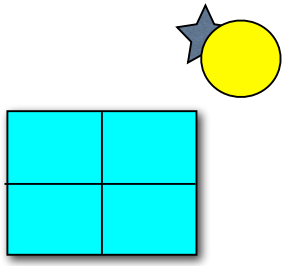
Université Nice Sophia Antipolis

Programmer avec des images



Livre PCPS, Chap. 3

- Racket propose des **teachpacks**, bibliothèques pour l'enseignement.
- Ce chapitre est une introduction au teachpack **2htdp/image**.
- Ce teachpack permet de construire des **images** composées de formes élémentaires, pour produire une **scène** statique.
- Nous pourrons dans le cours 4 programmer la **simulation** graphique **animée** d'un Monde virtuel. Une horloge sera automatiquement mise en place pour scander l'évolution du Monde, par exemple tous les 1/28 sec... Une animation n'est en effet pas autre chose qu'une suite d'images.
- Nous traitons donc ici la programmation sous un angle **multimédia** (la *sonorisation* sera peut-être traitée dans la seconde partie du cours.
- Le teachpack **image** est déjà intégré dans le teachpack **valrose** !

Images géométriques de base

- *Demandez la doc avec l'aide en ligne de DrRacket...*

```
> (rectangle 40 20 'solid "blue")
```

```
> (rectangle 40 20 'outline "blue")
```

```
> (circle 20 'solid "red")
```

```
> (ellipse 50 20 'outline "red")
```

```
> (line 50 10 "blue")
```

```
> (text "Hello !" 32 "red")
```
Hello !
```
> (star 40 'solid "green")
```


Les Couleurs

- Une couleur peut se représenter par une **chaîne de caractères** :

"red" "blue" "black" "green" "white" "gray" "yellow" *etc*

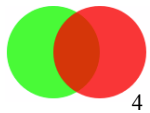
- ou par une structure représentant un **mélange RGB** avec la fonction (make-color r g b) qui attend des arguments dans [0,255] :

```
> (rectangle 50 20 'solid (make-color 0 0 255))
```


- La couleur **jaune** s'obtiendrait comme mélange de rouge et de vert :

```
(rectangle 50 20 'solid (make-color 255 255 0))
```


- Un quatrième argument optionnel représente la **transparence**. Il doit être choisi entre 0 (transparent) et 255 (opaque).

```
(underlay/xy (circle 50 'solid (make-color 0 255 0)) 50 0
             (circle 50 'solid (make-color 255 0 0 200)))
```


Le Rectangle qui permet de construire le fond d'une scène !

(rectangle *width* *height* *mode* *color*)

la largeur — la hauteur — 'solid ou 'outline — la couleur

Creates a width *by* height *rectangle, filled in according to* mode *and painted in color* color

(rectangle 50 20 'solid "yellow")

(rectangle 50 20 'outline "blue")

- A quoi sert **l'accent aigu** [on le prononce "**quote**"] ?

➡ A exprimer que l'expression qui suit ne doit pas être évaluée !

```
Welcome to DrRacket, version 5.3
> pi
#i3.141592653589793
> bonjour
ERROR : undefined identifier : bonjour
> 'bonjour
bonjour
> (+ 1 2)
3
> '(+ 1 2)
(+ 1 2)
```

Il croit que bonjour *est une variable et il cherche sa valeur !*

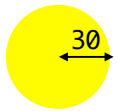
J'exprime que ce n'est pas une demande de calcul mais une donnée à l'état brut.

Le Cercle

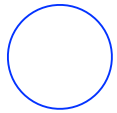
(circle *radius* *mode* *color*)

le rayon — 'solid ou 'outline — la couleur

Creates a circle or disk of radius radius*, filled in according to* mode *and painted in color* color



(circle 30 'solid "yellow")



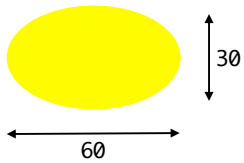
(circle 30 'outline "blue")

L'Ellipse

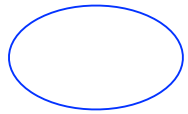
(ellipse *width* *height* *mode* *color*)

la largeur — la hauteur — 'solid ou 'outline — la couleur

Creates a width *by* height *ellipse, filled in according to* mode *and painted in color* color



(ellipse 60 30 'solid "yellow")



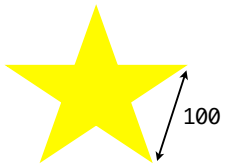
(ellipse 60 30 'outline "blue")

L'Etoile

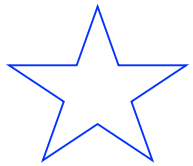
(star *side-length* *mode* *color*)

- *side-length* : longueur du côté
- *mode* : 'solid ou 'outline
- *color* : la couleur

Creates a multi-pointed star with 5 *points, the side-length argument determines the side length of the enclosing pentagon.*




```
(star 100 'solid "yellow")
```

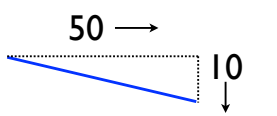


```
(star 100 'outline "blue")
```

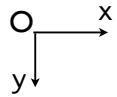

Le segment

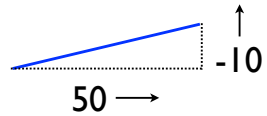
(line dx dy color)

- dx : longueur en x
- dy : longueur en y




```
(line 50 10 "blue")
⇳
(line -50 -10 "blue")
```

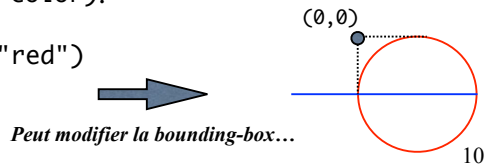





```
(line 50 -10 "blue")
⇳
(line -50 10 "blue")
```

- Pour ajouter à une image `img` un segment joignant $A(x_1,y_1)$ à $B(x_2,y_2)$, utilisez `(add-line img x1 y1 x2 y2 color)`.

```
(add-line (circle 40 'outline "red")
          -50 40 80 40
          "blue")
```



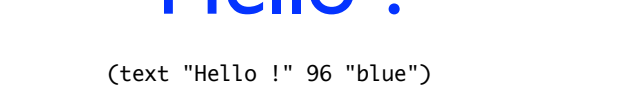
Peut modifier la bounding-box…

Le Texte

(text *string* *size* *color*)

- *string* : une chaîne de caractères
- *size* : taille de la police
- *color* : la couleur

Creates an image of the text `string` *at point size* `size` *and painted in color* `color`.



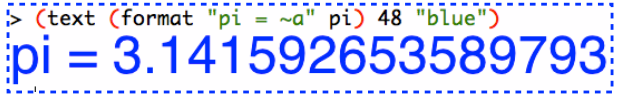

```
(text "Hello !" 96 "blue")
```


- La police de caractères n'est pas réglable. Seule la **taille** et la **couleur** le sont. Ou alors utilisez `text/font`...
- Le premier argument [le texte] doit être de type ***string***. Il peut être intéressant de **transformer un nombre en une chaîne de caractères** :

```
> (number->string 1789)
"1789"
> (number->string 1789 2)    ; en binaire [base 2]
"11011111101"
> (number->string 1789 16)   ; en hexadécimal [base 16]
"6fd"
```

- La fonction `(format str x1 x2 ...)` s'utilise de la même manière que `(printf str x1 x2 ...)` mais elle retourne une chaîne de caractères au lieu d'afficher !

```
> (text (format "pi = ~a" pi) 48 "blue")
```

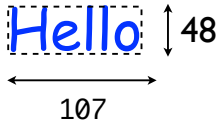


- On peut obtenir les **dimensions d'une image** :

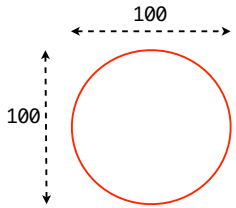
```
> (define T (text "Hello" 48 "blue"))
> T
```

Hello

```
> (image-width T)
107
> (image-height T)
48
```




```
> (define C (circle 50 'outline "red"))
> (image-width C)
100
> (image-height C)
100
```



- Une image est contenue dans un rectangle transparent : sa ***bounding box***.

Superposition centrée d'images [underlay]

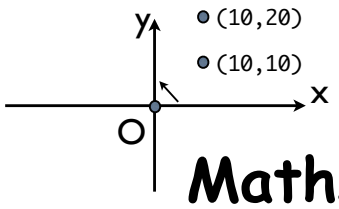
- La fonction `(underlay img1 img2 ...)` permet de combiner plusieurs images en une seule [img1 au-dessous de img2, etc] en les alignant par leurs centres.

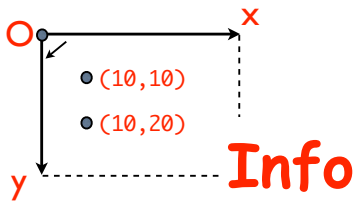
```
(underlay (rectangle 160 40 'solid "gray")
          (circle 10 'solid "black"))
```

- Comment construiriez-vous celles-ci ?...

Superposition relative [underlay/xy]

- ATTENTION : la plupart des langages de programmation graphiques n'utilisent pas les axes mathématiques usuels !





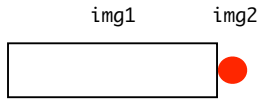
- La fonction `(underlay/xy img1 x y img2)` superpose img2 au-dessus de img1, mais en décalant img2 de x pixels vers la droite et de y pixels vers le bas par rapport au coin haut gauche de img1.

```
(underlay/xy (rectangle 160 40 'solid "gray")
             80 20
             (rectangle 160 40 'solid "black"))
```

Alignement horizontal centré [beside]

- La fonction `(beside img1 img2 ...)` permet de combiner plusieurs images en une seule [img1 à gauche de img2, etc] en les alignant par leurs centres.

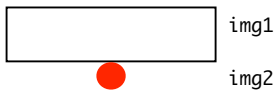
```
(beside (rectangle 160 40 'outline "black")
        (circle 10 'solid "red"))
```



Alignement vertical centré [above]

- La fonction `(above img1 img2 ...)` permet de combiner plusieurs images en une seule [img1 en haut de img2, etc] en les alignant par leurs centres.

```
(above (rectangle 160 40 'outline "black")
       (circle 10 'solid "red"))
```



Rotation d'une image [rotate]

• Il est possible de faire tourner une image d'un angle α exprimé en degrés avec la fonction `(rotate α img)`. La rotation aura lieu dans le sens inverse des aiguilles d'une montre.

r

`(rotate 45 r)`

Changement d'échelle d'une image [scale]

r

`(beside r (scale 2 r))`

Encadrement d'une image [frame]

• Chaque image est contenue dans un rectangle invisible qui l'entoure : sa bounding box. La fonction `(frame img)` permet de le visualiser !

r

`(frame (rotate 45 r))`

Extraction d'une sous-image [crop]

• La fonction `(crop x y larg haut img)` permet d'extraire la portion d'image de img, dont le coin haut gauche est à la position (x,y), et dont les dimensions sont larg et haut. C'est le **cropping** !

`c = (circle 40 'solid "red")`

`(crop 40 0 40 40 c)`

Le placement avec cropping [place-image]

• La fonction `(place-image img1 x y img2)` superpose img1 au-dessus de img2, mais en plaçant le centre de img1 au point (x ; y) dans le repère haut gauche de img2. En général, img2 est un rectangle. ***CROPPING*** !

```
(place-image (circle 100 'solid "black")
             60 60
             (rectangle 400 200 'solid "red"))
```

• Ne pas confondre avec `(underlay/xy img1 x y img2)` qui :
- place le **coin haut gauche de img1** par rapport au coin haut gauche de img2 alors que `place-image` s'occupe du **centre** de img1.
- ne fait **aucun cropping** et risque d'**agrandir la *bounding box***.

• ***Lorsque vous travaillez dans un cadre rectangulaire, utilisez toujours*** `place-image` ***et évitez*** `underlay/xy` ***qui agrandit la bounding box !***

• ***La variante*** `scene+line` ***de*** `add-line` ***fait un cropping automatique !***

Utiliser de véritables images au format PNG

• Il est souvent intéressant d'ajouter des images Scheme à un fond en provenance d'une véritable image (photo, Web, etc). **Le format PNG est conseillé**, mais GIF et JPG sont aussi admis...

• Le fichier "ballon.png" est dans le répertoire courant de mon disque dur. Je le **charge** en mémoire sous la forme d'une image Racket :

```
(define ballon (bitmap "ballon.png"))
```

et maintenant je peux l'utiliser pour d'autres images :

```
> (above ballon (beside ballon ballon))
```

• Inversement je peux **sauver** une image sur le disque au format PNG :

```
(save-image taz "taz.png")
```

A vous de faire de belles images !

`(bitmap "taz.png")`

Quelques images des années passées (BONUS)