



Developpez

Le Mag

Édition de octobre – novembre 2015

Numéro 60

Magazine en ligne gratuit

Diffusion de copies conformes à l'original autorisée

Réalisation : Alexandre Pottiez

Rédaction : la rédaction de Developpez

Contact : magazine@redaction-developpez.com

Sommaire

Pascal	Page	2
Delphi	Page	17
Windows	Page	26
CRM	Page	27
Qt	Page	41
Libres & Open Source	Page	42
Perl	Page	58

Éditorial

Le magazine des développeurs est de retour avec de nouveaux articles, de nouvelles rubriques, de nouvelles news. La rédaction est heureuse de vous présenter un florilège de ses meilleures ressources.

La rédaction

Article Delphi



Expressions régulières avec Delphi

En lisant cet article vous apprendrez à vous servir des expressions régulières avec l'unité RegularExpressions de Delphi par [Roland Chastain](#)

Page 17



Article Libre & Open Source

Scratch : création d'un mini-jeu

Nous allons apprendre à déplacer notre personnage le « Héros », et faire se déplacer plus ou moins intelligemment les personnages non joueurs (PNJ) de notre jeu vidéo.

par [Christophe Thomas et Vincent Viale](#)

Page 42



Pascal

Les derniers tutoriels et articles

Concepts en algorithmique Implémentés en Pascal

Les méthodes de conception des structures de données permettent de résoudre des problèmes particuliers comme l'élaboration d'un questionnaire arborescent pour un système de classification (faune, flore, etc.), ou d'une structure de base pour la représentation des solides dans un système de C.A.O.

Les méthodes utilisées conduisent à s'interroger sur les « objets » fondamentaux qui interviennent dans le problème. Elles peuvent être très rigoureuses, fondées par exemple sur la logique, ou sur l'algèbre comme les « types abstraits algébriques ».

Dans les chapitres 1 et 2, nous verrons comment spécifier des structures de données, et comment les implémenter dans un langage de programmation particulier comme Pascal. Nous étudierons en même temps le comportement des structures de données modélisé par des opérations abstraites. L'écriture des programmes se rapproche des manières humaines de poser et résoudre les problèmes, par l'élaboration de fonctionnalités de plus en plus abstraites, qui garantissent leur indépendance vis-à-vis de la mise en œuvre de fonctionnalités plus « primitives ».

Dans le chapitre 3, nous étudierons la technique de preuve de programme. Prouver un programme consiste à prouver que la spécification est conforme à l'algorithme.

1 Implémentation et manipulation de structures algébriques isomorphes à l'ensemble des entiers naturels \mathbb{N} muni de lois

On souhaite écrire un programme qui effectue des opérations sur les éléments d'un ensemble E ayant les mêmes propriétés que l'ensemble des entiers naturels \mathbb{N} . Nous allons démontrer qu'écrire un tel programme revient à implémenter le type des entiers naturels \mathbb{N} et à concevoir les opérations qui manipulent ce type. La conception du programme sera facile, car l'ensemble des entiers naturels nous est familier.

C'est l'ensemble « de base », nous connaissons bien ses propriétés. Rappelons que \mathbb{N} est l'ensemble des nombres entiers supérieurs ou égal à 0.

1.1 Propriétés des entiers naturels

Nous allons dans un premier temps énoncer les propriétés de \mathbb{N} ; c'est ce qui nous servira à créer les primitives opérant sur le type \mathbb{N} .

1. Il existe un élément qui est le plus petit élément de \mathbb{N} . Nous appellerons cet élément Zéro. $\text{Zéro} \in \mathbb{N}$ - Zéro est une fonction constante.
2. Il existe une fonction successeur dont la signature est $\mathbb{N} \rightarrow \mathbb{N}$. $\text{successeur}(n)$ est appelé « le successeur de n » :

3. Il existe une fonction booléenne notée Égal qui teste l'égalité de deux entiers. Sa signature est $\mathbb{N} \times \mathbb{N} \rightarrow \text{booléen}$. $\text{Égal}(n1, n2)$ a la valeur vrai si $n1$ et $n2$ sont égaux.

- a - $\text{successeur}(n) \neq n$ ($\text{successeur}(n)$ est différent de n) ;
- b - Zéro n'est le successeur d'aucun nombre.

En résumé : propriétés de \mathbb{N}



1. $\text{Zéro} \in \mathbb{N}$;
2. il existe une fonction successeur telle que :
 - a - $\text{successeur}(n) \neq n$;
 - b - Zéro n'est le successeur d'aucun nombre.

Ces deux axiomes suffisent pour définir l'ensemble \mathbb{N} .

Connaissant l'ensemble \mathbb{N} et ses propriétés, l'écriture des primitives sera facile.

1.2 Conception des primitives

Le type des entiers naturels TEntier sera implémenté par le type `integer` (ou `longint`).

On écrira un module MEntierPrim comportant le type TEntier et les primitives Zéro, Successeur et Égal.

```

1  unit MEntier_Prim ;
2
3  interface
4
5  type TEntier = integer ;
6  function Zero : TEntier ;
7  function Successeur ( n : TEntier ) :
   TEntier ;
8  function Egal ( n1 , n2 : TEntier ) :
   boolean ;
9
10 implementation
11
12 function Zero : TEntier ;
13 begin
14   Zero := 0 ;
15 end ;
16
17
18 function Successeur ( n : TEntier ) :
   TEntier ;
19 begin
20   Successeur := n + 1 ;
21 end ;
22
23
24 function Egal ( n1 , n2 : TEntier ) :
   boolean ;
25 begin
26   Egal := ( n1 = n2 ) ;
27 end ;
28
29 end.
```

1.3 Opérations complexes sur les entiers basées sur les primitives

Pour effectuer des calculs sur les entiers, nous aurons besoin d'opérations telles que la somme, la soustraction (différence), le produit, la division (s'il s'agit d'entiers, on écrira une procédure qui renverra le quotient et le reste) et pourquoi pas d'opérations comme le PGCD, le PPCM...

Pour écrire ces fonctions et procédures, on utilisera uniquement le type TEntier et les primitives Zéro, Successeur et Égal.

En ce qui concerne les entiers, pourquoi procéder ainsi, alors que n'importe quel langage de program-

mation comporte déjà les fonctions usuelles : + - * / sinus logarithme...

Imaginons que l'on veuille manipuler les éléments d'un ensemble qui possède les mêmes propriétés que l'ensemble des entiers naturels : par exemple, les mots d'un dictionnaire. Les mots sont mis dans un certain ordre dans le dictionnaire. L'ensemble des mots susceptibles d'appartenir au dictionnaire est totalement ordonné par une relation d'ordre R. Soit ordre(m) le rang d'un mot m dans le dictionnaire (le mot est à la n-ième place) : $\text{ordre}(m1) < \text{ordre}(m2)$ $m1 R m2$.

Le « plus petit élément » est le mot « a » (le mot de rang 1).

Chaque mot a un successeur dans le dictionnaire. Le successeur d'un mot m est différent de m.

Le mot « a » n'est le successeur d'aucun mot.

On mettra en œuvre la même conception. On se servira du même module de primitives ; il suffira de redéfinir le type des éléments de l'ensemble (par exemple un type de mots ou d'une autre entité mathématique) et de réécrire le corps des primitives : Zéro (plus petit élément), Successeur et Égal.

Puis on concevra un module de fonctionnalités utilisant le module des primitives (dans le cas des entiers, il s'agissait des opérations somme, soustraction...). Les fonctionnalités non primitives sont des fonctionnalités qui utilisent uniquement les primitives et le type des éléments de l'ensemble.

Dans ce sens, leur description algorithmique est indépendante de la description algorithmique des primitives. Nous n'avons pas besoin de savoir comment sont conçues les primitives pour écrire l'algorithme des fonctionnalités non primitives.

Et pourquoi ne pas concevoir un deuxième module de fonctionnalités non primitives qui utilise le premier ?

Par exemple, avec les entiers, on peut créer un module de fonctions calculant le PGCD, le PPCM, le modulo... Ces fonctionnalités se serviront uniquement des fonctionnalités somme, soustraction... du premier module de fonctionnalités non primitives. Ainsi on peut créer des modules comportant des fonctionnalités de plus en plus élaborées.

1.4 Écriture du module de fonctionnalités non primitives sur les entiers

Le module de fonctionnalités non primitives MEntier comprend les fonctions prédécesseur, somme, différence, produit, le prédicat EstAvant (n1 est avant n2) et la procédure division qui retourne le quotient et le reste de la division euclidienne de a (dividende) par b (diviseur).

```

1  unit MEntier ;
2
3  interface
4
5  uses MEntier_Prim ;
6
7  function Entier_Predcesseur ( n : TEntier ) : TEntier ;
8  { le prédécesseur de n différent de Zéro ou }
9  { l'entier dont le successeur est n }
10 { TEntier > TEntier }
```

```

11 { n < > Zero }
12
13
14 function Entier_EstAvant ( n,m : TEntier ) : boolean ;
15 { est égale à vrai si n est avant m, }
16 { est égale à faux si m est avant n }
17 { TEntier x TEntier > booléen }
18
19
20 function Entier_Somme( n,m : TEntier ) : TEntier ;
21 { le m-ième successeur de n }
22 { TEntier x TEntier > TEntier }
23
24
25 function Entier_Difference ( n,m : TEntier ) : TEntier ;
26 { le m-ième prédécesseur de n, }
27 { est égal à Zero si m est après n }
28 { TEntier x TEntier > TEntier }
29
30
31 function Entier_Produit ( n,m : TEntier ) : TEntier ;
32 { le produit de n par m }
33 { TEntier x TEntier > TEntier }
34
35
36 procedure Entier_Division ( a,b : TEntier ; var quotient,reste : TEntier ) ;
37 { retourne le quotient et le reste de la division }
38 { euclidienne de a par b }
39 { TEntier x TEntier > TEntier x TEntier }
40
41
42 implementation
43
44 function Entier_Predcesseur ( n : TEntier ) : TEntier ;
45 var i : TEntier ;
46 begin
47   i := Zero ;
48   while not ( Egal ( Successeur ( i ) , n ) ) do
49     i := Successeur ( i ) ;
50   Entier_Predcesseur := i ;
51 end;
52
53
54 function Entier_EstAvant ( n,m : TEntier ) : boolean ;
55 begin
56   if Egal ( n,m ) then
57     Entier_EstAvant := false
58   else
59     if Egal ( n , Zero ) then
60       Entier_EstAvant := true
61     else if Egal ( m , Zero ) then
62       Entier_EstAvant := false
63     else
64       Entier_EstAvant := Entier_EstAvant ( Entier_Predcesseur ( n ) ,
65         Entier_Predcesseur( m ) ) ;
66   end ;
67
68 function Entier_Somme ( n,m : TEntier ) : TEntier ;
69 begin
70   if Egal ( n , Zero ) then
71     Entier_Somme := m
72   else
73     Entier_Somme := Entier_Somme ( Entier_Predcesseur ( n ) , Successeur ( m )
74     ) ;
75 end;
76
77 function Entier_Difference ( n,m : TEntier ) : TEntier ;
78 var i,d : TEntier;
79 begin
80   if Entier_EstAvant ( n,m ) then
81     Entier_Difference := Zero

```

```

82     else
83     begin
84         i := Zero ;
85         d := n ;
86         while not ( Egal ( i , m ) do
87             begin
88                 d := Entier_Predecesseur ( d ) ;
89                 i := Successeur ( i ) ;
90             end ;
91             Entier_Difference := d ;
92         end ;
93 end;
94
95
96 function Entier_Produit ( n,m : TEntier ) : TEntier ;
97 var i,p : TEntier ;
98 begin
99     i := Zero ;
100    p := Zero ;
101    while not ( Egal ( i , m ) ) do
102        begin
103            p := Entier_Somme ( p , n ) ;
104            i := Successeur ( i ) ;
105        end ;
106        Entier_Produit := p ;
107    end ;
108
109
110 procedure Entier_Division ( a,b : TEntier ; var quotient,reste : TEntier ) ;
111 begin
112     quotient := Zero ;
113     while Entier_EstAvant ( Entier_Produit ( b , quotient ) , a ) do
114         quotient := Successeur ( quotient ) ;
115     quotient := Entier_Predecesseur ( quotient ) ;
116     reste := Entier_Difference ( a , Entier_Produit ( b , quotient ) ) ;
117 end ;
118
119
120 end.

```

1.5 Implémentation et manipulation de l'ensemble des mots

1.5.a Module de primitives

```

1  unit MMot_Prim ;
2
3  interface
4
5  type  TMot : String ;
6
7  function  PremierMot : TMot ;
8
9  function  Successeur ( m : TMot ) : TMot ;
10
11 function  Egal ( m1 , m2 : TMot ) : boolean ;
12
13
14 implementation
15
16 function  PremierMot : TMot ;
17 begin
18     PremierMot := 'a' ;
19 end ;
20
21
22 function  Successeur ( m : TMot ) : TMot ;
23
24     function  Succ_Lettre ( c : string ) : string;
25     begin
26         if c = 'z' then
27             Succ_Lettre := 'a'
28         else

```

```

29     Succ_Lettre := chr ( ord ( c ) + 1 ) ;
30   end ;
31
32   function Rang ( l , i : integer ) : integer ;
33   { l est la longueur du mot }
34   begin
35     Rang := l - i + 1 ;
36   end ;
37
38   function Mot_Jusqua ( m : TMot ; r : integer ) : TMot ;
39   begin
40     Mot_Jusqua := substring ( m , 1 , r ) ;
41   end ;
42
43   var i , r , l : integer ; mt : TMot ;
44   begin
45     i := 1 ;
46     l := length ( m ) ;
47     r := Rang ( l , i ) ;
48     if l = 25 then
49       begin
50         if m[r] = 'z' then
51           begin
52             while m[r] = 'z' do
53               begin
54                 i := i + 1 ;
55                 r := Rang ( l , i ) ;
56               end ;
57             mt := Mot_Jusqua ( m , r ) ;
58             mt[r] := Succ_Lettre ( mt[r] ) ;
59             Successeur := mt ;
60           end ;
61         else
62           begin
63             m[r] := Succ_Lettre ( m[r] ) ;
64             Successeur := m ;
65           end ;
66         end ;
67       else
68         Successeur := m + 'a' ;
69     end ;
70
71   function Egal ( m1 , m2 : TMot ) : boolean ;
72   begin
73     Egal := ( m1 = m2 ) ;
74   end ;
75
76
77 end.

```

Commentaire : le plus petit élément est le mot « a », pour la fonction successeur : le plus long mot est « anticonstitutionnellement », qui a 25 lettres ; on conviendra que les mots ont au plus 25 lettres ; si le mot se termine par ' zzz ... ' : exemple : si m = ' abzzz...z ' (mot de 25 lettres), le successeur est ' ac ' sinon si m = ' abc ' le successeur est ' abca '. Pour la fonction Égal : l'opérateur ' = ' du Pascal permet de déterminer si deux mots sont égaux.

1.5.b Module de fonctionnalités non primitives

L'ensemble des entiers naturels et l'ensemble des mots ont les mêmes propriétés : il existe un « plus petit mot » qui n'est le successeur d'aucun mot et chaque mot m a un successeur différent de m.

On peut donc effectuer des opérations sur les mots avec les fonctionnalités non primitives que nous avons conçues pour l'ensemble des entiers. Nous pouvons nous servir du module MEntier (fonctionnalités non primitives des entiers), car ces fonctionnalités garantissent leur indépendance vis-à-vis de la mise en œuvre des primitives.

```

1  unit MMot ;
2
3  interface
4
5  uses MMot_Prim ;
6
7  function Mot_Predcesseur ( n : TMot ) : TMot ;
8  { le prédécesseur de n différent de PremierMot ou }

```

```

9 { le mot dont le successeur est n }
10 { TMot > TMot }
11 { n < > PremierMot }
12
13
14 function Mot_EstAvant ( n,m : TMot ) : boolean ;
15 { est égale à vrai si n est avant m, }
16 { est égale à faux si m est avant n }
17 { TMot x TMot > booléen }
18
19
20 function Mot_Somme( n,m : TMot ) : TMot ;
21 { le m-ième successeur de n }
22 { TMot x TMot > TMot }
23
24
25 function Mot_Difference ( n,m : TMot ) : TMot ;
26 { le m-ième prédécesseur de n , }
27 { est égal à PremierMot si m est après n }
28 { TMot x TMot > TMot }
29
30
31 function Mot_Produit ( n,m : TMot ) : TMot ;
32 { le produit de n par m }
33 { TMot x TMot > TMot }
34
35
36 procedure Mot_Division ( a,b : TMot ; var quotient,reste : TMot ) ;
37 { retourne le quotient et le reste de la division }
38 { euclidienne de a par b }
39 { TMot x TMot > TMot x TMot }
40
41
42 implementation
43
44 function Mot_Predcesseur ( n : TMot ) : TMot ;var i : TMot ;
45 begin
46     i := PremierMot ;
47     while not ( Egal ( Successeur ( i ) , n ) ) do
48         i := Successeur ( i ) ;
49     Mot_Predcesseur := i ;
50 end;
51
52
53 function Mot_EstAvant ( n,m : TMot ) : boolean ;
54 begin
55     if Egal ( n,m ) then
56         Mot_EstAvant := false
57     else
58         if Egal ( n , PremierMot ) then
59             Mot_EstAvant := true
60         else
61             if Egal ( m , PremierMot ) then
62                 Mot_EstAvant := false
63             else
64                 Mot_EstAvant := Mot_EstAvant ( Mot_Predcesseur ( n ) ,
65                     Mot_Predcesseur( m ) ) ;
66
67 end ;
68
69
70 function Mot_Somme ( n,m : TMot ) : TMot ;
71 begin
72     if Egal ( n , PremierMot ) then
73         Mot_Somme := m
74     else
75         Mot_Somme := Mot_Somme ( Mot_Predcesseur ( n ) , Successeur ( m ) ) ;
76 end;
77
78
79 function Mot_Difference ( n,m : TMot ) : TMot ;
80 var i,d : Tmot;
81 begin
82     if Mot_EstAvant ( n,m ) then

```

```

81     Mot_Difference := PremierMot
82   else
83     begin
84       i := PremierMot ;
85       d := n ;
86       while not ( Egal ( i , m ) do
87         begin
88           d := Mot_Predecesseur ( d ) ;
89           i := Successeur ( i ) ;
90         end ;
91       Mot_Difference := d ;
92     end ;
93 end;
94
95
96 function Mot_Produit ( n,m : TMot ) : TMot ;
97 var i,p : TMot ;
98 begin
99   i := PremierMot ;
100  p := PremierMot ;
101  while not ( Egal ( i , m ) ) do
102    begin
103      p := Mot_Somme ( p , n ) ;
104      i := Successeur ( i ) ;
105    end ;
106  Mot_Produit := p ;
107 end ;
108
109
110 procedure Mot_Division ( a, b : TMot ; var quotient,reste : TMot ) ;
111 begin
112   quotient := PremierMot ;
113   while Mot_EstAvant ( Mot_Produit ( b , quotient ) , a ) do
114     quotient := Successeur ( quotient ) ;
115     quotient := Mot_Predecesseur ( quotient ) ;
116     reste := Mot_Difference ( a , Mot_Produit ( b , quotient ) ) ;
117 end ;
118
119
120 end.

```

Commentaire : la structure algébrique des mots munie des lois « somme, différence, produit, division » est isomorphe à la structure des entiers naturels munie des mêmes lois, car l'ensemble des entiers naturels et l'ensemble des mots ont les mêmes propriétés : « plus petit élément » et « successeur ». Ainsi on peut effectuer une division euclidienne avec les mots, qui retournera les deux mots quotient et reste.

Les primitives « Zéro » et « Successeur » servent à construire une structure d'entiers, c'est-à-dire une structure dont les éléments sont totalement ordonnés par une relation d'ordre. Dès que l'on connaîtra

un ensemble ayant la structure d'entiers, on pourra lui associer des fonctionnalités élaborées conçues uniquement à partir du type des éléments de l'ensemble et des primitives « Zéro », « Successeur » et « Égal ». Ces calculs seront fondés sur la structure construite par les primitives, en l'occurrence une relation d'ordre.

Calculer un produit ou une division de mots n'a pas beaucoup de sens, mais nous pouvons par exemple implémenter un type de nombres écrits en base 2, 5 ou hexadécimal, les trois primitives et manipuler cet ensemble avec les fonctionnalités élaborées.

1.6 Implémentation et manipulation de l'ensemble des entiers écrits en base n

1.6.a Module de primitives

```

1  unit MEntier_Base_n_Prim ;
2
3  interface
4
5  type    TEntier_Base_n = record
6          base_n : integer ;
7          entier : string ;
8      end ;
9
10
11 function Zero ( base : integer ) : TEntier_Base_n ;
12
13 function Successeur ( x : TEntier_Base_n ) : TEntier_Base_n ;
14
15 function Egal ( x1 , x2 : TEntier_Base_n ) : boolean ;
16
17
18 implementation
19
20 function Zero ( base : integer ) : TEntier_Base_n ;
21 var d : TEntier_Base_n ;
22 begin
23     d . base_n := base ;
24     d . entier := ' 0 ' ;
25     Zero := d ;
26 end ;
27
28
29 function Successeur ( x : TEntier_Base_n ) : TEntier_Base_n ;
30 var r , base : integer ; fini : boolean ; d : TEntier_Base_n ;
31 begin
32     r := length ( x . entier ) ;
33     base := x . base_n - 1 ;
34     fini := false ;
35     while not fini do
36         begin
37             if x . entier [ r ] = base then
38                 if r = 1 then
39                     begin
40                         x . entier [ r ] := ' 0 ' ;
41                         x . entier := '1' + x . entier ;
42                         fini := true ;
43                     end
44                 else
45                     begin
46                         x . entier [ r ] := ' 0 ' ;
47                         r := r - 1 ;
48                         fini := false ;
49                     end
50                 else
51                     begin
52                         x . entier [ r ] := chr ( ord ( x . entier [ r ] ) + 1 ) ;
53                         fini := true ;
54                     end ;
55                 end ;
56             d . base_n := x . base_n ;
57             d . entier := x . entier ;
58             Successeur := d ;
59         end ;
60
61
62 function Egal ( x1 , x2 : TEntier_Base_n ) : boolean ;
63 begin
64     Egal := ( ( x1 . base_n = x2 . base_n ) and ( x1 . entier = x2 . entier ) ) ;
65 end ;

```

Après avoir implémenté un ensemble d'entiers représentés en base n, nous pouvons manipuler les éléments de cet ensemble avec le même module de fonctionnalités élaborées (opérations d'arithmétique).

```

1  _Predecesseur
2  _EstAvant
3  _Somme
4  _Difference
5  _Produit
6  _Division
  
```

1.6.b Fonctionnalités élaborées

Ajoutons maintenant de nouvelles fonctions à ce module; voyons comment les primitives permettent la conversion des bases, en d'autres termes la conversion d'un entier représenté en base n en un entier représenté en base m.

```

1  function Entier_Base_n_Converti ( x :
    TEntier_Base_n ; base : integer ) :
    TEntier_Base_n ;
2  { entier x converti en une nouvelle
    base }
3  { TEntier_Base_n X integer >
    TEntier_Base_n }
4
5  var   base_de_x : integer ;
6       i , x_converti : TEntier_Base_n ;
7
8  begin
9     base_de_x := x . base_n ;
10    i := Zero ( base_de_x ) ;
11    x_converti := Zero ( base ) ;
12    while not ( Egal ( i , x ) ) do
13      begin
14        x_converti := Successeur (
          x_converti ) ;
15        i := Successeur ( i ) ;
16      end ;
17    Entier_Base_n_Converti :=
          x_converti ;
18  end ;
  
```

Là aussi, cette fonction utilise uniquement le type TEntier_Base_n et les trois primitives.

Nous allons maintenant créer les fonctions Modulo, PGCD, PPCM.

```

1  function Entier_Modulo ( a , m : TEntier
    ) : TEntier ;
2  { a modulo m ou bien : le reste de la
    division euclidienne de a par m }
3  { TEntier X TEntier > TEntier }
4  { m est inférieur ou égal à a }
5
6  { L'algorithmme est évident : nous avons
    déjà écrit la procédure de division
    euclidienne }
7
8  function Entier_Modulo ( a , m : TEntier
    ) : TEntier ;
  
```

```

10 var quotient , reste : TEntier ;
11 begin
12   Entier_Division ( a , m , quotient ,
    reste ) ;
13   Entier_Modulo := reste ;
14 end ;
15
16
17 function Entier_PGCD ( a , b : TEntier )
    : TEntier ;
18 { le PGCD ( plus grand commun diviseur )
    de a et b }
19 { TEntier X TEntier > TEntier }
  
```

Nous allons nous servir de la fonction Entier_Modulo : a est divisible par b si Entier_Modulo (a, b) = 0.

Cherchons lequel de a et b est le plus grand. Si a est avant b, par exemple, il suffit de chercher le plus grand entier inférieur ou égal à a qui soit un diviseur de a et de b.

```

1  var d : TEntier ;
2  begin
3     if Entier_EstAvant ( a , b ) then
4       d := a
5     else
6       d := b ;
7     while not ( ( Entier_Modulo ( a , d
    ) = 0 ) and ( Entier_Modulo ( b
    , d ) = 0 ) ) do
8       d := Entier_Predecesseur ( d )
9     ;
9     Entier_PGCD := d ;
10 end ;
  
```



Remarque : le PGCD de a et b est 1 si a et b sont premiers entre eux.

```

1  function Entier_PPCM ( a , b : TEntier )
    : TEntier ;
2  { le PPCM ( plus petit commun multiple )
    de a et b }
3  { TEntier X TEntier > TEntier }
  
```

Cherchons lequel de a et b est le plus grand. Si a est avant b, par exemple, il suffit de chercher le plus petit entier supérieur ou égal à b qui soit un multiple de a et de b.

```

1  var d : TEntier ;
2  begin
3     if Entier_EstAvant ( a , b ) then
4       d := b
5     else
6       d := a ;
7     while not ( ( Entier_Modulo ( a , d
    ) = 0 ) and ( Entier_Modulo ( b
    , d ) = 0 ) ) do
8       d := Successeur ( d ) ;
9     Entier_PPCM := d ;
10 end ;
  
```

2 Listes récursives

2.1 Définition

Une liste récursive est :

- soit une liste vide notée $\langle \rangle$;
- soit un doublet formé d'un objet et d'une liste récursive, qu'on notera $\langle e \text{ lr} \rangle$.

Dans ce dernier cas, l'objet e est appelé le premier élément de la liste, et la liste lr la liste restante, ou le **reste**, ou la suite de la liste.

On simplifiera les notations en remplaçant $\langle e1 \langle e2 \langle e3 \langle \rangle \rangle \rangle \rangle$ par $\langle e1 \text{ e2 e3} \rangle$.

Le reste de la liste $\langle e1 \text{ e2 e3} \rangle$ est donc la liste $\langle e2 \text{ e3} \rangle$, abréviation de $\langle e2 \langle e3 \langle \rangle \rangle$.

On conviendra que les éléments de la liste sont des valeurs du type TData quelconque de données, et on notera TListeRec le type des listes récursives de données.

2.2 Primitives

Pour manipuler des listes, on a besoin de deux **constructeurs** : un pour la liste vide, et un pour un doublet :

```

1 function ListeRec_Vide : TListeRec
2   { la liste vide (de données) < > }
3   { => TListeRec : => < > }
4
5 function ListeRec_Construire(e : TData;
6   lr : TListeRec) : TListeRec
7   { la liste non vide - le doublet <e
8     lr > - dont le premier élément
9     est e et le reste lr }
10  { TData X TListeRec => TListeRec : e
11    , lr => < e lr > }

```

On a également besoin d'un **prédicat** qui permet de reconnaître une liste vide :

```

1 function ListeRec_EstVide(l : TListeRec)
2   : boolean
3   { l est la liste vide }
4   { TListeRec => boolean }

```

Il faut également disposer de « **sélecteurs** » permettant de déterminer les deux composants d'une liste non vide :

```

1 function ListeRec_Premier(l : TListeRec) :
2   TData
3   { le premier élément d'une liste non
4     vide l }
5   { TListeRec => TData : < e lr > => e }
6   { domaine : l est non vide }
7
8 function ListeRec_Reste(l : TListeRec) :
9   TListeRec
10  { le reste d'une liste non vide l }
11  { TListeRec => TListeRec : < e lr >
12    => lr }
13  { domaine : l est non vide }

```

Enfin, il faut disposer de **transformations** qui permettent de modifier une liste :

```

1 procedure ListeRec_PremierA(d : TData ;
2   var l : TListeRec) ;
3   { affecter d comme premier élément d
4     'une liste non vide l }
5   { TData X TListeRec => TListeRec : d
6     , < e lr > => < d lr > }
7   { domaine : l est non vide }
8
9 Procedure ListeRec_ResteA(lr : TListeRec
10  ; var l : TListeRec) ;
11  { affecter lr comme reste d'une
12    liste non vide l }
13  { TListeRec X TListeRec => TListeRec
14    : lr , < e r > => < e lr > }
15  { domaine : l est non vide }

```

2.3 Implémentation par des pointeurs

L'implémentation « naturelle » serait de définir le type TListeRec comme une réunion :

```

1 TListeRec = record case vide : boolean
2   of
3     true : ( ) ;
4     false : ( element :
5       TData; reste :
6       TListeRec ) ;
7   end;

```

Malheureusement cette définition n'est pas « calculable » : en effet, le calcul de la taille t du type TListeRec donnerait l'équation suivante : $t = 1 + \text{tailleDe TData} + t$, qui n'admet pas de solution.

Le moyen de résoudre cette difficulté est d'identifier une liste à un pointeur sur (une référence à) un doublet.

Dans ces conditions, une liste vide serait identifiée à l'adresse Nil, et la taille de TListeRec devient calculable.

Autrement dit : une liste vide est identifiée à Nil et une liste non vide à l'adresse du doublet qu'elle définit.

Soit TPtrDoublet=TListeRec le type pointeur sur TDoublet, où TDoublet est le type :

```

1 Record premier : TData ; reste :
2   TListeRec end

```

Dans ces conditions, la taille du type TListeRec est égale à la taille d'un pointeur (en général quatre octets), et la taille d'un doublet est égale à TailleDe TData + TailleDe TListeRec

```

1 Type TPtrDoublet = ^ TDoublet ;
2   TListeRec = TPtrDoublet;
3   TDoublet = Record
4     premier : TData ;
5     reste : TListeRec ;
6   end ;
7
8 function ListeRec_Vide : TListeRec ;
9 begin
10  ListeRec_Vide := nil
11 end ;
12
13

```

```

14 function ListeRec_EstVide( l : TListeRec
    ) : boolean ;
15 begin
16   ListeRec_EstVide := ( l = nil ) ;
17 end;
18
19
20 function ListeRec_Construire ( elt :
    TData; lr : TListeRec ) : TListeRec;
21 var   PtrDoublet : TPtrDoublet;
22 begin
23   new(PtrDoublet);
24   PtrDoublet^.premier := elt ;
25   PtrDoublet^.reste := lr ;
26   ListeRec_Construire := PtrDoublet ;
27 end;
28
29
30 function ListeRec_Premier ( l :
    TListeRec ) : TData;
31 begin
32   ListeRec_Premier := l^.premier;
33 end;
34
35
36 function ListeRec_Reste ( l : TListeRec
    ) : TListeRec;
37 begin
38   ListeRec_Reste := l^.reste;
39 end;
40
41
42 procedure ListeRec_PremierA ( d : TData
    ; var l : TListeRec ) ;
43 begin
44   l^.premier := d ;
45 end;
46
47
48 procedure ListeRec_ResteA ( lr :
    TListeRec; var l : TListeRec ) ;
49 begin
50   l^.reste := lr;
51 end;

```

2.4 Fonctionnalités non primitives

Les fonctionnalités non primitives sont des fonctionnalités qui utilisent uniquement les primitives et le type TListeRec. Dans ce sens, leur description algorithmique est indépendante de la description algorithmique des primitives.

Avec les listes récursives, on cherche toujours à trouver une **description récursive** de l'algorithme d'une procédure Proc non primitive, en terme de liste vide, de premier élément, de reste et de la procédure Proc elle-même. En général, c'est sur le reste de la liste que la procédure Proc sera appelée dans l'algorithme.

Voyons deux exemples simples avant de construire les fonctionnalités d'ajout et de suppression :

```

1 function ListeRec_LeNeme ( n : integer
    ; l : TListeRec ) : TData;
2   { le nème élément d'une liste l }
3   { Integer X TListeRec => TData }
4   { domaine : 1<=n<=longueur de l }
5 begin

```

```

6   if n=1 then
7     ListeRec_LeNeme :=
      ListeRec_Premier(l)
8   else
9     ListeRec_LeNeme :=
      ListeRec_LeNeme ( n-1,
      ListeRec_Reste(l))
10 end;

```

Commentaire : l'algorithme est clair : ou bien $n = 1$ (et par hypothèse l contient au moins un élément) et c'est le premier élément, ou bien $n > 1$ (et \leq à longueur de l) et alors le n -ième élément de l est le $(n-1)$ ième du reste.

```

1 function ListeRec_Longueur(l : TListeRec
    ) : Integer ;
2   { la longueur de l }
3   { TListeRec => Integer }
4
5 function longueurIter( n :Integer ; l :
    TListeRec):Integer;
6 { ajouter à n la longueur de l }
7 begin
8   if ListeRec_EstVide(l) then
9     longueurIter :=n
10  else
11    longueurIter := longueurIter
      (n+1, ListeRec_Reste(l))
12  end ;
13 begin
14   ListeRec_Longueur := longueurIter(0,
      l)
15 end ;

```

Commentaire : on aurait pu écrire l'algorithme plus simplement : si l est vide c'est 0, sinon c'est 1 + la longueur du reste. Avec l'algorithme décrit dans l'exemple, la fonction ListeRec_Longueur n'est pas elle-même récursive. Elle fait appel à une fonction récursive et de ce fait l'est indirectement. Cependant la fonction longueurIter est « terminalement » récursive, et donc décrit un processus itératif.

```

1 procedure ListeRec_InsererAvant(d :TData
    ; n :Integer ; var l : TListeRec )
2   ;
3   { inserer d avant le nème élément de
      l }
4   { TData X Integer X TListeRec =>
      TListeRec }
5   { 1<=n<=longueur(l)+1 }
6 var lt: TListeRec ;
7 begin
8   if ListeRec_EstVide(l) or (n=1)
9   then l :=ListeRec_Construire(d,l)
10  else begin
11    lt :=ListeRec_Reste(l) ;
12    ListeRec_InsererAvant(d, n-1
      , lt) ;
13    ListeRec_ResteA(lt,l)
14  end ;
15 end ;

```

Commentaire : si la liste est vide ou s'il faut insérer en tête, l est un doublet dont l'élément est d et le reste l .

Sinon on insère d avant le $(n-1)$ ième élément du reste de l .

On utilise lt puisque ListeRec_InsererAvant(d , $n-1$, ListeRec_Reste(l)) n'est syntaxiquement pas

correct.

```

1  procedure ListeRec_SuppPrem ( d :TData
   ; var l : TListeRec ) ;
2  { supprimer de l la première
   occurrence de d }
3  { TData X TListeRec => TListeRec }
4  var lt : TListeRec ;
5  begin
6  if not ListeRec_EstVide( l ) then
7  begin
8  if Data_Identique ( d,
   ListeRec_Premier ( l ))
   then
9  begin
10  lt := l;
11  l := ListeRec_Reste ( l
   );
12  dispose ( lt) ;
13  end
14  else
15  begin
16  lt := ListeRec_Reste (
   l );
17  ListeRec_SuppPrem( d, lt
   );
18  ListeRec_ResteA( lt , l
   );
19  end;
20  end;
21 end;

```

Commentaire : le prédicat `Data_Identique` est une fonctionnalité du type `TData` qui permet de comparer deux données.

On remarque l'utilisation d'une liste « de travail » pour dans un cas permettre de rendre le premier Doublet au manager de la mémoire (avec la procédure `Dispose`), et dans le deuxième cas de pouvoir appeler récursivement la procédure en passant, comme il faut, un identificateur comme paramètre : l'instruction `ListeRec_SuppPrem(d, ListeRec_Reste(l))` n'est en effet pas valide.

2.5 Chapitres I et II : Conclusion

Le terme **structure de données** désigne une composition de données unies par une même sémantique.

Nous avons implémenté une structure de données en définissant le type des données.

Dans le cas des entiers, il s'agissait d'un type simple (`integer`) ; les listes ont été implémentées par des pointeurs de manière récursive : « une liste est un doublet formé d'un objet et d'une liste ».

Puis, nous avons spécifié et créé des primitives pour opérer sur la structure de données.

3 Preuve de programme

3.1 Notation

Si `T` est un type, `ValeurDe (T)` est l'ensemble de toutes les données de type `T` implémentables par

La mise en œuvre des primitives se fait conformément aux propriétés de la structure de données.

Enfin nous avons conçu un module de fonctionnalités élaborées qui utilisent uniquement les primitives, pour effectuer la résolution de problèmes d'arithmétique en ce qui concerne les entiers ; pour les listes, des opérations de calcul de la longueur d'une liste, de recherche d'un élément, d'ajout et de suppression d'un élément...

Les fonctionnalités élaborées doivent garantir leur indépendance vis-à-vis de la mise en œuvre des primitives. Leur description algorithmique est indépendante de la description algorithmique des primitives.

Supposons que deux structures `A` et `B` aient strictement les mêmes propriétés (on peut les définir par les mêmes axiomes). On peut concevoir pour l'une et l'autre exactement les mêmes modules de fonctionnalités élaborées (non primitives), après avoir implémenté `A` et `B` et écrit le programme complet pour la structure `A`, afin d'écrire le programme travaillant sur la structure `B`, il suffira de concevoir convenablement l'algorithme des primitives qui opèrent sur la structure `B`. C'est ce que nous avons démontré en concevant le programme qui traite l'ensemble des mots (isomorphe à l'ensemble des entiers).

En procédant comme nous l'avons fait, on passe par différentes étapes, du langage informatique au langage humain (le français).

Utilisateur	Français, spécification du problème
Cogniticien	Mathématiques, Logique
Logicien	Domaine de l'intelligence artificielle
Informaticien	Mise en équation du problème Conception des structures de données
Informaticien	Algorithmique : description ordonnée du procédé de résolution
Informaticien	Programmation : codage de l'algorithme dans un langage de programmation
Ordinateur	Programme codé dans le langage de l'ordinateur (langage machine) et chargé en mémoire

la machine utilisée. On confondra souvent, dans la suite, `T` avec `ValeurDe (T)`.

3.2 Procédure

Une procédure P sera représentée par le quadruplet (S, D, T, A) où :

- S la spécification (voir plus loin) ;
- D est le domaine, c'est-à-dire l'ensemble des valeurs du domaine de la signature T, pour lesquelles la procédure est applicable. Le domaine de P est toujours inclus dans le domaine de T, mais pas toujours identique ;
- T (comme type) la signature, et A l'algorithme.

On conviendra que l'identificateur de la procédure est également P.

On note que D, S et T font partie de l'interface (déclaration publique) alors que A fait partie de l'implémentation (à partir de fonctionnalités « plus primitives »).

3.3 Spécification

La spécification S est un énoncé paramétré par D : elle contient un paramètre d dont les valeurs sont les éléments de D.

On conviendra que :

- si P est un prédicat, S est une proposition ;
- si P est une fonction, S spécifie un objet ;
- si P est une commande (par définition, l'application de P modifie l'état de la machine ou provoque un « effet de bord » (affichage à l'écran, impression, exportation de données...), ou bien les deux) S décrit sans ambiguïté une modification de l'état de la machine, et/ou un effet de bord.

C'est le point délicat de la preuve de programme : « sans ambiguïté une modification de l'état de la machine, et/ou un effet de bord » n'a malheureusement pas toujours un sens très précis, dans le modèle des machines que nous utilisons implicitement.

3.4 Algorithme

Appliquer une procédure P à un élément de son domaine D consiste à appliquer la suite des instructions de A à d ; on note A(d) ou quelquefois, par abus de langage, P(d), le résultat de l'application de A à d :

- si P est un prédicat, A(d) est un booléen ;
- si P est une fonction, A(d) est un élément du codomaine de T ;
- si P est une commande, A(d) est une modification ou un effet de bord.

3.5 Exemples

Aff = (« Afficher à l'écran tous les éléments d'une liste l »,

TListe ,
 l : TListe => ,

« si l est vide
 ne rien faire
 sinonChoseAfficher (ListePremier (l))
 Aff (ListeReste (l)) »)

où ChoseAfficher = (« Afficher à l'écran la chose ch »,

ch : TChose ,
 TChose => ,
 ...)

ListePremier = (« Le premier élément de la liste non vide l »,

l : l est une liste de TListe non vide ,
 TListe => TChose ,
 ...)

ListeReste = (« Le reste de la liste non vide l » ;
 l : l est une liste de TListe non vide ,
 TListe => TListe ,
 ...)

Rang = (« Le rang de la première occurrence de la chose ch dans la liste l, ou -1 si ch n'est pas dans la liste »,

l , ch : TListe x TChose
 TListe x TChose => Entier ,

« au cas ou

: (l est vide) retourner -1

: (ChoseEgale (ch , ListePremier (l))) retourner 1

sinon n <= Rang (ListeReste (l) , ch)

si n = -1 retourner -1

sinon retourner n+1

fin de cas »)

3.6 Égalité de procédures

C'est l'égalité des quadruplets : P = (S, D, T, A) est égale à P' = (S', D', T', A') - on note P = P' - ssi S = S' , D = D' , T = T' et A = A'.

3.7 Équivalence des procédures

C'est « l'équivalence des algorithmes » : P = (S, D, T, A) est équivalent à P' = (S', D', T', A') - on note P † P' - ssi S † S' , D † D' , T † T' et l'application de P à tout élément d de D produit le même résultat que celle de P' à d.

Encore une fois, si P est une commande, l'expression « produit le même résultat que » n'a pas toujours un sens très précis.

3.8 Calculabilité d'un algorithme

Un algorithme est calculable si chacune de ses procédures est appliquée à un argument de son domaine.

On pourrait imposer une condition supplémentaire : que l'exécution de l'algorithme se termine après un nombre fini d'instructions.

3.9 Prouver une procédure

Définition : la procédure P est prouvée si :

- (p) toutes les procédures de A, autres que P sont prouvées (P est prouvable);
- (c) pour tout d de D, A est calculable (P est calculable);
- (s) pour tout d de D, P(d) est décrit par S(d) (P est bien spécifié) :
 - si P est un prédicat, S(d) est vrai si et seulement si P(d) est vrai;
 - si P est une fonction, l'objet P(d) est spécifié par S(d);
 - si P est une commande à effet de bord, S(d) doit décrire l'effet de bord obtenu par application de P à d;
 - si P est une commande qui modifie l'état du système, S(d) doit décrire la modification apportée au système après application de P à d.

Dire que « P est bien spécifié » signifie que la spécification est conforme à l'algorithme.

Prouver une procédure consiste à démontrer que P est prouvé.

Il n'y a pas de difficultés particulières à montrer qu'un algorithme est prouvable quand les procédures qui le composent sont elles-mêmes prouvables.

La preuve qu'une procédure est bien spécifiée est généralement beaucoup plus difficile, en particulier quand il s'agit de commandes.

Cependant, lorsque la procédure est rédigée récursivement, on dispose d'un outil qui généralement fait bien l'affaire : les techniques de preuve par récurrence.

La récursion est en effet implicitement associée à une partition de D : $D : D_i, D_{i+1}, \dots$

Si on note P_n la restriction de P à $D_i U \dots U D_n$ ($P_n = (S, D_i U \dots U D_n, T, A)$), montrer que P est bien spécifié pour tout d de D est équivalent à montrer que P_n est bien spécifiée pour tout $n \geq i$.

Ce qu'on peut faire par récurrence :

1. (Valeur initiale) On vérifie que P_i est bien spécifiée;
2. (Induction) On démontre que si P_k est bien spécifiée pour $i \leq k \leq n$ alors P_{n+1} est bien spécifiée.

3.9.a Exemple 1 : montrer que Aff est bien spécifiée

(h) Nous supposons (p) et (c)

$D = TListe$. On partitionne D selon la relation d'équivalence $L_g : L_g (l, l')$ si l et l' ont même longueur. D_k est l'ensemble des listes de TListe de longueur k; $D_0, D_1, \dots, D_k, \dots$

forment une partition de D.

Par définition, Aff_n est la restriction de Aff aux listes de longueur $\leq n$.

1. Vérifions que Aff_0 est bien spécifiée; il faut montrer que si on applique A à une liste l de longueur 0, le résultat est bien décrit par S(l) : S(l) est l'énoncé « Afficher à l'écran tous les éléments d'une liste vide l » Appliquons A à l : comme la liste est vide aucune instruction n'est exécutée : A(l) ne fait rien. Si on considère que « Afficher à l'écran tous les éléments d'une liste vide l » (S(l)) c'est ne rien faire (A(l)), Aff_0 est bien spécifiée.
2. En faisant l'hypothèse que Aff_k est bien spécifiée revient à montrer que pour toute liste l de longueur n+1, A(l) est décrit par S(l).

S(l) est l'énoncé « Afficher tous les éléments de la liste l »

Appliquons A à l : puisque l n'est pas vide, ChoseAfficher(ListePremier(l)) est d'abord exécutée. Comme ChoseAfficher et ListePremier sont bien spécifiées (car prouvées par hypothèse (h))

ChoseAfficher(ListePremier(l)) est décrit par l'énoncé « Afficher à l'écran le premier élément de l ».

Puis Aff(ListeReste(l)) est exécutée. Comme ListeReste(l) est une liste de longueur n (ListeReste est prouvée), Aff(ListeReste(l)) est équivalent à $Aff_n(ListeReste(l))$.

Comme par hypothèse de récurrence Aff_n est bien spécifiée, Aff(ListeReste(l)) est décrit par l'énoncé « Afficher à l'écran le premier élément de l, puis afficher à l'écran tous les éléments du reste de la liste ».

A(l) est donc finalement décrit par l'énoncé « Afficher à l'écran le premier de l, puis afficher à l'écran tous les éléments du reste de la liste », qui est une forme équivalente de S(l).

3.9.b Exemple 2 : montrer que Rang est bien spécifiée

(h) : Nous supposons (p) et (c)

On partitionne TListe de la même façon : D_k est maintenant $L_k \times TChose$, où L_k est l'ensemble des listes de TListe de longueur k.

1. Vérifions que $Rang_0$ est bien spécifiée : S(l, ch) est l'objet -1. Appliquons A à (l, ch) : comme la liste est vide A retourne -1 : C.Q.F.D.
2. Montrons $Rang_{n+1}$ Pour une liste de longueur n+1 S(l, ch) est l'énoncé « Le rang de la première occurrence de la chose ch dans la liste l,

ou -1 si ch n'est pas dans la liste ». Appliquons A à (l , ch). De trois choses l'une :

- (a) Ou bien le premier élément de l est égal à ch ;
- (b) Ou bien ch n'est pas dans l ;
- (c) Ou bien ch est dans le reste de l.

Dans le premier cas, A retourne 1 : c'est bien « le rang de la première occurrence de la chose ch dans la liste l ».

Dans le second cas (ch n'est pas dans l) n est affectée de la valeur $\text{Rang}(\text{ListeReste}(l), ch)$. Comme ListeReste est bien spécifiée (hypothèse (h)) c'est le reste de la liste l, donc une liste de longueur n ; $\text{Rang}(\text{ListeReste}(l), ch)$ étant alors équivalent à

$\text{Rang}_n(\text{ListeReste}(l), ch)$ qui est prouvée par hypothèse de récurrence , n est affectée de la valeur -1 (ch n'est pas dans l!). A retourne -1 , ce qui est conforme à la spécification.

Dans le troisième cas (ch est dans le reste de l), on peut affirmer, par un raisonnement analogue, que n est affectée du rang de ch dans le reste de l ; A retourne ce rang plus 1 qui est effectivement « le rang de la première occurrence de la chose ch dans la liste l » : en effet, si ch a le rang r dans le reste de l, il a le rang r+1 dans l.



Remarque : si $P = (S , D , T , A)$ est prouvé ainsi que $P' = (S , D , T , A')$, alors $P \approx P'$.

Retrouvez l'article de **Philippe Quet** en ligne : [lien 1](#)



Delphi

Les derniers tutoriels et articles

Expressions régulières avec Delphi

Utilisation des expressions régulières avec l'unité RegularExpressions

En lisant cet article vous apprendrez à vous servir des expressions régulières avec l'unité RegularExpressions de Delphi, unité apparue avec la version XE. Nous commencerons par quelques rappels généraux sur les expressions régulières. À la fin de chaque partie, des exercices corrigés vous seront proposés.

1 Introduction

On appelle expression régulière une chaîne de caractères représentant un ensemble de chaînes de caractères.

Ainsi la chaîne 'abc', considérée comme expression régulière, représente un ensemble contenant un seul élément, la chaîne 'abc'. Chacun des trois caractères se représente lui-même.

1.1 Classes de caractères

Plus intéressante, l'expression régulière '[abc]' représente l'ensemble des chaînes constituées d'un seul caractère appartenant à l'ensemble ['a', 'b', 'c']. Ajoutons que les trois caractères 'a', 'b' et 'c' étant consécutifs dans l'alphabet, notre expression régulière aurait pu s'écrire '[a-c]', de même qu'en Pascal, on pourrait écrire ['a'..'c'] au lieu de ['a', 'b', 'c'].

À la différence des lettres de l'alphabet qui se représentent simplement elles-mêmes, les crochets et le trait d'union sont donc des caractères affectés d'une signification spéciale, que nous venons de voir. Pour que ces caractères spéciaux (qu'on appelle aussi métacaractères) soient traités comme des caractères ordinaires, c'est-à-dire interprétés littéralement, il faut les « échapper », comme dit l'anglais, en les faisant précéder d'une barre oblique inversée :

'\'

',''. La barre oblique inversée est donc elle aussi un métacaractère.

Un autre caractère spécial dont il faut parler à cet endroit, c'est l'accent circonflexe. Lorsqu'il est placé immédiatement après un crochet ouvrant, il signifie la négation ou l'exclusion. Par exemple la classe '[^0-9]' contient tous les caractères qui ne sont pas des chiffres.

1.2 Classes prédéfinies

La barre oblique inversée sert aussi à donner une signification spéciale à des caractères qui par défaut sont interprétés littéralement.

Par exemple la lettre 'd', précédée d'une barre oblique inversée, représente la classe des chiffres (digits en anglais). Les trois expressions suivantes sont donc synonymes : '[0123456789]', '[0-9]', ''.

Un utilisateur averti des expressions régulières en Perl me faisait remarquer que dans les versions récentes de Perl, ces trois expressions ne sont plus synonymes, car la classe '' comprend désormais les chiffres de tous les alphabets, et non pas seulement ceux de l'alphabet latin.

Voici un tableau contenant toutes les classes prédéfinies.

Notation	Sens	Notation équivalente
<code>\a</code>	Cloche	<code>\x07</code>
<code>\d</code>	Chiffre	<code>[0-9]</code>
<code>\e</code>	Échappement	<code>\x1B</code>
<code>\f</code>	Nouvelle feuille	<code>\x0C</code>
<code>\n</code>	Nouvelle ligne	<code>\0A</code>
<code>\r</code>	Retour chariot	<code>\0D</code>
<code>\s</code>	Espace (au sens large)	<code>[\t]</code>
<code>\t</code>	Tabulation	<code>\x09</code>
<code>\v</code>	Tabulation verticale	<code>\x0B</code>
<code>\w</code>	Caractère alphanumérique (y compris <code>'_'</code>)	<code>[A-Za-z0-9_]</code>
<code>.</code>	Tous les caractères (sauf <code>#13</code> et <code>#10</code>)	<code>[\r\n]</code>

Notons que si nous remplaçons `'a'`, `'d'`, `'e'` ou l'une des autres lettres par une majuscule, l'expression désigne alors l'ensemble des caractères *n'étant pas* la cloche, un chiffre, etc. Ainsi la classe `'\D'` est équivalente à la classe `'[0-9]'` que nous avons vue tout à l'heure.

Le point est une classe de caractères à utiliser avec précaution : elle contient tous les caractères, à l'exception des caractères *Retour chariot* (`#13`) et *Nouvelle ligne* (`#10`).

La barre oblique inversée sert encore à former l'expression `'_'` qui désigne, non pas une classe de caractères, mais le début ou la fin d'un mot, entendu comme séquence de caractères alphanumériques. C'est ce qu'on appelle une ancre.

1.3 Quantificateurs

Puisque nous savons maintenant comment désigner des ensembles de caractères, voyons comment noter le nombre de caractères attendus. Nous avons vu, sans le remarquer jusqu'ici, que par défaut c'est le nombre un.

Pour indiquer un autre nombre, ou même un intervalle, on peut se servir des accolades, mais aussi des caractères `'+'`, `'*'`, et `'?'`. Les accolades permettent d'indiquer un nombre exact de caractères, ou un intervalle exact. Le symbole `'+'` veut dire « une fois ou plus ». L'étoile veut dire « zéro fois ou plus ». Le point d'interrogation veut dire « zéro ou une fois ».

Par exemple, l'expression `'2'` veut dire « deux chiffres » ; `'2,4'` veut dire « de deux à quatre chiffres » ; `'2,'` veut dire « deux chiffres ou plus ». L'expression `'_+'` veut dire « un chiffre ou plus » ; `'*'` veut dire « zéro chiffre ou plus » ; `'?'` veut dire « zéro chiffre

ou un ».

Par défaut, les opérateurs `'+'`, `'*'`, et `'?'` sont gourmands (*greedy* en anglais), c'est-à-dire que la recherche renvoie non pas la première correspondance trouvée, mais la plus longue. Cela a souvent son importance. Pour changer ce comportement, autrement dit pour rendre les opérateurs paresseux (*lazy*), il faut les faire suivre d'un point d'interrogation : `'+?'`, `'*?'`, `'??'`.

1.4 Autres opérateurs

Les parenthèses sont également des caractères spéciaux : elles servent à délimiter des groupes de caractères, par exemple pour y appliquer des quantificateurs. Ainsi l'expression `'(abc)?def'` désigne un ensemble contenant deux éléments, les chaînes `'abc-def'` et `'def'`. Le point d'interrogation rend le groupe `'abc'` facultatif.

Les parenthèses peuvent aussi être utilisées en combinaison avec la barre verticale, qui signifie « ou ». Par exemple l'expression `'a(b|c)d'` désigne un ensemble contenant les chaînes `'abd'` et `'acd'`.

Enfin, les caractères `'^'` et `'$'` signifient respectivement le début et la fin de la chaîne. Nous allons voir dans un instant quelle est l'utilité de ces caractères. Mais oui, vous avez raison, cela fait deux significations différentes pour l'accent circonflexe. Suivant l'endroit où on le place, il ne sera pas interprété de la même façon, et cela vaut aussi pour les autres caractères.

1.5 Exercices

Testez vos connaissances sur l'article en ligne : [lien 2](#).

2 Vérifier qu'une chaîne appartient à un ensemble

Il est temps à présent de faire connaissance avec l'unité *RegularExpressions*. Cette unité, apparue avec Delphi XE, va nous permettre d'effectuer diverses opérations sur des chaînes de caractères.

La plus simple de ces opérations est celle qui consiste à déterminer si telle chaîne appartient à tel

ensemble.

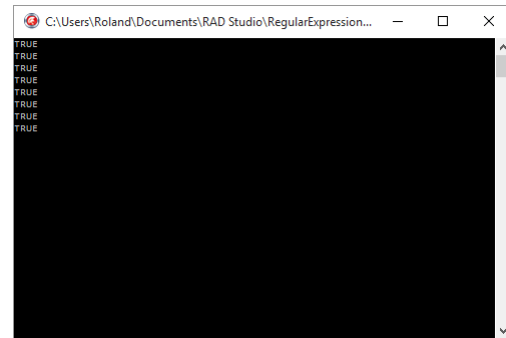
2.1 La fonction IsMatch

La fonction dont nous avons besoin pour cela est la fonction `IsMatch`. Elle peut être appelée directe-

ment, avec deux chaînes de caractères comme arguments : le sujet (la chaîne à étudier) et l'expression régulière.

```

1 program IsMatch1;
2
3 {$APPTYPE CONSOLE}
4
5 uses
6   RegularExpressions;
7
8 begin
9   WriteLn(TRegex.IsMatch('bonjour', '\w'
10  )); // un caractère alphanu-
11  mérique
12  WriteLn(TRegex.IsMatch('bonjour', '\w+'
13  )); // un ou plusieurs
14  caractères alphanumériques
15  WriteLn(TRegex.IsMatch('bonjour', '\w*'
16  )); // zéro ou plus
17  caractères alphanumériques
18  WriteLn(TRegex.IsMatch('bonjour', '\w{7}'
19  )); // sept
20  WriteLn(TRegex.IsMatch('bonjour', '[a-
21  z]{7}')); // sept minuscules
22  WriteLn(not TRegex.IsMatch('bonjour',
23  '\d')); // un chiffre
24  WriteLn(not TRegex.IsMatch('bonjour',
25  '\s')); // un espace au sens
26  large (équivalent à '[0-9]')
27  WriteLn(TRegex.IsMatch('bonjour', '\D'
28  )); // un caractère qui n'
29  est pas un chiffre
30  ReadLn;
31 end.
```



Comme vous l'avez remarqué si vous avez regardé de près le code ci-dessus, la fonction IsMatch vérifie seulement *qu'une partie au moins* de la chaîne passée comme premier argument appartient à l'ensemble représenté par l'expression régulière.

Si l'on veut s'assurer que la chaîne entière appartient à l'ensemble, il faut utiliser les opérateurs '^' et '\$', qui signifient respectivement le début de chaîne et sa fin.

```

1 WriteLn(TRegex.IsMatch('bonjour', '\w^'
2 )); // TRUE
3 WriteLn(not TRegex.IsMatch('bonjour',
4 '\w$')); // TRUE
```

Dans 'bonjour', il y a plusieurs caractères alphanumériques, mais il n'y a pas qu'un seul caractère entre le début et la fin de la chaîne.

2.2 Exercice

Testez vos connaissances sur l'article en ligne : [lien 3](#).

3 Extraire d'une chaîne des groupes de caractères

3.1 Exemple

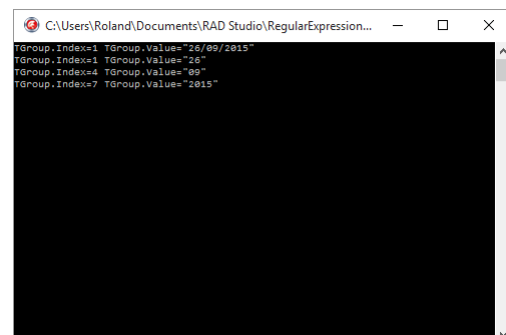
Imaginons que nous voulions, non seulement vérifier qu'une chaîne donnée appartient à un ensemble, mais aussi extraire de cette chaîne certains groupes de caractères. Disons par exemple que nous voulons extraire les groupes de chiffres d'une chaîne contenant une date, comme '26/09/2015'.

```

1 program Group1;
2
3 {$APPTYPE CONSOLE}
4
5 uses
6   SysUtils, RegularExpressions;
7
8 const
9   SUBJECT = '26/09/2015';
10  PATTERN = '(\d{2})/(\d{2})/(\d{4})';
11
12 var
13  expr: TRegex;
14  match: TMatch;
15  group: TGroup;
16
17 begin
18  expr := TRegex.Create(PATTERN);
```

```

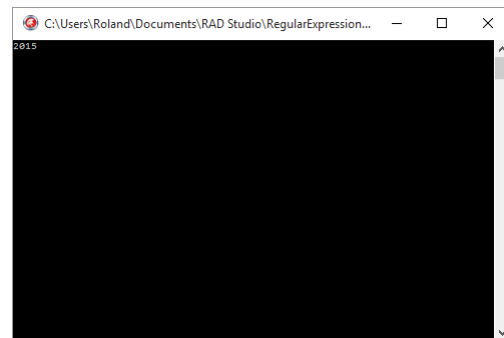
19
20 match := expr.Match(SUBJECT);
21
22 if match.Success then
23   for group in match.Groups do
24     WriteLn(Format('TGroup.Index=%d
25     TGroup.Value="%s"', [group.
26     Index, group.Value]));
27
28 ReadLn;
29 end.
```



Nous avons utilisé dans notre expression régulière des parenthèses pour délimiter les groupes de caractères à extraire, puis nous avons utilisé la fonction Match. Soit dit en passant, nous n'avons pas vérifié que les chiffres extraits formaient en effet une date valide. L'expression utilisée ne tient pas compte du fait qu'il n'y a que 31 jours au plus dans le mois, 12 mois dans l'année, etc. Une vérification complète devrait même tenir compte des années bissextiles, mais il est peu probable que cela soit faisable par les expressions régulières !

3.2 Groupes nommés

Nous pouvons également nommer les groupes. Pour nommer un groupe, il faut insérer juste après la parenthèse ouvrante les caractères '?<nom>'.
28 ReadLn;
29 end.



```

1 program Group2;
2
3 {$APPTYPE CONSOLE}
4
5 uses
6   SysUtils, RegularExpressions;
7
8 const
9   SUBJECT = '26/09/2015';
10  PATTERN = '(' + '?<day>' + '\d{2}')(
11             + '?<month>' + '\d{2}')(
12             + '?<year>' + '\d{4}');
13
14 var
15   group: TGroup;
16   match: TMatch;
17   regEx: TRegEx;
18
19 begin
20   regEx := TRegEx.Create(PATTERN, []);
21
22   match := regEx.Match(SUBJECT);
23
24   if match.Success then
25   begin
26     group := match.Groups['year'];
27     WriteLn(group.Value);
28   end;
29 end;

```

3.3 Parenthèses non capturantes

Quelquefois on a besoin de délimiter un groupe de caractères, mais on n'a pas besoin de capturer les caractères correspondants. Dans ce cas, pour ne pas donner de travail inutile au programme, on aura recours à des parenthèses non capturantes.

Imaginons par exemple qu'on veuille détecter dans un code source en Pascal les *Write* et les *WriteLn*, sans faire de différence entre les deux. On pourrait utiliser l'expression 'Write(Ln)?'. Mais de cette façon on capturerait à chaque fois ou une chaîne vide ou la chaîne 'Ln'. Cependant nous avons supposé qu'on ne voulait pas tenir compte de la différence entre *Write* et *WriteLn*. On utilisera donc l'expression suivante :

```
1 'Write(?:Ln)?'
```

Les caractères '? :?' ont été ajoutés immédiatement après la parenthèse ouvrante pour rendre les parenthèses non capturantes.

3.4 Exercice

Testez vos connaissances sur l'article en ligne : [lien 4](#).

4 Détecter des correspondances multiples

Il est possible de détecter les multiples groupes de caractères qui, dans une chaîne donnée, appartiennent à un même ensemble. Pour cela nous aurons besoin des fonctions Match, Success et NextMatch.

4.1 La fonction NextMatch

L'exemple suivant détecte les groupes de caractères représentant des nombres.

```

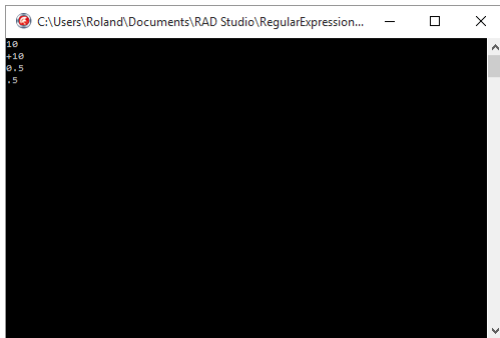
1 program Match1a;
2
3 {$APPTYPE CONSOLE}
4
5 uses
6   SysUtils, RegularExpressions;
7

```

```

8   var
9     match: TMatch;
10
11 begin
12   match := TRegEx.Match('10 +10 0.5 .5',
13                         '\s*[-+]?[0-9]*\.[0-9]+\s*');
14
15   while match.Success do
16   begin
17     WriteLn(match.Value);
18
19     match := match.NextMatch;
20   end;
21
22   ReadLn;
23 end.

```



```

14 begin
15   expr.Create('\w');
16
17   collection := expr.Matches('abc');
18
19   for i := 0 to collection.Count - 1 do
20     with collection[i] do
21       WriteLn(Format('%d %d %d %s', [i,
22         Index, Length, Value]));
23
24   ReadLn;
25 end.

```

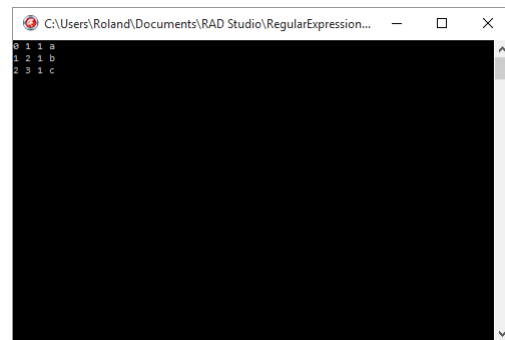
4.2 Le type TMatchCollection

La même opération peut être effectuée d'une autre façon : au moyen de la fonction Matches, laquelle renvoie un résultat de type TMatchCollection.

```

1 program MatchCollection1;
2
3 {$APPTYPE CONSOLE}
4
5 uses
6   SysUtils,
7   RegularExpressions;
8
9 var
10  expr: TRegex;
11  collection: TMatchCollection;
12  i: Integer;
13

```



4.3 Exercice

Testez vos connaissances sur l'article en ligne : [lien 5](#).

5 Éclatement d'une chaîne

Voyons à présent comment éclater une chaîne au moyen d'une expression régulière, c'est-à-dire la découper selon une certaine règle et disposer les morceaux découpés dans un tableau.

5.1 Exemple

L'expression représente un groupe de caractères à traiter comme délimiteurs. Dans le cas le plus simple, c'est un seul caractère :

```

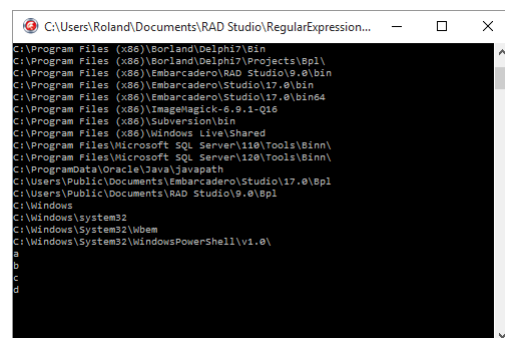
1 program Split1;
2
3 {$APPTYPE CONSOLE}
4
5 uses
6   SysUtils, RegularExpressions;
7
8 var
9   a: TArray<string>;
10  s: string;
11  i: integer;
12
13 begin
14   a := TRegex.Split(
15     GetEnvironmentVariable('PATH'), ',';
16
17   for s in a do

```

```

17   WriteLn(s);
18
19   a := TRegex.Split('a b,c-d', '[ , -]');
20
21   for i := 0 to High(a) do
22     WriteLn(a[i]);
23
24   ReadLn;
25 end.

```



5.2 Exercice

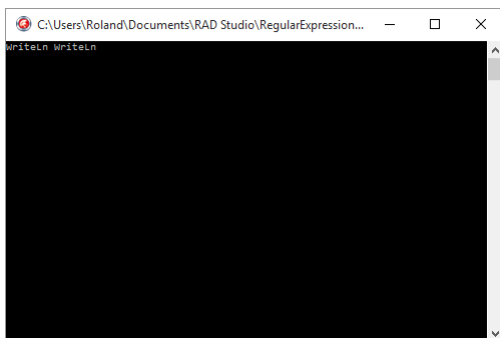
Testez vos connaissances sur l'article en ligne : [lien 6](#).

6 Remplacement de groupes de caractères

L'opération suivante ressemble un peu à la précédente : au lieu de traiter certains groupes de caractères comme délimiteurs, nous les remplacerons, au moyen de l'expression Replace.

6.1 Remplacement par une chaîne constante

```
1 program Replace1;
2
3 {$APPTYPE CONSOLE}
4
5 uses
6   RegularExpressions;
7
8 begin
9   WriteLn(TRegEx.Replace('WRITELN
10    writeln', 'writeln', 'WriteLn', [
11     roIgnoreCase]));
12   ReadLn;
13 end.
```



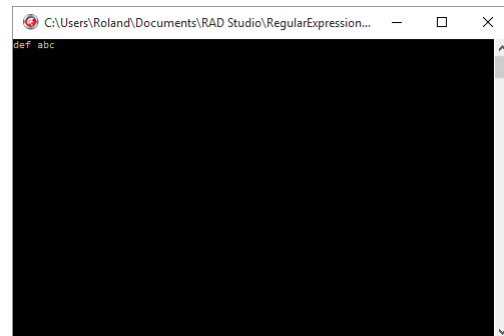
Comme vous le voyez, la fonction Replace admet quatre paramètres : la chaîne à traiter, l'expression régulière représentant les groupes à remplacer, la chaîne par laquelle ces groupes seront remplacés et enfin un ensemble d'options. Dans l'exemple ci-dessus, on a utilisé (sans aucune utilité d'ailleurs) l'option permettant d'obtenir un remplacement qui ne tienne pas compte de la casse des groupes à remplacer.

6.2 Remplacement par une chaîne composée

Au lieu de remplacer les groupes par une chaîne constante, on peut les remplacer par une chaîne variable, composée d'éléments des groupes détectés.

```
1 program Replace2;
2
3 {$APPTYPE CONSOLE}
4
5 uses
6   RegularExpressions;
7
8 var
9   expr: TRegEx;
10
11 begin
12   expr := TRegEx.Create('(\\w+)\\s(\\w+)');
13   WriteLn(expr.Replace('abc def', '\\2 \\1
14   '));
```

```
14 ReadLn;
15 end.
```



Dans la chaîne '\\2 \\1', la barre oblique inversée suivie d'un chiffre représente l'élément capturé qui devra être inséré à cette place dans la chaîne de remplacement. Ainsi, '\\1' signifie « le premier élément capturé ».

Pratique, non ? Mais ce n'est pas tout. La chaîne de remplacement peut même être le résultat d'une fonction recevant comme arguments les éléments capturés.

6.3 Remplacement par le résultat d'une fonction

Supposons que nous voulions valider une chaîne représentant une position donnée d'une partie d'échecs. La chaîne devra être conforme à la notation standard, à savoir la notation FEN (*Forsyth-Edwards Notation*). Voici la chaîne FEN correspondant à la position initiale d'une partie de jeu d'échecs :

```
1 'rnbqkbnr/pppppppp/8/8/8/PPPPPPPP/
2 RNBQKBNR w KQkq - 0 1'
```

Comme on le voit, l'occupation des lignes de l'échiquier est représentée par des groupes de caractères contenant des lettres (les pièces) et des chiffres (le nombre de cases vides consécutives). Nous devons nous assurer que le nombre total de cases pour chaque ligne de l'échiquier est bien égal à huit, et pour ce faire nous décidons de remplacer les chiffres par des caractères répétés autant de fois qu'il y a de cases vides.

```
1 type
2   TFENValidator = class
3     function ReplaceWith(const aMatch:
4       TMatch): string;
5     function ExpandRow(const aRow:
6       string): string;
7     function IsFEN(const aStr: string):
8       boolean;
9   end;
10
11 function TFENValidator.ReplaceWith(const
12   aMatch: TMatch): string;
```



```

9  const
10  EMPTY_SQUARE_SYMBOL = '-';
11  begin
12  result := StringOfChar(
      EMPTY_SQUARE_SYMBOL, StrToInt(
      aMatch.Groups.Item[1].Value));
13 end;
14
15 function TFENValidator.ExpandRow(const
      aRow: string): string;
16 begin
17   with TRegex.Create('[1-8]') do
18     result := Replace(aRow, ReplaceWith)
19   ;
19 end;
20
21 function TFENValidator.IsFEN(const aStr:
      string): boolean;
22 var
23   a,
24   b: TStringList;
25   expr: TRegex;
26   i: integer;
27   s: string;
28 begin
29   a := TStringList.Create;
30   b := TStringList.Create;
31
32   b.Delimiter := '/';
33   b.StrictDelimiter := TRUE;
34
35   a.DelimitedText := aStr;
36   result := (a.Count = 6);
37
38   if not result then
39     Exit;
40
41   b.DelimitedText := a[0];
42   result := result and (b.Count = 8);
43
44   if not result then
45     Exit;
46
47   expr.Create('[1-8BKNPQRbknqpr]+$');
48
49   for i := 0 to b.Count - 1 do
50   begin
51     s := ExpandRow(b[i]);
52     {WriteLn(s);}
53
54     result := result and expr.IsMatch(b[
      i]) and (Length(s) = 8);
55   end;
56
57   result := result and TRegex.IsMatch(a[
      1], '^(w|b)$');
58   result := result and TRegex.IsMatch(a[
      2], '^(KQkq|+|\-)$');
59   result := result and TRegex.IsMatch(a[
      3], '^(a-h|[36]|\-)$');
60
61   expr.Create('[^d+$');
62
63   result := result and expr.IsMatch(a[4
      ]) and (StrToInt(a[4]) >= 0);
64   result := result and expr.IsMatch(a[5
      ]) and (StrToInt(a[5]) >= 1);
65 end;

```

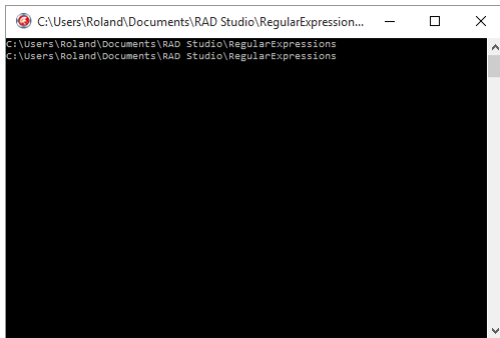
Voici un autre exemple de remplacement par fonction. C'est un programme qui cherche dans une chaîne de caractères des variables à remplacer par leur valeur. Les variables sont notées suivant la

syntaxe utilisée pour les variables d'environnement comme %date% ou %username%. La fonction renvoyant la chaîne à substituer tirera ses résultats d'un dictionnaire.

```

1  program Replace3b;
2
3  {$APPTYPE CONSOLE}
4
5  uses
6   System.SysUtils,
7   System.Classes,
8   System.RegularExpressions;
9
10 type
11   TExpander = class
12     fDictionary: TStrings;
13     constructor Create(aDictionary:
      TStrings);
14     function ReplaceWith(const aMatch:
      TMatch): string;
15     function Expand(const s: string):
      string;
16   end;
17
18 constructor TExpander.Create(aDictionary
      : TStrings);
19 begin
20   fDictionary := aDictionary;
21 end;
22
23 function TExpander.ReplaceWith(const
      aMatch: TMatch): string;
24 begin
25   result := fDictionary.Values[aMatch.
      Groups.Item[1].Value];
26 end;
27
28 function TExpander.Expand(const s:
      string): string;
29 var
30   expr: TRegex;
31 begin
32   expr.Create('%(.+)%');
33   result := expr.Replace(s, ReplaceWith)
34   ;
34 end;
35
36 var
37   dictionary: TStrings;
38   expander: TExpander;
39
40 begin
41   dictionary := TStringList.Create;
42   expander := TExpander.Create(
      dictionary);
43
44   dictionary.Values['REPertoire_PROJET']
      := ExtractFileDir(ParamStr(0));
45
46   WriteLn(expander.Expand('%
      REPertoire_PROJET%'#13#10%'
      REPertoire_PROJET%'));
47
48   dictionary.Free;
49   expander.Free;
50
51   ReadLn;
52 end.

```



6.4 Exercice

Testez vos connaissances sur l'article en ligne : [lien 7](#).

7 Expressions alternatives

Rassurez-vous, nous avons pratiquement fini notre tour des fonctions de l'unité *RegularExpressions*. Si vous avez encore une minute, je vous propose un dernier exemple montrant comment on peut utiliser des expressions alternatives, pour savoir si une chaîne donnée appartient à tel ensemble, ou à un tel autre, ou à un troisième, etc.

7.1 Exemples

Dans le langage des expressions régulières, c'est le caractère '|' qui symbolise cette alternative.

```

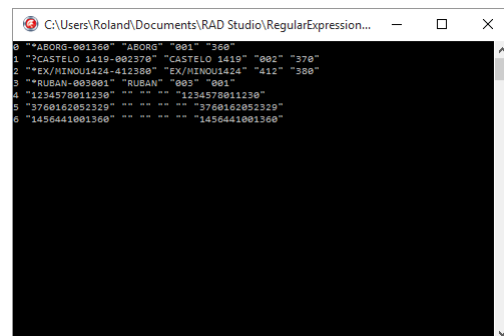
1 program Alternative1;
2
3 {$APPTYPE CONSOLE}
4
5 uses
6   SysUtils, RegularExpressions;
7
8 const
9   SAMPLE: array[0..10] of string = (
10    '*ABORG-001360',
11    '?CASTELO 1419-002370',
12    '*EX/MINOU1424-412380',
13    '*RUBAN-003001',
14    '1234578011230',
15    '3760162052329',
16    '1456441001360',
17    'AAAAAAA',
18    '*AAAAA-001',
19    '*AAAAAA001360',
20    '123121212',
21   );
22
23 const
24   PATTERN1 = '^[*?]([:-]{1,15})-(\d{3})
25              (\d{3})$';
26   PATTERN2 = '^(\d{10})$';
27   PATTERN3 = '^(\d{13})$';
28   PATTERN = PATTERN1 + '|' + PATTERN2 +
29             '|' + PATTERN3;
30
31 var
32   group: TGroup;
33   expr: TRegex;
34   match: TMatch;
35   i: integer;
36
37 begin
38   expr := TRegex.Create(PATTERN);

```

```

38   for i := Low(SAMPLE) to High(SAMPLE)
39     do
40       begin
41         match := expr.Match(SAMPLE[i]);
42         if match.Success then
43           begin
44             Write(i);
45             for group in match.Groups do
46               Write(' ', group.Value, ' ');
47             WriteLn;
48           end;
49         end;
50       end;
51   ReadLn;
52 end.

```



Le code ci-dessus affiche tous les résultats sans les analyser. Si l'on veut savoir à laquelle des expressions régulières correspond tel groupe de caractères, on pourra procéder de la façon suivante :

```

1 procedure ShowResult(aGroupName: string)
2   ;
3   begin
4     WriteLn(Format(' %s = "%s"', [
5       aGroupName, match.Groups[
6         aGroupName].Value]));
7   end;
8
9   begin
10    expr := TRegex.Create(PATTERN);
11
12    for i := Low(SAMPLE) to High(SAMPLE)
13      do
14        begin
15          WriteLn('SAMPLE[' + i + ']');
16          match := expr.Match(SAMPLE[i]);
17          if match.Success then
18            begin
19              case match.Groups.Count of

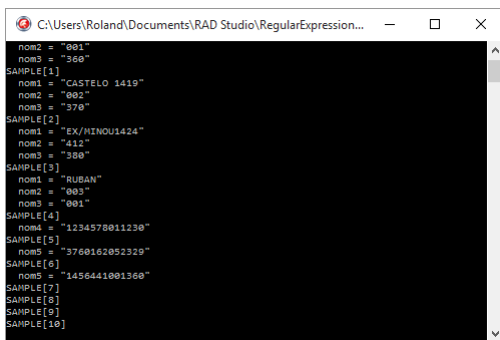
```



```

16     4: begin
17         ShowResult('nom1');
18         ShowResult('nom2');
19         ShowResult('nom3');
20     end;
21     5: ShowResult('nom4');
22     6: ShowResult('nom5');
23 end;
24 end;
25 end;
26
27 ReadLn;
28 end.

```



On remarque dans l'exemple ci-dessus que lorsque la chaîne étudiée correspond à la troisième expression alternative, la variable `match.Groups.Count` vaut 6, parce que le groupe capturé correspond à la sixième sous-expression entre parenthèses.

Ce comportement peut être modifié, de telle sorte que le comptage des groupes soit fait séparément pour chacune des expressions alternatives, au lieu d'être un comptage global. Pour cela, il faudra utiliser l'expression suivante :

```

1 PATTERN = '(?|' + PATTERN1 + '|' +
  PATTERN2 + '|' + PATTERN3 + ')';

```

Attention, le comptage pour chaque expression alternative commencera à deux, à cause des parenthèses que nous avons ajoutées. Ces parenthèses produiront une capture supplémentaire (et inutile).

7.2 Exercice

Testez vos connaissances sur l'article en ligne : [lien 8](#).

8 Unicode

L'unité *RegularExpressions* supporte l'Unicode. Voici une expression régulière représentant les lettres capitales de l'alphabet grec :

```

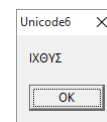
1 program Unicode6;
2
3 {$APPTYPE CONSOLE}
4
5 uses
6   System.RegularExpressions,
7   Vcl.Dialogs;
8
9 var
10  s: UnicodeString;
11  match: TMatch;

```

```

12
13 begin
14   s := 'IXΘΥΣ';
15
16   match := TRegEx.Match(s, '[Α-Ω]{5}');
17
18   ShowMessage(match.Value);
19 end.

```



9 Conclusion

L'unité *RegularExpressions* est basée sur l'unité *RegularExpressionsCore* qui, sauf le nom, est identique à l'unité *PerlRegEx* de Jan Goyvaerts.

```

1 uses
2   {$IF CompilerVersion >= 22.0}
3   RegularExpressionsCore;
4   {$ELSE}
5   PerlRegEx; (* http://www.regular-
  expressions.info/download/
  TPerlRegEx.zip *)
6   {$IFEND}

```

L'unité *PerlRegEx* est, elle, basée sur la bibliothèque PCRE (Perl Compatible *Regular Expressions*) de Philip Hazel.

sions) de Philip Hazel.

Si vous souhaitez en savoir plus sur la syntaxe des expressions régulières, vous pouvez consulter le tutoriel de Jan Goyvaerts : [Regular Expressions \(lien 9\)](#). Vous pouvez télécharger les exemples commentés dans cet article (et quelques autres) : [lien 10](#).

La solution des exercices se trouve dans le dossier *Solutions*. Le dossier *RegularExpressionsCore* contient des exemples d'utilisation de l'unité *RegularExpressionsCore* ou *PerlRegEx*.

Les exemples ont été testés avec Delphi XE2.

Retrouvez l'article de **Roland Chastain** en ligne : [lien 11](#)



Windows

Les dernières news Windows

Avez-vous installé Windows 10 ? Quelles sont vos impressions sur ce nouvel OS de Microsoft ?

La fin brutale du support de Windows XP n'a pas incité ses utilisateurs à migrer vers Windows 8. Windows XP a en effet continué tranquillement son chemin malgré les mises en garde de Microsoft, mais en cédant toutefois quelques parts de marché à Windows 7. Avec Windows 8, Microsoft n'a pas pu convaincre les utilisateurs dont la majorité se trouvait dépaycée sur le nouvel OS. La firme de Redmond a donc sorti la version 8.1 pour rattraper ses erreurs et a ainsi réussi à remonter d'un petit cran la pente.

Avec une plus large communication, Windows 10 a réussi à embarquer de nombreux utilisateurs. Il faut dire que Microsoft a beaucoup misé sur son nouvel OS et espère dominer le marché des mobiles et des objets connectés avec les innovations apportées dans Windows 10.

Cela fait maintenant plus d'un mois que Windows 10 est sorti. Le nouvel opus de Microsoft a été téléchargé plus de 14 millions de fois en seulement 24 heures et cumule actuellement plus de 75 millions d'installations en moins de 6 semaines. La firme a déjà effectué trois mises à jour cumulatives

pour corriger les bogues et stabiliser l'OS sur les différentes plateformes (smartphones, tablettes, PC...). Si pour certains, Windows 10 est la meilleure version depuis Windows XP, pour d'autres, c'est la meilleure manière que Microsoft a trouvée pour faire de l'espionnage par la collecte des données utilisateur.

Lors d'un précédent sondage, nous vous avons demandé si vous alliez migrer : Avez-vous/allez-vous migrer vers Windows 10 ? Une large majorité des répondants a indiqué qu'elle va migrer ou l'a déjà fait. Mais qu'en est-il vraiment ? Faites-vous partie de ces personnes qui ont déjà installé Windows 10 ? Sur la toile, nous trouvons toutes sortes de remarques sur l'installation : des problèmes, des personnes satisfaites, mais qu'en est-il de votre nouvelle expérience ?

Êtes-vous satisfait de cette nouvelle version ?

Microsoft a-t-il réussi un bel OS ?

La migration s'est-elle bien passée ?

Avez-vous eu des problèmes de compatibilité avec certaines applications ? Lesquelles ?

Commentez la news de **Francis Walter** en ligne : [lien 12](#)

CRM



Les derniers tutoriels et articles

Comment intégrer le framework PAD dans un environnement Salesforce.com

Le framework PAD est un processus de développement favorisant la normalisation d'un projet avec quelques bonnes pratiques à respecter. Il facilite ainsi l'arrivée d'un nouveau développeur au sein de l'équipe d'un projet. Cet article explique comment l'intégrer dans un environnement Salesforce.com.

1 Introduction

Lorsque l'on travaille sur un projet Salesforce.com, il peut arriver de reprendre le code d'une autre équipe ou de travailler avec plusieurs personnes sur un même projet et peut-être même des personnes parlant une autre langue. De ce fait, ça peut rapidement devenir le désordre s'il n'y a pas un minimum de rigueur et de règles mises en place.

Le framework PAD est là pour corriger tout ça, il s'agit d'un processus de développement favorisant la normalisation d'un projet. Ainsi, lorsqu'un développeur rejoint une équipe, il n'a pas besoin de s'adapter à son équipe et il s'y retrouve ainsi beaucoup plus facilement.

Cela permet à un projet de gagner :

- en vitesse de développement ;
- en coût ;
- en délivrabilité.

Que demande le peuple (enfin, surtout les managers) ?

Je préfère préciser tout de suite pour qu'il n'y ait pas d'ambiguïté, que je ne suis pas le créateur de ce framework, mais simplement un utilisateur. Les créateurs sont Jean-Luc Antoine et Sovan Bin.

Avant d'aller plus loin dans cet article, je vous propose quelques rappels sur les termes techniques qui vont être utilisés :

- Salesforce.com ([lien 13](#)) est un CRM (Customer Relationship Manager) orienté SaaS (Soft-

ware as a Service : [lien 14](#)) et PaaS (Platform as a Service : [lien 15](#)) permettant la gestion des relations clients d'une entreprise ;

- Apex, est un langage de programmation côté serveur permettant de modifier la logique métier et le traitement des données ;
- SOQL (Salesforce Object Query Language) est le langage permettant d'effectuer des requêtes dans la base de données de Salesforce.com.

Le but de cet article n'est pas que vous maîtrisiez complètement cette méthode de développement (je ne la connais pas entièrement non plus) mais plutôt que vous en compreniez le principe pour que vous sachiez l'intégrer vous-même dans une instance Salesforce.com.

Afin de faciliter la compréhension du concept, je vais l'intégrer dans un petit projet comprenant une page Visualforce avec un contrôleur Apex ainsi qu'un trigger Apex.

Le scénario du projet est le suivant, un manager souhaite suivre les opportunités de son équipe commerciale et ainsi consulter l'avancement des objectifs de chacun par rapport à celui attendu.

Bien entendu, certains de mes arguments peuvent être sujets à discussion, je donne simplement mon point de vue par rapport à mon expérience, à ce que l'on m'a conseillé ou à ce que j'ai pu entendre et constater.

2 Prérequis

Avant toute chose, vous devez disposer d'une instance afin de pouvoir suivre ce tutoriel. Si ce n'est pas le cas, je vous invite à vous rendre à cette adresse : [lien 16](#). Salesforce met à disposition plusieurs types d'instances qui proposent différentes

fonctionnalités et donc à différents prix (prix par licence utilisateur par mois). La seule instance que vous pourrez généreusement obtenir de la part de Salesforce est celle de développement (celle que je vous propose via le lien ci-dessus) qui, comme son nom

l'indique, sert à tester et développer des services. Je vous laisse juger par vous-même la grille tarifaire sur les diverses instances que propose le CRM : lien 17.

Vous aurez également besoin d'un environnement de développement pour écrire votre code, vous pouvez utiliser celui fourni par Salesforce ou alors vous procurer celui compatible avec Eclipse et qui s'ap-

3 Récupérer le framework

Derrière cette adresse mail, il y a un batch qui tourne toutes les 10 minutes environ et qui vous enverra la dernière version du framework accompagné d'un document PowerPoint vous expliquant le fonctionnement en détail de tout ceci. À l'heure où j'écris

pelle Force.com IDE (je l'utiliserai tout au long de ce tutoriel). Pour ce dernier, rendez-vous sur l'Eclipse Marketplace et installez-le.

Pour récupérer le framework PAD, c'est assez simple, il vous suffit d'envoyer un mail à l'adresse contact en y saisissant le mot-clé PAD (dans l'objet du mail par exemple) : lien 18.

cet article, la dernière version est la 2.14.

Le framework se compose de deux composants :

- une classe Apex ;
- un composant Visualforce.

4 Configurer une instance Salesforce.com avec le framework PAD

Comme précisé dans le chapitre précédent, le framework n'est composé que de seulement deux composants, mais il faut également créer un champ personnalisé sur l'objet User (j'utilise Salesforce.com en anglais) de type multipicklist : PAD_BypassTrigger__c, peu importe le label, ce qui est important ici est le nom API puisqu'il est utilisé dans la classe Apex du framework, vous comprendrez tout à l'heure le pourquoi du comment.

Voilà à quoi doit ressembler le champ :



Maintenant que le champ personnalisé est créé, vous pouvez insérer dans Salesforce.com, la classe Apex PAD ainsi que le composant Visualforce du même nom.

Si vous le faites dans l'autre sens, vous ne pourrez pas sauvegarder puisque Salesforce vous dira que vous mentionnez une référence qui n'existe pas.

Je pense que la classe Apex date d'un certain moment puisque lorsque j'ai tenté de l'intégrer dans

un projet, Salesforce.com me l'a refusé et pour plusieurs raisons :

- les méthodes de test ne peuvent pas cohabiter dans le même fichier qu'une classe. Chaque classe de test doit maintenant obligatoirement avoir son propre « fichier » ;
- certaines méthodes depuis la version 31 de l'API ont été supprimées et ne peuvent plus être utilisées (Limits.getFieldsDescribes(), Limits.getPicklistDescribes() et Limits.getScriptStatements()) dans notre cas).

Pour corriger cela, j'ai dû créer une nouvelle classe (PAD_TEST) pour y couper et coller les méthodes de test de la classe Apex et commenter les lignes 80, 85 et 89.

En attendant que cela soit corrigé par les créateurs du framework PAD, je vous fournis tout cela dans une archive compressée disponible dans la partie 8 : Ressources utiles.

5 Quelques bonnes pratiques

Pour des raisons d'optimisation du temps de développement (compréhension du projet, des fonctionnalités et des développements déjà effectués), la méthode de conception PAD dispose de best practices qu'il est recommandé de respecter afin d'avoir un projet bien ordonné et dont je vais vous en présenter quelques-unes (puisque je ne les connais pas toutes).

Ce chapitre n'étant pas une étape obligatoire à l'intégration du framework PAD dans un environnement Salesforce, vous pouvez passer au chapitre suivant 6 : Utilisation du framework PAD.

5.1 Les triggers Apex

Comme il n'est pas possible dans Salesforce.com de choisir l'ordre d'exécution de deux triggers portant sur un même objet et un même événement (l'ordre peut différer d'un environnement à un autre), il est nécessaire de créer un seul trigger Apex par objet et par événement pour ensuite placer dans l'ordre que vous le souhaitez, vos traitements. Il est même recommandé de ne pas mettre vos traitements dans vos triggers Apex, mais de créer des classes pour ça et de les appeler depuis vos triggers, cela permet de factoriser vos traitements et d'y voir

plus clair, car cela minimise les lignes dans le trigger Apex et si plusieurs ont besoin de cette même méthode, elle est stockée à un seul endroit et il n'y a pas de redondance du code.

5.2 Les étiquettes personnalisées

Ne jamais utiliser de valeur en dur dans votre code, toujours passer par une étiquette personnalisée pour la simple et bonne raison que c'est bien plus maintenable et que si, cette valeur vient à changer, il n'y aura simplement qu'à modifier la valeur du custom label alors que si la valeur est directement ancrée dans votre développement, chaque modification souhaitée en production entraînera obligatoirement un nouveau déploiement et fera donc appel à un développeur et cela lui prendra du temps pour presque rien.

5.3 Les règles de nommage

À quoi servent les règles de nommage ? Elles servent tout simplement à ce que le nom soit très explicite sur le comportement ou le but (un contrôleur, une simple classe Apex, un batch Apex), sur quels objet et événement porte un trigger, etc.

Les règles de nommage portent sur tout ce qui se crée dans Salesforce :

- les classes Apex ;
- les triggers Apex ;
- les pages Visualforce ;
- les étiquettes personnalisées ;
- les règles de validation ;
- les workflows ;
- etc.

5.3.a Les classes Apex

Elles doivent commencer par « AP » suivi de leur index et enfin d'un nom très court décrivant son utilité. Par exemple, si vous avez déjà trois classes Apex présentes dans votre instance Salesforce.com et que la prochaine que souhaitez créer porte sur les comptes et doit compter le nombre de contacts qui lui sont rattachés ayant le type d'enregistrement « Blue » (champ Record Type, vous pouvez attribuer à cette classe ce nom (à titre d'exemple) AP04_Count_Blue_Contacts.

5.3.b Les pages Visualforce

Les pages Visualforce commencent toujours par « VF » suivi de leur index (le nombre total de pages déjà créées + 1) et d'un nom décrivant leur fonction. Par exemple VF02_CreateNewUser.

5.3.c Les contrôleurs Visualforce

Les contrôleurs Visualforce sont des classes Apex qui servent à exécuter des traitements lors d'interactions avec des pages Visualforce. De ce fait, ils portent le même nom que les pages Visualforce à l'exception qu'ils possèdent en plus un « C » après le « VF » soit en reprenant l'exemple ci-dessus de la page Visualforce, cela donnerait VFC02_CreateNewUser.

5.3.d Les triggers Apex

Comme vu précédemment, un trigger ne doit porter que sur un objet et que sur un événement pour s'assurer de l'ordre d'exécution du code que l'on souhaite. Son nom dépend donc des conditions citées précédemment, c'est-à-dire qu'il comporte une abréviation du nom de l'objet sur lequel il porte et du nom de l'événement. Par exemple, pour un trigger sur les comptes se déclenchant avant la mise à jour d'un enregistrement, il se nommera Account-BeforeUpdate ou OpportunityAfterInsert pour une opportunité après sa création.

5.3.e Les étiquettes personnalisées

Les étiquettes personnalisées (custom labels en anglais) servent à stocker toutes sortes de valeurs (une phrase traduite dans plusieurs langues, l'identifiant d'un enregistrement, le nombre d'enregistrements retournés pour une pagination, etc.) et peuvent être utilisées dans les pages Visualforces, des contrôleurs Apex, des triggers Apex ou encore des formules personnalisées (custom formulas).

Cela vous permet d'éviter de stocker des valeurs en dur dans votre code et vous permet ainsi une plus grande maintenabilité si vous devez modifier des valeurs puisque vous n'avez pas besoin d'effectuer un nouveau déploiement en production, juste la modification de l'étiquette personnalisée suffit et le tour est joué.

Pour leur nommage, je préconise de préfixer avec l'endroit où il est utilisé comme le nom d'une page Visualforce, d'une règle de validation ou le nom d'un profile, etc.

5.3.f Les workflows

Pour correctement nommer ses workflows, il faut les préfixer avec le nom de l'objet sur lequel il porte, puis de à quel(s) moment(s) il se déclenche.

Par exemple, s'il s'agit d'un workflow qui se déclenche à la création de comptes localisée dans la région de l'Île-de-France, il pourrait s'appeler Account Creation Ile de France.

5.3.g Les actions de mise à jour de champ

Le nommage des field updates est assez similaire à celui des workflows, à la différence qu'il suffit de spécifier sur quel objet il agit ainsi qu'une brève description de son action.

S'il met à jour le département d'un compte, son nom pourrait être Account update name.

5.3.h Les règles de validation

Les règles de validations ont un nom assez simple. C'est-à-dire qu'il faut qu'elles commencent par « VR » (pour validation rule) suivi de leur index (nombre de validation rules + 1) ainsi que de l'objet sur lequel elle porte et enfin d'une brève description, exemple VR03_Account_Mandatory_PostalCode.

5.3.i Les classes de test

Elles se nomment de la même façon que la classe Apex ou le contrôleur Visualforce qu'elles testent sauf qu'elles finissent par « _TEST ». Grâce à cela, vous pouvez facilement savoir qu'elle est la classe de test d'une classe ou d'un contrôleur et elle est également placée juste en dessous dans la liste.

5.4 Penser Bulk

Certes, il est plus facile lorsque que l'on développe un trigger de se dire que lorsque l'on crée ou met à jour un enregistrement, il n'y en aura forcément qu'un seul dans la liste des enregistrements du trigger (Trigger.new). Mais dans ce cas, que se passe-t-il si vous insérez des enregistrements en masse via le dataloader de Salesforce.com ?

Votre trigger ne fonctionnera pas correctement et vous pouvez vous retrouver avec des données erronées, ce qui est embêtant pour une production.

C'est pourquoi il est indispensable que lorsque l'on effectue un développement, que ce soit un trigger Apex, une classe Apex, une page Visualforce ou autres, de toujours penser Bulk (grande quantité de données).

5.5 Utiliser la méthode System.debug()

Cette méthode peut devenir votre meilleur ami lorsque vous effectuez du débogage puisqu'elle sert à afficher dans les logs, le texte que vous lui donnez en paramètre.

Je vous recommande de l'utiliser au début et à la fin de vos méthodes et de vos triggers Apex afin de suivre en détail le cheminement de l'exécution de vos méthodes. Vous pouvez également afficher le résultat de vos requêtes et insérer du débogage dans vos conditions (if) pour voir où votre code passe.

5.6 Commenter son code

Cela peut vous paraître inutile, mais il s'agit là simplement de clarté et de gagner du temps lorsque vous revenez sur votre code plusieurs mois après l'avoir développé ou que vous intervenez sur le travail d'une autre personne. Cela afin de plus facilement rentrer dans sa logique et de vous rappeler plus facilement son utilité. Vous pouvez aussi ajouter des commentaires au début de vos classes pour rapidement expliquer dans quelles circonstances elles sont utilisées et de même qu'à vos fonctions (paramètres d'entrées, de sorties, ce qu'elles font, expliquer une condition if).

5.7 Les classes de test

Même s'il s'agit de classes de test et qu'elles ne seront pas exécutées en production en utilisation normale (mis à part lors d'un déploiement), cela n'empêche pas qu'elles doivent être correctement développées pour être performantes et le code doit être factorisé au maximum pour éviter la redondance.

Tout ce qui est plusieurs fois doit être factorisé dans une fonction et vous devez ensuite l'appeler.

Cela facilite la maintenance lorsqu'il y a une modification à effectuer, il ne vous faudra le faire qu'à un seul endroit et non à plusieurs.

Par exemple, dans une classe de test, on met généralement en œuvre plusieurs méthodes de test et il faut donc un jeu de données pour chacune d'entre elles. Donc au lieu d'écrire plusieurs fois le même jeu de données, que doit-on faire ? Écrire une méthode et y insérer un jeu de données et appeler cette méthode au début de chaque méthode de test.

Tout comme lorsqu'il s'agit d'initialiser une page Visualforce, cela peut être fait à l'intérieur d'une méthode.

Quel est le bon taux de couverture à obtenir pour vos classes ?

Salesforce exige au minimum 75 % de couverture générale sur tout l'environnement, mais plus cette couverture sera élevée, plus vous vous assurerez du bon fonctionnement de votre code.

Mais comment s'assurer que lorsque vous exécutez votre classe de test, même si vous obtenez un taux de couverture de 90 % ou plus, que tout s'est correctement déroulé et que le résultat obtenu est bien celui attendu ?

Pour cela, vous avez une tripotée de méthodes pour vérifier vos valeurs obtenues en renseignant également celles que vous auriez dû obtenir ou même ne pas obtenir, ce sont les méthodes System.assert(condition, msg), System.assertEquals(expected, actual, msg) et System.assertNotEquals(expected, actual, msg). Vous pouvez retrouver la documentation à ce sujet pour en apprendre davantage : [lien 19](#).

J'en profite à ce sujet pour proposer un article que j'ai rédigé sur les tests dans Salesforce : Tester son code Apex (lien 20) et How To Test Your Apex Triggers (rédigé en anglais et intégré dans la bibliothèque technique de Salesforce) (lien 21).

6 Utilisation du framework PAD

Le framework PAD a quelques fonctionnalités plutôt intéressantes.

Si vous respectez la règle d'un trigger Apex par objet et par événement et de la règle de nommage des classes Apex (cf. chapitre 5.3.1. Les classes Apex), en plus de choisir l'ordre d'exécution de vos traitements, vous pouvez également décider si un utilisateur a le droit ou non d'exécuter un trigger Apex grâce au fameux champ

7 Le projet

Revenons au projet, le but sera qu'un manager puisse visualiser les opportunités de son équipe commerciale groupées par mois et années avec leur objectif en cours par rapport à celui qu'elle doit atteindre où il pourra le modifier en temps réel.

Pour ce faire, nous aurons besoin d'une page Visualforce accompagnée de son contrôleur Apex et d'un trigger Apex qui mettra à jour les objectifs des commerciaux.

7.1 Le modèle de données

Le modèle de données pour ce projet est assez simple.

Nous avons simplement besoin d'un objet personnalisé que l'on appellera Commercial objective où l'on y stockera pour un commercial :

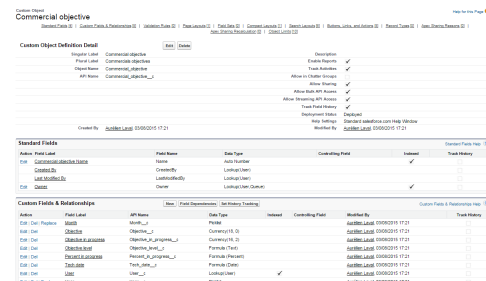
- son objectif;
- le mois de l'objectif;
- l'année de l'objectif;
- l'objectif en cours (à combien il en est actuellement);
- le pourcentage de réussite de son objectif;
- une représentation graphique si le commercial a réussi ou non à atteindre son objectif (croix verte = Oui, croix rouge = Non);
- une date représentant le mois et l'année de l'objectif (utilisé pour filtrer dans les rapports et les tableaux de bord);
- le lien vers le commercial.

5.8 Nom des méthodes et des variables explicites

Toujours pour une facilité de compréhension de votre code, n'hésitez pas à nommer vos méthodes ainsi que vos variables de façon explicite, c'est-à-dire que vous compreniez tout de suite pourquoi elles existent grâce à une simple lecture, cela vous évite de relire tout votre code pour comprendre et vous évite donc une perte de temps inutile.

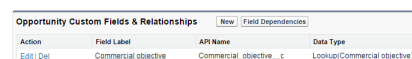
PAD_BypassTrigger__c sur l'objet User» dont je vous avais parlé dans le chapitre IV. Configurer une instance Salesforce.com avec le framework PAD et à la méthode PAD.canTrigger() en renseignant en paramètre le nom de la classe Apex (mais uniquement le AP[index] (« AP03 » par exemple)) pour que celle-ci ne soit exécutée qu'une seule fois puisque PAD garde en mémoire, le nom des classes exécutées. Cela évite ainsi les appels en cascade.

Ci-dessous, une image de à quoi doit ressembler l'objet :

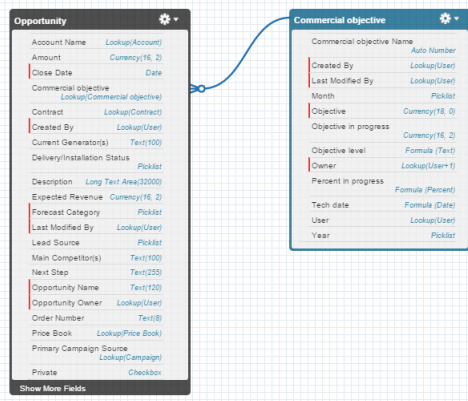


Le champ Objective in progress sera calculé par rapport à la somme des champs Amount sur les opportunités qui lui sont reliées.

Pour ce faire, nous avons besoin de créer une relation entre les opportunités et les objectifs :



Pour finir sur cette partie, je vous montre un aperçu du Visualisateur de schéma (Schema Builder) dans Salesforce :



Il permet de visualiser le modèle de données, mais également de créer des objets personnalisés, des champs personnalisés, etc.

7.2 Le contrôleur Apex

Ce que nous avons besoin de faire dans le contrôleur Apex est de récupérer les opportunités des commerciaux du mois et de l'année sélectionnés (via la page Visualforce).

Nous partons du principe que le manager voit toutes les opportunités de ses commerciaux.

Nous avons simplement besoin de récupérer en paramètres, le mois et l'année sélectionnés ensuite de récupérer les objectifs des commerciaux ainsi que les opportunités qui leur sont associées.

Afin de faciliter l'affichage des données dans la page Visualforce, j'ai créé une classe interne dans laquelle je stocke le commercial, l'enregistrement Objective et la liste des opportunités de l'objectif.

Voici le code du contrôleur :

```

1 public with sharing class VFC01
2     _OpportunitiesByCommercial {
3
4     public Commercial_objectif__c
5         theFakeCommercialObjectif {get;
6         set;}
7
8     public List<WrapperClass> wrapperList
9         {get; set;}
10
11     private List<User> commercialList;
12     private List<Commercial_objectif__c>
13         theObjectifList;
14     private Date dateToday;
15     public static map<Integer, String>
16         monthMap = new map<Integer,
17         String>{
18         1 => 'January',
19         2 => 'February',
20         3 => 'March',
21         4 => 'April',
22         5 => 'May',
23         6 => 'June',
24         7 => 'July',
25         8 => 'August',
26         9 => 'September',
27         10 => 'October',
28         11 => 'November',
29         12 => 'December'
30     };

```

```

31 /** Constructeur du contrôleur Apex
32     */
33     public VFC01
34         _OpportunitiesByCommercial () {
35         this.theFakeCommercialObjectif =
36         new Commercial_objectif__c ();
37
38         this.dateToday = Date.today ();
39
40         this.setMonthAndYear ();
41
42         this.commercialList = this.
43         getCommercialList ();
44
45         this.wrapperList = this.
46         getOpportunitiesByCommercial (
47         theFakeCommercialObjectif.
48         Month__c,
49         theFakeCommercialObjectif.
50         Year__c, this.commercialList)
51         ;
52     }
53
54 /** Récupère la liste des objectifs
55     et des opportunités par rapport
56     au mois et l'année sélectionnés
57     et de la liste des commerciaux */
58     /
59     public List<WrapperClass>
60     getOpportunitiesByCommercial (
61     String theSelectedMonth, String
62     theSelectedYear, List<User>
63     theCommercialList) {
64     List<WrapperClass> result = new
65     List<WrapperClass> ();
66
67     try {
68         theObjectifList = [
69             SELECT Id, User__c,
70                 Month__c, Year__c,
71                 Objectif__c,
72                 Percent_in_progress__c
73             ,
74                 Objectif_in_progress__c
75             , (SELECT Id, Name,
76                 CloseDate, Amount,
77                 Account.Id, Account.
78                 Name FROM
79                 Opportunities__r)
80             FROM
81                 Commercial_objectif__c
82
83             WHERE User__c IN :
84                 theCommercialList
85             AND Year__c = :
86                 theSelectedYear
87             AND Month__c = :
88                 theSelectedMonth
89             ];
90
91         map<Id,
92             Commercial_objectif__c>
93             objectifMap = this.
94             setObjectifMap (
95             theObjectifList);
96
97         for (User anUser :
98             theCommercialList) {
99             if (objectifMap.
100                 containsKey (anUser.Id
101                 )) {

```



```

55         Commercial_objectif__c
           anObjectif =
           objectifMap.get(
           anUser.Id);
56
57         result.add(new
58             WrapperClass(
59                 anUser,
60                 anObjectif.
61                 Opportunities__c,
62                 anObjectif
63             ));
64     }catch(Exception e){
65         this.displayErrorMessage(e.
66             getMessage());
67     }
68     return result;
69 }
70
71 /** Récupère la liste des commerciaux
72 **/
73 public List<User> getCommercialList()
74 {
75     return [
76         SELECT Id, Name
77         FROM User
78     ];
79
80 /** Configure le mois et l'année **/
81 public void setMonthAndYear(){
82     if(theFakeCommercialObjectif.
83         Year__c == NULL ||
84         theFakeCommercialObjectif.
85         Year__c.equals('')){
86         theFakeCommercialObjectif.
87         Year__c = String.valueOf(
88             this.dateToday.year());
89     }
90
91     if(theFakeCommercialObjectif.
92         Month__c == NULL ||
93         theFakeCommercialObjectif.
94         Month__c.equals('')){
95         theFakeCommercialObjectif.
96         Month__c = VF001
97         _OpportunitiesByCommercial
98         .monthMap.get(this.
99         dateToday.month());
100     }
101 }
102
103 /** Rafraîchit les opportunités en
104 fonction du mois et de l'année **/
105 /
106 public PageReference
107 refreshOpportunities(){
108     try{
109         // Récupère le mois et l'année
110         des opportunités
111         this.setMonthAndYear();
112
113         // Récupère les objectifs des
114         commerciaux ainsi que
115         leurs opportunités
116         this.wrapperList = this.
117         getOpportunitiesByCommercial(
118             theFakeCommercialObjectif
119             .Month__c,
120             theFakeCommercialObjectif
121             .Year__c, this.
122             commercialList);
123     }catch(Exception e){
124         this.displayErrorMessage(e.
125             getMessage());
126     }
127
128     return NULL;
129 }
130
131 /** Met à jour les objectifs **/
132 public PageReference updateData(){
133     try{
134         update this.theObjectifList;
135
136         this.refreshOpportunities();
137     }catch(Exception e){
138         this.displayErrorMessage(e.
139             getMessage());
140     }
141
142     return NULL;
143 }
144
145 /** Crée la map des objectifs triés
146 par commercial **/
147 public map<Id, Commercial_objectif__c
148 > setObjectifMap(List<
149 Commercial_objectif__c>
150 theObjectifList){
151     map<Id, Commercial_objectif__c>
152     result = new map<Id,
153     Commercial_objectif__c>();
154
155     for(Commercial_objectif__c
156         anObjectif : theObjectifList)
157     {
158         result.put(anObjectif.User__c
159         , anObjectif);
160     }
161
162     return result;
163 }
164
165 /** Classe interne pour faciliter l'
166 affichage dans la page
167 Visualforce **/
168 public class WrapperClass{
169
170     public User theUser{get; set;}
171     public List<Opportunity>
172     theOpportunityList{get; set;}
173     public Commercial_objectif__c
174     theObjectif{get; set;}
175
176     public WrapperClass(User anUser,
177     List<Opportunity>
178     anOpportunityList,
179     Commercial_objectif__c
180     anObjectif){
181         this.theUser = anUser;
182         this.theOpportunityList =
183         anOpportunityList;
184         this.theObjectif = anObjectif
185         ;
186     }
187 }
188
189 /** Affiche un message d'erreur sur
190 la page Visualforce **/

```

```

145     public void displayErrorMessage(
146         String theErrorMessage){
147         ApexPages.addMessage(new
148             ApexPages.Message(ApexPages.
                Severity.Error,
                theErrorMessage));
147     }
148 }
  
```

Comme énoncé dans le chapitre des best prac-

tices :

- le nom du contrôleur Apex est le nom de la page Visualforce avec un « C » après le « VF » ;
- le code est commenté, mais surtout au niveau de chaque méthode pour me rappeler quel est son rôle dans le cas où je reviens dessus quelque temps après et que je ne m'en souviens plus.

7.3 La page Visualforce

Au niveau de la page Visualforce, nous affichons un menu déroulant permettant au manager de sélectionner un mois et une année (par défaut lors du premier chargement de la page, il s'agit du mois et de l'année actuels).

La modification du mois ou de l'année (après validation du formulaire) aura pour conséquence, le rechargement de la page et de son contenu en fonction du mois et de l'année :

```

1 <apex:page controller="VFC01_OpportunitiesByCommercial">
2
3     <apex:form >
4
5         <apex:pageBlock >
6
7             <apex:pageBlockButtons location="both">
8                 <apex:commandButton value="{!$Label.Labs_Sf_Valider}" action="{!
                    updateData}" />
9             </apex:pageBlockButtons>
10
11             <div>
12                 <apex:outputText value="{!$Label.Labs_Sf_Select_Month_Year} : " />
13                 <apex:inputField value="{!theFakeCommercialObjectif.Month__c}" />
14                 <apex:inputField value="{!theFakeCommercialObjectif.Year__c}" />
15                 <apex:commandButton value="{!$Label.Labs_Sf_Select_Opportunities}"
                    action="{!refreshOpportunities}" />
16             </div>
17
18             <apex:repeat value="{!wrapperList}" var="aWrapperClass">
19
20                 <apex:pageBlockSection title="{!aWrapperClass.theUser.Name}">
21
22                     <table>
23                         <tr>
24                             <td><apex:inputField value="{!aWrapperClass.theObjectif.
                                    Objectif__c}" style="width:100px;" /></td>
25                         </tr>
26                         <tr>
27                             <td><apex:outputField value="{!aWrapperClass.theObjectif.
                                    Percent_in_progress__c}" /></td>
28                         </tr>
29                     </table>
30
31                     <apex:pageBlockTable value="{!aWrapperClass.theOpportunityList}"
                        var="anOpportunity">
32
33                         <apex:column value="{!anOpportunity.Account.Name}" >
34                             <apex:facet name="header">{!$ObjectType.Account.Fields.Name
                                    .Label}</apex:facet>
35                         </apex:column>
36                         <apex:column value="{!anOpportunity.Name}" >
37                             <apex:facet name="header">{!$ObjectType.Opportunity.Fields.
                                    Name.Label}</apex:facet>
38                         </apex:column>
39                         <apex:column value="{!anOpportunity.Amount}" >
40                             <apex:facet name="header">{!$ObjectType.Opportunity.Fields.
                                    Amount.Label}</apex:facet>
41                         </apex:column>
42                         <apex:column value="{!anOpportunity.CloseDate}" >
43                             <apex:facet name="header">{!$ObjectType.Opportunity.Fields.
                                    CloseDate.Label}</apex:facet>
  
```

```

44         </apex:column>
45     </apex:pageBlockTable>
46
47     </apex:pageBlockSection>
48
49     </apex:repeat>
50
51 </apex:pageBlock>
52
53 </apex:form>
54
55 </apex:page>

```

Voilà visuellement à quoi elle ressemble :

The screenshot shows a Salesforce page with two sections. The first section, 'Addition des', has an 'Objectif' field set to 100,000 and 'Percent in progress' at 100.00%. It contains a table with three rows: Jones Smith Enterprise (\$15,000.00, 8/20/2015), Polar Mack Enterprises (\$75,000.00, 8/20/2015), and Je remplace ça (\$10,000.00, 8/18/2015). The second section, 'Changement de', has an 'Objectif' field set to 75,000 and 'Percent in progress' at 48.67%. It contains a table with two rows: Pepsi (\$20,000.00, 8/16/2015) and Sulfur (\$15,000.00, 8/12/2015). Both sections have an 'Update' button at the bottom.

7.4 Le trigger Apex

Le trigger Apex est utilisé lorsqu'une opportunité est créée ou modifiée.

Nous avons donc deux triggers Apex qui appellent tous deux la même méthode (comme vu dans les best practices pour externaliser le comportement) et qui se charge de mettre à jour le champ Objective in progress de l'objet Commercial objective en faisant la somme du champ Amount des opportunités qui lui sont attachées.

Voici les deux triggers Apex que je propose :
OpportunityAfterInsert

```

1 trigger OpportunityAfterInsert on
2   Opportunity (after insert) {
3
4     List<Opportunity>
5       opportunitiesToExecute = new
6       List<Opportunity>();
7
8     for(Opportunity anOpportunity :
9       Trigger.new){
10
11       if(anOpportunity.Amount != NULL)
12       {
13         opportunitiesToExecute.add(
14           anOpportunity);
15       }
16
17     if(opportunitiesToExecute.size() > 0)
18     {
19
20       // Mise à jour des objectifs des
21       commerciaux
22       AP01_UpdateObjectifInProgress.
23       updateObjectifInProgress(
24         oportunitiesToExecute);
25     }
26 }

```

17 }

OpportunityAfterUpdate

```

1 trigger OpportunityAfterUpdate on
2   Opportunity (after update) {
3
4     List<Opportunity>
5       opportunitiesToExecute = new
6       List<Opportunity>();
7
8     for(Opportunity anOpportunity :
9       Trigger.new){
10
11       if(anOpportunity.Amount != NULL)
12       {
13         oportunitiesToExecute.add(
14           anOpportunity);
15       }
16
17     if(opportunitiesToExecute.size() > 0)
18     {
19
20       // Mise à jour des objectifs des
21       commerciaux
22       AP01_UpdateObjectifInProgress.
23       updateObjectifInProgress(
24         oportunitiesToExecute);
25     }
26 }

```

Les deux triggers Apex sont déclenchés après la création et la mise à jour parce que nous avons obligatoirement besoin de leur identifiant pour récupérer toutes les opportunités d'un objectif et de faire leur somme.

Voici la méthode qui effectue le traitement :

```

1 public with sharing class AP01
2   _UpdateObjectifInProgress{
3
4   /** Met à jour les objectifs des
5     commerciaux **/
6   public static void
7     updateObjectifInProgress(List<
8       Opportunity> theOpportunityList)
9     {
10      Set<String>
11        commercialObjectifIdSet = new
12          Set<String>();
13
14      for(Opportunity anOpportunity :
15        theOpportunityList){
16
17        // Récupération de l'id de l'
18          objectif attaché à l'
19          opportunité
20        if(!commercialObjectifIdSet.
21          contains(anOpportunity.
22            Commercial_objectif__c)){
23          commercialObjectifIdSet.
24            add(anOpportunity.
25              Commercial_objectif__c
26            );
27        }
28      }
29
30      // Récupération des objectifs
31      ainsi que des opportunités
32      attachées
33      List<Commercial_objectif__c>
34        commercialsObjectifsList = [
35        SELECT Id, Year__c, Month__c,
36          Objectif__c,
37          Objectif_in_progress__c,
38          User__c, (SELECT Id,
39            Amount FROM
40            Opportunities__r WHERE
41            Amount != NULL)
42        FROM Commercial_objectif__c
43        WHERE Id IN :
44          commercialObjectifIdSet
45      ];
46
47      if(commercialsObjectifsList.size
48        () > 0){
49
50        // Pour chaque objectif
51        for(Commercial_objectif__c
52          aCommercialObjectif :
53          commercialsObjectifsList)
54        {
55          aCommercialObjectif.
56            Objectif_in_progress__c
57            = 0;
58
59          // Pour chaque opportunit
60          é attachée à l'
61          objectif
62          for(Opportunity
63            anOpportunity :
64            aCommercialObjectif.
65            Opportunities__r){
66
67            // Calcul du montant
68            de l'objectif en
69            cours
70            aCommercialObjectif.
71              Objectif_in_progress__c
72              += anOpportunity

```

```

34          . Amount;
35        }
36      }
37      update
38        commercialsObjectifsList;
39    }
40  }

```

En clair, je récupère dans un premier temps, les identifiants des objectifs des opportunités de la liste pour ensuite les récupérer via une requête SOQL ainsi que toutes leurs opportunités qui leur sont attachées (d'où l'intérêt que les triggers Apex (surtout celui de l'insertion) se déclenchent en after.

J'itère ensuite sur chaque objectif puis sur chaque opportunité pour calculer la somme de l'objectif en cours (via le champ Amount sur les opportunités).

La fin de la méthode est assez évidente, je mets à jour les objectifs avec leur nouvelle valeur.

7.5 Test du contrôleur Apex

Pour correctement tester le contrôleur Apex, il nous faut reconstruire un jeu de données identique à ce qu'il y aurait dans notre instance.

Nous avons donc besoin de :

- une liste de commerciaux ;
- une liste d'objectifs pour les commerciaux ;
- une liste d'opportunités attachées aux objectifs des commerciaux.

Voilà ce que ça peut donner :

```

1 @isTest
2 private class VFC01
3   _OpportunitiesByCommercial_TEST {
4
5   static Profile aProfile;
6   static User anUser;
7   static List<Commercial_objectif__c>
8     commercialsObjectifsList;
9   static List<Opportunity>
10     opportunityList;
11   static String stageName = '
12     Prospecting';
13   static String yearLabel = '
14     selectedYear';
15   static String monthLabel = '
16     selectedMonth';
17   static Decimal amount1 = 100;
18   static Decimal amount2 = 400;
19   static Decimal amount3 = 200;
20   static Decimal objectif1 = 1000;
21   static Decimal objectif2 = 1025;
22   static Decimal objectif3 = 7500;
23   static Date todayDate = Date.today()
24     ;
25   static map<Integer, String> monthMap
26     = VFC01
27     _OpportunitiesByCommercial.
28     monthMap;
29
30   static void init(){
31
32     /** Profile **/

```

```

23     aProfile = [
24         SELECT Id
25         FROM Profile
26         WHERE Name='Standard
           User'
27     ];
28
29     /** Utilisateur **/
30     anUser = new User(
31         Alias = 'standt',
32         Email='
           standarduser@testorg
           .com',
33         EmailEncodingKey='UTF-8
           ',
34         LastName='Testing',
35         LanguageLocaleKey='
           en_US',
36         LocaleSidKey='en_US',
37         ProfileId = aProfile.Id,
38         TimeZoneSidKey='America/
           Los_Angeles',
39         UserName='
           testUser@testTrailhead
           .com'
40     );
41
42     insert anUser;
43
44     /** Objectifs des commerciaux **
45     /
46     commercialsObjectifsList =
47         new List<
48         Commercial_objectif__c>
49         ();
50     commercialsObjectifsList.add
51     (new
52     Commercial_objectif__c(
53         Year__c = String.valueOf
54         (todayDate.year()),
55         Month__c = monthMap.get(
56         todayDate.month()),
57         Objectif__c = objectif1,
58         User__c = anUser.Id
59     ));
60     commercialsObjectifsList.add
61     (new
62     Commercial_objectif__c(
63         Year__c = String.valueOf
64         (todayDate.year()),
65         Month__c = monthMap.get(
66         todayDate.month() +
67         2),
68         Objectif__c = objectif2,
69         User__c = anUser.Id
70     ));
71     commercialsObjectifsList.add
72     (new
73     Commercial_objectif__c(
74         Year__c = String.valueOf
75         (todayDate.year()),
76         Month__c = monthMap.get(
77         todayDate.month() +
78         3),
79         Objectif__c = objectif3,
80         User__c = anUser.Id
81     ));
82     insert
83     commercialsObjectifsList
84     ;
85     /** Opportunités **/
86
87     opportunityList = new List<
88     Opportunity>();
89     opportunityList.add(new
90     Opportunity(
91         Name = 'Test opportunity
92         1',
93         CloseDate = Date.
94         newInstance(
95         todayDate.year(),
96         todayDate.month(), 1
97         ),
98         Amount = amount1,
99         Commercial_objectif__c =
100         commercialsObjectifsList
101         [0].Id,
102         StageName = stageName
103     ));
104     opportunityList.add(new
105     Opportunity(
106         Name = 'Test opportunity
107         2',
108         CloseDate = Date.
109         newInstance(
110         todayDate.year(),
111         todayDate.month(), 1
112         ),
113         Amount = amount1,
114         Commercial_objectif__c =
115         commercialsObjectifsList
116         [1].Id,
117         StageName = stageName
118     ));
119     opportunityList.add(new
120     Opportunity(
121         Name = 'Test opportunity
122         3',
123         CloseDate = Date.
124         newInstance(
125         todayDate.year(),
126         todayDate.month(), 1
127         ),
128         Amount = amount1,
129         Commercial_objectif__c =
130         commercialsObjectifsList
131         [2].Id,
132         StageName = stageName
133     ));
134     insert opportunityList;
135 }
136
137 /** Test de la page Visualforce **/
138 static testMethod void testVFPage() {
139     init();
140
141     Test.startTest();
142
143     // Utilisation de la page
144     Visualforce pour être dans le
145     bon contexte
146     PageReference pageRef = Page.VF01
147     _OpportunitiesByCommercial;
148
149     Test.setCurrentPage(pageRef);
150
151     VF01_OpportunitiesByCommercial
152     theController = new VF01
153     _OpportunitiesByCommercial();
154
155     ApexPages.currentPage().

```

```

    getParameters().put(yearLabel
    , String.valueOf(todayDate.
    addYears(1).year()));
108 ApexPages.currentPage().
    getParameters().put(
    monthLabel, monthMap.get(
    todayDate.month()));
109
110 // Rafraîchit les opportunités en
    fonction du mois et de l'ann
    ée sélectionnés
111 theController.
    refreshOpportunities();
112
113 // Met à jour les objectifs
    theController.updateData();
114
115 // Pour la couverture de code
    theController.displayErrorMessage
    ('Error !');
116
117 Test.stopTest();
118
119 }
120
121 }

```

7.6 Test des triggers Apex

Pareil que pour la classe de test du contrôleur Apex, nous avons besoin de reconstituer un environnement de données à tester.

Comme nous testons des opportunités, nous avons donc besoin d'opportunités, mais comme elles sont attachées à des objectifs, nous avons également besoin d'objectifs et de commerciaux.

Voici la liste de ce dont nous avons besoin :

- une liste de commerciaux;
- une liste d'objectifs des commerciaux;
- une liste d'opportunités attachées aux objectifs des commerciaux.

```

1 @isTest
2 private class AP01
3     _UpdateObjectifInProgress_TEST {
4
5     static List<Commercial_objectif__c>
        commercialsObjectifsList;
6     static List<Opportunity>
        opportunityList;
7     static String stageName = '
        Prospecting';
8     static Decimal amount1 = 100;
9     static Decimal amount2 = 400;
10    static Decimal amount3 = 200;
11    static Decimal objectif1 = 1000;
12    static Decimal objectif2 = 1025;
13    static Decimal objectif3 = 7500;
14    static Date todayDate = Date.today()
        ;
15    static map<Integer, String> monthMap
        = new map<Integer, String>{
16        1 => 'January',
17        2 => 'February',
18        3 => 'March',
19        4 => 'April',
20        5 => 'May',
21        6 => 'June',
22        7 => 'July',
23        8 => 'August',

```

```

23        9 => 'September',
24        10 => 'October',
25        11 => 'November',
26        12 => 'December'
27    };
28
29    static void init(){
30        /** Objectifs des commerciaux **
31        /
32        commercialsObjectifsList =
            new List<
33            Commercial_objectif__c>
34            ();
35        commercialsObjectifsList.add
            (new
36            Commercial_objectif__c(
37                Year__c = String.valueOf
                    (todayDate.year()),
38                Month__c = monthMap.get(
                    todayDate.month()),
39                Objectif__c = objectif1
                    ));
40        commercialsObjectifsList.add
            (new
41            Commercial_objectif__c(
42                Year__c = String.valueOf
                    (todayDate.year()),
43                Month__c = monthMap.get(
                    todayDate.month() +
44                    2),
45                Objectif__c = objectif2
                    ));
46        commercialsObjectifsList.add
            (new
47            Commercial_objectif__c(
48                Year__c = String.valueOf
                    (todayDate.year()),
49                Month__c = monthMap.get(
                    todayDate.month() +
50                    3),
51                Objectif__c = objectif3
                    ));
52        insert
53            commercialsObjectifsList
54            ;
55        /** Opportunités **/
56        opportunityList = new List<
            Opportunity>();
57        opportunityList.add(new
            Opportunity(
58            Name = 'Test opportunity
            1',
59            CloseDate = Date.
                newInstance(
60                todayDate.year(),
61                todayDate.month(), 1
                ),
            Amount = amount1,
            Commercial_objectif__c =
                commercialsObjectifsList
                [0].Id,
            StageName = stageName
            ));
62        opportunityList.add(new
            Opportunity(
63            Name = 'Test opportunity
            2',
64            CloseDate = Date.
                newInstance(
65                todayDate.year(),

```

```

        todayDate.month(), 1
    ),
62     Amount = amount1,
63     Commercial_objectif__c =
        commercialsObjectifsList
        [1].Id,
64     StageName = stageName
65 ));
66     opportunityList.add(new
67     Opportunity(
68         Name = 'Test opportunity
        3',
        CloseDate = Date.
        newInstance(
        todayDate.year(),
        todayDate.month(), 1
        ),
69         Amount = amount1,
70         Commercial_objectif__c =
        commercialsObjectifsList
        [2].Id,
71         StageName = stageName
72     ));
73 }
74
75 /** Test à la création */
76 static testMethod void testCreation()
77 {
78     init();
79     Test.startTest();
80     insert opportunityList;
81     Test.stopTest();
82
83     Commercial_objectif__c
84     theCommercialObjectif = [
85     SELECT Id,
86     Objectif_in_progress__c
87     FROM Commercial_objectif__c
88     WHERE Id = :
        commercialsObjectifsList[
        0].Id
89 ];
90
91     System.assertEquals(amount1,
        theCommercialObjectif.
        Objectif_in_progress__c);
92 }
93
94
95 /** Test à la mise à jour */
96 static testMethod void testUpdate() {
97     init();
98     insert opportunityList;
99     Test.startTest();
100
101     opportunityList[0].Amount =
        amount3;
102
103     opportunityList.add(new
104     Opportunity(
105         Name = 'Test opportunity 2',
106         CloseDate = Date.newInstance(
        todayDate.year(),
        todayDate.month(), 15),
107         Amount = amount2,
108         Commercial_objectif__c =
        commercialsObjectifsList
        [0].Id,
109         StageName = stageName
110     ));
111     upsert opportunityList;
112     Test.stopTest();
113
114     Commercial_objectif__c
115     theCommercialObjectif = [
116     SELECT Id,
117     Objectif_in_progress__c
118     FROM Commercial_objectif__c
119     WHERE Id = :
        commercialsObjectifsList[
        0].Id
120 ];
121
122     System.assertEquals((amount2 +
        amount3),
        theCommercialObjectif.
        Objectif_in_progress__c);
123 }
124 }

```

Pour vérifier le bon comportement des triggers Apex, je vérifie à la fin les valeurs des objectifs des commerciaux par rapport à la valeur qu'elles devraient avoir grâce à la méthode `System.assertEquals()`.

8 Plus si affinités

Comme je l'ai mentionné au début de cet article, je ne suis qu'un simple utilisateur du framework. Il y a plein de fonctionnalités que je n'utilise pas parce que je n'ai pas encore eu l'occasion ou que je n'ai

pas connaissance de ce qu'il est possible de faire.

Donc si vous êtes tombés amoureux du framework PAD, je vous invite à lire la documentation si vous désirez aller plus loin dans son utilisation.

9 Conclusion

Le framework PAD est une bonne solution pour mener correctement un projet, il impose des normes qu'il est conseillé de respecter lorsque l'on travaille

à plusieurs ou seul sur un projet.

Il vous permettra de gagner du temps lors de vos développements.

10 Ressources utiles

Pour obtenir le framework PAD, vous avez deux possibilités :

- envoyer un mail à contact en y saisissant le mot-clé « PAD » (dans l'objet du mail par exemple) pour recevoir la dernière version : [lien 22](#) ;
- télécharger cette archive pour recevoir la version 2.14 corrigée pour la version 31 de l'API de Salesforce.com.

Le code de cet article est disponible dans un dépôt sur Github accessible à tous : [Manage-commercials-opportunities \(lien 23\)](#).

Retrouvez l'article d'**Aurélien Laval** en ligne : [lien 24](#)

Qt



Les derniers tutoriels et articles

Sortie de Qt 5.6 Alpha

Qt 5.6 vient de sortir en préversion Alpha. Les nouveautés s'égrainent le long des divers modules, avec un grand nombre d'optimisations, notamment au niveau de la mémoire utilisée. Elles sont en grande partie dues à l'utilisation d'analyse statique du code proposée par Clang : utiliser un profileur pour toutes les exécutions possibles dans le code prendrait trop de temps pour un gain trop faible ; par contre, un analyseur statique peut passer sur tout le code et faire remarquer tous les endroits à améliorer. Il est donc très utile quand le problème de performance n'est pas localisé, mais bien réparti sur tout le code. Ce genre d'outil a une compréhension plus fine du C++ qu'un compilateur : là où le compilateur s'arrête à la syntaxe, l'analyseur statique tente de comprendre la sémantique du code (ce qu'il tente de faire) et propose de meilleures manières de l'écrire. Par exemple, un algorithme de la STL pourrait fonctionner plus vite en allouant à l'avance de la mémoire avec `std::vector::reserve()` ; il pourrait même réécrire automatiquement le code pour suivre les meilleures pratiques concernant `QStringLiteral` et `QLatin1String`.

Qt 3D n'est toujours pas finalisé, mais les fonctionnalités de la préversion technologique ([lien 25](#)) s'affinent : de nouvelles API pour les tampons et attributs, pour les entrées souris, pour la détection de collisions ; la gestion de l'instanciation, le chargement de scènes glTF (avec un outil pour les compiler : `qgltf`). De nouveaux exemples ont également été ajoutés.

Dans les modules, quelques changements ont eu lieu. Notamment, des modules désapprouvés par Qt 5.5 ([lien 26](#)) ont été supprimés (Qt WebKit et Qt Declarative, qui correspond à Qt Quick 1). De nouveaux modules sont maintenant déconseillés : Qt Script (remplacé par le moteur JavaScript V4, inclus dans Qt QML) et Qt Enginio ; ils pourraient être supprimés dans une prochaine version de Qt. De nouveaux modules font également leur apparition en tant que préversions technologiques : Qt Quick Controls 2.0 ([lien 27](#)), des contrôles plus légers, prévus plus particulièrement pour l'embarqué ; Qt Speech pour la reconnaissance vocale ([lien 28](#)), en faisant appel aux API disponibles sur chaque système (le module est actuellement compatible avec Android, OS X et Windows) ; Qt SerialBus ([lien 29](#)), pour faciliter l'accès aux nombreux bus série qui peuplent le monde de l'embarqué, actuellement compatible uniquement avec CAN ; Qt Wayland ([lien 30](#)), avec une API pour le compositeur Wayland stabilisée.

Cependant, la principale avancée concerne les outils utilisés pour le développement de Qt lui-même : l'ancien moteur d'intégration continue, Jenkins, a été remplacé par un nouveau, Coin, développé exclusivement pour Qt (qui pourrait d'ailleurs être distribué sous licence propriétaire ou libre dans le futur). Ainsi, chaque modification apportée au code source de Qt est compilée (voire testée) sur vingt-cinq à trente plateformes, ce qui assure une plus grande stabilité du code.

*Commentez la news de **Thibaut Cuvelier** en ligne : [lien 31](#)*

Libres & Open Source



Les derniers tutoriels et articles

Scratch : création d'un mini-jeu

Le but du jeu sera simple : « Un héros doit protéger un gentil d'un méchant qui veut l'attaquer ». Nous allons apprendre à déplacer notre personnage le « Héros », et faire se déplacer plus ou moins intelligemment les personnages non joueurs (PNJ) de notre jeu vidéo.

1 Introduction

Scratch : c'est quoi ? Un langage visuel et coloré en français pour apprendre les bases de la programmation.



Apprendre à programmer, c'est aussi apprendre à se poser des questions et être créatif.

C'est pour cela qu'à chaque étape de ce tutoriel, vous retrouvez les questions du QQCOQP (lien 32) : Qui, Quoi, Comment, etc. parce qu'apprendre à programmer, c'est aussi apprendre à se poser des ques-

tions!



Pour savoir quoi faire avant de savoir comment faire, je vous invite à ouvrir ce PDF (lien 33), il contient tous les éléments pour la création de votre jeu.

Le rôle de notre mini-jeu va se baser sur le principe de trois personnages, et d'une histoire bien connue ; celle du « méchant » qui veut attaquer le « gentil », mais il y a un « héros » pour le défendre.

2 Création du héros

Qui : nous le nommerons « Le Héros » pour ce tutoriel, « *nous allons faire simple* ».

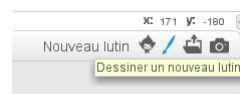
Quoi : à quoi cela va-t-il ressembler ?

Nous allons commencer en créant un nouvel objet ou lutin :

Version 1.4

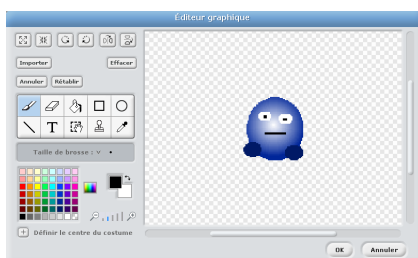


Version 2

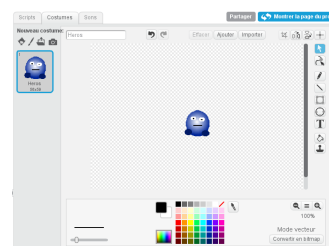


Cela nous ouvre l'éditeur graphique :

Version 1.4



Version 2





Dessignons notre personnage comme celui montré ci-dessus ou créez le vôtre.

Comment va-t-il agir (personnage) ou fonctionner (objet)? Il va se déplacer avec les curseurs (flèches du clavier) :

Version 1.4



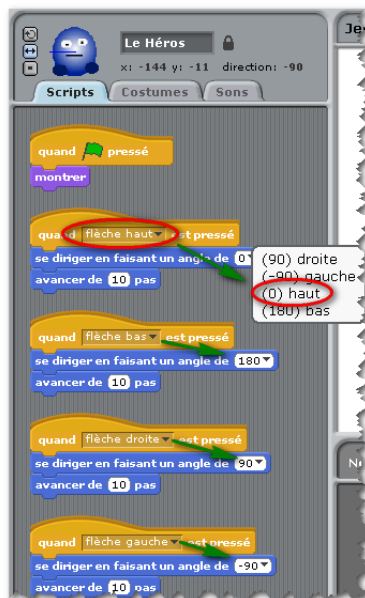
Version 2



Nous devons aller chercher le mouvement qui se trouve dans « Evènements » qui nous permet de définir une action, ensuite il faut associer à ce mouvement l'orientation et le déplacement.

Voici la liste complète des mouvements avec les paramètres à mettre :

Version 1.4



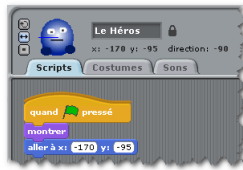
Version 2



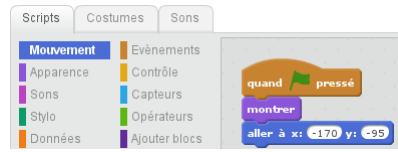
Nous devons rajouter les quatre mouvements possibles, en répétant l'opération précédente.

Où va-t-il intervenir dans le jeu ?

Version 1.4



Version 2



Une fois que nous allons lancer le programme, celui-ci va apparaître à une position sur l'écran. Nous avons choisi de le positionner dans la partie basse de la gauche.

Quand va-t-il intervenir dans le jeu ?

Version 1.4



Version 2



Pourquoi : à quoi cela va-t-il servir dans l'histoire? C'est celui qui va attaquer le « méchant » et sauver le « gentil ».

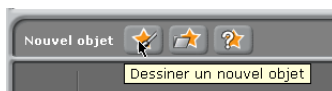
3 Création du Gentil

Qui : nous le nommerons « Le gentil ». C'est un personnage non joueur (PNJ).

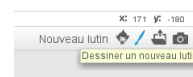
Quoi : à quoi cela va-t-il ressembler ?

Créons un nouvel objet :

Version 1.4

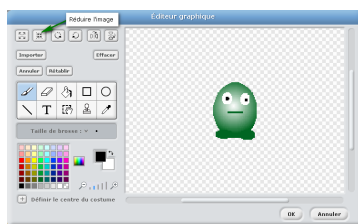


Version 2

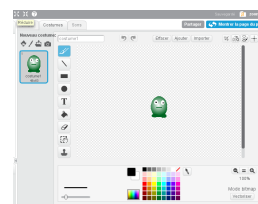


Cela nous ouvre l'éditeur graphique :

Version 1.4



Version 2



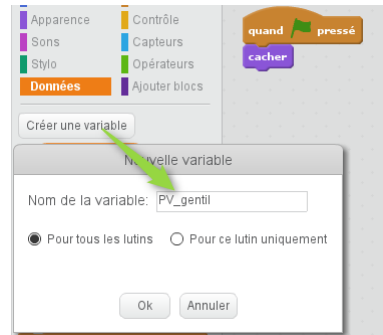
Dessignons notre personnage comme celui montré ci-dessus ou créez le vôtre.

Comment va-t-il agir (personnage) ou fonctionner (objet)? Il va se déplacer aléatoirement en évitant le « héros ». Et une autre indication, c'est qu'il va se déplacer tant que ses points de vie sont supérieurs à 0.

Version 1.4

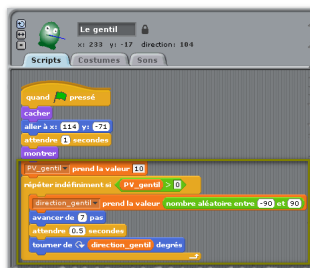


Version 2

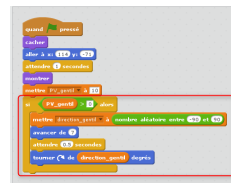


Nous allons devoir créer une variable pour que le personnage puisse avoir un état de vie.

Version 1.4



Version 2



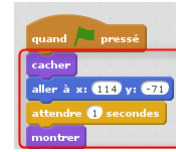
Maintenant, rajoutons cette notion de vie dans le personnage, nous initialisons la variable avec une valeur ici 10. Mettons en place une boucle pour que le test se réalise sur son état de vie. Dans la boucle nous allons rajouter un déplacement aléatoire, il faudra donc créer une autre variable « direction_gentil » qui permettra au « gentil » de s'orienter, et ensuite nous le faisons avancer. Nous avons dans l'encadré les différents éléments de la boucle. Mais vous pouvez l'adapter à votre cas, en modifiant les données ou en ajoutant.

Où et Quand va-t-il intervenir dans le jeu ?

Version 1.4



Version 2



Une fois que nous allons lancer le programme, celui-ci va apparaître à une position sur l'écran. Nous avons choisi de le positionner dans la partie basse de la droite, mais il n'apparaîtra qu'une seconde après le lancement du programme.

Pourquoi : à quoi cela va-t-il servir dans l'histoire ? Il doit être protégé du « méchant ».

4 Création du méchant

Qui : nous le nommerons « Le méchant », c'est un personnage non joueur (PNJ).

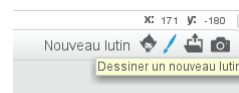
Quoi : à quoi cela va-t-il ressembler ?

Créons un nouvel objet :

Version 1.4

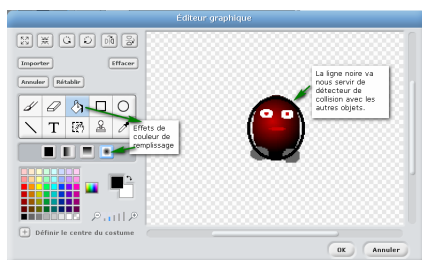


Version 2

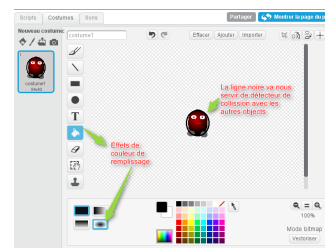


Cela nous ouvre l'éditeur graphique :

Version 1.4



Version 2



Dessignons notre personnage comme celui montré ci-dessus ou créez le vôtre. Nous créons un contour noir, il nous servira tout à l'heure pour détecter les collisions entre les différents personnages.

Comment va-t-il agir (personnage) ou fonctionner (objet) ? Il va se diriger vers le « héros » ou le « gentil » pour l'attaquer. La ligne noire va nous servir de détecteur de collisions avec les autres objets. Et une autre indication, c'est qu'il va se déplacer tant que ses points de vie sont supérieurs à 0.

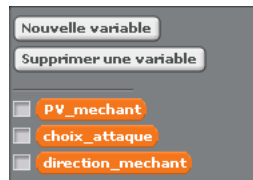
5 Création des variables



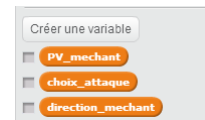
En informatique, une variable permet d'associer un nom et une valeur. Cette valeur peut être alphanumérique, numérique, etc. Dans la plupart des cas, les variables sont numériques, elles servent ainsi de valeur à atteindre ou à évoluer dans le temps.

Pour notre cas, voici les variables dont nous aurons besoin :

Version 1.4

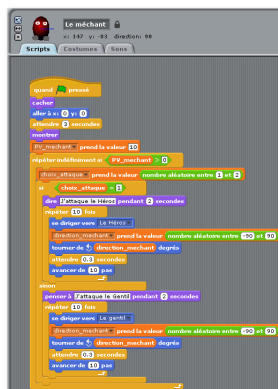


Version 2

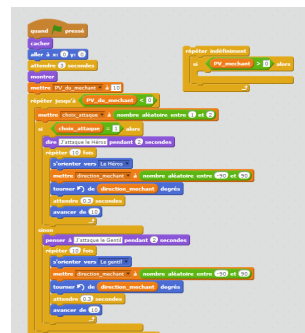


- *PV_mechant* pour stocker les points de vie du méchant ;
- *Choix_attaque* : variable aléatoire pour décider si le méchant attaque le gentil ou le héros ;
- *Direction_mechant* : variable aléatoire pour parasiter le trajet du méchant :

Version 1.4



Version 2



Où et quand va-t-il intervenir dans le jeu ?

Version 1.4



Version 2

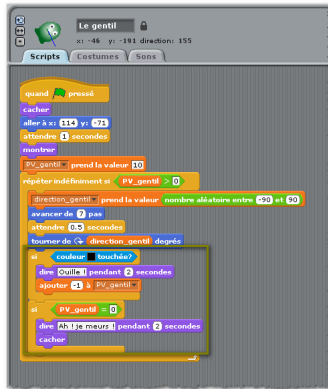


Une fois que nous allons lancer le programme, celui-ci va apparaître à une position sur l'écran. Nous avons choisi de le positionner au centre, mais il n'apparaîtra que trois secondes après le lancement du programme.

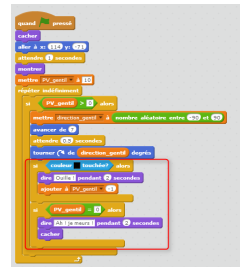
Pourquoi : à quoi cela va-t-il servir dans l'histoire ?

Rappel du principe du jeu : le rôle du « méchant » est d'attaquer le « héros » ou le « gentil » et de lui faire perdre des points de vie. Son action a un impact sur les autres « lutins », c'est-à-dire leur faire perdre des points. Nous allons donc modifier le script du « gentil » lutin :

Version 1.4



Version 2



Rajoutons maintenant une boucle qui permet de mettre à jour l'état de vie du « Gentil ». Si le gentil lutin touche le noir du méchant, il dit « Ouille! » et perd un point de vie. Lorsque les points de vie du gentil lutin arrivent à zéro : il meurt et disparaît avec « cacher ».

6 Amélioration du jeu

Revenons au « Héros » :



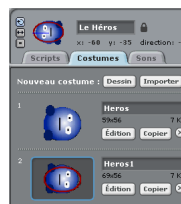
Pour l'instant il ne fait que se déplacer. Il n'y a aucune interaction avec les autres personnages.

Rappel du *Pourquoi* : à quoi cela va-t-il servir dans l'histoire ? C'est celui qui va attaquer le « méchant » et sauver le « gentil ».

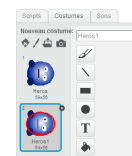
Attaquer le méchant = lui faire perdre des points de vie (PV).

Comment : avec une arme qui apparaît avec la barre d'espace. On lui dessine une arme, nous prendrons alors une auréole rouge qui va servir de détecteur de collision :

Version 1.4



Version 2



Rajoutons un nouveau costume au « Héros » qui nous servira d'arme, dans notre cas une ellipse rouge. Vous pouvez lui rajouter une épée, il vous faudra alors adapter le programme à la nouvelle arme, elle devra avoir une couleur distincte.

Nous allons aussi le rajouter dans un script :

Version 1.4

```

quand espace est pressé
  basculer sur le costume Héros1
  si couleur touche ?
    ajouter -1 à PV_du_mechant
  attendre 1 secondes
  costume suivant
  
```

Version 2

```

quand espace est pressé
  basculer sur costume Héros1
  si couleur touche ? alors
    ajouter à PV_du_mechant -1
  attendre 1 secondes
  costume suivant
  
```



Rajoutons maintenant le fait que si l'arme du « Héros » entre en contact avec l'ellipse noire du « Méchant », et décrémentation la variable « PV_mechant ».

Cela a un impact sur le méchant : il perd des PV et meurt (s'il atteint la valeur 0), ce qui implique la fin du jeu :

Version 1.4

```

quand pressé
  cacher
  aller à x:0 y:0
  attendre 1 secondes
  montrer
  PV_du_mechant prend la valeur 10
  répéter jusqu'à PV_du_mechant < 1
    choix_attaque prend la valeur nombre aléatoire entre 1 et 2
    choix_attaque = 1
    dire attaque le héros pendant 2 secondes
    répéter 10 fois
      se diriger vers Le Héros
      direction_mechant prend la valeur nombre aléatoire entre 90 et 180
      tourner de 45 direction_mechant degrés
      attendre 0.5 secondes
      avancer de 10 pas
    sinon
      passer à l'attaque le client pendant 2 secondes
      répéter 10 fois
        se diriger vers Le méchant
        direction_mechant prend la valeur nombre aléatoire entre 90 et 180
        tourner de 45 direction_mechant degrés
        attendre 0.5 secondes
        avancer de 10 pas
      dire Ah! Tu as gagné! pendant 2 secondes
      cacher
      envoyer à tous Fin_du_jeu
  
```

Version 2

```

quand pressé
  cacher
  aller à x:0 y:0
  attendre 1 secondes
  montrer
  PV_du_mechant = 10
  répéter jusqu'à PV_du_mechant < 1
    choix_attaque = 1
    dire attaque le héros pendant 2 secondes
    répéter 10 fois
      se diriger vers Le Héros
      direction_mechant prend la valeur nombre aléatoire entre 90 et 180
      tourner de 45 direction_mechant degrés
      attendre 0.5 secondes
      avancer de 10 pas
    sinon
      passer à l'attaque le client pendant 2 secondes
      répéter 10 fois
        se diriger vers Le méchant
        direction_mechant prend la valeur nombre aléatoire entre 90 et 180
        tourner de 45 direction_mechant degrés
        attendre 0.5 secondes
        avancer de 10 pas
      dire Ah! Tu as gagné! pendant 2 secondes
      cacher
      envoyer à tous Fin_du_jeu
  
```



Rajoutons maintenant cette nouvelle contrainte dans le programme ainsi qu'un petit message qui indique que le jeu est fini.



Nous avons changé la boucle **répéter indéfiniment si** par **répéter jusqu'à**. À la fin, nous envoyons à tous le message « Fin_du_jeu » pour indiquer que le jeu est terminé.

Retrouvez l'article de **Christophe Thomas** et **Vincent Viale** en ligne : [lien 34](#)

Scratch : Thésée et le Minotaure

Le but de ce tutoriel est de vous montrer pas à pas la réalisation du jeu « Thésée et le Minotaure ».

1 Introduction

Scratch est un nouveau langage de programmation qui facilite la création d'histoires interactives, de dessins animés, de jeux, de compositions musicales, de simulations numériques, etc. et leur partage sur le Web.

Il est conçu pour initier les enfants, âgés de 8 ans et plus, à des concepts importants en mathématiques et informatique, tout en apprenant à développer une pensée créative, un raisonnement systématique et à travailler en équipe.

2 Installer et découvrir Scratch

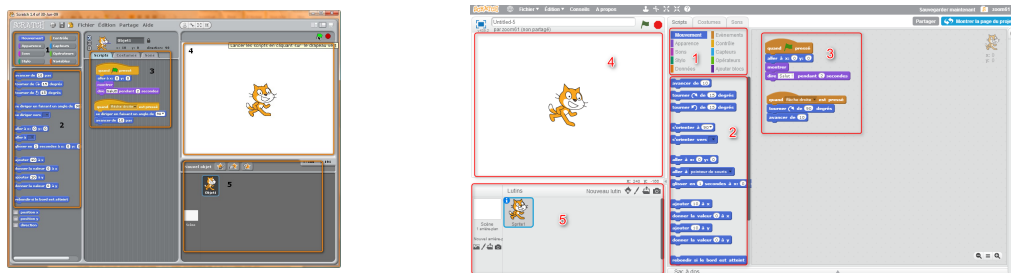
Télécharger SCRATCH à l'adresse suivante : [lien 35](#).

Il est disponible pour Mac OS X, Windows et pour Linux.

L'aide standard est en anglais. Il vous faut télécharger l'aide en français à l'adresse suivante : [lien 36](#).

Ensuite vous ouvrez Scratch. Vous obtenez l'écran suivant :

Figure 1 : interface de programmation



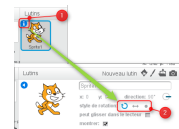
- La **zone 1** est la palette qui vous permet de choisir la catégorie de blocs.
- Dans la **zone 2** apparaissent les blocs de la catégorie choisie.
- Vous déplacez vos blocs dans la **zone 3** pour assembler vos scripts.
- La **zone 4** affiche le résultat de votre programme.
- Dans la **zone 5** s'affichent tous les objets graphiques que vous utilisez.

Automatiquement au démarrage la mascotte Scratch apparaît comme « lutin » ou « sprite » par défaut. Essayez le script suivant en recherchant les blocs correspondants :

Figure 2 : notre premier script



Attention à ces trois petits boutons :



Ils définissent l'orientation automatique de votre lutin.

Cliquez sur celui du milieu.

Cliquez sur le drapeau vert et appuyez sur les touches de déplacement du curseur de votre clavier.

Voilà! le résultat est immédiat.

Je vous propose maintenant un jeu à réaliser : « Thésée et le Minotaure ».

Le cycle de développement d'un programme SCRATCH est itératif, comme le montre la figure ci-dessous :

Figure 3 : cycle de développement SCRATCH



Les étapes proposées ci-dessous sont la première itération de développement de notre jeu. Vous pourrez refaire une itération pour améliorer les différents aspects du jeu.

3 Créer son premier jeu vidéo avec Scratch

3.1 Le but du jeu

Il s'agira d'aider Thésée à traverser le labyrinthe en évitant le Minotaure, en trouvant le trésor puis la sortie.

Ce jeu simple doit permettre de couvrir les principes élémentaires de programmation d'un jeu :

Figure 4 : ce que vous allez apprendre avec ce tutoriel



3.2 Dessiner les différents éléments

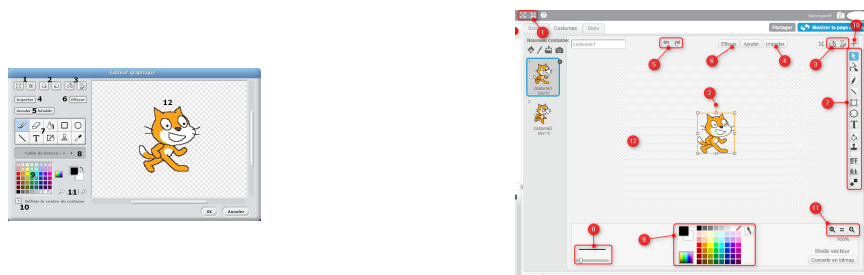
Il faudra dessiner les personnages (Thésée, le Minotaure), les objets (le trésor), les décors (le « générique/intro », le labyrinthe).












Scratch est pourvu d'un outil de dessin intégré qui couvre la plupart des besoins :

Figure 5 : accéder à l'outil de dessin



Figure 6 : l'outil de dessin Scratch



- **Zone 1**  : agrandir ou réduire votre dessin.
- **Zone 2**  : effectuer une rotation à votre dessin.
- **Zone 3**  : retourner votre dessin.
- **Zone 4**  : importer une image.
- **Zone 5**  : annuler ou rétablir une action.
- **Zone 6**  : effacer votre dessin.
- **Zone 7**  : palette d'outil de dessin.
- **Zone 8**  : taille du crayon à dessin.
- **Zone 9**  : palette de couleurs.
- **Zone 10**  : définir l'axe de rotation de votre dessin.
- **Zone 11**  : zoom.
- **Zone 12** : zone de dessin.

3.3 Créer et dessiner Thésée

Nous allons reprendre la mascotte Scratch et lui ajouter un casque d'hoplite en définissant le centre du costume :

Figure 7 : dessiner Thésée



3.4 Créer le personnage du Minotaure

Pour aller vite, nous allons importer un lutin de la bibliothèque fournie avec Scratch en cliquant sur le dossier avec l'étoile :

Figure 8 : insérer un nouveau lutin



Nous accédons à la bibliothèque d'objets graphiques et nous choisissons celui qui se rapproche le plus du Minotaure dans le répertoire /Costumes/Fantasy :

Figure 9 : choisir le Minotaure



3.5 Dessiner le labyrinthe

Cliquez sur scène :

Figure 10 : choisir Scène



L'outil de dessin pour la scène est le même que pour les autres objets :

Figure 11 : dessiner l'arrière-plan labyrinthe



Le point vert symbolise l'arrivée.

Après avoir réalisé notre labyrinthe, nous nous apercevons que nos personnages sont trop gros par rapport au décor :

Figure 12 : résultat intermédiaire




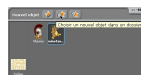
Il faut donc les réduire. Pour cela, nous allons utiliser le bloc  qui se trouve dans le groupe « Apparence » :

Figure 13 : script de réduction des lutins



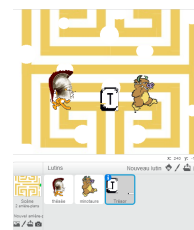
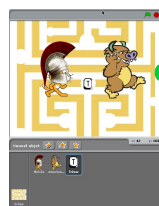
Nous allons maintenant ajouter le trésor :

Figure 14 : insérer le trésor en choisissant un nouvel objet



Choisir un nouvel objet dans la bibliothèque symbolisant le trésor dans le répertoire /Costumes/letters. Pour nous, ce sera la lettre « T » :

Figure 15 : les éléments de notre jeu



Nous avons maintenant tous les éléments de notre jeu : Thésée, le Minotaure, le trésor et le labyrinthe.

3.6 Déplacer son personnage (Thésée)

Les directions dans Scratch sont les suivantes :

Figure 16 : les angles de direction



Allez dans le groupe de blocs jaunes « Contrôle » et choisissez « *Quand Espace est pressé* ».
Allez dans le groupe de blocs bleus « Mouvements » et choisissez « *se diriger en faisant un angle puis avancer de 5 pas* ».

Ce qui nous donne le script suivant pour déplacer notre lutin :

Figure 17 : script de déplacement



3.7 Détecter les obstacles (murs du labyrinthe, sortie) avec les capteurs

Les murs du labyrinthe sont jaunes. Il faut que si notre personnage touche un mur jaune il recule.

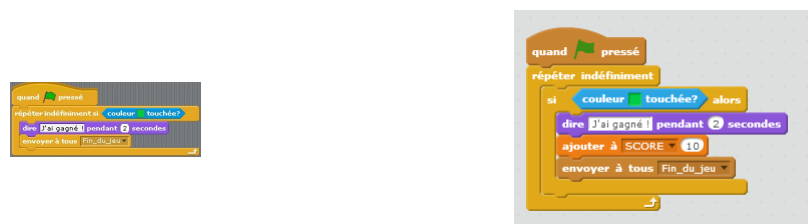
Vous savez maintenant où trouver les blocs jaunes de « contrôle ». Les capteurs sont dans le groupe bleu clair. Choisissez le capteur qui convient le mieux : « *couleur touchée* » dans notre cas. Lorsque vous cliquez sur le carré de couleur jaune du bloc « *couleur touchée?* », une pipette apparaît à la place du curseur et vous cliquez ensuite sur la couleur à détecter :

Figure 18 : détecter le mur jaune



S'il atteint la sortie (de couleur verte), c'est la fin du jeu. Il faut donc le signaler en « envoyant un message » :

Figure 19 : sortie atteinte



Nous verrons plus loin comment utiliser les messages avec « *Trésor_trouvé* ».

3.8 Faire apparaître le trésor aléatoirement dans le labyrinthe

Dans le groupe « Mouvement » choisir « *Aller à x : y :* », puis dans « Opérateurs » choisir « *nombre aléatoire entre _ et _* ». Glissez et déposez ce bloc comme dans la figure ci-dessous :

Figure 20 : apparition aléatoire du trésor



3.9 Détecter les collisions/interactions avec les capteurs et incrémenter la variable SCORE

Qui dit jeu, dit SCORE. Nous allons donc créer une variable SCORE :

Figure 21 : variable SCORE



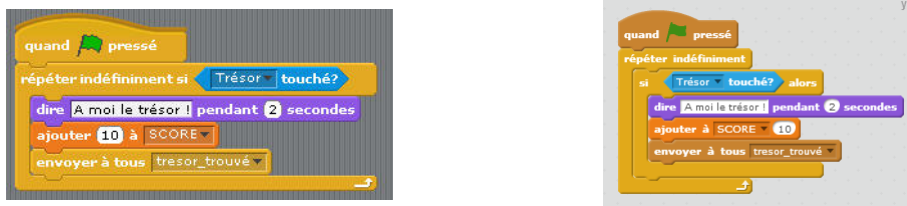
Lorsque nous créons une variable, de nouveaux blocs apparaissent :

Figure 22 : blocs de gestion des variables



3.10 Thésée a-t-il trouvé le trésor ?

Figure 23 : 10 points si le trésor est trouvé



Si le trésor est trouvé et que Thésée l'emporte, il doit disparaître avec le bloc « cacher » :

Figure 24 : le trésor est emporté par Thésée



3.11 Déplacement pseudo-aléatoire du Minotaure

Le déplacement des personnages non joueurs (PNJ) est la fonction la plus difficile à programmer. C'est elle qui fait la difficulté du jeu. Difficulté tant du côté du programmeur que du joueur. C'est pour cela que nous avons choisi une option relativement simple. Ici, nous allons utiliser une variable pour déterminer une direction aléatoire :

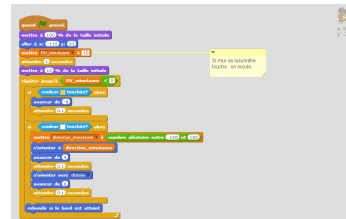
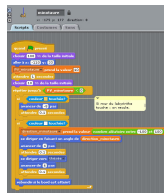
Figure 25 : créer la variable de direction aléatoire



Dans « Variables » vous choisissez le bloc *prend la valeur*. Dans le groupe « Opérateur » vous choisissez *nombre aléatoire entre __ et __* pour obtenir le script suivant.

Voici ensuite un algorithme possible pour déplacer votre Minotaure :

Figure 26 : exemple de déplacement aléatoire du Minotaure



3.12 Fin de la première itération

Voilà, nous avons fini notre première itération. Nous avons un labyrinthe dans lequel le joueur déplace le personnage Thésée. Si Thésée trouve le trésor, il marque 10 points. S’il arrive à la sortie, il a terminé le jeu. Nous avons utilisé six groupes de blocs d’instructions sur les huit disponibles.

Dans vos itérations suivantes, vous pourrez :

- déterminer si Thésée doit tuer le Minotaure et avec quoi (une épée, une boule de feu...);
- ajouter un générique de début et de fin;
- améliorer le déplacement du Minotaure.

Amusez-vous bien ! Retrouvez l’article de *Christophe Thomas et Vincent Viale* en ligne : [lien 37](#)

Perl



Les derniers tutoriels et articles

De Perl 5 à Perl 6 - annexe 1

Ce qui change entre Perl 5 et Perl 6

Ce document fait suite à une série de trois articles décrivant les principaux changements entre la version 5 de Perl, une vénérable dame qui a commencé sa carrière il y a plus de 20 ans (en 1994), et la nouvelle mouture, Perl 6, radicalement nouvelle et bien plus moderne et plus expressive, qui devrait sortir en version de production avant la fin de l'année 2015.

Cette série d'articles décrivait en détail de nombreuses différences entre les deux versions du langage et de nombreuses nouveautés de Perl 6.

La présente annexe 1 vise à résumer succinctement les différences syntaxiques et sémantiques afin de constituer une référence de poche (incomplète), une sorte d'« antisèche » permettant au lecteur de retrouver rapidement un élément de syntaxe qui lui échapperait.

L'annexe 2 examinera les nouveautés du langage, et elle est à notre humble avis bien plus importante pour le lecteur qui désire apprendre cette nouvelle version du langage.

1 Introduction

La présente annexe 1 du tutoriel *De Perl 5 à Perl 6* décrit les changements qu'il va falloir apporter à un programme Perl 5 pour qu'il fonctionne en Perl 6. Il peut donc servir de guide de traduction (assez complet, mais non exhaustif) de Perl 5 en Perl 6.

Le but est de permettre au lecteur de « traduire » un programme Perl 5 en Perl 6 (bien que ce ne soit pas toujours une bonne idée, il vaut souvent mieux réécrire, mais ne soyons pas dogmatiques sur ce point, les situations réelles ne sont pas toujours si simples). Tout au moins, nous espérons être utile au lecteur qui se dit : « Ah, zut, ça, je sais le faire en Perl 5, comment faire en Perl 6 ? »

Les sources employées pour la rédaction de ce document sont multiples : outre le contenu du tutoriel dont nous reprenons quelques parties (il y aura quelques redites, mais très peu, espérons-nous, il y aura surtout beaucoup de points de détail nouveaux), nous avons abondamment utilisé la documentation officielle Perl 6 en anglais (lien 38), qui s'est considérablement étoffée depuis la parution de la première partie du tutoriel (il existe encore quelques lacunes, que nous signalerons à l'occasion, mais elle est maintenant assez complète). Certains passages sont des adaptations en français, voire occasionnellement de simples traductions, de la documentation officielle, mais celle-ci étant anonyme, je ne peux rendre à ses auteurs l'hommage qu'ils mé-

riteraient sinon.

Support de l'Unicode en Perl 6

Lorsque nous avons rédigé les trois parties du tutoriel (juin à décembre 2014), nous avons signalé que la version de *Rakudo Star* disponible à l'époque ne supportait pas l'Unicode, si bien que tous nos exemples évitaient soigneusement les lettres françaises accentuées, trémas, cédilles, guillemets français, etc., même dans les chaînes de caractères ou les commentaires.

La situation a bien évolué depuis, et Rakudo / Perl 6 supporte maintenant (août 2015, et probablement depuis au moins mars 2015) des caractères non ASCII (caractères accentués, signes diacritiques, symboles divers, émoticônes (*smileys*), lettres étrangères, etc.) aussi bien dans les chaînes de caractères et les commentaires que dans les identifiants de variables, de méthodes ou de fonctions, et ce, sans la moindre déclaration préalable.

Le lecteur pourra constater que de nombreux exemples de ces annexes illustrent cette faculté.



2 Syntaxe générale

2.1 Commentaires

```

1 use v6; # utilisation de Perl 6
2
3 # ceci est un commentaire (même chose qu
  'en Perl 5)
4
5 #\{ Ceci
6 est un commentaire multiligne.
7 Ce commentaire est ouvert avec une
  accolade ouvrante.
8 Il faut le fermer avec une accolade
  fermante.
9 Si on l'avait ouvert avec un crochet
  ouvrant,
10 il faudrait le fermer avec un
  crochet fermant. }
11
12
13 my $pi #\{{{
14 Commentaire multiligne utilisant
  plusieurs accolades (ou crochets)
15 → Ce commentaire est ouvert avec trois
  accolades ouvrantes.
16 → Il faut le fermer avec trois accolades
  fermantes }.
17 }}} = 3.14159;
18 say "Pi is: $pi"; # Affiche : 3.14159

```

Voir aussi le chapitre Commentaires de la première partie du tutoriel : [lien 39](#).

2.2 Appels de méthode

Pour les appels de méthodes, la flèche « -> » est remplacée par le point « . » (la concaténation, qui utilisait le point, est maintenant faite avec le tilde « ~ »).

```

1 $personne->nom # Perl 5
2 $personne.nom # Perl 6
3
4 # Appel d'une méthode dont le nom n'est
  connu qu'à l'exécution:
5
6 $objet->$methodname(@args); # Perl 5
7 $objet."$methodname"(@args); # Perl 6

```

2.3 Espaces blancs

Perl 5 est très laxiste sur les espaces blancs : on peut en mettre ou ne pas en mettre presque partout.

Perl 6 ne veut pas limiter la créativité du programmeur, mais le besoin de pouvoir établir une grammaire cohérente, déterministe et extensible travaillant en une seule passe et fournissant des messages d'erreurs qui soient une réelle aide au programmeur a conduit à adopter un compromis. Il en résulte qu'il y a quelques endroits où les espaces blancs sont obligatoires et d'autres où ils sont proscrits.

Pas d'espace autorisé avant la parenthèse ouvrant une liste d'arguments

```

1 substr ($s, 4, 1); # Perl 5 (en Perl 6
  ce code essaierait de passer un
2 # seul
  argument de type
  List à substr)

```

```

3 substr($s, 4, 1); # Perl 6
4 substr $s, 4, 1; # Perl 6 - le +
  simple est d'omettre les parenthèses

```

Espace obligatoire après les mots-clefs

```

1 my($alpha, $bravo); # Perl 5. En
  Perl 6, essaie d'appeler
2 # la
  fonction
  my()
3
4 my ($alpha, $bravo); # Perl 6
5
6 if($a < 0) { ... } # Perl 5,
  erreur en Perl 6
7 if ($a < 0) { ... } # Perl 6
8 if $a < 0 { ... } # Perl 6,
  plus idiomatique sans parenthèses
9
10 while($x-- > 5) { ... } # Perl 5,
  erreur en Perl 6
11 while ($x-- > 5) { ... } # Perl 6
12 while $x-- > 5 { ... } # Perl 6,
  plus idiomatique sans parenthèses

```

Pas d'espace autorisé après un opérateur préfixé ou avant un opérateur postfixé ou postcircrconfixé (y compris pour indices des tableaux et des hachages)

```

1 $seen {$_} ++; # Perl 5
2 %seen{$_}++; # Perl 6

```

À noter cependant que l'on peut utiliser l'opérateur *unspace* « \ » pour ajouter des espaces (et même des commentaires) presque n'importe où :

```

1 # Perl 5
2 my @books = $xml->parse_file($file)
  # commentaire quelconque
  ->findnodes("/library/
  book");
3
4 # Perl 6
5 my @books = $xml.parse-file($file)\
  # commentaire quelconque
  .findnodes("/library/
  book");
7

```

2.4 Démêler le vrai du faux

Perl 5 et Perl 6 ont à peu près la même notion du vrai et du faux (les nombres 0, 0.0 et la chaîne vide sont faux, presque tout le reste est vrai), à cette différence près que, contrairement à Perl 5, Perl 6 considère la chaîne "0" comme vraie.

Perl 6 possède un type booléen (Bool) définissant des valeurs True et False.

Il n'y a pas de valeur undef en Perl 6. Une variable déclarée, mais non initialisée, est évaluée à son type. Mais le type sans valeur définie renvoie une valeur fausse dans un contexte booléen, si bien qu'une variable non initialisée renvoie une valeur fausse, comme le fait undef en Perl 5.

```

1 my $x; say $x; # affiche:
  (Any)
2 my Int $i; say $i; # affiche:
  (Int)
3 say "Défini" if $i; # n'affiche
  rien, $i est évalué à False
4 say "Non défini" unless $i; # affiche "
  Non défini"

```

2.5 Les sigils

- En Perl 5, les tableaux et les hachages ont des *sigils* (signes précédant le nom des variables, comme \$, @, %, etc.) qui changent selon la manière dont on accède à la variable. Ce n'est plus le cas en Perl 6, les sigils sont invariants, on peut considérer qu'ils font partie du nom de la variable.
- Le sigil \$ est utilisé pour les variables scalaires (« une seule chose ») et n'est plus utilisé pour accéder aux éléments individuels d'un tableau ou d'un hachage.
- Le sigil @ est utilisé pour accéder aux variables de type tableau (par exemple : @mois, @mois[2], @mois[2, 4]) et n'est plus utilisé pour les tranches de hachage.
- Le sigil % est utilisé pour accéder aux variables de type hachage (par exemple : %calories, %calories<pomme>, %calories<poire prune>) et n'est plus utilisé pour les tranches clef-valeur de tableaux.
- Le sigil & est utilisé de façon cohérente (et sans l'aide d'un antislash « \») pour prendre une référence sur une fonction ou un opérateur nommé sans l'invoquer.

```

1 my $sub = \&toto; # Perl 5
2 my $sub = &toto; # Perl 6
3
4 callback => sub { say @_ } # Perl 5: ne
  peut passer directement la sub
5 callback => &say # Perl 6: &
  transforme une fonction en nom

```

Voir aussi le chapitre 2 de la première partie de ce tutoriel : lien 40.

2.6 Accès aux valeurs d'un tableau

Les opérations d'accès aux valeurs d'un tableau ne modifient plus le sigil « @ », quel que soit le mode d'accès.

```

1 # accès à un élément d'un tableau
2 say $mois[2]; # Perl 5
3 say @mois[2]; # Perl 6 - @ au lieu de $
4
5 # tranche d'un tableau
6 say join ', ', @mois[6, 8..11]; # Perl 5
  et Perl 6
7
8 # tranche par clef-valeur
9 say join ', ', %mois[6, 8..11]; # Perl
  5

```

```

10 say join ', ', @mois[6, 8..11]:kv; # Perl
  6 - @ au lieu de %;
11 # utilisation de l'adverbe :kv

```

2.7 Accès aux valeurs d'un hachage

Les opérations d'accès aux valeurs d'un hachage ne modifient plus le sigil « % », quel que soit le mode d'accès.

```

1 say $calories{"pomme"}; # Perl 5
2 say %calories{"pomme"}; # Perl 6 - % au
  lieu de $
3
4 say $calories{pomme}; # Perl 5
5 say %calories<pomme>; # Perl 6 -
  chevrons; % au lieu de $
6
7 # Tranches
8 say join ', ', @calories{'poire', 'prune'}; # Perl 5
9 say join ', ', %calories{'poire', 'prune'}; # Perl 6 - % au lieu de @
10 say join ', ', %calories<poire prune>;
  # Perl 6 (version + concise)
11
12 # Tranches clefs-valeurs
13 say join ', ', %calories{'poire', 'prune'}; # Perl 5
14 say join ', ', %calories{'poire', 'prune'}:
  kv; # Perl 6 - avec:kv
15 say join ', ', %calories<poire prune>:kv;
  # Perl 6 (plus propre)

```

2.8 Créations de références

Les créateurs de références vers des tableaux [...] et des hachages ... restent inchangés.

En Perl 5, les références vers des tableaux, hachages ou fonctions sont renvoyées lors de leur création. Les références vers des variables ou fonctions existantes sont créées à l'aide de l'opérateur « \».

En Perl 6, il en va de même pour les tableaux, hachages et fonctions anonymes. Mais les références vers des fonctions nommées sont générées en préfixant le nom de la fonction avec le sigil « & ». On utilise le contexte item pour générer des références vers des variables nommées.

```

1 my $aref = [ 1, 2, 9 ]; # Perl
  5 et Perl 6
2 my $href = { A => 98, Q => 99 }; # Perl
  5 et Perl 6
3
4 my @aaa = <1 4 6>;
5 my $aref = \@aaa; # Perl 5
6 my $aref = item(@aaa); # Perl 6
7 # ou :
8 my $aref = @aaa.item; # Perl 6,
  notation objet/méthode
9
10 my $href = \%hhh; # Perl 5
11 my $href = item(%hhh); # Perl 6
12
13 my $sref = \&foo; # Perl 5
14 my $sref = &foo; # Perl 6

```

2.9 Déréférencement

En Perl 5, la syntaxe pour déréférencer une référence entière est d'utiliser le sigil du type voulu et des accolades autour de la référence.

En Perl 6, les accolades sont remplacées par des parenthèses.

```

1 # Perl 5
2 say    ${$scalar_ref};
3 say    @{$arrayref };
4 say keys %{$hashref };
5 say    &{$subref };
6
7 # Perl 6
8 say    $($scalar_ref);
9 say    @($arrayref );
10 say keys %($hashref );
11 say    &($subref );

```

À noter qu'en Perl 5 comme en Perl 6, les accolades ou parenthèses sont souvent optionnelles, mais leur omission peut rendre le code moins clair.

En Perl 5, l'opérateur flèche « -> » est utilisé pour un accès unique à une référence composée ou pour appeler une fonction à l'aide de sa référence. En Perl 6, c'est l'opérateur point « . » qui joue ce rôle :

```

1 # Perl 5
2 say $arrayref->[7];
3 say $hashref->{'Tartempion'};
4 say $subref->($toto, $titi);
5
6 # Perl 6
7 say $arrayref.[7];
8 say $hashref.{'Tartempion'};

```

3 Les opérateurs

Les opérateurs suivants ne sont pas modifiés :

,	Séparateur de liste
+ - * /	Addition, soustraction, multiplication et division numériques
= += -= *= **=	Affectation (avec éventuellement addition, soustraction, etc.)
%	Modulo numérique (reste de la division entière)
**	Élévation à la puissance
++ --	Incréméntation et décrémentation numériques
&& ^	Opérateurs booléens, haute priorité
not and or xor	Opérateurs booléens, basse priorité
//	« Défini ou » logique. Il existe aussi une version de basse priorité, <code>orelse</code> .
== != < > <= >=	Comparaisons numériques
eq ne lt gt le ge	Comparaisons de chaînes de caractères

Les règles de priorité et d'associativité ont peu changé pour les opérateurs qui ont été maintenus entre les deux versions de Perl. Un tableau de priorité est donné au chapitre *Précedence des opérateurs* du tutoriel : [lien 42](#). Voir aussi [lien 43](#)

3.1 Opérateurs de comparaison pour les tris (`cmp` et `<=>`)

En Perl 5, ces opérateurs renvoient -1, 0 ou 1. En Perl 6, ils renvoient `Order : :Increase`, `Order : :Same`, ou `Order : :decrease`. Cela ne fait une différence que

```
9 say $subref.($toto, $titi);
```

2.10 Documentation intégrée (Pod)

La documentation *Pod* (qui s'appelait `POD`, *Plain Ol' Documentation*, tout en lettres capitales, en Perl 5) a subi des changements entre Perl 5 et Perl 6. La principale différence est qu'il faut insérer un Pod entre des directives `=begin pod` et `=end pod`. Plus généralement, le Pod de Perl 6 est conçu pour être plus régulier et uniforme, un peu plus compact et nettement plus expressif que le `POD` de Perl 5, tout en limitant dans la mesure du possible les différences.

Les autres modifications sont plus des détails qui peuvent néanmoins s'avérer irritants quand on est habitué au `POD` de Perl 5.

Le mieux est sans doute d'utiliser l'interpréteur Perl pour vérifier son Pod, en utilisant l'option `-doc` de la ligne de commande, par exemple `perl6 -doc pod_quelconque.pod`, ce qui affichera tout problème sur la sortie d'erreur. (Selon l'emplacement où Perl 6 est installé et la façon dont il a été installé, il peut être nécessaire de spécifier l'emplacement de Pod : `:To : :Text`.)

La documentation Pod est un point suffisamment important pour mériter ce paragraphe, mais nous ne pouvons vraiment pas entrer dans les détails ici.

Pour plus de détails sur le Pod de Perl 6, voir la *Synopsis S26* : [lien 41](#).

si nous voulons écrire notre propre fonction personnalisée de comparaison.

L'opérateur `cmp` de Perl 5 est remplacé par `leg`,

qui force une comparaison en contexte de chaîne.

L'opérateur `<=>` est inchangé et force une comparaison en contexte numérique.

En Perl 6, l'opérateur `cmp` se comporte soit comme `cmp`, soit comme `<=>`, selon le type de ses arguments.

3.2 Opérateurs de liaison et opérateur de reconnaissance intelligente (ou `smart match`, `~~`)

Les opérateurs de liaison (pour les regex) `< =~ >` et `< !~ >` sont remplacés respectivement par `< ~~ >` et `< !~~ >`.

L'opérateur `~~` de reconnaissance intelligente de Perl 5 n'a pas changé sur le fond, mais les règles gouvernant ce qui est reconnu dépendent du type des arguments, et, les types ayant été très approfondis en Perl 6, ces règles sont loin d'être les mêmes en Perl 5 et en Perl 6.

Surtout, tout semble indiquer que l'opérateur de comparaison intelligente fonctionne de façon satisfaisante en Perl 6, alors que son implémentation en Perl 5 laisse suffisamment à désirer pour que son utilisation soit dépréciée (ou plus exactement qu'il soit déclaré « expérimental ») dans les dernières versions de Perl 5.

Pour plus de détails sur le fonctionnement de `< ~~ >`, voir [lien 44](#).

3.3 Opérateurs bit à bit

Comme en Perl 5, les opérateurs `< ! >` et `< - >` assurent respectivement la négation logique et arithmétique (opérateurs non bit à bit).

En Perl 5, le comportement des opérateurs `&` | dépendait du type des arguments : par exemple, `31 | 33` ne renvoie pas la même chose que `"31" | "33"`.

En Perl 6, ces opérateurs à un seul caractère ont été supprimés et sont remplacés par des opérateurs à deux caractères qui forcent le contexte approprié.

```

1 # Opérateurs infixés (deux arguments, un
  de chaque côté de l'opérateur)
2 +& +| +^ Et, Ou, Ou exclusif: Numé
  rique
3 ~& ~| ~^ Et, Ou, Ou exclusif: Chaîne
  de caractères
4 ?& ?| ?^ Et, Ou, Ou exclusif: Booléen
5
6 # Opérateurs préfixés (un argument, après
  s l'opérateur)
7 +^ Non: Numérique
8 ~^ Non: Chaîne de caractères
9 ?^ Non: Booléen (même chose que l'opé
  rateur !)
```

Les opérateurs de décalage de bits `< et >` sont remplacés par `<+<` et `<+>` :

```

1 say 42 << 3; # Perl 5
2 say 42 +< 3; # Perl 6
```

3.4 Opérateur flèche (`->`)

Il est beaucoup moins fréquent en Perl 6 qu'en Perl 5 d'utiliser des références, si bien que l'on a aussi moins souvent besoin d'un opérateur de déréférencement. Mais en cas de besoin, l'opérateur de déréférencement est le point `< . >` et non plus la flèche `< ->`, et le point remplace aussi la flèche pour les appels de méthodes. Donc, une construction `$tabl_ref->[3]` de Perl 5 devient `$tabl_ref.[3]` en Perl 6.

3.5 La « grosse virgule » (`=>`)

En Perl 5, `< =>` agissait comme une virgule et transformait aussi ce qui précédait en une chaîne de caractères.

En Perl 6, `< =>` est l'opérateur de paire (type `Pair`), ce qui est assez différent en principe, mais fonctionne de la même façon dans un bon nombre de situations.

Par exemple, pour initialiser un hachage ou passer des arguments à une fonction attendant une référence à un hachage, l'utilisation est la même :

```

1 # Marche en Perl 5 et en Perl 6
2 my %hachage = ( AAA => 1, BBB => 2 );
3 prends_le_butin( 'diamants', { niveau =>
  'élevé', nombre => 9 });
4
  # Noter les accolades, le
  second argument est un
  hashref
```

Mais si vous utilisiez cet opérateur pour ne pas devoir mettre de guillemets sur une partie d'une liste, ou pour passer une liste plate d'arguments de type `CLEF, VALEUR, CLEF, VALEUR`, alors il y a de bonnes chances qu'utiliser ainsi `=>` ne fonctionne pas comme prévu. La solution de contournement consiste à remplacer la grosse virgule par une virgule ordinaire et à ajouter des guillemets ou des apostrophes à son argument de gauche. Ou vous pouvez modifier l'API de la fonction pour qu'elle accepte un hachage en argument.

Une meilleure solution à long terme est de modifier l'API de la fonction pour qu'elle accepte des paires. Mais cela oblige à modifier d'un seul coup tous les appels de la fonction.

```

1 # Perl 5
2 sub prends_le_butin {
3   my $butin = shift;
4   my %options = @_;
5   # ...
6 }
7 prends_le_butin( 'diamants', niveau =>
  'élevé', nombre => 9 );
8
  # Noter: pas d'accolade
  cette fois
9
10 # Perl 6, API d'origine
11 sub prends_le_butin ( $butin, %options
  ) {
12
  # L'astérisque * signifie
  de prendre tous les
  params
```

```

13     ...
14 }
15 prends_le_butin( 'diamants', niveau => '
    élevé', nombre => 9 );
16         # Noter: pas d'accolade
            dans cette API
17
18 # Perl 6, API modifiée pour spécifier
    les options valides
19 # Les deux-points avant les sigils
    signifient que l'on attend une paire
20 # dont la clef a le même nom que la
    variable
21
22 sub prends_le_butin ( $butin, :$niveau?,
    :$nombre = 1 ) {
23     # Cette version va vérifier les
        arguments inattendus!
24     ...
25 }
26 prends_le_butin( 'diamants', nouveau => '
    élevé' );
27         # Génère une erreur en
            raison de la faute
28         # d'orthographe sur le nom
            du paramètre (nouveau)

```

3.6 Opérateur de concaténation

L'opérateur point « . » étant maintenant un opérateur d'appel de méthode et de déréréférencement, l'opérateur de concaténation n'est plus le point, mais le tilde « ~ ».

De même, l'opérateur d'affectation-concaténation n'est plus « .= », mais « ~= ».

3.7 Opérateur ternaire « ? : » remplacé par « ?? !! »

Cet opérateur utilise désormais deux points d'interrogation au lieu d'un seul et deux caractères points d'exclamation au lieu du deux-points :

```

1 my $résultat = ( $note > 60 ) ? 'Réussi
    ' : 'Loupé'; # Perl 5
2 my $résultat = ( $note > 60 ) ?? 'Réussi
    ' !! 'Loupé'; # Perl 6

```

3.8 Opérateur de répétition (de liste ou de chaîne)

En Perl 5, x est l'opérateur de répétition. En contexte scalaire ou si l'opérande à sa gauche n'est pas entre parenthèses, x renvoie une chaîne de caractères. En contexte de liste et si l'argument à gauche est entre parenthèses, il répète la liste. Il fallait simplifier ces règles alambiquées.

En Perl 6, x répète les chaînes de caractères, quel que soit le contexte.

En Perl 6, le nouvel opérateur xx répète des listes, quel que soit le contexte.

```

1 # Perl 5
2 print '-' x 80; # Affiche
    une rangée de 80 tirets

```

```

3 @uns = 1 x 80; # Un
    tableau contenant un seul élé
    ment,
4 # une cha
    îne
    de 80
    fois
    le
    chiffre
    1
5 @uns = (1) x 80; # Un
    tableau de 80 chiffres 1
6 @uns = (5) x @uns; # Met à 5
    tous les éléments de @uns
7 # Perl 6
8 print '-' x 80; # Inchang
    é
9 @uns = 1 x 80; # Inchang
    é
10 @uns = 1 xx 80; # Parenth
    èses plus nécessaires
11 @uns = 5 xx @uns; # Parenth
    èses plus nécessaires

```

3.9 Opérateurs de citation et assimilés

Il existe un opérateur de citation qui garantit des chaînes absolument littérales, sans aucune interpolation, c'est « Q » ou « [...] » (à supposer que vous sachiez comment obtenir « [» et «] » sur votre clavier). Même les échappements par antislash ne s'appliquent pas. Par exemple :

```

1 my $littéral = Q{Cela reste une accolade
    fermante → \};
2 # renvoie la chaîne : "Cela reste une
    accolade fermante → \"

```

L'opérateur « q » fait ce à quoi l'on s'attend en venant de Perl 5, à savoir pas d'interpolation à l'exception des séquences d'échappement par antislash. Par exemple :

```

1 my $var-échap = q{Ce n'est pas une
    accolade fermante → \} mais ceci →
    };
2 renvoie : "Ce n'est pas une accolade
    fermante → } mais c'est ceci →"

```

Comme en Perl 5, vous pouvez utiliser les apostrophes ou guillemets simples ('. . .') pour obtenir le même résultat.

L'opérateur « qq » autorise l'interpolation des variables (de même que les guillemets doubles « " »). Cependant, par défaut, seules les variables scalaires sont interpolées. Pour interpoler les autres types de variables (tableaux, hachages, etc.), il faut les suffixer avec des crochets. Par exemple, @a = <1 2 3>; say qq/@a] exemple@exemple.com/; donne : 1 2 3 exemple@exemple.com. Toutefois, l'interpolation de hachages donne actuellement un résultat visuellement quelque peu inattendu :

```

1 my %hachage = 1 => 2, 3 => 4;
2 say "%a["; # affiche : 1 2 3
    4.

```

comme si les clefs étaient séparées des valeurs par des tabulations et les valeurs des clefs suivantes

par des espaces. Il est également possible d'interpoler dans des chaînes de caractères du code Perl 6 en le mettant entre accolades :

```
1 my $x = 5;
2 say "Le double de $x est {$x * 2} et le
   carré de $x est {$x ** 2}.";
3 # Imprime : Le double de 5 est 10 et le
   carré de $x est 25.
4 say qq/Le double de $x est {$x * 2} et
   le carré de $x est {$x ** 2}./;
5 # idem
```

L'opérateur de citation de mots « qw » fonctionne comme en Perl 5 et peut aussi être rendu par <...>. Par exemple, qw/ a b c/ est équivalent à <a b c>. Cet opérateur ne fait pas d'interpolation. Il existe une version, qqw, qui fait l'interpolation des variables. Par exemple :

```
1 my $a = 42; # la variable $a n'a
   rien de spécial en Perl 6
2 say qqw/$a b c/; # -> 42 b c
```

L'opérateur de citation du shell (appel de fonctions système) est qx, comme en Perl 5, mais il faut noter que les accents graves (ou *backticks*) '...' ne fonctionnent pas en Perl 6 et que les variables ne sont pas interpolées dans les chaînes qx. Pour interpoler les variables, il suffit de remplacer qx par qqx.

L'opérateur qr n'existe plus en Perl 6, mais les regex de Perl 6 offrent des possibilités bien plus puissantes, comme le mécanisme des regex nommées.

L'opérateur de translittération tr/// n'est pas bien documenté à l'heure actuelle, mais il semble fonctionner comme en Perl 5, avec cependant cette différence que les intervalles de caractères s'écrivent « a..z », donc avec l'opérateur standard d'intervalle (au lieu de « a-z » en Perl 5). Il existe une version méthode de tr///, mieux documentée, qui s'appelle .trans. Cette méthode utilise des listes de paires :

```
1 $x.trans(['a'..'c'] => ['A'..'C'], ['d'
   ..'q'] => ['D'..'Q'], ['r'..'z'] =>
   ['R'..'Z']);
```

Les deux côtés de chaque paire sont interprétés comme le ferait tr///. Cela donne une grande flexibilité. Voir lien 45 pour une description complète.

L'opérateur y///, qui était en Perl 5 un simple synonyme de tr///, a été abandonné en Perl 6.

Les documents « ici même » (*heredocs*) sont spécifiés de façon un peu différente en Perl 6 : il faut utiliser le mot-clef :to immédiatement après l'opérateur de citation, par exemple, la séquence q :to/FIN commence un document se terminant par FIN. Les échappements et l'interpolation dépendent de l'opérateur de citation utilisé (citation littérale avec Q, échappement des antislashes avec q et interpolation des variables avec qq).

Ce paragraphe ne fait que résumer les principaux opérateurs de citation et assimilés ressemblant à ceux de Perl 5. Le chapitre Citation et analyse

lexicale du tutoriel (lien 46) explique plus en détail la logique de fonctionnement du très puissant mécanisme sous-jacent de citation de chaînes de caractères, permettant à l'utilisateur de maîtriser très finement les caractéristiques de sa chaîne.

Pour encore plus de détails, voir lien 47.

3.10 Opérateurs d'entrées-sorties (E/S ou IO)

Comme l'opérateur « diamant » (ou « carreau ») <...> est un opérateur de citation de mots (cf. § 3.9 ci-dessus), il n'est plus possible d'utiliser <> pour lire les lignes d'un fichier. Pour ce faire, il faut soit créer un objet IO à partir d'un nom de fichier, soit utiliser un descripteur de fichier (*filehandle*) ouvert et, dans les deux cas, utiliser la méthode .lines sur l'objet ou le descripteur :

```
1 # Objet IO :
2 my @tableau-de-lignes = "filename".IO.
   lines; # remarquer la concision
3 # Descripteur de fichier (ou FH) :
4 my $fh = open "filename", :r;
   # :r -> FH en
   lecture
5 my @a = $fh.lines;
```

Pour lire un fichier ligne à ligne itérativement :

```
1 for 'gigantesque-csv'.IO.lines -> $line
   {
2     # Traiter $line
3 }
```

À noter que la syntaxe utilisant for ne serait pas recommandée en Perl 5 pour un très gros fichier, car cela impliquerait de stocker l'ensemble du fichier dans un tableau temporaire et pourrait saturer la mémoire; en Perl 6, cela ne pose aucun problème grâce à la paresse de l'opérateur : la ligne n'est lue que quand on en a besoin et il n'y a jamais plus d'une ligne en mémoire. Noter également l'utilisation ici d'un « bloc pointu » avec la syntaxe -> (voir le chapitre sur les Boucles du tutoriel).

Pour avaler d'un seul coup (*to slurp*) tout un fichier dans un scalaire, il faut utiliser la méthode .slurp. Par exemple :

```
1 my $x = "filename".IO.slurp;
2
3 # Ou, façon procédurale (non objet)
   encore plus simple :
4 my $x = slurp "filename";
5
6 # Ou, avec un FH :
7 my $fh = open "filename", :r;
8 my $x = $fh.slurp;
```

Le descripteur de fichier en entrée magique ARGV est remplacé par \$*ARGFILES et le tableau @ARGV des arguments de la ligne de commande par @*ARGS.

Pour écrire dans un fichier :

```
1 my $fh = open "filename", :w; # :w -
   > ouverture en écriture
```



```
2 $fh.say("données, trucs, bidules");
3 $fh.close;
```

Pour plus de détails sur les entrées-sorties, voir lien 48.

3.11 Deux points (« .. ») et trois points (« ... », création d'intervalles et opérateur flip-flop

En Perl 5, le « .. » était deux opérateurs complètement différents, selon le contexte. En contexte de liste, c'est l'opérateur bien connu de création d'intervalles. Il y a beaucoup à dire sur les intervalles en Perl 6 (par exemple, il est possible d'inclure ou non les bornes de l'intervalle dans la liste créée, ainsi 1..5 exclut la borne supérieure de l'intervalle et est donc équivalent à 1..4), mais un intervalle de Perl 5 ne devrait pas nécessiter de traduction en Perl 6.

En contexte scalaire, .. et ... étaient en Perl 5 les opérateurs *flip-flop*, beaucoup moins utilisés. Ils ont été remplacés en Perl 6 par les opérateurs ff et fff.

Ces deux opérateurs sont évalués à faux jusqu'à ce que \$_ soit reconnu par le premier argument, puis à vrai jusqu'à ce que le second argument reconnaisse \$_. L'exemple ci-dessous illustre la différence assez subtile entre les deux. À noter qu'il existe également des versions $\hat{f}f$, $\hat{f}f\hat{f}$, $f\hat{f}$, $f\hat{f}\hat{f}$, $\hat{f}\hat{f}\hat{e}t$ $\hat{f}\hat{f}\hat{q}$ qui renvoient faux pour la borne du premier ou du second argument où se trouve l'accent circonflexe.

```
1 # Dès que le premier argument reconnaît
   $_, ff renvoie vrai puis
2 # évalue le second. Ci-dessous, AB
   reconnaît B et ff renvoie faux
3 for <X Y Z AB C D B E F> {
4   say $_ if /A/ ff /B/; # imprime
   seulement "AB"
5 }
6
7 # fff n'évalue pas le second argument
   pour la valeur de $_ qui
8 # a été reconnue par le premier argument
9 for <X Y Z AB C D B E F> {
10  say $_ if /A/ fff /B/; # Imprime "AB
   ", "C", "D" et "B"
11 }
12
13 # ^fff renvoie faux quand le premier
   argument reconnaît $_ puis
14 # renvoie vrai jusqu'à ce que le second
   argument reconnaisse $_
15 for <X Y Z AB C D B E F> {
16  say $_ if /A/ ^fff /B/; # Imprime "AB
   ", "C" et "D", mais pas "B"
17 }
18 # fff^ renvoie faux dès que le second
   argument reconnaît $_
19 for <X Y Z AB C D B E F> {
20  say $_ if /A/ fff^ /B/; # Imprime "AB
   ", "C" et "D", mais pas "B"
21 }
```

3.12 Voir aussi

Voir aussi le chapitre Opérateurs de la seconde partie du tutoriel lien 49.

Retrouvez la suite de l'article de **Laurent Rosenfeld** en ligne : lien 50

Liste des liens

Page 16

lien 1 : ... <http://philippe-quet.developpez.com/articles/concepts-en-algorithmique/>

Page 18

lien 2 : ... <http://rchastain.developpez.com/tutoriel/delphi/regularexpressions/#LI-E>

Page 19

lien 3 : ... <http://rchastain.developpez.com/tutoriel/delphi/regularexpressions/#LII-B>

Page 20

lien 4 : ... <http://rchastain.developpez.com/tutoriel/delphi/regularexpressions/#LIII-D>

Page 21

lien 5 : ... <http://rchastain.developpez.com/tutoriel/delphi/regularexpressions/#LIV-C>

lien 6 : ... <http://rchastain.developpez.com/tutoriel/delphi/regularexpressions/#LV-B>

Page 24

lien 7 : ... <http://rchastain.developpez.com/tutoriel/delphi/regularexpressions/#LVI-D>

Page 25

lien 8 : ... <http://rchastain.developpez.com/tutoriel/delphi/regularexpressions/#LVII-B>

lien 9 : ... <http://www.princeton.edu/~mlovett/reference/Regular-Expressions.pdf>

lien 10 : ... <http://rchastain.developpez.com/tutoriel/delphi/regularexpressions/fichiers/ExemplesRegular.zip>

lien 11 : ... <http://rchastain.developpez.com/tutoriel/delphi/regularexpressions/>

Page 26

lien 12 : ... <http://www.developpez.net/forums/d1542381/systemes/windows/avez-installe-windows-10-vos-impressions-nouvel-os-microsoft/>

Page 27

lien 13 : ... <https://www.salesforce.com/fr/company/>

lien 14 : ... <https://www.salesforce.com/fr/company/>

lien 15 : ... http://fr.wikipedia.org/wiki/Logiciel_en_tant_que_service

lien 16 : ... <https://developer.salesforce.com/>

Page 28

lien 17 : ... <http://www.salesforce.com/fr/crm/editions-pricing.jsp>

lien 18 : ... <mailto:pad@fsgbu.com>

Page 30

lien 19 : ... https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_methods_system_system.htm

Page 31

lien 20 : ... <http://aurelien-laval.developpez.com/tutoriels/salesforce/tester-son-code-apex/>

lien 21 : ... https://developer.salesforce.com/page/How_To_Test_Your_Apex_Triggers

Page 40

lien 22 : ... <mailto:pad@fsgbu.com>

lien 23 : ... <https://github.com/AurelienLaval/Manage-commerciaux-opportunities>

lien 24 : ... <http://aurelien-laval.developpez.com/tutoriels/salesforce/integrer-le-framework-pad/>

Page 41

lien 25 : ... <http://qt.developpez.com/actu/87204/La-premiere-preversion-technologique-de-Qt-3D-est-disponible-avec-Qt-5-5-le-moteur-3D-de-Qt-approche-de-la-maturite/>

lien 26 : ... <http://qt.developpez.com/actu/87212/Sortie-de-Qt-5-5-avec-de-nouveaux-modules-pour-la-3D-Canvas3D-et-Qt-3D-et-des-fonctionnalites-de-cartographie/>

lien 27 : ... <http://code.qt.io/cgit/qt/qtquickcontrols2.git/>

lien 28 : ... <http://code.qt.io/cgit/qt/qtspeech.git/>

lien 29 : ... <http://code.qt.io/cgit/qt/qtserialbus.git/>

lien 30 : ... <http://code.qt.io/cgit/qt/qtwayland.git/>

lien 31 : ... <http://www.developpez.net/forums/d1541638/c-cpp/bibliotheques/qt/sortie-qt-5-6-alpha/>

Page 42

lien 32 : ... <https://fr.wikipedia.org/wiki/QQQCCP>

lien 33 : ... http://vviale.developpez.com/tutoriels/scratch/creation-mini-jeu/fichiers/Utilisez_cette_fiche_pour_creer_vos_descriptions.pdf

Page 49

lien 34 : ... <http://vviale.developpez.com/tutoriels/scratch/creation-mini-jeu/>

Page 50

lien 35 : ... <https://scratch.mit.edu/scratch2download/>

lien 36 : ... http://scratchfr.free.fr/Scratchfr_v2014/blocks_editor_Experiment.po.zip

Page 57

lien 37 : ... <http://vviale.developpez.com//tutoriels/scratch/thesee-minotaure/>

Page 58

lien 38 : ... <http://doc.perl6.org/>

Page 59

lien 39 : ... <http://laurent-rosenfeld.developpez.com/tutoriels/perl/perl6/les-bases/#L2-2-1>

Page 60

lien 40 : ... <http://laurent-rosenfeld.developpez.com/tutoriels/perl/perl6/les-bases/#L2>

Page 61

lien 41 : ... <http://design.perl6.org/S26.html>

lien 42 : ... <http://laurent-rosenfeld.developpez.com/tutoriels/perl/perl6/les-nouveautes/#L4-3>

lien 43 : ... http://doc.perl6.org/language/operators#Operator_Precedence

Page 62

lien 44 : ... http://design.perl6.org/S03.html#Smart_matching

Page 64

lien 45 : ... <http://design.perl6.org/S05.html#Transliteration>

lien 46 : ... <http://laurent-rosenfeld.developpez.com/tutoriels/perl/perl6/approfondissements/#L4>

lien 47 : ... <http://doc.perl6.org/language/quotinq>

Page 65

lien 48 : ... <http://doc.perl6.org/language/io>

lien 49 : ... <http://laurent-rosenfeld.developpez.com/tutoriels/perl/perl6/les-nouveautes/#L4>

lien 50 : ... <http://laurent-rosenfeld.developpez.com/tutoriels/perl/perl6/annexe-01/>