDe: b parCentre: c etAngle: alpha1.
demiDroite1 := figure demiDroiteOrigine: c passantPar: b1.
alpha2 := figure angleCentre: 10@10 de: 12@13 a: 12@10.
c1 := figure rotationDe: c parCentre: b etAngle: alpha2.
demiDroite2 := figure demiDroiteOrigine: b passantPar: c1.
a := (figure intersectionDe: demiDroite1 et: demiDroite2) nommer: 'A'.
```

```
figure polygone: {a . b . c}
```

Exercice 4.26

```
| figure alpha1 alpha2 b c b1 c1 demiDroite1 demiDroite2|
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
alpha1 := figure angleCentre: 10@10 de: 12@10 a: 12@13.
b1 := figure rotationDe: b parCentre: c etAngle: alpha1.
demiDroite1 := figure demiDroiteOrigine: c passantPar: b1.
alpha2 := figure angleCentre: 10@10 de: 12@13 a: 12@10.
c1 := figure rotationDe: c parCentre: b etAngle: alpha2.
demiDroite2 := figure demiDroiteOrigine: b passantPar: c1.
a := (figure intersectionDe: demiDroite1 et: demiDroite2) nommer: 'A'.
b1 cacher.
c1 cacher.
demiDroite1 cacher.
demiDroite2 cacher.
(figure point: 12@10) montrer.
figure polygone: {a . b . c}
```

Exercice 4.27

```
| figure a b c cercle1 cercle2 |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
cercle1 := figure cercleCentre: b passantPar: c.
cercle2 := figure cercleCentre: c passantPar: b.
a := figure intersectionDe: cercle1 et: cercle2.
a nommer: 'A'.
cercle1 cacher.
cercle2 cacher.
figure polygone: {a . b . c}
```

Exercice 4.28

```
| figure a b c cercle mediatrice |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
cercle := figure cercleCentre: b passantPar: c.
mediatrice := figure mediatriceDe: b a: c.
a := figure intersectionDe: cercle et: mediatrice.
a nommer: 'A'.
cercle cacher.
mediatrice cacher.
figure polygone: {a . b . c}
```

Exercice 4.29

```
| figure a b c bc perp |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
```

```
c := (figure point: 0@0) nommer: 'C'.
bc := figure segmentDe: b a: c.
perp := figure perpendiculaireA: bc passantPar: b.
a := (figure pointSurLigne: perp a: 0.1) nommer: 'A'.
figure polygone: {a . b . c}
```

Exercice 4.30

```
| figure a b c bc perp cercle |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
bc := figure segmentDe: b a: c.
perp := figure perpendiculaireA: bc passantPar: b.
cercle := figure cercleCentre: b passantPar: c.
a := (figure intersectionDe: perp et: cercle) nommer: 'A'.
figure polygone: {a . b . c}
```

Exercice 4.31

```
| figure a b c bc ba perp cercle |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
bc := figure segmentDe: b a: c.
perp := figure perpendiculaireA: bc passantPar: b.
cercle := figure cercleCentre: b passantPar: c.
a := (figure intersectionDe: perp et: cercle) nommer: 'A'.
figure polygone: {a . b . c}.
perp cacher.
cercle cacher.
figure angleGeometriqueCentre: b de: a a: c.
bc marquerAvecSimpleTrait.
ba := figure segmentDe: b a: a.
ba marquerAvecSimpleTrait.
```

Exercice 4.32

```
| figure a b c m1 m2 m3 m |
figure := DrGeoFigure nouveau.
a := (figure point: 2@1) nommer: 'A'.
b := (figure point: 7@2) nommer: 'B'.
c := (figure point: 4@7) nommer: 'C'.
figure polygone: {a . b . c}.
m1 := figure mediatriceDe: a a: b.
m2 := figure mediatriceDe: b a: c.
m3 := figure mediatriceDe: a a: c.
m := (figure intersectionDe: m1 et: m2) nommer: 'M'
```

Exercice 4.33

```
| figure a b c m1 m2 m3 m |
figure := DrGeoFigure nouveau.
a := (figure point: 2@1) nommer: 'A'.
b := (figure point: 7@2) nommer: 'B'.
```

```
c := (figure point: 4@7) nommer: 'C'.
figure polygone: {a . b . c}.
m1 := figure mediatriceDe: a a: b.
m2 := figure mediatriceDe: b a: c.
m3 := figure mediatriceDe: a a: c.
m := (figure intersectionDe: m1 et: m2) nommer: 'M'.
figure cercleCentre: m passantPar: a
```

Exercice 4.34

```
| figure a b c b1 b2 b3 o |
figure := DrGeoFigure nouveau.
a := (figure point: 2@1) nommer: 'A'.
b := (figure point: 7@2) nommer: 'B'.
c := (figure point: 4@7) nommer: 'C'.
figure polygone: {a . b . c}.
b1 := figure bissectriceSommet: a cote1: b cote2: c.
b2 := figure bissectriceSommet: b cote1: a cote2: c.
b3 := figure bissectriceSommet: c cote1: b cote2: a.
o := (figure intersectionDe: b1 et: b2) nommer: 'O'
```

Exercice 4.35

```
| figure a b c b1 b2 b3 o s1 h |
figure := DrGeoFigure nouveau.
a := (figure point: 2@1) nommer: 'A'.
b := (figure point: 7@2) nommer: 'B'.
c := (figure point: 4@7) nommer: 'C'.
figure polygone: {a . b . c}.
b1 := figure bissectriceSommet: a cote1: b cote2: c.
b2 := figure bissectriceSommet: b cote1: a cote2: c.
b3 := figure bissectriceSommet: c cote1: b cote2: a.
o := (figure intersectionDe: b1 et: b2) nommer: 'O'.
s1 := figure segmentDe: a a: b.
h := figure
   intersectionDe: s1
   et: (figure perpendiculaireA: s1 passantPar: o).
figure cercleCentre: o passantPar: h
```

Exercice 4.36

```
| figure a b c ab bc ac h1 h2 h3 |
figure := DrGeoFigure nouveau.
a := (figure point: 2@1) nommer: 'A'.
b := (figure point: 7@2) nommer: 'B'.
c := (figure point: 4@7) nommer: 'C'.
figure polygone: {a . b . c}.
ab := figure segmentDe: a a: b.
bc := figure segmentDe: b a: c.
ac := figure segmentDe: a a: c.
h1 := figure perpendiculaireA: ab passantPar: c.
h2 := figure perpendiculaireA: bc passantPar: a.
h3 := figure perpendiculaireA: ac passantPar: b.
(figure intersectionDe: h1 et: h2) nommer: 'H'
```

Exercice 4.36

```
| figure a b c mi1 mi2 mi3 m1 m2 m3 |
figure := DrGeoFigure nouveau.
a := (figure point: 2@1) nommer: 'A'.
b := (figure point: 7@2) nommer: 'B'.
c := (figure point: 4@7) nommer: 'C'.
figure polygone: {a . b . c}.
mi1 := figure milieuDe: a et: b.
mi2 := figure milieuDe: b et: c.
mi3 := figure milieuDe: a et: c.
m1 := figure droitePassantPar: a et: mi2.
m2 := figure droitePassantPar: b et: mi3.
m3 := figure droitePassantPar: c et: mi1.
(figure intersectionDe: m1 et: m2) nommer: 'G'
```

Exercice 4.38

```
| figure a b c ab bc d1 m mac|
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
b := (figure point: 6@0) nommer: 'B'.
c := (figure point: 4@9) nommer: 'C'.
ab := figure droitePassantPar: a et: b.
(figure segmentDe: a a: b) normal.
bc := (figure segmentDe: b a: c) normal.
(figure segmentDe: a a: c) normal.
(figure angleGeometriqueCentre: b de: a a: c) couleur: Color red.
(figure angleGeometriqueCentre: a de: b a: c) couleur: Color blue.
(figure angleGeometriqueCentre: c de: a a: b) couleur: Color brown.
d1 := figure paralleleA: bc passantPar: a.
m := (figure pointSurLigne: d1 a: 0.89) nommer: 'M'.
(figure angleGeometriqueCentre: a de: m a: c) couleur: Color brown
```

Exercice 4.39

A ajouter à la suite de la solution de l'Exercice 4.38.

```
| figure ... n |
../..
n := (figure pointSurLigne: ab a: 0.2) nommer: 'N'.
(figure angleGeometriqueCentre: a de: m a: n) couleur: Color red
```

Exercice 4.40

```
| figure ancre a b c d |
figure := DrGeoFigure nouveau.
figure polygone: { 0@0. 6@0. -3@8 . 4@9 }.
(figure droitePassantPar: 0@0 et: 4@9) pointille.
a := figure angleGeometriqueCentre: 0@0 de: 6@0 a: 4@9.
b := figure angleGeometriqueCentre:  6@0 de: -3@8 a: 0@0.
c := figure angleGeometriqueCentre: 4@9 de: -3@8 a: 0@0.
d := figure angleGeometriqueCentre: -3@8 de: 6@0 a: 4@9.
ancre := figure point: -2 @ -2.
figure point: [
   ancre nommer: 'Somme des angles : ',
```

```
(a mathItem  degreeAngle
+ b mathItem degreeAngle
+ c mathItem degreeAngle
+ d mathItem degreeAngle) rounded asString]
```

Exercice 4.41

```
| figure carre o |
figure := DrGeoFigure nouveau.
o := figure point: 3 @ -2.
carre := figure polygone: { 0@0. 4@0. 4@4 . 0@4 }.
figure symetriqueDe: carre selonCentre: o
```

Exercice 4.42

```
| figure carre d |
figure := DrGeoFigure nouveau.
d := figure droitePassantPar: -3 @ 3 et: -8 @ 0.
carre := figure polygone: { 0@0. 4@0. 4@4 . 0@4 }.
figure symetriqueDe: carre selonAxe: d
```

Exercice 4.43

```
| figure carre a b v |
figure := DrGeoFigure nouveau.
a := figure point: -1 @ -1.
b := figure point: -4 @ -3.
v := figure vecteurOrigine: a extremite: b.
carre := figure polygone: { 1@0. 5@0. 5@4 . 1@4 }.
figure translationDe: carre parVecteur: v
```

Exercice 4.44

```
| figure carre o a1 a2 |
figure := DrGeoFigure nouveau.
o := figure point: 0 @ 0.
a1 := 90 degreesToRadians.
a2 := -90 degreesToRadians.
carre := figure polygone: { 0@0. 4@0. 4@4 . 0@4 }.
figure rotationDe: carre parCentre: o etAngle: a1.
figure rotationDe: carre parCentre: o etAngle: a2
```

Exercice 4.45

```
| figure carre a b k1 k2 |
figure := DrGeoFigure nouveau afficherAxes afficherGrille.
a := figure point: -8 @ 5.
b := figure point: 4 @ -7.
k1 := -1/2.
k2 := 5/2.
carre := figure polygone: { 0@0. 4@0. 4@4 . 0@4 }.
figure homothetieDe: carre parCentre: a etFacteur: k1.
figure homothetieDe: carre parCentre: b etFacteur: k2
```

Exercice 4.46

Dans la collection, il est nécessaire d'envoyer le message `#montrer` au point. En effet il a été créé en même temps que le cercle mais masqué. Nous invoquons ce point et demandons qu'il se montre.

```
| figure collection d |
figure := DrGeoFigure nouveau.
 := figure droitePassantPar: -7 @ 0 et: 0 @ -8.
collection := {figure segmentDe: -2 @ 2 a: 2 @ 2 .
   figure segmentDe: 2 @ 2 a: 2 @ -2 .
   figure segmentDe: 2 @ -2 a: -2 @ -2 .
   figure segmentDe: -2 @ -2 a: -2 @ 2 .
   figure cercleCentre: 0 @ 0 rayon: 2.
   figure segmentDe: 2 @ 2 a: -2 @ -2.
   figure segmentDe: 2 @ -2 a: -2 @ 2.
   (figure point: 0 @ 0) montrer}.
collection faire: [:forme | figure symetriqueDe: forme selonAxe: d]
```

Exercice 4.47

```
| figure collection o k |
figure := DrGeoFigure nouveau.
d := figure point: -10 @ -10.
k := 1/4.
collection := {figure segmentDe: -2 @ 2 a: 2 @ 2 .
   figure segmentDe: 2 @ 2 a: 2 @ -2 .
   figure segmentDe: 2 @ -2 a: -2 @ -2 .
   figure segmentDe: -2 @ -2 a: -2 @ 2 .
   figure cercleCentre: 0 @ 0 rayon: 2}.
collection faire: [:forme |
   figure homothetieDe: forme parCentre: o etFacteur: k]
```

Exercice 4.48

```
| figure collection d |
figure := DrGeoFigure nouveau.
d := (figure droitePassantPar: 4@0 et: 4@5) cacher.
collection := {figure segmentDe: 4@1 a: 1@1.
   figure segmentDe: 1@1 a: 1@4.
   figure segmentDe: 1@4 a: 4@4.
   figure segmentDe: 4@4 a: 4@2.
   figure segmentDe: 4@2 a: 2@2.
   figure segmentDe: 2@2 a: 2@3.
   figure segmentDe: 2@3 a: 3@3}.
collection faire: [:forme|
   figure symetriqueDe: forme selonAxe: d]
```

Exercice 4.49

```
| figure collection o |
figure := DrGeoFigure nouveau.
o := figure point: -1 @ -1.
collection := {figure cercleCentre: 3@3 rayon: 3.
   figure cercleCentre: 2@4 rayon: 1/2.
   figure cercleCentre: 4@4 rayon: 1/2.
   figure polygone: {(3/2)@2 . (5/2)@(3/2) . (7/2)@(3/2).
```

```
      (9/2)@2 . (7/2)@1 . (5/2)@1} }.
collection faire: [:forme|
   figure symetriqueDe: forme selonCentre: o]
```

Exercice 4.50

```
| figure collection o1 o2 o3 o4 o5 |
figure := DrGeoFigure nouveau.
o1 := figure point: 4@2.5.
o2 := figure point: 1@2.5.
o3 := figure point: -2@2.5.
o4 := figure point: -5@2.5.
o5 := figure point: -8@2.5.
collection := {(figure segmentDe: 7@1 a: 4@1) normal.
   (figure segmentDe: 4@1 a: 4@4) normal.
   (figure segmentDe: 4@4 a: 7@4) normal.
   (figure segmentDe: 7@4 a: 7@2) normal.
   (figure segmentDe: 7@2 a: 5@2) normal.
   (figure segmentDe: 5@2 a: 5@3) normal.
   (figure segmentDe: 5@3 a: 6@3) normal}.
{o1 . o2 . o3 . o4 . o5} faire: [:centre |
   collection := collection collecter: [:forme |
      figure symetriqueDe: forme selonCentre: centre] ]
```

Exercice 4.51

```
| figure collection |
figure := DrGeoFigure nouveau.
collection := {figure segmentDe: 0@0 a: (1/2)@0.
   figure segmentDe: (1/2)@0 a: 2@1.
   figure segmentDe: 2@1 a: 2@0.
   figure segmentDe: 2@0 a: 3@0}
```

Exercice 4.53

```
| figure collection |
figure := DrGeoFigure nouveau.
collection := {figure segmentDe: 0@0 a: 4@0.
   figure segmentDe: 0@4 a: 4@4.
   figure segmentDe: 0@1 a: 0@3.
   figure segmentDe: 0@3 a: 3@3.
   figure segmentDe: 3@3 a: 3@2.
   figure segmentDe: 3@2 a: 2@2.
   figure segmentDe: 2@2 a: 2@1.
   figure segmentDe: 2@1 a: 4@1}
```

Exercice 4.52

```
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
b := (figure point: 5@1) nommer: 'B'.
ab := figure droitePassantPar: a et: b.
droite := figure perpendiculaireA:  ab passantPar: b.
cercle := figure cercleCentre: b passantPar: a.
```

```
c := figure intersectionDe: droite et: cercle.
c nommer: 'C'
```

Exercice 4.21

```
| figure a b c ab bc droite cercle |
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
b := (figure point: 5@1) nommer: 'B'.
ab := figure droitePassantPar: a et: b.
droite := figure perpendiculaireA:  ab passantPar: b.
cercle := figure cercleCentre: b passantPar: a.
c := figure intersectionDe: droite et: cercle.
c nommer: 'C'.
bc := figure segmentDe: b a: c.
d := figure
  intersectionDe: (figure paralleleA: ab passantPar: c) cacher
  et: (figure paralleleA: bc passantPar: a) cacher.
d nommer: 'D'.
figure segmentDe: a a: d.
figure segmentDe: c a: d.
figure segmentDe: a a: b.
ab cacher.
cercle cacher.
droite cacher.
```

Exercice 4.22

```
| figure a b c d i ab droite cercle |
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
b := (figure point: 5@1) nommer: 'B'.
ab := figure droitePassantPar: a et: b.
droite := figure perpendiculaireA:  ab passantPar: b.
cercle := figure cercleCentre: b passantPar: a.
c := figure intersectionDe: droite et: cercle.
c nommer: 'C'.
i := (figure milieuDe: a et: c) nommer: 'I'.
d := (figure symetriqueDe: b selonCentre: i) nommer: 'D'.
figure polygone: {a . b . c . d}
```

Exercice 4.23

```
| figure a b c cercle1 cercle2 |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
cercle1 := figure cercleCentre: b rayon: 4.
cercle2 := figure cercleCentre: c rayon: 4.
a := (figure intersectionDe: cercle1 et: cercle2) nommer: 'A'.
cercle1 cacher.
cercle2 cacher.
figure polygone: {a . b . c}
```

Exercice 4.24

```
| figure a b c bc mediatrice |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
bc := figure segmentDe: b a: c.
mediatrice := figure mediatrice: bc.
a := (figure pointSurLigne: mediatrice a: 0.2) nommer: 'A'.
figure polygone: {a . b . c}
```

Exercice 4.25

La difficulté pour terminer l'exercice vient de l'angle nécessaire pour la 2e rotation, ce n'est pas le même. C'est le complémentaire à 360 degrés du premier angle. Le plus simple étant définir un deuxième angle *alpha2* en inversant les deux extrémités du premier angle.

```
| figure alpha1 alpha2 b c b1 c1 demiDroite1 demiDroite2|
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
alpha1 := figure angleCentre: 10@10 de: 12@10 a: 12@13.
b1 := figure rotationDe: b parCentre: c etAngle: alpha1.
demiDroite1 := figure demiDroiteOrigine: c passantPar: b1.
alpha2 := figure angleCentre: 10@10 de: 12@13 a: 12@10.
c1 := figure rotationDe: c parCentre: b etAngle: alpha2.
demiDroite2 := figure demiDroiteOrigine: b passantPar: c1.
a := (figure intersectionDe: demiDroite1 et: demiDroite2) nommer: 'A'.
figure polygone: {a . b . c}
```

Exercice 4.26

```
| figure alpha1 alpha2 b c b1 c1 demiDroite1 demiDroite2|
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
alpha1 := figure angleCentre: 10@10 de: 12@10 a: 12@13.
b1 := figure rotationDe: b parCentre: c etAngle: alpha1.
demiDroite1 := figure demiDroiteOrigine: c passantPar: b1.
alpha2 := figure angleCentre: 10@10 de: 12@13 a: 12@10.
c1 := figure rotationDe: c parCentre: b etAngle: alpha2.
demiDroite2 := figure demiDroiteOrigine: b passantPar: c1.
a := (figure intersectionDe: demiDroite1 et: demiDroite2) nommer: 'A'.
b1 cacher.
c1 cacher.
demiDroite1 cacher.
demiDroite2 cacher.
(figure point: 12@10) montrer.
figure polygone: {a . b . c}
```

Exercice 4.27

```
| figure a b c cercle1 cercle2 |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
```

```
cercle1 := figure cercleCentre: b passantPar: c.
cercle2 := figure cercleCentre: c passantPar: b.
a := figure intersectionDe: cercle1 et: cercle2.
a nommer: 'A'.
cercle1 cacher.
cercle2 cacher.
figure polygone: {a . b . c}
```

Exercice 4.28

```
| figure a b c cercle mediatrice |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
cercle := figure cercleCentre: b passantPar: c.
mediatrice := figure mediatriceDe: b a: c.
a := figure intersectionDe: cercle et: mediatrice.
a nommer: 'A'.
cercle cacher.
mediatrice cacher.
figure polygone: {a . b . c}
```

Exercice 4.29

```
| figure a b c bc perp |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
bc := figure segmentDe: b a: c.
perp := figure perpendiculaireA: bc passantPar: b.
a := (figure pointSurLigne: perp a: 0.1) nommer: 'A'.
figure polygone: {a . b . c}
```

Exercice 4.30

```
| figure a b c bc perp cercle |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
bc := figure segmentDe: b a: c.
perp := figure perpendiculaireA: bc passantPar: b.
cercle := figure cercleCentre: b passantPar: c.
a := (figure intersectionDe: perp et: cercle) nommer: 'A'.
figure polygone: {a . b . c}
```

Exercice 4.31

```
| figure a b c bc ba perp cercle |
figure := DrGeoFigure nouveau.
b := (figure point: 5@1) nommer: 'B'.
c := (figure point: 0@0) nommer: 'C'.
bc := figure segmentDe: b a: c.
perp := figure perpendiculaireA: bc passantPar: b.
cercle := figure cercleCentre: b passantPar: c.
a := (figure intersectionDe: perp et: cercle) nommer: 'A'.
```

```
figure polygone: {a . b . c}.
perp cacher.
cercle cacher.
figure angleGeometriqueCentre: b de: a a: c.
bc marquerAvecSimpleTrait.
ba := figure segmentDe: b a: a.
ba marquerAvecSimpleTrait.
```

Exercice 4.32

```
| figure a b c m1 m2 m3 m |
figure := DrGeoFigure nouveau.
a := (figure point: 2@1) nommer: 'A'.
b := (figure point: 7@2) nommer: 'B'.
c := (figure point: 4@7) nommer: 'C'.
figure polygone: {a . b . c}.
m1 := figure mediatriceDe: a a: b.
m2 := figure mediatriceDe: b a: c.
m3 := figure mediatriceDe: a a: c.
m := (figure intersectionDe: m1 et: m2) nommer: 'M'
```

Exercice 4.33

```
| figure a b c m1 m2 m3 m |
figure := DrGeoFigure nouveau.
a := (figure point: 2@1) nommer: 'A'.
b := (figure point: 7@2) nommer: 'B'.
c := (figure point: 4@7) nommer: 'C'.
figure polygone: {a . b . c}.
m1 := figure mediatriceDe: a a: b.
m2 := figure mediatriceDe: b a: c.
m3 := figure mediatriceDe: a a: c.
m := (figure intersectionDe: m1 et: m2) nommer: 'M'.
figure cercleCentre: m passantPar: a
```

Exercice 4.34

```
| figure a b c b1 b2 b3 o |
figure := DrGeoFigure nouveau.
a := (figure point: 2@1) nommer: 'A'.
b := (figure point: 7@2) nommer: 'B'.
c := (figure point: 4@7) nommer: 'C'.
figure polygone: {a . b . c}.
b1 := figure bissectriceSommet: a cote1: b cote2: c.
b2 := figure bissectriceSommet: b cote1: a cote2: c.
b3 := figure bissectriceSommet: c cote1: b cote2: a.
o := (figure intersectionDe: b1 et: b2) nommer: 'O'
```

Exercice 4.35

```
| figure a b c b1 b2 b3 o s1 h |
figure := DrGeoFigure nouveau.
a := (figure point: 2@1) nommer: 'A'.
b := (figure point: 7@2) nommer: 'B'.
c := (figure point: 4@7) nommer: 'C'.
```

```
figure polygone: {a . b . c}.
b1 := figure bissectriceSommet: a cote1: b cote2: c.
b2 := figure bissectriceSommet: b cote1: a cote2: c.
b3 := figure bissectriceSommet: c cote1: b cote2: a.
o := (figure intersectionDe: b1 et: b2) nommer: 'O'.
s1 := figure segmentDe: a a: b.
h := figure
   intersectionDe: s1
   et: (figure perpendiculaireA: s1 passantPar: o).
figure cercleCentre: o passantPar: h
```

Exercice 4.36

```
| figure a b c ab bc ac h1 h2 h3 |
figure := DrGeoFigure nouveau.
a := (figure point: 2@1) nommer: 'A'.
b := (figure point: 7@2) nommer: 'B'.
c := (figure point: 4@7) nommer: 'C'.
figure polygone: {a . b . c}.
ab := figure segmentDe: a a: b.
bc := figure segmentDe: b a: c.
ac := figure segmentDe: a a: c.
h1 := figure perpendiculaireA: ab passantPar: c.
h2 := figure perpendiculaireA: bc passantPar: a.
h3 := figure perpendiculaireA: ac passantPar: b.
(figure intersectionDe: h1 et: h2) nommer: 'H'
```

Exercice 4.36

```
| figure a b c mi1 mi2 mi3 m1 m2 m3 |
figure := DrGeoFigure nouveau.
a := (figure point: 2@1) nommer: 'A'.
b := (figure point: 7@2) nommer: 'B'.
c := (figure point: 4@7) nommer: 'C'.
figure polygone: {a . b . c}.
mi1 := figure milieuDe: a et: b.
mi2 := figure milieuDe: b et: c.
mi3 := figure milieuDe: a et: c.
m1 := figure droitePassantPar: a et: mi2.
m2 := figure droitePassantPar: b et: mi3.
m3 := figure droitePassantPar: c et: mi1.
(figure intersectionDe: m1 et: m2) nommer: 'G'
```

Exercice 4.38

```
| figure a b c ab bc d1 m mac|
figure := DrGeoFigure nouveau.
a := (figure point: 0@0) nommer: 'A'.
b := (figure point: 6@0) nommer: 'B'.
c := (figure point: 4@9) nommer: 'C'.
ab := figure droitePassantPar: a et: b.
(figure segmentDe: a a: b) normal.
bc := (figure segmentDe: b a: c) normal.
(figure segmentDe: a a: c) normal.
(figure angleGeometriqueCentre: b de: a a: c) couleur: Color red.
```

```
(figure angleGeometriqueCentre: a de: b a: c) couleur: Color blue.
(figure angleGeometriqueCentre: c de: a a: b) couleur: Color brown.
d1 := figure paralleleA: bc passantPar: a.
m := (figure pointSurLigne: d1 a: 0.89) nommer: 'M'.
(figure angleGeometriqueCentre: a de: m a: c) couleur: Color brown
```

Exercice 4.39

A ajouter à la suite de la solution de l'Exercice 4.38.

```
| figure ... n |
../..
n := (figure pointSurLigne: ab a: 0.2) nommer: 'N'.
(figure angleGeometriqueCentre: a de: m a: n) couleur: Color red
```

Exercice 4.40

```
| figure ancre a b c d |
figure := DrGeoFigure nouveau.
figure polygone: { 0@0. 6@0. -3@8 . 4@9 }.
(figure droitePassantPar: 0@0 et: 4@9) pointille.
a := figure angleGeometriqueCentre: 0@0 de: 6@0 a: 4@9.
b := figure angleGeometriqueCentre:  6@0 de: -3@8 a: 0@0.
c := figure angleGeometriqueCentre: 4@9 de: -3@8 a: 0@0.
d := figure angleGeometriqueCentre: -3@8 de: 6@0 a: 4@9.
ancre := figure point: -2 @ -2.
figure point: [
   ancre nommer: 'Somme des angles : ',
      (a mathItem  degreeAngle
      + b mathItem degreeAngle
      + c mathItem degreeAngle
      + d mathItem degreeAngle) rounded asString]
```

Exercice 4.41

```
| figure carre o |
figure := DrGeoFigure nouveau.
o := figure point: 3 @ -2.
carre := figure polygone: { 0@0. 4@0. 4@4 . 0@4 }.
figure symetriqueDe: carre selonCentre: o
```

Exercice 4.42

```
| figure carre d |
figure := DrGeoFigure nouveau.
d := figure droitePassantPar: -3 @ 3 et: -8 @ 0.
carre := figure polygone: { 0@0. 4@0. 4@4 . 0@4 }.
figure symetriqueDe: carre selonAxe: d
```

Exercice 4.43

```
| figure carre a b v |
figure := DrGeoFigure nouveau.
a := figure point: -1 @ -1.
b := figure point: -4 @ -3.
v := figure vecteurOrigine: a extremite: b.
carre := figure polygone: { 1@0. 5@0. 5@4 . 1@4 }.
```

```
figure translationDe: carre parVecteur: v
```

Exercice 4.44

```
| figure carre o a1 a2 |
figure := DrGeoFigure nouveau.
o := figure point: 0 @ 0.
a1 := 90 degreesToRadians.
a2 := -90 degreesToRadians.
carre := figure polygone: { 0@0. 4@0. 4@4 . 0@4 }.
figure rotationDe: carre parCentre: o etAngle: a1.
figure rotationDe: carre parCentre: o etAngle: a2
```

Exercice 4.45

```
| figure carre a b k1 k2 |
figure := DrGeoFigure nouveau afficherAxes afficherGrille.
a := figure point: -8 @ 5.
b := figure point: 4 @ -7.
k1 := -1/2.
k2 := 5/2.
carre := figure polygone: { 0@0. 4@0. 4@4 . 0@4 }.
figure homothetieDe: carre parCentre: a etFacteur: k1.
figure homothetieDe: carre parCentre: b etFacteur: k2
```

Exercice 4.46

Dans la collection, il est nécessaire d'envoyer le message `#montrer` au point. En effet il a été créé en même temps que le cercle mais masqué. Nous invoquons ce point et demandons qu'il se montre.

```
| figure collection d |
figure := DrGeoFigure nouveau.
 := figure droitePassantPar: -7 @ 0 et: 0 @ -8.
collection := {figure segmentDe: -2 @ 2 a: 2 @ 2 .
   figure segmentDe: 2 @ 2 a: 2 @ -2 .
   figure segmentDe: 2 @ -2 a: -2 @ -2 .
   figure segmentDe: -2 @ -2 a: -2 @ 2 .
   figure cercleCentre: 0 @ 0 rayon: 2.
   figure segmentDe: 2 @ 2 a: -2 @ -2.
   figure segmentDe: 2 @ -2 a: -2 @ 2.
   (figure point: 0 @ 0) montrer}.
collection faire: [:forme | figure symetriqueDe: forme selonAxe: d]
```

Exercice 4.47

```
| figure collection o k |
figure := DrGeoFigure nouveau.
d := figure point: -10 @ -10.
k := 1/4.
collection := {figure segmentDe: -2 @ 2 a: 2 @ 2 .
   figure segmentDe: 2 @ 2 a: 2 @ -2 .
   figure segmentDe: 2 @ -2 a: -2 @ -2 .
   figure segmentDe: -2 @ -2 a: -2 @ 2 .
   figure cercleCentre: 0 @ 0 rayon: 2}.
collection faire: [:forme |
   figure homothetieDe: forme parCentre: o etFacteur: k]
```

Exercice 4.48

```
| figure collection d |
figure := DrGeoFigure nouveau.
d := (figure droitePassantPar: 4@0 et: 4@5) cacher.
collection := {figure segmentDe: 4@1 a: 1@1.
   figure segmentDe: 1@1 a: 1@4.
   figure segmentDe: 1@4 a: 4@4.
   figure segmentDe: 4@4 a: 4@2.
   figure segmentDe: 4@2 a: 2@2.
   figure segmentDe: 2@2 a: 2@3.
   figure segmentDe: 2@3 a: 3@3}.
collection faire: [:forme|
   figure symetriqueDe: forme selonAxe: d]
```

Exercice 4.49

```
| figure collection o |
figure := DrGeoFigure nouveau.
o := figure point: -1 @ -1.
collection := {figure cercleCentre: 3@3 rayon: 3.
   figure cercleCentre: 2@4 rayon: 1/2.
   figure cercleCentre: 4@4 rayon: 1/2.
   figure polygone: {(3/2)@2 . (5/2)@(3/2) . (7/2)@(3/2).
      (9/2)@2 . (7/2)@1 . (5/2)@1} }.
collection faire: [:forme|
   figure symetriqueDe: forme selonCentre: o]
```

Exercice 4.50

```
| figure collection o1 o2 o3 o4 o5 |
figure := DrGeoFigure nouveau.
o1 := figure point: 4@2.5.
o2 := figure point: 1@2.5.
o3 := figure point: -2@2.5.
o4 := figure point: -5@2.5.
o5 := figure point: -8@2.5.
collection := {(figure segmentDe: 7@1 a: 4@1) normal.
   (figure segmentDe: 4@1 a: 4@4) normal.
   (figure segmentDe: 4@4 a: 7@4) normal.
   (figure segmentDe: 7@4 a: 7@2) normal.
   (figure segmentDe: 7@2 a: 5@2) normal.
   (figure segmentDe: 5@2 a: 5@3) normal.
   (figure segmentDe: 5@3 a: 6@3) normal}.
{o1 . o2 . o3 . o4 . o5} faire: [:centre |
   collection := collection collecter: [:forme |
      figure symetriqueDe: forme selonCentre: centre] ]
```

Exercice 4.51

```
| figure collection |
figure := DrGeoFigure nouveau.
collection := {figure segmentDe: 0@0 a: (1/2)@0.
   figure segmentDe: (1/2)@0 a: 2@1.
   figure segmentDe: 2@1 a: 2@0.
```

```
   figure segmentDe: 2@0 a: 3@0}
```

Exercice 4.53

```
| figure collection |
figure := DrGeoFigure nouveau.
collection := {figure segmentDe: 0@0 a: 4@0.
   figure segmentDe: 0@4 a: 4@4.
   figure segmentDe: 0@1 a: 0@3.
   figure segmentDe: 0@3 a: 3@3.
   figure segmentDe: 3@3 a: 3@2.
   figure segmentDe: 3@2 a: 2@2.
   figure segmentDe: 2@2 a: 2@1.
   figure segmentDe: 2@1 a: 4@1}
```

Exercice 4.52

```
| figure collection v |
figure := DrGeoFigure nouveau.
v := figure vecteur: 3 @ 0.
collection := {figure segmentDe: 0 @ 0 a: (1/2) @ 0.
   figure segmentDe: (1/2) @ 0 a: 2 @ 1.
   figure segmentDe: 2 @1 a: 2 @ 0.
   figure segmentDe: 2 @ 0 a: 3 @ 0}.
5 foisRepete: [
   collection := collection collecter: [:forme |
     figure translationDe: forme parVecteur:v ] ]
```

Exercice 4.54

```
| figure collection v |
figure := DrGeoFigure nouveau.
v := figure vecteur: 4@0.
collection := {figure segmentDe: 0@0 a: 4@0.
   figure segmentDe: 0@4 a: 4@4.
   figure segmentDe: 0@1 a: 0@3.
   figure segmentDe: 0@3 a: 3@3.
   figure segmentDe: 3@3 a: 3@2.
   figure segmentDe: 3@2 a: 2@2.
   figure segmentDe: 2@2 a: 2@1.
   figure segmentDe: 2@1 a: 4@1}.
5 foisRepete: [
  collection := collection collecter: [:forme |
     figure translationDe: forme parVecteur:v ] ]
```

Exercice 4.55

```
| figure collection v |
figure := DrGeoFigure nouveau.
v := figure vecteur: 4@0.
collection := {figure segmentDe: 0@0 a: 4@0.
   figure segmentDe: 0@4 a: 4@4.
   figure segmentDe: 0@1 a: 0@3.
   figure segmentDe: 0@3 a: 3@3.
   figure segmentDe: 3@3 a: 3@2.
```

```
    figure segmentDe: 3@2 a: 2@2.
    figure segmentDe: 2@2 a: 2@1.
    figure segmentDe: 2@1 a: 4@1}.
  collection faire: [:forme | forme epais].
  5 foisRepete: [
    collection := collection collecter: [:forme |
      figure translationDe: forme parVecteur:v ] ]
```

Exercice 4.56

Deux solutions à cet exercice sont proposées. La première ci-dessous est dans la suite de ce qui a été appris jusqu'à présente. Elle a l'avantage d'être relativement facile à comprendre, mais son code est assez répétitif.

L'autre solution est écrite comme le ferait un programmeur professionnel, le code n'est pas répétitif et utilise un message subtile `injecter:dans:` pour construire la ligne du motif à partir de la liste de ses sommets.

Solution naïve.

```
| figure sommets collection v |
figure := DrGeoFigure nouveau.
v := figure vecteur: 5@0.
collection := {figure segmentDe: 0@0 a: 5@0.
   figure segmentDe: 0@6 a: 5@6.
   figure segmentDe: 0@1 a: 0@5.
   figure segmentDe: 0@5 a: 4@5.
   figure segmentDe: 4@5 a: 4@2.
   figure segmentDe: 4@2 a: 2@2.
   figure segmentDe: 2@2 a: 2@3.
   figure segmentDe: 2@3 a: 3@3.
   figure segmentDe: 3@3 a: 3@4.
   figure segmentDe: 3@4 a: 1@4.
   figure segmentDe: 1@4 a: 1@1.
   figure segmentDe: 1@1 a: 5@1}.
5 foisRepete: [
  collection := collection collecter: [:forme |
    figure translationDe: forme parVecteur:v ] ]
```

Solution experte.

```
| figure sommets collection v |
figure := DrGeoFigure nouveau.
v := figure vecteur: 5@0.
collection := OrderedCollection new.
collection
   ajouter: (figure segmentDe: 0@0 a: 5@0);
   ajouter: (figure segmentDe: 0@6 a: 5@6).
sommets := {0@5. 4@5. 4@2. 2@2. 2@3. 3@3. 3@4. 1@4. 1@1. 5@1}.
sommets injecter: 0@1 dans: [ :pointPrec : pointSuiv |
   collection ajouter: (figure segmentDe: pointPrec a: pointSuiv).
   pointSuiv].
5 foisRepete: [
   collection := collection collecter: [:forme |
      figure translationDe: forme parVecteur:v ] ]
```

Exercice 4.57

```
| figure collection |
figure := DrGeoFigure nouveau.
collection := {figure segmentDe: 0@0 a: (1/2)@0.
   figure segmentDe: (1/2)@0 a: (1/2)@2.
   figure segmentDe: (1/2)@2 a: 1@2}
```

Exercice 4.58

```
| figure collection symetriques axe v |
figure := DrGeoFigure nouveau.
axe := figure droitePassantPar: 1@0 et: 1@3.
v := figure vecteur: 2@0.
collection := {figure segmentDe: 0@0 a: (1/2)@0.
   figure segmentDe: (1/2)@0 a: (1/2)@2.
   figure segmentDe: (1/2)@2 a: 1@2} commeCollectionOrdonnee.
symetriques := collection collecter: [:forme |
   figure symetriqueDe: forme selonAxe: axe].
collection ajouterTout: symetriques.
5 foisRepete: [
   collection := collection collecter: [:forme |
      figure translationDe: forme parVecteur:v ] ]
```

Exercice 4.59

```
| figure collection symetriques centre v |
figure := DrGeoFigure nouveau.
centre := figure point: 3@0.
v := figure vecteur: 6@0.
collection := {figure segmentDe: 0@3 a: 1@1.
   figure segmentDe: 1@1 a: 3@0.
   figure segmentDe: 3@0 a: 2@2.
   figure segmentDe: 2@2 a: 0@3} commeCollectionOrdonnee.
symetriques := collection collecter: [:forme |
   figure symetriqueDe: forme selonCentre: centre].
collection ajouterTout: symetriques.
5 foisRepete: [
   collection := collection collecter: [:forme |
      figure translationDe: forme parVecteur:v ] ]
```

Exercice 4.60

```
| figure collection symetriques centre v d |
figure := DrGeoFigure nouveau.
centre := figure point: 3@0.
v := figure vecteur: 6@0.
d := figure droitePassantPar: 3@0 et: 3@1.
collection := {figure segmentDe: 0@3 a: 1@1.
   figure segmentDe: 1@1 a: 3@0.
   figure segmentDe: 3@0 a: 2@2.
   figure segmentDe: 2@2 a: 0@3} commeCollectionOrdonnee.
"Construction de Motif 2, symétrique de Motif 1 selon le centre"
symetriques := collection collecter: [:forme |
   figure symetriqueDe: forme selonCentre: centre].
```

```
collection ajouterTout: symetriques.
"Construction de Motif 3 et Motif 4, symétriques de Motif 1 et
Motif 2 selon l'axe d"
symetriques := collection collecter: [:forme |
   figure symetriqueDe: forme selonAxe: d].
collection ajouterTout: symetriques.
5 foisRepete: [
   collection := collection collecter: [:forme |
      figure translationDe: forme parVecteur:v ] ]
```

D.4 Fonctions

Exercice 5.1

L'affectation du bloc de code définissant la fonction à une variable `f` est superflue puisque le bloc de code est utilisé une seule fois dans le programme.

```
| figure |
figure := DrGeoFigure nouveau afficherAxes afficherGrille.
figure courbeDe:  [:x | -2 * x] de: -5 a: 5
```

Exercice 5.2

Lorsque la valeur de a est positive, la droite est montante de la gauche vers la droite. La fonction linéaire est dite *croissante*.

Lorsque la valeur de a est négative, la droite est descendante de la gauche vers la droite. La fonction linéaire est dite *décroissante*.

Lorsque $a = 0$, la droite est confondue avec le premier axe (abscisses). La fonction linéaire est dite constante.

Exercice 5.3

```
| figure f a |
figure := DrGeoFigure nouveau afficherAxeafficherGrille echelle: 50.
a := figure decimal: 1 a: 5 @ -5 min: -8 max: 8 nom: 'a' afficherValeur: true.
f := [:x | a valeur * x].
figure courbeDe: f de: -10 a: 10
```

Exercice 5.4

```
| figure f a b |
figure := DrGeoFigure nouveau afficherAxes afficherGrille echelle: 50.
a := figure entier: 1 a: 5 @ -5 min: -8 max: 8 nom: 'a' afficherValeur: true.
b := figure entier: 1 a: 5 @ -6 min: -8 max: 8 nom: 'b' afficherValeur: true.
f := [:x | a valeur * x + b valeur].
figure courbeDe: f de: -10 a: 10
```

Exercice 5.5

Lorsque la valeur de b augmente, la droite se déplace parallèlement vers le haut de la figure, dans le sens positif de l'axe vertical (ordonnées).

Lorsque la valeur de b diminue, la droite se déplace parallèlement vers le bas de la figure, dans le sens négatif de l'axe vertical (ordonnées).

Lorsque $b = 0$, la droite passe par l'origine des axes, la fonction est alors linéaire.

Lorsque $a = 0$, la droite est parallèle au premier axe (abscisses). La fonction est dite constante de la forme $x \longmapsto b$.

Exercice 5.6

Attention. Dans l'expression de la fonction quadratique, dans le bloc de code, les parenthèses sont nécessaires autour de la deuxième multiplication. En effet, comme expliqué au chapitre sur la syntaxe, le système a une notion différente des priorités (priorités des messages et non pas des opérateurs).

```
| figure f a b c|
figure := DrGeoFigure nouveau afficherAxes afficherGrille echelle: 50.
a := figure decimal: 1 a: 5 @ -1 min: -8 max: 8 nom: 'a' afficherValeur: true.
b := figure decimal: 1 a: 5 @ -2 min: -8 max: 8 nom: 'b' afficherValeur: true.
c := figure decimal: 1 a: 5 @ -3 min: -8 max: 8 nom: 'c' afficherValeur: true.
f := [:x | a valeur * x squared + (b valeur * x) + c valeur].
figure courbeDe: f de: -10 a: 10
```

Exercice 5.7

Lorsque le signe de a est positif, la parabole est orientée vers le haut, ses branches partent vers l'infini positif.

Lorsque le signe de a est négatif, la parabole est orientée vers le bas, ses branches partent vers l'infini négatif.

Lorsque a est égale à zéro, c'est une fonction affine de pente b et ordonnée à l'origine c.

Exercice 5.8

```
| figure f n|
figure := DrGeoFigure nouveau afficherAxes afficherGrille echelle: 50.
n := figure entier: 1 a: 5 @ -1 min: 1 max: 7 nom: 'n' afficherValeur: true.
f := [:x | x puissance: n valeur].
figure courbeDe: f de: -10 a: 10
```

Exercice 5.9

```
| figure f a|
figure := DrGeoFigure nouveau afficherAxes afficherGrille echelle: 50.
a := figure entier: 1 a: 5 @ -1 min: -10 max: 10 nom: 'a' afficherValeur: true.
f := [:x | a valeur / x].
figure courbeDe: f de: -10 a: 10
```

Exercice 5.10

Lorsque a est positif, la fonction homographique est décroissante.

Lorsque a est négatif, la fonction homographique est croissante.

Annexe E Liste des exemples

Exemple 2.1: Premier programme ... 6
Exemple 2.2: Figure avec les axes des abscisses et des ordonnées ... 6
Exemple 2.3: Des messages comme des perles sur un collier ... 7
Exemple 2.4: Changement d'échelle ... 7
Exemple 2.5: Bonjour tout le monde ... 7
Exemple 2.6: Figure avec un point ... 8
Exemple 2.7: Un vecteur, c'est un déplacement ... 9
Exemple 2.8: Droite passant par (0;0) et (1;1) ... 10
Exemple 2.9: Des messages en cascade ... 11
Exemple 2.10: Des messages en cascade ... 11
Exemple 2.11: Cascade avec toutes sortes de messages ... 11
Exemple 2.12: Une variable pour notre figure ... 12
Exemple 2.13: Une variable utilisée plusieurs fois ... 12
Exemple 2.14: Deux variables ... 13
Exemple 2.15: Trois variables ... 13
Exemple 2.16: Variable et attributs ... 14
Exemple 2.17: Commentaire ... 15
Exemple 2.18: Triangle facile ! ... 15
Exemple 2.19: Point au hasard ... 16
Exemple 2.20: Point au hasard, les négatifs aussi ! ... 16
Exemple 2.21: Point farceur ... 17
Exemple 2.22: Point farceur au dixième ... 17
Exemple 2.23: Point farceur sur diagonale ... 18
Exemple 2.24: Point farceur sur diagonale avec variable ... 19
Exemple 2.25: Boucle de 1 à 10 ... 20
Exemple 2.26: Boucle de 1 à 10 à demi-pas ... 21
Exemple 2.27: Une boucle sur des valeurs en vrac ... 21
Exemple 2.28: Abscisse pair ... 22
Exemple 3.1: Les nombres naturels de 0 à 100 ... 27
Exemple 3.2: Ordinateur dyscalculique ! ... 28
Exemple 3.3: Enfin juste avec les fractions ! ... 28
Exemple 3.4: Encore la dyscalculie ... 29
Exemple 3.5: Ordre de calculs ... 30
Exemple 3.6: Calculs fractionnaires ... 30
Exemple 3.7: Inverse d'une fraction ... 31
Exemple 3.8: Division euclidienne ... 31
Exemple 3.9: Tester un multiple ... 32
Exemple 3.10: Programme interactif avec multiple ... 32
Exemple 3.11: Diviseurs de 100 ... 33
Exemple 3.12: Diviseurs de 155 et 100 ... 33
Exemple 3.13: Bloc de code diviseurs communs ... 35
Exemple 3.14: Nombre premier ? ... 36
Exemple 4.1: Trois droites parallèles ... 38
Exemple 4.2: Deux droites perpendiculaires ... 39
Exemple 4.3: Droites paires ou impaires colorées ... 39
Exemple 4.4: Distance entre deux points ... 40
Exemple 4.5: Distance d'un point à une droite ... 41
Exemple 4.6: Un autre chemin ... 42
Exemple 4.7: Parallélogramme et parallèles ... 43
Exemple 4.8: Parallélogramme et côtés isométriques ... 44

Exemple 4.9: Triangle quelconque 50
Exemple 4.10: Triangle isocèle et cercle 51
Exemple 4.11: Angle et rotation 52
Exemple 4.12: Angles géométrique et orienté 55
Exemple 4.13: Angles correspondants 56
Exemple 4.14: Angles alternes-internes 57
Exemple 4.15: Triangle et angles 58
Exemple 4.16: Angles d'un quadrilatère convexe 60
Exemple 4.17: Image d'un triangle par une symétrie centrale 62
Exemple 4.18: Image d'un triangle par une symétrie axiale 62
Exemple 4.19: Image d'un triangle par une translation 63
Exemple 4.20: Images d'un triangle par deux rotations 63
Exemple 4.21: Images réduites d'un triangle par une homothétie 65
Exemple 4.22: Images agrandies d'un triangle par une homothétie 65
Exemple 4.23: Symétrique d'un carré et d'un cercle inscrit 66
Exemple 4.24: Symétrique d'un groupe d'objets 66
Exemple 4.25: Triangles à gogo 68
Exemple 4.26: Spirales à gogo 69
Exemple 4.27: Compléter le motif élémentaire 73
Exemple 5.1: Fonction linéaire de pente 3 76
Exemple 5.2: Fonction linéaire à pente un nombre entier variable 77

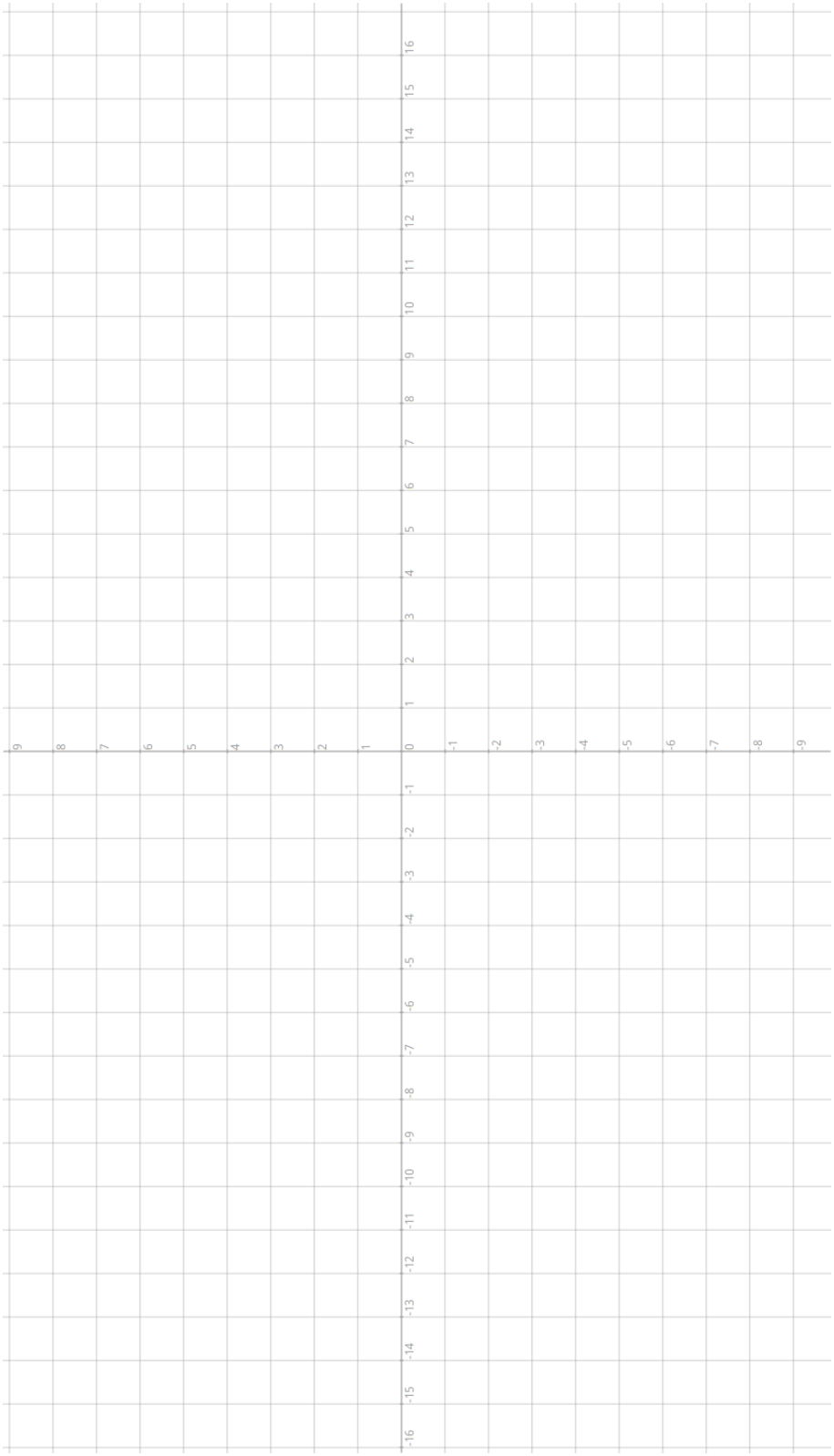
Annexe F Liste des figures

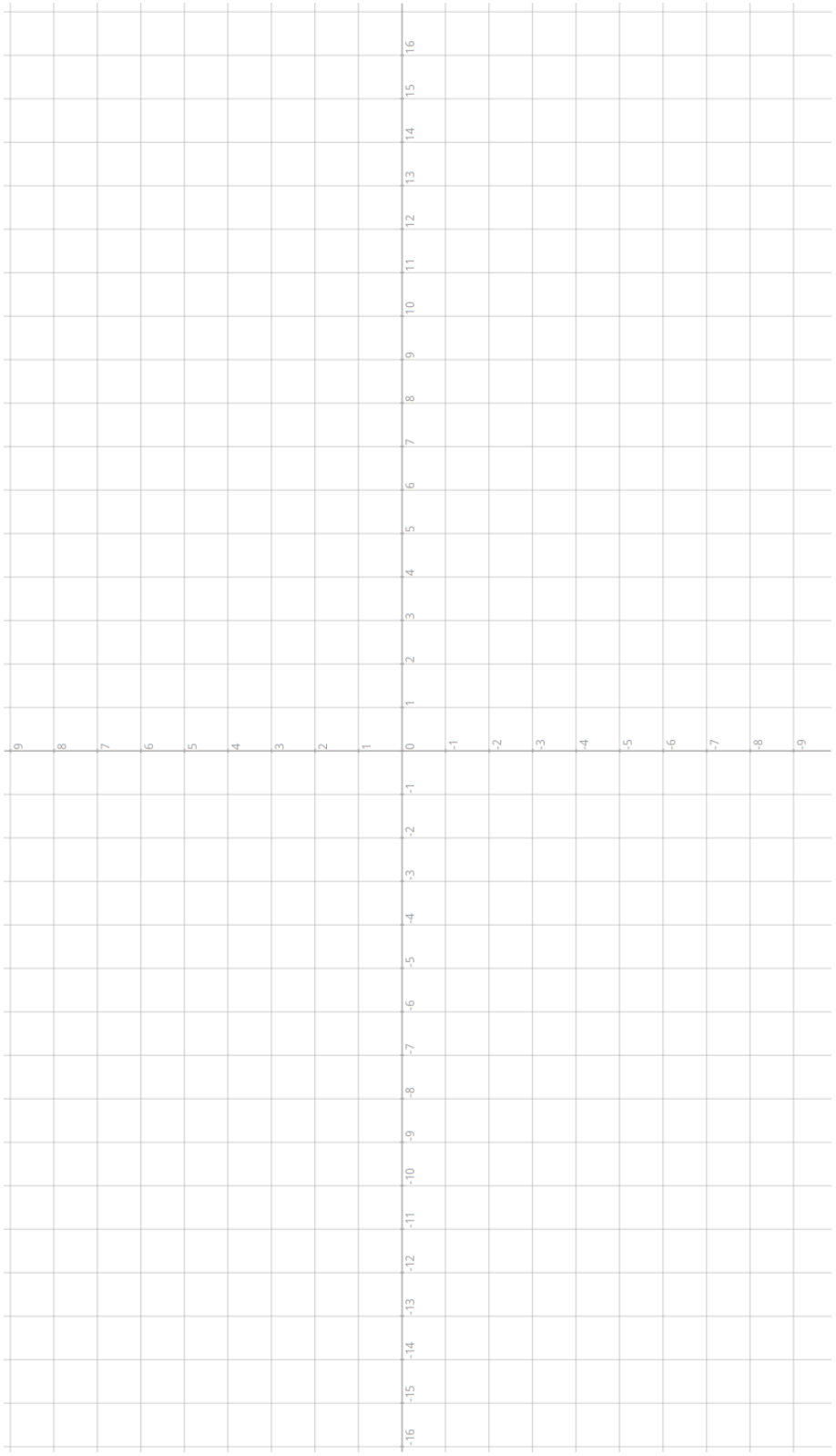
Figure 2.1: Notre “Espace de travail” ou “Playground” avec un code source d’une ligne ! 5
Figure 2.2: Figure Dr.Geo avec le point (0;1) 8
Figure 2.3: Attributs d’un segment 14
Figure 2.4: Point sur diagonale 18
Figure 4.1: Les nombres premiers en bleu, parmi les autres nombres pairs et impairs non premiers, respectivement rouge et orange 40
Figure 4.2: Distance d’un point à une droite 41
Figure 4.3: Distance entre deux droites parallèles 41
Figure 4.4: Que de chemins ! 43
Figure 4.5: Côtés opposés du parallélogramme 44
Figure 4.6: Parallélogramme et cercles 45
Figure 4.7: Diagonales du parallélogramme 45
Figure 4.8: Losange 46
Figure 4.9: Rectangle 47
Figure 4.10: Carré 49
Figure 4.11: Triangle isocèle et côtés isométriques 50
Figure 4.12: Angles géométrique (bleu) et orienté (marron avec flèche) 56
Figure 4.13: Angles correspondants portés par deux droites parallèles 56
Figure 4.14: Angles alternes-internes portés par deux droites parallèles 57
Figure 4.15: Triangle et angles alternes-internes 58
Figure 4.16: Triangle et angles correspondants 59
Figure 4.17: Polygones convexe et non convexe 59
Figure 4.18: Somme des angles d’un quadrilatère convexe 60
Figure 4.19: Somme des angles d’un quadrilatère non convexe, non croisé 61
Figure 4.20: Somme des angles d’un quadrilatère non convexe, et **croisé** 61
Figure 4.21: Deux rotations avec des angles de 70° et -70° 64
Figure 4.22: Spirale 67
Figure 4.23: Smiley 67
Figure 4.24: Frise de spirales 69
Figure 4.25: Une frise par une translation 70
Figure 4.26: Frise vue dans une rue de Rhodes (galets de plage) 70
Figure 4.27: Mise en évidence du motif élémentaire d’une frise 71
Figure 4.28: Encore une spirale 72
Figure 4.29: Une frise par translation mais... 72
Figure 4.30: ...aussi une symétrie axiale 73
Figure 4.31: Frise et symétrie 74
Figure 4.32: Frise et symétries 75
Figure 5.1: Représentation graphique d’une fonction linéaire de pente 3 76
Figure 5.2: Représentation graphique d’une fonction linéaire de pente un entier variable 77
Figure 6.1: Triangle de Sierpinski 99
Figure 7.1: Votre espace de travail avec le code source collé et son menu contextuel 101
Figure 7.2: Résultat de l’exécution du code source : intégrale de la fonction sur [-1 ; 1] 101
Figure 7.3: Exécution du code, inspection de l’objet `figure` et exécution d’instructions supplémentaires depuis l’inspecteur 102
Figure 7.4: Courbe interactive de Sierpinski 103
Figure 7.5: Le profileur Dr.Geo à l’issue de la construction de la courbe de Sierpinski 104
Figure 7.6: Le débogueur Dr.Geo 104
Figure 7.7: L’inspecteur sur la variable *sommets* 106
Figure 7.8: Inspecteur et codes source des figures 106
Figure 7.9: Spirale de Galilée 107

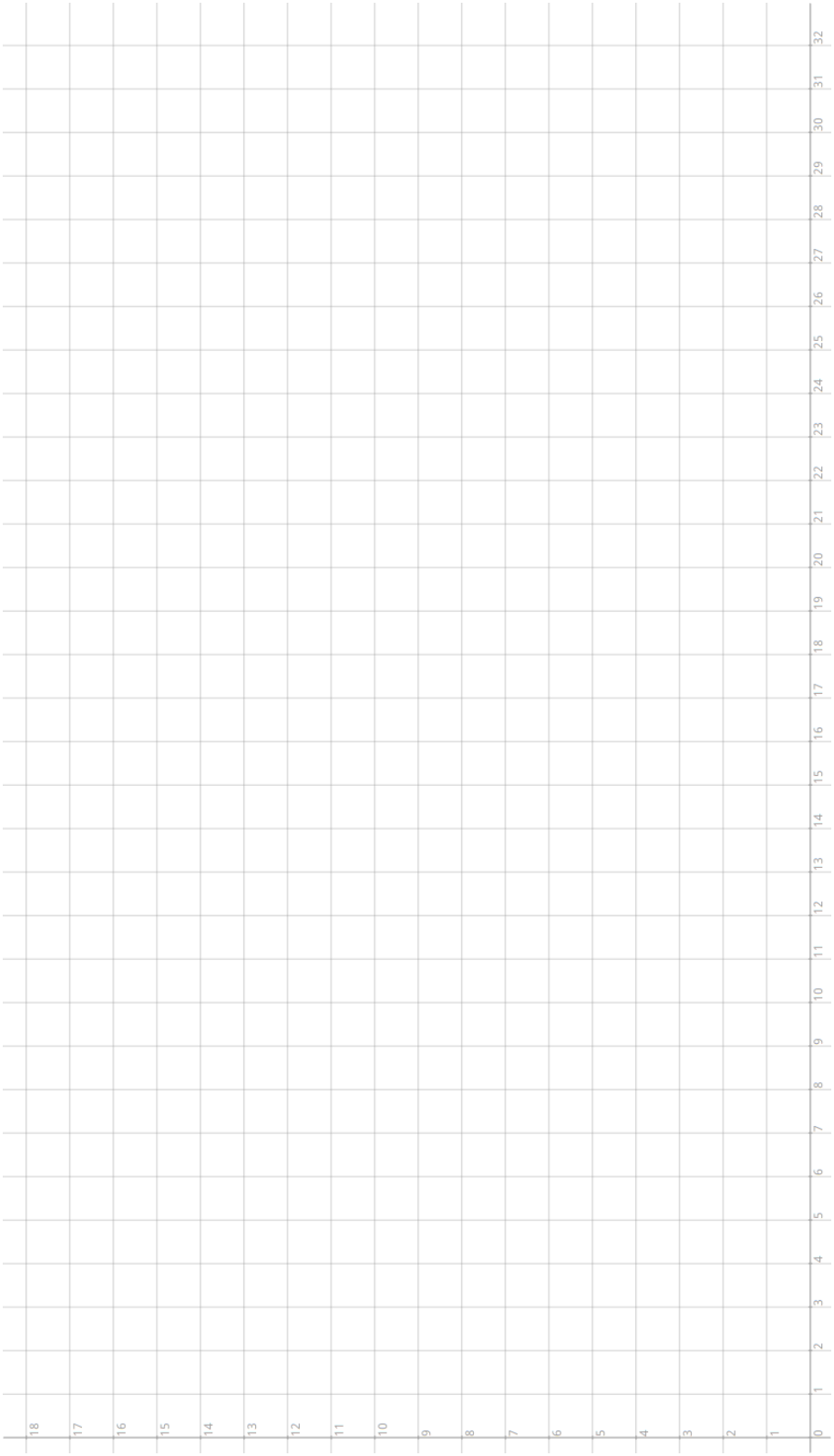
Figure 7.10: Réponse du Chercheur sur un motif de calcul et sa réponse souhaitée 107
Figure 7.11: Spotter l'outil de recherche 109

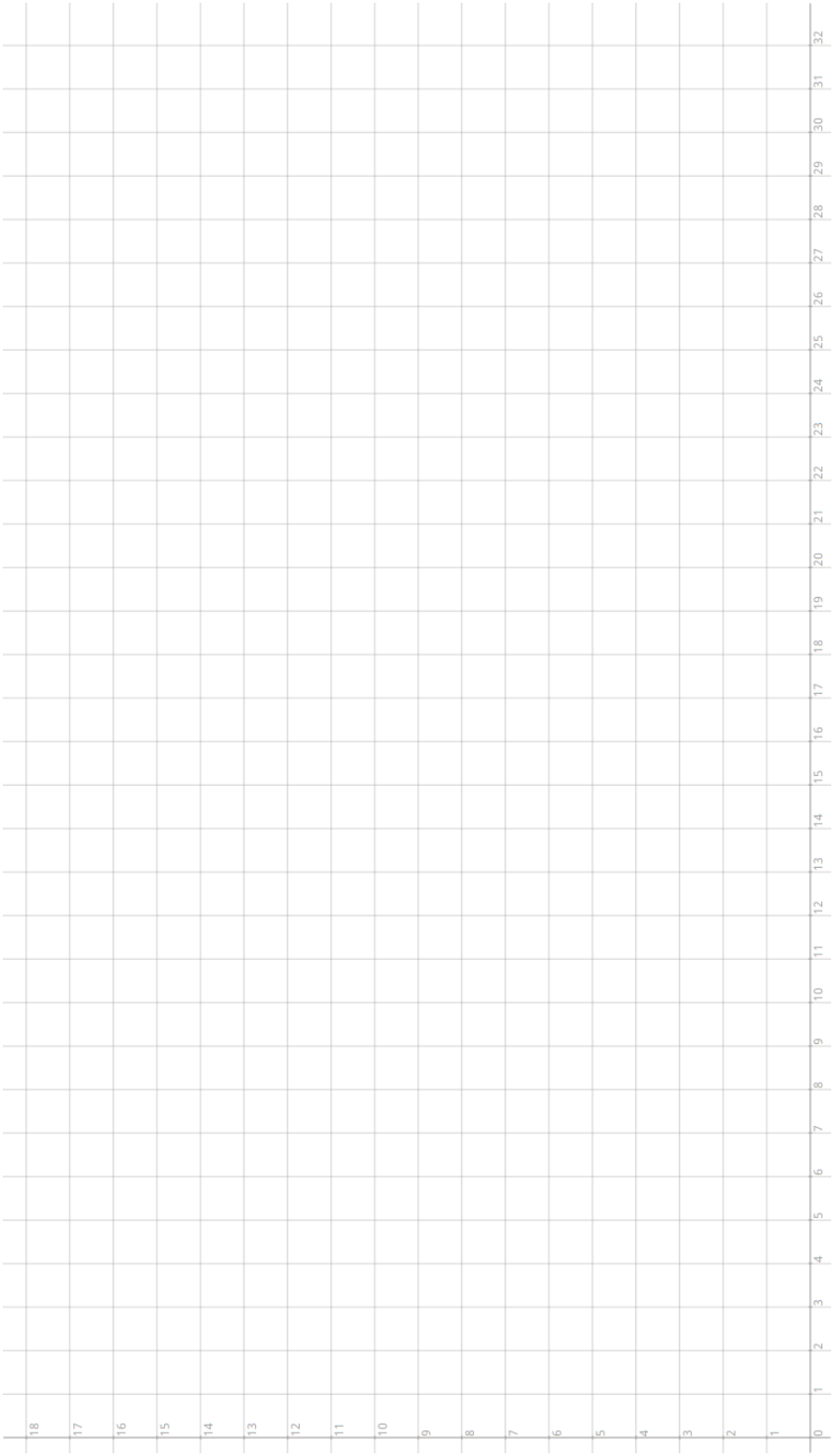
Annexe G Aides

Cette annexe propose des documents à utiliser lors de la résolution de certains exercices.









Annexe H Index conceptuel

A

angle ... 51
angle (méthodes) ... 92
angle, alterne-interne ... 57
angle, correspondant ... 56
angle, géométrique ... 55
angle, géométrique (méthodes) ... 93
angle, orienté ... 55
angle, orienté (méthodes) ... 93
animation (figure programmée) ... 98
arc de cercle (méthodes) ... 89
aspect ... 13
attributs objets (méthodes) ... 94
axes des abscisses et ordonnées ... 6

B

bissectrice ... 54
bloc de code ... 16
bloc de code, paramètre ... 20
bloc de code, valeur de retour ... 18
boucle ... 20
boucle, a:faire: ... 20
boucle, a:par:faire: ... 20
boucle, foisRepete: ... 68, 70

C

cacher ... 43
calcul, fraction ... 30
calcul, priorité ... 29
carré ... 48
cercle (méthodes) ... 88
cercle, circonscrit ... 53
cercle, inscrit ... 54
cercle, rayon ... 10
cercle, segment ... 44
chaîne de caractères ... 8
chercher une méthode ... 107
collection ... 15
collection, ajouterTout: ... 73
collection, choisir: ... 33
collection, collecter: ... 68
collection, commeCollectionOrdonnee ... 73
collection, faire: ... 21
collection, nombres ... 27
commentaire ... 15
convexe (quadrilatère) ... 59
coordonnées ... 8
couleur ... 14
croisé (quadrilatère) ... 61

D

déboguer une figure codée ... 104
demi-droite ... 10
demi-droite (méthodes) ... 88
distance, droites parallèles ... 41
distance, entre points ... 40
distance, point à droite ... 40
diviseur ... 31, 33
diviseur, commun ... 33
diviseur, pgdc ... 35
droite ... 10
droite (méthodes) ... 87
droite, parallèle ... 38
droite, perpendiculaire ... 38, 47

E

échelle ... 7
ensemble ... 27
équation (méthodes) ... 93
espace de travail ... 5, 100
espace de travail, coller le code d'une figure ... 100
espace de travail, compiler le code d'une figure ... 101
espace de travail, inspecter une figure ... 101
espace de travail, nommer ... 102
espace de travail, partager du code sur Internet ... 102
espace de travail, sauver/charger du code ... 102
exemples (figure programmée) ... 97

F

figure programmée ... 83
figure programmée, exécuter ... 83
figure programmée, exemples ... 83
figure programmée, messages divers ... 84
figure programmée, nouvelle ... 6
fonction, affine ... 78
fonction, courbe ... 76
fonction, linéaire ... 76
fonction, quadratique ... 78
frise ... 69

G

grille ... 6

H

hasard ... 16
hauteur ... 55
homothétie ... 64

I

inspecter un objet ... 105
inverse ... 31

L

lieu géométrique (méthodes) ... 91
ligne, épaisseur ... 14
ligne, style ... 14
losange ... 46

M

médiane ... 55
médiatrice ... 51, 53
méthodes complémentaires ... 96
message ... 5
message, binaire ... 8
message, cascade ... 10
message, mot clé ... 7
message, priorités ... 9
message, types ... 9
message, unaire ... 6
milieu, deux points ... 45
milieu, segment ... 13
montrer ... 38
multiple ... 31

N

nom ... 14
nombre, décimal ... 22, 27
nombre, entier ... 22, 27
nombre, fraction ... 28
nombre, impair ... 22
nombre, naturel ... 31
nombre, pair ... 22
nombre, positif ... 22
nombre, premier ... 22, 35
nombre, rationnel ... 28

P

parallélogramme ... 43
pente d'une droite ... 78
phrase ... 8
playground ... 5
plein écran (figure) ... 6
point (méthodes) ... 85
point, coordonnées ... 8
point, forme ... 14
point, intersection ... 13, 41
point, sur ligne ... 42, 47
point, taille ... 14
polygone ... 15
polygone (méthodes) ... 89
profileur ... 103
puissance ... 78

Q

quadrilatère, somme des angles ... 59

R

réglette, entier ... 76
rectangle ... 47
rotation ... 51, 63

S

segment ... 10
segment (méthodes) ... 88
segment, codage ... 14
spotter ... 103, 108
style (méthodes) ... 94
symétrie, axiale ... 62
symétrie, centrale ... 46, 61
syntaxe ... 5

T

test ... 22
test, siVrai ... 22
test, siVrai siFaux ... 31, 39
texte (méthodes) ... 93
transformations (méthodes) ... 90
transformations géométriques ... 61
translation ... 62
triangle ... 50
triangle de Sierpinski ... 98
triangle, équilatéral ... 52
triangle, isocèle ... 50
triangle, rectangle ... 53
triangle, rectangle isocèle ... 53
triangle, somme des angles ... 57

V

valeurs (méthodes) ... 91
variable, affectation ... 12
variable, déclaration ... 12
vecteur (méthodes) ... 91
vecteur, coordonnées ... 9

Z

zoom ... 7

Annexe I Index des méthodes Dr.Geo

A

abscisseDe: sur DrGeoFigure 92
actualiser sur DrGeoFigure 85
afficherGrille sur DrGeoFigure 85
angleCentre:de:a sur DrGeoFigure 93
angleGeometriqueCentre:de:a sur DrGeoFigure ... 93
angleVecteur:et: sur DrGeoFigure 93
arcCentre:de:a: sur DrGeoFigure 89
arcDe:a:passantPar: sur DrGeoFigure 89
autreIntersectionDe:et: sur DrGeoFigure 86

B

bissectriceDe: sur DrGeoFigure 88
bissectriceSommet:cote1:cote2
sur DrGeoFigure 88
bloquer sur WrpItem 96

C

cacher sur WrpItem 94
carre sur WrpPoint 96
centrerVueEn: sur DrGeoFigure 85
cercleCentre:passantPar: sur DrGeoFigure 89
cercleCentre:rayon: sur DrGeoFigure 89
cercleCentre:segment: sur DrGeoFigure 89
coordonnees sur WrpPoint 92, 94
couleur: sur WrpItem 94
couleurFond: sur WrpText 94
courbeDe:de:a: sur DrGeoFigure 97
croix sur WrpPoint 96

D

debloquer sur WrpItem 96
decimal:a:min:max sur DrGeoFigure 97
decimal:a:min:max:nom: sur DrGeoFigure 97
decimal:a:min:max:nom:affciherValeur:■
sur DrGeoFigure 97
demiDroiteOrigine:passantPar:
sur DrGeoFigure 88
deplacerA: sur WrpItem 96
distanceDe:a: sur DrGeoFigure 92
DrGeoFigure> sur DrGeoFigure 85
droitePassantPar:et: sur DrGeoFigure 87

E

echelle: sur DrGeoFigure 85
eniter:a:min:max:nom: sur DrGeoFigure 97
entier:a:min:max sur DrGeoFigure 97
entier:a:min:max:nom:affciherValeur:■
sur DrGeoFigure 97
epais sur WrpCurve 95
equationDe: sur DrGeoFigure 93
exporterVersImage: sur DrGeoFigure 97

F

faire: sur DrGeoFigure 85
fin sur WrpCurve 95
flecheDebut sur wrpFinitCurve 95
flecheFin sur wrpFinitCurve 95
fleches sur wrpFinitCurve 95

H

homothetieDe:parCentre:etFacteur:
sur DrGeoFigure 90

I

intersectionDe:et: sur DrGeoFigure 86

L

large sur WrpPoint 96
lieuDe:lorsqueBouge: sur DrGeoFigure 91
longueurDe: sur DrGeoFigure 92

M

marquerAucun sur wrpSegment 96
marquerAvecCercle sur wrpSegment 95
marquerAvecDisque sur wrpSegment 95
marquerAvecDoubleTrait sur wrpSegment 96
marquerAvecSimpleTrait sur wrpSegment 95
marquerAvecTripleTrait sur wrpSegment 96
mediatriceDe: sur DrGeoFigure 88
mediatriceDe:a: sur DrGeoFigure 88
milieuDe: sur DrGeoFigure 86
milieuDe:et: sur DrGeoFigure 86
montrer sur WrpItem 94

N

nommer: sur WrpItem 94
normal sur WrpCurve 95

O

ordonneeDe: sur DrGeoFigure 92

P

paralleleA:passantPar: sur DrGeoFigure 87
penteDe: sur DrGeoFigure 92
perpendiculaireA:passantPar: sur DrGeoFigure .. 87
plein sur WrpCurve 95
pleinEcran sur DrGeoFigure 85
point: sur DrGeoFigure 86
point:parent: sur DrGeoFigure 87
point:parents sur DrGeoFigure 87
pointille sur WrpCurve 95
pointSurLigne:a: sur DrGeoFigure 86
pointX:Y: sur DrGeoFigure 86
polygone: sur DrGeoFigure 90
polygoneRegulierCentre:sommet:cotes:
sur DrGeoFigure 90

R

rond sur WrpPoint 96
rotationDe:parCentre:etAangle:
sur DrGeoFigure 90

S

segmentDe:a: sur DrGeoFigure 88
small sur WrpPoint 96
supprimer sur DrGeoFigure 85
symetriqueDe:selonAxe: sur DrGeoFigure 91
symetriqueDe:selonCentre sur DrGeoFigure 91

T

texte: sur DrGeoFigure 93
texte: sur WrpText 94
texte:a sur DrGeoFigure 93
textPositionDelta: sur MathItemCostume 94
tiret sur WrpCurve 95
translationDe:parVecteur: sur DrGeoFigure 91

V

valeur: sur WrpValue 92
valeurLibre: sur DrGeoFigure 92
vecteur: sur DrGeoFigure 91
vecteurOrigine:extremite: sur DrGeoFigure 91

X

x sur WrpPoint 94

Y

y sur WrpPoint 94