

PICO-8 / LIB_MATH / L1

Préparation

- Lancer PICO-8

La touche [Echap] permet de basculer entre l'invite de commandes et l'éditeur.

La séquence de touches [Ctrl]-[R] exécute le programme situé dans l'éditeur.

- Télécharger la librairie : http://pico8scope101.fr/downloads/lib_math.p8 dans le dossier des cartouches PICO-8 (accessible à l'aide de la commande `folder`).

La commande `reboot` permet de relancer PICO-8.

Programmation

Ligne de code à mettre au début du programme pour charger librairie (LIB_MATH.P8) :

```
#include lib_math.p8
```

Pour effacer l'écran, il faut utiliser la commande `CLS`. Dans un programme, il faut faire suite la commande d'une parenthèse ouvrante et fermante.

Les instructions d'un programme s'exécutent dans l'ordre.

Programme 1 : Effacer l'écran et afficher un point (au milieu)

```
#include lib_math.p8
cls()
m:point(0,0):trace(7)
```

➔ `m:point` crée un point en (0,0)

➔ `<un point>:trace(7)` affiche le point à l'écran (avec la couleur 7 = blanc). Il y a 16 couleurs numérotées de 0 à 15.

Il est possible de stocker un point dans une variable (ex. `mon_point`).

```
#include lib_math.p8
cls()
mon_point=m:point(0,0)
mon_point:trace(7)
```

Programme 2 : Effacer l'écran et afficher un segment (défini par 2 points)

Il est possible de stocker un segment dans une variable (ex. `mon_segment`).

```
#include lib_math.p8
cls()
pointa=m:point(-50,0)
pointb=m:point(50,0)
mon_segment=m:segment(pointa,pointb)
mon_segment:trace(8)
```

Programme 3 : Effacer l'écran et afficher une figure géométrique (un carré)

Il est possible de stocker un point dans une variable (ex. CARRE). La déclaration d'une figure autorise l'emploi d'une notation simplifiée pour la déclaration des points : $\{-10,-10\}$ est équivalent à `m:point(-10 , -10)`.

```
#include lib_math.p8
cls()
carre=m:figure({
  points={
    {-10,-10},
    {-10,10},
    {10,10},
    {10,-10}
  },
  segments={
    {1,2},
    {2,3},
    {3,4},
    {4,1}
  }
})

carre:trace(9)
```

Programme 4 : Faire tourner le carré (de 10°)

Reprendre le programme 3 et remplacer la dernière ligne par :
`carre:tourne(45):trace(9)`

Programme 5 : Déplacer le carré (de 10 pixels vers la droite et de 30 pixels vers le haut = vecteur, noté $\{10,30\}$)

Reprendre le programme 3 et remplacer la dernière ligne par :
`carre:deplace({10,30}):trace(9)`

Programme 6 : Zoomer le carré (x2)

Reprendre le programme 3 et remplacer la dernière ligne par :
`carre:zoome(2):trace(9)`

Programme 7 : Produire un clone du carré pour chacune des actions des programme 4 à 6

Reprendre le programme 3 et ajouter après la dernière ligne :

```
carre4=m:clone(carre)
carre4:tourne(45):trace(9)
carre5=m:clone(carre)
carre5:deplace({10,30}):trace(9)
carre6=m:clone(carre)
carre6:zoome(2):trace(9)
```

Programme 8 : Faire un clone du carré auquel est appliquée une symétrie axiale par rapport à la droite $y=x-20$ ($y=a*x+b$ avec $a=1$ et $b=-20$, noté $\{1,-20\}$).

Reprendre le programme 3 et ajouter après la dernière ligne :

```
carre:trace(9)
carre2=m:clone(carre)
carre2:symayb({1,-20}):trace(9)
```

Programme 9 : Faire un clone du carré auquel est appliquée une symétrie axiale par rapport à la droite $x=y+20$ ($x=a*y+b$ avec $a=1$ et $b=20$, noté $\{1,20\}$).

Reprendre le programme 3 et ajouter après la dernière ligne :

```
carre:trace(9)
carre2=m:clone(carre)
carre2:symaxb({1,20}):trace(9)
```

Les résultats obtenus pour les programmes 8 et 9 sont identiques. En effet, il s'agit du même axe de symétrie, car $y=a*x+b \Rightarrow x=(y-b)/a$ si a non nul).

Programme 10 : Faire tourner un carré

```
#include lib_math.p8
carre=m:figure({
  points={
    {-10,-10},
    {-10,10},
    {10,10},
    {10,-10}
  },
  segments={
    {1,2},
    {2,3},
    {3,4},
    {4,1}
  }
})

function _draw()
  cls()
  carre:tourne(4):trace(9)
end
```

➔ La fonction `_draw` est appelée 30 fois par seconde.

Programme 11 : Changement du centre de rotation

Reprendre le programme 10, supprimer la fonction `_draw` (et son contenu) et ajouter :

```
carre:deplace({0,30})
function _draw()
  cls()
  carre
  :centre({0,0})
  :tourne(-1)
```

```

:centre()
:tourne(5)
:trace(9)
end

```

- ➔ centre ({0,0}) déplace le point de rotation au point (0,0)
- ➔ centre() déplace le point de rotation au centre initial
- ➔ Le centre initial correspond au point (0,0) lors de la déclaration de la figure après l'ensemble des actions (déplacement, symétrie, homothétie).

Programme 12 : Faire tourner un carre pendant 5s dans un sens puis 5s dans le sens contraire

Reprendre le programme 10, supprimer la fonction `_draw` (et son contenu) et ajouter :

```

i=0
function _draw()
  cls()
  if i<300 then
    i+=1
  else
    i=0
  end
  if i<150 then
    carre:tourne(4):trace(9)
  else
    carre:tourne(-4):trace(9)
  end
end
end

```

Programme 13 : Même chose en plus simple (vraiment ?)

Reprendre le programme 10, supprimer la fonction `_draw` (et son contenu) et ajouter :

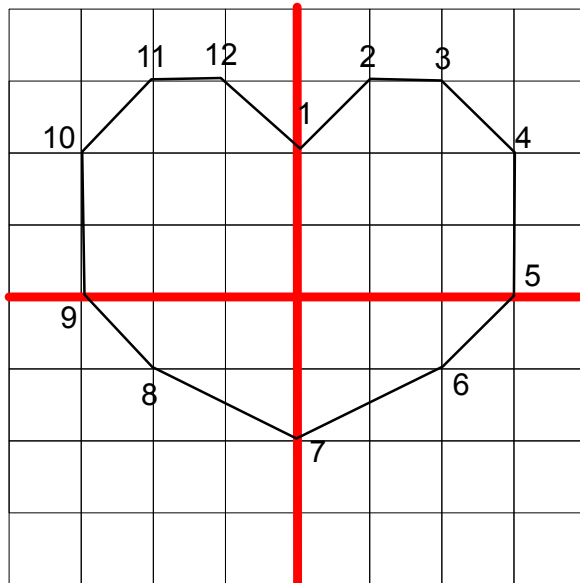
```

function _draw()
  cls()
  if (t()%10)<5 then
    carre:tourne(4):trace(9)
  else
    carre:tourne(-4):trace(9)
  end
end
end

```

- ➔ T() renvoie le nombre de secondes écoulées depuis le lancement du programme
- ➔ $a \% b$ = reste de la division entière de a / b ; permet de filtrer la valeur de a à la plage $0 .. |b-1|$.

Programme 14 : Reproduire un dessin



Liste des points :

- 1 : {0,20}
- 2 : {10,30}
- 3 : {20,30}
- 4 : {30,20}
- 5 : {30,0}
- 6 : {20,-10}
- 7 : {0,-20}
- 8 : {-20,-10}
- 9 : {-30,0}
- 10 : {-30,20}
- 11 : {-20,30}
- 12 : {-10,30}

```
#include lib_math.p8
coeur=m:figure({
  points={
    {0,20},
    {10,30},
    {20,30},
    {30,20},
    {30,0},
    {20,-10},
    {0,-20},
    {-20,-10},
    {-30,0},
    {-30,20},
    {-20,30},
    {-10,30}
  },
  segments={
    {1,2},
    {2,3},
    {3,4},
    {4,5},
    {5,6},
    {6,7},
    {7,8},
    {8,9},
    {9,10},
    {10,11},
    {11,12},
    {12,1}
  }
})

cls()
```

```
coeur:trace(14)
```

Programme 15 : Animer le dessin

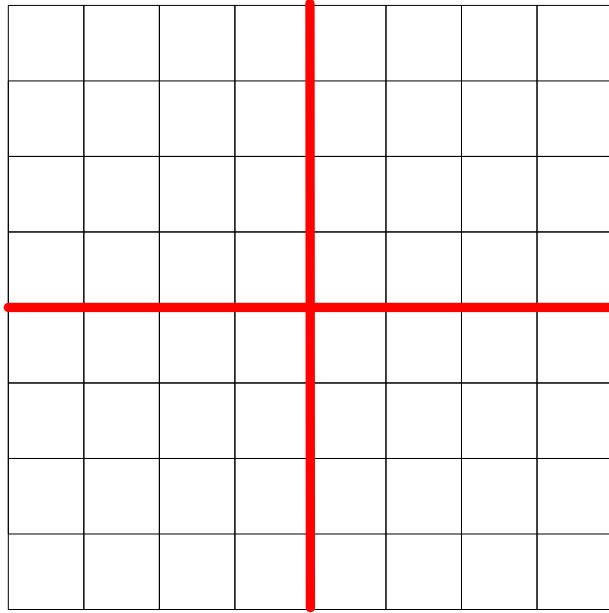
Exemple :

```
#include lib_math.p8
coeur=m:figure({
  points={
    {0,20},    {10,30},
    {20,30},    {30,20},
    {30,0},    {20,-10},
    {0,-20},    {-20,-10},
    {-30,0},    {-30,20},
    {-20,30},    {-10,30}
  },
  segments={
    {1,2},    {2,3},
    {3,4},    {4,5},
    {5,6},    {6,7},
    {7,8},    {8,9},
    {9,10},    {10,11},
    {11,12},    {12,1}
  }
})

coeurs={}
for i=1,12 do
  add(coeurs,m:clone(coeur))
  coeurs[i]
    :zoome(.2)
    :deplace(coeur.points[i])
    :centre({0,0})
end

function _draw()
  cls()
  coeur:tourne(1):trace(8)
  for i=1,12 do
    coeurs[i]
      :tourne(1)
      if i%2==flr(t()%2) then
        coeurs[i]:trace(14)
      end
    end
  end
end
```

Programme 16 : Créer votre propre dessin à animer



PICO-8

Langage de programmation (LUA)

Les données manipulées peuvent être des nombres (ex. 25, 8.4), des portions de texte (ex. "EXEMPLE"), des listes (ex. { }). Les listes peuvent contenir tous les types de données : des nombres, des portions de texte, d'autres listes.

- La présence d'apostrophes ou de guillemets situés de part et d'autre d'une expression indique que celle-ci est une portion de texte : **25** (nombre), **' 25 '** ou **"25"** (portion de texte).
- La présence d'une accolade ouvrante et fermante respectivement situées de part et d'autre d'une expression indique que celle-ci est une liste : **{1, 2, 3, 4}** (liste de nombre), **{ 'un', 'deux', 'trois' }** (liste de portion de texte), **{1, "deux", 3, 4}** (liste mixte).

Les données peuvent être stockées dans des variables. Le nom de la variable est alors utilisé en lieu et place de la donnée. Il est possible de modifier le contenu d'une variable :

- **nombre=18**
- **jour='jeudi'**
- **liste_vide = {}**

Il est possible d'utiliser l'instruction **print** pour afficher à l'écran un nombre, une portion de texte ou le contenu d'une variable :

- **print(10)**
- **print('exemple')**
- **print(nombre)**

Principales instructions

cls()

- Efface l'écran.

print(donnée, variable ou expression)

- Affiche à l'écran la *donnée*, le contenu de la variable ou le résultat du calcul de l'expression.

variable = valeur

- AFFECTE la valeur à la variable (à ne pas confondre avec l'égalité booléenne ==).

donnée1 .. donnée2

- Si *donnée1* est une portion de texte alors concatène donnée2 à donnée1 SINON renvoie une erreur, ex. **nom='pierre' print('bonjour ' .. nom)**

flr(donnée)

- Renvoie le nombre entier inférieur le plus proche de *donnée* (ex. **print(flr(15.4))** renvoie 15, **print(flr(-15.4))** renvoie -16)

t()

- Renvoie le nombre de secondes passées à exécuter le programme.

if condition then traitement1 else traitement2 end

- Si *condition* est vrai ALORS exécute le traitement1 SINON exécute le traitement2 FIN

for variable = début, fin [,pas] do traitement end

- POUR *variable* = *début* jusqu'à *fin* [de *pas* en *pas*] FAIRE *traitement* FIN.

add(liste, élément)

- AJOUTE l'*élément* à la *liste*.

#liste

- Renvoie le nombre d'éléments de la *liste*.

liste[donnée]

- Renvoie l'élément à la position *donnée* de la *liste* (ou **nil** s'il n'existe pas). Le premier élément d'une liste porte le numéro 1.

liste.élément

- Renvoie l'*élément* de la liste, ex. **personne = { nom = 'popeye', emploi='marin' } print(personne.nom .. ' est un ' .. personne.emploi .. '.')**

donnée1 % donnée2

- Renvoie le modulo = reste de la division entière de *donnée1* par *donnée2* (ex. 1%3 renvoie 1, 2%3 renvoie 2, 3%3 renvoie 0)

variable += donnée

- Ajoute la *donnée* à la *variable*, ex. **nombre=5 nombre+=1 print('six=' .. nombre)**

function _draw() traitement end

- Exécute le *traitement* 30 fois par seconde (utile pour les animations)

Exemples

Afficher les nombres d'une liste

```
cls()
print('nombres premiers inferieurs a 10:')
premiers={1,3,5,7}
for i=1,#premiers do
  print(premiers[i])
end
```

Afficher les nombres pairs inférieurs ou égaux à 10 (pair ⇔ multiple de 2 ⇔ reste de la division entière par 2 = 0)

```
cls()
print('Nombres pairs <= 10:')
for i=0,10 do
  if i%2==0 then
    print(i)
  end
end
```

Afficher les nombres pairs inférieurs ou égaux à 10 (version bison futé)

```
cls()
print('nombres pairs <= 10:')
for i=0,10,2 do
  print(i)
end
```

Afficher le nombre de secondes écoulées depuis le lancement du programme

```
function _draw()
  cls()
  print(flwr(t()))
end
```

Afficher un message déroulant

Attention : dans un texte, le caractère « \ » doit être doublé pour pouvoir être affiché.

```
message={
  texte={
    '- ',
    '\\ ',
    '| '
  },
  vitesse=0.2
}
add(message.texte, '/')
i=1
function _draw()
  cls()
  i=(i+message.vitesse)%4
  print(message.texte[flwr(i)+1])
end
```

end