DES ALLIGATORS AUX FONDEMENTS DE L'INFORMATIQUE

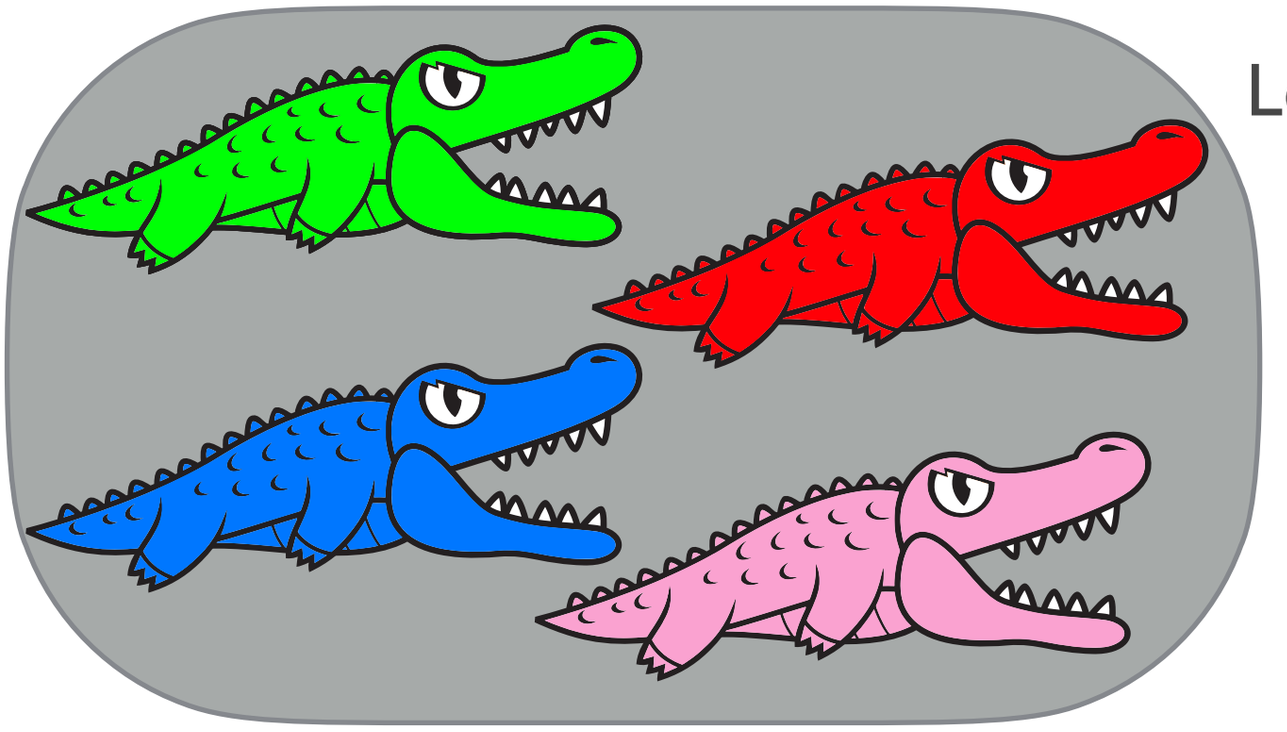
Benjamin Monmege

Journée pour l'enseignement de l'informatique

Marseille - 17 mai 2023

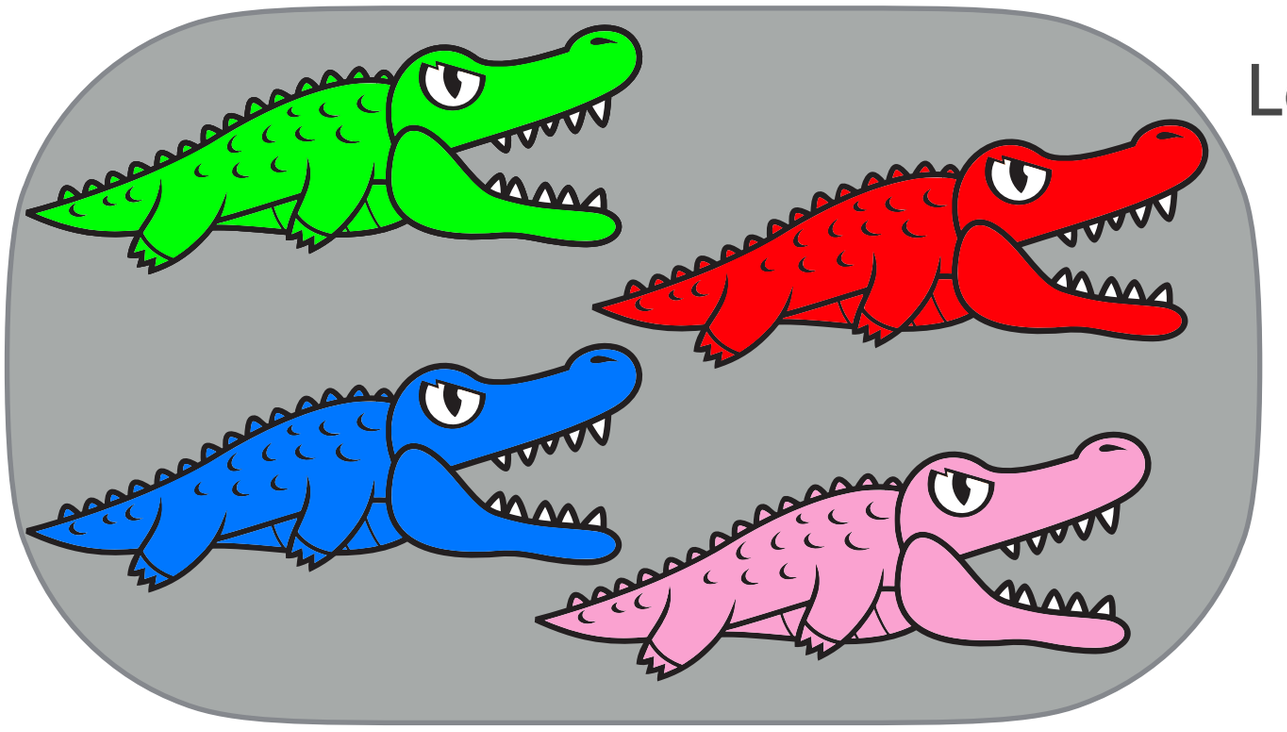
Sur une idée originale de Bret Victor : http://worrydream.com/AlligatorEggs/

La dure vie des alligators

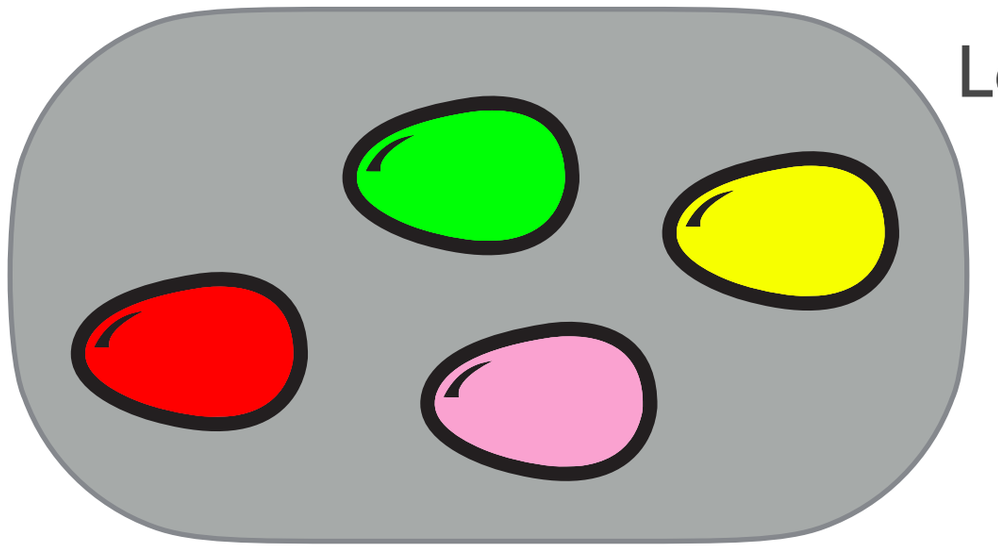


Les alligators

La dure vie des alligators

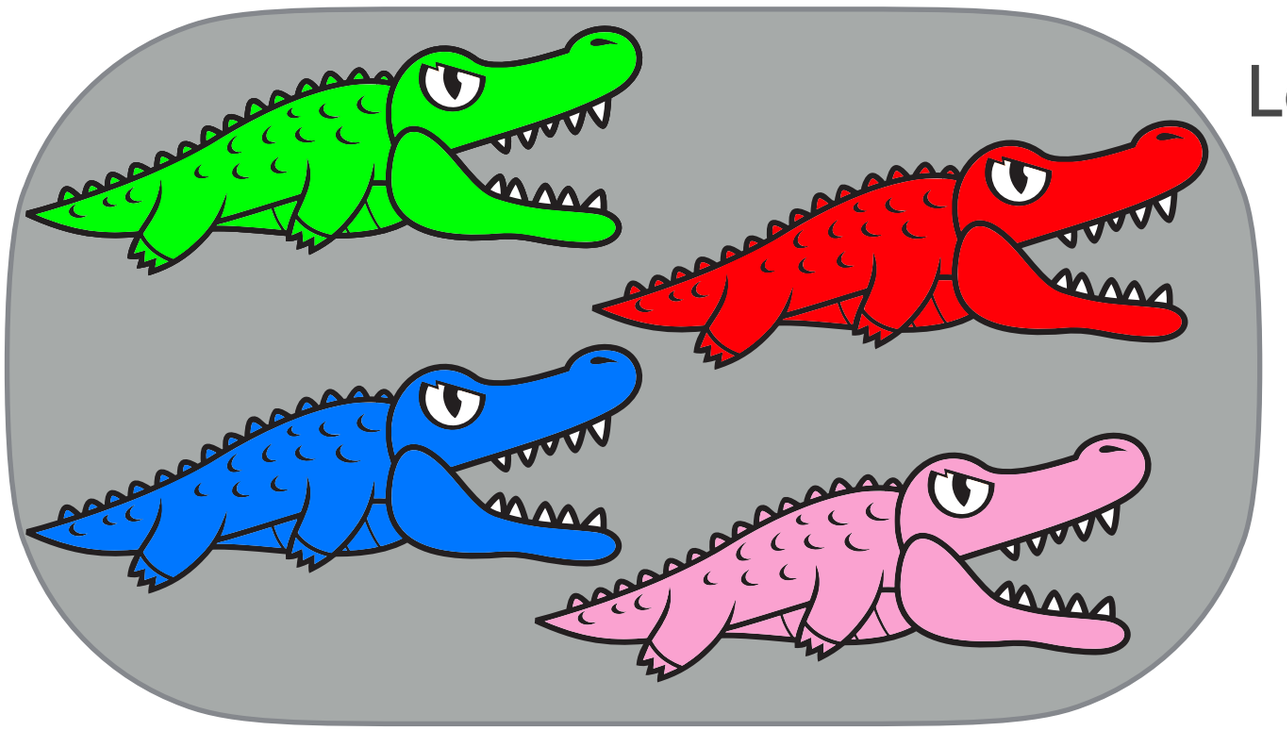


Les alligators

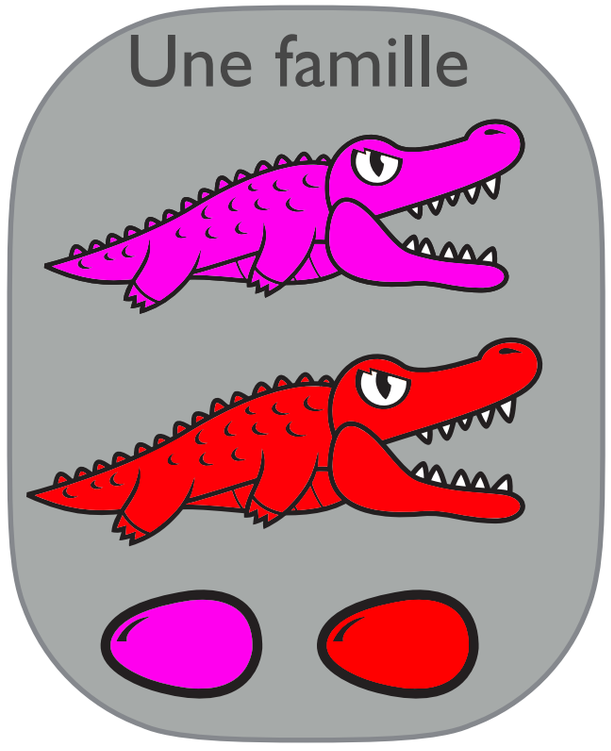


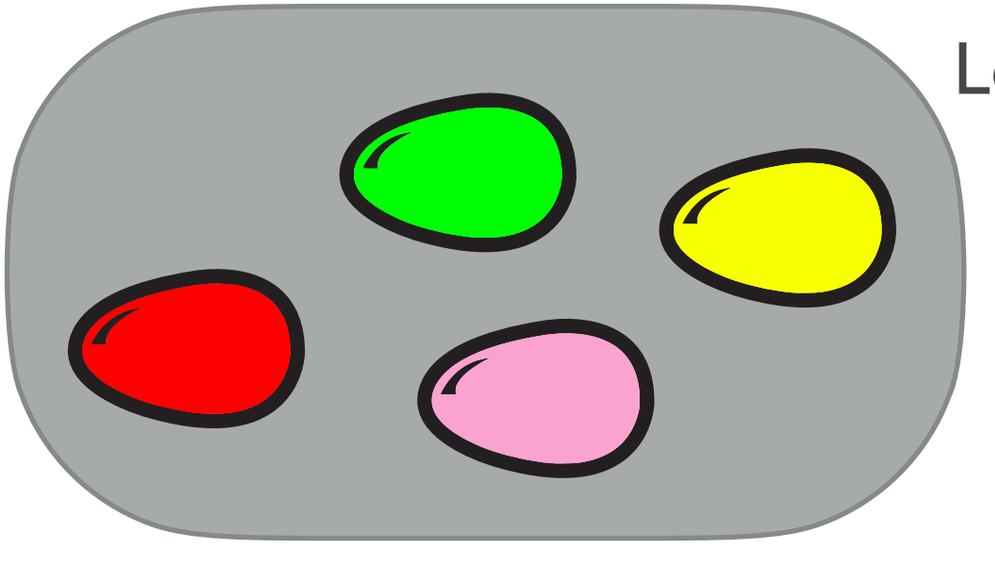
Les œufs

La dure vie des alligators



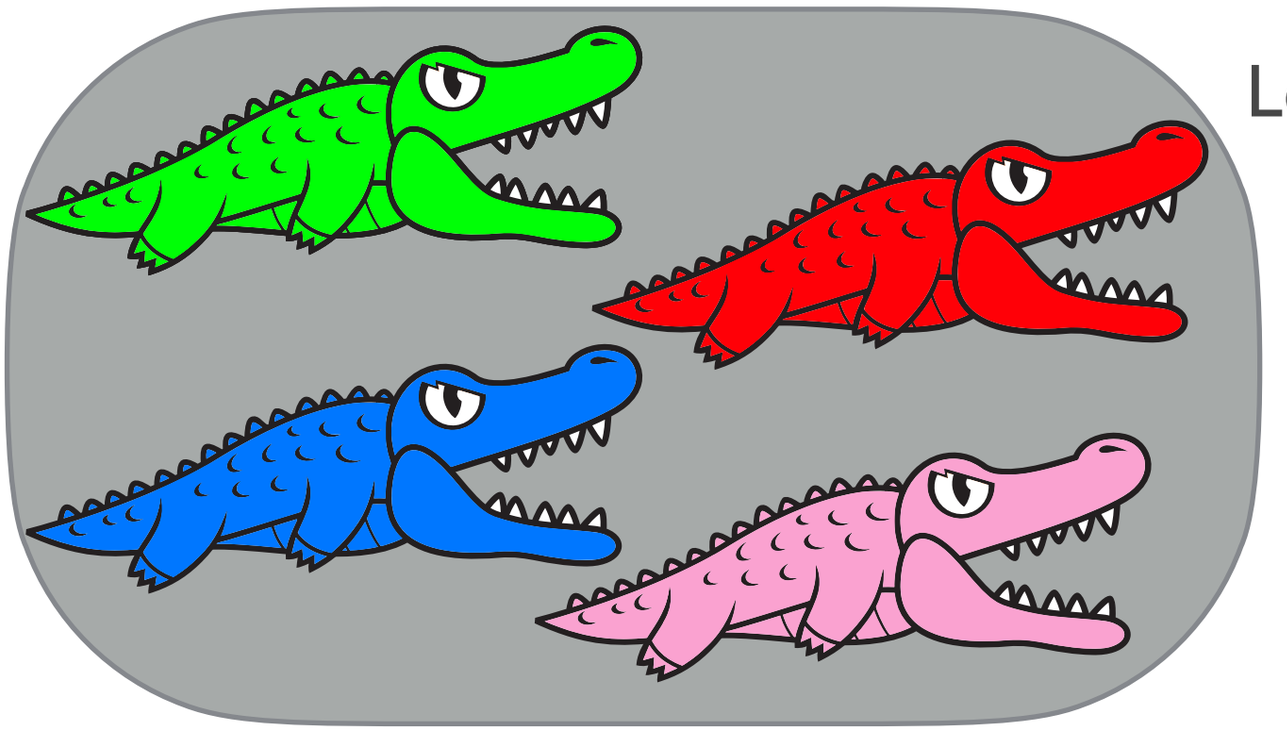
Les alligators



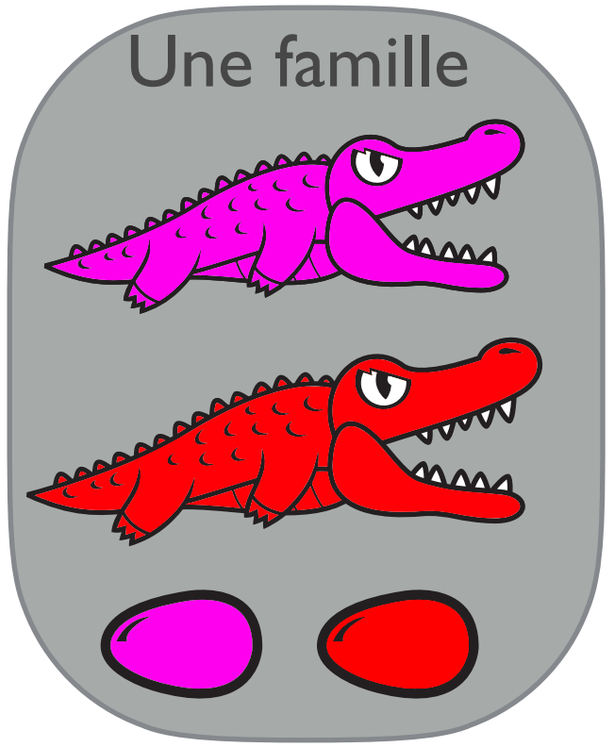


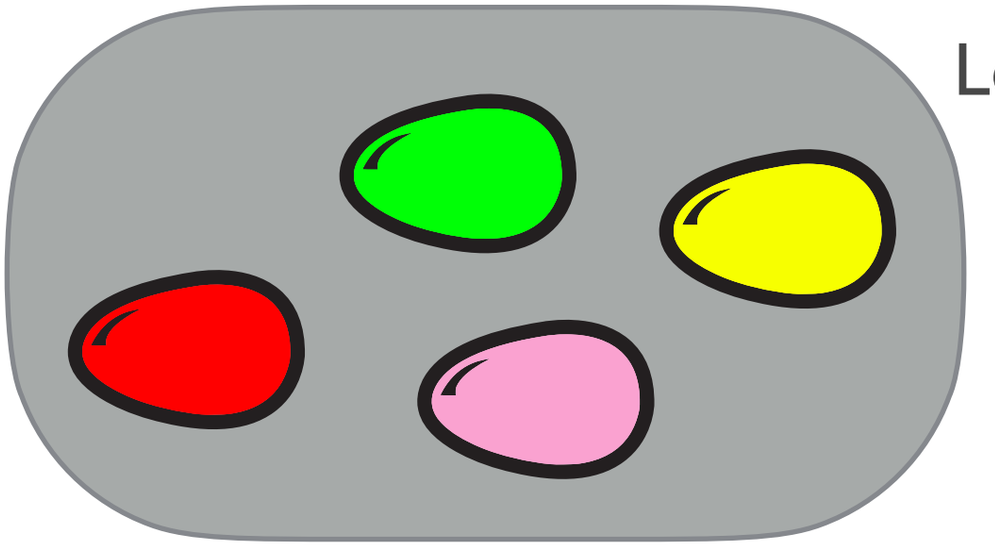
Les œufs

La dure vie des alligators

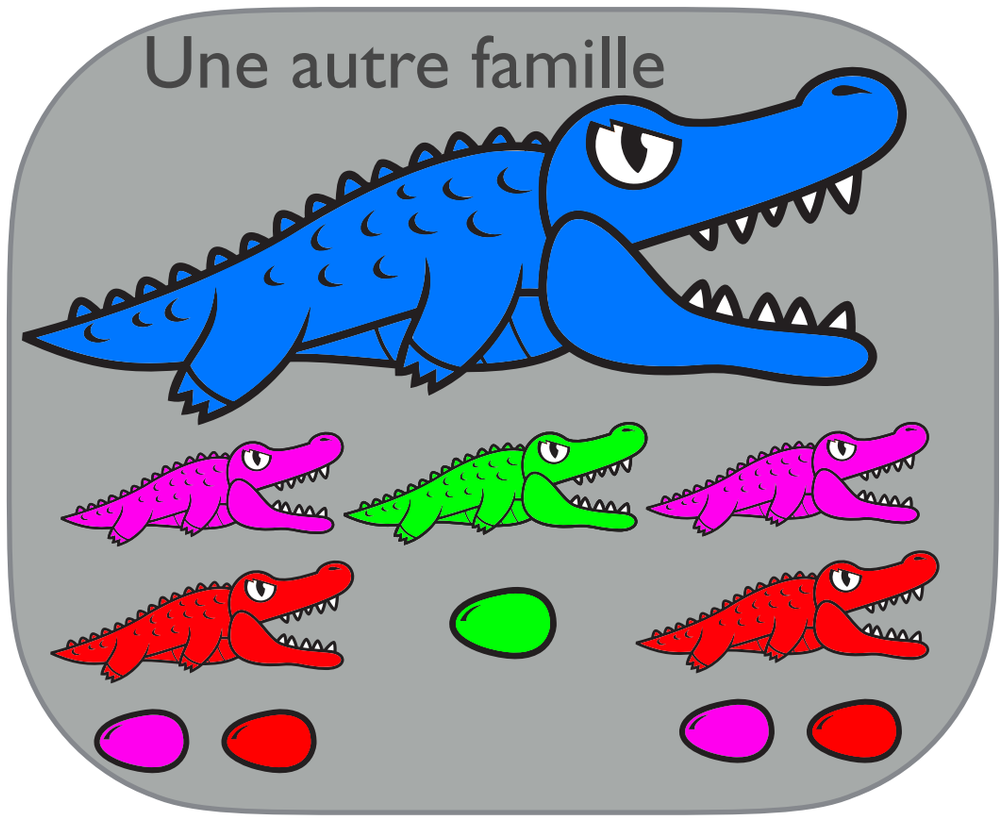


Les alligators

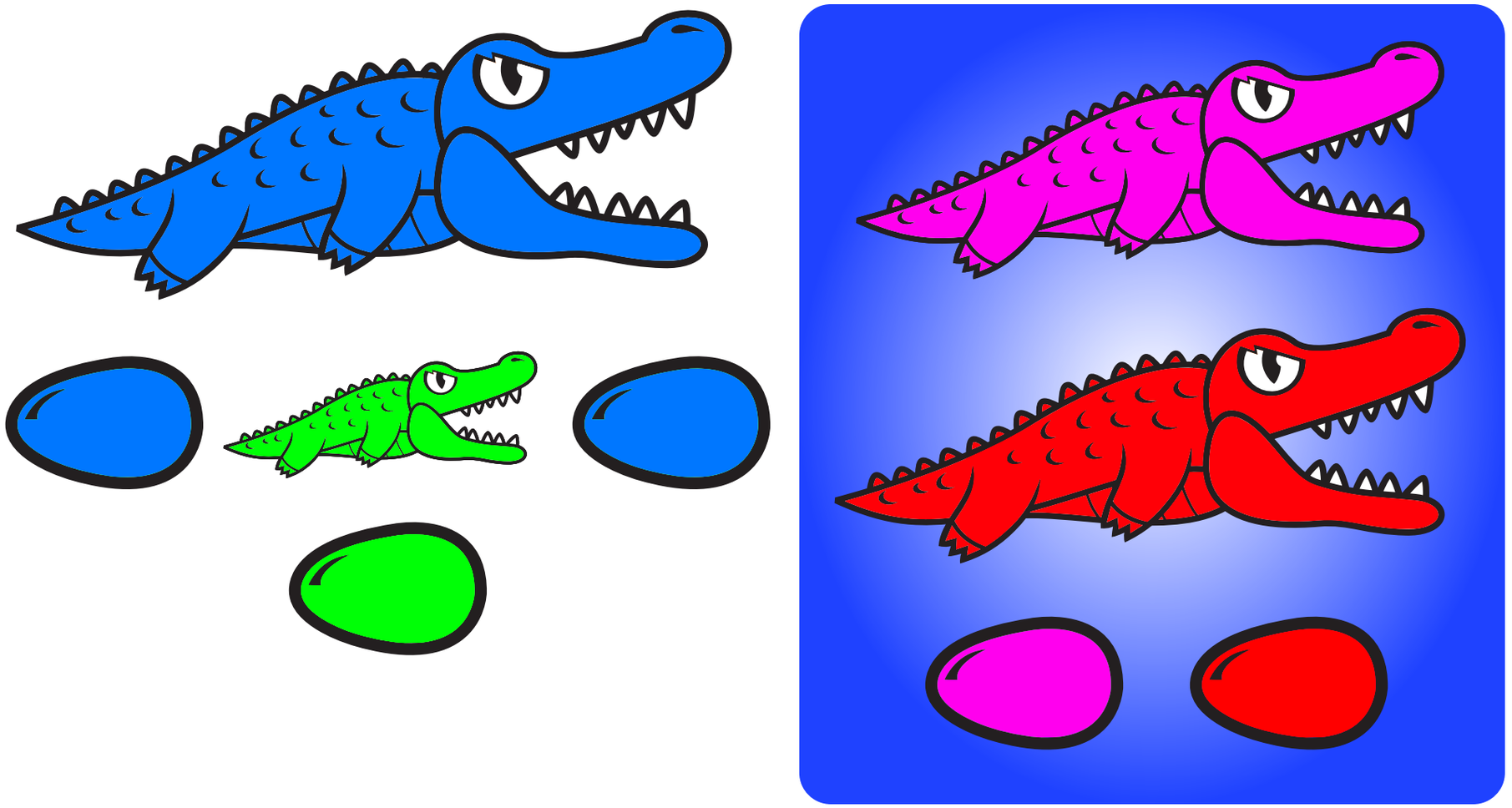




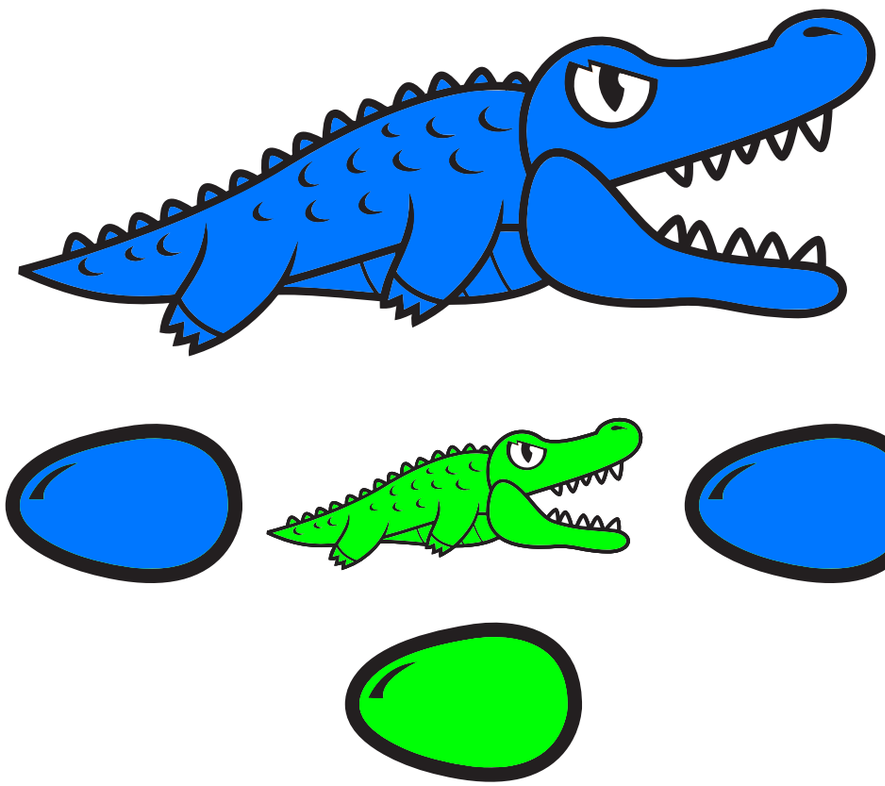
Les œufs



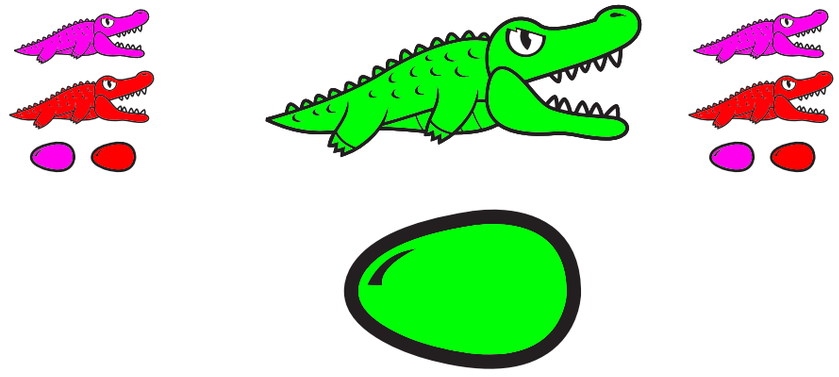
Règle 1 : le repas

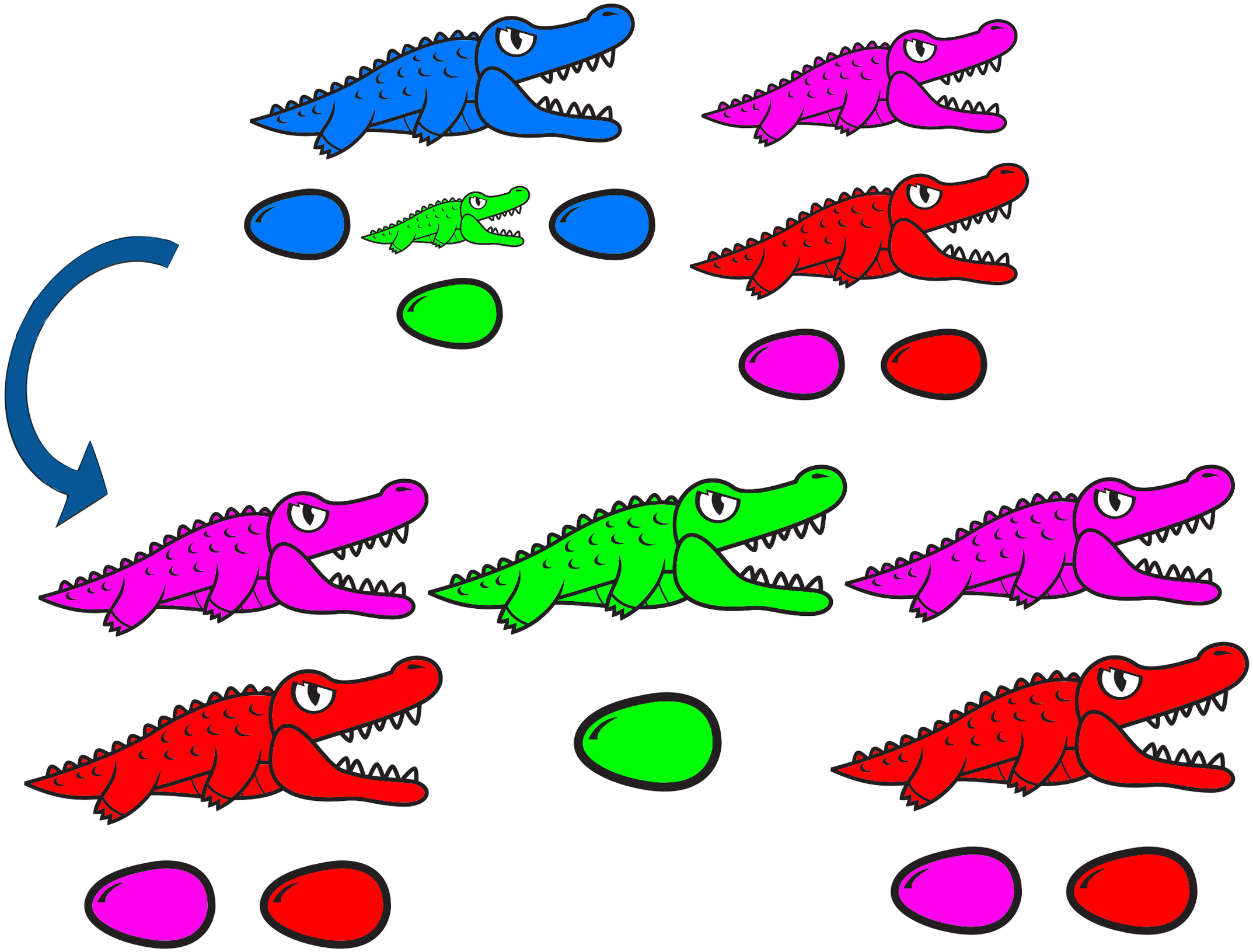


Règle 1 : le repas

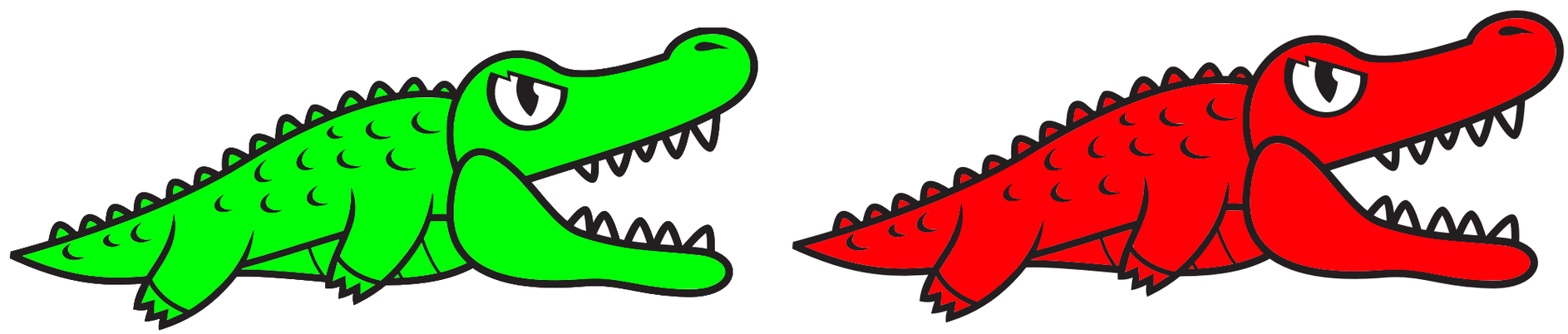


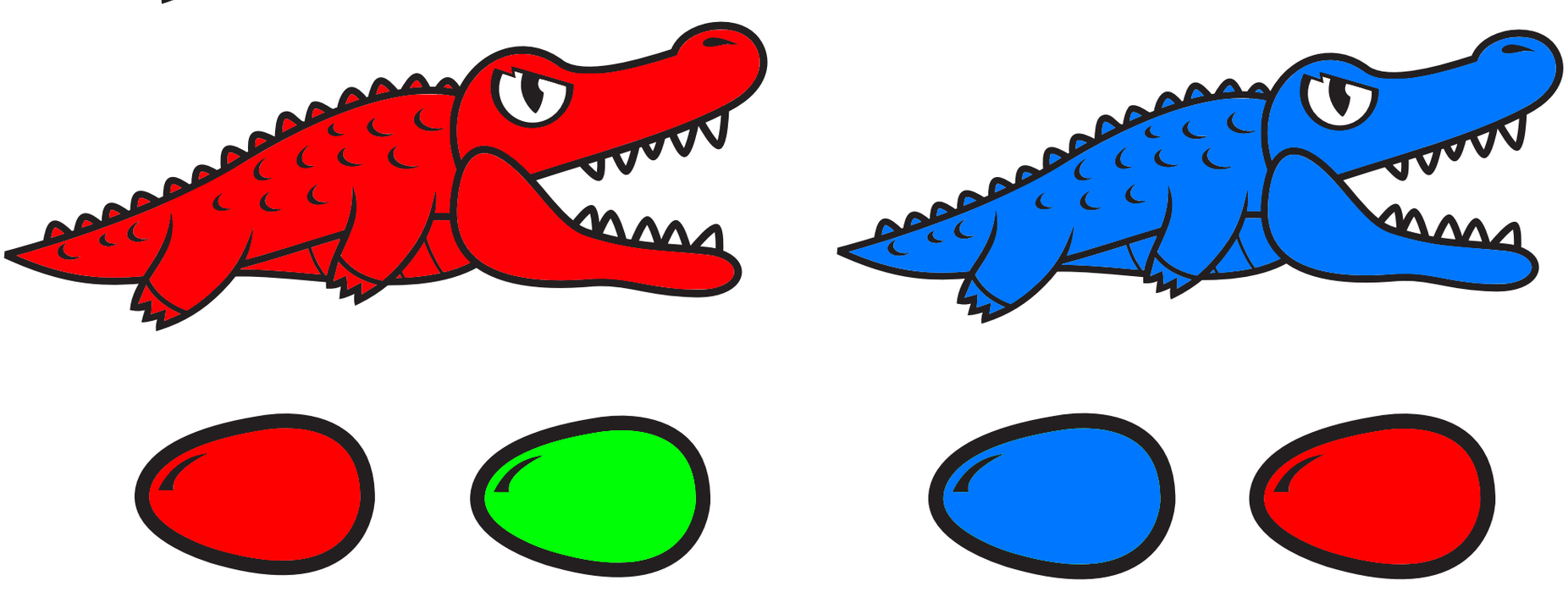
Règle 1 : le repas



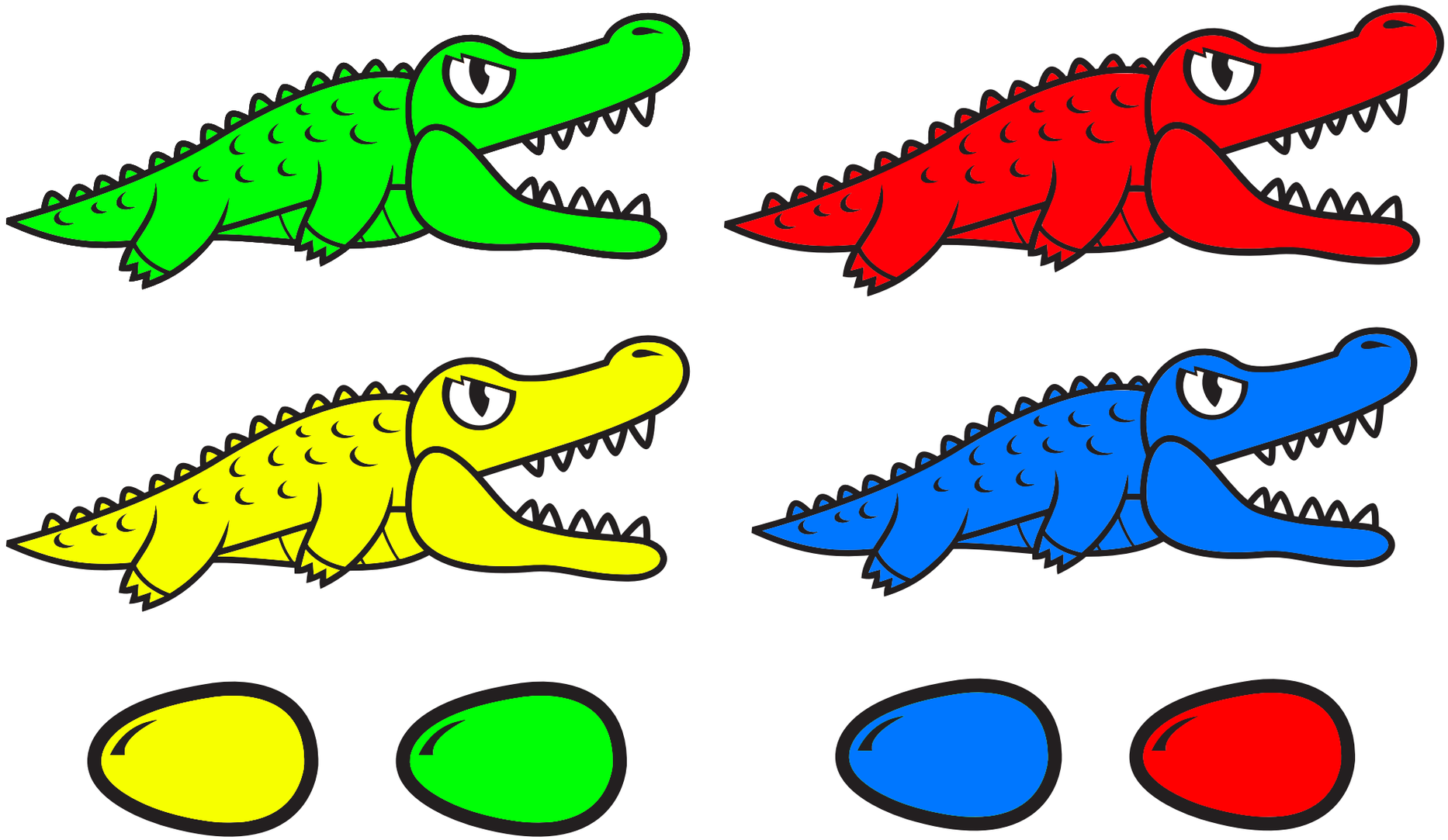


Règle 2 : la couleur

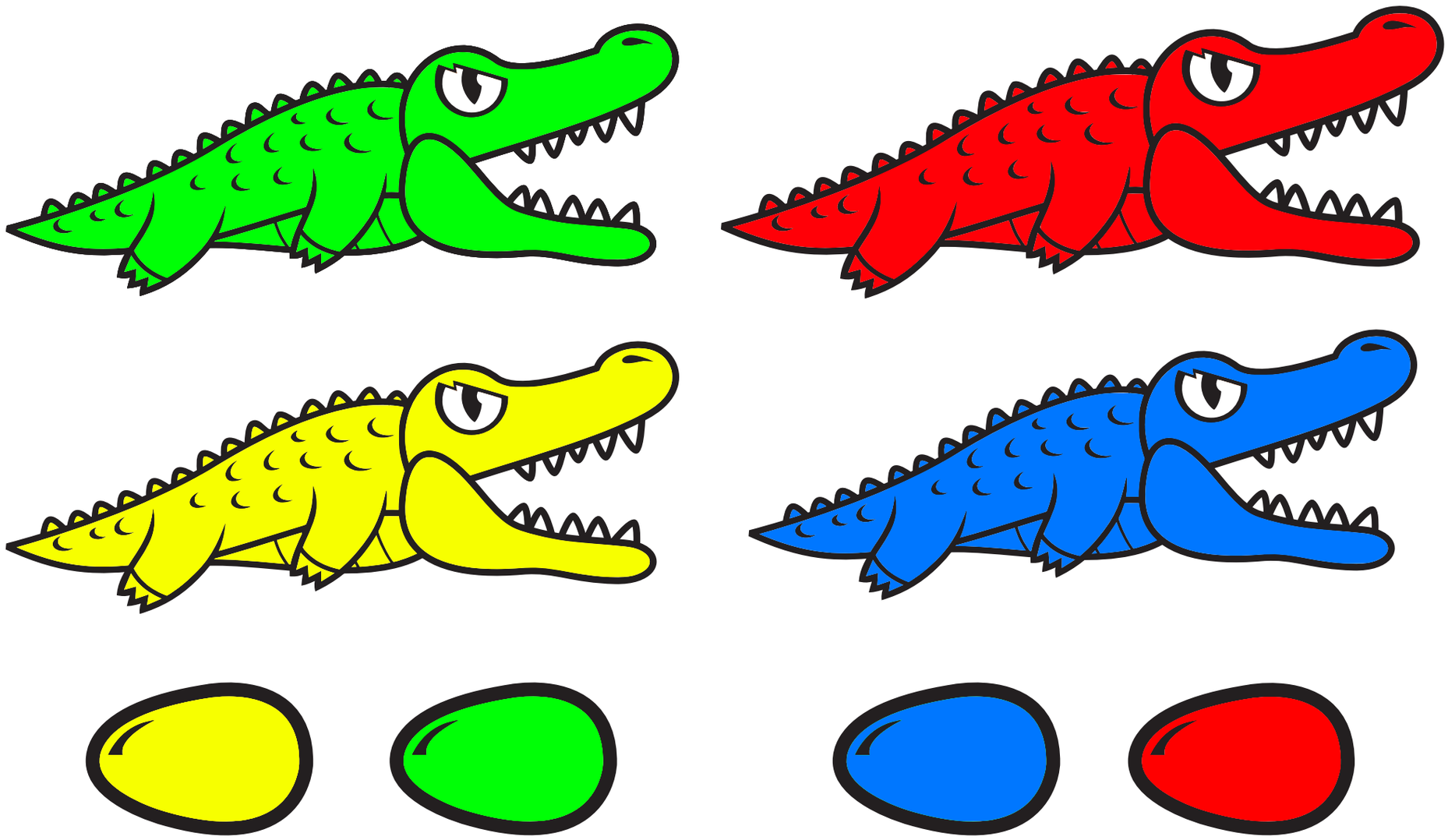




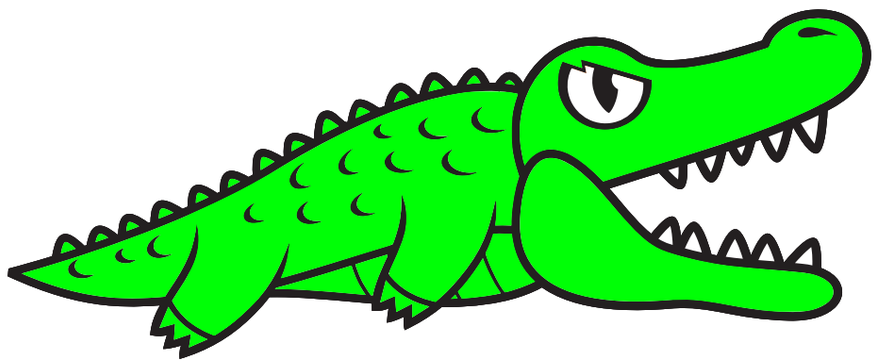
Règle 2 : la couleur

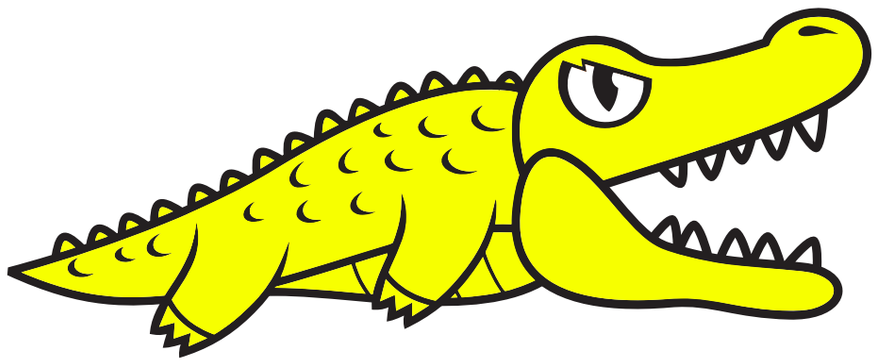


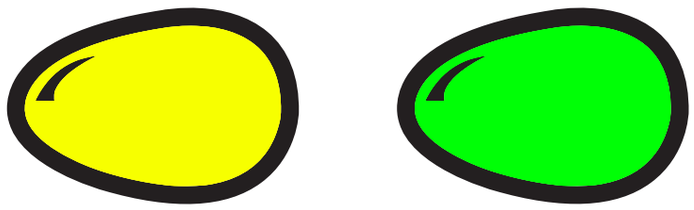
Règle 2 : la couleur



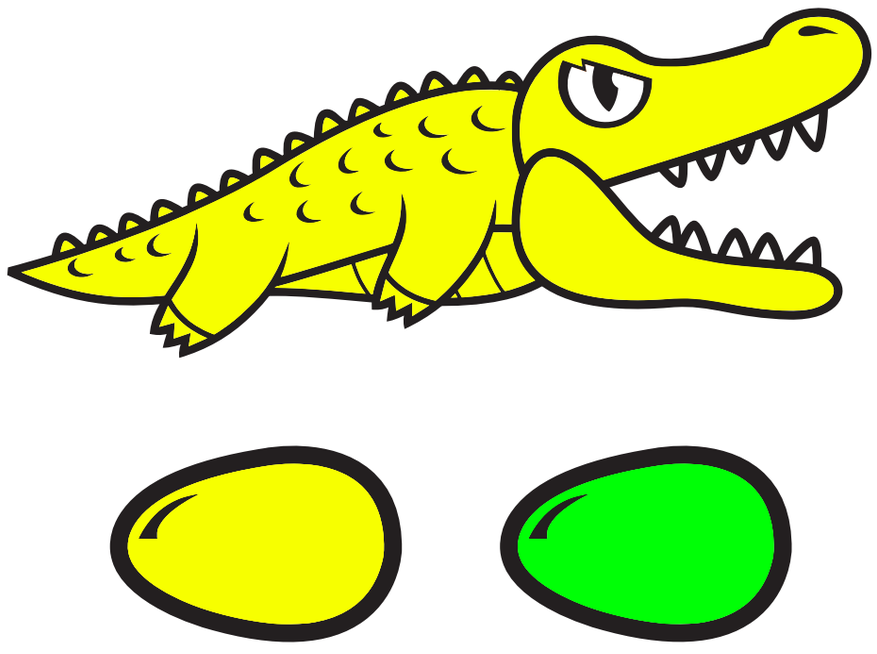
Règle 2 : la couleur



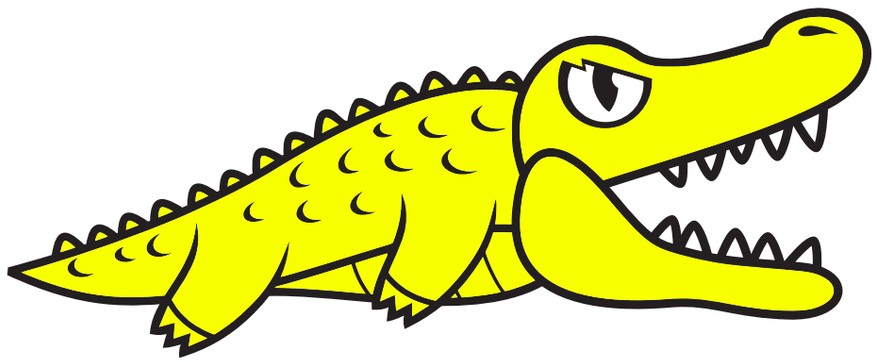


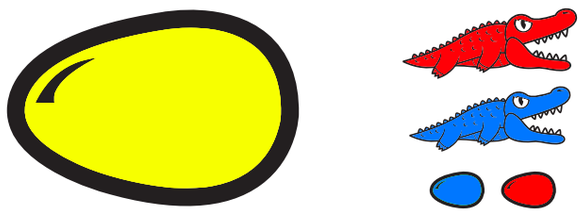


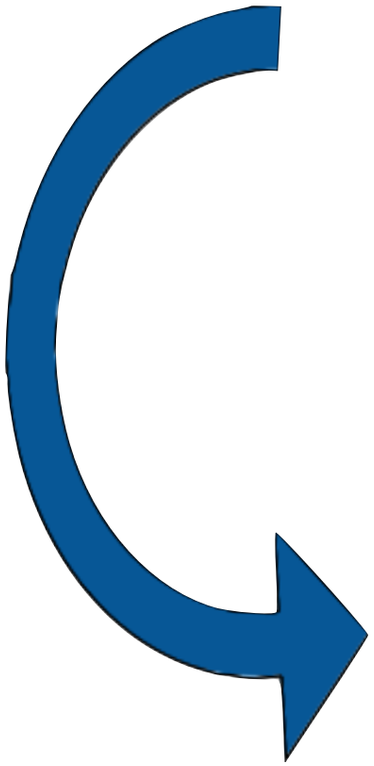
Règle 2 : la couleur

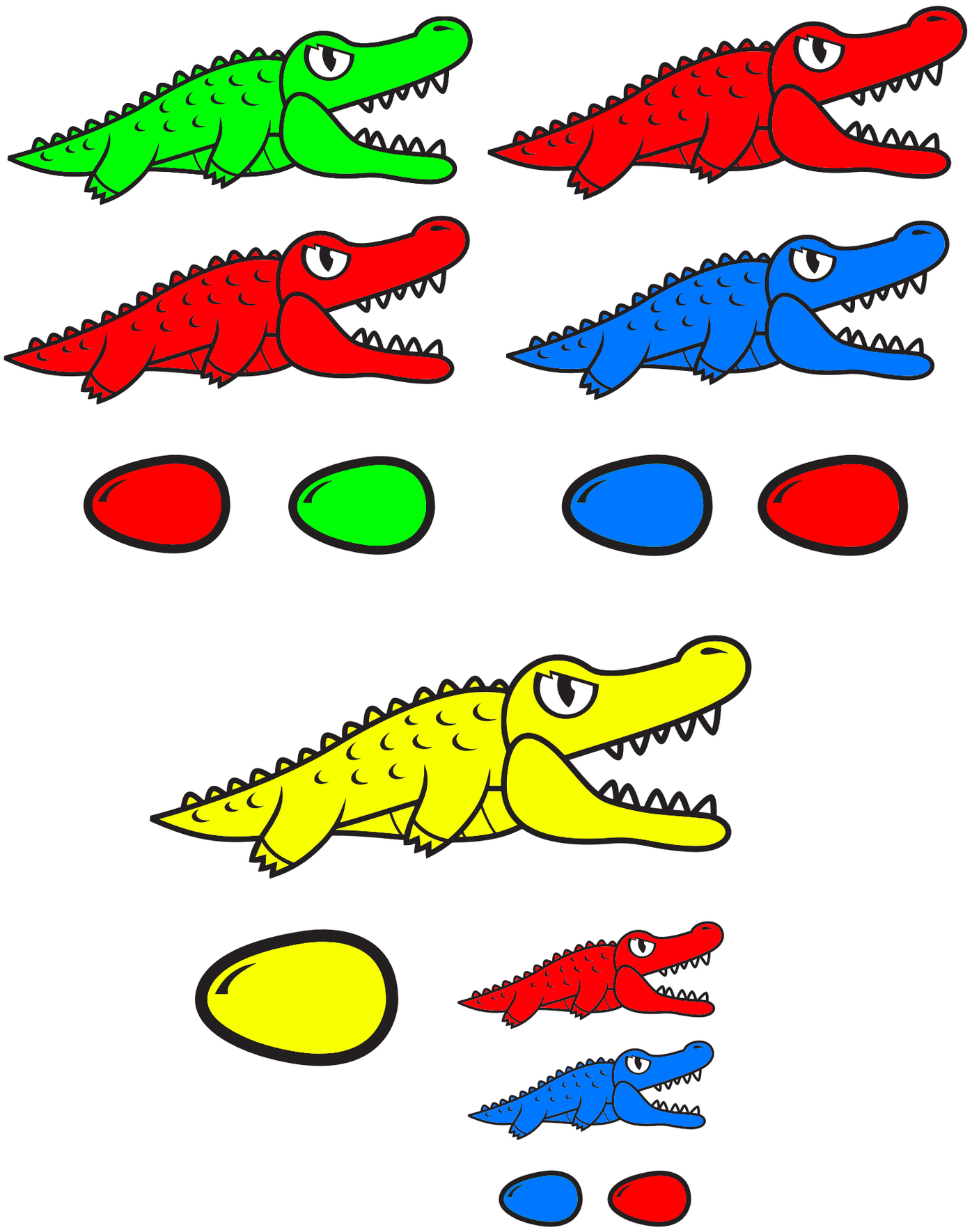


Règle 2 : la couleur



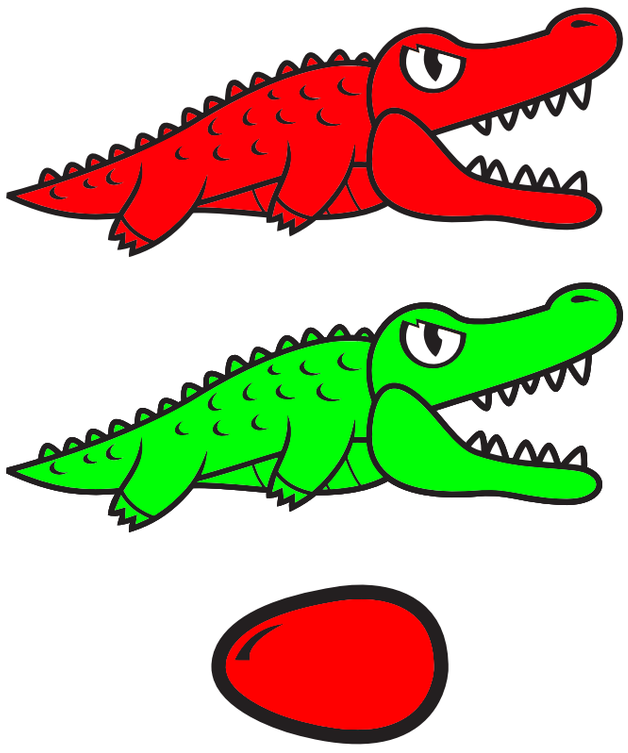




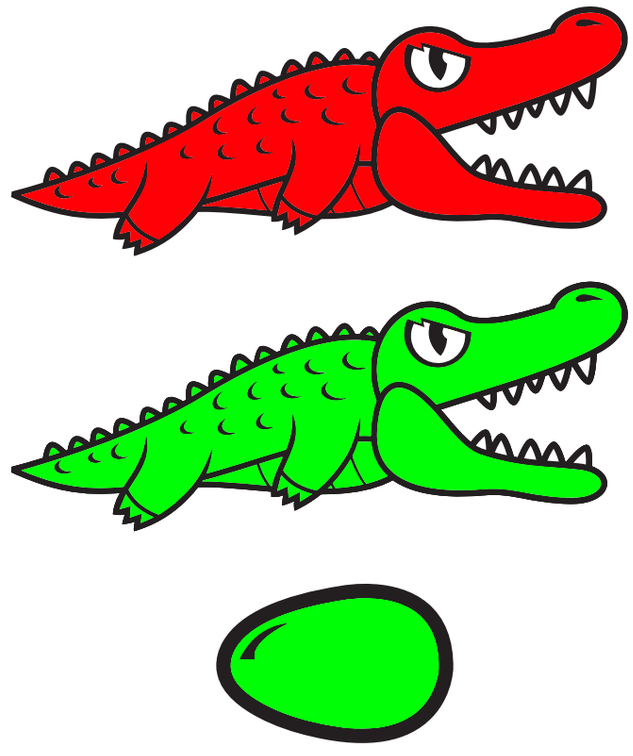


Les familles Thompson et Foster

Thompson

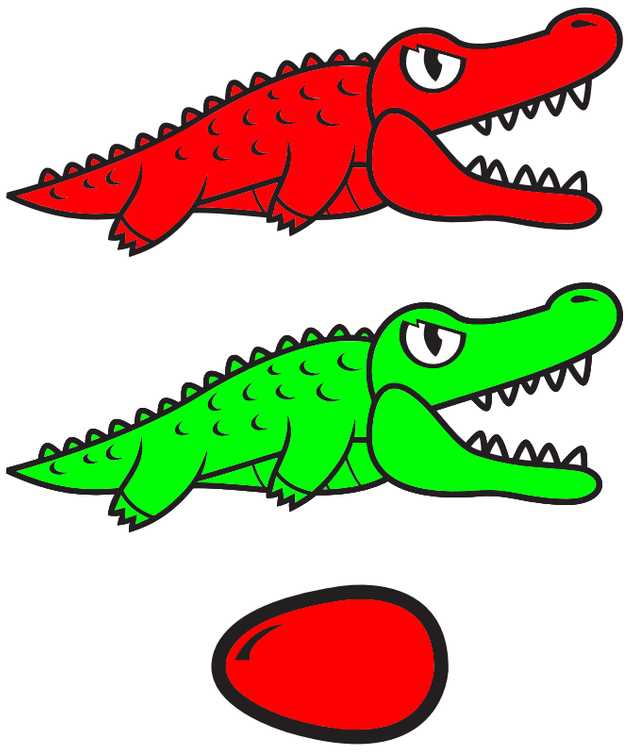


Foster

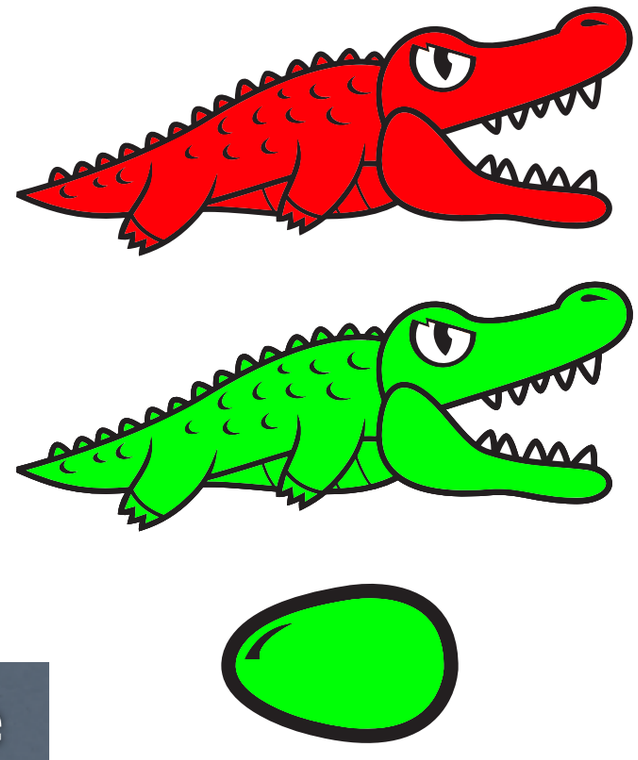


Les familles Thompson et Foster

Thompson



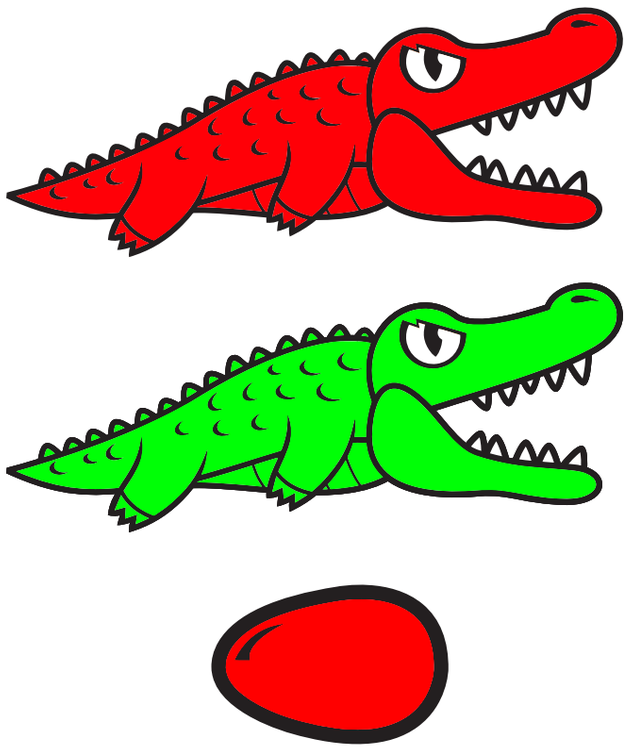
Foster

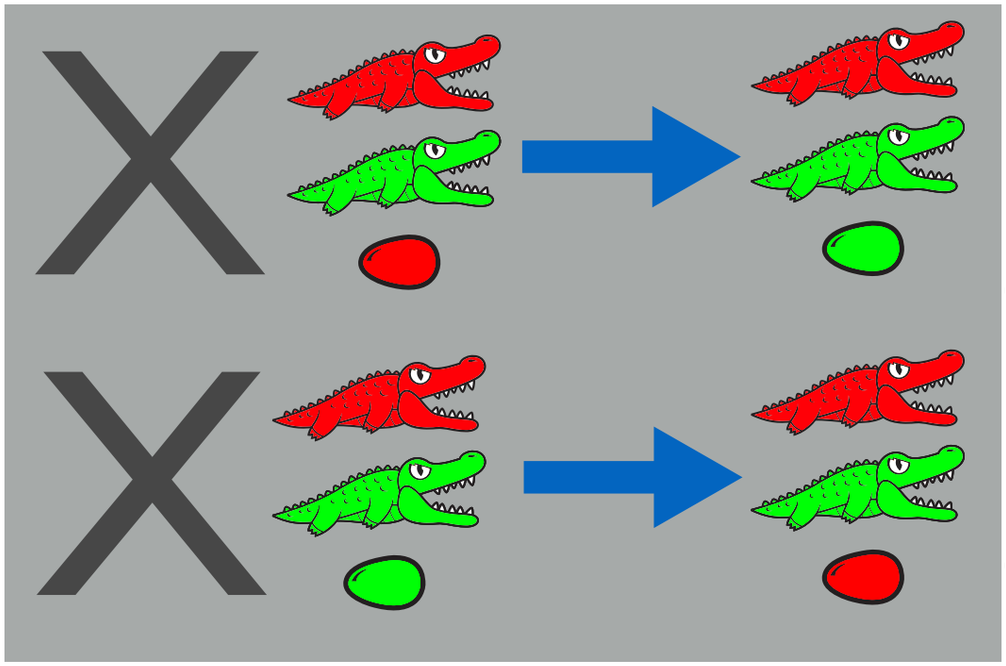


Pouvez-vous trouver une famille qui, après application des règles, résulte en l'inversion des familles Thompson et Foster ?

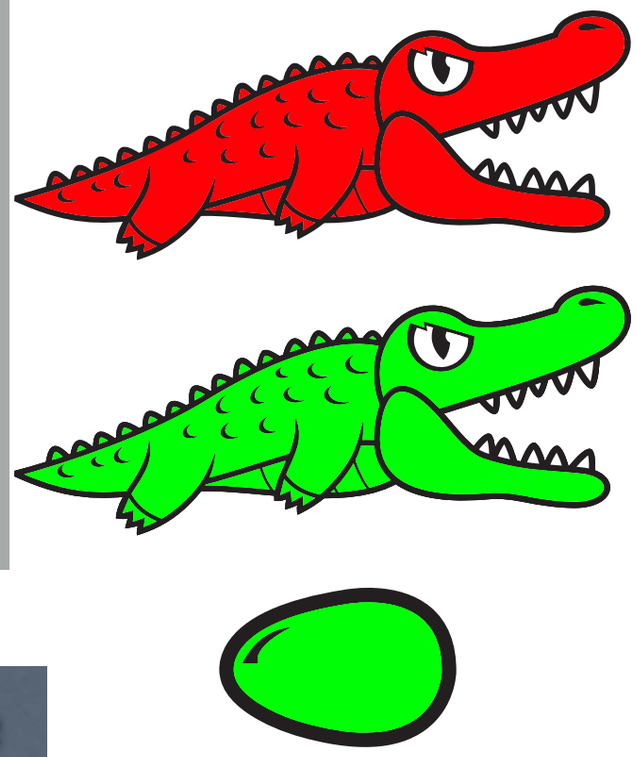
Les familles Thompson et Foster

Thompson





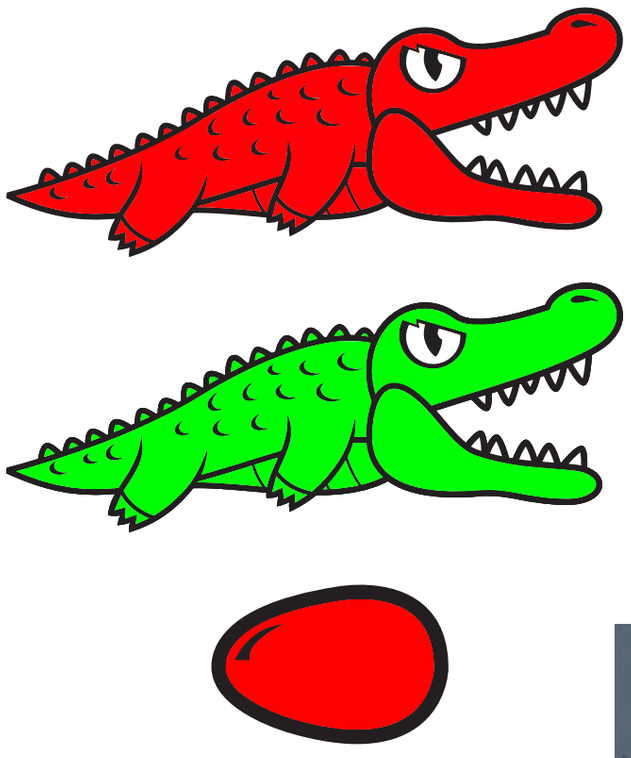
Foster



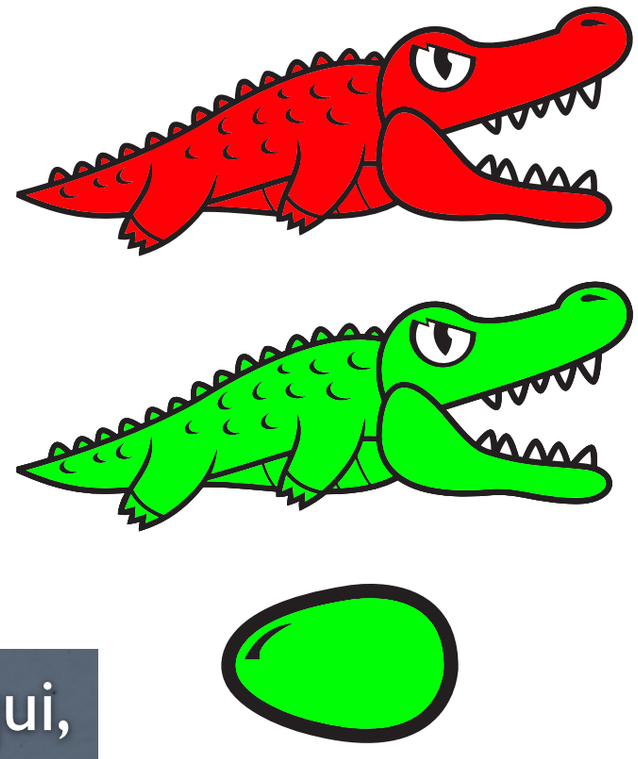
Pouvez-vous trouver une famille qui, après application des règles, résulte en l'inversion des familles Thompson et Foster ?

Les familles Thompson et Foster

Thompson



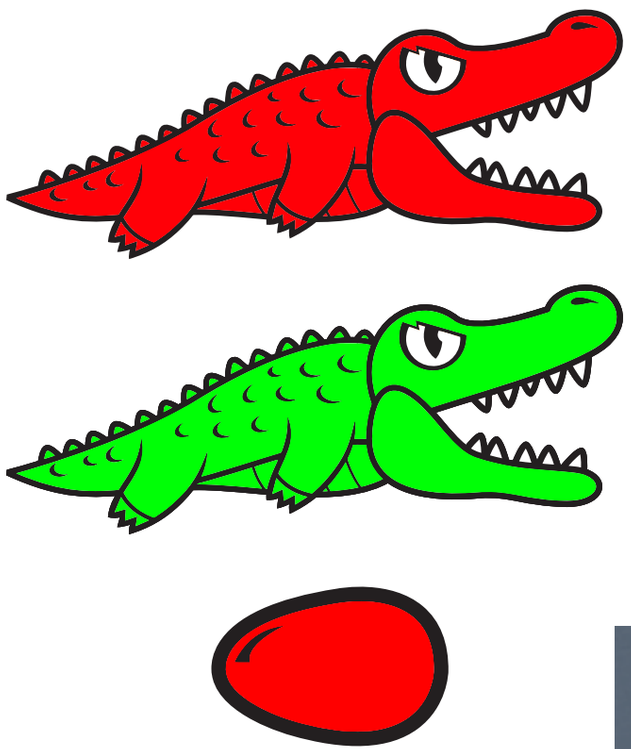
Foster

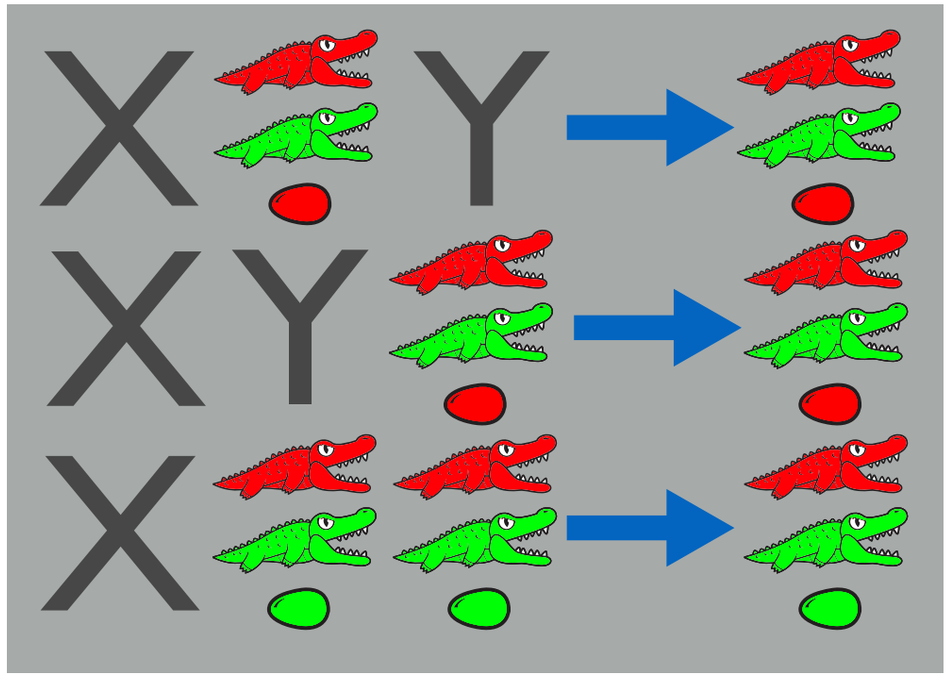


Pouvez-vous trouver une famille qui, après application des règles, détecte la présence de la famille Thompson parmi les deux familles devant, en produisant la famille Thompson, et sinon produit la famille Foster ?

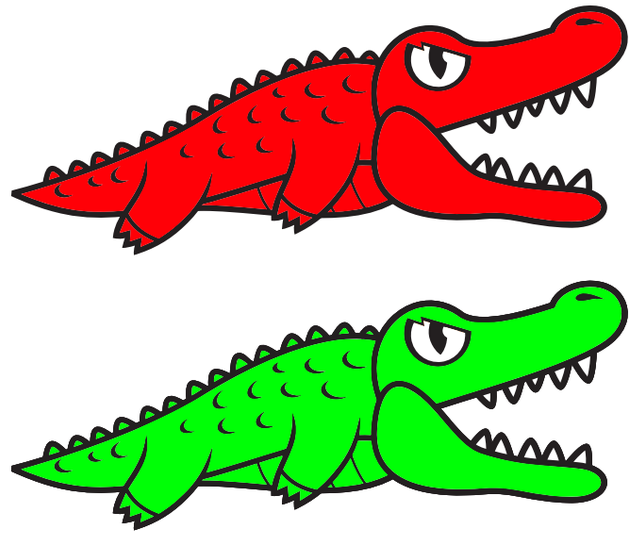
Les familles Thompson et Foster

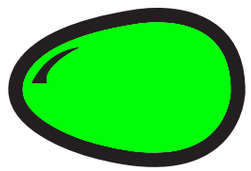
Thompson





Foster

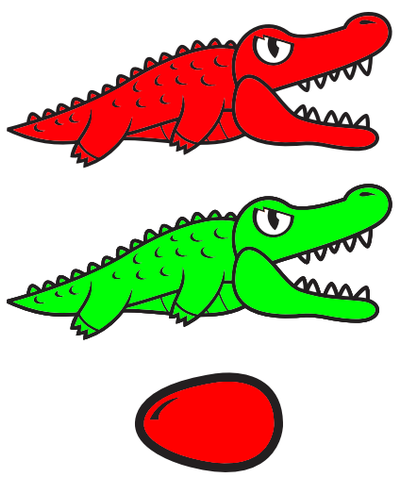




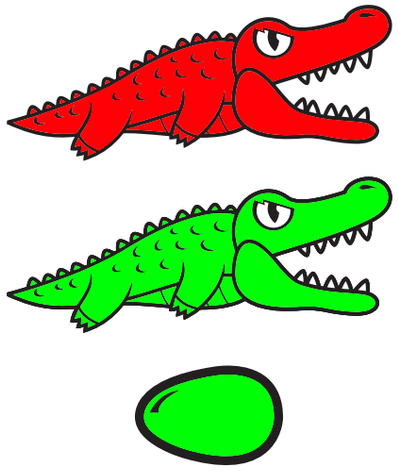
Pouvez-vous trouver une famille qui, après application des règles, détecte la présence de la famille Thompson parmi les deux familles devant, en produisant la famille Thompson, et sinon produit la famille Foster ?

Qu'avons-nous fait ?

Thompson

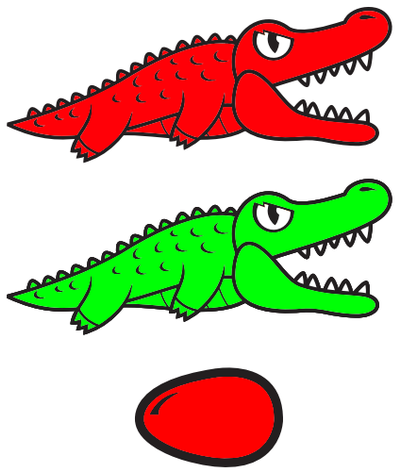


Foster

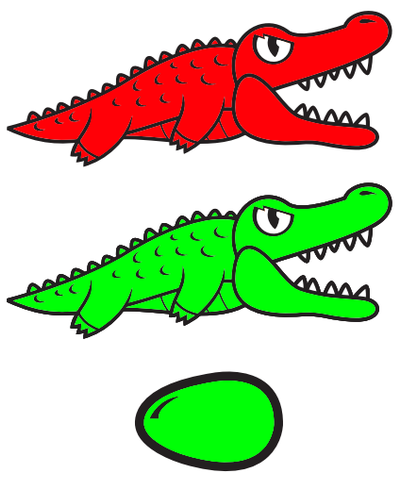


Qu'avons-nous fait ?

Thompson



Foster

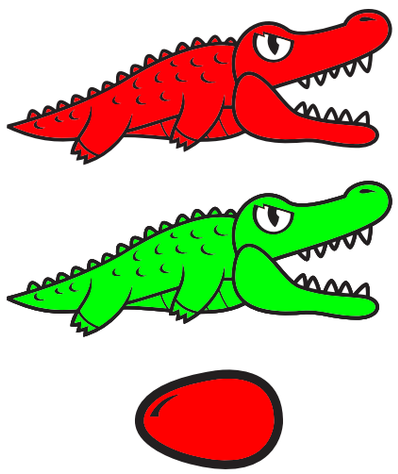


Famille Nelson

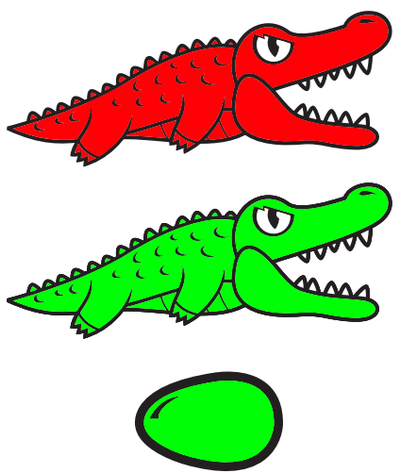
Thompson	Foster
Foster	Thompson

Qu'avons-nous fait ?

Thompson



Foster



Famille Nelson

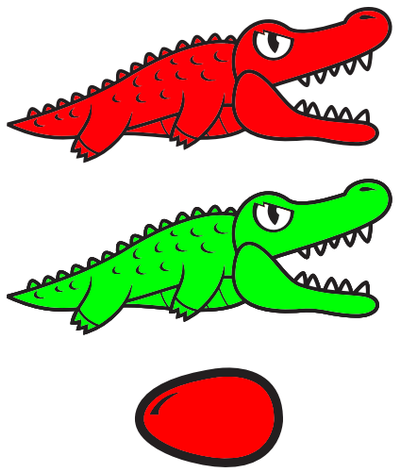
Thompson	Foster
Foster	Thompson

Famille Ortiz

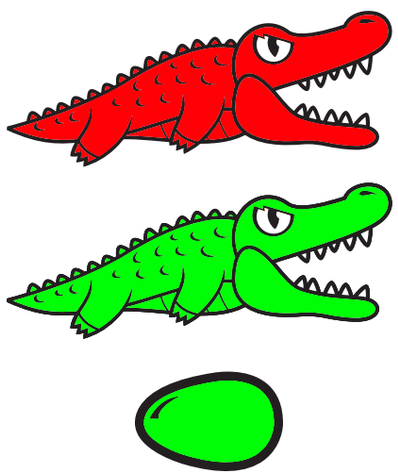
	Thompson	**Foster**
Thompson	Thompson	Thompson
Foster	Thompson	Foster

Qu'avons-nous fait ?

Thompson



Foster



Famille Nelson

Thompson	Foster
Foster	Thompson

Famille Ortiz

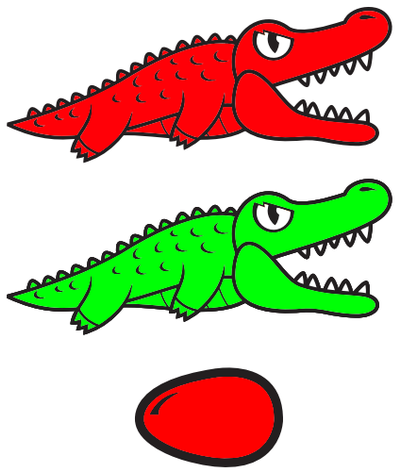
	Thompson	**Foster**
Thompson	Thompson	Thompson
Foster	Thompson	Foster

Famille Anderson

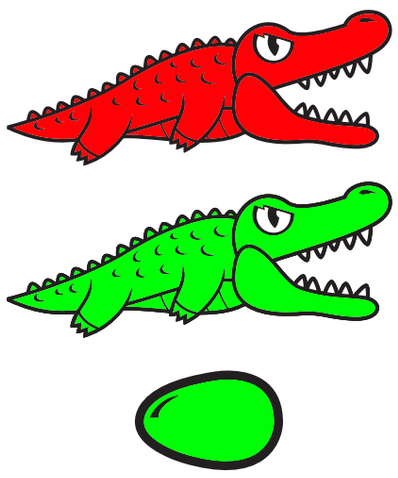
	Thompson	**Foster**
Thompson	Thompson	Foster
Foster	Foster	Foster

Qu'avons-nous fait ?

Thompson



Foster



Famille Nelson

T	F
F	T

Famille Ortiz

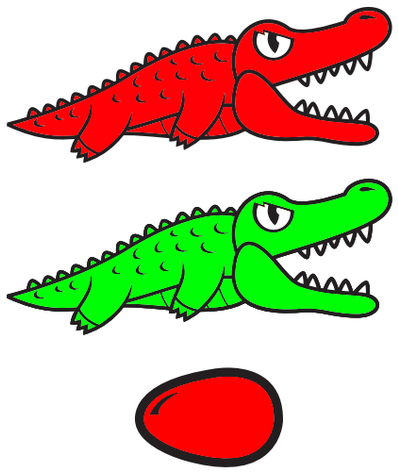
	T	**F**
T	T	T
F	T	F

Famille Anderson

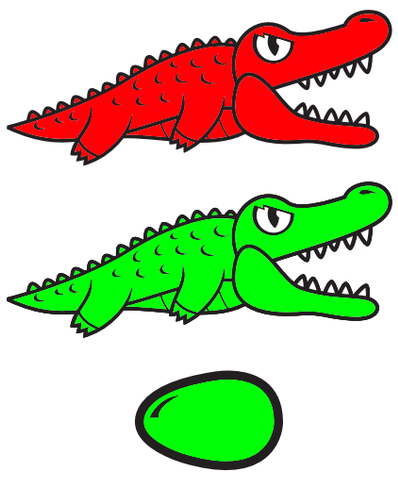
	T	**F**
T	T	F
F	F	F

Des alligators à la logique booléenne

Thompson



Foster



N : negation ¬

T	F
F	T

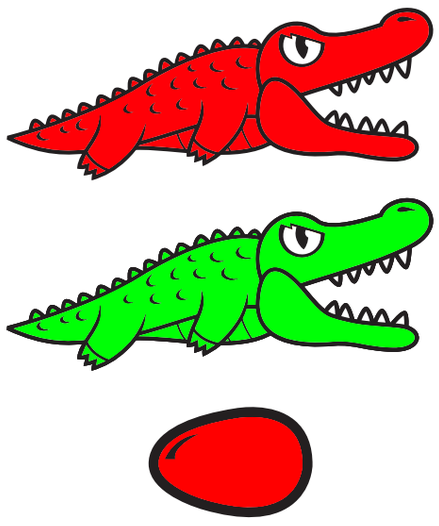
Or : disjonction

∨	**T**	**F**
T	T	T
F	T	F

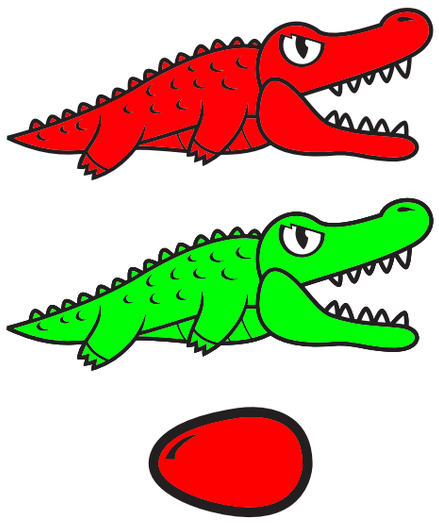
And : conjonction

∧	**T**	**F**
T	T	F
F	F	F

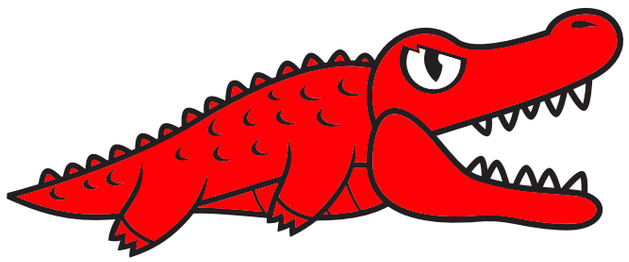
Comment écrire plus aisément les familles d'alligators ?



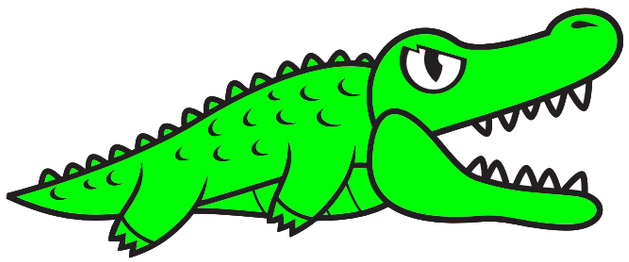
Comment écrire plus aisément les familles d'alligators ?



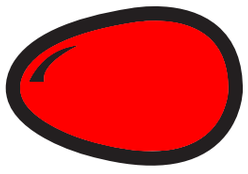
Problème n°1 : les couleurs



x

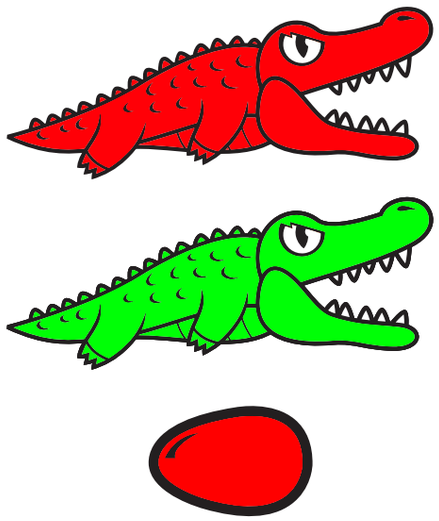


y

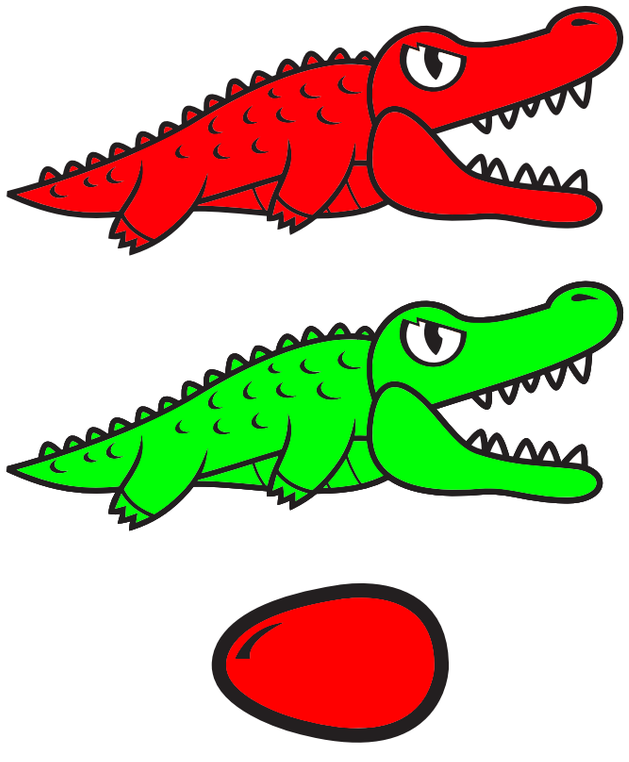


x

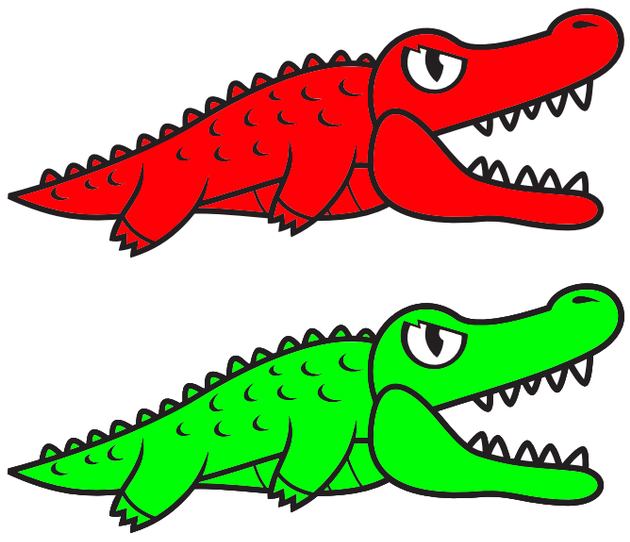
Comment écrire plus aisément les familles d'alligators ?



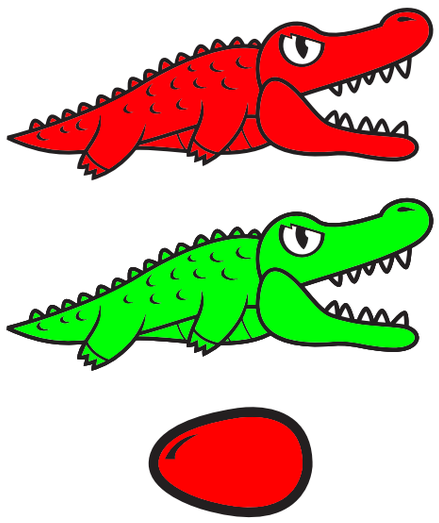
Problème n°1 : les couleurs



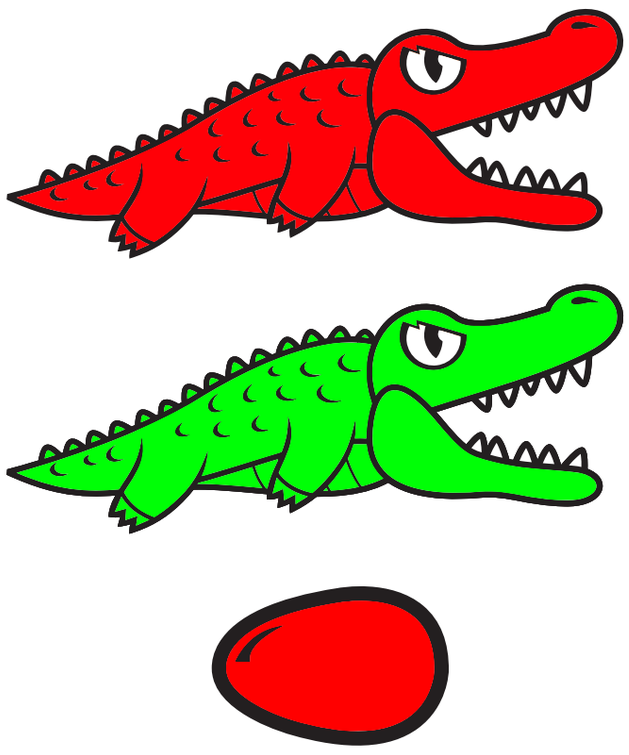
Problème n°2 : les dessins



Comment écrire plus aisément les familles d'alligators ?



Problème n°1 : les couleurs

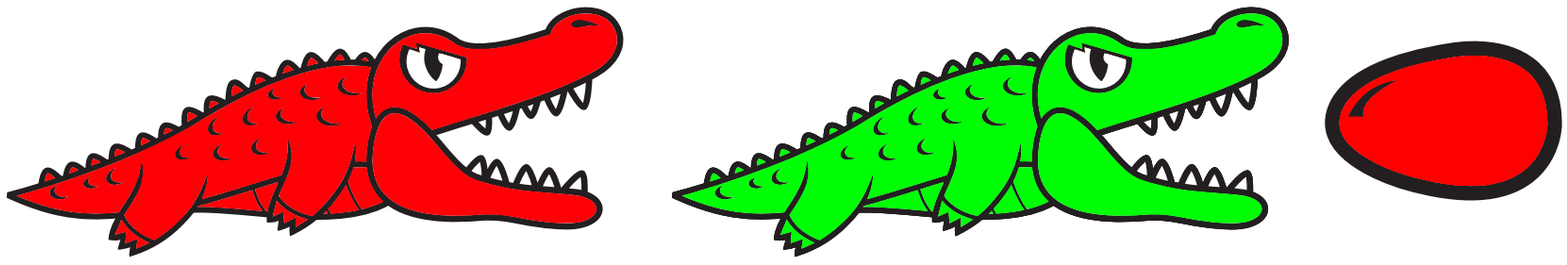


x

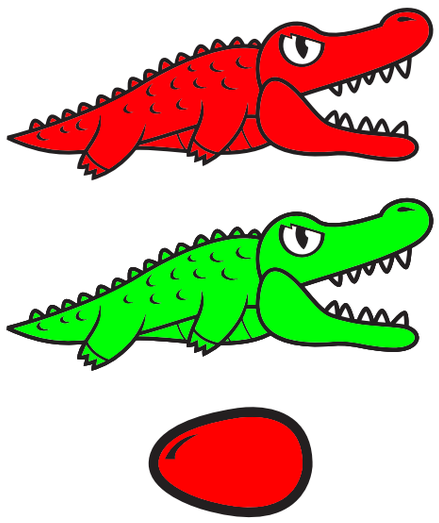
y

x

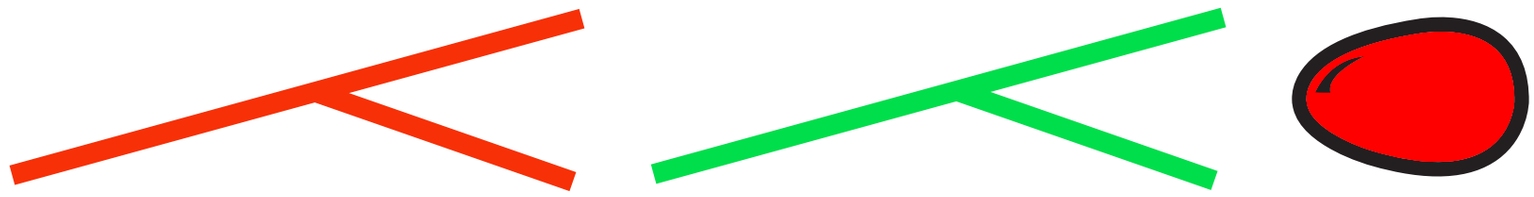
Problème n°2 : les dessins



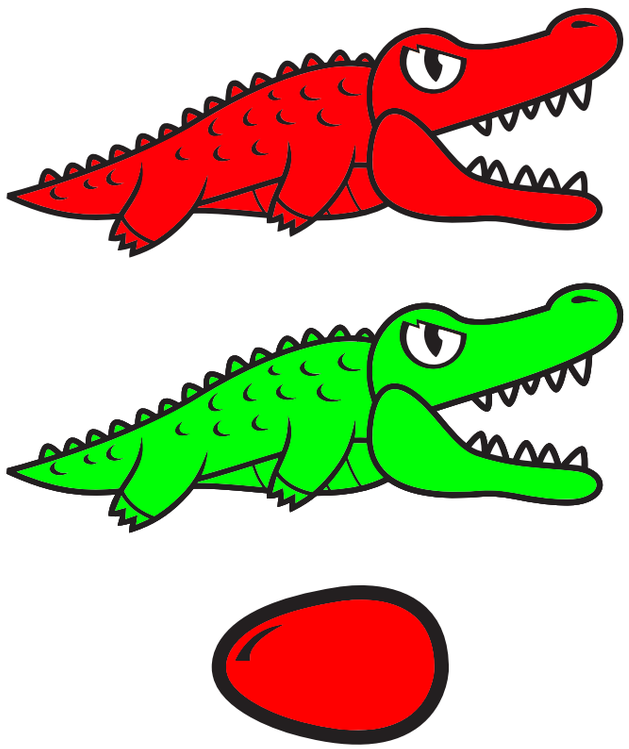
Comment écrire plus aisément les familles d'alligators ?



Problème n°2 : les dessins



Problème n°1 : les couleurs

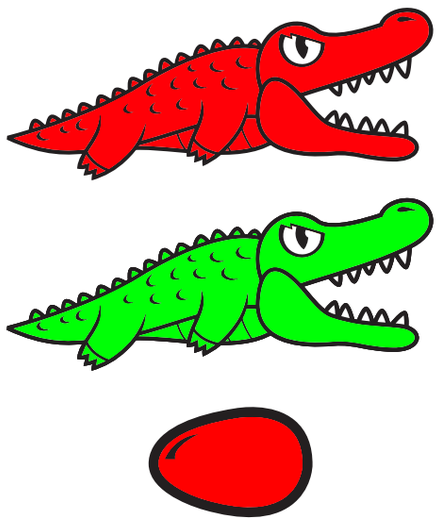


x

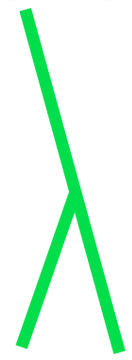
y

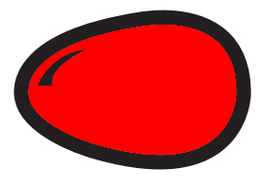
x

Comment écrire plus aisément les familles d'alligators ?

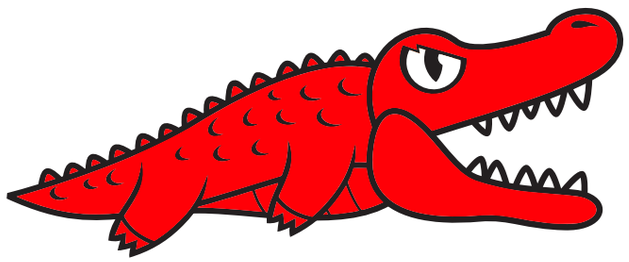


Problème n°2 : les dessins

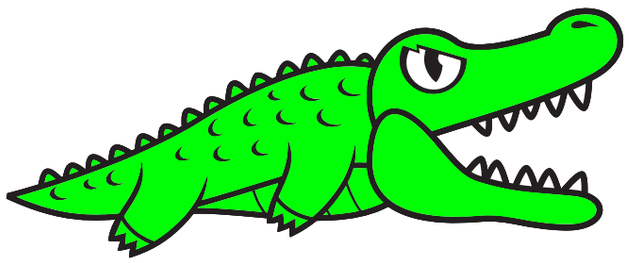




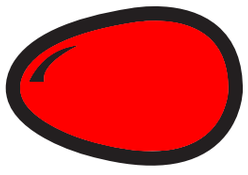
Problème n°1 : les couleurs



x

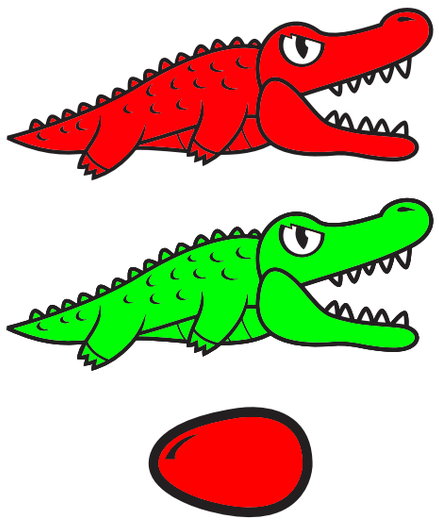


y

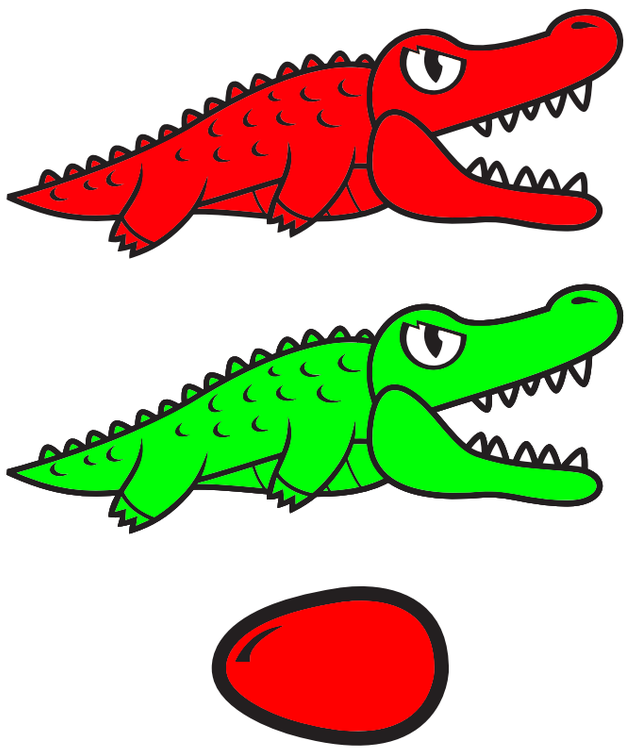


x

Comment écrire plus aisément les familles d'alligators ?



Problème n°1 : les couleurs

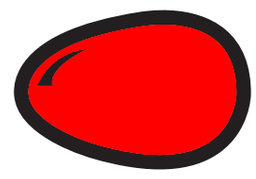


x

y

x

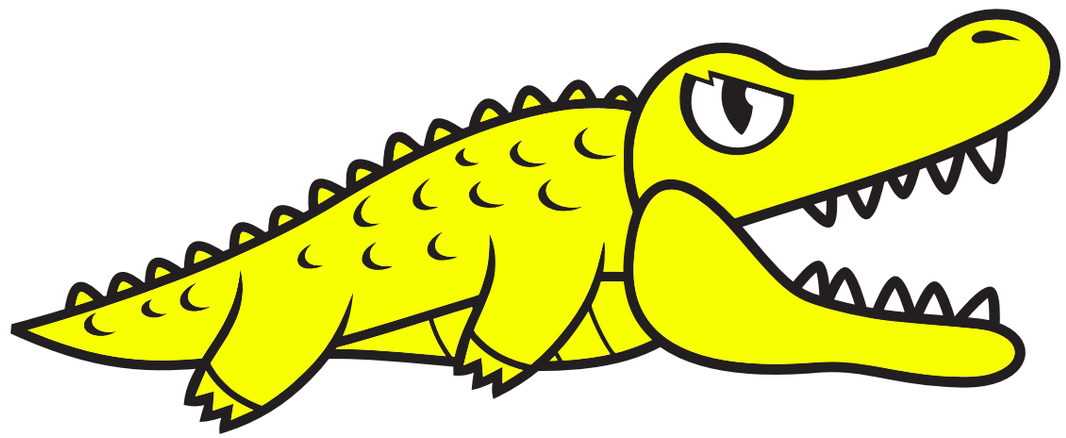
Problème n°2 : les dessins

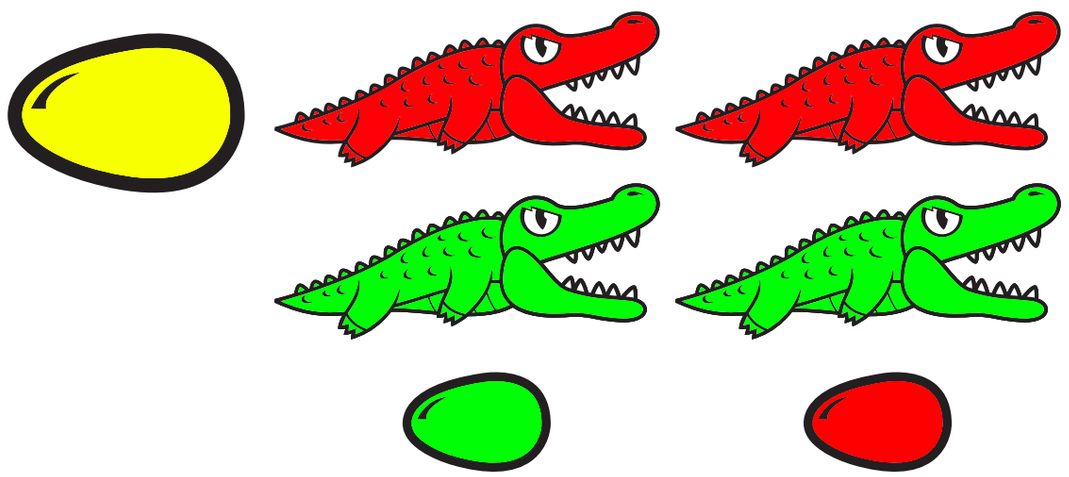


Solution : $\lambda x\, \lambda y\, x$

Comment écrire plus aisément les familles d'alligators ?

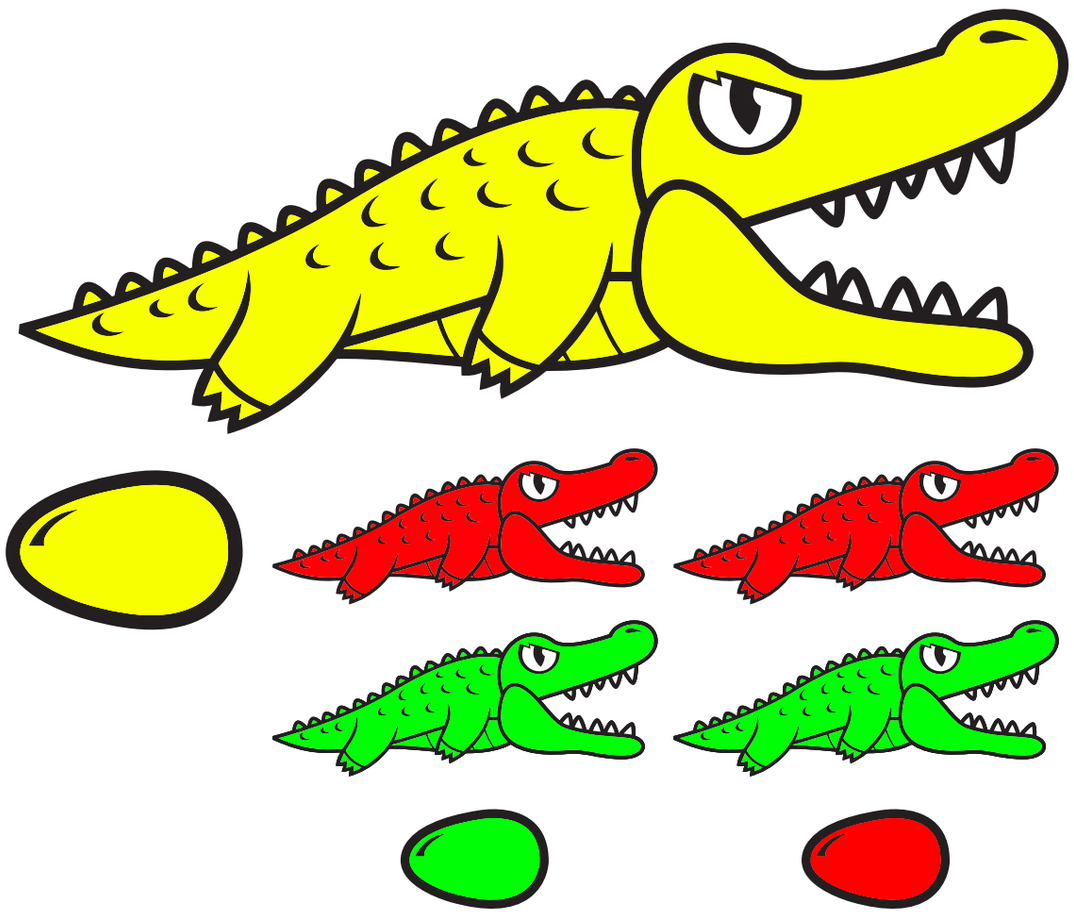
Famille Nelson





Comment écrire plus aisément les familles d'alligators ?

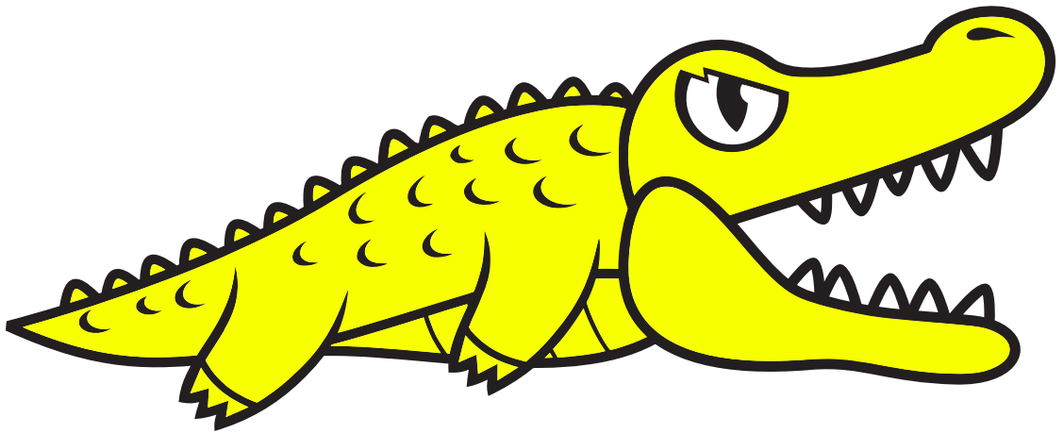
Famille Nelson

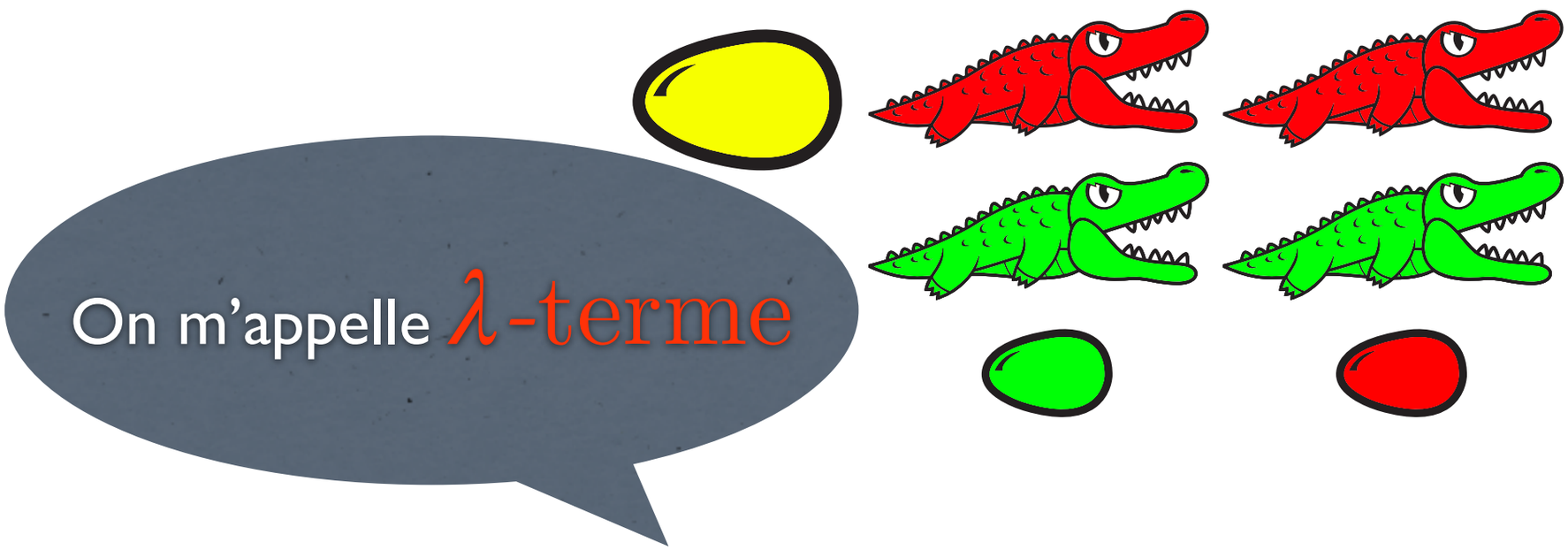


$$\lambda z \quad z\ (\lambda x\ \lambda y\ y) \quad (\lambda x\ \lambda y\ x)$$

Comment écrire plus aisément les familles d'alligators ?

Famille Nelson

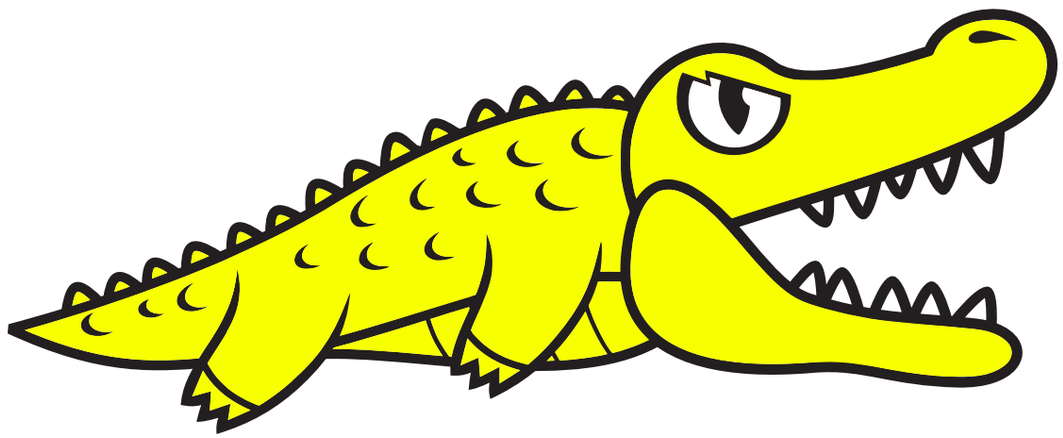


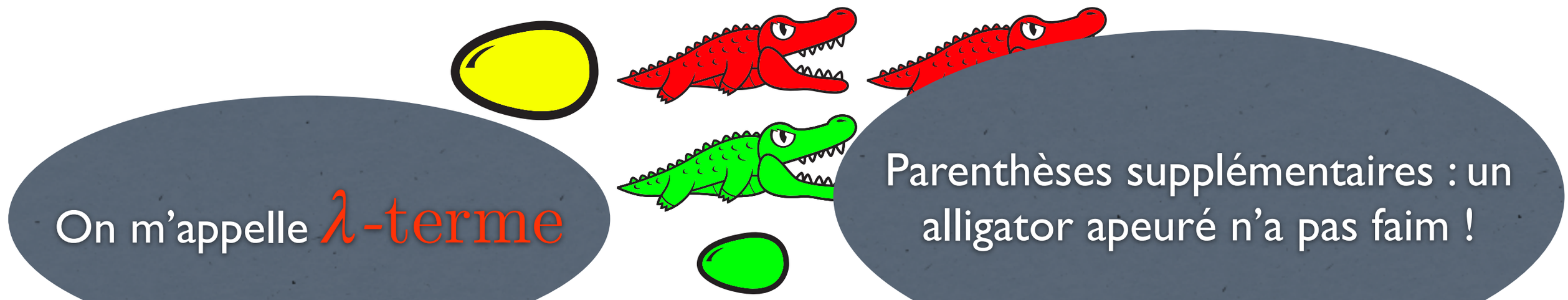


$$\lambda z \;\; z\,(\lambda x\,\lambda y\;\, y)\;\;\; (\lambda x\,\lambda y\;\, x)$$

Comment écrire plus aisément les familles d'alligators ?

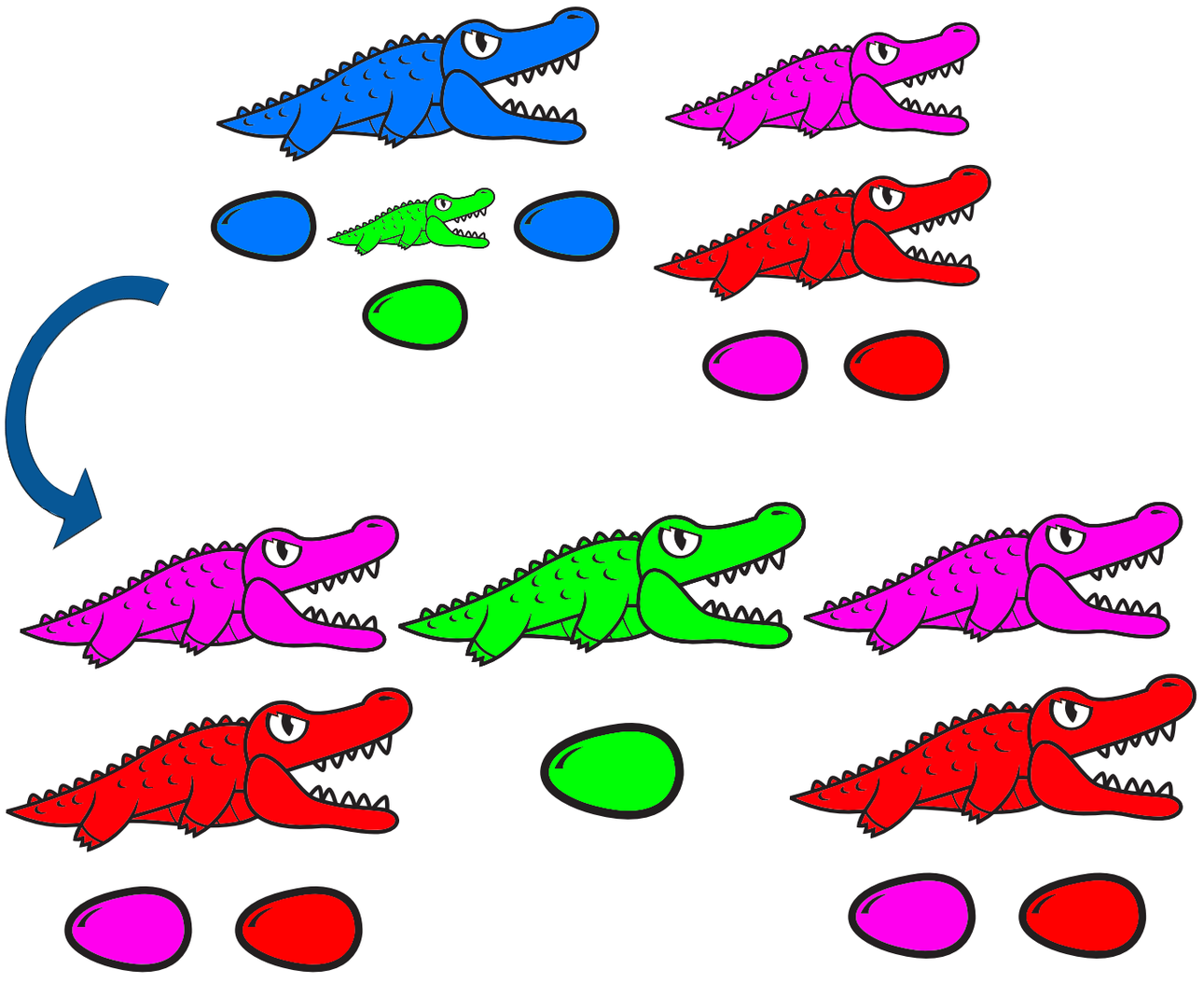
Famille Nelson





$$\lambda z\,\big(z\,(\lambda x\,\lambda y\;y)\big)\;(\lambda x\,\lambda y\;x)$$

Règle de repas : β-réduction

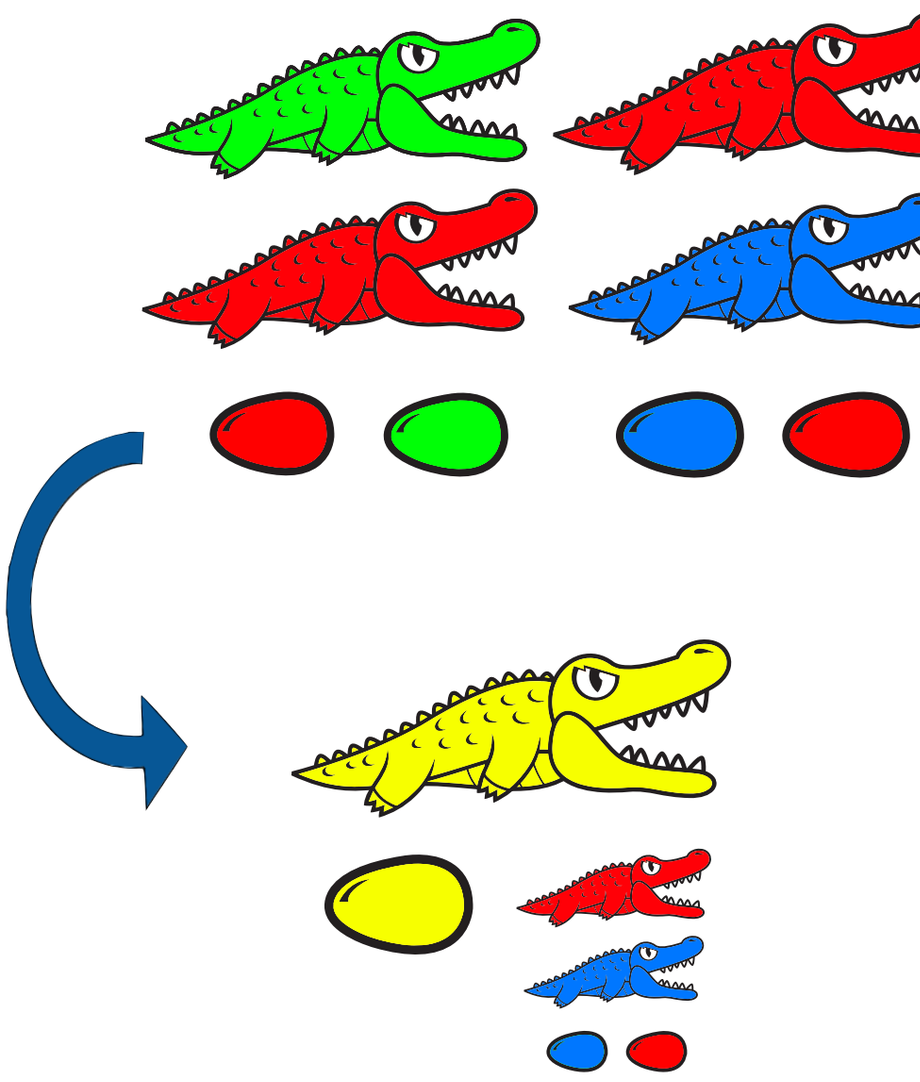


$$(\lambda x\ t)\ u$$

$$t[x := u]$$

λ-terme t où toute occurrence de x est remplacée par le λ-terme u

Règle de couleur : α-conversion

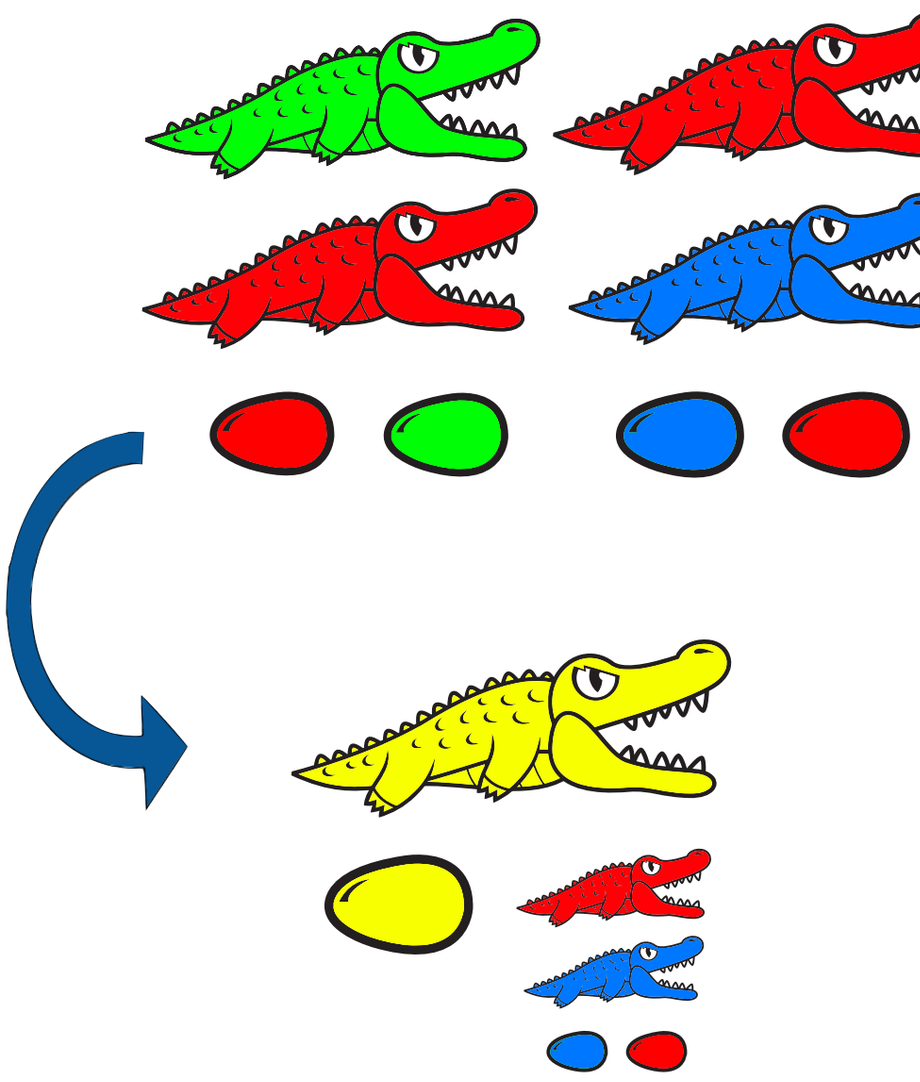


λ-terme t sans la variable y

$$\lambda x\ t$$

$$\lambda y\ t[x := y]$$

Règle de couleur : α-conversion



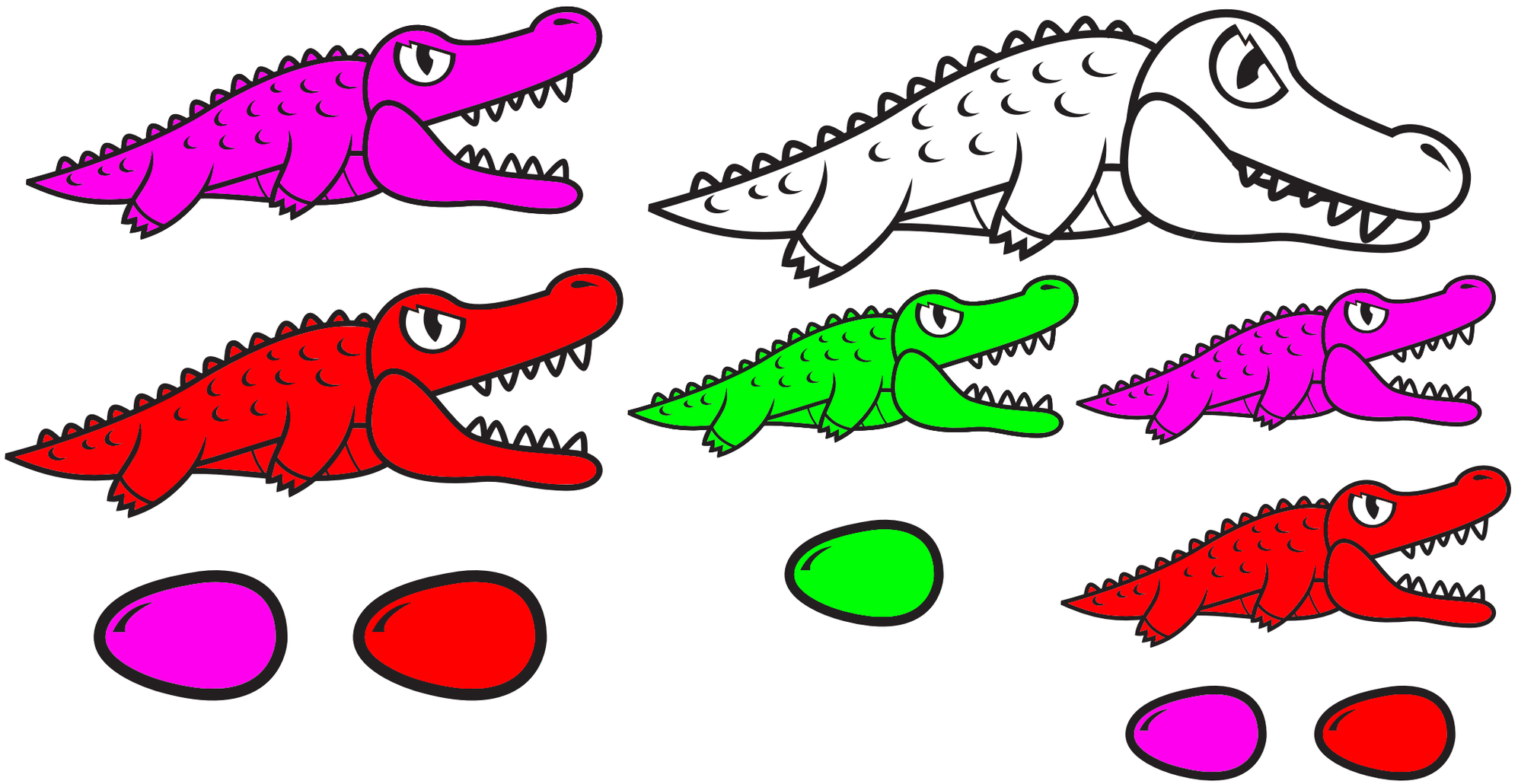
λ-terme t sans la variable y

$$\lambda x\ t$$

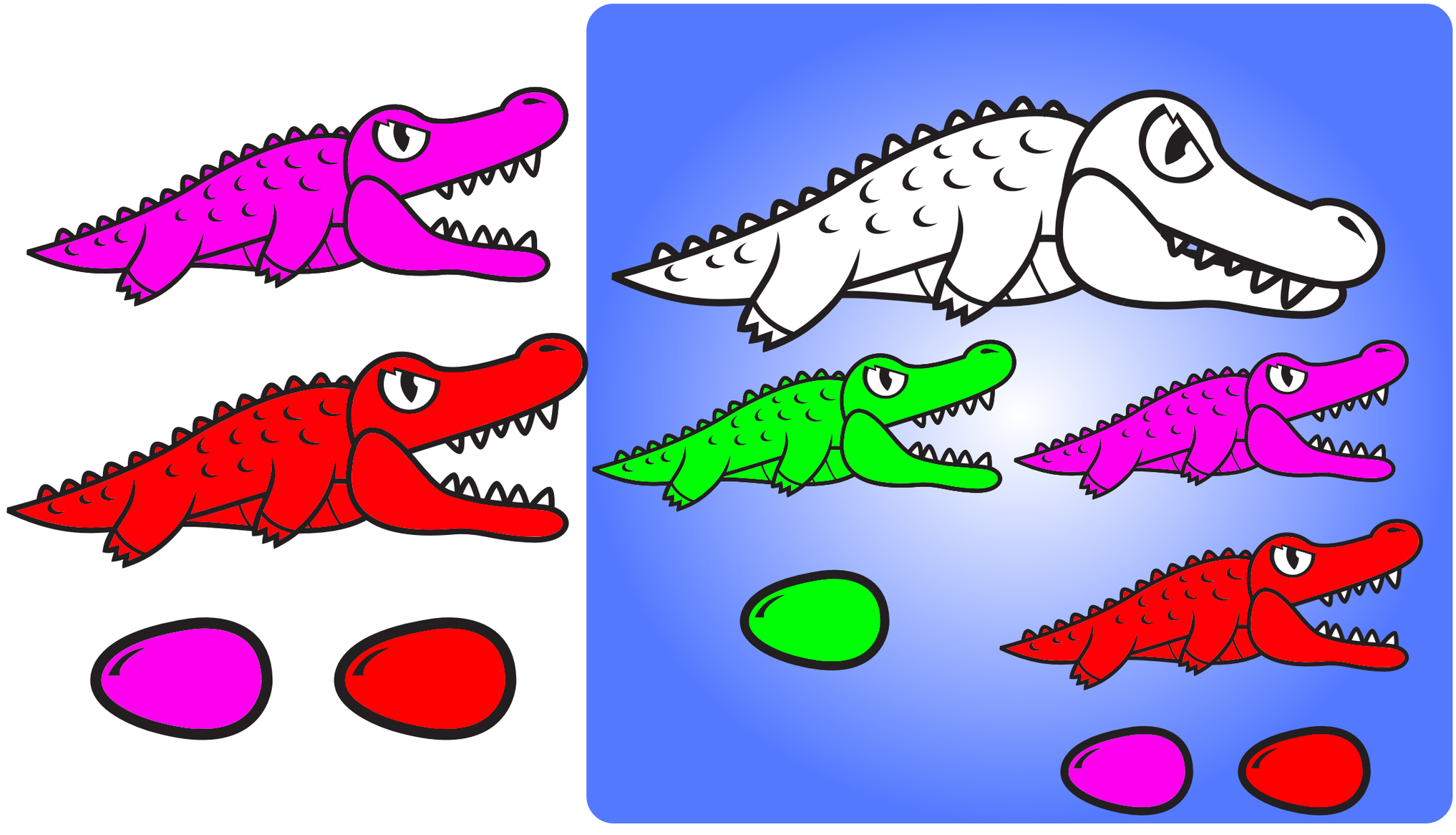
$$\lambda y\ t[x := y]$$

On peut appliquer une règle à tout endroit !

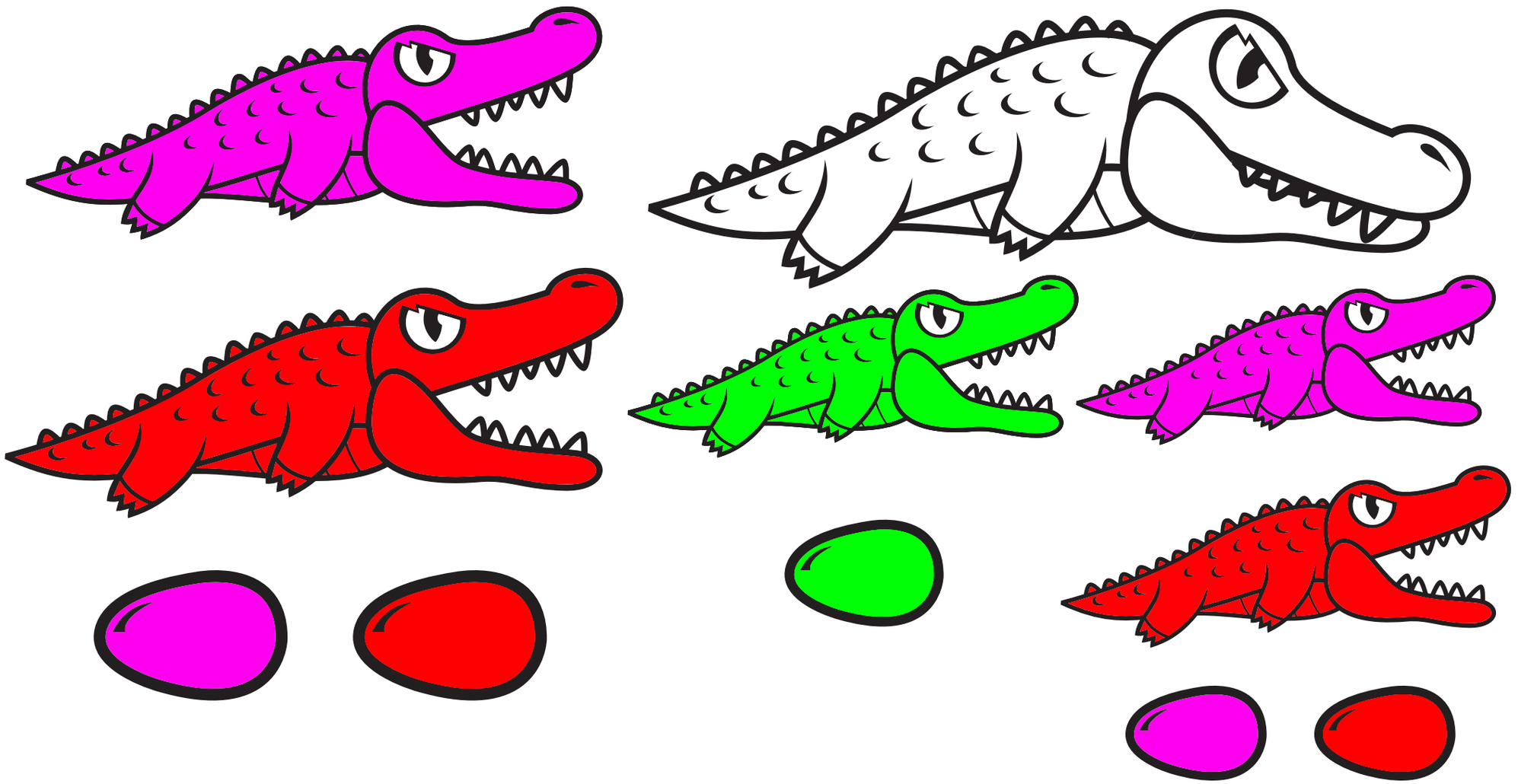
Alligators âgés : des parenthèses en plus



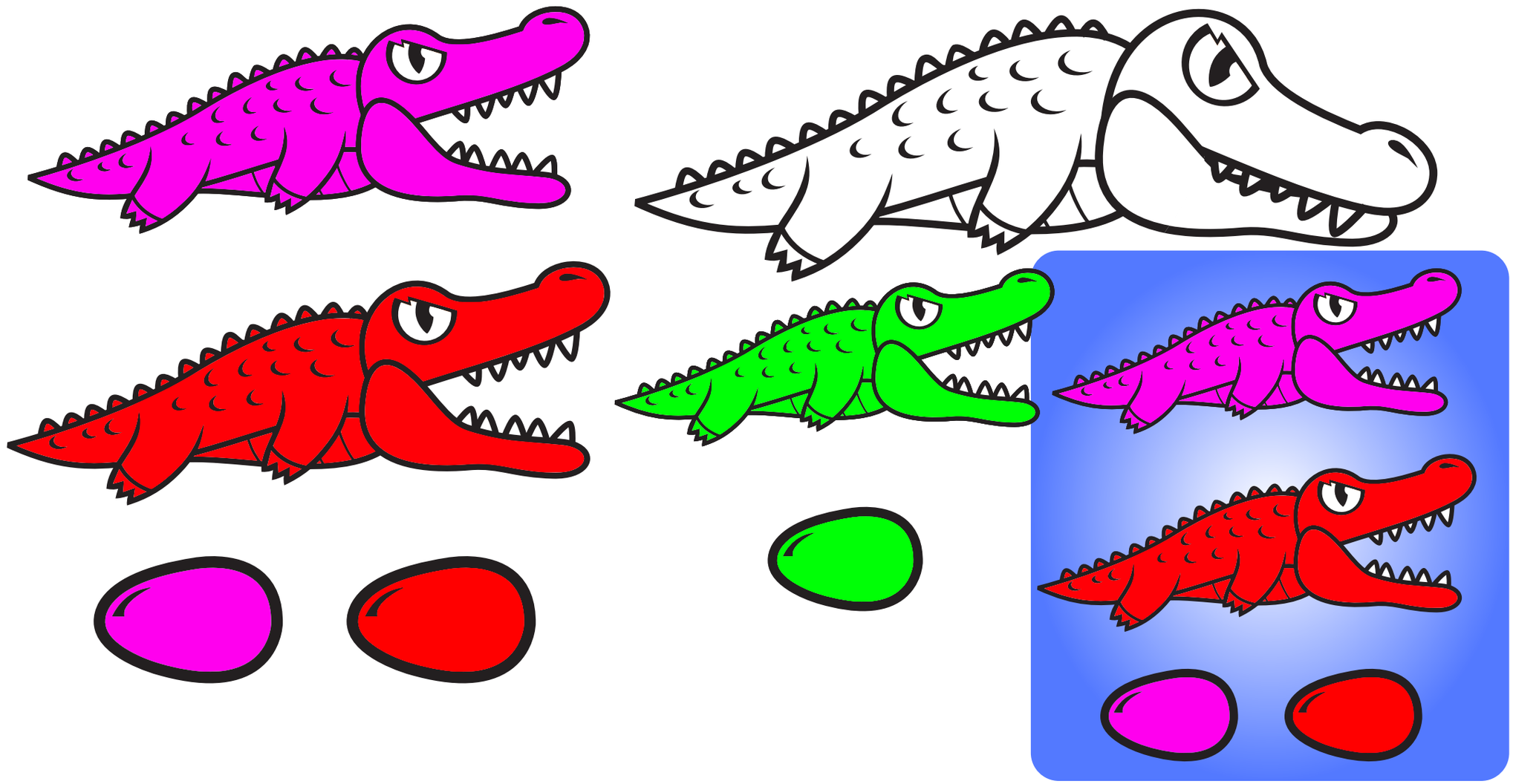
Alligators âgés : des parenthèses en plus



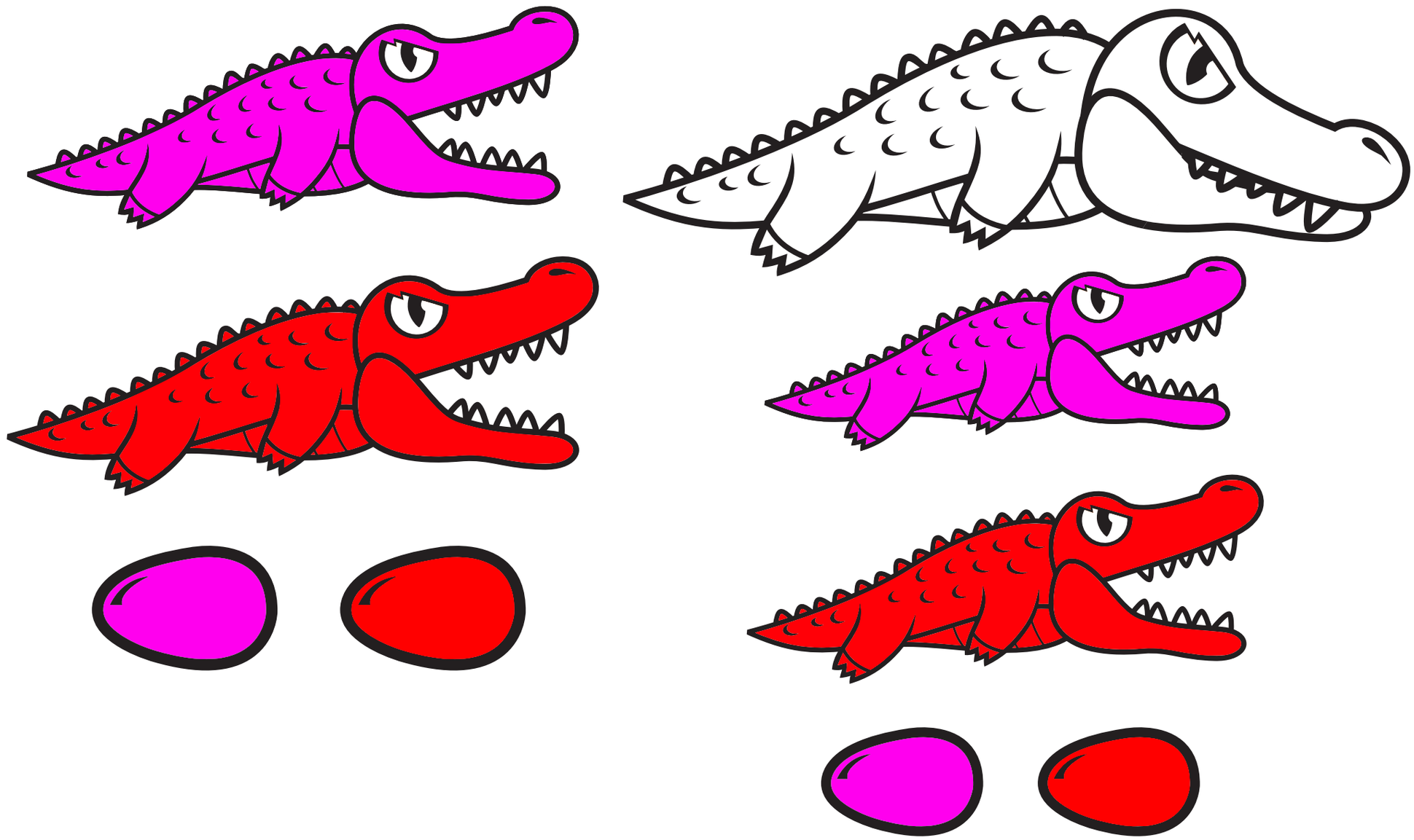
Alligators âgés : des parenthèses en plus



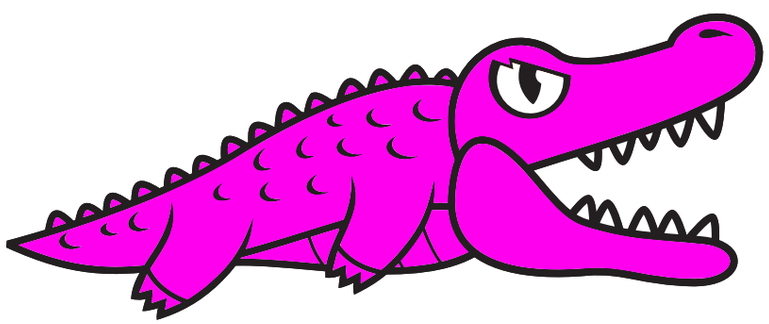
Alligators âgés : des parenthèses en plus

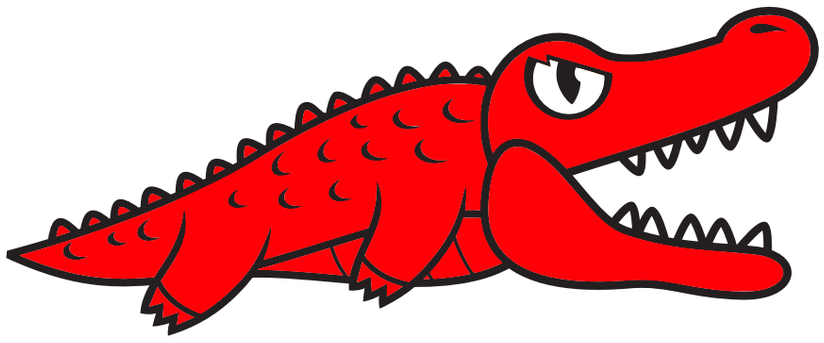


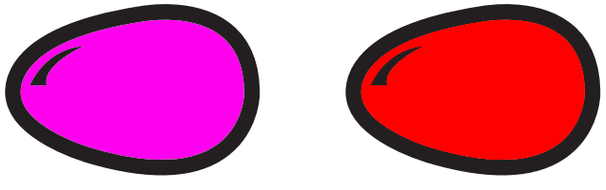
Alligators âgés : des parenthèses en plus

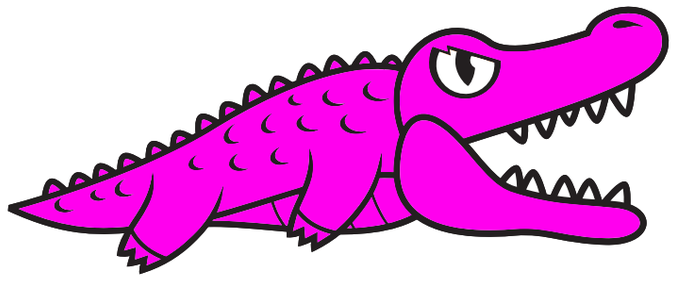


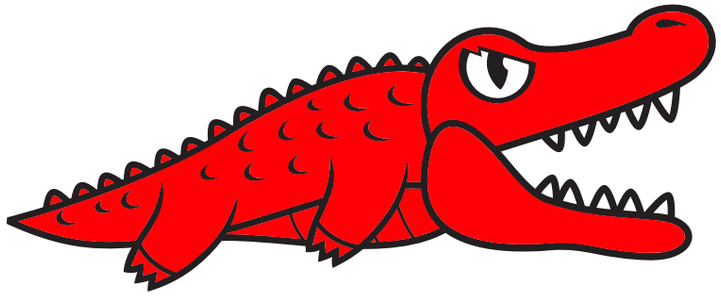
Alligators âgés : des parenthèses en plus

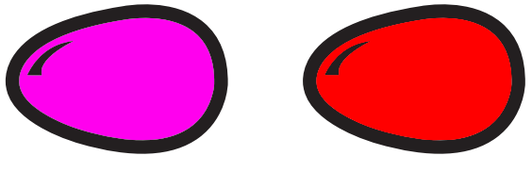




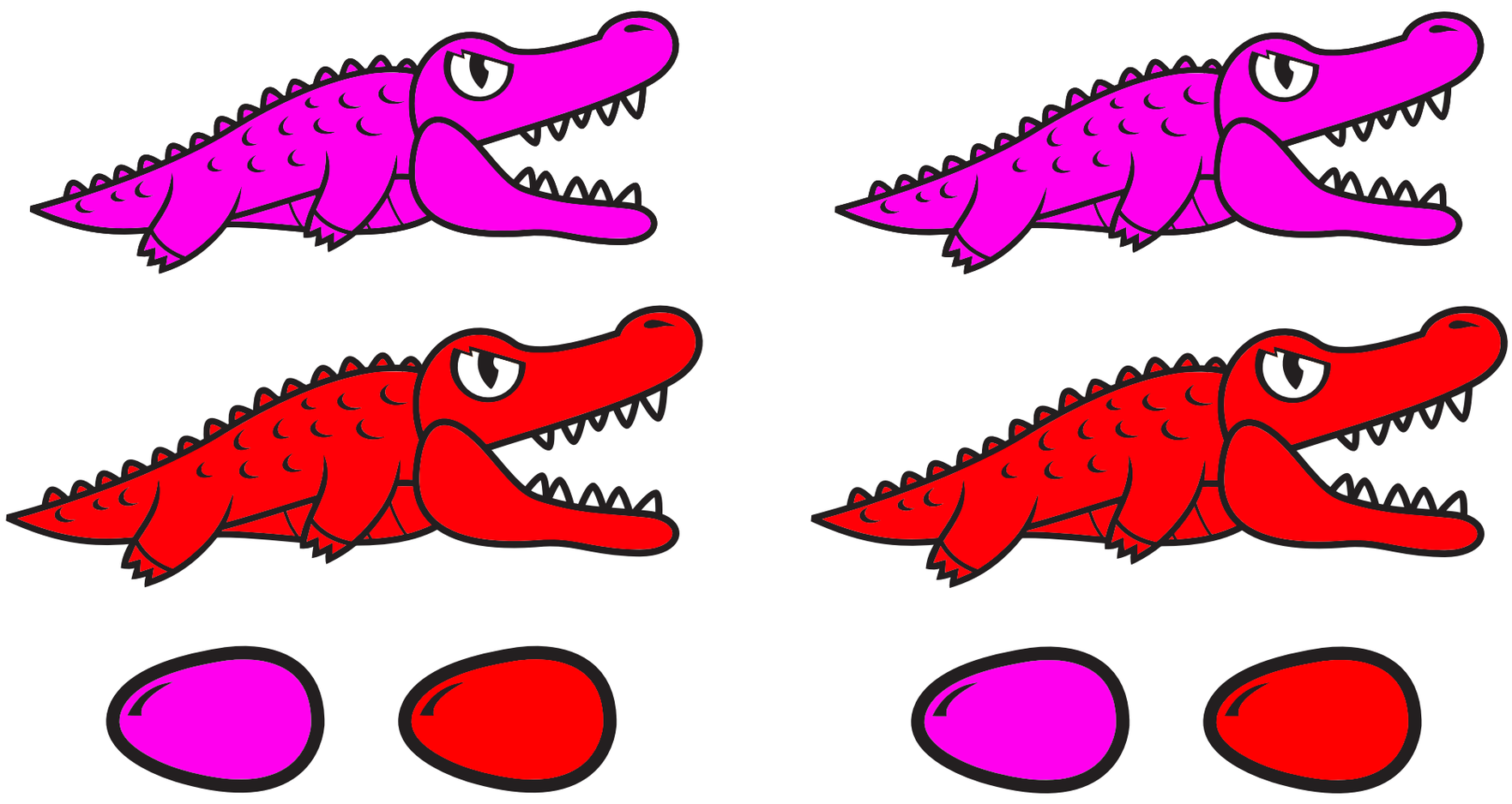






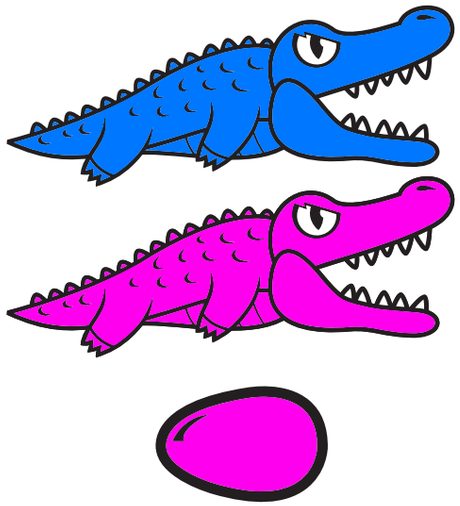


Alligators âgés : des parenthèses en plus



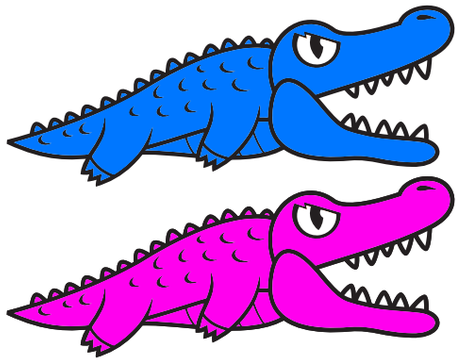
Trop de familles : numérotons-les

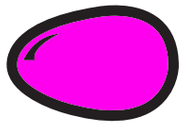
Famille 0



Trop de familles : numérotons-les

Famille 0

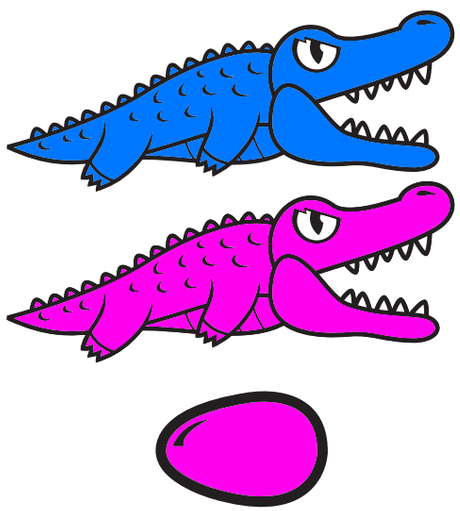




$$0 = \lambda x\ \lambda y\ y$$

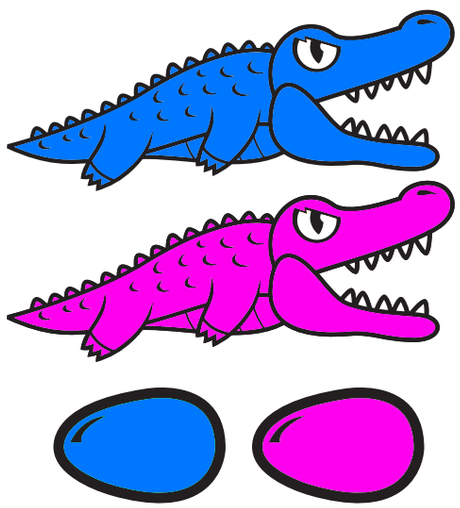
Trop de familles : numérotons-les

Famille 0



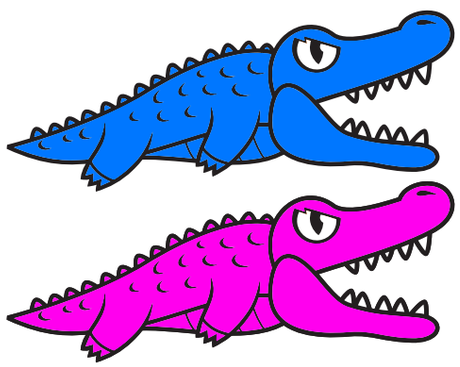
$0 = \lambda x\ \lambda y\ y$

Famille 1



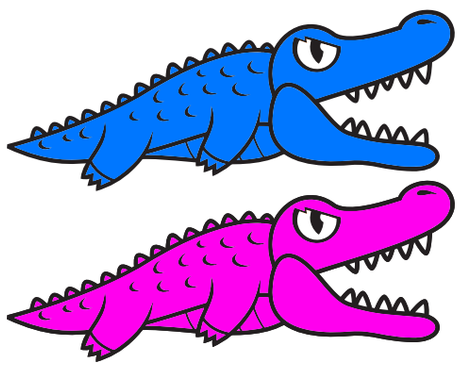
Trop de familles : numérotons-les

Famille 0



$0 = \lambda x\ \lambda y\ y$

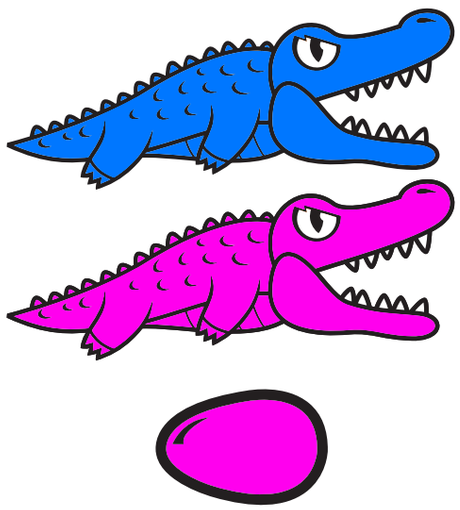
Famille 1



$1 = \lambda x\ \lambda y\ x\ y$

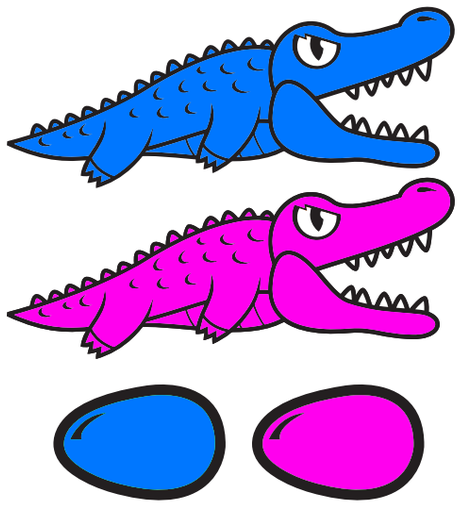
Trop de familles : numérotons-les

Famille 0



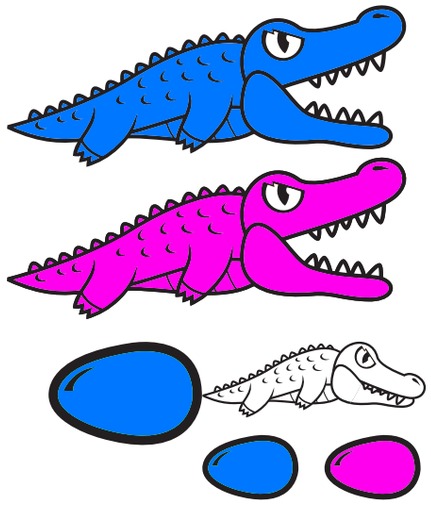
$0 = \lambda x\ \lambda y\ y$

Famille 1



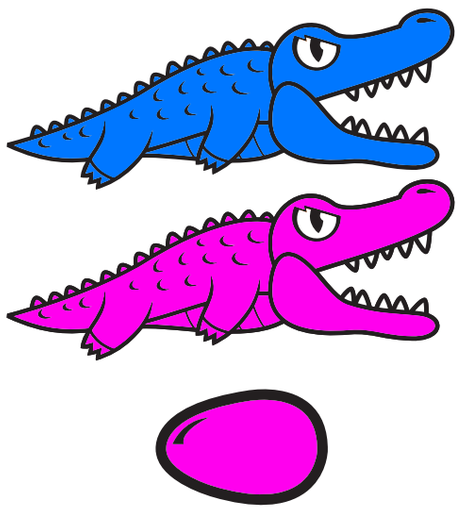
$1 = \lambda x\ \lambda y\ x\ y$

Famille 2



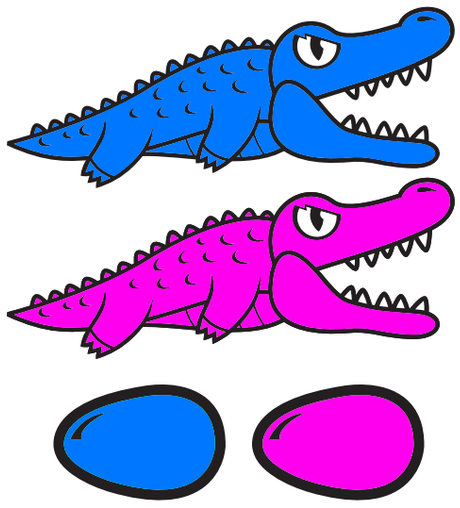
Trop de familles : numérotons-les

Famille 0



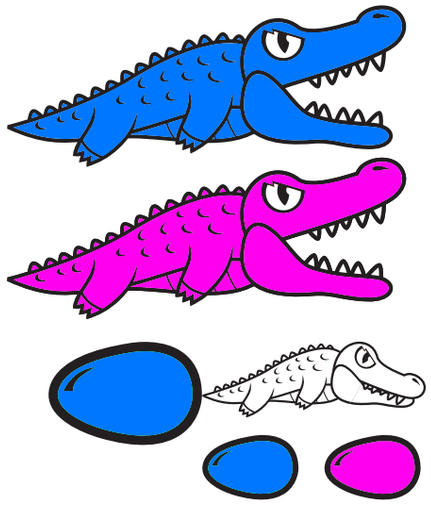
$0 = \lambda x\ \lambda y\ y$

Famille 1



$1 = \lambda x\ \lambda y\ x\ y$

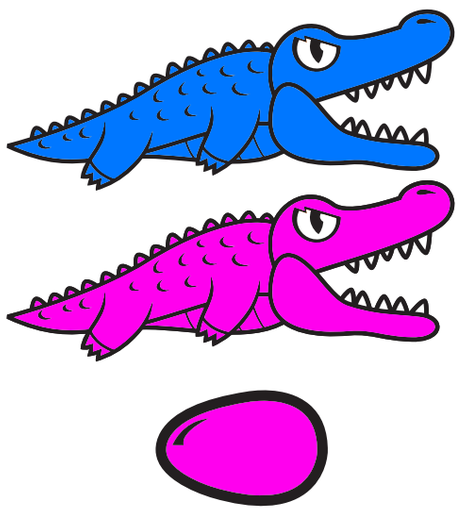
Famille 2



$2 = \lambda x\ \lambda y\ x\ (x\ y)$

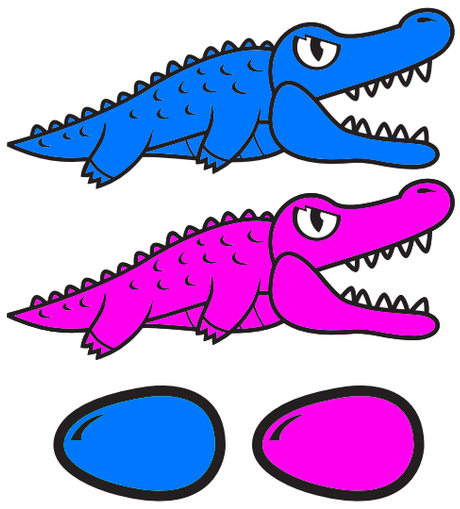
Trop de familles : numérotons-les

Famille 0



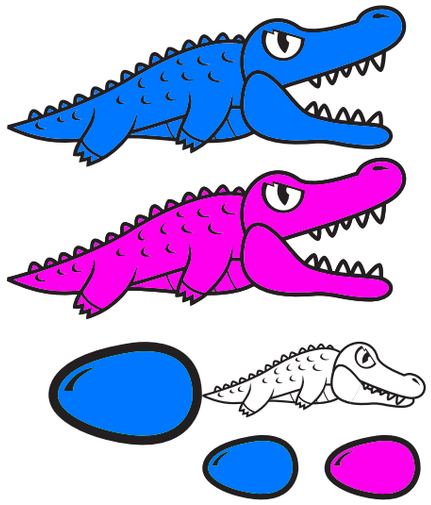
$0 = \lambda x\ \lambda y\ y$

Famille 1



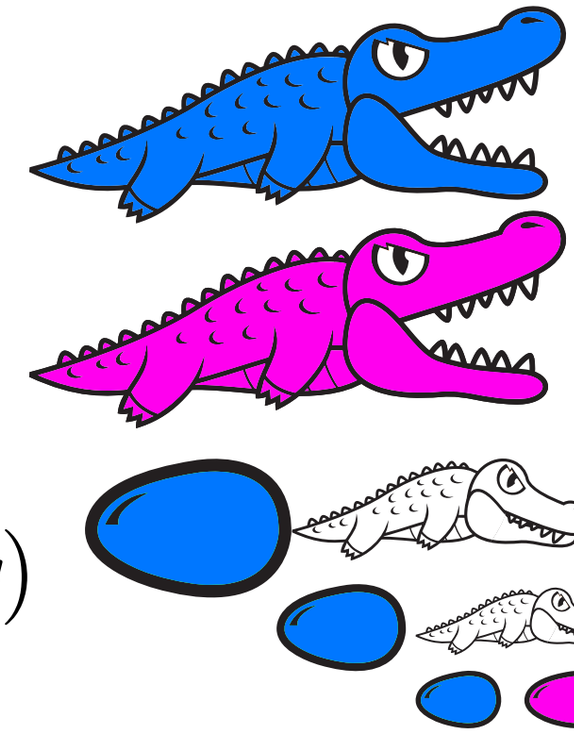
$1 = \lambda x\ \lambda y\ x\ y$

Famille 2



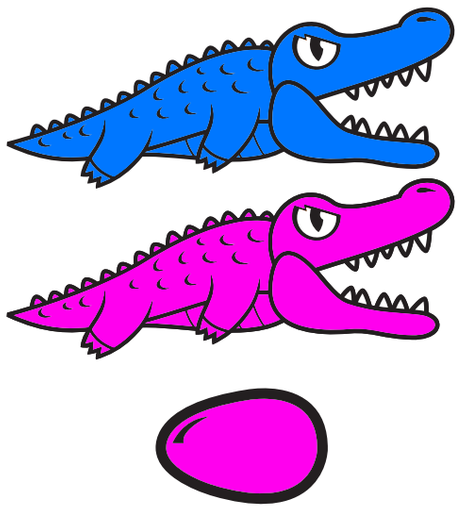
$2 = \lambda x\ \lambda y\ x\ (x\ y)$

Famille 3



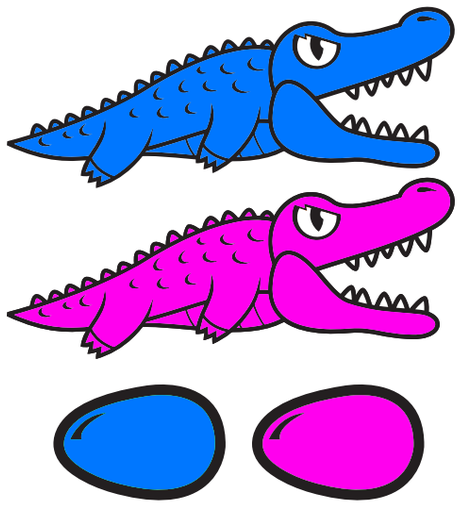
Trop de familles : numérotons-les

Famille 0



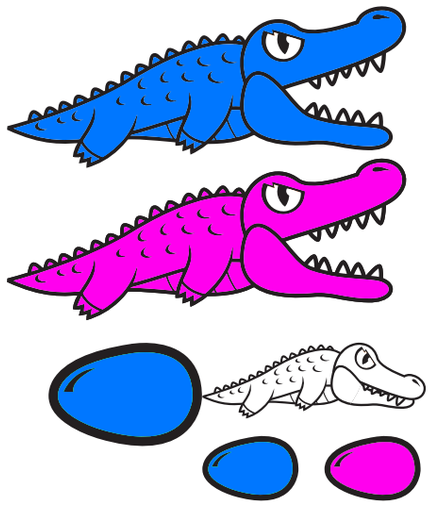
$0 = \lambda x\ \lambda y\ y$

Famille 1



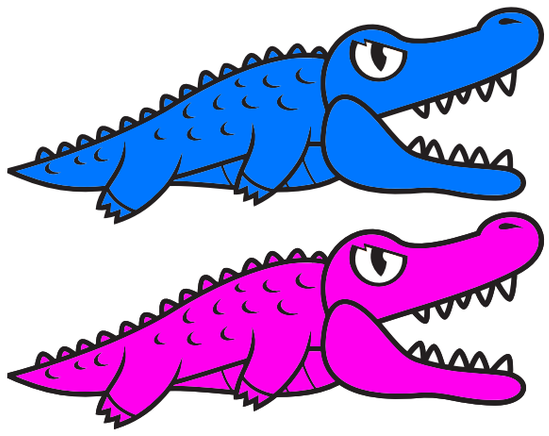
$1 = \lambda x\ \lambda y\ x\ y$

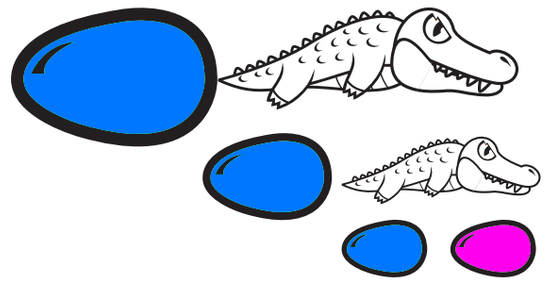
Famille 2



$2 = \lambda x\ \lambda y\ x\ (x\ y)$

Famille 3





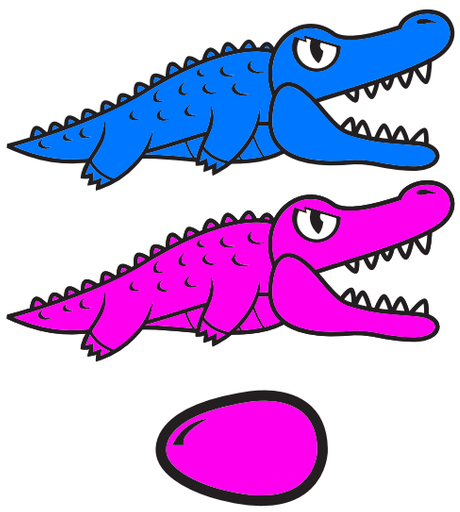
$3 = \lambda x\ \lambda y\ x\ (x\ (x\ y))$

…

…

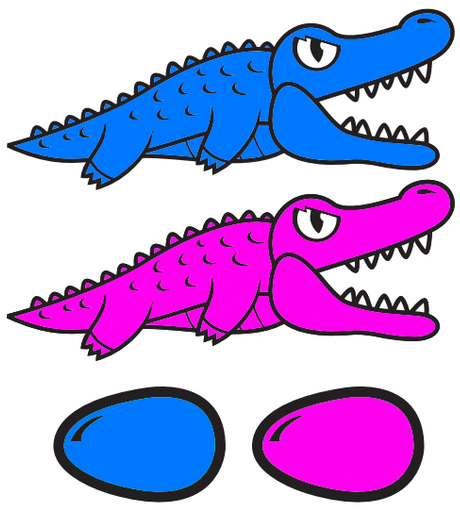
Trop de familles : numérotons-les

Famille 0



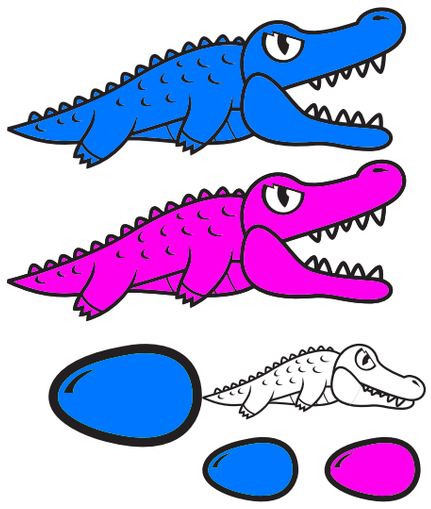
$0 = \lambda x\ \lambda y\ y$

Famille 1



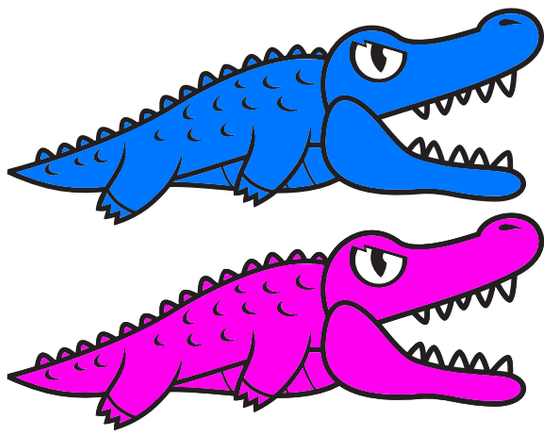
$1 = \lambda x\ \lambda y\ x\ y$

Famille 2

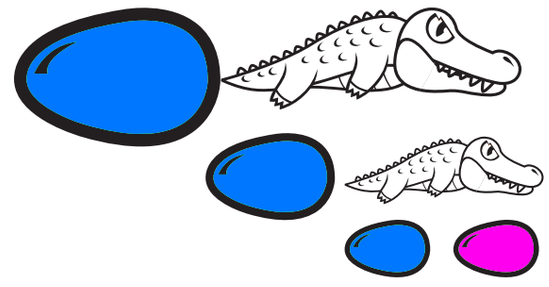


$2 = \lambda x\ \lambda y\ x\ (x\ y)$

Famille 3



…

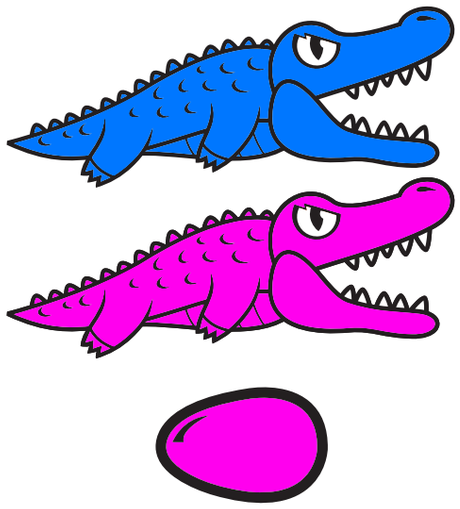


$3 = \lambda x\ \lambda y\ x\ (x\ (x\ y))$

Pouvez-vous trouver une famille qui, après application des règles, transforme la famille *n* en la famille *n*+1 ?

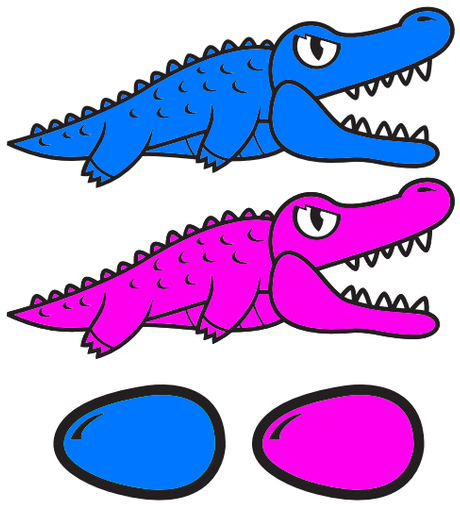
Trop de familles : numérotons-les

Famille 0



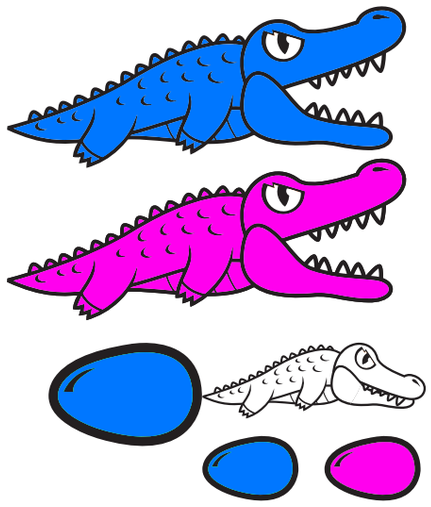
$0 = \lambda x\ \lambda y\ y$

Famille 1



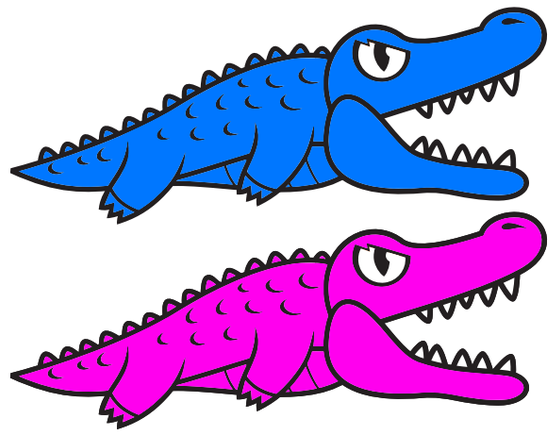
$1 = \lambda x\ \lambda y\ x\ y$

Famille 2

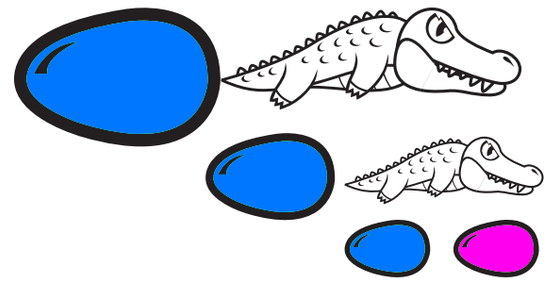


$2 = \lambda x\ \lambda y\ x\ (x\ y)$

Famille 3



…



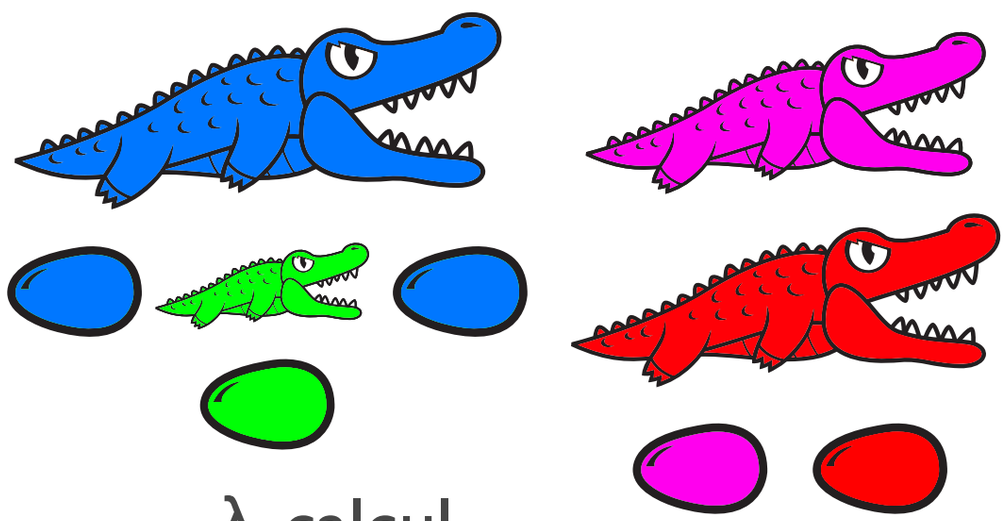
$3 = \lambda x\ \lambda y\ x\ (x\ (x\ y))$

Pouvez-vous trouver une famille qui, après application des règles, transforme la famille *n* en la famille *n*+1 ?

c'est-à-dire un λ-terme t tel que pour tout n, $t\ n$ se réduit en $(n{+}1)$

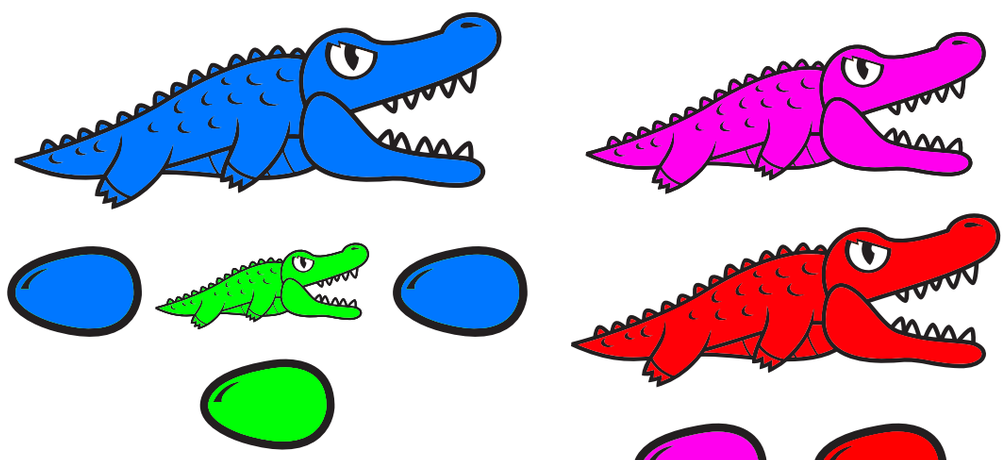
Outil en ligne pour produire des familles d’alligator et les faire évoluer…

https://gitlab.lis-lab.fr/benjamin.monmege/alligators-python



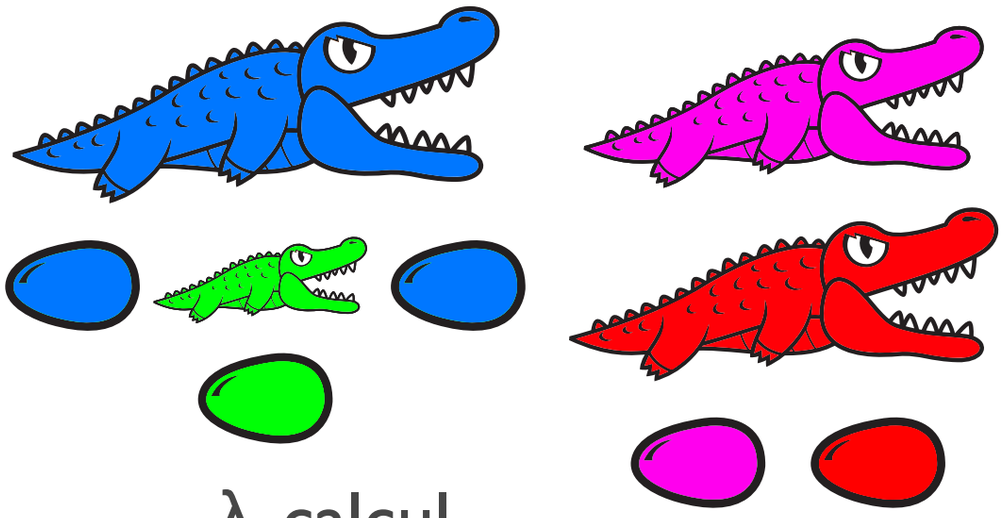
λ-calcul

Alonzo Church
(1903-1995)

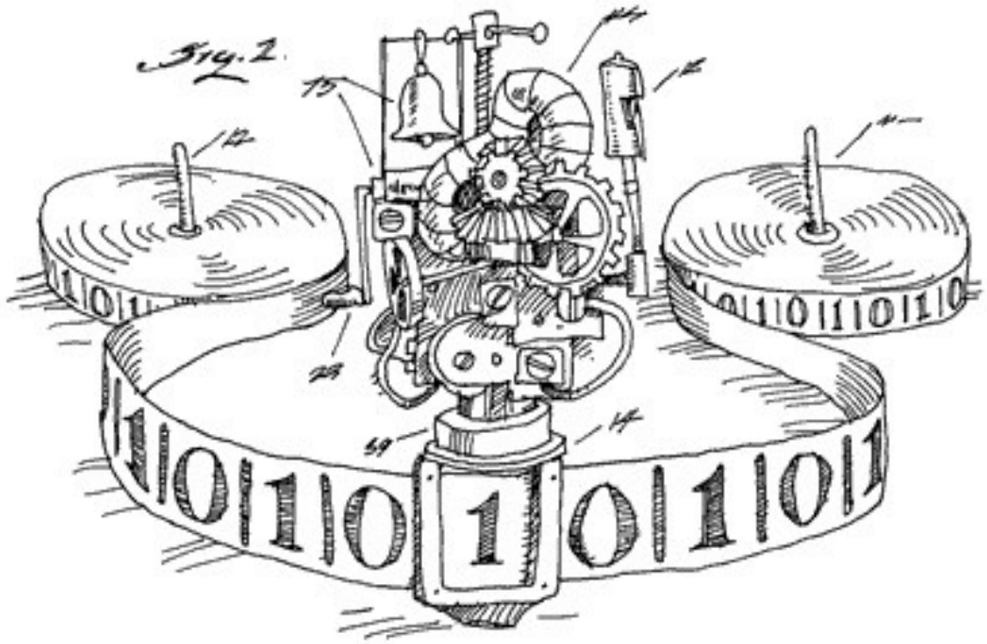


λ-calcul

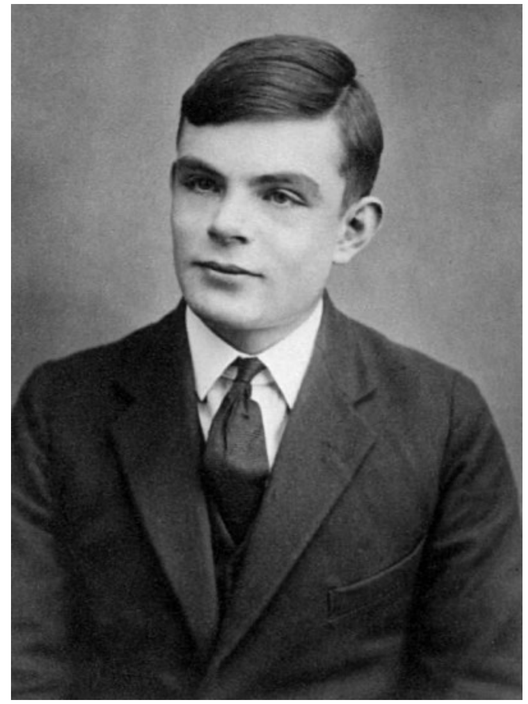
Alonzo Church
(1903-1995)



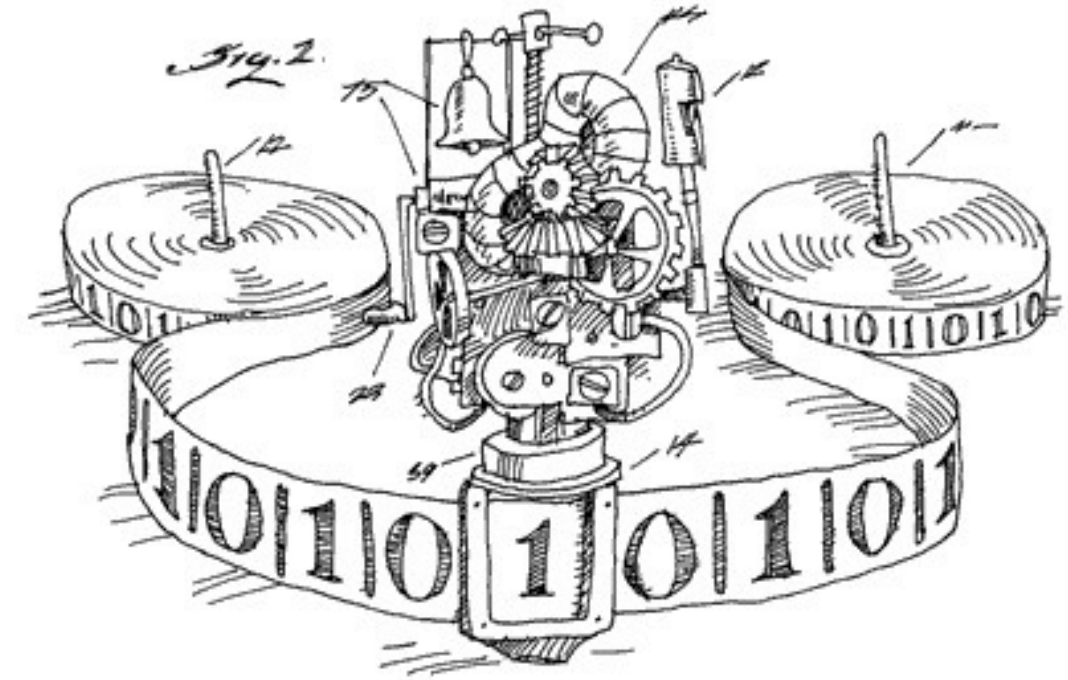
λ-calcul



Machines de Turing

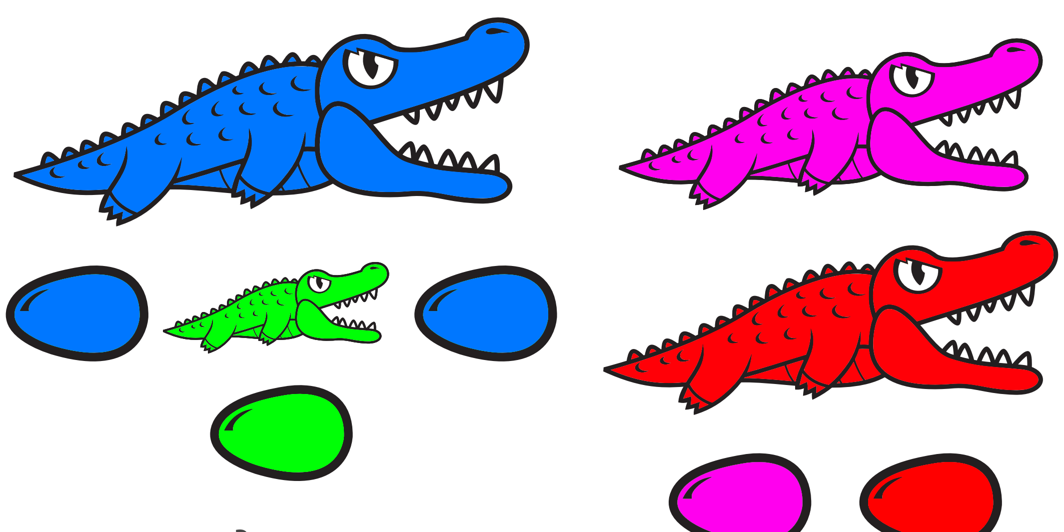


Alan Turing
(1912-1954)

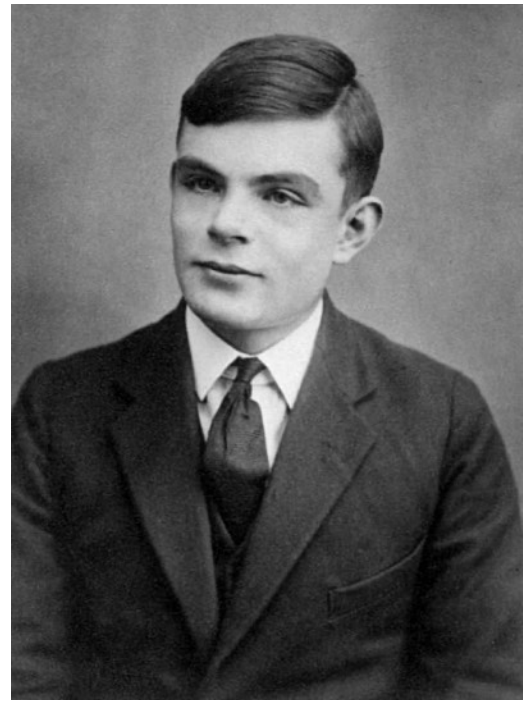


Machines de Turing

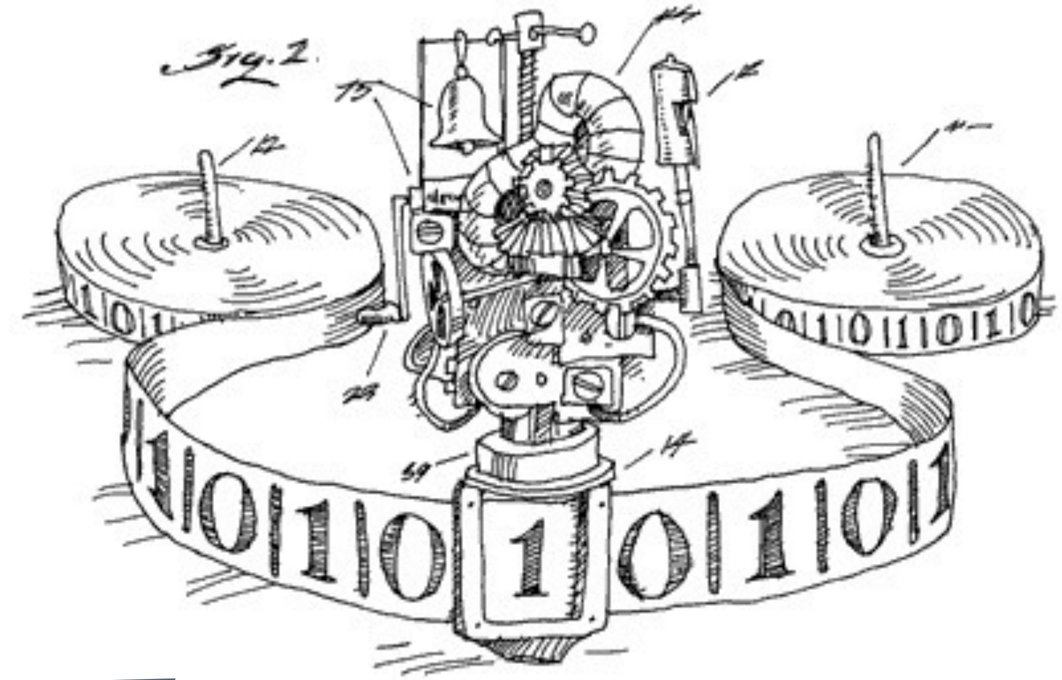
Alonzo Church
(1903-1995)



λ-calcul

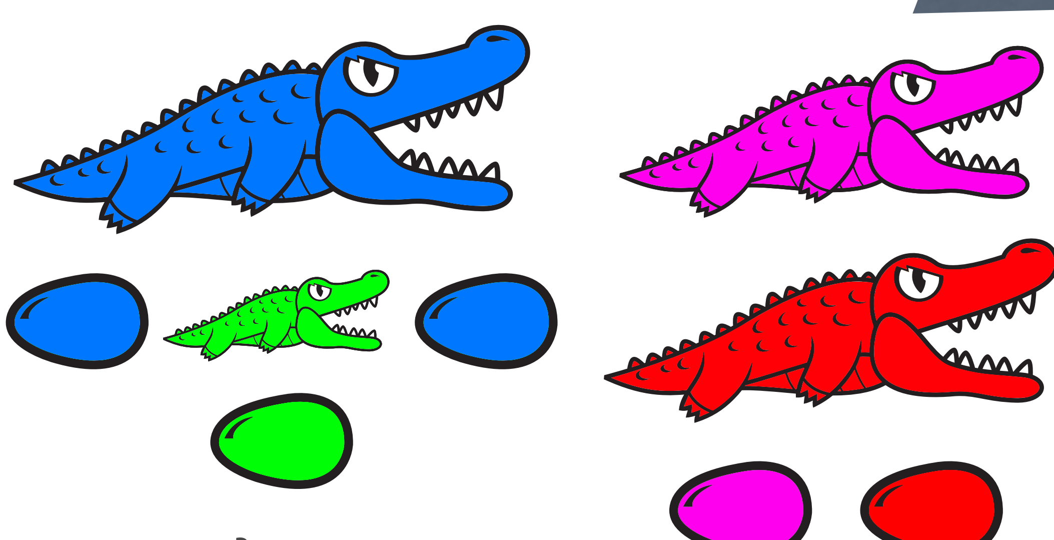


Alan Turing
(1912-1954)



Machines de Turing

Alonzo Church
(1903-1995)



λ-calcul

Thèse de Church-Turing

Lien avec la programmation (en Python)

```
def appliquer(f, x):
    return f(x)
```

Lien avec la programmation (en Python)

```
def appliquer(f, x):
    return f(x)
```

```
def carré(x):
    return x**2
```

Lien avec la programmation (en Python)

```
def appliquer(f, x):
    return f(x)
```

```
def carré(x):
    return x**2
```

```
>>> appliquer(carré, 2)
```

Lien avec la programmation (en Python)

```
def appliquer(f, x):
    return f(x)
```

```
def carré(x):
    return x**2
```

```
>>> appliquer(carré, 2)
4
```

Lien avec la programmation (en Python)

```
def appliquer(f, x):
    return f(x)
```

```
def carré(x):
    return x**2
```

```
>>> appliquer(carré, 2)
4
>>> appliquer(carré, 3)
```

Lien avec la programmation (en Python)

```
def appliquer(f, x):
    return f(x)
```

```
def carré(x):
    return x**2
```

```
>>> appliquer(carré, 2)
4
>>> appliquer(carré, 3)
9
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> appliquer(composer(carré, successeur), 2)
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> appliquer(composer(carré, successeur), 2)
9
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> appliquer(composer(carré, successeur), 2)
9
>>> appliquer(composer(successeur, carré), 2)
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> appliquer(composer(carré, successeur), 2)
9
>>> appliquer(composer(successeur, carré), 2)
5
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> composer(carré, successeur)(2)
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> composer(carré, successeur)(2)
9
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> composer(carré, successeur)(2)
9
>>> composer(successeur, carré)(2)
```

Programmes en sortie

```
def composer(f, g):
    def h(x):
        return f(g(x))
    return h
```

```
def carré(x):
    return x**2

def successeur(x):
    return x + 1
```

```
>>> composer(carré, successeur)(2)
9
>>> composer(successeur, carré)(2)
5
```

Un exemple plus intéressant : la fonction dérivée

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                    # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                    # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                    # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                    # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
0
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                    # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
0 0.001
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                    # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
0 0.001
1
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                    # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
0 0.001
1 2.0009999999996975
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                    # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
0 0.001
1 2.0009999999996975
2
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                    # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
0 0.001
1 2.0009999999996975
2 4.000999999999699
```

Approximation de la dérivée

```
def dérivée(f):
    h = 0.001                       # petit
    def f_prime(x):
        return (f(x+h) - f(x)) / h
    return f_prime
```

```
>>> g = dérivée(carré)
>>> for x in range(5): print(x, g(x))
0 0.001
1 2.0009999999996975
2 4.000999999999699
3 6.000999999999479
4 8.001000000000037
```

Programmation fonctionnelle en Python :

lambda expressions, map, filter

à voir sur le Jupyter Notebook