ФУНКЦІОНАЛЬНЕ ПРОГРАМУВАННЯ

Лекція 9

РЕКУРСИВНЕ ПРОГРАМУВАННЯ ГРАФІКИ

2020

Повний текст лекції буде розміщений на сайті baklaniv.at.ua

Рейтінг THIOBE

30	Kotlin
31	Logo
32	Lisp
33	F#
34	Fortran
35	Lua
36	Ada
37	Prolog
38	VBScript
39	LabVIEW
40	Apex
41	Haskell
42	TypeScript
43	PowerShell
44	ML
45	RPG
46	Scheme
47	Erlang
48	Groovy
49	Bash
50	Julia

Apr 2020	Apr 2019	Change	Programming Language
1	1		Java
2	2		C
3	4	^	Python
4	3	v	C++
5	6	^	C#
6	5	v	Visual Basic
7	7		JavaScript
8	9	^	PHP
9	8	v	SQL
10	16	^^	R
11	19	^^	Swift
12	18	^^	Go
13	13		Ruby
14	10	vv	Assembly language
15	22	^^	PL/SQL
16	14	v	Perl
17	11	vv	Objective-C
18	12	vv	MATLAB
19	17	v	Classic Visual Basic
20	27	^^	Scratch

Position	Programming Language
21	SAS
22	Delphi
23	Dart
24	Transact-SQL
25	D
26	COBOL
27	Rust
28	Scala
29	ABAP

РЕКУРСИВНА ГРАФІКА. Racket Turtle

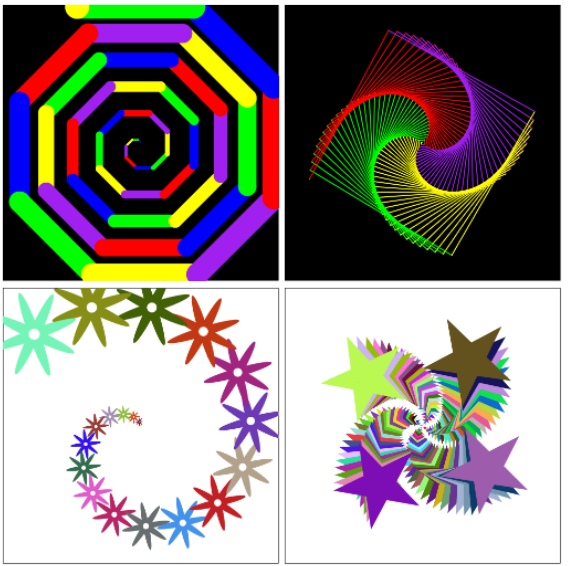
Racket Turtle - бібліотека забезпечує простий інтерфейс для малювання традиційної графіки черепах. Racket Turtle має також функціональність штампів, тому її зображення не обмежуються лише лінійними малюнками.

Racket Turtle була розроблена для навчання програмування та геометричних концепцій для учнів середньої школи, але вона також може бути використана для викладання більш прогресивних концепцій програмування, таких як списки та рекурсії.

Для **Racket** (require teachpacks/racket-turtle)

Для **Wescheme** (require wescheme/Engb0r37Kg)

Зображення, намальовані за допомогою ракетки-черепахи:



1 Основи Racket Turtle

За допомогою Racket Turtle - бібліотеки ви можете намалювати традиційну графіку черепахи, наказавши черепашці (чорний трикутник) рухатися вперед, повернути ліворуч, підняти перо вгору, відкласти його і т. д. Крім цього ви можете намалювати штампи в черепаху положення, дзеркальні рухи черепахи вертикально і горизонтально або наказуйте черепашці рухатися за певними координатами. Ви можете зробити свої зображення більш цікавими, змінивши колір ручки, ширину та тип лінії, також колір тла та зображення можна встановити під свої вподобання.

Для наказу черепашці потрібно створити список команд черепахи. Цей список подається як аргумент функції малювання, яка читає команди та малює зображення. Вибравши іншу функцію малювання, ви можете змінити розмір вікна анімації та швидкість малювання, зберегти анімацію як анімований **gif** або намалювати зображення поетапно.

Переважно визначити список команд і дати йому хороше визначальне ім'я. Найпростіший список команд містить одну команду та її аргументи, записані всередині дужок.

Простий приклад

Цей список команд малює одну синю лінію довжиною 100 пікселів від середини екрана анімації до верхньої частини екрана (початкова точка черепахи знаходиться в середині екрана в (250, 250) і вона спрямована вгору) .

Визначення рядка списку команд line1 для малювання рядка (у вікні визначень):

```
(define line1 (list (forward 100)))
```

Виклик функції малювання draw з line1 у якості аргумента (у вікні взаємодії):

```
(draw line1)
```

Усі команди та функції малювання Racket Turtle представлені на наступних слайдах.

2 Команди Racket Turtle

(forward x) → procedure

x : real?

Пересуває черепаху вперед на x пікселів, якщо x є позитивним, назад - якщо негативним.

(turn-left a) → procedure

a : real?

Повертає голову черепахи вліво на градуси, якщо **a** - позитивний. Негативний **a** змусить черепаху повернути праворуч **(abs a)** градусів.

(turn-right a) → procedure

a : real?

Повертає голову черепахи вправо на градуси, якщо дане значення є позитивним. Негативний **a** змусить черепаху повернути ліворуч **(abs a)** градусів.

(repeat k command-list) →
procedure

k : (and/c integer? positive?)
command-list : (list-of procedure)

Змушує черепаху повторювати **k** рази команд, поданих у **command-list**.

(pen-up) → procedure

Піднімає ручку вгору, щоб черепаха не малювала лінію при русі. Позиція за замовчуванням знижена.

(pen-down) → procedure

Опустіть ручку так, щоб черепаха малювала лінію під час руху.

(go-to x y) → procedure

x : real?
y : real?

Змушує черепаху йти до заданої точки **(x, y)**. Походження зображення черепахи знаходиться в лівому нижньому куті, якщо воно не було переміщено.

(go-to-origin) → procedure

Змушує черепаху перейти до початку вікна анімації. Походження знаходиться в лівому нижньому куті, якщо воно не переміщено

(change-color color) → pro
cedure

color : image-color?
(change-color color-list) → proced
ure
color-list : (list-of image-color?)

Змінює колір ручки (колір за замовчуванням - синій). Аргументом може бути один колір або список кольорів, який містить один або більше кольорів. Кольори у списку кольорів використовуються по одному, і коли список закінчений, він починається з початку.

(change-pen-size width) →
procedure

width : (and/c integer? (<= 0 255)
)

Встановлює ширину ручки. Ширина - це ціле число між 0 - 255.

Примітка! *WeScheme це не підтримує.*

(change-pen-style style) →
procedure

style : pen-style?

Змінює стиль ручки. Стиль може бути одним із таких: "твердий", "крапка", "довгий тире", "короткий тире" або "крапка".

Примітка! *WeScheme не підтримує це.*

(change-bg-color color) →
procedure

color : image-color?

Змінює колір фону (колір за замовчуванням - білий). Цю команду слід використовувати спочатку, оскільки вона заповнює все зображення заданим кольором.

(set-bg-image img) → proce
dure

img : image?

Встановлює заданий **img** зображення як фонове зображення. Цю команду слід використовувати перед малюванням фактичного зображення, оскільки **img** розміщено поверх існуючих малюнків черепахи.

(set-bg-grid width height color
) → procedure

width : (and/c integer? positive?)
height : (and/c integer? positive?)
color : image-color?

Малює фонову сітку із заданою шириною, висотою та кольором комірки. Цю команду слід використовувати перед тим, як намалювати фактичні малюнки черепахи.

(set-origin) → procedure

Зберігає поточне положення черепахи як нового походження. Місце походження відображається у вигляді червоної точки під час анімації черепах.

(stamper-on stamp) → proce
dure

stamp : image?
(stamper-on stamp-list) → procedur
e
stamp-list : (list-of image?)

Активує штамп. Штампер малює по одному штампу в кожному новому місці черепахи. Якщо аргументом є список зображень, штампи списків штампів використовуються одна за одною відповідно до списку. Після того, як всі зображення штампу були використані один раз, список починається знову з початку.

Штамп не впливає на ручку, тому, якщо ручка знищена, в доповненнях до марок буде намальована лінія.

(stamper-off) → procedure

Знімає штамп.

(mirror-x-on) → procedu re

Копіює команди черепахи так, щоб вони відображалися горизонтально, наприклад крім оригінального зображення буде друге дзеркальне зображення. Положення осі дзеркального відображення встановлюється як вертикальна лінія, **y**-координата якої взята з точки, в якій було використано дзеркальне відображення **x-on**. Якщо дзеркальне відображення використовується в тому самому місці, дзеркальне відображення буде здійснено також вертикально.

(mirror-y-on) → procedure

Копіює команди черепахи так, щоб вони відображалися вертикально, наприклад крім оригінального зображення буде друге дзеркальне зображення. Положення осі дзеркального відображення встановлюється горизонтальною лінією, **x**-координата якої взята з точки, в якій було використано дзеркальне відображення. Якщо дзеркало-**x-on** використовується в тому самому місці, дзеркальне відображення буде здійснено також горизонтально.

(mirror-x-off) → procedure

Знімає горизонтальне дзеркальне відображення.

(mirror-y-off) → procedure

Знімає вертикальне дзеркальне відображення.

(clean-up) → procedure

Очищує екран анімації, включаючи фонове зображення та сітку, але зберігає колір тла. Також зберігаються місце розташування та заголовок черепахи.

(hide-turtle) → procedure

Приховує спрайт черепашки протягом тривалості анімації (черепаха видно за замовчуванням). Спрайт черепахи - це чорний трикутник, що показує місце розташування та поточний заголовок черепахи.

(show-turtle) → procedure

Знову показує спрайт-черепаху під час анімації. Спрайт черепахи - це чорний трикутник, що показує місце розташування та поточний заголовок черепахи.

3 Функції малювання для Racket Turtle

(draw command-list) → image?

command-list : (list-of procedure)

Читає та виконує команди у списку команд та повертає створене зображення. Відкриває вікно анімації та показує черепаху в дії.

(draw-custom command-list width height speed) → image?

command-list : (list-of procedure)
width : (and/c positive? real?)
height : (and/c positive? real?)
speed : (and/c positive? real?)

Встановлює розмір вікна анімації на ширину та висоту, з швидкістю малювання. Якщо для швидкості встановлено значення 0, використовується швидкість за замовчуванням. Розмір за замовчуванням вікна анімації становить 500 x 500 пікселів.

(draw-step-by-step command-list) → image?

command-list : (list-of procedure)

Як і **draw**, але список команд обробляється поетапно. Кожен крок виконується після того, як користувач натиснув пробіл. Зауважте, що зміна, наприклад, кольору пера на черепаху, - це один крок, тому конус не потрібно переміщувати кожного разу.

(draw-step-by-step-custom command-list width height speed) → image?

command-list : (list-of procedure)
width : (and/c positive? real?)
height : (and/c positive? real?)
speed : (and/c positive? real?)

Як і **draw-custom**, але список команд обробляється поетапно. Кожен крок виконується після того, як користувач натиснув пробіл. Зауважте, що зміна, наприклад, кольору пера на черепаху, - це один крок, тому конус не потрібно переміщувати кожного разу.

(draw-and-store command-list) → image?

command-list : (list-of procedure)

Як і **draw**, але зберігає анімований **gif** анімації у папці під назвою **turtle_animations**. Ця папка повинна розташовуватися в тому ж шляху, що і файл **.rkt**. Якщо цієї папки немає, анімований **gif**-файл не створюється. Зауважте, що створення файлу займає певний час, тому не закривайте вікно анімації, перш ніж текст "Створення анімованих **gif**" зникне, а остаточне зображення відобразиться знову.

Примітка! *WeScheme не підтримує це.*

```
(draw-and-store-custom comman
                       d-list
                       width
                       height
                       speed)  →image?
  command-list : (list-of procedure)
  width : (and/c positive? real?)
  height : (and/c positive? real?)
  speed : (and/c positive? real?)
```

Як і **draw**, але зберігає анімацію як анімований **gif**-файл.

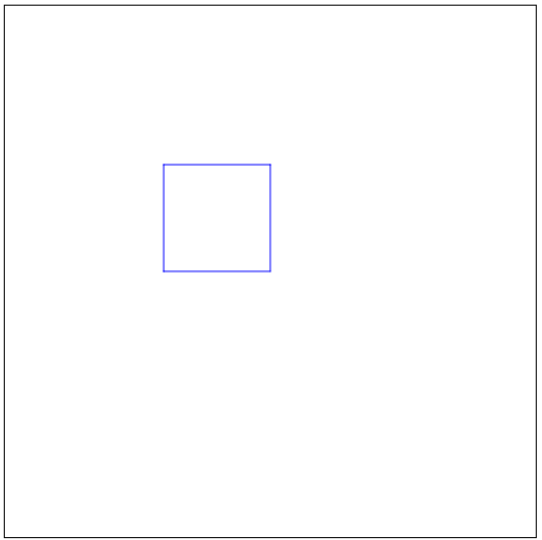
Примітка! *WeScheme не підтримує це.*

4 Приклади

4.1 Малювання квадрата

Спочатку потрібно визначити список команд черепахи для малювання квадрата. Назвемо його **square1**. Фактичне малювання зображення черепахи робиться за допомогою функції **draw**, список команд черепахи подається як аргумент для **draw**.

```
(define square1
  (list (forward 100)
        (turn-left 90)
        (forward 100)
        (turn-left 90)
        (forward 100)
        (turn-left 90)
        (forward 100)))

(draw square1)
```



4.2 Малювання квадрату за допомогою **repeat**

Оскільки список команд для малювання квадрата має повторюваний малюнок, то краще визначити його окремо і використовувати його повторно. Ми назвемо його стороною і повторимо її 4 рази, використовуючи repeat.

```
(define side
  (list (forward 100)
        (turn-left 90)))

(define repeat-square
  (repeat 4 side))

(draw repeat-square)
```

4.3 Малювання двох квадратів на одному малюнку.

Ми можемо використовувати раніше визначений квадрат і намалювати два з них на одній картині. Ми визначимо список команд **move** для переміщення на нове місце та змінимо колір пера перед малюванням другого квадрата.

```
(define move
  (list (pen-up)
        (turn-right 90)
        (forward 100)
        (pen-down)
        (change-color "red")))

(define two-squares
  (list square1
        move
        square1))

(draw two-squares)
```

4.4. Малювання квадрата, використовуючи функцію

Щоб мати можливість малювати квадрати зі зміною довжини сторони, нам потрібно визначити функцію, яку ми назвемо **changing-square**. Також нам потрібна допоміжна функція, що змінюється, щоб намалювати сторони зі зміною довжини. Змінна **x** - бічна довжина. Нарешті, ми називаємо нову функцію, що **changing-square**, аргументом 30 (x = 30).

```
(define (changing-side x)
  (list (forward x)
        (turn-left 90)))

(define (changing-square x)
  (repeat 4 (changing-side x)))

(draw (changing-square 30))
```

4.5 Малювання квадрата, використовуючи координати

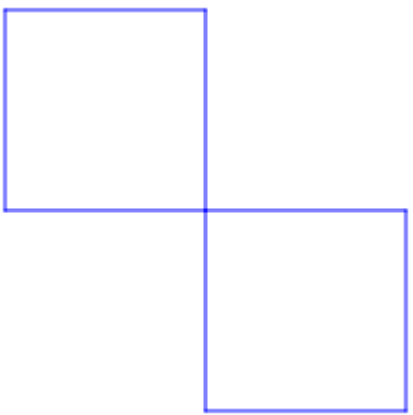
Ви також можете намалювати квадрат, наказавши черепашці пройти через деякі точки координатної площини за допомогою команд **go-to**. Для отримання більш легких координат ми змінюємо місце початку за допомогою команди **set-origin**.

```
(define coordinate-square
  (list (set-origin)
        (go-to 100 0)
        (go-to 100 100)
        (go-to 0 100)
        (go-to 0 0)))

(draw coordinate-square)
```

4.6 Відзеркалення квадрату

Ми можемо намалювати два квадрати, щоб другий був створений дзеркальним відображенням першого за допомогою точки зсуву. Точка повороту - це місце, де було включено дзеркальне відображення (**mirron-x-on, mirron-y-on**), у цьому прикладі це вихідна точка.

```
(define mirroring-square
  (list (mirror-x-on) (mirror-y-on)
        square1))

(draw mirroring-square)
```



4.7 Малювання квадрату, вікористовуючи штамп

Ми можемо активувати функцію штампування за допомогою команди **stamper-on**. Тепер черепаха буде «тиснути» заданим зображенням після кожного руху. Тут ми використовуємо червоний круг як штамп. Ви також можете дозволити ручці також намалювати лінію (ось вона взята).

```
(define STAMP (circle 5 "solid" "red"))

(define stamper-square
  (list (stamper-on STAMP)
        (pen-up)
        square1))

(draw stamper-square)
```

4.8 Зміна стиля і розміру ручки

Ви можете змінити стиль рядка, використовуючи команду **change pen-style** та ширину лінії за допомогою команди **change-pen size**.

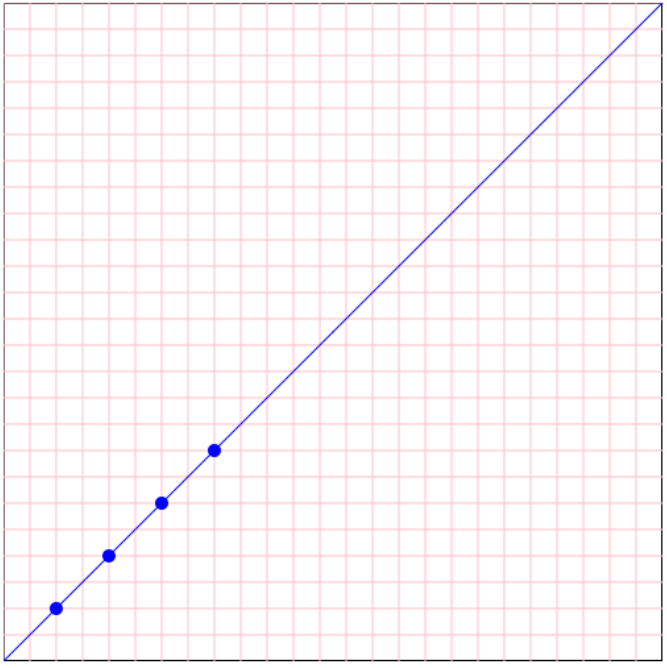
Примітка! *Це не працює в WeScheme.*

```
(define special-pen-square
  (list (change-pen-size 5)
        (change-pen-style "dot")
        square1))

(draw speacial-pen-square)
```

4.9 Малювання лінії на сітці

Спочатку ми малюємо фонову сітку за допомогою **set-bg-grid**. Переходимо до початку, використовуючи **(go-to 0 0)** і активуємо штамп синім колом як штамп. Для отримання штампів у правильних координатах нам потрібно командувати черепашкою, використовуючи кілька разів кілька разів. У цьому прикладі штамп вимкнено після чотирьох точок.

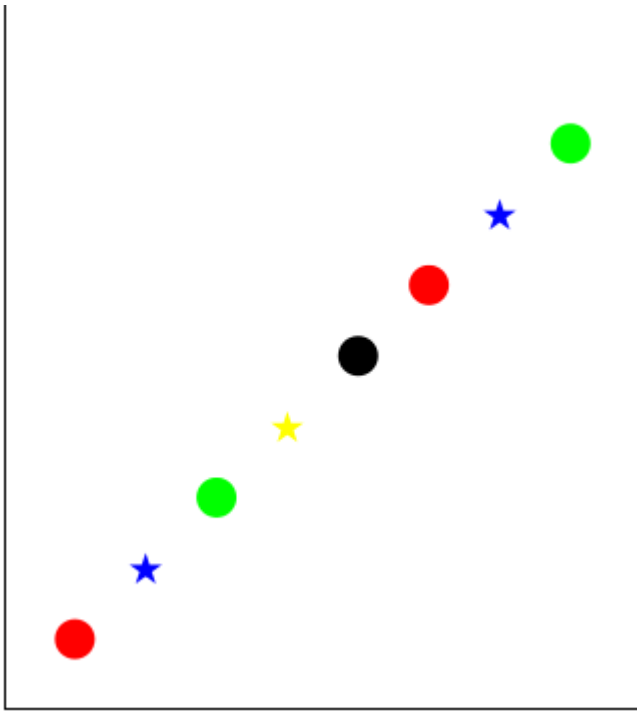
```
(define line-with-grid
  (list (set-bg-grid 20 20 "pink")
        (pen-up)
        (go-to 0 0)
        (stamper-on (circle 5 "solid" "blue"))
        (pen-down)
        (go-to 40 40)
        (go-to 80 80)
        (go-to 120 120)
        (go-to 160 160)
        (stamper-off)
        (go-to 500 500)))

(draw line-with-grid)
```



4.10 Нанесення лінії з декількома штампами (марками)

Ми рухаємося до місця походження за допомогою команди **go-to-origin** і малюємо лінію, використовуючи список марок.

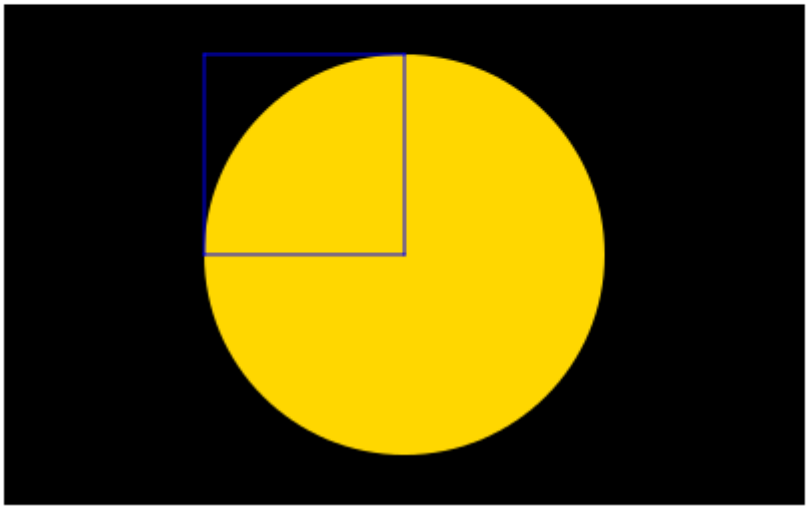
```
(define STAMPS
  (list (circle 10 "solid" "red")
        (star 10 "solid" "blue")
        (circle 10 "solid" "green")
        (star 10 "solid" "yellow")
        (circle 10 "solid" "black")))

(define line-with-stamps
  (list (pen-up)
        (go-to-origin)
        (turn-right 45)
        (stamper-on STAMPS)
        (repeat 8 (forward 50))))

(draw line-with-stamps)
```



4.11 Зміна кольору фону, фонового зображення та розміру анімації

Ви можете змінити колір фону за допомогою **change-bg-color**, а поверх нього можна додати додаткове зображення, використовуючи **set-bg-image**. У прикладі ми хочемо намалювати зображення з різними розмірами, тому використовується малюнок на замовлення (встановлення швидкості малювання до нуля не вплине на швидкість малювання).

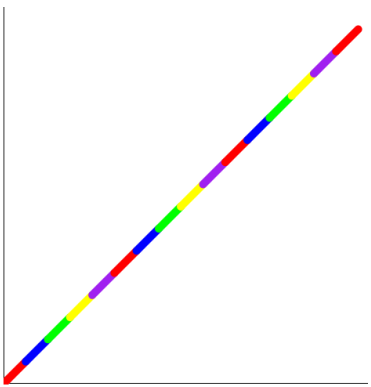
```
(define square-over-bg
  (list (change-bg-color "black")
        (set-bg-image (circle 100 "solid" "gold"))
        square1))

(draw square-over-bg 400 250 0)
```



4.12 Лінія зі змінними кольорами

У цьому прикладі ми встановлюємо колір ручки як список кольорів. Черепаха буде використовувати кожен колір один за одним. Швидкість анімації встановлена повільніше (останній аргумент **draw-custom**). Черепаха також прихована під час першої частини анімації.

```
(define COLORS
  (list "red" "blue" "green" "yellow" "purple"))

(define color-line
  (list (pen-up)
        (go-to 0 0)
        (turn-right 45)
        (pen-down)
        (change-color COLORS)
        (change-pen-size 10)
        (hide-turtle)
        (repeat 8 (forward 40))
        (show-turtle)
        (repeat 8 (forward 40))))

(draw-custom color-line 500 500 0.5)
```



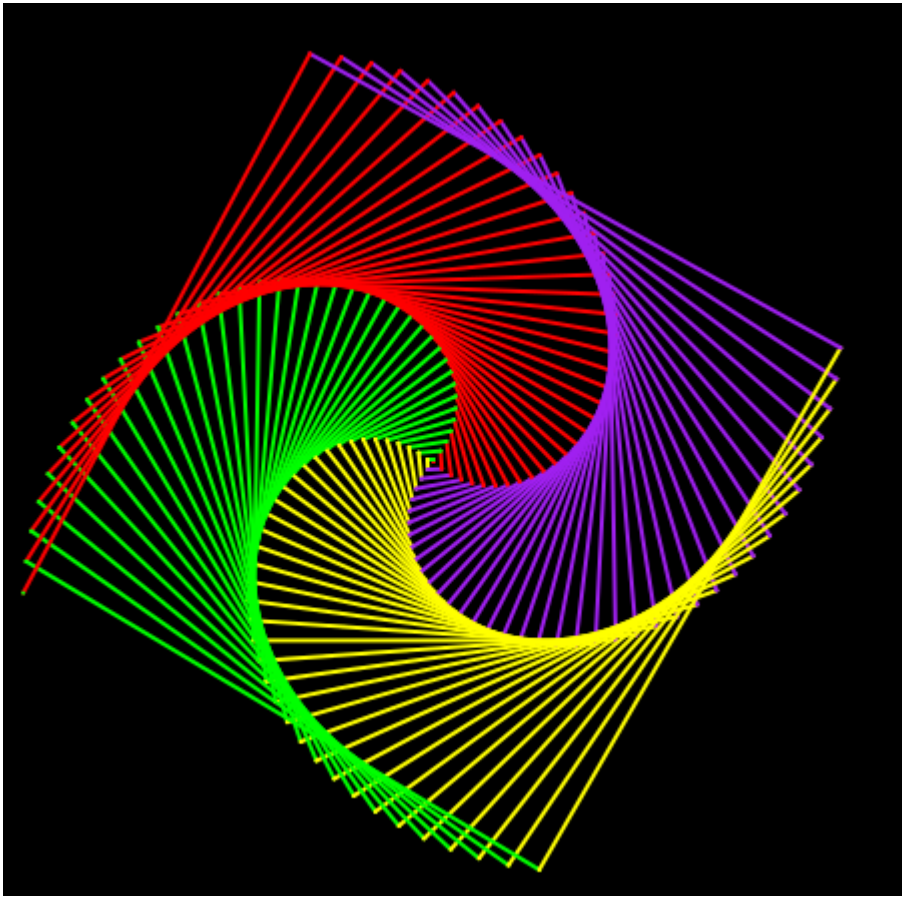
5. Приклади рекурсивної графіки

5.1. Спіраль із зміною кольору ручки

Спіраль створена рекурсивним збільшенням відстані, яку рухається черепаха. Кольори ручки змінюються відповідно до списку кольорів.

```
(define COLORS1 (list "red" "green" "yellow" "purple"))

(define (spiral a x times)
  (if (< times 0)
      empty
      (append (list (forward x)(turn-left a))
              (spiral a (+ x 2)(sub1 times)))))

(define spiral-image
  (list (change-pen-size 2)
        (change-bg-color "black")
        (change-color COLORS1)
        (spiral 91 1 152)))
```

```
(draw spiral-image)
```

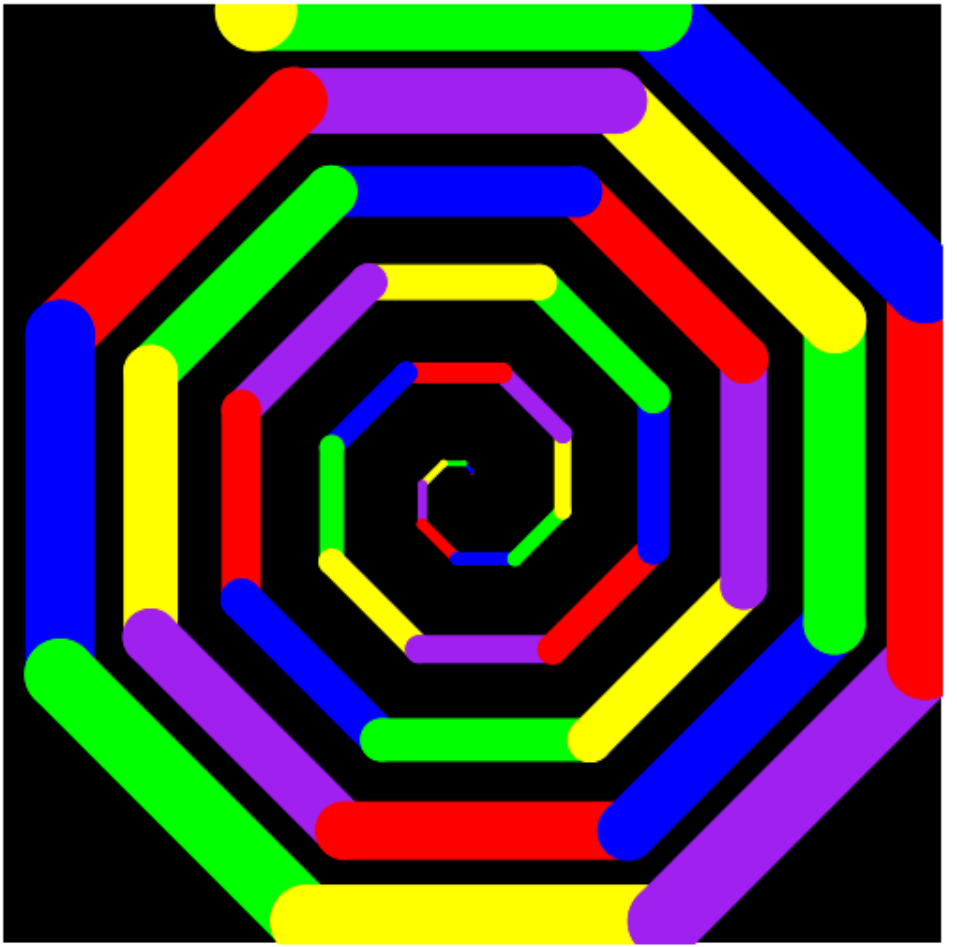


5.2 Спіраль із зміною кольору та розміру ручки

Спіраль створена рекурсивним збільшенням відстані, яку рухається черепаха. Кольори ручки змінюються відповідно до списку кольорів, а розмір ручки щоразу збільшується.

```
(define COLORS2 (list "red" "blue" "green" "yellow" "purple"))

(define (side x w a)
  (list (change-pen-size w)(forward x)(turn-left a)))

(define (spiral2 x w a times)
  (if (<= times 0)
      empty
      (cons (side x w a)
            (spiral2 (+ x 5) (+ w 1) a (sub1 times)))))

(define spiral-image2 (list (change-bg-color "black")
                            (change-color COLORS2)
                            (spiral2 1 1 45 45)))
```

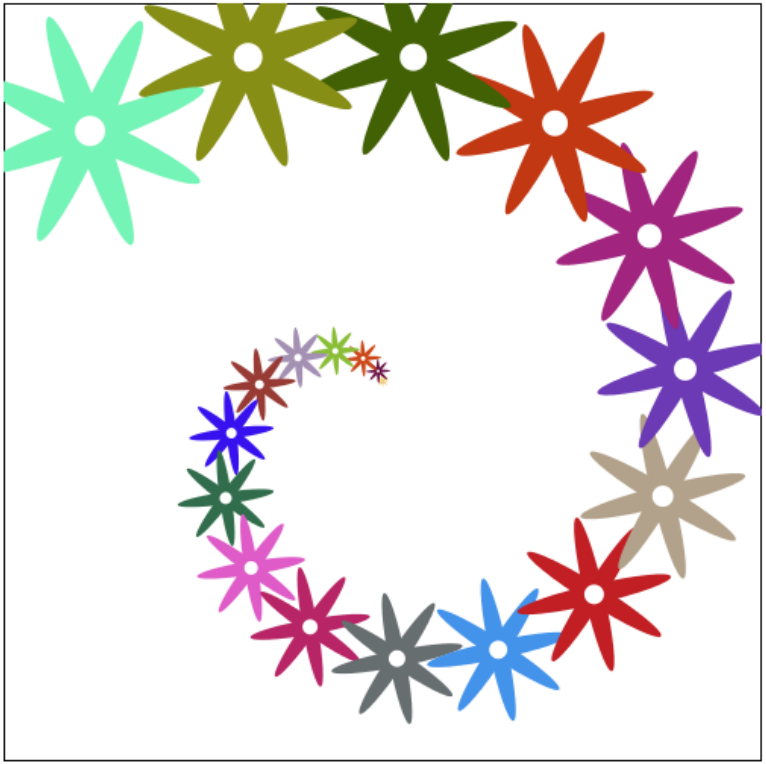
```
(draw spiral-image2)
```



5.3 Квіткова спіраль із зміною розмірів та кольорів

Квіткова спіраль запрограмована за допомогою функції штампа. Кольори квітів змінюються випадковим чином.

```
(define (leaf size color)
  (ellipse (* 8 size) size "solid" color))

(define (leafs size color)
  (overlay (leaf size color)
           (rotate 90 (leaf size color))))

(define (draw-flower size color)
  (overlay (circle (/ size 2) "solid" "white")
           (leafs size color)
           (rotate 45 (leafs size color))))

(define (flower-spiral a x times)
  (if (< times 0)
      empty
      (append (list (forward x)(turn-left a))
              (flower-spiral a (+ x 6)(sub1 times)))))
```



5.4 Зоряна спіраль із зміною розмірів та кольорів

```
(define (draw-stars n size)
(if (<= n 0)
    empty
    (cons (rotate size (star size "solid" (make-color (random 255)
                                                      (random 255)
                                                      (random 255))))
(draw-stars (sub1 n)(add1 size)))))

(define spiral-image4 (list (stamper-on (draw-stars 100 1))
                            (pen-up)
                            (spiral 91 1 100)))

(draw spiral-image4)
```