

# Le langage Racket

---

- Un langage de programmation fonctionnelle
- 08/12/2021

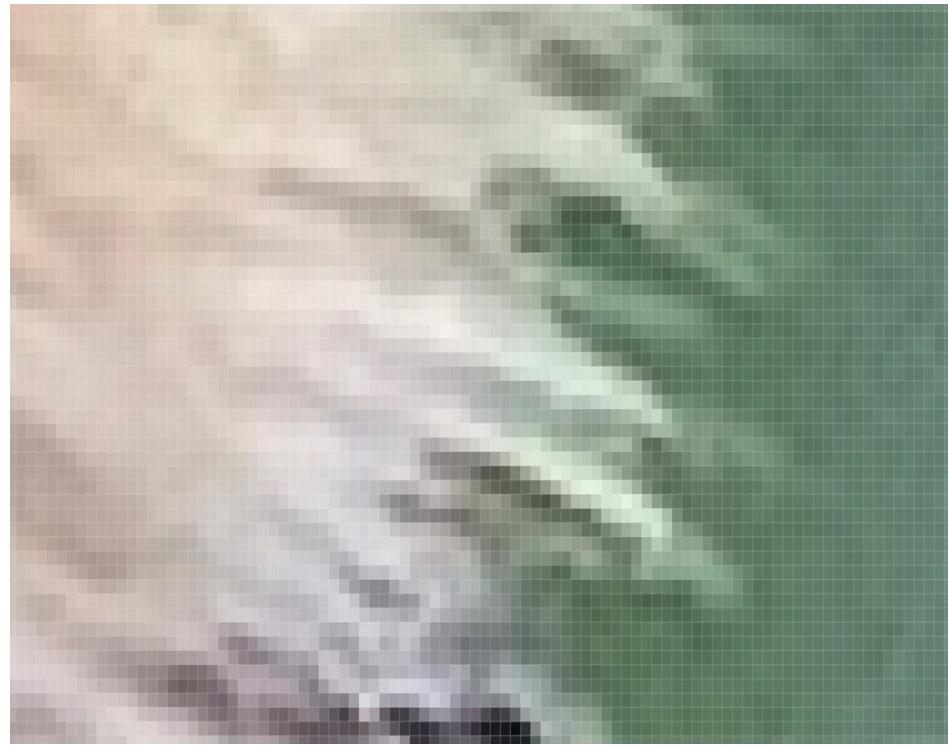
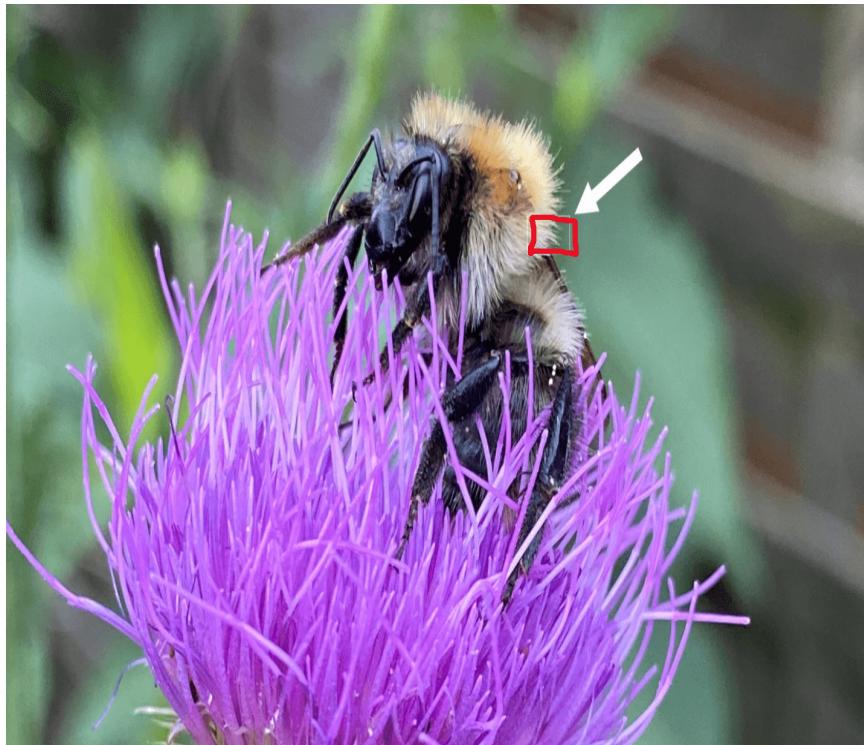
# Plant du cours

1. Dessiner des images pixel par pixel
2. Dessiner des fractales
3. Exercices

# 1. Dessiner des images pixel par pixel

**Rappel:**

Pixel: Picture elements



# 1. Dessiner des images pixel par pixel

**Rappel:**

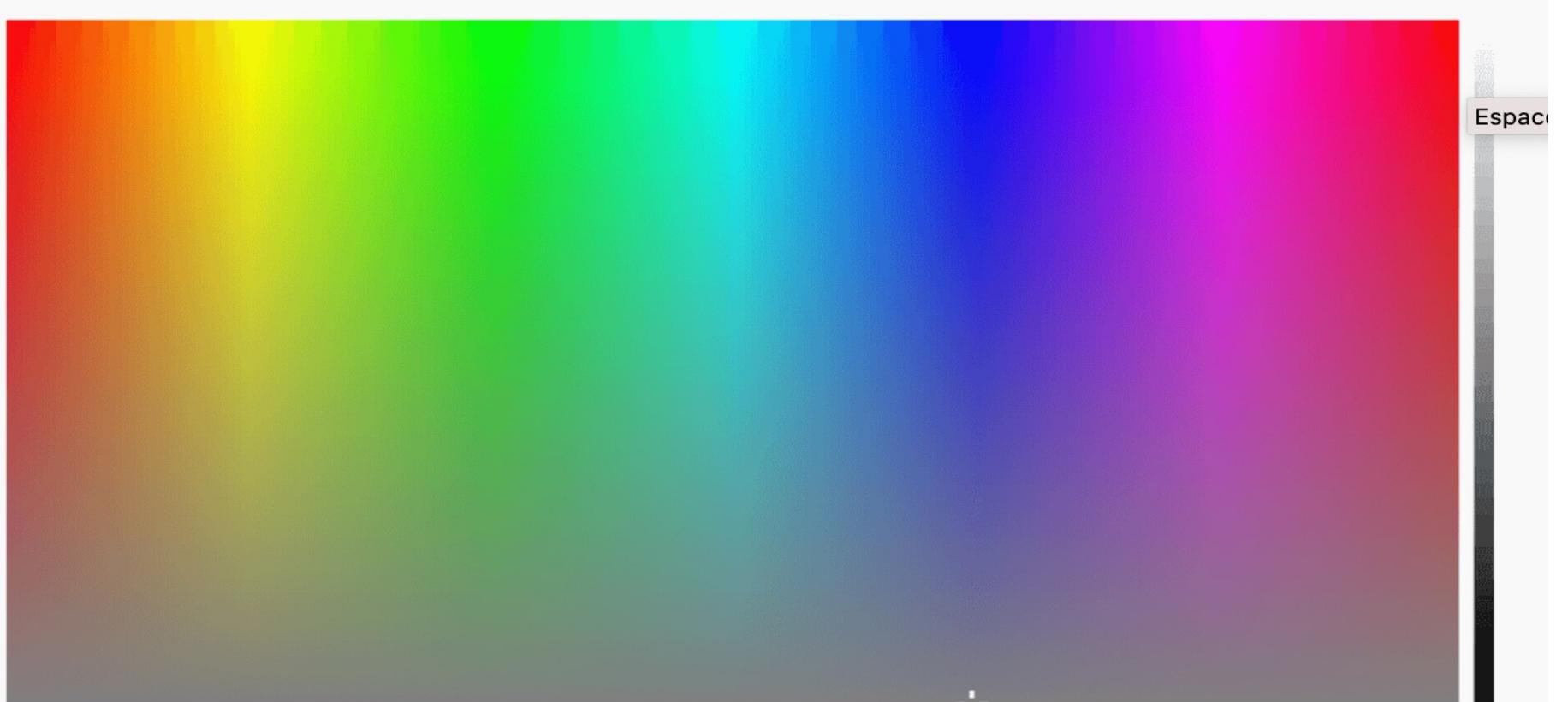
Pixel: Picture elements



# 1. Dessiner des images pixel par pixel

**Rappel:**

Pixel: Picture elements



# 1. Dessiner des images pixel par pixel

Le package **2htdp/image** permet de manipuler des images sous forme matricielle.

La fonction **bitmap** permet de charger une image png ou jpg;

```
#lang racket  
  
(require racket/trace)  
  
(require 2htdp/image)  
  
; Fonction bitmap permet de charger une image png ou jpg  
  
(bitmap "nuage.png")
```

# 1. Dessiner des images pixel par pixel

```
1 (require 2htdp/image)
2 (bitmap "/home/n/nuages.png")
```



# 1. Dessiner des images pixel par pixel

**1. image-width:** permet de connaître la largeur d'une image.

Exemple:

(image-width (bitmap "nuage.png"))

# 1. Dessiner des images pixel par pixel

**1. image-width:** permet de connaître la largeur d'une image.

**2. image-height:** permet de connaître la hauteur d'une image.

**Exemple:**

**(image-height (bitmap "nuage.png"))**

# 1. Dessiner des images pixel par pixel

**1. image-width:** permet de connaître la largeur d'une image.

**2. image-height:** permet de connaître la hauteur d'une image.

**3. image->color-list:** retourne la liste des pixels d'une image.

**Exemple: (image->color-list (rectangle 2 2 "solid" "black"))**

# 1. Dessiner des images pixel par pixel

**1. image-width:** permet de connaître la largeur d'une image.

**2. image-height:** permet de connaître la hauteur d'une image.

**3. image->color-list:** retourne la liste des pixels d'une image.

**4. color-list->image:** permet de définir une image à partir d'une liste de pixels.

Remarque: Pour chaque élément de la liste de type color-list, on obtient les valeurs r, g, b à l'aide des fonctions:

**color-red**

**color-green**

**color-blue**

**color-alpha**

Exemple: Ecrire une fonction **image-color->gray** qui passe l'image « nuage.png » vue précédemment en niveaux de gris.

Considérant que passer une image en niveau de gris est équivalent à avoir sur chacune des composantes la moyenne des composantes rgb de l'image de couleur.

Exemple: Ecrire une fonction **image-color->gray** qui passe l'image « nuage. png » vue précédemment en niveaux de gris.

Considérant que passer une image en niveau de gris est équivalent à avoir sur chacune des composantes la moyenne des composantes rgb de l'image de couleur.

|                       |                       |                       |                       |                       |  |
|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|--|
| $\frac{r + g + b}{3}$ |  |
| $\frac{r + g + b}{3}$ | $\frac{r + g + b}{3}$ | $\frac{r + g + b}{3}$ | ...                   |                       |  |
|                       |                       |                       |                       |                       |  |
|                       |                       |                       |                       |                       |  |
|                       |                       |                       |                       |                       |  |

Exemple: Ecrire une fonction **image-color->gray** qui passe l'image « nuage.png » vue précédemment en niveaux de gris.

Considérant que passer une image en niveau de gris est équivalent à avoir sur chacune des composantes la moyenne des composantes rgb de l'image de couleur.

```
1 (require 2htdp/image)
2 (define (image-color->gray img)
3   (define (f L)
4     (if (empty? L) empty
5         (let* ([pix (first L)]
6               [sum_pix (+ (color-red pix)
7                           (color-green pix) (color-blue pix))])
8             [gpix (round (/ sum_pix 3))])
9               (cons (color gpix gpix gpix (color-alpha pix)) (f (rest L))))
10            )))
11 (color-list->bitmap
12   (f (image->color-list img)
13     (image-width img) (image-height img)
14   ))
15 (image-color->gray (bitmap "/home/n/nuages.png"))
```

```
1 (require 2htdp/image)
2 (define (image-color->gray img)
3   (define (f L)
4     (if (empty? L) empty
5         (let* ([pix (first L)]
6               [sum_pix (+ (color-red pix)
7                           (color-green pix)(color-blue pix))])
8             [gpix (round (/ sum_pix 3))]))
9         (cons (color gpix gpix gpix (color-alpha pix)) (f (rest L)))
10        )))
11 (color-list->bitmap
12   (f (image->color-list img))
13   (image-width img) (image-height img)
14  ))
15 (image-color->gray (bitmap "/home/n/nuages.png"))
```

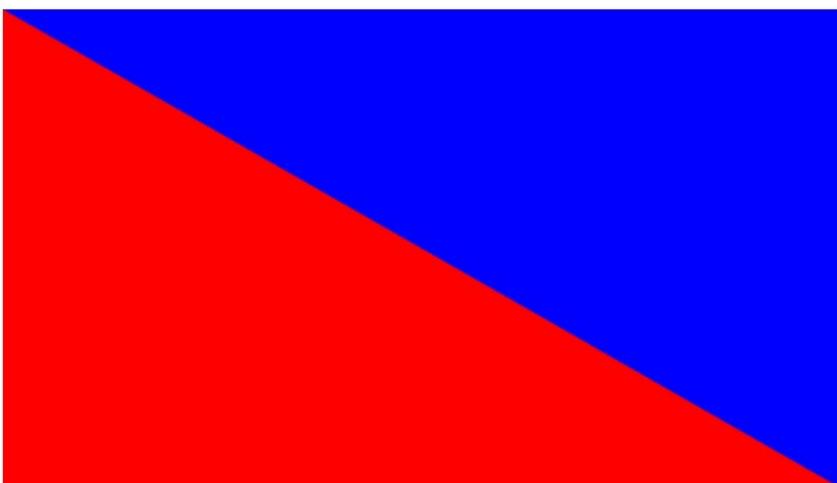


```
1 (require 2htdp/image)
2 (define (mk-image-line y fun width)
3   (define (f x)
4     (if (= x width) empty
5         (cons (fun x y) (f (add1 x))))
6     ))
7   (f 0)
8 )
9 (define (mk-image fun width height)
10  (define (f y L)
11    (if (= y height) (color-list->bitmap L width height)
12        (f (add1 y) (append L (mk-image-line y fun width))))
13    ))
14  (f 0 empty)
15 )
16 (define (rgb-fun x y)
17  (if (= y 0) (color 0 0 255)
18      (if (< (/ x y) 2)
19          (color 255 0 0)
20          (color 0 0 255)
21      )))
22 (mk-image rgb-fun 600 300)
```

XX

|   |       |       |       |       |       |  |
|---|-------|-------|-------|-------|-------|--|
| 0 | 0     | 1     | 2     | 3     | 4     |  |
| 0 | R,G,B | R,G,B | R,G,B | R,G,B | R,G,B |  |
| 1 | R,G,B | ...   |       |       |       |  |
| 2 |       |       |       |       |       |  |
| 3 |       |       |       |       |       |  |
| 4 |       |       |       |       |       |  |

```
1 (require 2htdp/image)
2 (define (mk-image-line y fun width)
3   (define (f x)
4     (if (= x width) empty
5         (cons (fun x y) (f (add1 x))))
6     ))
7 (f 0)
8 )
9 (define (mk-image fun width height)
10  (define (f y L)
11    (if (= y height) (color-list->bitmap L width height)
12        (f (add1 y) (append L (mk-image-line y fun width))))
13    ))
14 (f 0 empty)
15 )
16 (define (rgb-fun x y)
17   (if (= y 0) (color 0 0 255)
18       (if (< (/ x y) 2)
19           (color 255 0 0)
20           (color 0 0 255)
21       )))
22 (mk-image rgb-fun 600 300)
```



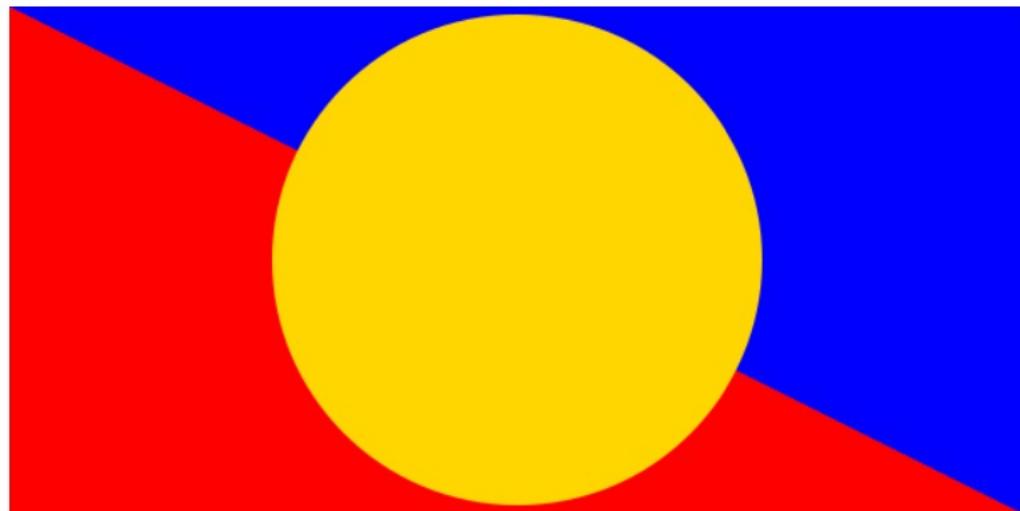
# 1. Dessiner des images pixel par pixel

## Remarques:

- Il est possible de charger une image à partir d'un URL avec la fonction bitmap/url
- On peut utiliser le prédicat **image?** pour savoir si un élément est une image
- On peut utiliser le prédicat **image-color?** Pour savoir si un élément est un pixel.
- On pourra superposer une image sur le dessin vu précédemment.

1

```
(overlay (circle 145 "solid" "gold") (mk-image rgb-fun 600 300))
```



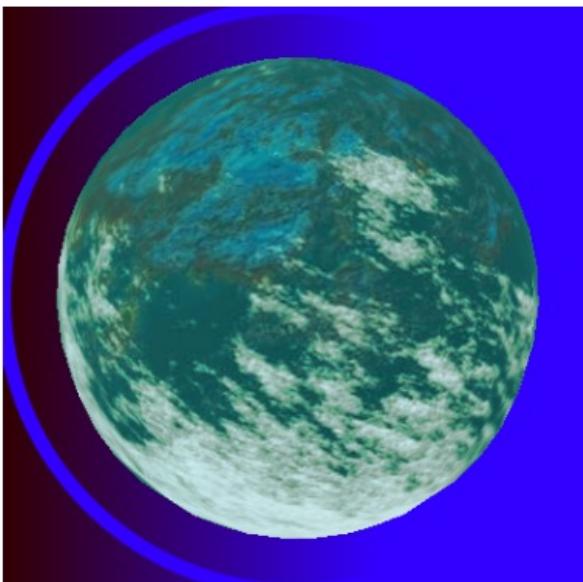
# 1. Dessiner des images pixel par pixel

## Remarques:

- Il est possible de charger une image à partir d'un URL avec la fonction bitmap/url
- On peut utiliser le prédicat **image?** pour savoir si un élément est une image
- On peut utiliser le prédicat **image-color?** Pour savoir si un élément est un pixel.
- On pourra superposer une image sur le dessin vu précédemment.
- On pourra superposer un dégradé du noir vers le bleu horizontalement, des primitives de dessin et une image de la planète Namek.

```
1 (define (clamp pix-val) (max 0 (min 255 pix-val)))
2 (define (grad1 x y) (clamp x))
3 (define (rgb-fun x y) (color 0 0 (grad1 x y)))
4 (underlay
5   (underlay
6     (mk-image rgb-fun 400 400)
7     (circle 200 "outline"
8       (make-pen (color 50 0 255) 10 "solid" "round" "round")
9     )))
10  (bitmap "/home/n/namek.png")
11 )
```

```
1 (define (clamp pix-val) (max 0 (min 255 pix-val)))
2 (define (grad1 x y) (clamp x))
3 (define (rgb-fun x y) (color 0 0 (grad1 x y)))
4 (underlay
5   (underlay
6     (mk-image rgb-fun 400 400)
7     (circle 200 "outline"
8       (make-pen (color 50 0 255) 10 "solid" "round" "round")
9     )))
10  (bitmap "/home/n/namek.png"))
11 )
```

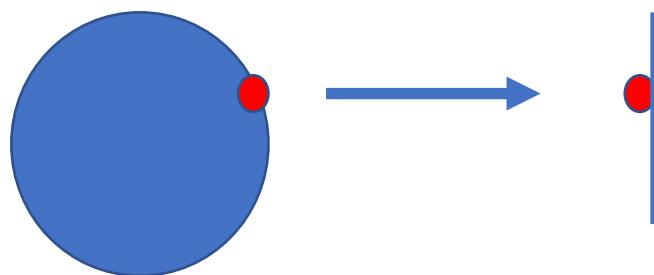


## 2. Dessiner des fractales

**Rappel:** une fractale est une figure géométrique qui possède la particularité d'avoir des détails quel que soit l'échelle sur laquelle on la regarde.

### Exemples:

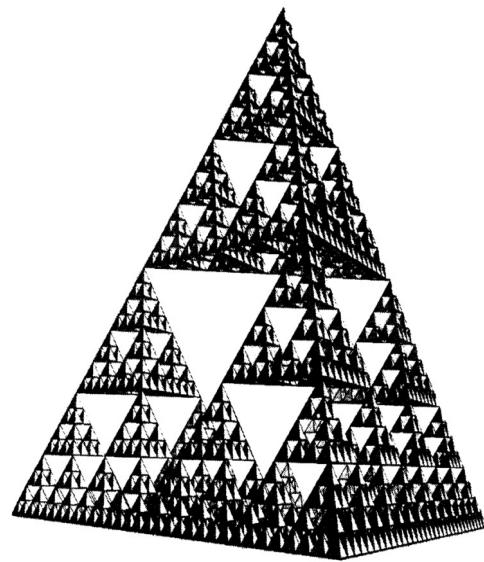
- Un cercle n'est pas une fractale!



## 2. Dessiner des fractales

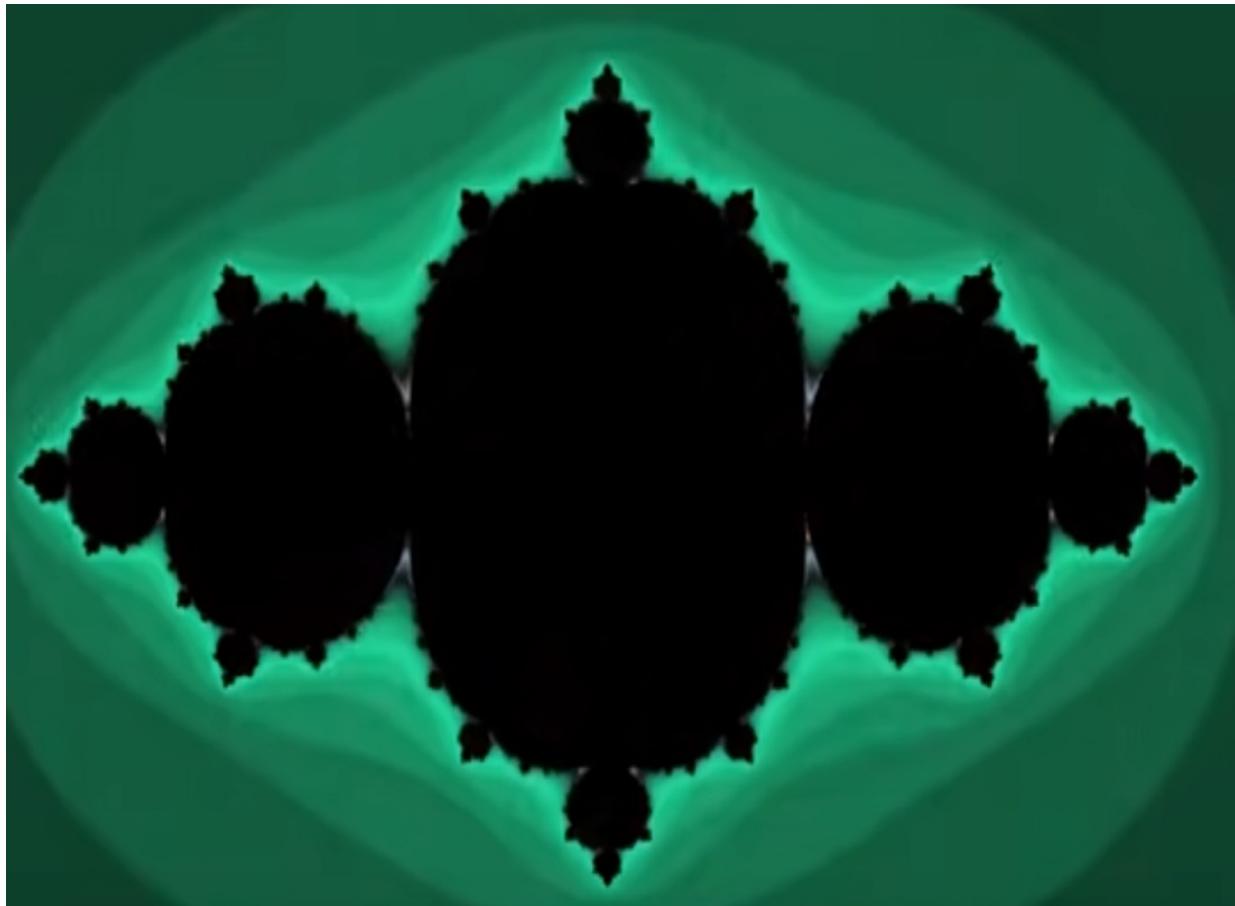
**Rappel:** une fractale est une figure géométrique qui possède la particularité d'avoir des détails quel que soit l'échelle sur laquelle on la regarde.

**Exemples:**



## 2. Dessiner des fractales

### L'ensemble de Mandelbrot



## 2. Dessiner des fractales

### L'ensemble de Mandelbrot



## 2. Dessiner des fractales

### L'ensemble de Mandelbrot



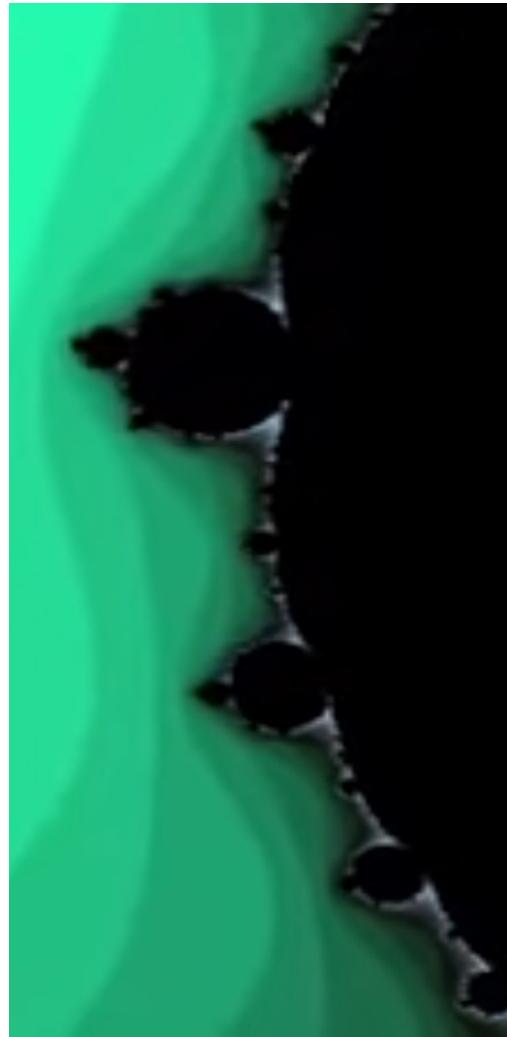
## 2. Dessiner des fractales

### L'ensemble de Mandelbrot



## 2. Dessiner des fractales

### L'ensemble de Mandelbrot



## 2. Dessiner des fractales

### L'ensemble de Mandelbrot

#### L'algorithme:

Affecter à **maxite** une valeur seuil

Pour chaque point de coordonnées (x,y) du plan:

$$\text{définir } z_i = \frac{x-320}{160} + \frac{y-240}{140} i$$

définir  $z = z_i$

définir  $\text{ite}=0$

Tant Que le module de  $z_i$   $\leq \text{maxdist}$  et que  $\text{ite} \leq \text{maxite}$

Affecter à  $\text{ite}$  la valeur  $\text{ite}+1$

$$z_i = z_i^2 + z$$

Fin Tant Que

Si  $\text{ite} = \text{maxite}$  Alors

Colorier le pixel en vert

Sinon

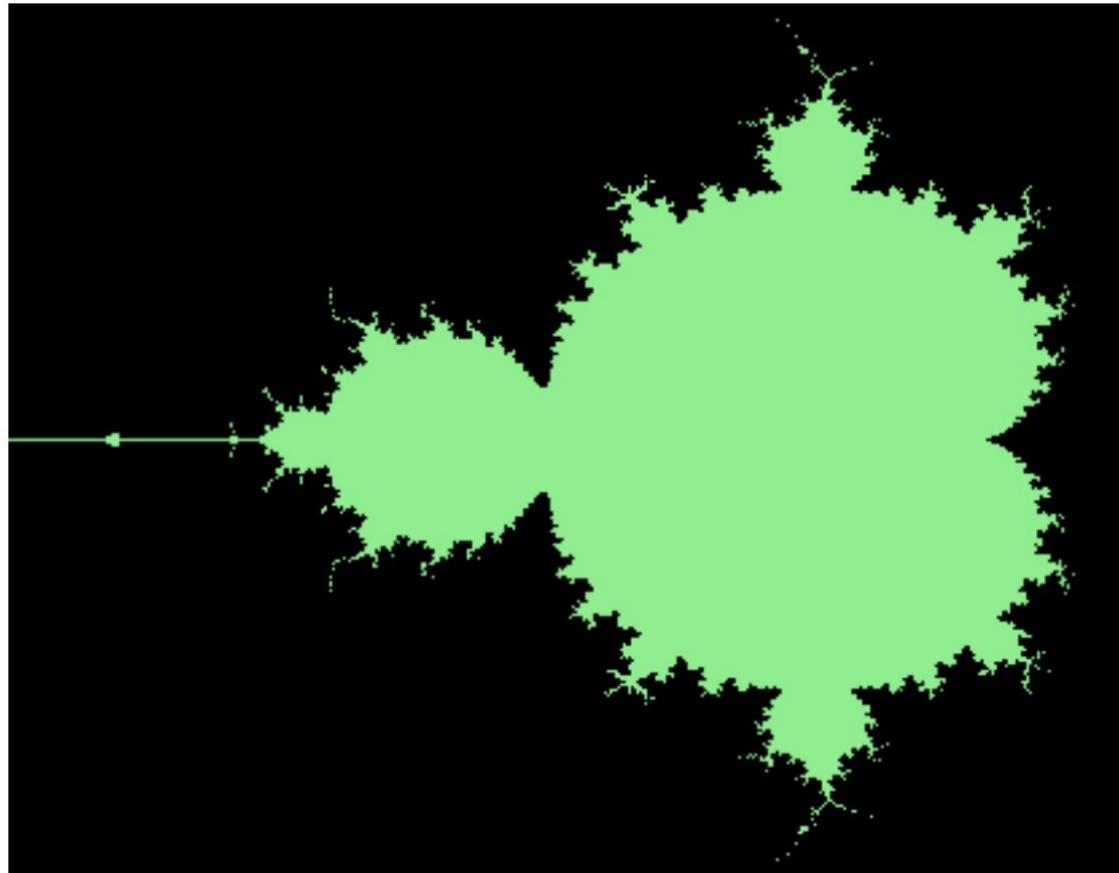
Colorier le pixel en noir

Fin Si

Fin Pour

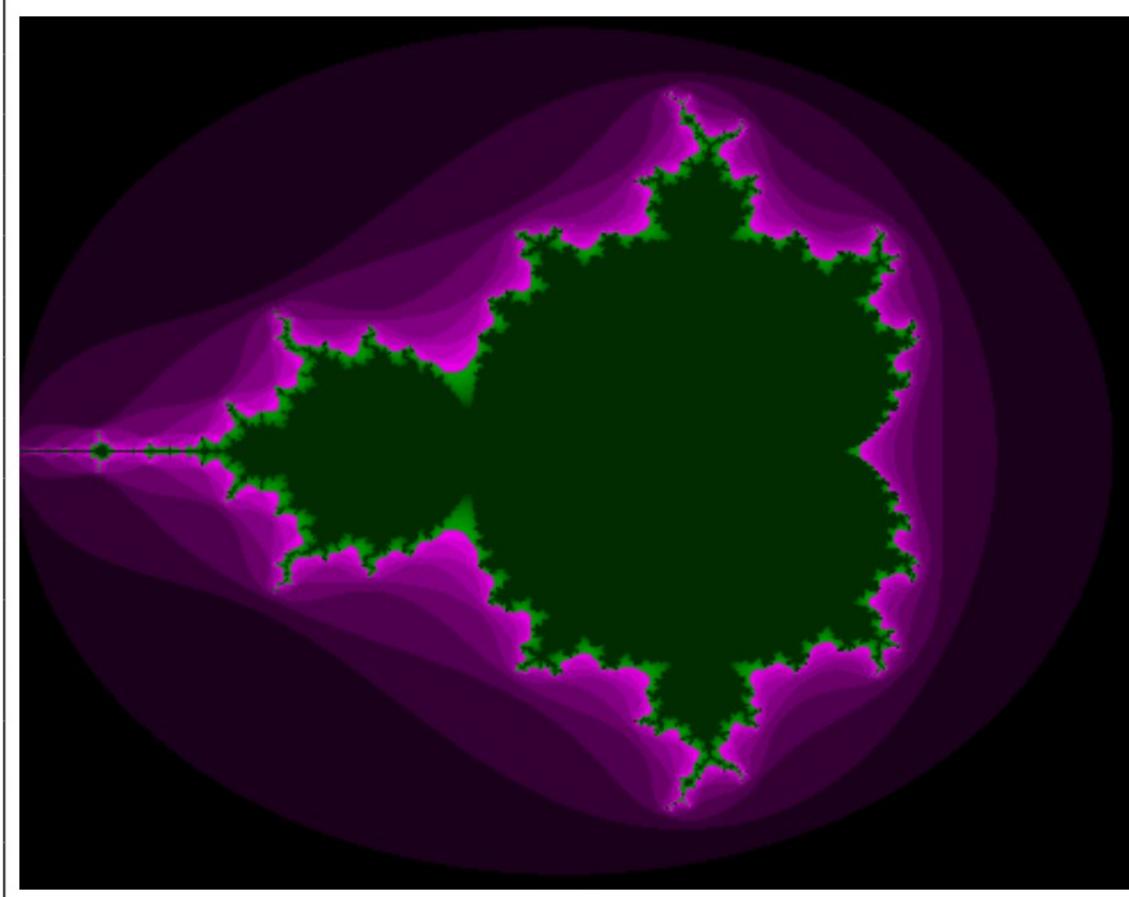


```
1 (define maxite 20)
2 (define maxdist 2.0)
3 (define (dist z)
4   (sqrt (+
5     (* (real-part z) (real-part z))
6     (* (imag-part z) (imag-part z))
7   )))
8 (define (f1 z zi) (+ (* zi zi) z))
9 (define (ite-in x fun)
10  (define (f xi ite)
11    (if (> ite maxite) ite
12      (let ([nx (fun x xi)])
13        (if (> (dist nx) maxdist) ite
14          (f nx (add1 ite))
15        )))
16      )))
17  (f x 0))
18 (define (f x y)
19  (let ([nx (/ (- x 320) 160.0)]
20    [ny (/ (- y 240) 140.0)])
21    (if (< (ite-in (make-rectangular nx ny) f1) maxite)
22      "black" "lightgreen")
23    )))
24 (mk-image f 640 480)
```



Pour ajouter un dégradé sur les valeurs inférieures à 10 et un dégradé sur les valeurs supérieures à 10, on modifie la fonction `f` en testant le nombre d'itérations pour être au-delà de la distance `maxdist`.

```
1 (define (f x y)
2   (let* ([nx (/ (- x 320) 160.0)]
3         [ny (/ (- y 240) 140.0)]
4         [nite (ite-in (make-rectangular nx ny) f1)])
5     (if (< nite 10)
6         (color (* nite 26) 0 (* nite 26))
7         (color 0 (- 255 (* nite 10)) 0)))
8   )
9 (mk-image f 640 480)
```



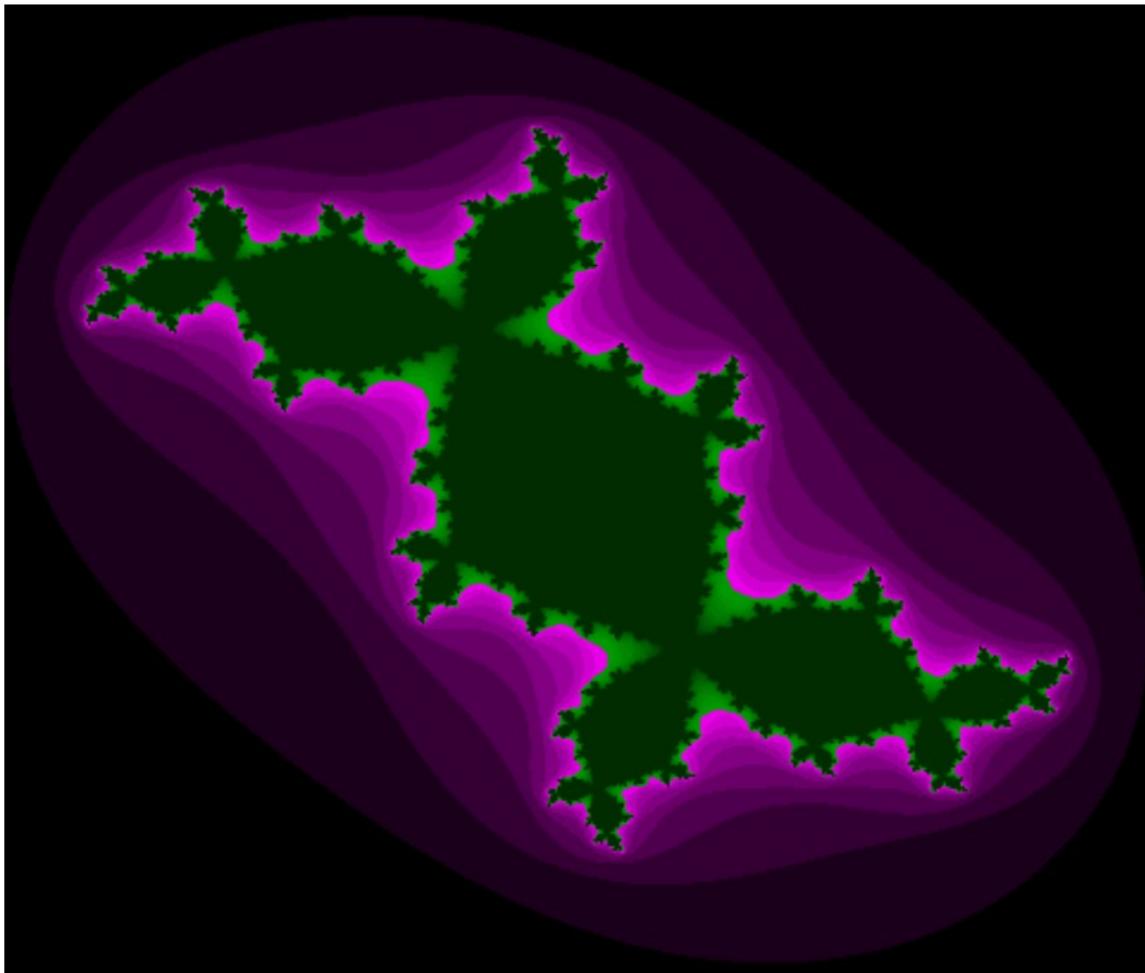


Fig. 29 – Un lapin avec (**define** (**f1** **z** **zi**) **(+(\* zi zi) -0.125-0.758i)**).

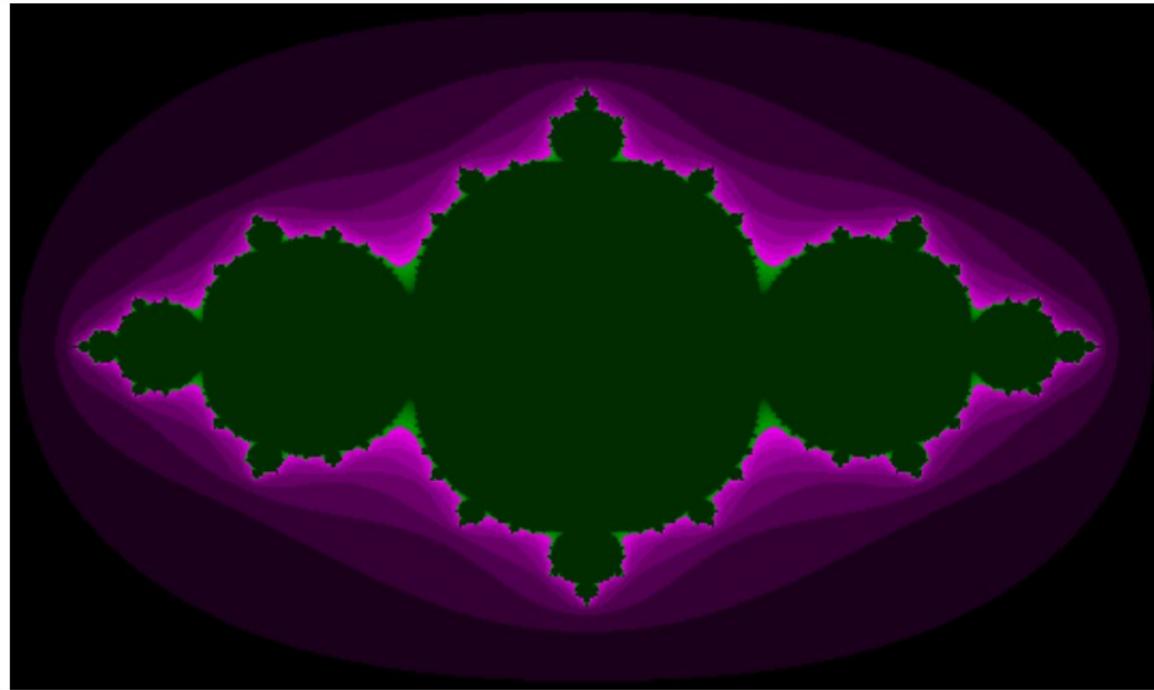


Fig. 30 – Une Julia avec (define (f1 z zi) (+(\* zi zi) -0.75+0.0i)).

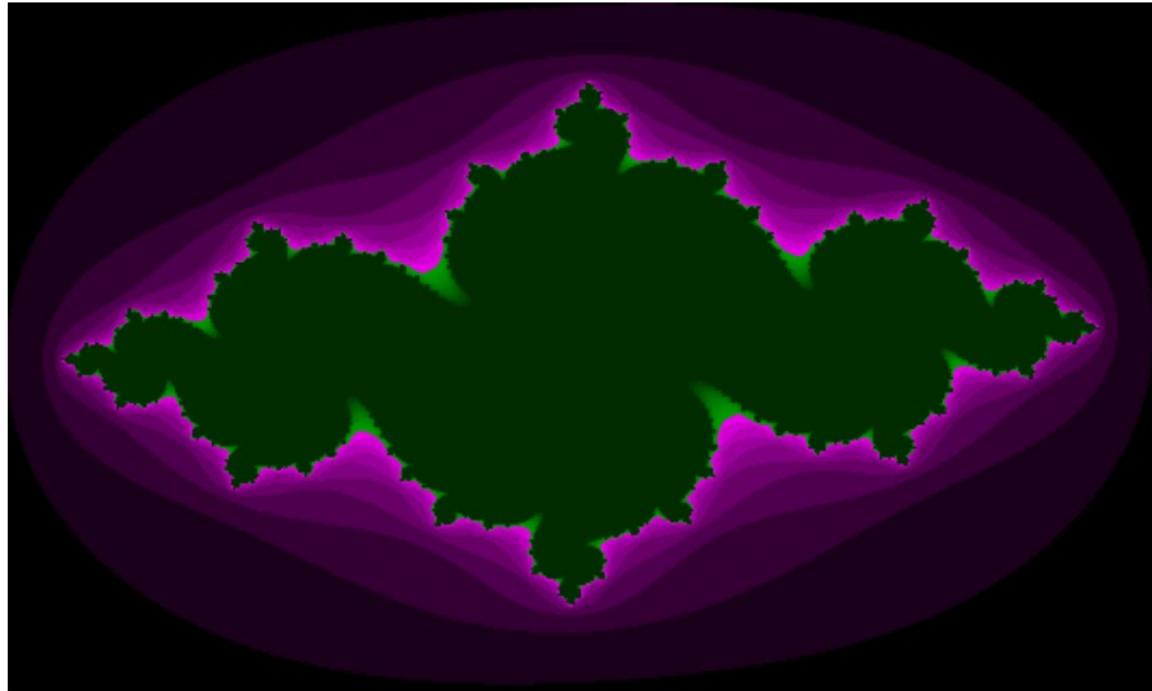


Fig. 31 – Une Julia avec (define (f1 z zi) (+(\* zi zi) -0.75+0.1i)).

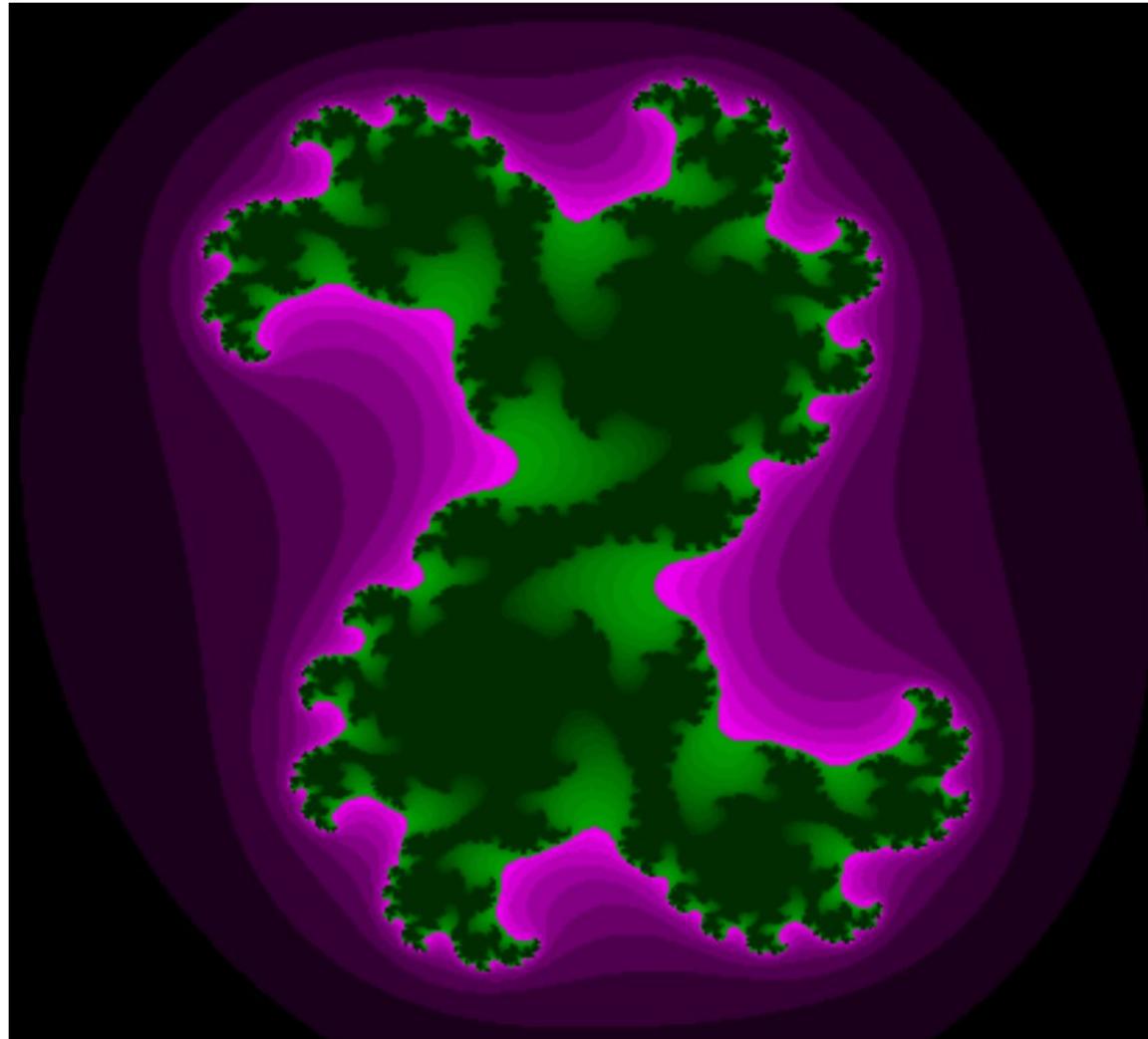


Fig. 32 – Un dragon avec (define (f1 z zi) (+(\* zi zi) 0.4-0.192i)).

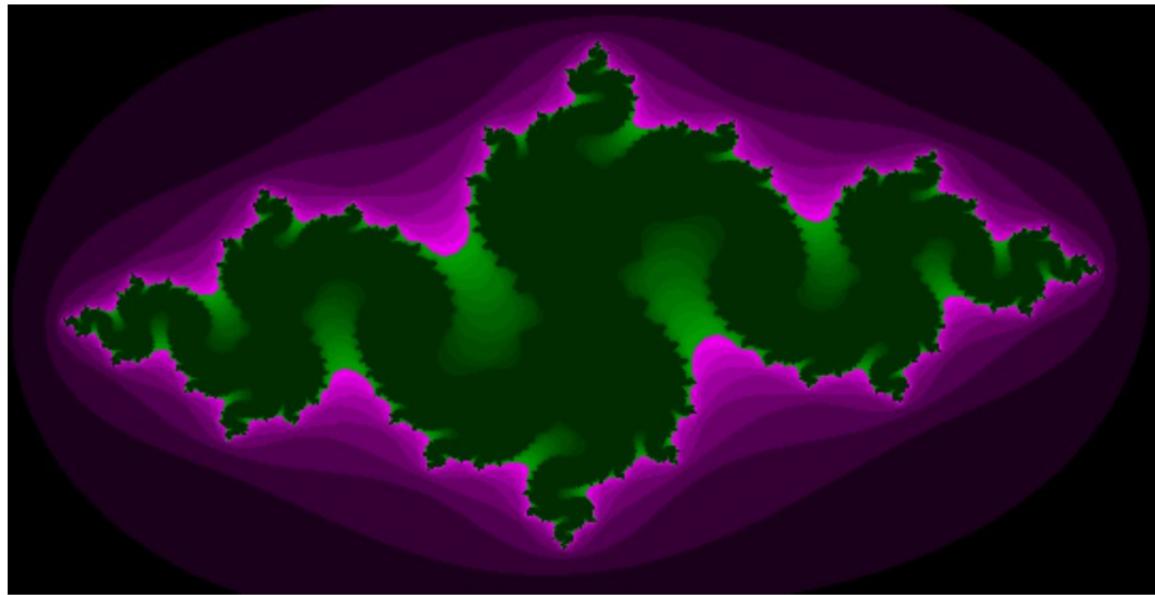


Fig. 33 – Un dragon avec (**define** (*f1 z zi*) *(+(\* zi zi) -0.8+0.168i)*).

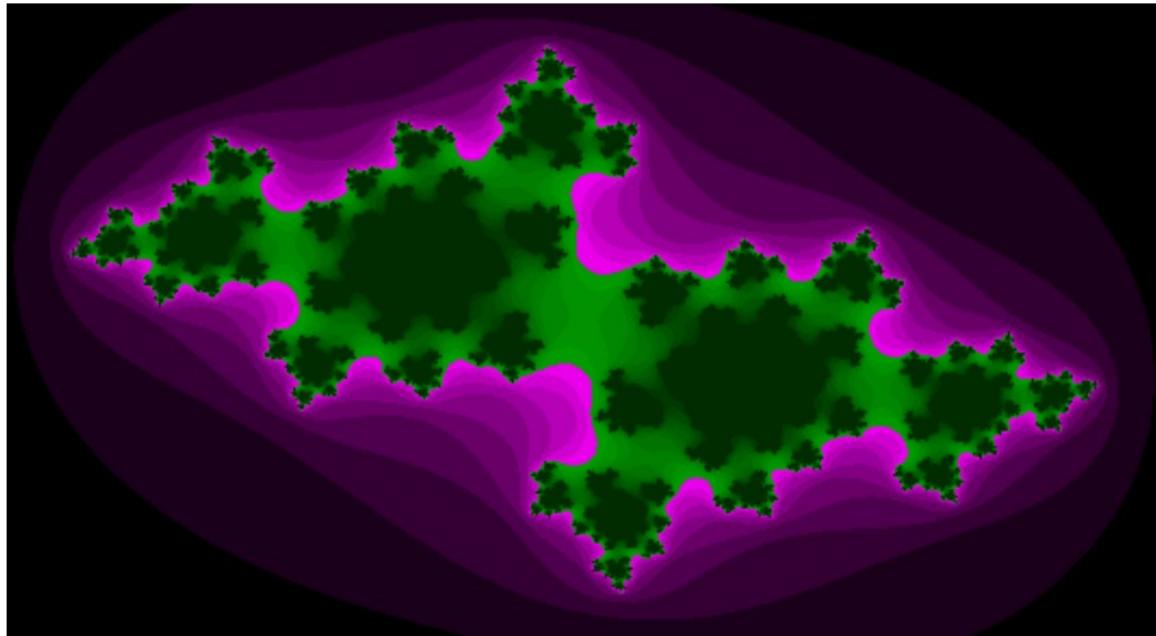


Fig. 34 – Des îles avec (**define** (*f1 z zi*) *(+(\* zi zi) -0.683-0.408i)*).

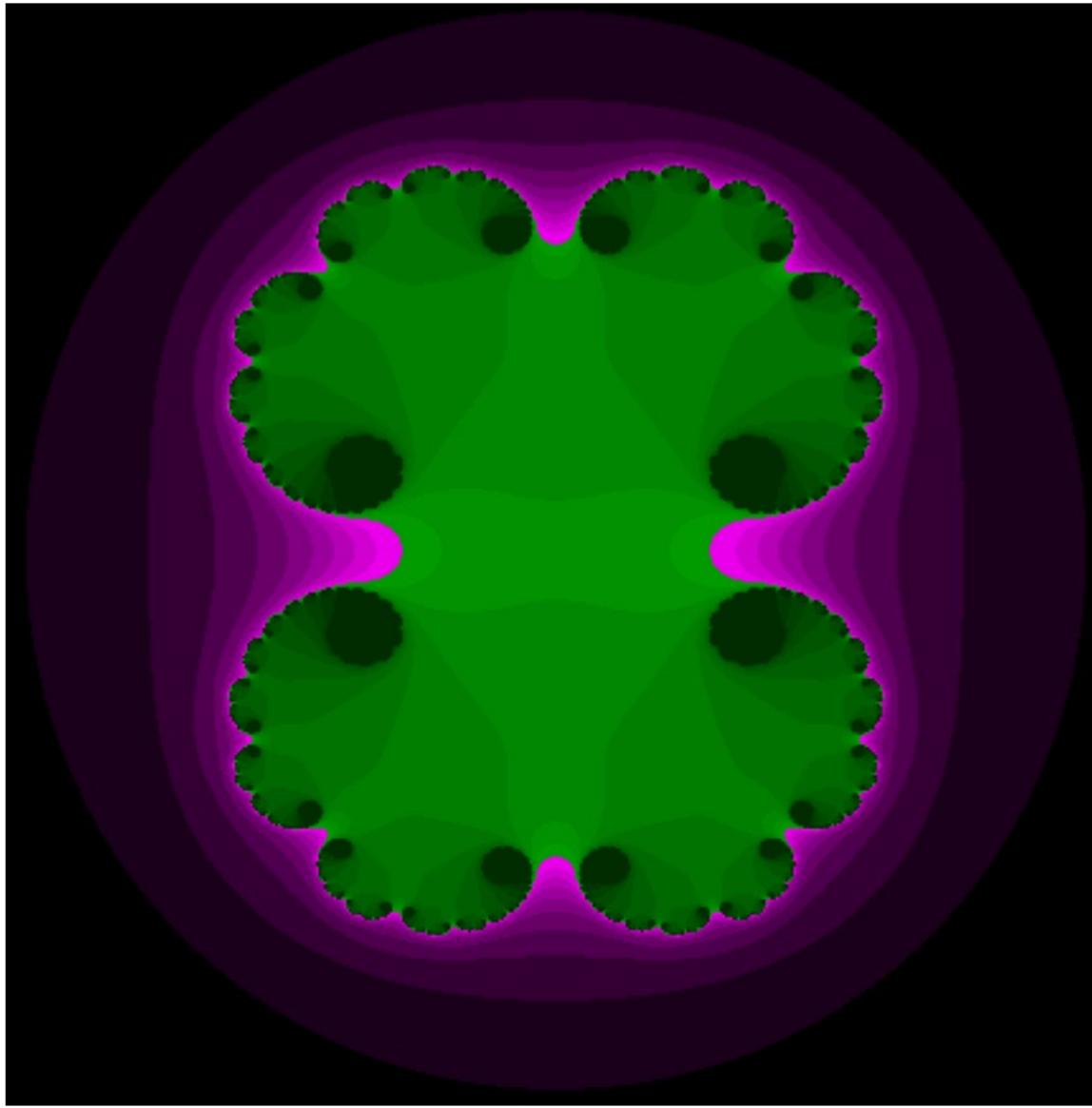


Fig. 35 – Un trèfle avec (define (f1 z zi) (+(\* zi zi) 0.3)).

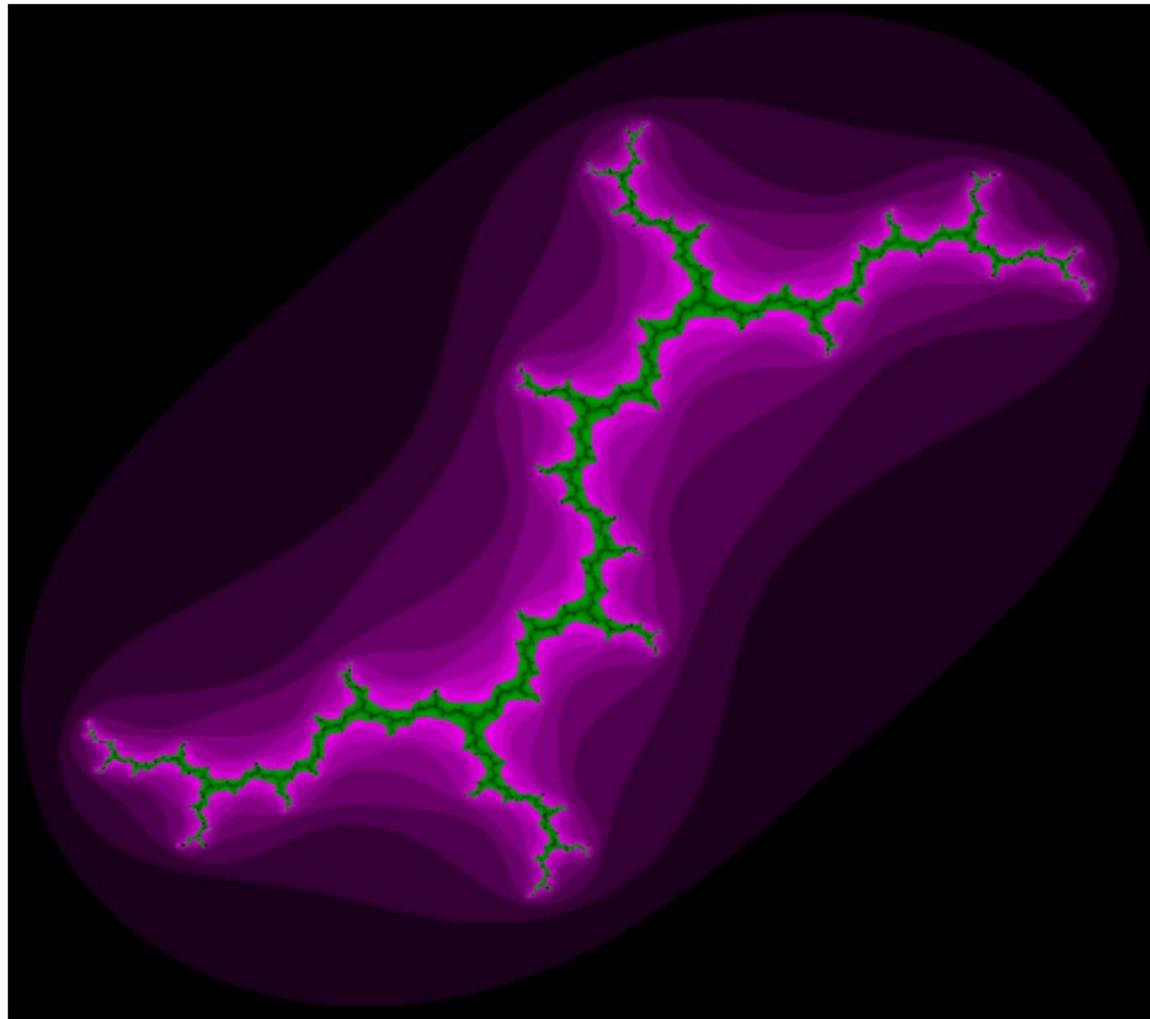


Fig. 36 – Un dendrite avec (define (f1 z zi) (+ (\* zi zi) 0.0+i)).

### 3. Exercices

#### Question 4

---

Réaliser la mosaïque correspondant à la figure 43 ; en définissant des fonctions et des variables locales, minimiser la taille du programme correspondant.

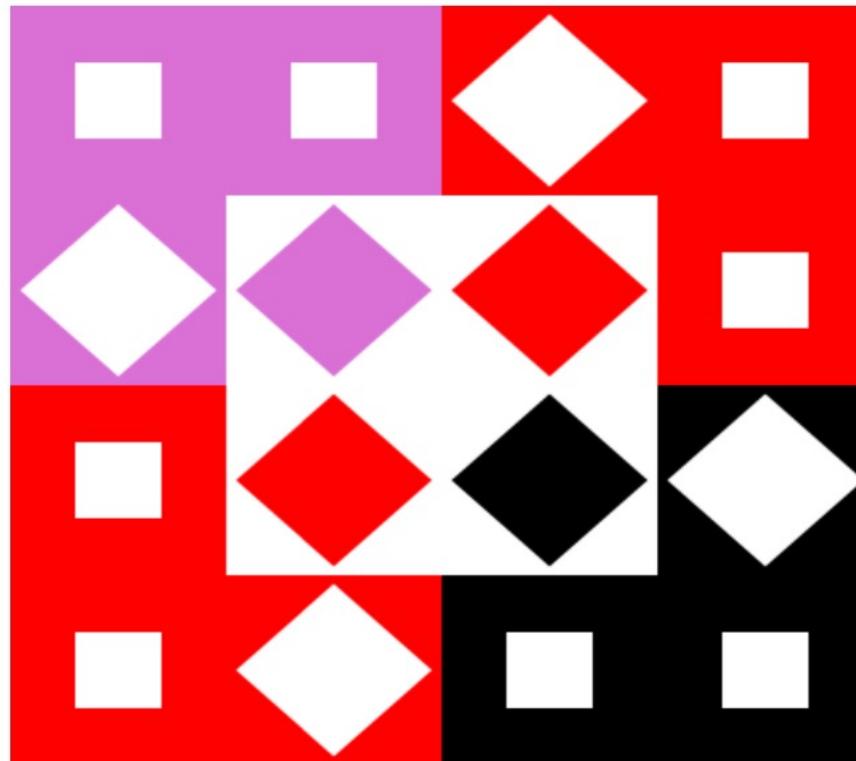
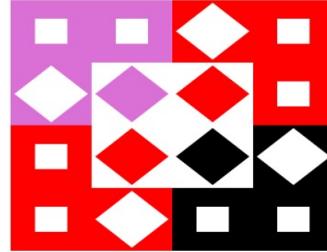


Fig. 43 – Mosaïque.

### 3. Exercices

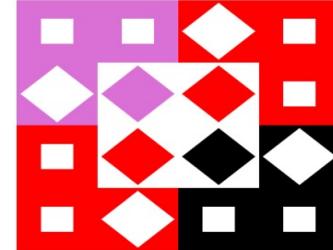
#### Indications:



1. Définir une fonction `ss` qui met un carré de couleur `c1` et de dimension « `side` » à l'intérieur d'un carré de couleur `c2` et de dimension « `side/2` ».

### 3. Exercices

#### Indications:



1. Définir une fonction ss qui met un carré de couleur c1 et de dimension « side » à l'intérieur d'un carré de couleur c2 et de dimension « side/2 ».

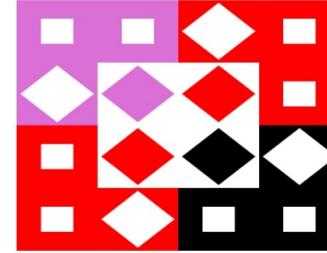
2. Début du programme:

(require 2htdp/image)

(define (ss c1 c2 [side 100])

### 3. Exercices

#### Indications:



1. Définir une fonction ss qui met un carré de couleur c1 et de dimension « side » à l'intérieur d'un carré de couleur c2 et de dimension « side/2 ».
2. Début du programme:

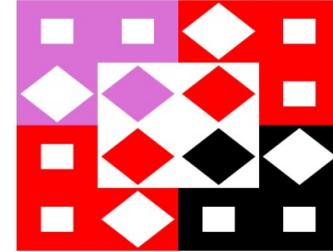
(require 2htdp/image)

(define (ss c1 c2 [side 100])

```
1 (require 2htdp/image)
2 (define (ss c1 c2 [side 100])
3   (underlay (square side "solid" c1) (square (/ side 2.5) "solid" c2)))
4 )
```

### 3. Exercices

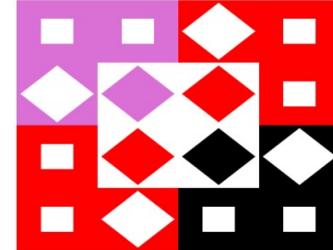
#### Indications:



1. Définir une fonction **ss** qui met un carré de couleur c1 et de dimension « side » à l'intérieur d'un carré de couleur c2 et de dimension « side/2 ».
  
2. Début du programme:  
(require 2htdp/image)  
(define (ss c1 c2 [side 100])
  
3. Définir une fonction **srs** qui permet de mettre un losange à l'intérieur d'un carré.

### 3. Exercices

#### Indications:



1. Définir une fonction **ss** qui met un carré de couleur c1 et de dimension « side » à l'intérieur d'un carré de couleur c2 et de dimension « side/2 ».

2. Début du programme:

(require 2htdp/image)

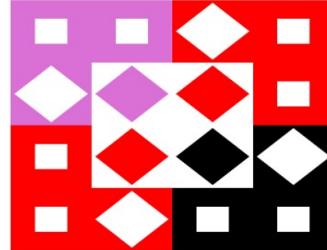
(define (ss c1 c2 [side 100])

3. Définir une fonction **srs** qui permet de mettre un losange de couleur c1 à l'intérieur d'un carré de couleur c2.

4. Un losange est une rotation de 45° d'un carré.

### 3. Exercices

#### Indications:

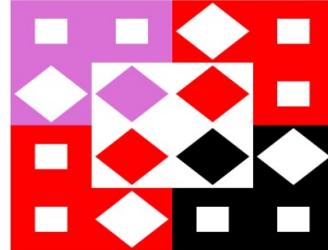


3. Définir une fonction **srs** qui permet de mettre un losange de couleur **c1** à l'intérieur d'un carré de couleur **c2**.
4. Un losange est une rotation de 45° d'un carré.

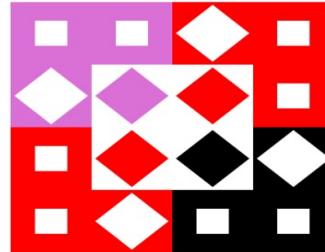
```
5 | (define (srs c1 c2 [side 100])
6 |   (underlay (square side "solid" c1)
7 |     (rotate 45 (square (/ side (* 1.1 (sqrt 2))) "solid" c2)))
8 |   )
```

### 3. Exercices

#### Indications:



3. Définir une fonction **srs** qui permet de mettre un losange de couleur **c1** à l'intérieur d'un carré de couleur **c2**.
4. Un losange est une rotation de 45° d'un carré.
5. Afficher toute la table à l'aide des fonctions « **above** » et « **beside** ».



```
1 (require 2htdp/image)
2 (define (ss c1 c2 [side 100])
3   (underlay (square side "solid" c1) (square (/ side 2.5) "solid" c2)))
4 )
5 (define (srs c1 c2 [side 100])
6   (underlay (square side "solid" c1)
7     (rotate 45 (square (/ side (* 1.1 (sqrt 2))) "solid" c2))))
8 )
9 (let ([O "orchid"] [W "white"] [R "red"] [B "black"])
10  (above
11    (beside (ss O W) (ss O W) (srs R W) (ss R W))
12    (beside (srs O W) (srs W O) (srs W R) (ss R W))
13    (beside (ss R W) (srs W R) (srs W B) (srs B W))
14    (beside (ss R W) (srs R W) (ss B W) (ss B W)))
15  ))
```

## Question 5

---

Définir quatres fonctions permettant de réaliser des dégradés sur des images de 255 par 255 telles que :

- **f1** réalise un dégradé du rouge vers le noir
- **f2** réalise un dégradé du noir vers le vert
- **f3** réalise un dégradé du vert vers le noir
- **f4** réalise un dégradé du noir vers le bleu

Ces quatres fonctions retournent une couleur en fonction de deux paramètres **x** et **y**; avec la fonction (**beside (f f1) (f f2) (f f3) (f f4)**), on obtient les dégradés présentés en figure 44.

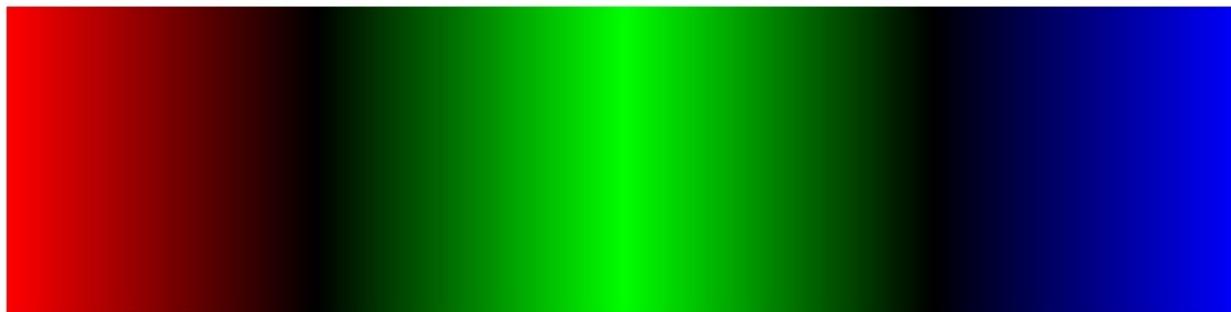
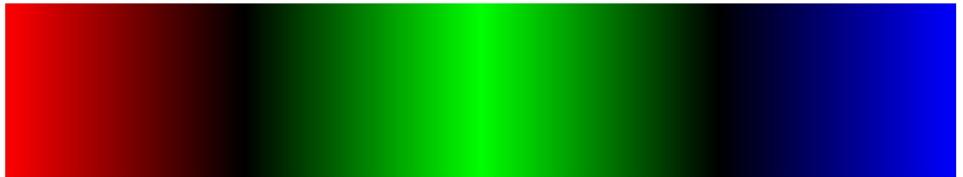


Fig. 44 – Quelques dégradés horizontaux.

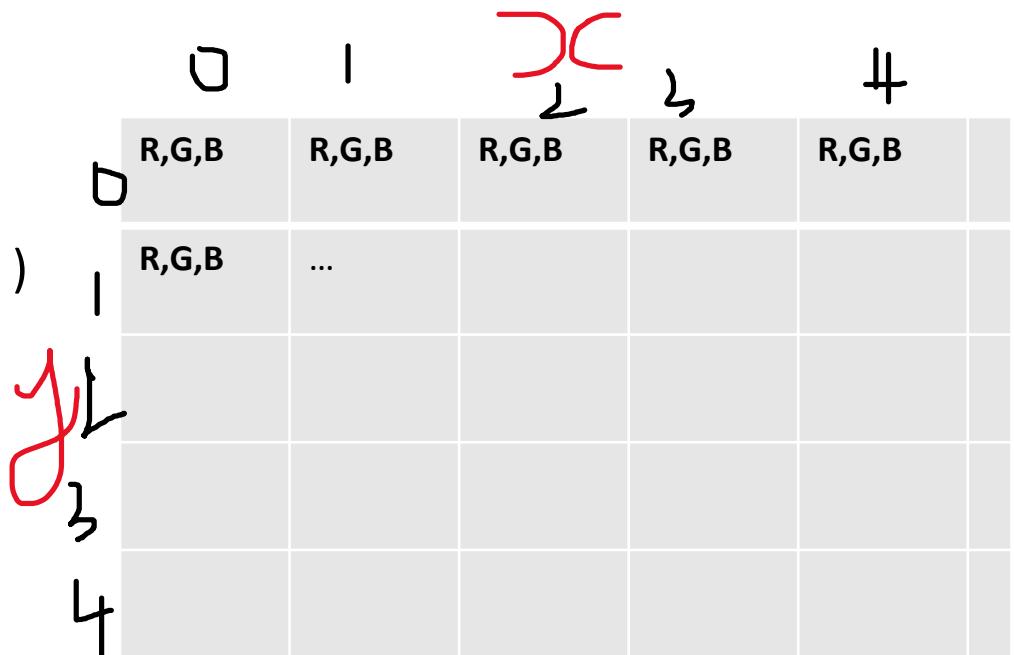


## Indications:

f1: dégradé R->N

1. Paramètres f1: x et y.
2. La couleur rouge: (color 255 0 0 )
3. La couleur noir: (color 0 0 0)

Est-ce (color x 0 0)?



## Solution 5

---

```
1 (require 2htdp/image)
2 (define (f1 x y) (color (- 255 x) 0 0))
```

### Question 5

---

Définir quatres fonctions permettant de réaliser des dégradés sur des images de 255 par 255 telles que :

- **f1** réalise un dégradé du rouge vers le noir
- **f2** réalise un dégradé du noir vers le vert
- **f3** réalise un dégradé du vert vers le noir
- **f4** réalise un dégradé du noir vers le bleu

Ces quatres fonctions retournent une couleur en fonction de deux paramètres **x** et **y** ; avec la fonction (**beside** (**f f1**) (**f f2**) (**f f3**) (**f f4**)), on obtient les dégradés présentés en figure 44.

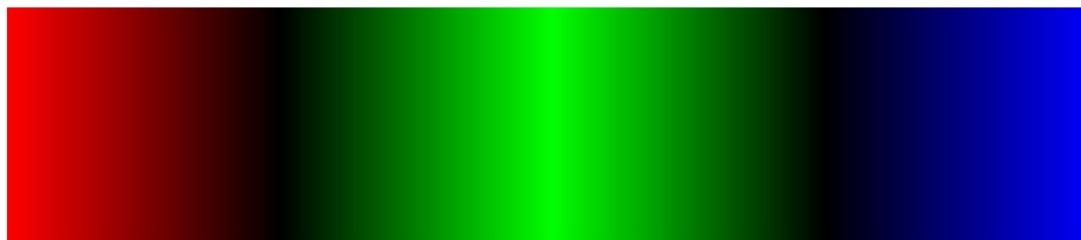
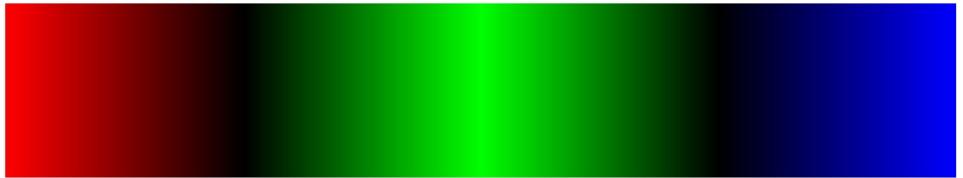


Fig. 44 – Quelques dégradés horizontaux.



## Indications:

f2: dégradé N->V

1. Paramètres f2: x et y.
2. La couleur noir: (color 0 0 0 )
3. La couleur verte: (color 0 255 0)



## Solution 5

---

```
1 (require 2htdp/image)
2 (define (f1 x y) (color (- 255 x) 0 0))
3 (define (f2 x y) (color 0 x 0))
```

## Question 5

---

Définir quatres fonctions permettant de réaliser des dégradés sur des images de 255 par 255 telles que :

- **f1** réalise un dégradé du rouge vers le noir
- **f2** réalise un dégradé du noir vers le vert
- **f3** réalise un dégradé du vert vers le noir
- **f4** réalise un dégradé du noir vers le bleu

Ces quatres fonctions retournent une couleur en fonction de deux paramètres **x** et **y** ; avec la fonction (**beside** (**f f1**) (**f f2**) (**f f3**) (**f f4**)), on obtient les dégradés présentés en figure 44.

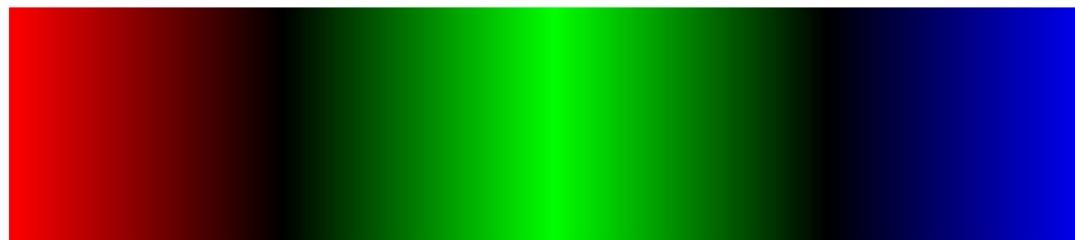
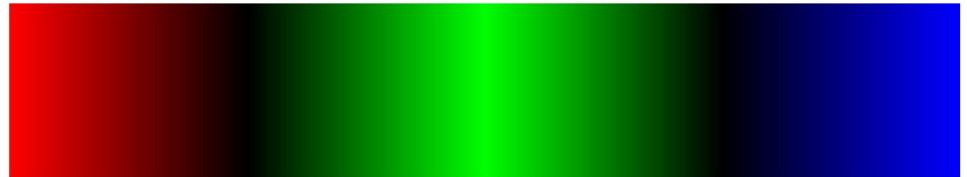


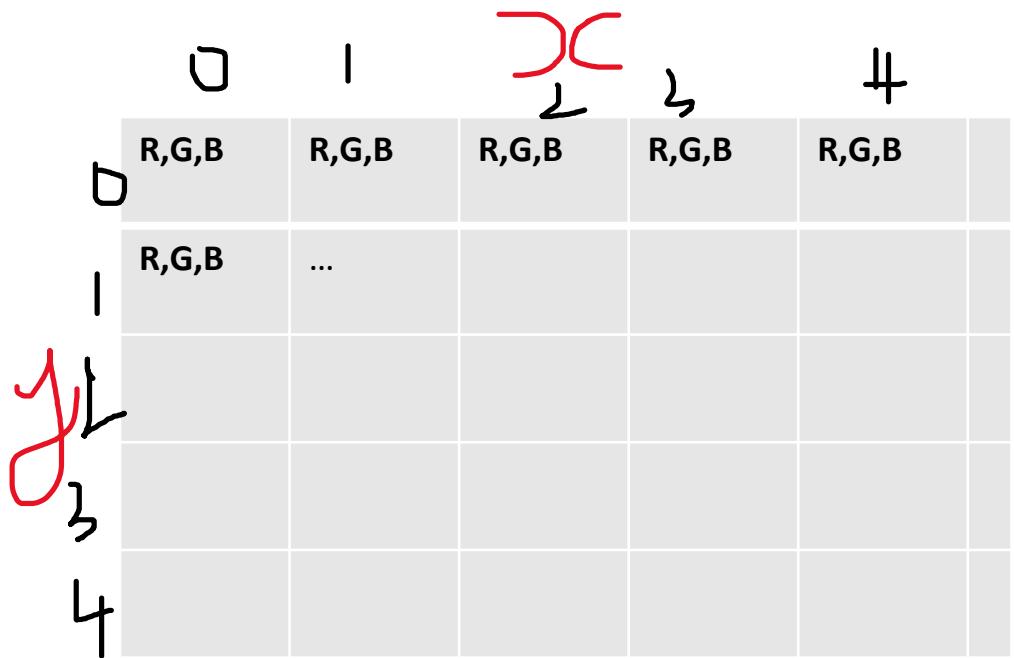
Fig. 44 – Quelques dégradés horizontaux.



## Indications:

f3: dégradé V->N

1. Paramètres f3: x et y.
2. La couleur noir: (color 0 0 0 )
3. La couleur verte: (color 0 255 0)



## Solution 5

---

```
1 (require 2htdp/image)
2 (define (f1 x y) (color (- 255 x) 0 0))
3 (define (f2 x y) (color 0 x 0))
4 (define (f3 x y) (color 0 (- 255 x) 0))
```

## Question 5

---

Définir quatres fonctions permettant de réaliser des dégradés sur des images de 255 par 255 telles que :

- **f1** réalise un dégradé du rouge vers le noir
- **f2** réalise un dégradé du noir vers le vert
- **f3** réalise un dégradé du vert vers le noir
- **f4** réalise un dégradé du noir vers le bleu

Ces quatres fonctions retournent une couleur en fonction de deux paramètres **x** et **y** ; avec la fonction (**beside** (**f f1**) (**f f2**) (**f f3**) (**f f4**)), on obtient les dégradés présentés en figure 44.

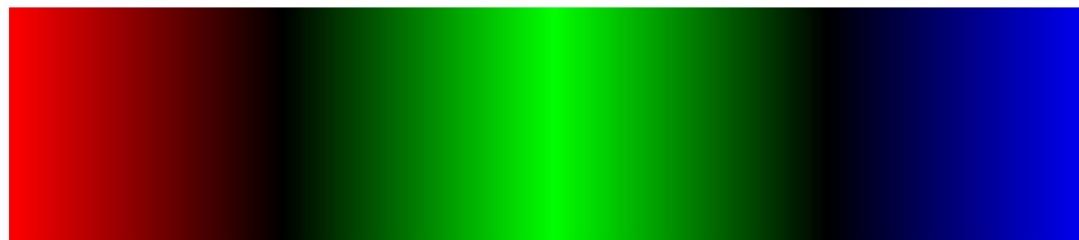
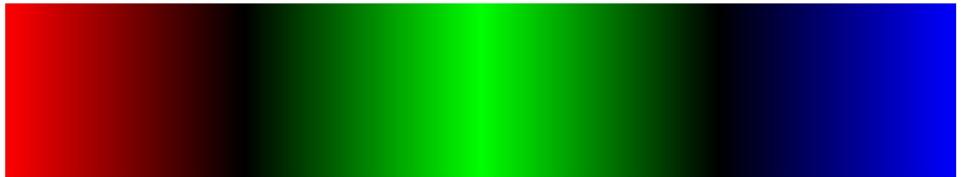


Fig. 44 – Quelques dégradés horizontaux.



## Indications:

f4: dégradé N->B

1. Paramètres f4: x et y.
2. La couleur noir: (color 0 0 0 )
3. La couleur verte: (color 0 0 255)



## Solution 5

---

```
1 (require 2htdp/image)
2 (define (f1 x y) (color (- 255 x) 0 0))
3 (define (f2 x y) (color 0 x 0))
4 (define (f3 x y) (color 0 (- 255 x) 0))
5 (define (f4 x y) (color 0 0 x))
6 (define (f fun) (mk-image fun 255 255))
7 (beside (f f1) (f f2) (f f3) (f f4))
```

## Question 6

---

Définir quatres fonctions permettant de réaliser des dégradés sur des images de 255 par 255 telles que :

- $f_1$  réalise un dégradé du noir vers le rouge en diagonal vers le bas à droite
- $f_2$  réalise le dégradé complémentaire de  $f_1$
- $f_3$  réalise un dégradé du rouge vers le noir
- $f_4$  réalise un dégradé du rouge au rouge en passant par le noir

Ces quatres fonctions produisent des dégradés diagonaux ; elles retournent une couleur en fonction de deux paramètres  $x$  et  $y$  ; avec la fonction (`beside (f f1) (f f2) (f f3) (f f4)`), on obtient les dégradés présentés en figure 45.



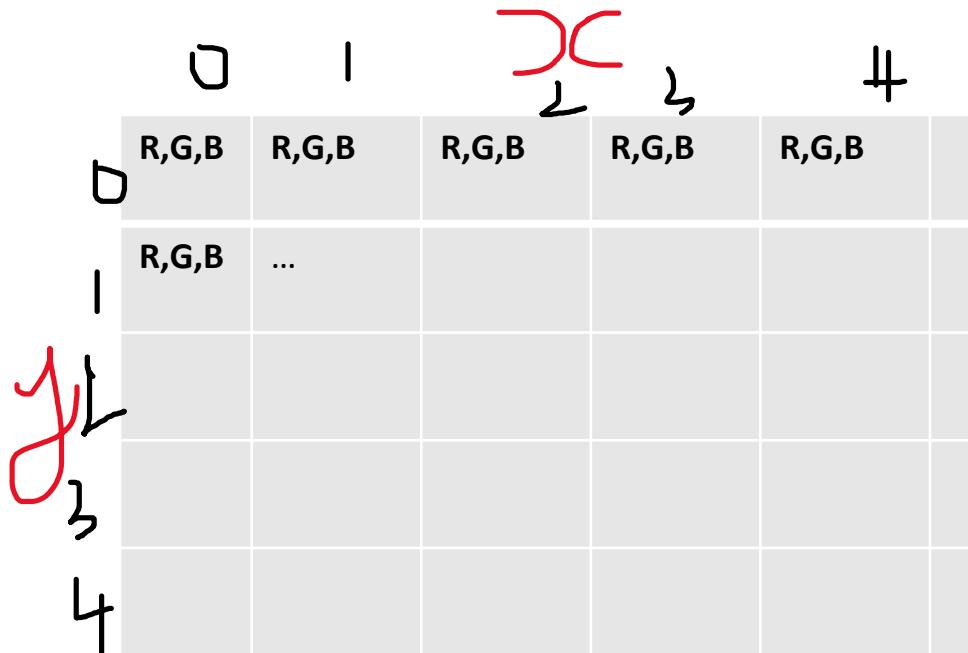
Fig. 45 – Quelques dégradés diagonaux entre noir et rouge.



### Indications:

f4: dégradé N->R diagonal

1. Paramètres f1: x et y.
2. La couleur noir: (color 0 0 0 )
3. La couleur rouge: (color 255 0 0)
4. Utiliser la fonction clamp pour l'intervalle de couleur à [0,255]



```
| (define (clamp pix-val) (max 0 (min 255 pix-val)))
```

## Solution 6

---

```
1 (require 2htdp/image)
2 (define (f1 x y) (color (clamp (+ x y)) 0 0))
```

## Question 6

---

Définir quatres fonctions permettant de réaliser des dégradés sur des images de 255 par 255 telles que :

- **f1** réalise un dégradé du noir vers le rouge en diagonal vers le bas à droite
- **f2** réalise le dégradé complémentaire de **f1**
- **f3** réalise un dégradé du rouge vers le noir
- **f4** réalise un dégradé du rouge au rouge en passant par le noir

Ces quatres fonctions produisent des dégradés diagonaux ; elles retournent une couleur en fonction de deux paramètres **x** et **y** ; avec la fonction (**beside** (**f1**) (**f2**) (**f3**) (**f4**)), on obtient les dégradés présentés en figure 45.



Fig. 45 – Quelques dégradés diagonaux entre noir et rouge.

## Question 6

---

Définir quatres fonctions permettant de réaliser des dégradés sur des images de 255 par 255 telles que :

- **f1** réalise un dégradé du noir vers le rouge en diagonal vers le bas à droite
- **f2** réalise le dégradé complémentaire de **f1**
- **f3** réalise un dégradé du rouge vers le noir
- **f4** réalise un dégradé du rouge au rouge en passant par le noir

Ces quatres fonctions produisent des dégradés diagonaux ; elles retournent une couleur en fonction de deux paramètres **x** et **y** ; avec la fonction (**beside** (**f1**) (**f f2**) (**f f3**) (**f f4**)), on obtient les dégradés présentés en figure 45.



Fig. 45 – Quelques dégradés diagonaux entre noir et rouge.

## Solution 6

---

```
1 (require 2htdp/image)
2 (define (f1 x y) (color (clamp (+ x y)) 0 0))
3 (define (f2 x y) (color (- 255 (clamp (+ x y))) 0 0))
```



## Solution 6

```
1 (require 2htdp/image)
2 (define (f1 x y) (color (clamp (+ x y)) 0 0))
3 (define (f2 x y) (color (- 255 (clamp (+ x y))) 0 0))
4 (define (f3 x y) (color (- 255 (clamp (- (+ x y) 255))) 0 0))
5 (define (f4 x y)
6   (if (< (+ x y) 255) (color (- 255 (clamp (+ x y))) 0 0)
7       (color (clamp (- (+ x y) 255)) 0 0)
8   ))
9 (define (f fun) (mk-image fun 255 255))
10 (beside (f f1) (f f2) (f f3) (f f4))
```

## Question 7

---

Définir une fonction  $f$  permettant d'obtenir le dégradé présenté en figure 46 ; la première et la deuxième composante de couleur sont fixées respectivement à 50 et 20 ; la troisième composante varie en fonction de  $(\cos x)$ .

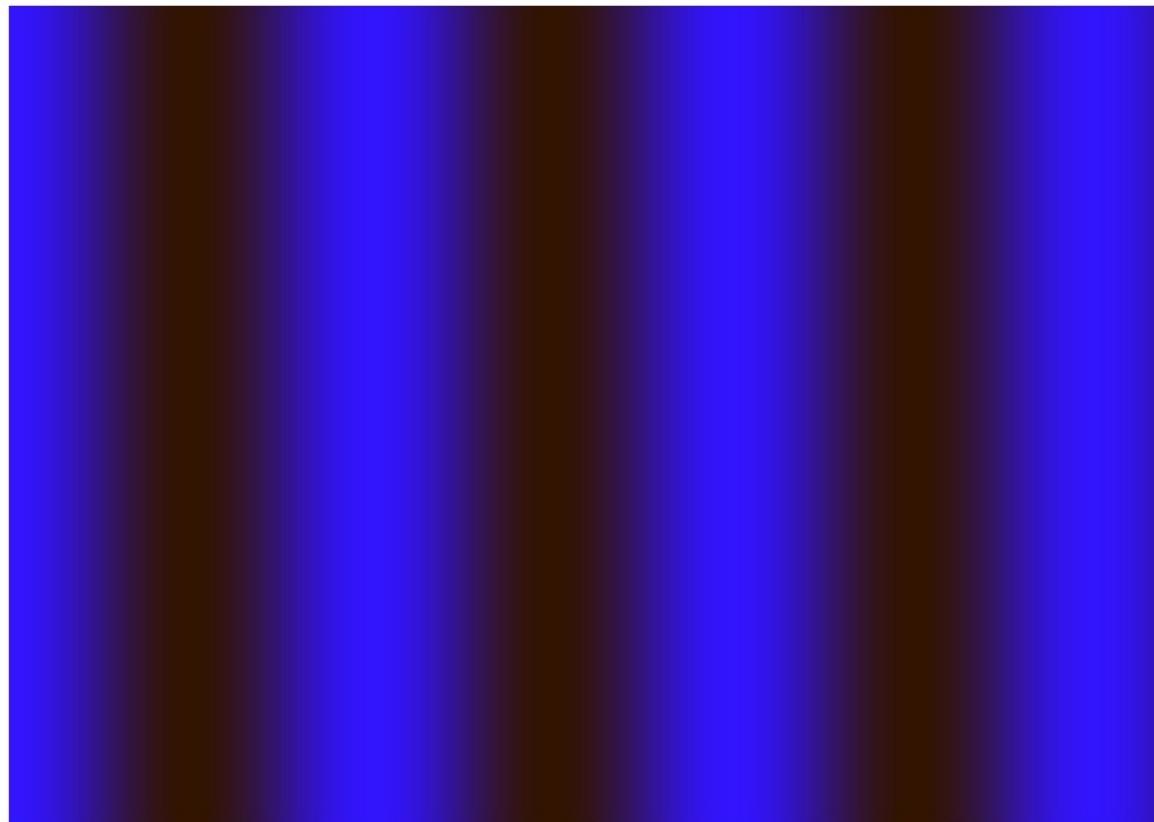


Fig. 46 – Dégradé avec la fonction  $\cos$ .

## **Indications:**

(color 50 20 c1)

$-1 \leq \cos(x) < 1$

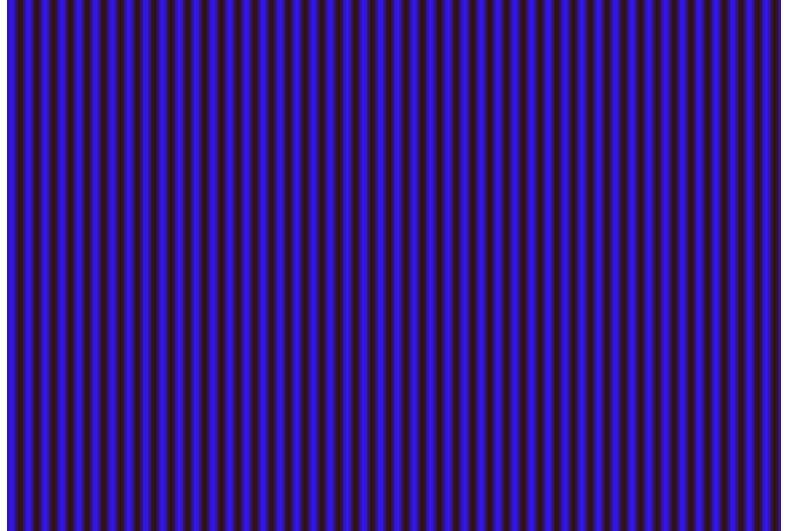
Or c1 on veut qu'elle varie entre 0 et 256!

## **Indications:**

$$-1 \leq \cos(x) < 1$$

Soit  $c1 = 127 + \cos(x) * 127$

$$0 \leq c1 \leq 254$$



Si  $\cos(x)=1$  alors  $c1$  bleu

Si  $\cos(x)=-1$  alors  $c1$  est noir

## Indications:

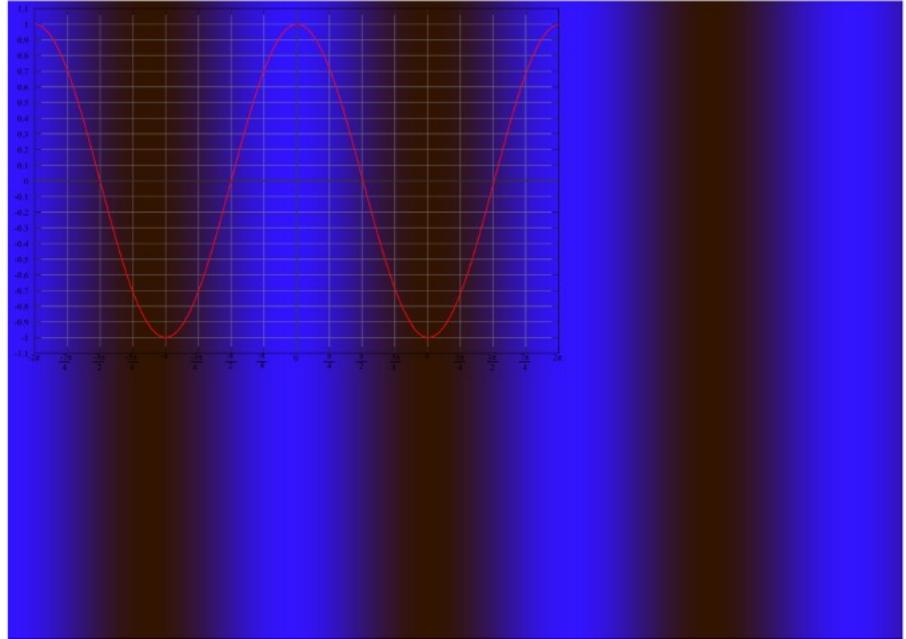
$$-1 \leq \cos(x) \leq 1$$

$$\text{Soit } c1 = 127 + \cos\left(\frac{x}{32}\right) * 127$$

$$0 \leq c1 \leq 254$$

Si  $\cos\left(\frac{x}{32}\right) = 1$  alors c1 bleu

Si  $\cos\left(\frac{x}{32}\right) = -1$  alors c1 est noir



## Solution 7

---

```
1 (require 2htdp/image)
2 (define (f x y)
3   (let ([c1 (+ 127 (* (cos (/ x 32)) 127))])
4     (color 50 20 (clamp (exact-floor c1))))
5   )
6 (mk-image f 640 400)
```

## Question 8

---

Définir une fonction  $f$  permettant d'obtenir le dégradé présenté en figure 47 ; la première composante de couleur est fixée à 50 ; la deuxième composante varie en fonction de  $(\cos x)$  et la troisième composante en fonction de  $(\cos (+ x y))$ .

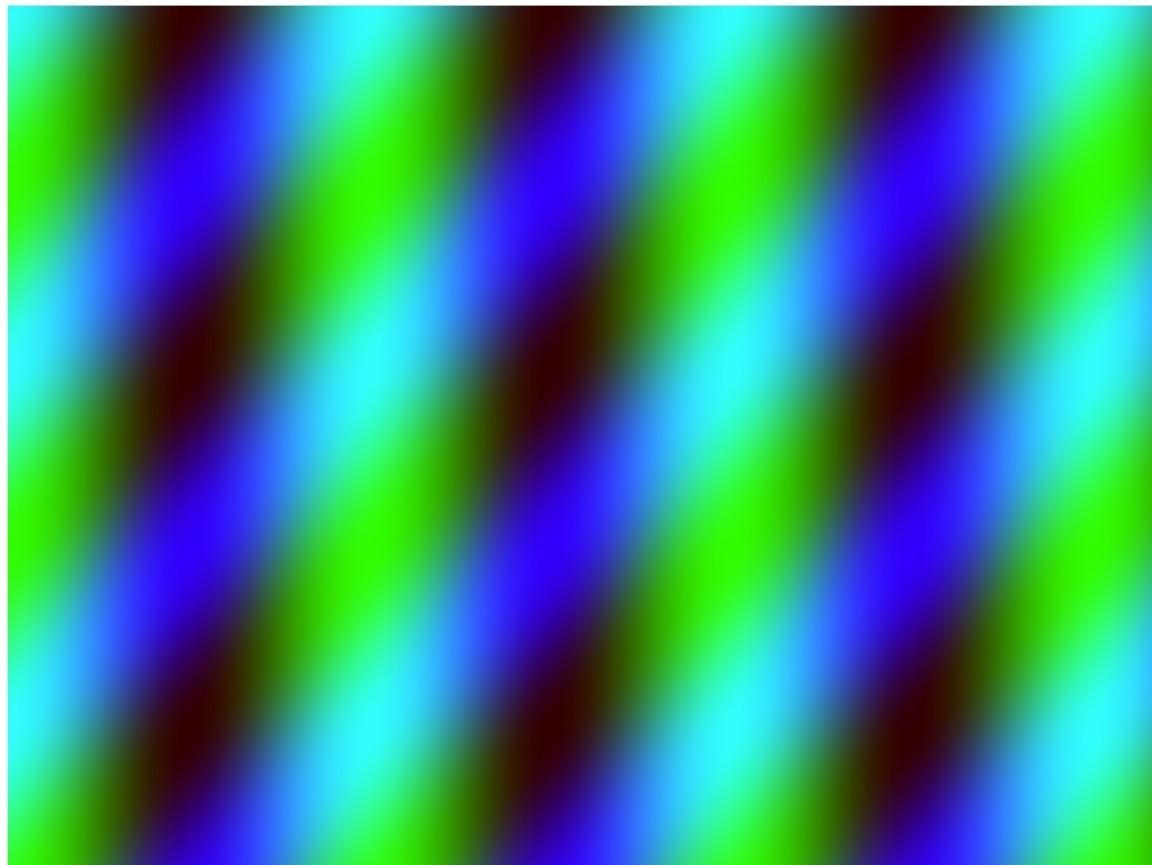


Fig. 47 – Dégradé avec la fonction cos.

## Solution 8

---

```
1 (require 2htdp/image)
2 (define (f x y)
3   (let ([c1 (+ 127 (* (cos (/ x 32)) 127))]
4       [c2 (+ 127 (* (cos (/ (+ x y) 16)) 127))])
5     (color 50 (clamp (exact-floor c1)) (clamp (exact-floor c2))))
6   ))
7 (mk-image f 640 400)
```

## Question 11

---

Définir une fonction  $f$  permettant de réaliser un filtre médian sur une image ; pour calculer un pixel médian  $p$  de coordonnées  $(x \ y)$ , on pourra faire la moyenne des pixels de coordonnées  $(x \ y)$ ,  $((x-1) \ y)$ ,  $((x+1) \ y)$ ,  $(x \ (y-1))$  et  $(x \ (y+1))$  ; pour tout pixel en dehors de l'image, on utilisera le pixel de valeurs rgb  $(0,0,0)$ .

Définir une fonction `get-pix` permettant récupérer le pixel de coordonnées  $(x \ y)$  d'une image.

Définir une fonction `filter-color-list` permettant d'appliquer une fonction de filtrage sur une liste de pixels.

Définir une fonction `filter-image` permettant d'appliquer une fonction de filtrage sur une image.

Avec `(filter-image (bitmap "/home/n/30x30.png") f)`, on obtient la figure 51 par application du filtre médian sur la figure 50.



Fig. 50 – Image initiale.

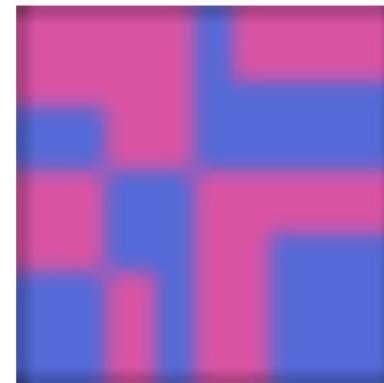


Fig. 51 – Image médiane.

Définir une fonction  $f$  permettant de réaliser un filtre médian sur une image ; pour calculer un pixel médian  $p$  de coordonnées  $(x \ y)$ , on pourra faire la moyenne des pixels de coordonnées  $(x \ y)$ ,  $((x-1) \ y)$ ,  $((x+1) \ y)$ ,  $(x \ (y-1))$  et  $(x \ (y+1))$  ; pour tout pixel en dehors de l'image, on utilisera le pixel de valeurs  $rgb(0,0,0)$ .

Définir une fonction **get-pix** permettant récupérer le pixel de coordonnées  $(x \ y)$  d'une image.

|       |       |       |       |       |
|-------|-------|-------|-------|-------|
| R,G,B | R,G,B | R,G,B | R,G,B | R,G,B |
| R,G,B | ...   | Blue  |       |       |
|       | Blue  | Grey  | Blue  |       |
|       |       | Blue  |       |       |
|       |       |       |       |       |

### Indications:

1. La fonction **get-pix** a pour paramètres:

Le pixel sous forme d'une liste

La largeur de l'image

La hauteur de l'image

Les coordonnées x et y

2. Fonction utile: **list-ref**:

Retourne l'élément présent à une position spécifique dans une liste.

```
1 (require srfi-1)
2 (define (get-pix L w h x y)
3   (if (or (< x 0) (>= x w)) (color 0 0 0)
4       (if (or (< y 0) (>= y h)) (color 0 0 0)
5           (list-ref L (+ (* y w) x)))
6   )))
```

## Question 11

---

Définir une fonction  $f$  permettant de réaliser un filtre médian sur une image ; pour calculer un pixel médian  $p$  de coordonnées  $(x \ y)$ , on pourra faire la moyenne des pixels de coordonnées  $(x \ y)$ ,  $((x-1) \ y)$ ,  $((x+1) \ y)$ ,  $(x \ (y-1))$  et  $(x \ (y+1))$  ; pour tout pixel en dehors de l'image, on utilisera le pixel de valeurs rgb  $(0,0,0)$ .

Définir une fonction `get-pix` permettant récupérer le pixel de coordonnées  $(x \ y)$  d'une image.

Définir une fonction `filter-color-list` permettant d'appliquer une fonction de filtrage sur une liste de pixels.

### Indications:

- `filter-color-list`, Prend comme paramètre :
  1. La liste de pixels.
  2. Pix-fun: une fonction de filtrage
  3. Les coordonnées x et y
  4. La largeur w
  5. La hauteur h
- On parcourt notre image et pour chaque pixel on lui applique la fonction de filtrage

## Solution 11

---

```
1 (require 2htdp/image)
2 (define (get-pix L w h x y)
3     (if (or (< x 0) (>= x w)) (color 0 0 0)
4         (if (or (< y 0) (>= y h)) (color 0 0 0)
5             (list-ref L (+ (* y w) x))
6             )))
7 (define (filter-color-list L pix-fun x y w h)
8     (if (= y h) empty
9         (let* ([tmpx (add1 x)]
10            [ny (if (= tmpx w) (add1 y) y)])
11            [nx (if (= tmpx w) 0 (add1 x))])
12            (cons (pix-fun L x y w h) (filter-color-list L pix-fun nx ny w h)))
13        )))
```

nx parcourt les colonnes  
ny parcourt les lignes

## Question 11

---

Définir une fonction  $f$  permettant de réaliser un filtre médian sur une image ; pour calculer un pixel médian  $p$  de coordonnées  $(x \ y)$ , on pourra faire la moyenne des pixels de coordonnées  $(x \ y)$ ,  $((x-1) \ y)$ ,  $((x+1) \ y)$ ,  $(x \ (y-1))$  et  $(x \ (y+1))$  ; pour tout pixel en dehors de l'image, on utilisera le pixel de valeurs rgb  $(0,0,0)$ .

Définir une fonction `get-pix` permettant récupérer le pixel de coordonnées  $(x \ y)$  d'une image.

Définir une fonction `filter-color-list` permettant d'appliquer une fonction de filtrage sur une liste de pixels.

Définir une fonction `filter-image` permettant d'appliquer une fonction de filtrage sur une image.

## Question 11

---

Définir une fonction  $f$  permettant de réaliser un filtre médian sur une image ; pour calculer un pixel médian  $p$  de coordonnées  $(x, y)$ , on pourra faire la moyenne des pixels de coordonnées  $(x, y)$ ,  $((x-1), y)$ ,  $((x+1), y)$ ,  $(x, (y-1))$  et  $(x, (y+1))$  ; pour tout pixel en dehors de l'image, on utilisera le pixel de valeurs rgb  $(0, 0, 0)$ .

Définir une fonction `get-pix` permettant récupérer le pixel de coordonnées  $(x, y)$  d'une image.

Définir une fonction `filter-color-list` permettant d'appliquer une fonction de filtrage sur une liste de pixels.

Définir une fonction `filter-image` permettant d'appliquer une fonction de filtrage sur une image.

### Indications:

- Filter-image, Prend comme paramètre :
  1. L'image
  2. Pix-fun: une fonction de filtrage
  3. Fonctions utiles: `image->color-list` , `color-list->bitmap`, `filter-color-list`

## Solution 11

---

```
1 (require 2htdp/image)
2 (define (get-pix L w h x y)
3     (if (or (< x 0) (>= x w)) (color 0 0 0)
4         (if (or (< y 0) (>= y h)) (color 0 0 0)
5             (list-ref L (+ (* y w) x))
6             )))
7 (define (filter-color-list L pix-fun x y w h)
8     (if (= y h) empty
9         (let* ([tmpx (add1 x)]
10            [ny (if (= tmpx w) (add1 y) y)])
11            [nx (if (= tmpx w) 0 (add1 x))])
12            (cons (pix-fun L x y w h) (filter-color-list L pix-fun nx ny w h)))
13        )))
14 (define (filter-image img pix-fun)
15     (let ([cl (image->color-list img)]
16       [w (image-width img)])
17       [h (image-height img)])
18       (color-list->bitmap (filter-color-list cl pix-fun 0 0 w h) w h)
19    ))
```

## Question 11

---

Définir une fonction  $f$  permettant de réaliser un filtre médian sur une image ; pour calculer un pixel médian  $p$  de coordonnées  $(x \ y)$ , on pourra faire la moyenne des pixels de coordonnées  $(x \ y)$ ,  $((x-1) \ y)$ ,  $((x+1) \ y)$ ,  $(x \ (y-1))$  et  $(x \ (y+1))$  ; pour tout pixel en dehors de l'image, on utilisera le pixel de valeurs rgb  $(0,0,0)$ .

Définir une fonction `get-pix` permettant récupérer le pixel de coordonnées  $(x \ y)$  d'une image.

Définir une fonction `filter-color-list` permettant d'appliquer une fonction de filtrage sur une liste de pixels.

Définir une fonction `filter-image` permettant d'appliquer une fonction de filtrage sur une image.

Avec `(filter-image (bitmap "/home/n/30x30.png") f)`, on obtient la figure 51 par application du filtre médian sur la figure 50.

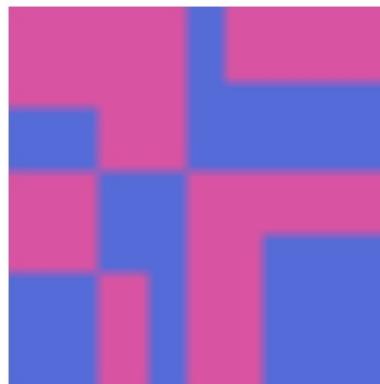


Fig. 50 – Image initiale.

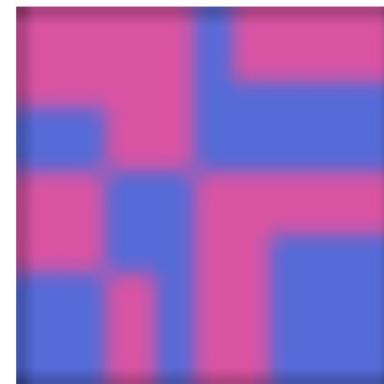


Fig. 51 – Image médiane.

## Solution 11

---

```
1 (require 2htdp/image)
2 (define (get-pix L w h x y)
3     (if (or (< x 0) (>= x w)) (color 0 0 0)
4         (if (or (< y 0) (>= y h)) (color 0 0 0)
5             (list-ref L (+ (* y w) x)))
6         )))
7 (define (filter-color-list L pix-fun x y w h)
8     (if (= y h) empty
9         (let* ([tmpx (add1 x)]
10            [ny (if (= tmpx w) (add1 y) y)])
11            [nx (if (= tmpx w) 0 (add1 x))])
12            (cons (pix-fun L x y w h) (filter-color-list L pix-fun nx ny w h)))
13        )))
14 (define (filter-image img pix-fun)
15     (let ([cl (image->color-list img)])
16        [w (image-width img)])
17        [h (image-height img)])
18        (color-list->bitmap (filter-color-list cl pix-fun 0 0 w h) w h)
19    ))
20 (define (f L x y w h)
21     (let ([a (get-pix L w h (sub1 x) y)]
22       [b (get-pix L w h x (sub1 y))])
23       [c (get-pix L w h (add1 x) y)])
24       [d (get-pix L w h x (add1 y))])
25       [e (get-pix L w h x y)])
26     (color (round (/ (+ (color-red a) (color-red b) (color-red c)
27           (color-red d) (color-red e)) 5))
28           (round (/ (+ (color-green a) (color-green b) (color-green c)
29             (color-green d) (color-green e)) 5))
30           (round (/ (+ (color-blue a) (color-blue b) (color-blue c)
31             (color-blue d) (color-blue e)) 5))
32        )))
33 (filter-image (bitmap "/home/n/fun-motif.png") f)
```

## Question 12

---

Définir une fonction  $f$  permettant de réaliser un seuillage sur une image ; pour chaque pixel  $p$  de coordonnées  $(x \ y)$  on a une couleur résultat  $r$  égale à 0 si la composante rouge de  $p$  est supérieure à 70 et sinon on a une couleur résultat  $r$  égale à 255 ; définir la fonction de filtrage  $f$  retournant une couleur en fonction de deux paramètres  $x$  et  $y$ .

Avec `(filter-image (bitmap "/home/n/nuages.png") f)`, on obtient la figure 53 par application du seuillage sur la figure 52.



Fig. 52 – Image initiale.



Fig. 53 – Image seuillée.

Le **seuillage d'image** est une technique simple de **binarisation d'image**<sup>1</sup>, elle consiste à transformer une image en **niveau de gris** en une image dont les valeurs de pixels ne peuvent avoir que la valeur 1 ou 0. On parle alors d'une **image binaire** ou **image en noir et blanc**.

## Solution 12

---

```
1 (require 2htdp/image)
2 (define (f L x y w h)
3   (if (> (color-red (get-pix L w h x y)) 80)
4     (color 0 0 0)
5     (color 255 255 255)))
6 (filter-image (bitmap "/home/n/nuages.png") f)
```

## Question 14

---

Réaliser le dessin présenté ci-dessous en figure 58, dans lequel :

- Le fond est une `empty-image`.
- Deux courbes sont tracées.
- Il y a une continuité dans le tracé des deux courbes.

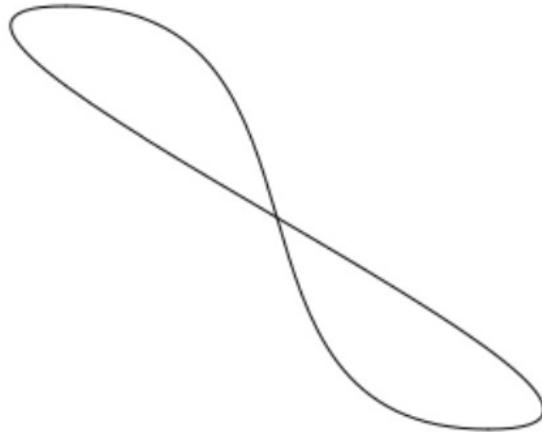


Fig. 58 – Deux courbes avec continuité.

## Question 18

---

Réaliser le dessin présenté ci-dessous en figure 62, en définissant une fonction  $f$  qui trace des cercles cocentriques avec des valeurs décroissantes de rayon (avec un rayon minimal de 10).

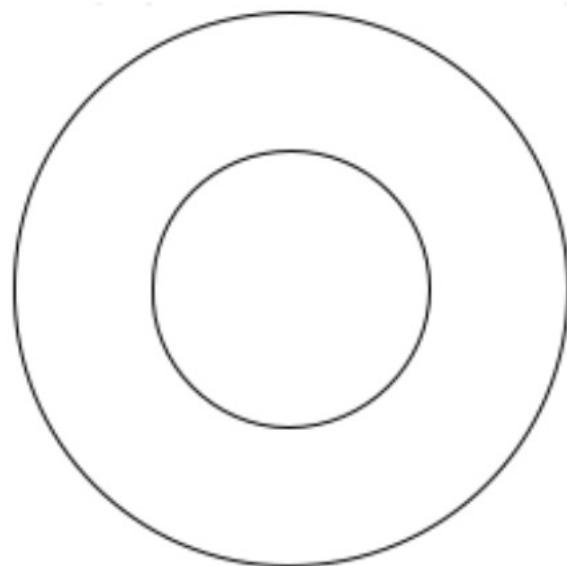


Fig. 61 – Deux cercles cocentriques.

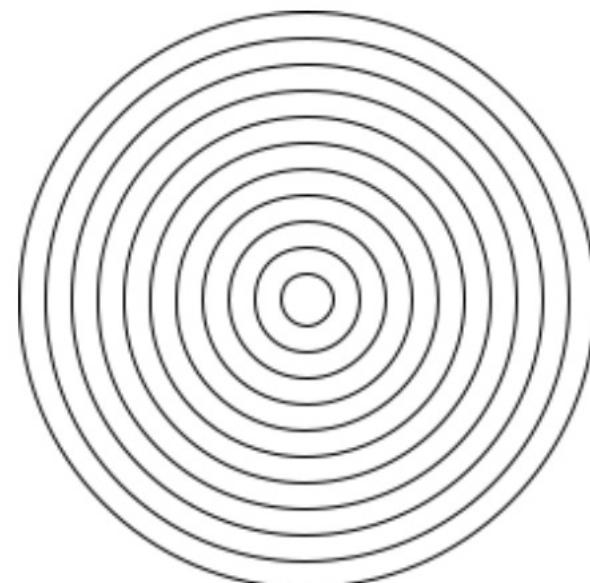


Fig. 62 – ( $f$  110).

## Question 24

---

Réaliser le dessin présenté ci-dessous en figure 68, dans lequel :

- On dessine un carré avec des cercles,
- Les cercles sont de rayon 10,
- Le côté du carré est composé de 11 cercles.

Réaliser le dessin présenté ci-dessous en figure 69, dans lequel :

- On dessine un cercle avec des carrés,
- Le cercle est de rayon  $40\sqrt{2}$ ,
- Les carrés sont de 20 sur 20,
- Le décalage angulaire entre les carrés est de 10 degrés.

Réaliser le dessin présenté ci-dessous en figure 70, dans lequel :

- On dessine un triangle avec des cercles,
- Les cercles sont de rayon 10.

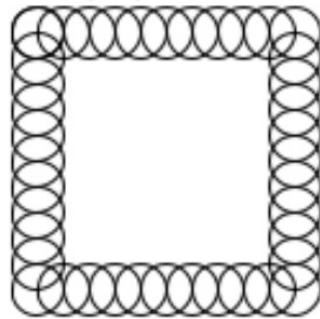


Fig. 68 – Carré de cercles.

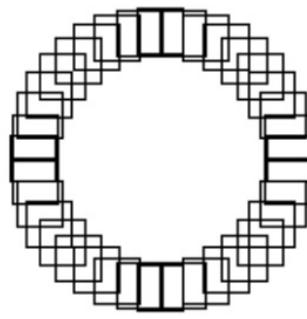


Fig. 69 – Cercle de carrés.

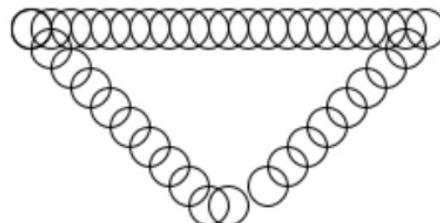


Fig. 70 – Triangle de cercles.