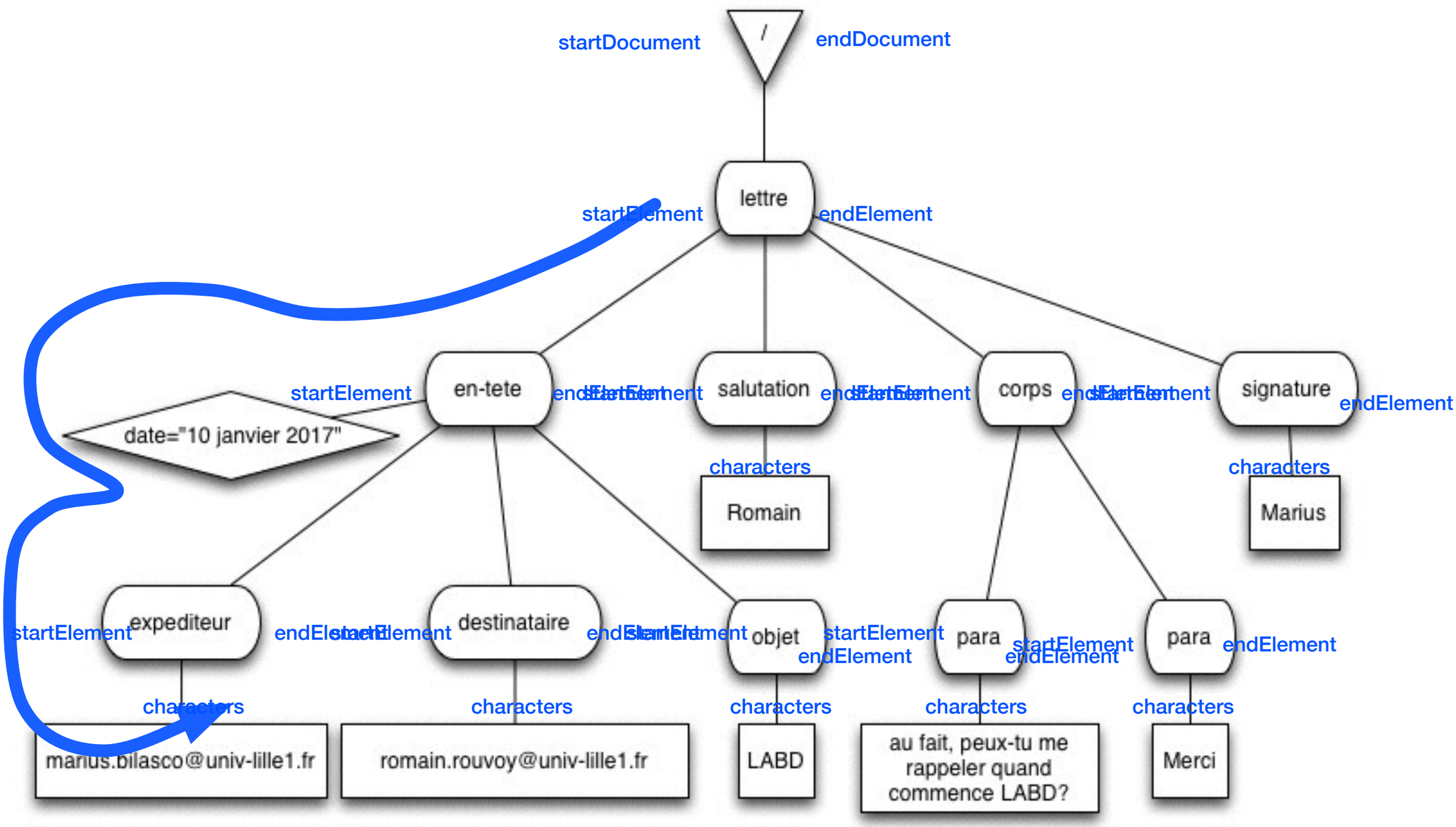
LABD

Master Info M1 2017-2018

Cours 2 : Naviguer avec XPath

Simple Api for XML



Simple Api for XML

org.w3c.dom.bootstrap
org.w3c.dom.events
org.w3c.dom.ls
org.xml.sax
org.xml.sax.ext
org.xml.sax.helpers

org.xml.sax

Interfaces
AttributeList
Attributes
ContentHandler
DocumentHandler
DTDHandler
EntityResolver
ErrorHandler
Locator
Parser
XMLFilter
XMLReader

Classes
HandlerBase
InputSource

Exceptions
SAXException
SAXNotRecognizedException
SAXNotSupportedException
SAXParseException

Method Summary

void	**characters**(char[] ch, int start, int length) Receive notification of character data.
void	**endDocument**() Receive notification of the end of a document.
void	**endElement**(String uri, String localName, String qName) Receive notification of the end of an element.
void	**endPrefixMapping**(String prefix) End the scope of a prefix-URI mapping.
void	**ignorableWhitespace**(char[] ch, int start, int length) Receive notification of ignorable whitespace in element content.
void	**processingInstruction**(String target, String data) Receive notification of a processing instruction.
void	**setDocumentLocator**(Locator locator) Receive an object for locating the origin of SAX document events.
void	**skippedEntity**(String name) Receive notification of a skipped entity.
void	**startDocument**() Receive notification of the beginning of a document.
void	**startElement**(String uri, String localName, String qName, Attributes atts) Receive notification of the beginning of an element.
void	**startPrefixMapping**(String prefix, String uri) Begin the scope of a prefix-URI Namespace mapping.

XPath

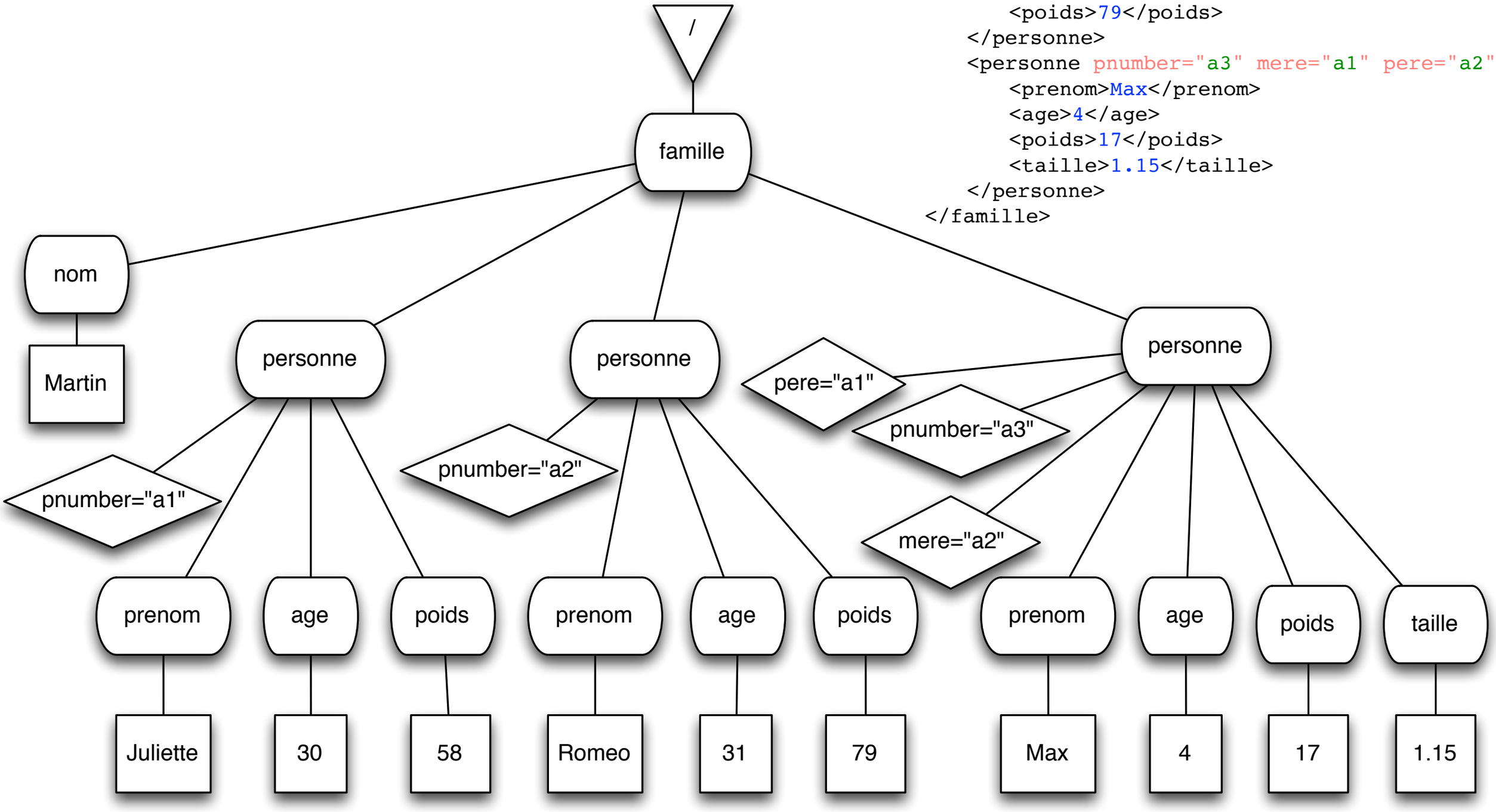
- Le langage XPath est utilisé dans de nombreux langages de manipulation de données XML (XSLT, XML-Schema, XQuery).
- Une donnée XML est un arbre composé de nœuds (de différents types) et XPath est un langage permettant de désigner des nœuds dans l'arbre XML, en décrivant des chemins dans cet arbre à l'aide d'expressions de chemin (un peu comme des chemins `unix` dans le système de fichiers).
- XPath 1 : juste des expressions de chemin. Recommandation depuis1999, au cœur de XSLT.
- XPath 2 : des boucles `for`, des conditions `if`, des variables, . . . Recommandation depuis 2006 (à l'étude depuis 2001), au cœur de XQuery

Exemple de document

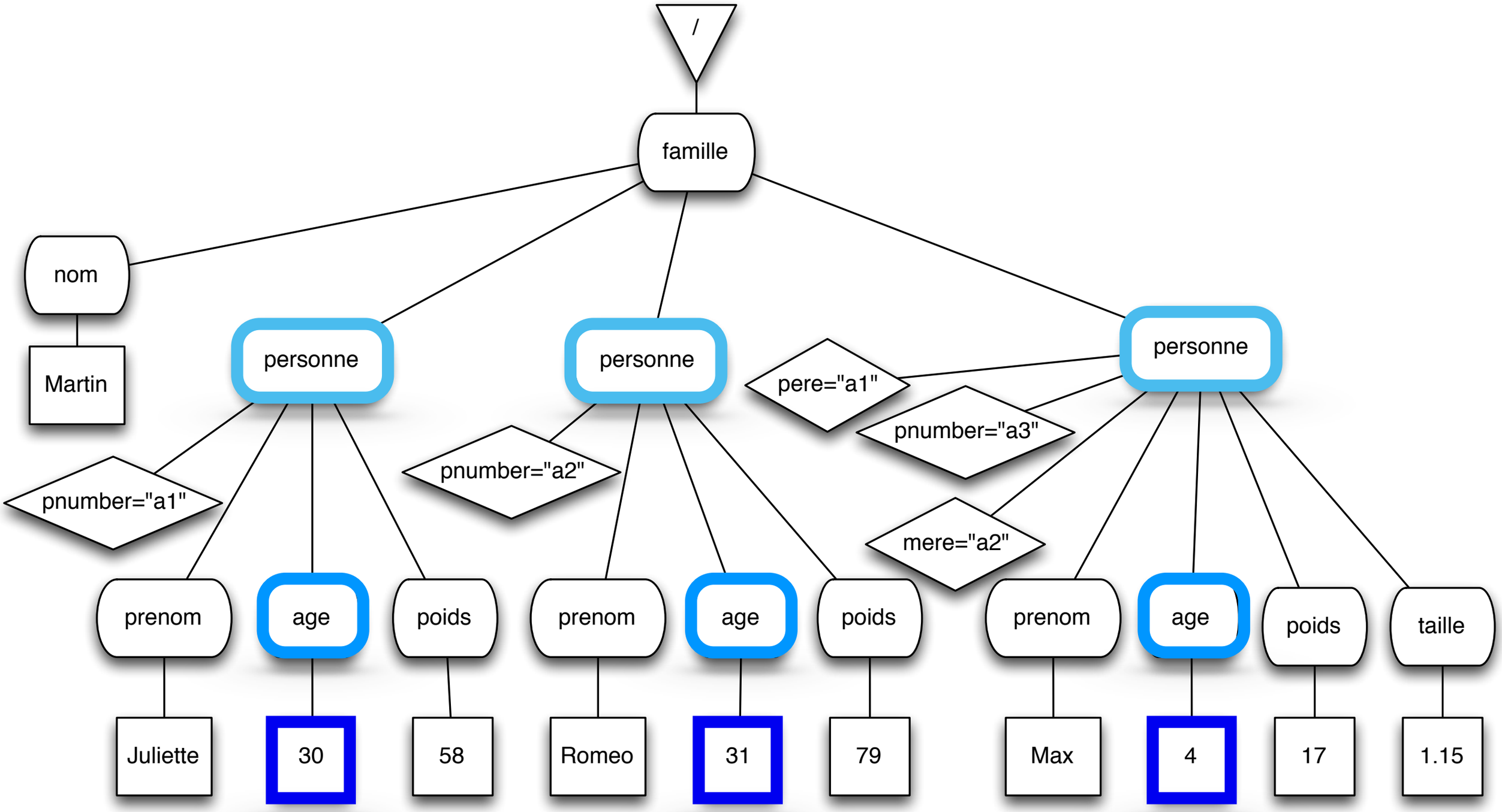
```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE famille SYSTEM "famille.dtd">
<famille>
   <nom>Martin</nom>
   <personne pnumber="a1">
      <prenom>Juliette</prenom>
      <age>30</age>
      <poids>58</poids>
   </personne>
   <personne pnumber="a2">
      <prenom>Romeo</prenom>
      <age>31</age>
      <poids>79</poids>
   </personne>
   <personne pnumber="a3" mere="a1" pere="a2">
      <prenom>Max</prenom>
      <age>4</age>
      <poids>17</poids>
      <taille>1.15</taille>
   </personne>
</famille>
```

Arbre correspondant

```
<famille>
    <nom>Martin</nom>
    <personne pnumber="a1">
        <prenom>Juliette</prenom>
        <age>30</age>
        <poids>58</poids>
    </personne>
    <personne pnumber="a2">
        <prenom>Romeo</prenom>
        <age>31</age>
        <poids>79</poids>
    </personne>
    <personne pnumber="a3" mere="a1" pere="a2">
        <prenom>Max</prenom>
        <age>4</age>
        <poids>17</poids>
        <taille>1.15</taille>
    </personne>
</famille>
```



```
/descendant::personne/child::age/child::text()
```



Les expressions de chemin

Une expression de chemin XPath :

- s'évalue en fonction d'un nœud contexte,
- désigne un ou plusieurs chemins dans l'arbre à partir du nœud contexte,
- a pour résultat une séquence d’items

Modèle de données de XPath 1.0

Le résultat d'une requête XPath est un ensemble de nœuds qu'il faut voir comme un ensemble de références vers des nœuds de l'arbre XML.

Qui dit ensemble dit

- non ordonné (en réalité on récupère les nœuds dans l'ordre du document)
- sans doublon

Modèle de données de XPath 2.0

Le résultat d'une requête XPath est une séquence de nœuds ou de valeurs atomiques.

Qui dit séquence dit

- ordonné
- doublons possibles

Un modèle simple

- Une valeur XPath est une séquence d'items
- Un item est un nœud ou une valeur atomique.
- Il y a 7 sortes de noeuds:
  - Document
  - Élément
  - Attribut
  - Texte
  - Commentaire
  - Instruction
  - Espace de nom

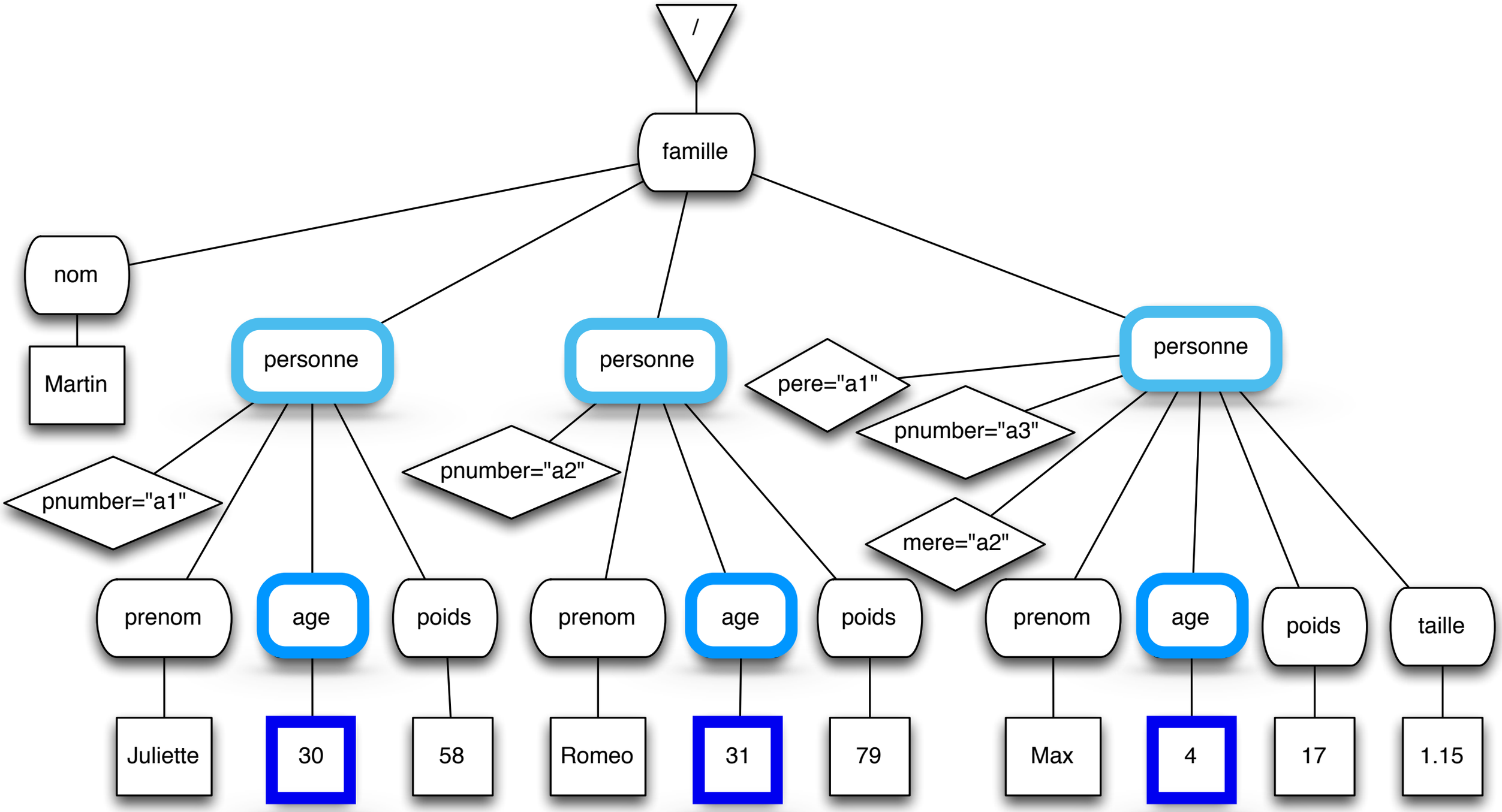
Exemples de valeurs

- `47`
- `<A/>`
- `(1, 2, 3)`
- `(47, <A/>, "Hello")`
- `( )`
- Un document XML
- Un attribut seul

Remarques sur les valeurs

- Il n’y a pas de distinction entre un item et une séquence de longueur 1.
- Il n’y a pas de séquences imbriquées
- Il n’y a pas de valeur nulle
- Une séquence peut être vide
- Une séquence peut contenir des données hétérogènes
- Toutes les séquences sont ordonnées

```
/descendant::personne/child::age/child::text()
```



Les expressions de chemin

Une expression de chemin XPath consiste en une séquence d'étapes séparées par `/` ou `//`

Elle peut être

- absolue comme `/A/B/C` ou `//A/B//C`. Le nœud contexte (point de départ) est alors la racine du document (le nœud document) ou
- relative comme `A/B/C` ou `A/B//C`. Le nœud contexte dépend alors...du contexte dans lequel on utilise le chemin XPath.

Les étapes

Une étape est composée de 3 composants : un axe, un filtre et une liste éventuellement vide de prédicats en suivant la syntaxe

```
axe::filtre[prédicat1][prédicat2]...
```

axe sens de navigation dans l'arbre par rapport au nœud contexte

filtre type des nœuds à retenir

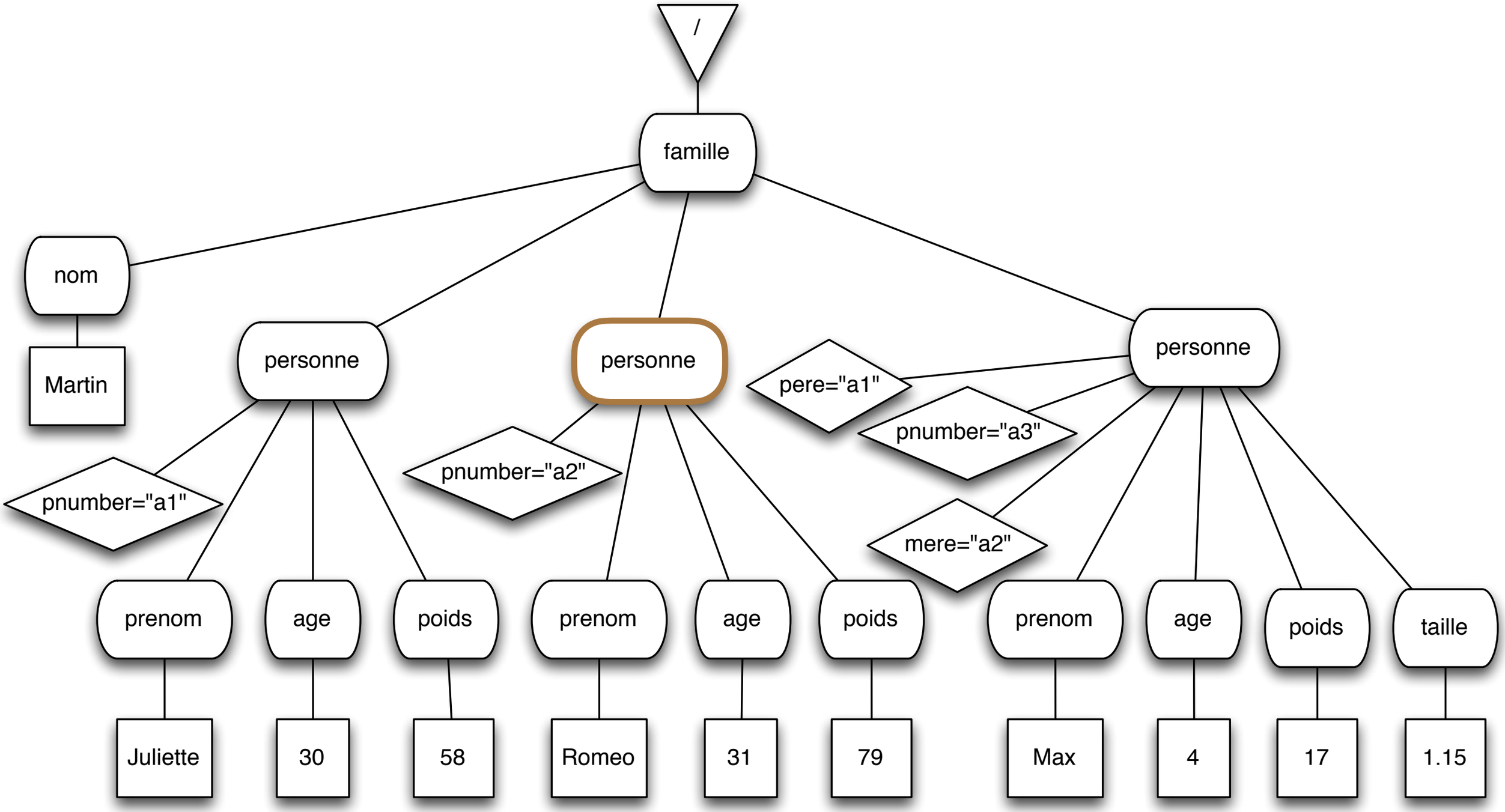
prédicat(s) propriétés que doivent satisfaire les nœuds parmi les nœuds retenus.

```
/descendant::prenom[child::text()=”Romeo“]/parent::*
```

Les axes

- **self** le nœud courant lui-même

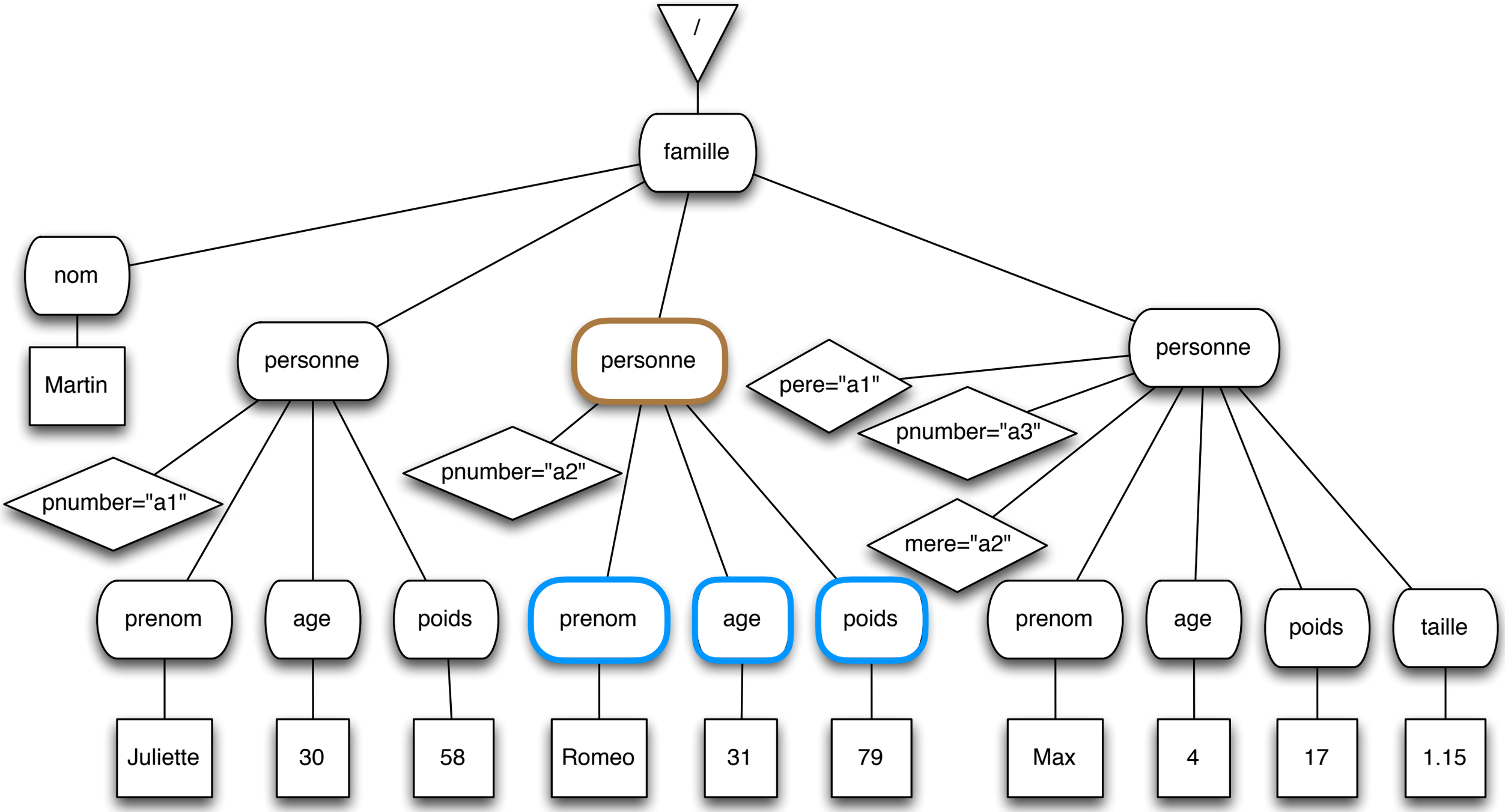
self::


/
famille
nom
Martin
personne
pnumber="a1"
prenom
Juliette
age
30
poids
58
personne
pnumber="a2"
prenom
Romeo
age
31
poids
79
pere="a1"
pnumber="a3"
mere="a2"
personne
prenom
Max
age
4
poids
17
taille
1.15

Les axes

- self le nœud courant lui-même
- **child sélectionne les nœuds éléments fils du nœud courant**

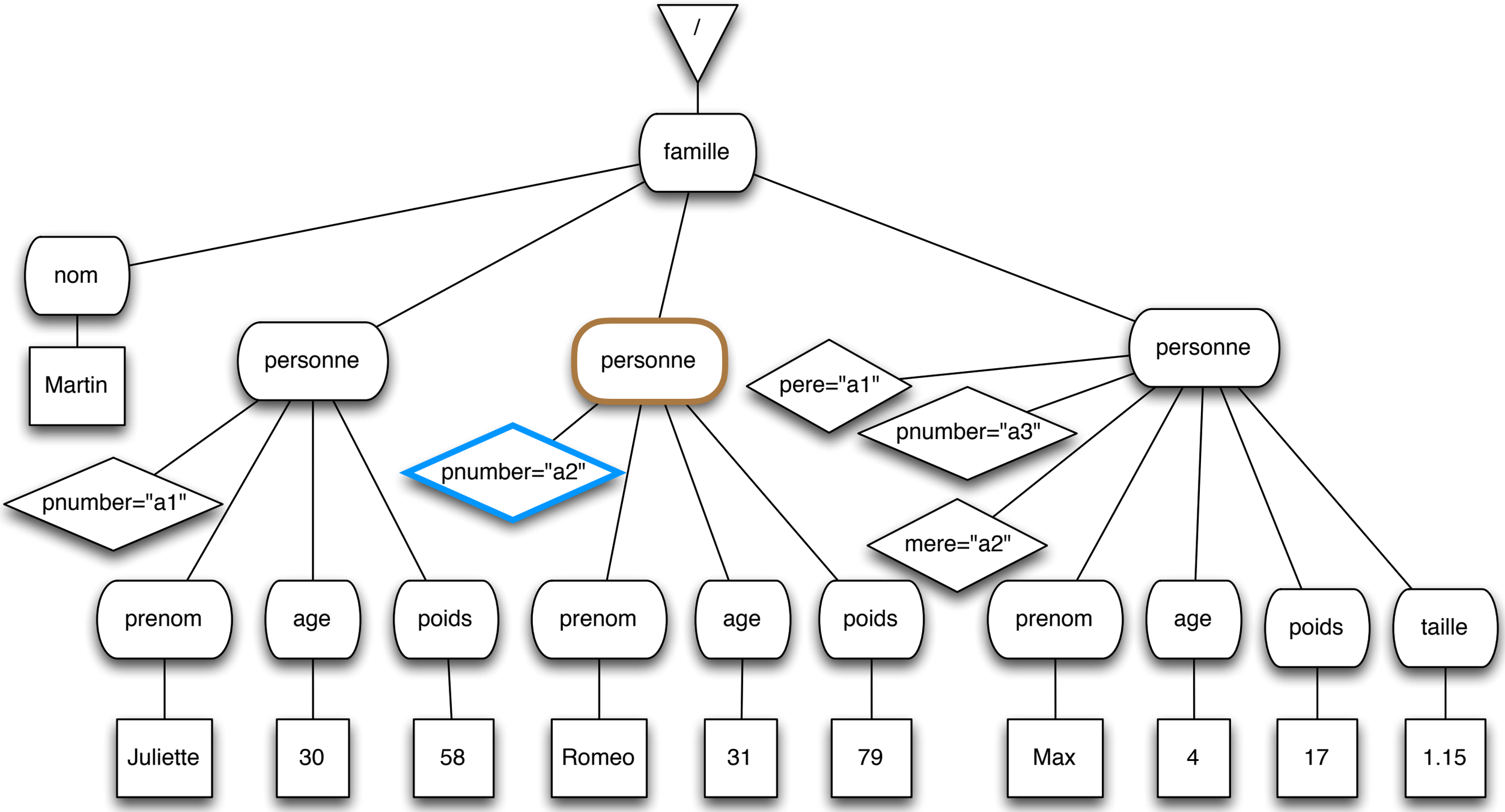
child::


/
famille
nom
Martin
personne
pnumber="a1"
prenom
Juliette
age
30
poids
58
personne
pnumber="a2"
prenom
Romeo
age
31
poids
79
pere="a1"
pnumber="a3"
mere="a2"
personne
prenom
Max
age
4
poids
17
taille
1.15

Les axes

- self le nœud courant lui-même
- child sélectionne les nœuds éléments fils du nœud courant
- **attribute les attributs de l'élément courant**

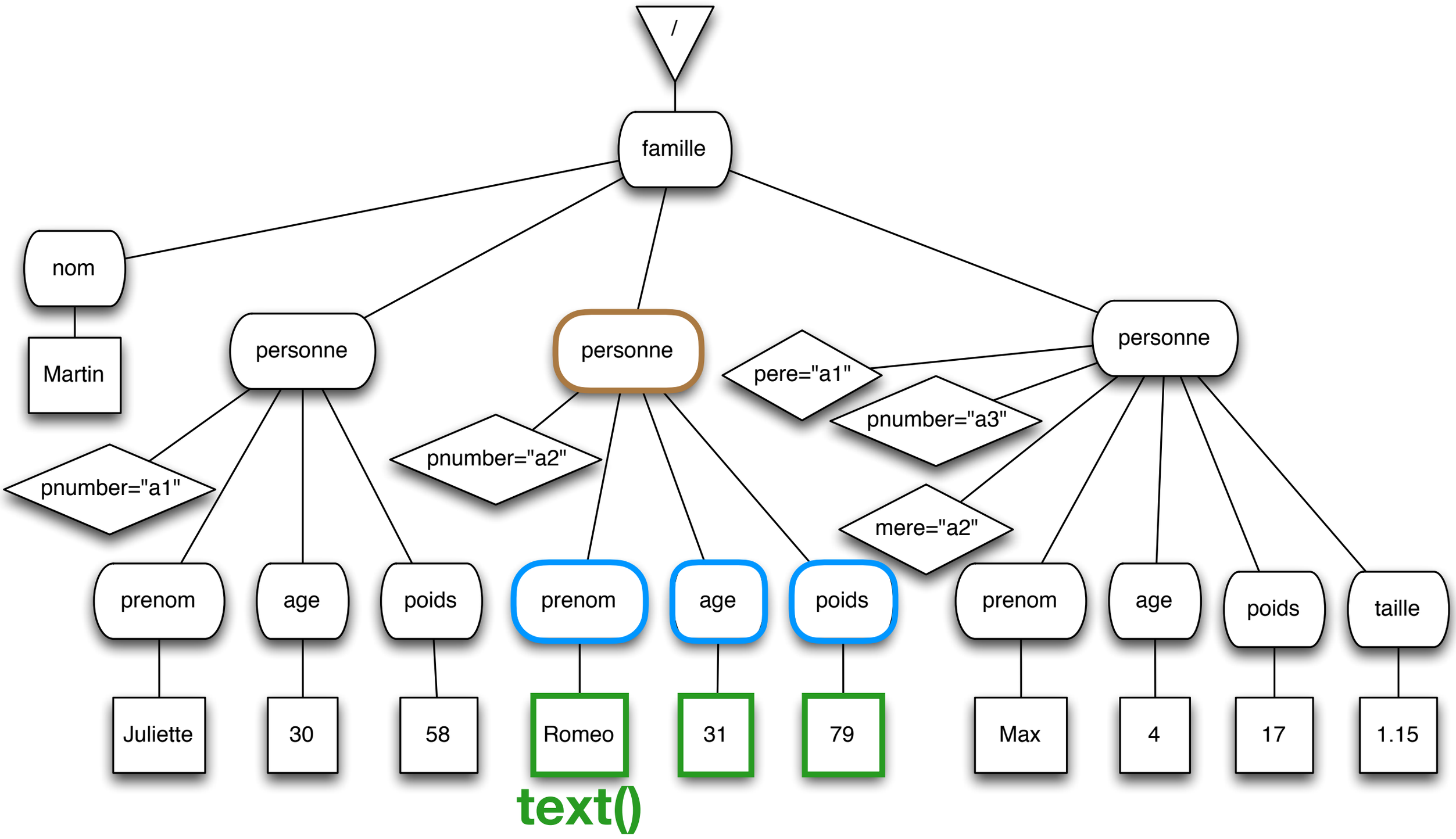
attribute::


/
famille
nom
Martin
personne
pnumber="a1"
prenom
Juliette
age
30
poids
58
personne
pnumber="a2"
prenom
Romeo
age
31
poids
79
pere="a1"
pnumber="a3"
mere="a2"
personne
prenom
Max
age
4
poids
17
taille
1.15

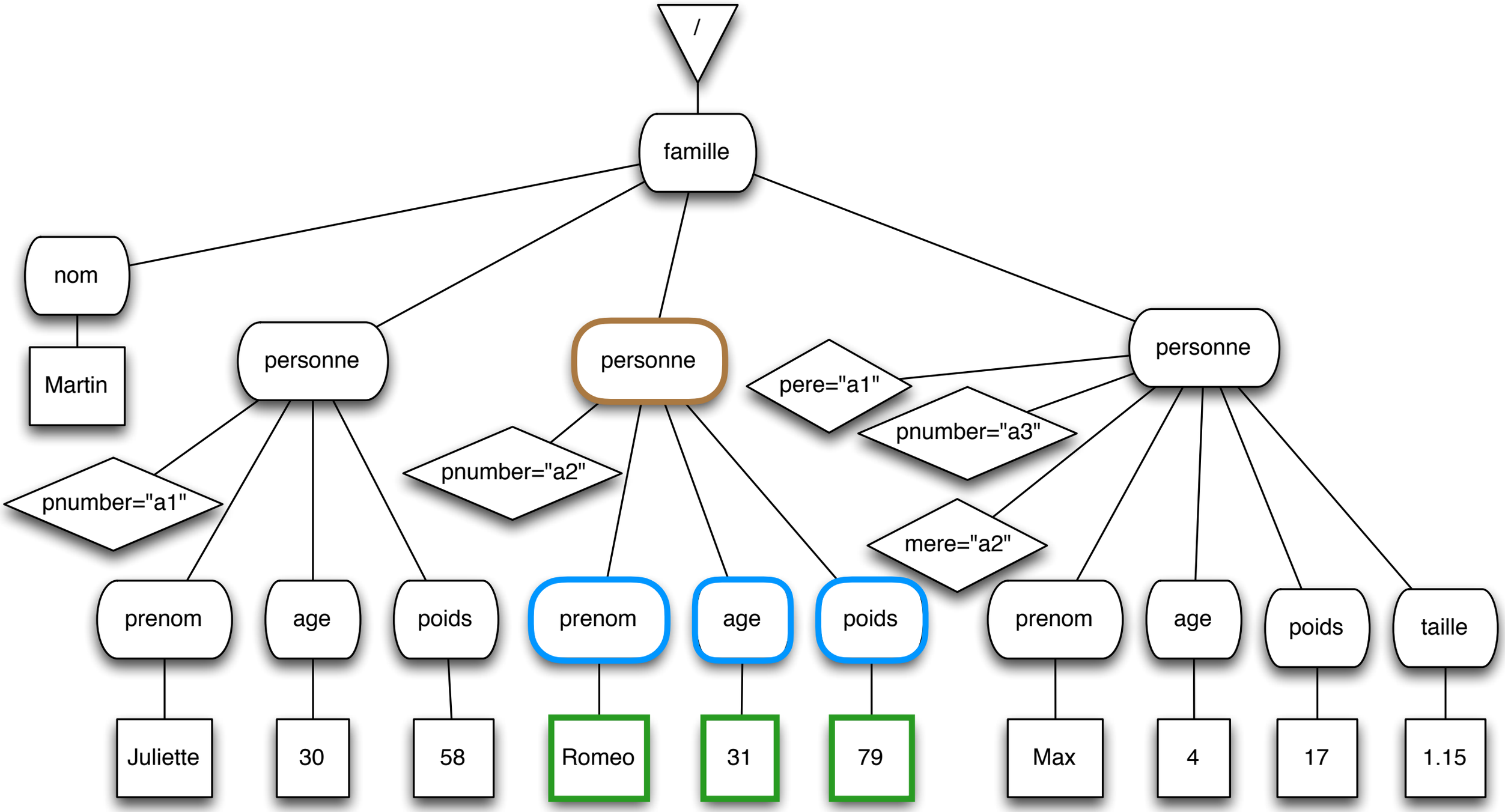
Les axes

- self le nœud courant lui-même
- child sélectionne les nœuds éléments fils du nœud courant
- attribute les attributs de l'élément courant
- **descendant les nœuds éléments descendants du nœud courant**

descendant::


/
famille
nom
Martin
personne
pnumber="a1"
prenom
Juliette
age
30
poids
58
personne
pnumber="a2"
prenom
Romeo
age
31
poids
79
text()
pere="a1"
pnumber="a3"
mere="a2"
personne
prenom
Max
age
4
poids
17
taille
1.15

descendant::


/
famille
nom
Martin
personne
pnumber="a1"
prenom
Juliette
age
30
poids
58
personne
pnumber="a2"
prenom
Romeo
age
31
poids
79
pere="a1"
pnumber="a3"
mere="a2"
personne
prenom
Max
age
4
poids
17
taille
1.15

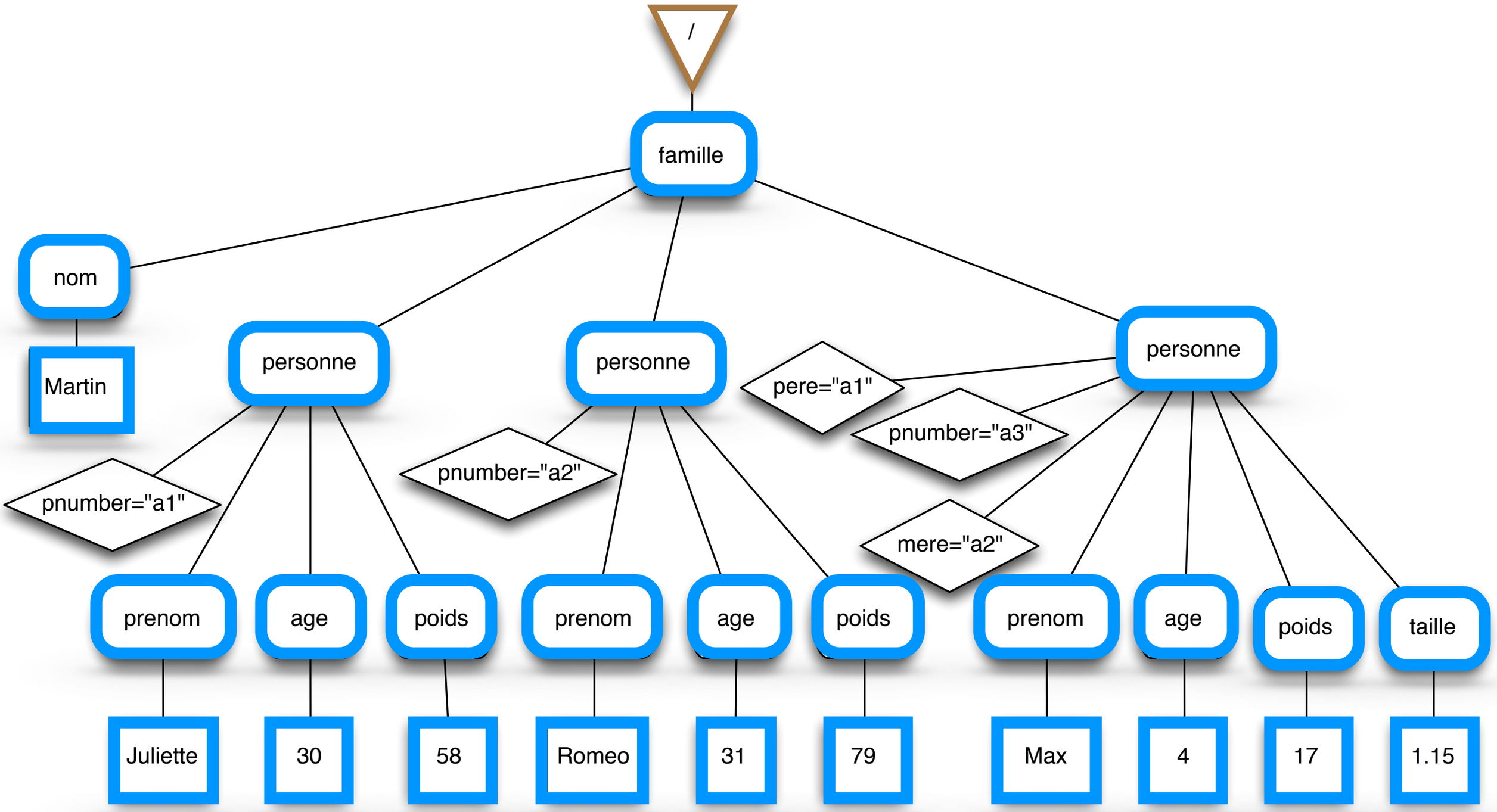
Ordre du document

Les requêtes XPath utilisent un ordre total sur les nœuds, appelé ordre du document.

- C'est l'ordre en profondeur d'abord et de gauche à droite lorsque le document est représenté par un arbre (c'est l'ordre SAX).
- Pour les éléments, c'est donc l'ordre dans lequel on rencontre les balises ouvrantes lorsque l'on considère la représentation textuelle du document.
- Les nœuds `namespace` sont immédiatement successeurs du nœud élément qui leur est associé.
- Les nœuds `attributs` viennent immédiatement après ces nœuds `namespace`.

  Attention, l'ordre des nœuds attributs d'un même élément dépend de l'implémentation.

descendant

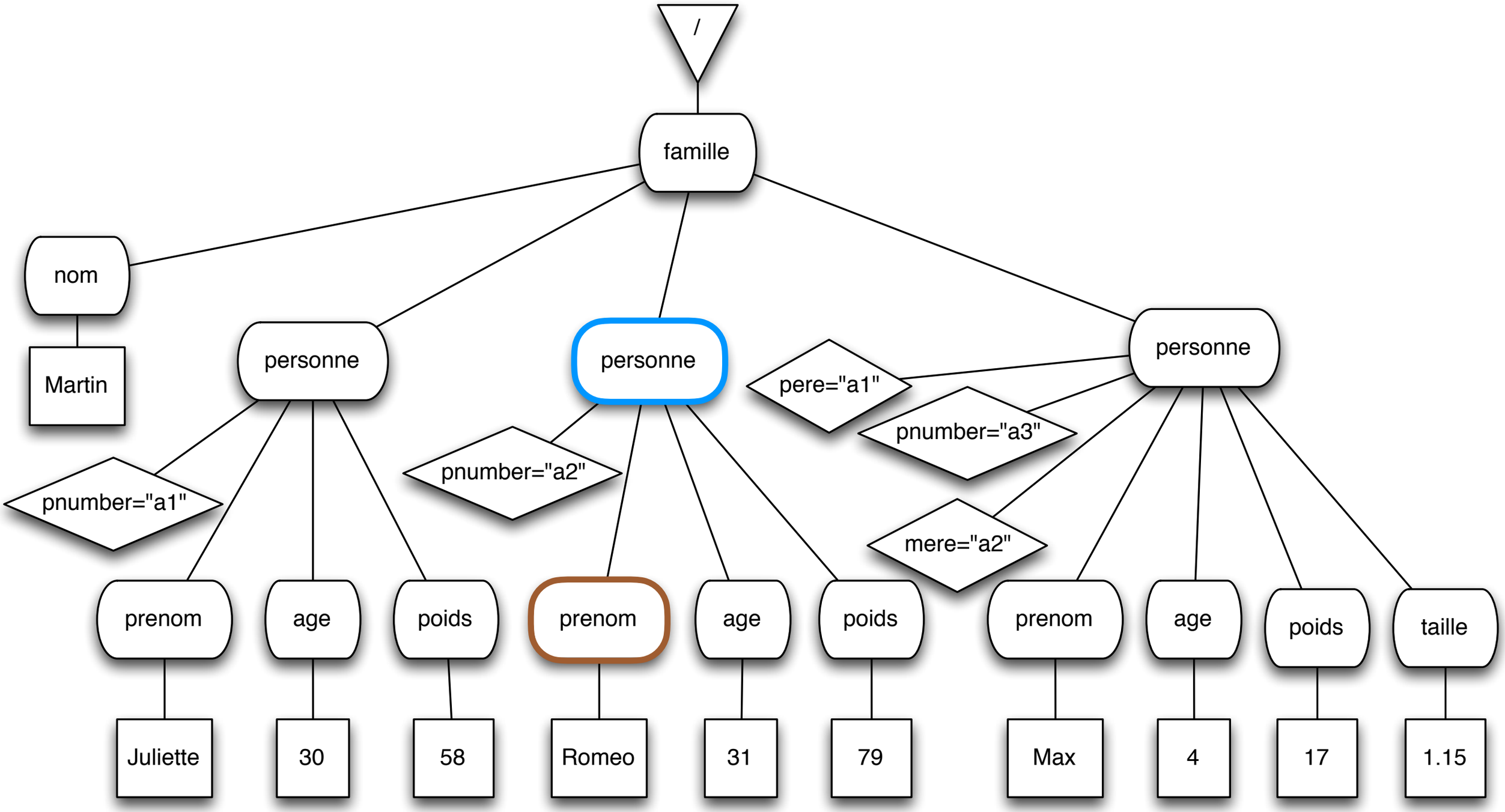


Attention, les nœuds attributs ne sont pas atteignables par l'axe descendant

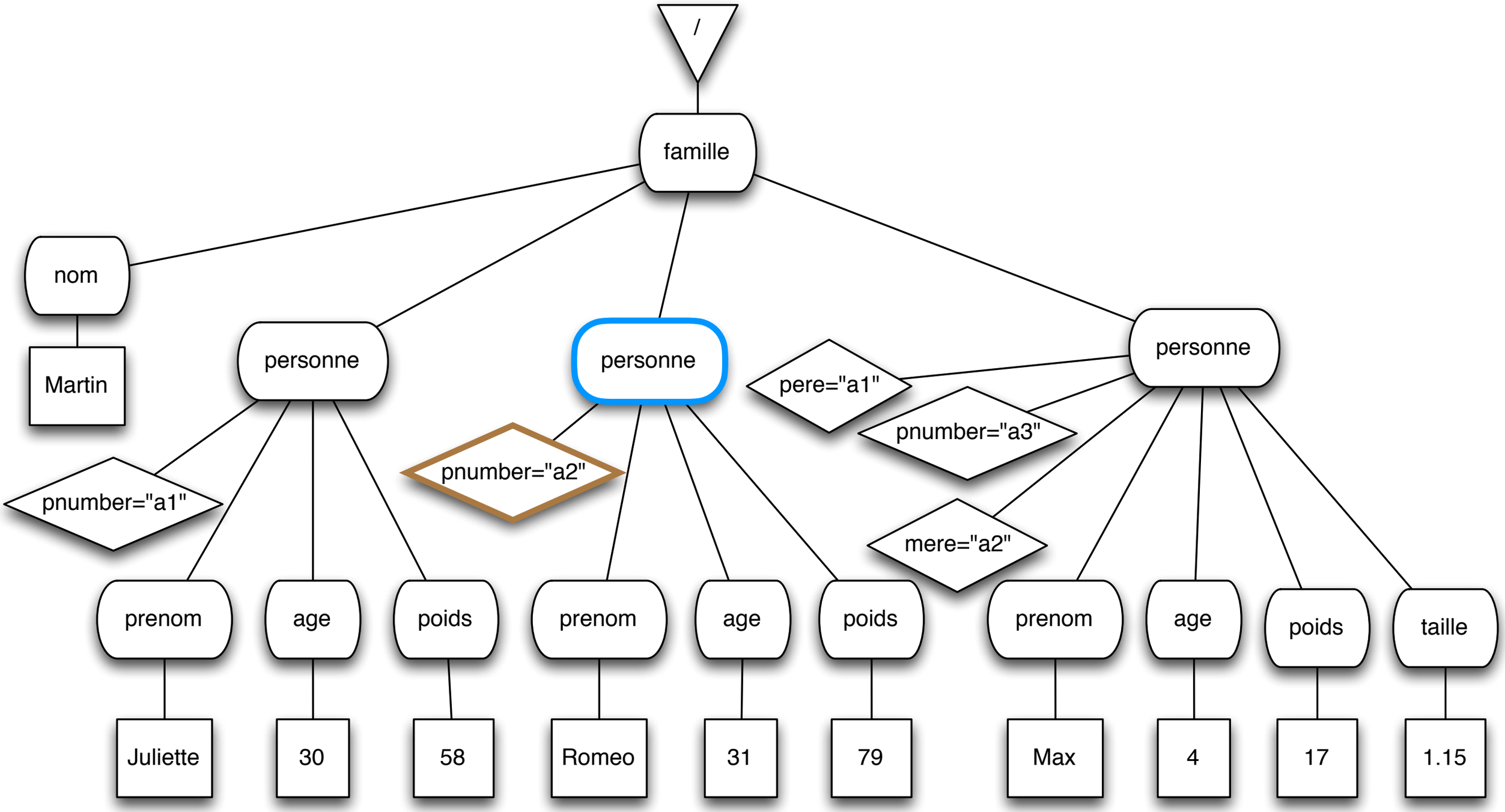
Les axes

- self le nœud courant lui-même
- child sélectionne les nœuds éléments fils du nœud courant
- attribute les attributs de l'élément courant
- descendant les nœuds éléments descendants du nœud courant
- **parent le nœud père du nœud courant**

parent::


/
famille
nom
Martin
personne
pnumber="a1"
prenom
Juliette
age
30
poids
58
personne
pnumber="a2"
prenom
Romeo
age
31
poids
79
pere="a1"
pnumber="a3"
mere="a2"
personne
prenom
Max
age
4
poids
17
taille
1.15

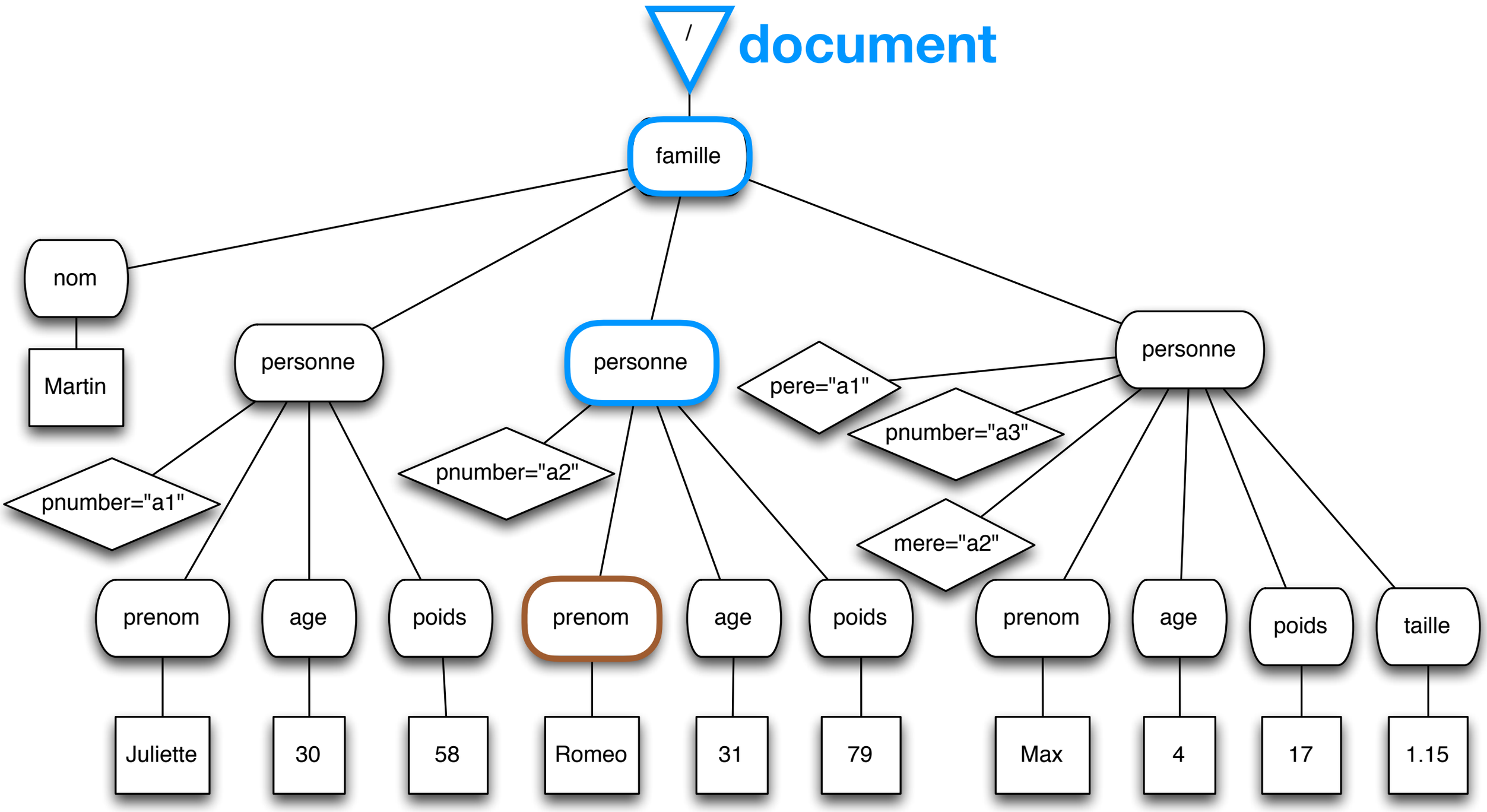
parent::


/
famille
nom
Martin
personne
pnumber="a1"
prenom
Juliette
age
30
poids
58
personne
pnumber="a2"
prenom
Romeo
age
31
poids
79
pere="a1"
pnumber="a3"
mere="a2"
personne
prenom
Max
age
4
poids
17
taille
1.15

Les axes

- self le nœud courant lui-même
- child sélectionne les nœuds éléments fils du nœud courant
- attribute les attributs de l'élément courant
- descendant les nœuds éléments descendants du nœud courant
- parent le nœud père du nœud courant
- **ancestor les éléments ancêtres**

ancestor::

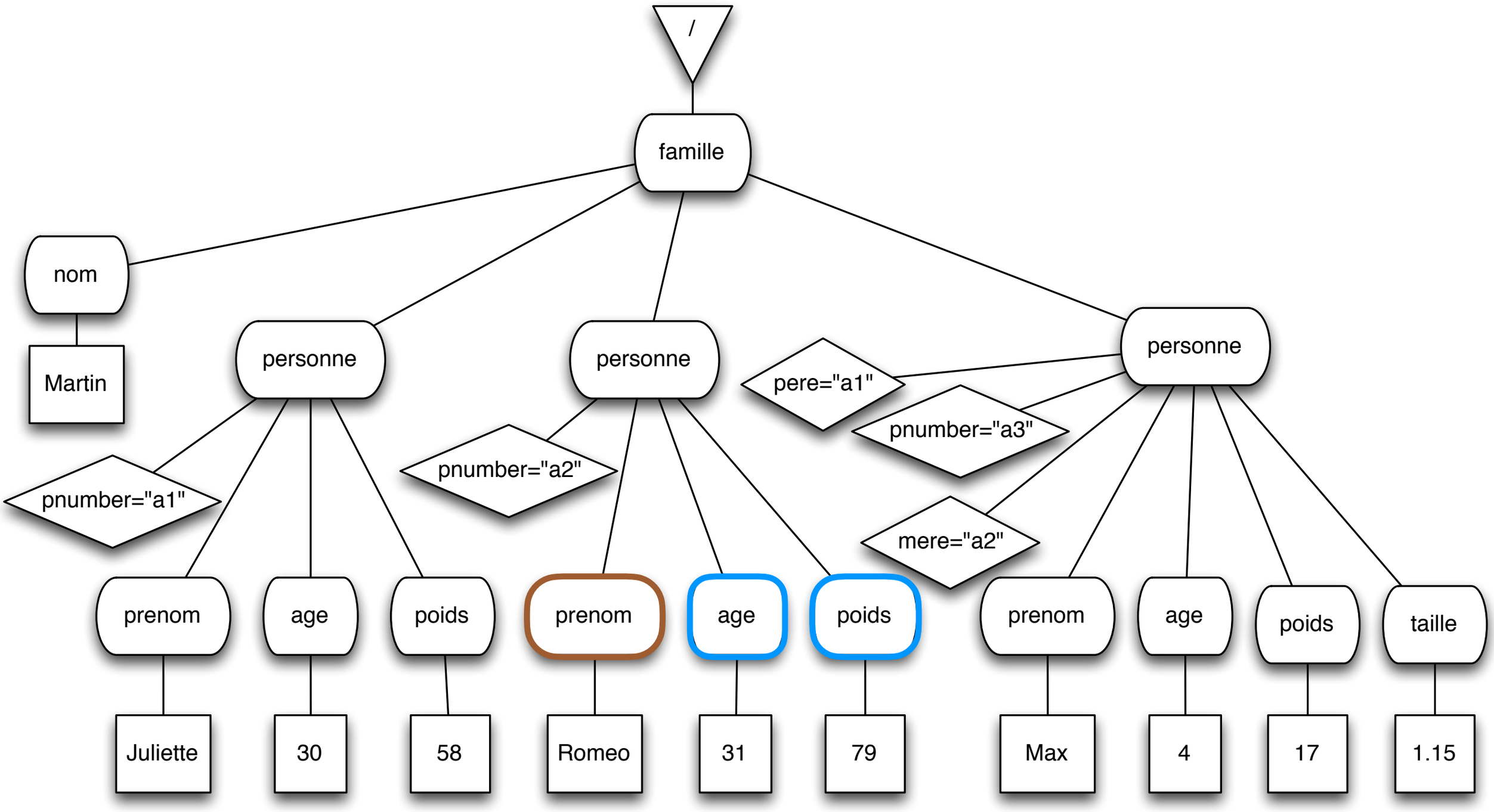


Attention, c'est toujours l'ordre du document qui prévaut : /, famille, personne

Les axes

- self le nœud courant lui-même
- child sélectionne les nœuds éléments fils du nœud courant
- attribute les attributs de l'élément courant
- descendant les nœuds éléments descendants du nœud courant
- parent le nœud père du nœud courant
- ancestor les éléments ancêtres
- **following-sibling (preceding-sibling) les éléments frères droits (les éléments frères gauches)**

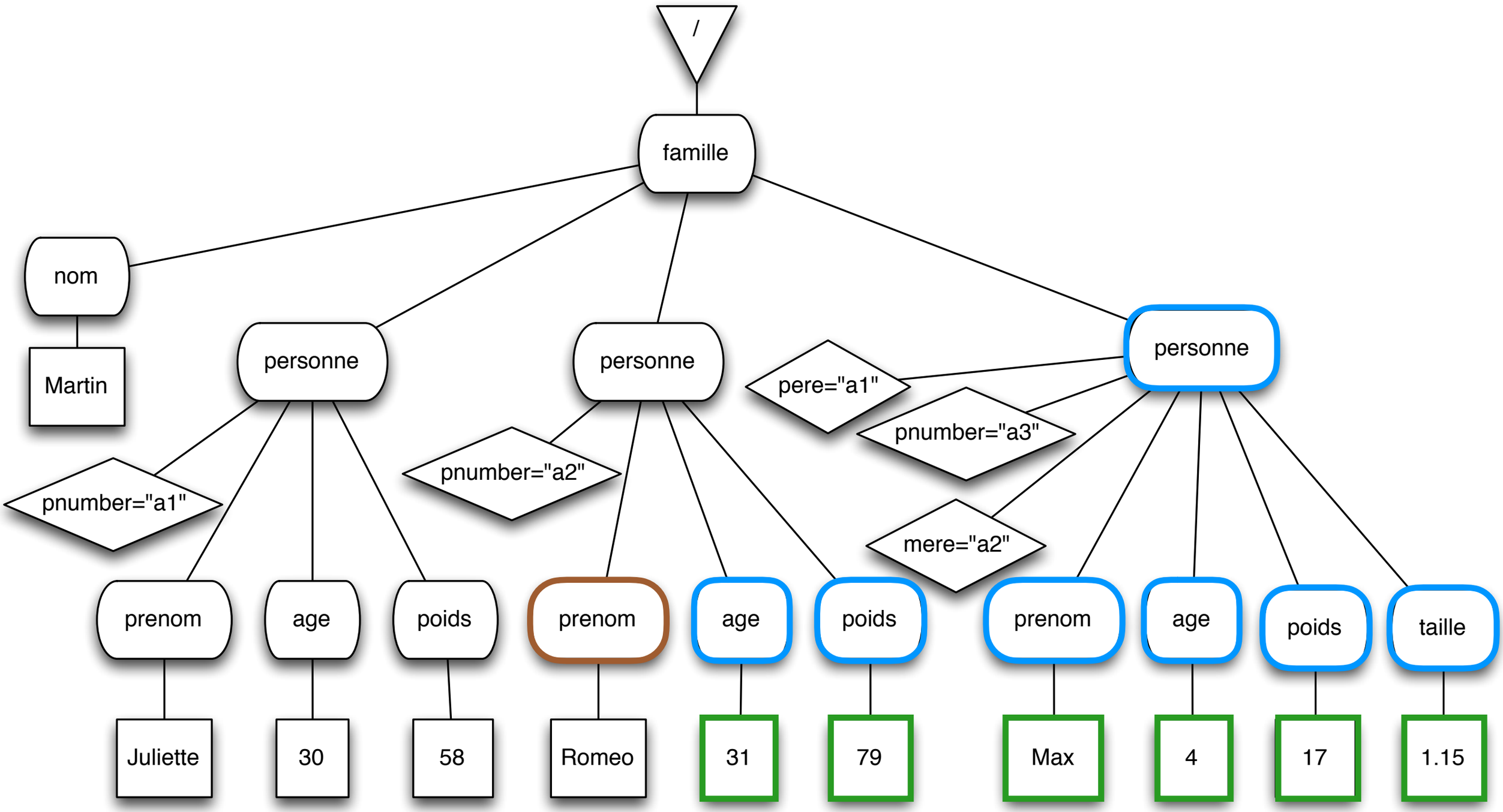
following-sibling::



Les axes

- self le nœud courant lui-même
- child sélectionne les nœuds éléments fils du nœud courant
- attribute les attributs de l'élément courant
- descendant les nœuds éléments descendants du nœud courant
- parent le nœud père du nœud courant
- ancestor les éléments ancêtres
- following-sibling (preceding-sibling) les éléments frères droits (les éléments frères gauches)
- **following (preceding) les éléments dont la balise ouvrante (fermante) apparaît après (avant) dans le document**

following::


/
famille
nom
Martin
personne
pnumber="a1"
prenom
Juliette
age
30
poids
58
personne
pnumber="a2"
prenom
Romeo
age
31
poids
79
pere="a1"
pnumber="a3"
mere="a2"
personne
prenom
Max
age
4
poids
17
taille
1.15

Les axes

- self le nœud courant lui-même
- child sélectionne les nœuds éléments fils du nœud courant
- attribute les attributs de l'élément courant
- descendant les nœuds éléments descendants du nœud courant
- parent le nœud père du nœud courant
- ancestor les éléments ancêtres
- following-sibling (preceding-sibling) les éléments frères droits (les éléments frères gauches)
- following (preceding) les éléments dont la balise ouvrante (fermante) apparaît après (avant) dans le document
- namespace les espaces de noms
- ancestor-or-self, descendant-or-self

Les étapes

Une étape est composée de 3 composants : un axe, un filtre et une liste éventuellement vide de prédicats en suivant la syntaxe

```
axe::filtre[prédicat1][prédicat2]...
```

axe sens de navigation dans l'arbre par rapport au nœud contexte

filtre type des nœuds à retenir

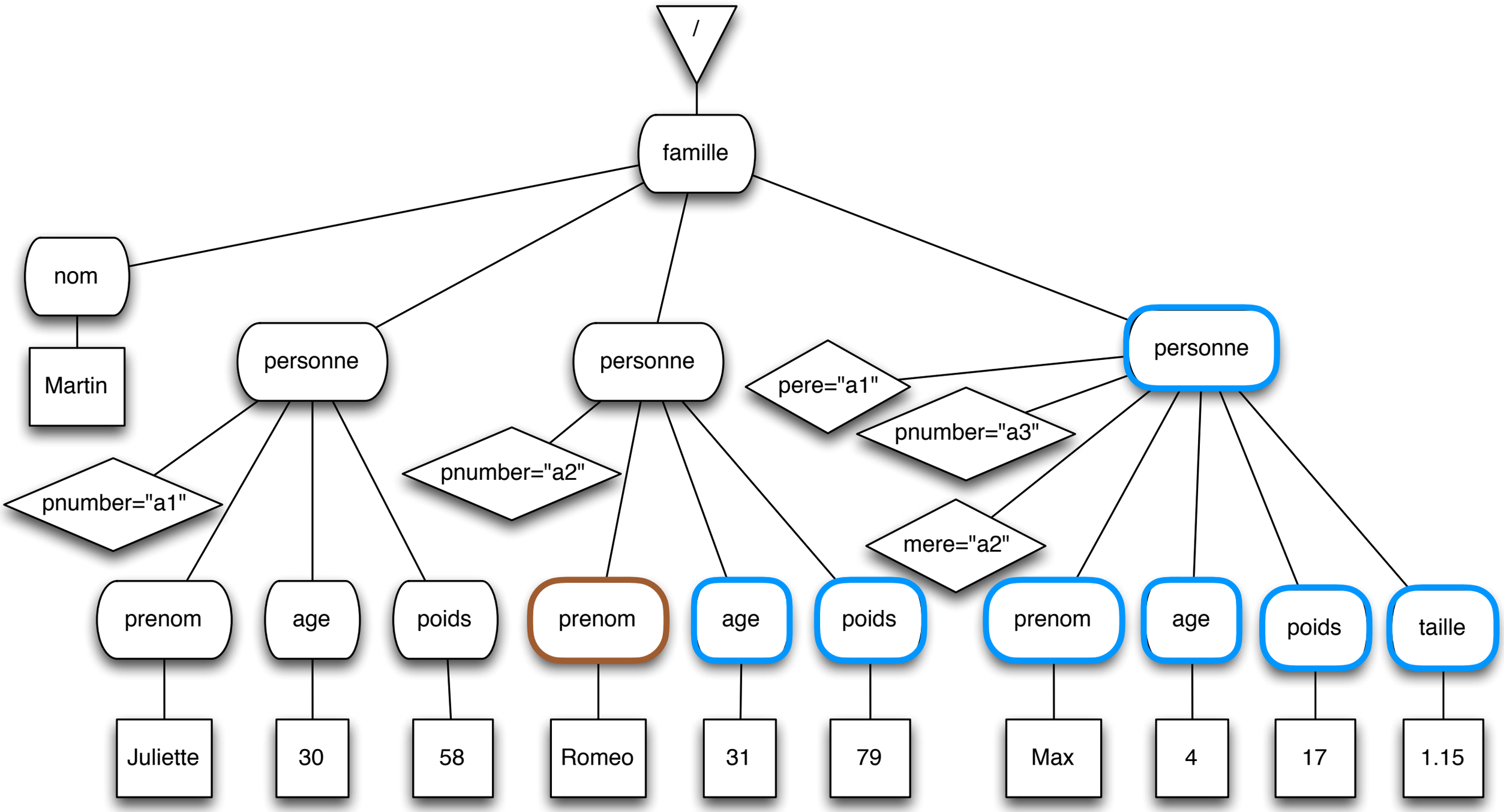
prédicat(s) propriétés que doivent satisfaire les nœuds parmi les nœuds retenus.

Les filtres (node tests)

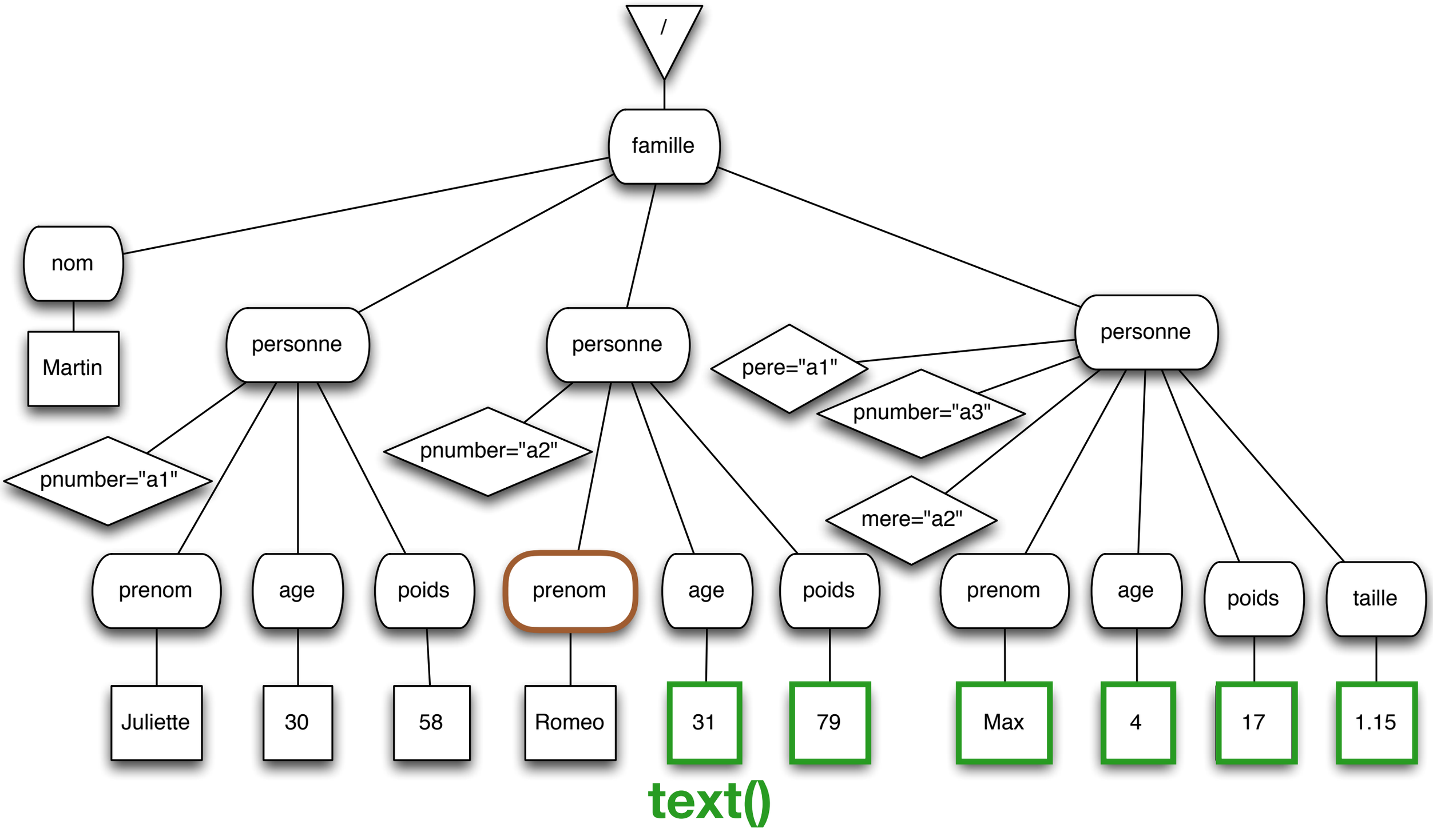
Un filtre permet de sélectionner par un nom ou par un type : un filtre peut être

- un nom d'élément ou d'attribut
- le caractère * qui sélectionne tous les objets de même nature (mais uniquement élément ou attribut).
  - child::* sélectionne tous les éléments fils du nœud courant,
  - attribute::* sélectionne tous les attributs du nœud courant.
- comment() qui sélectionne les nœuds de type commentaire
- text() qui sélectionne les nœuds de type texte
- node() qui sélectionne les nœuds de n'importe quel type.
- processing-instruction() ...

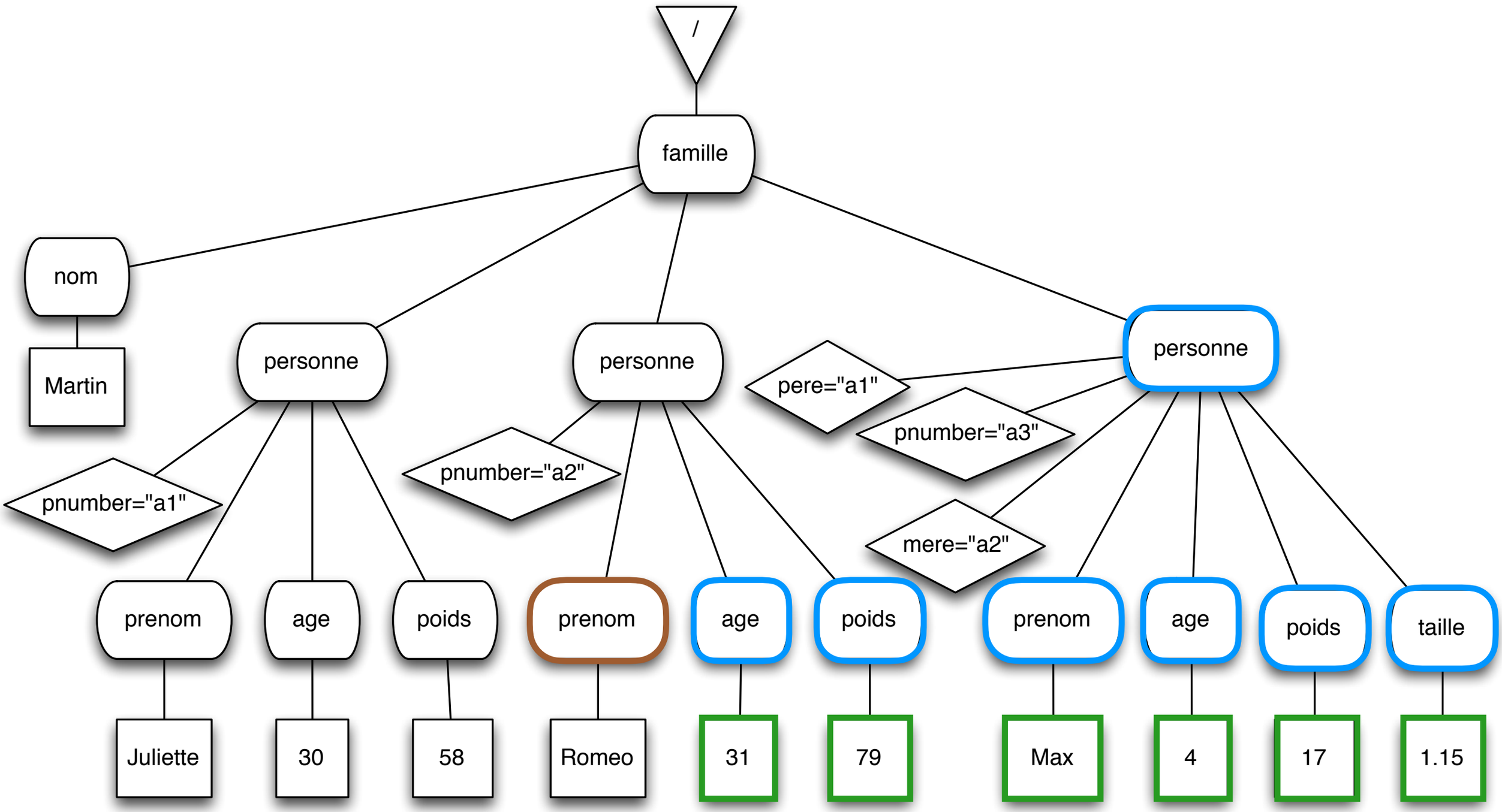
following::*


/
famille
nom
Martin
personne
pnumber="a1"
prenom
Juliette
age
30
poids
58
personne
pnumber="a2"
prenom
Romeo
age
31
poids
79
pere="a1"
pnumber="a3"
mere="a2"
personne
prenom
Max
age
4
poids
17
taille
1.15

following::text()



following::node()



Les étapes

Une étape est composée de 3 composants : un axe, un filtre et une liste éventuellement vide de prédicats en suivant la syntaxe

`axe::filtre[`***`prédicat1`***`][`***`prédicat2`***`]...`

axe sens de navigation dans l'arbre par rapport au nœud contexte

filtre type des nœuds à retenir

prédicat(s) propriétés que doivent satisfaire les nœuds parmi les nœuds retenus.

Les prédicats

Un prédicat est une expression booléenne qui peut être évaluée à vrai ou faux. On dispose des connecteurs logiques `and` et `or` et on peut faire intervenir dans les expressions des valeurs de type :

- numérique
- chaîne de caractères
- booléens (`true` et `false`)
- liste

Des règles de conversion s'appliquent pour pouvoir convertir toute expression XPath en valeur booléenne en fonction de ce qu'elle retourne.

Valeurs numériques

- on dispose des comparaisons habituelles `<` , `>` , `<=` , `>=` , `!=` , `=`
    - `XPath2.0` pour valeurs atomiques `eq, lt, gt, le, ge`
- on dispose des opérations `+`, `-`, `*`, `/`, `div`, `mod`
- la fonction `number(arg)` permet de tenter une conversion si la conversion échoue, on obtient la valeur `NaN`

```
/descendant::personne[child::age > 20]
```

```
/descendant::personne[number(child::age/text()) gt 20]
```

Règles de conversion en booléens

- une liste vide vaut `false` sinon `true`
- une chaîne vide vaut `false` sinon `true`
- `0` ou `NaN` valent `false`, les autres valeurs numériques valent `true`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE famille SYSTEM "famille.dtd">
<famille>
    <nom>Martin</nom>
    <personne pnumber="a1">
        <prenom>Juliette</prenom>
        <age>30</age>
        <poids>58</poids>
    </personne>
    <personne pnumber="a2">
        <prenom>Romeo</prenom>
        <age>31</age>
        <poids>79</poids>
    </personne>
    <personne pnumber="a3" mere="a1" pere="a2">
        <prenom>Max</prenom>
        <age>4</age>
        <poids>17</poids>
        <taille>1.15</taille>
    </personne>
</famille>
```

`/descendant::*[attribute::pere]` liste des éléments qui ont un attribut `pere`.

`/descendant::personne[not(attribute::pere)]` liste des personnes qui n'ont pas d'attribut `pere`.

`/descendant::personne[attribute::pnumber="a2"]` l'élément `personne` dont l'attribut `pnumber` vaut `"a2"`

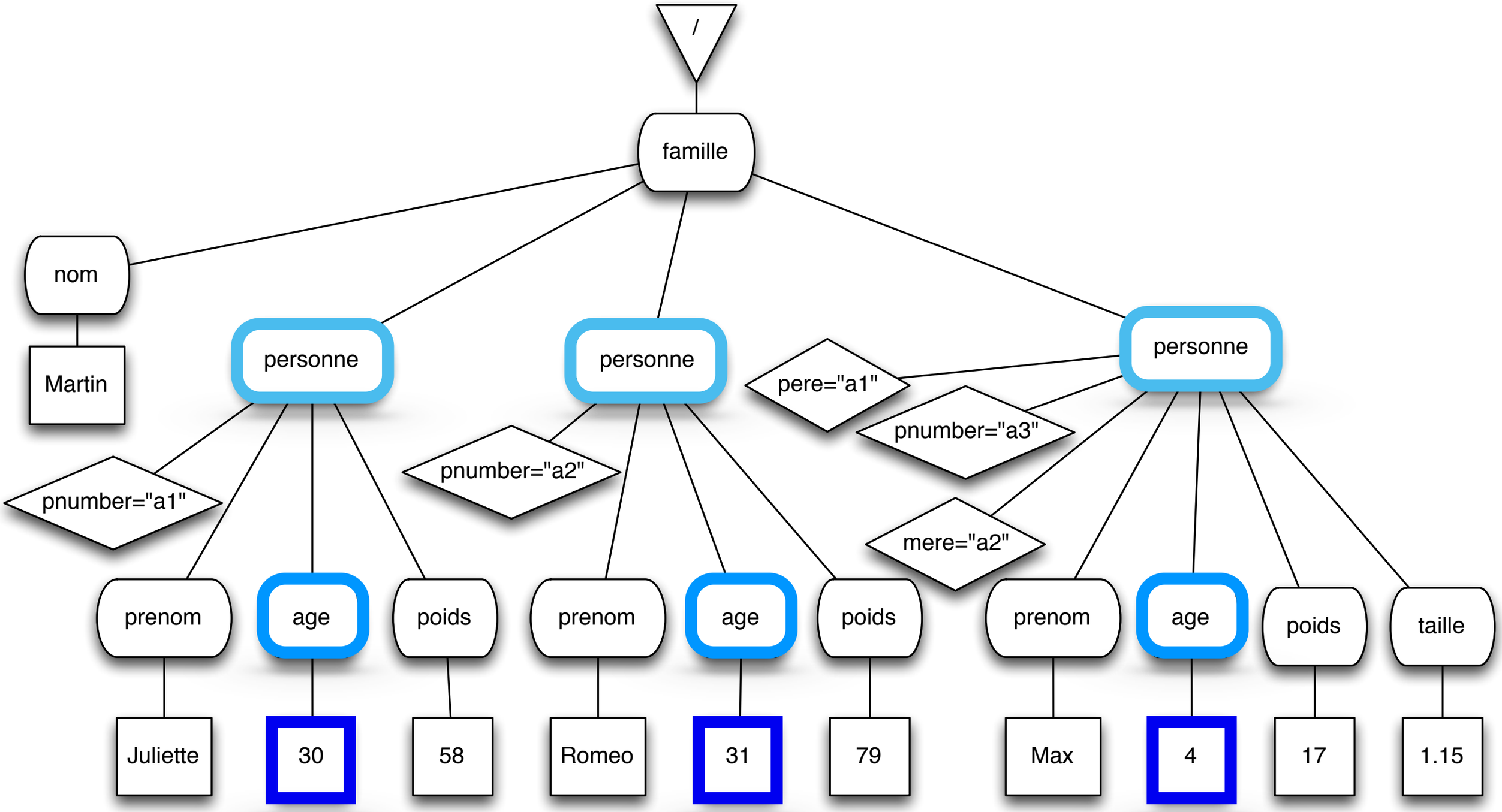
`/descendant::*[attribute::pnumber="a2" or attribute::pnumber="a1"]` Romeo et Juliette

Fonctions (communes à XPath 1.0 et XPath 2.0)

Voir `http://xmlfr.org/w3c/TR/xpath/#corelib`

- `id("valeur")` retourne l'élément dont l'attribut de type ID vaut `"valeur"`. On peut passer une séquence de valeurs en paramètre, le résultat est la séquence des éléments correspondants.
- `position()` retourne la position du nœud contexte dans la séquence résultat courante. La première position vaut `1`, la dernière vaut `last()`.
- `count(expression)` retourne le nombre d'éléments dans le résultat de l'évaluation de l'expression.
- `not(expression)` négation logique

```
/descendant::personne/child::age/child::text()
```



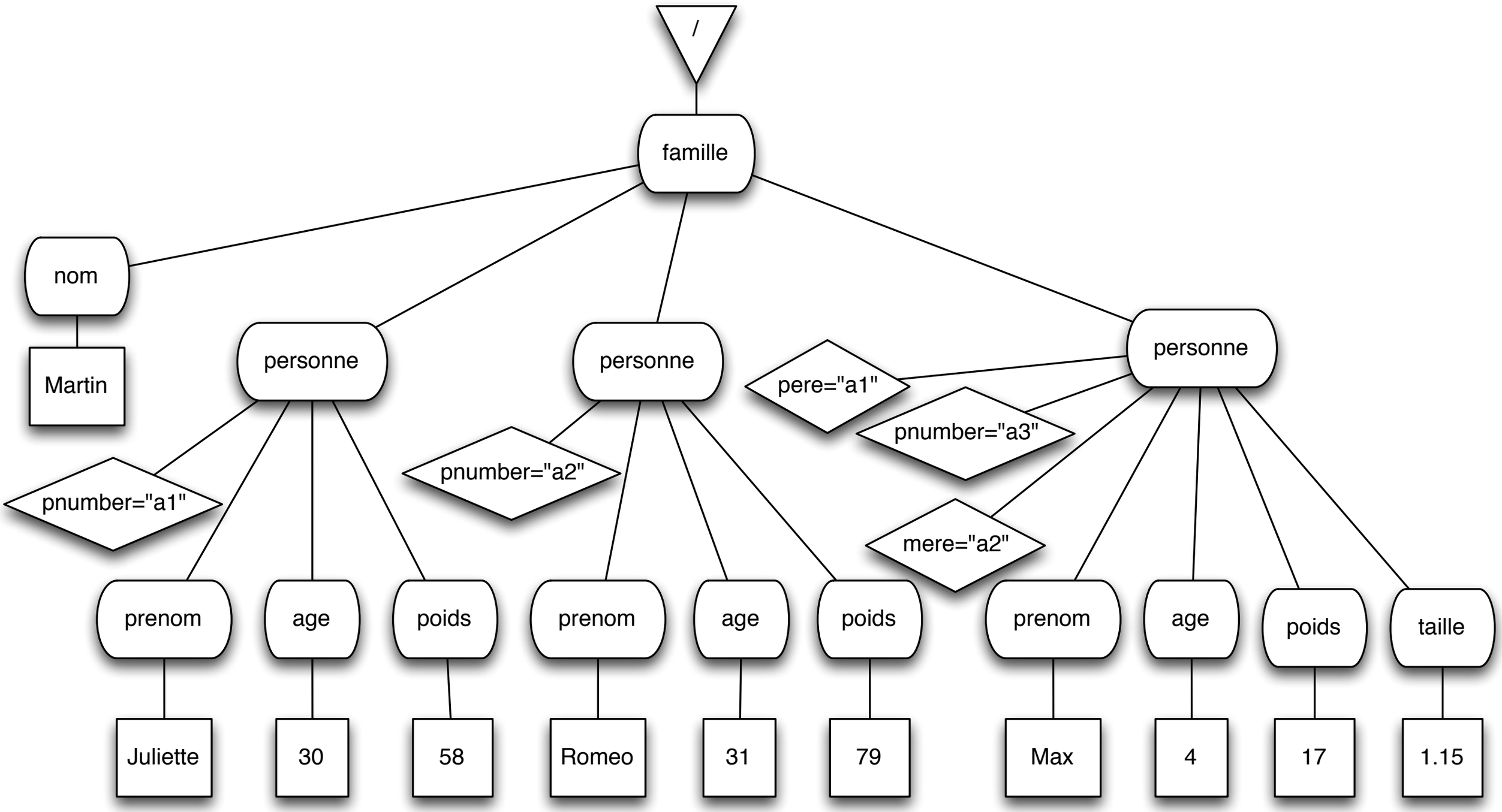
Évaluation d'une expression XPath

- à partir du nœud contexte, on évalue la première étape, on obtient alors une liste d'items (en XPath 1, un node-set).

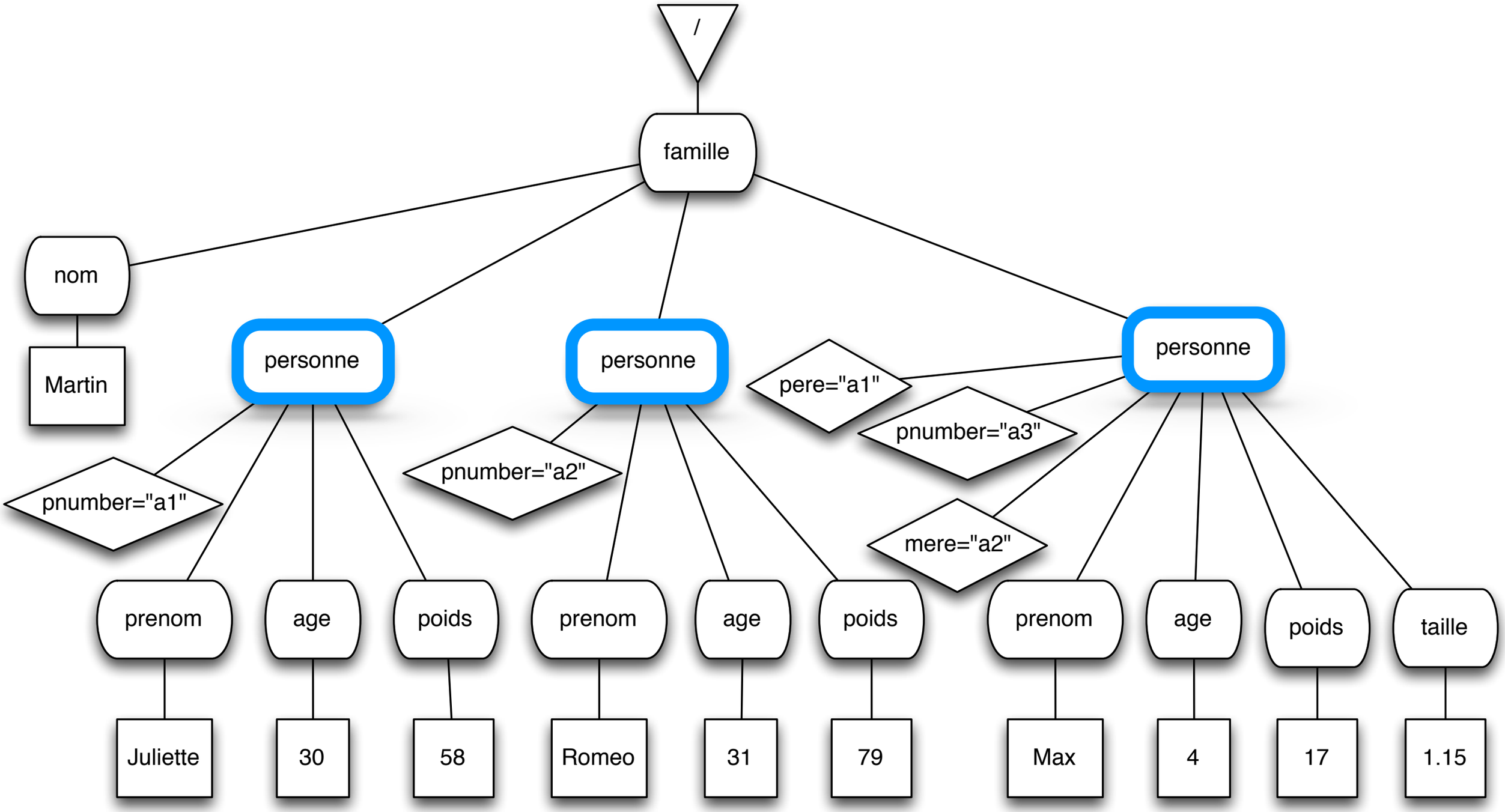
- on prend alors, un par un, les items de cette liste qui servent chacun leur tour de nœud contexte pour la deuxième étape.

- à chaque nouvelle étape *i*, la liste courante résultat S_i de cette étape remplace la liste résultat S_{i-1} qu'on avait à l'étape précédente en appliquant la nouvelle étape à chacun des items de S_{i-1} et en faisant la concaténation de toutes les listes obtenues.

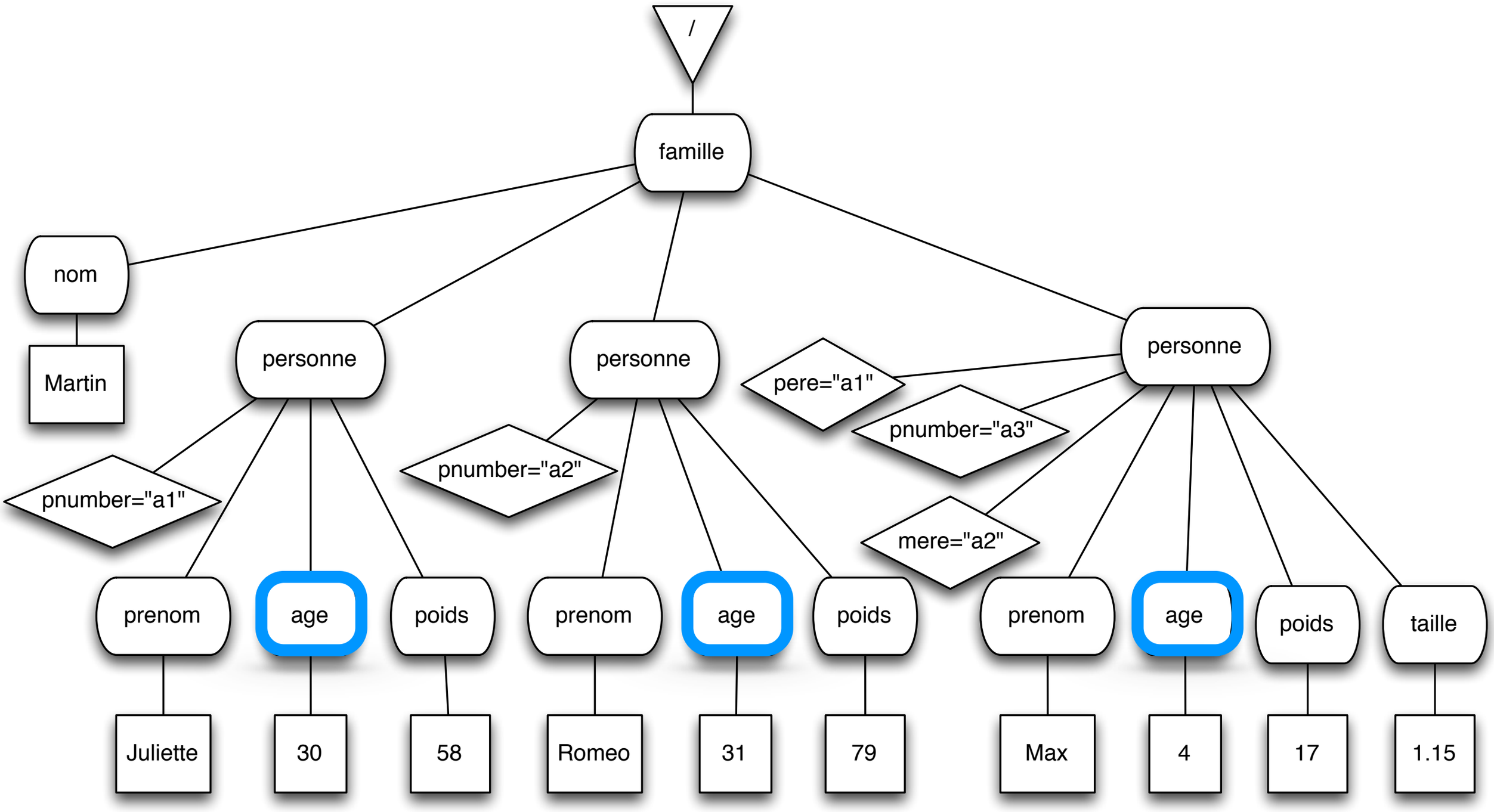
```
/descendant::personne/child::age/child::text()
```



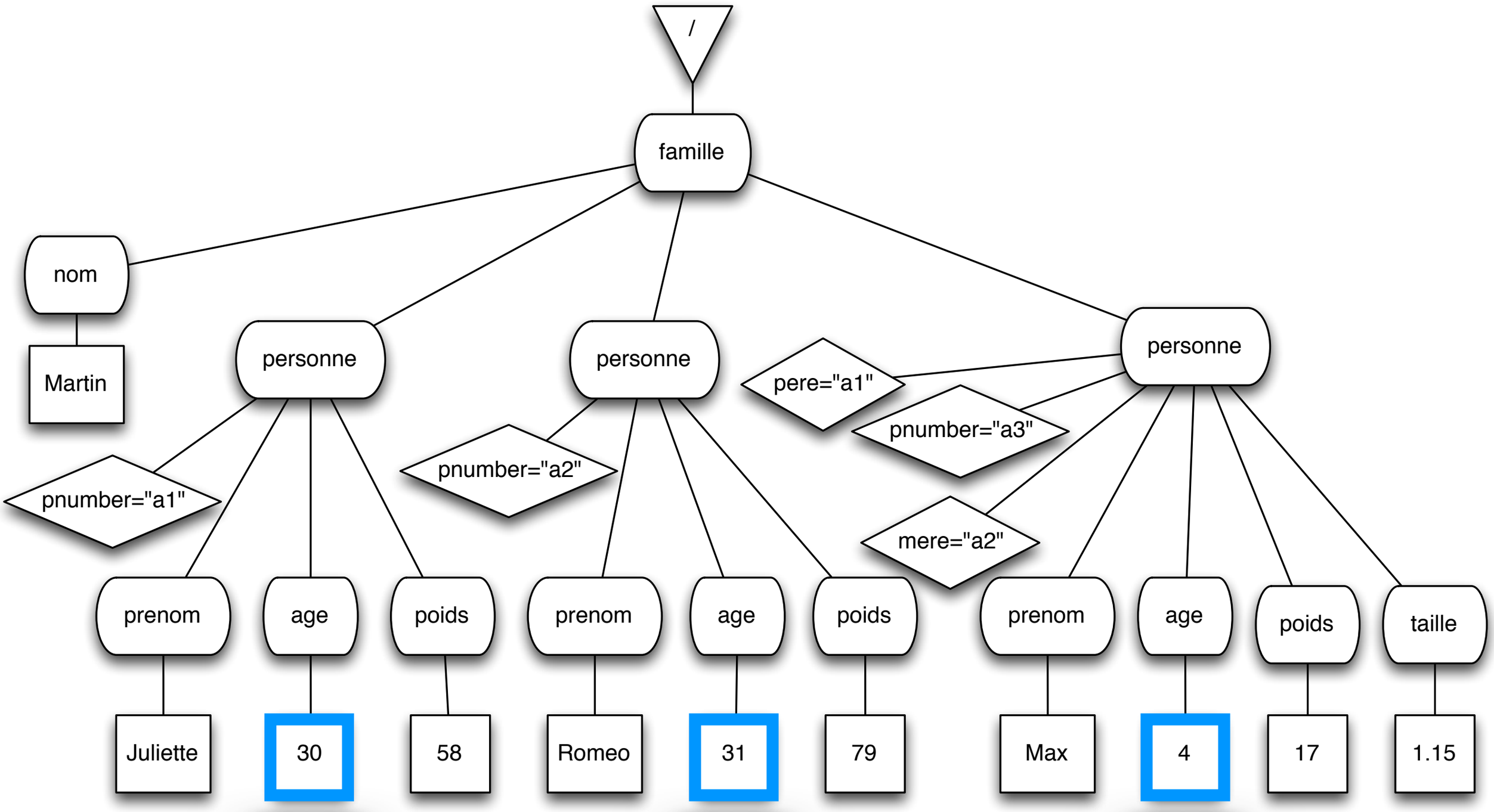
```
/descendant::personne/child::age/child::text()
```



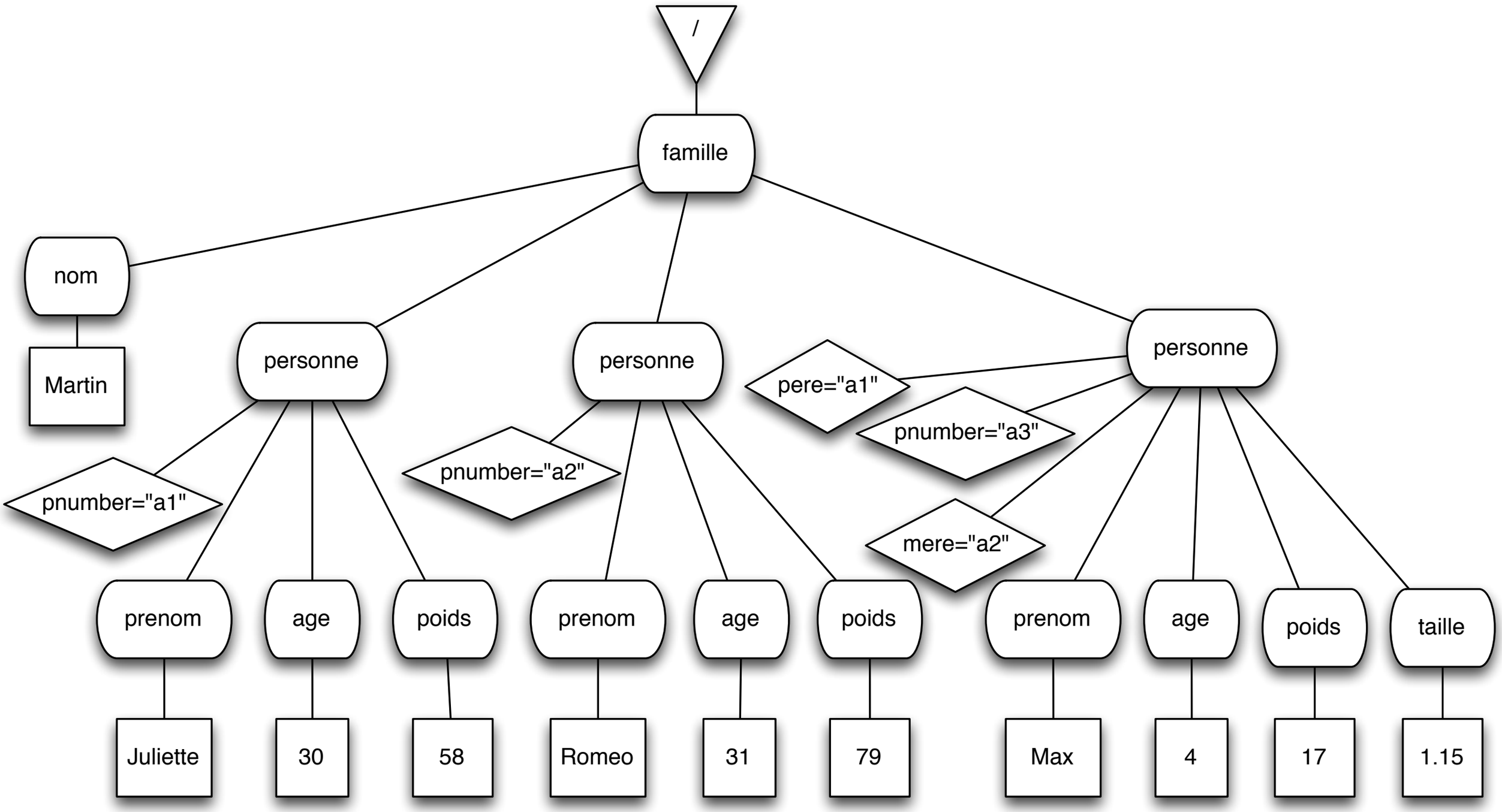
```
/descendant::personne/child::age/child::text()
```



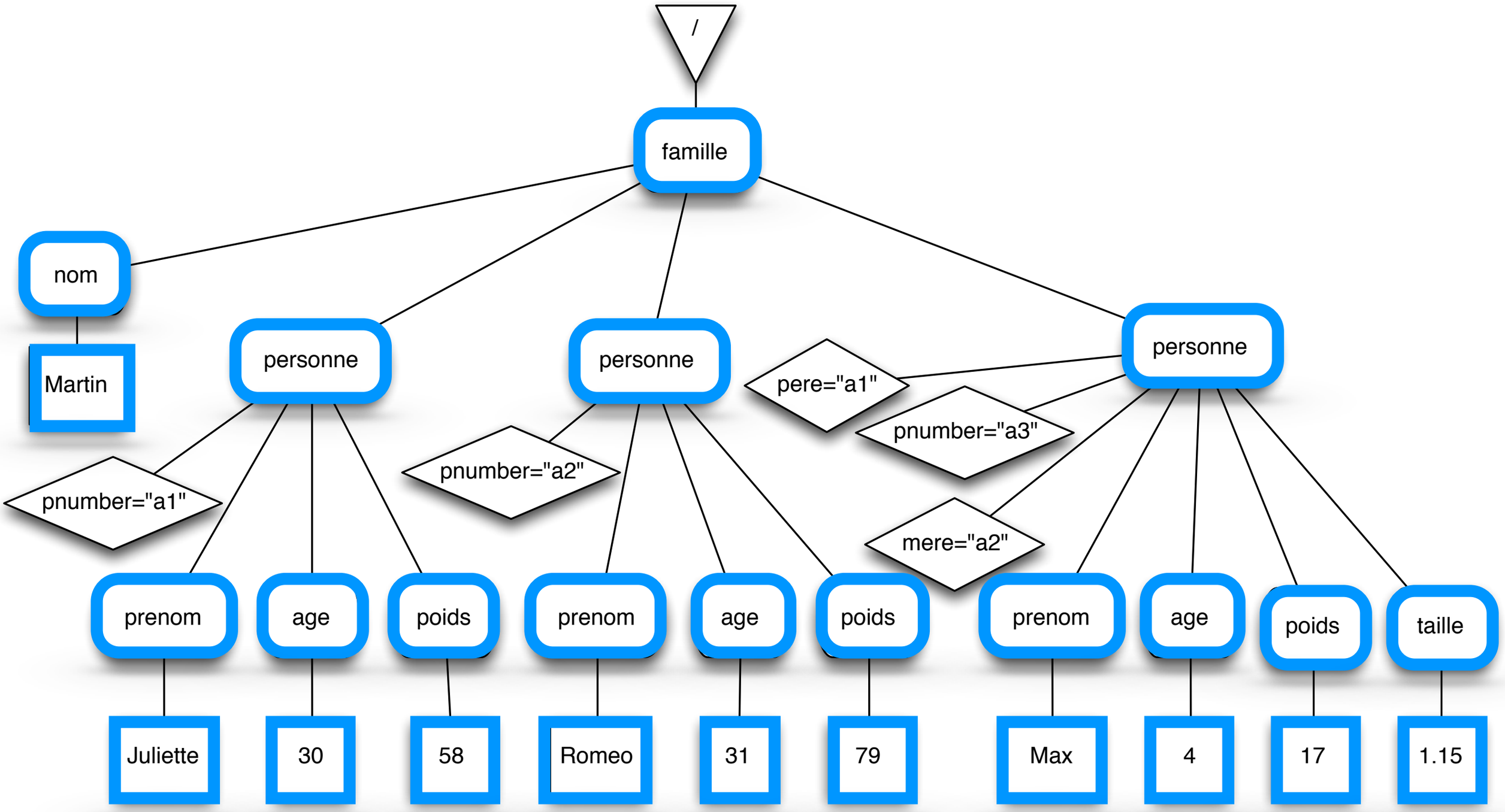
```
/descendant::personne/child::age/child::text()
```



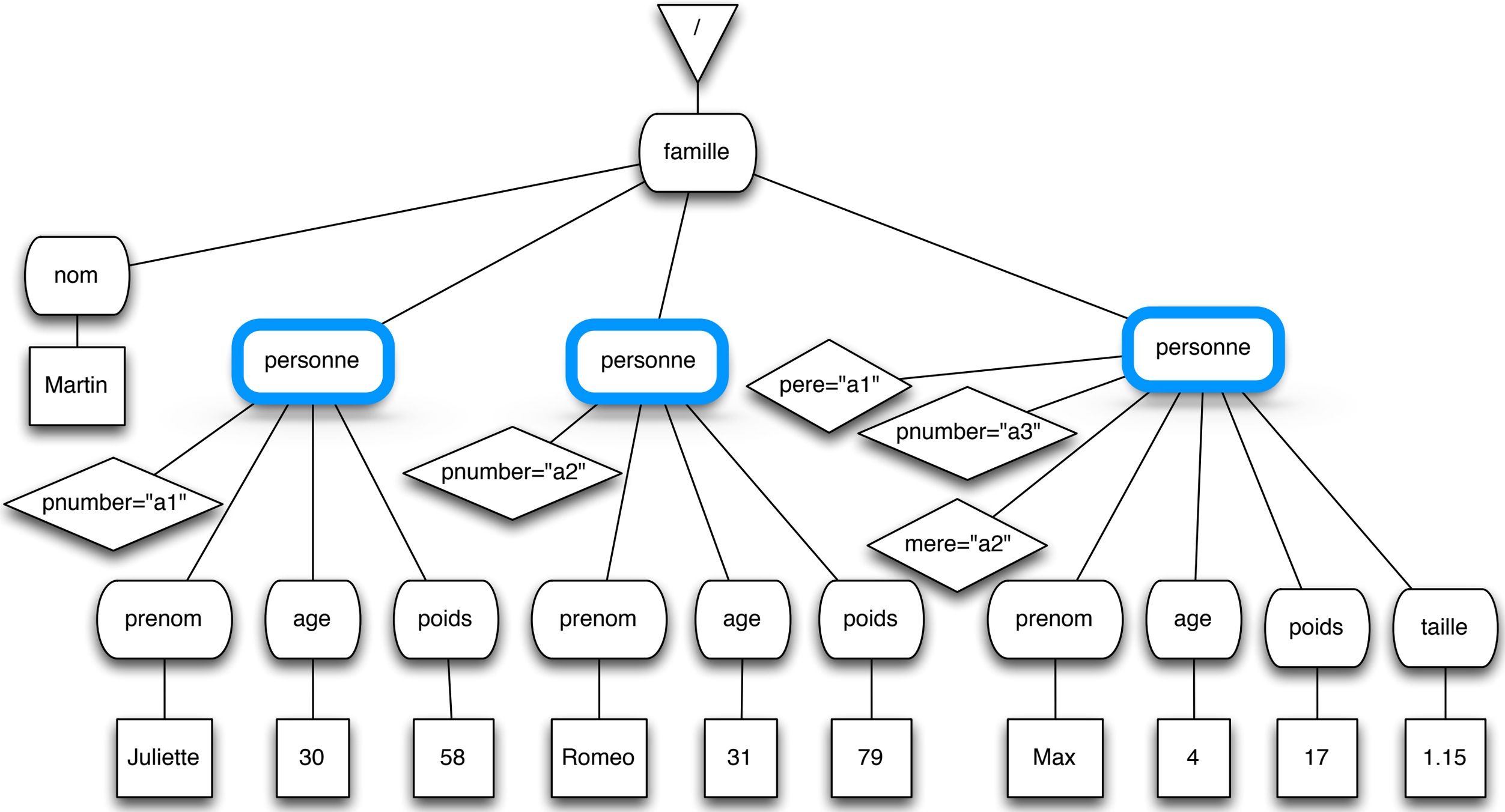
```
/descendant::personne/child::age/child::text()
```



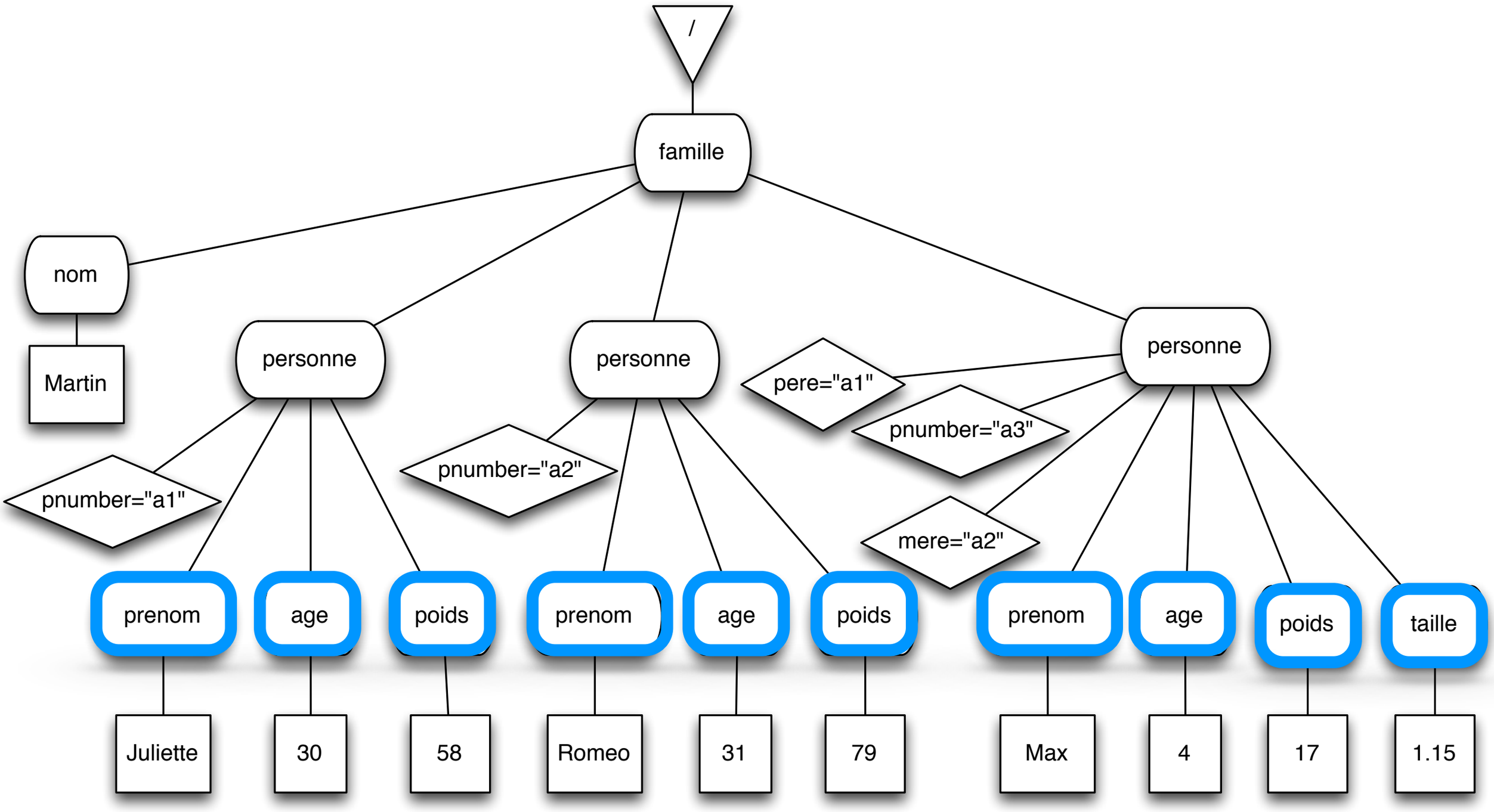
```
/descendant::personne/child::age/child::text()
```



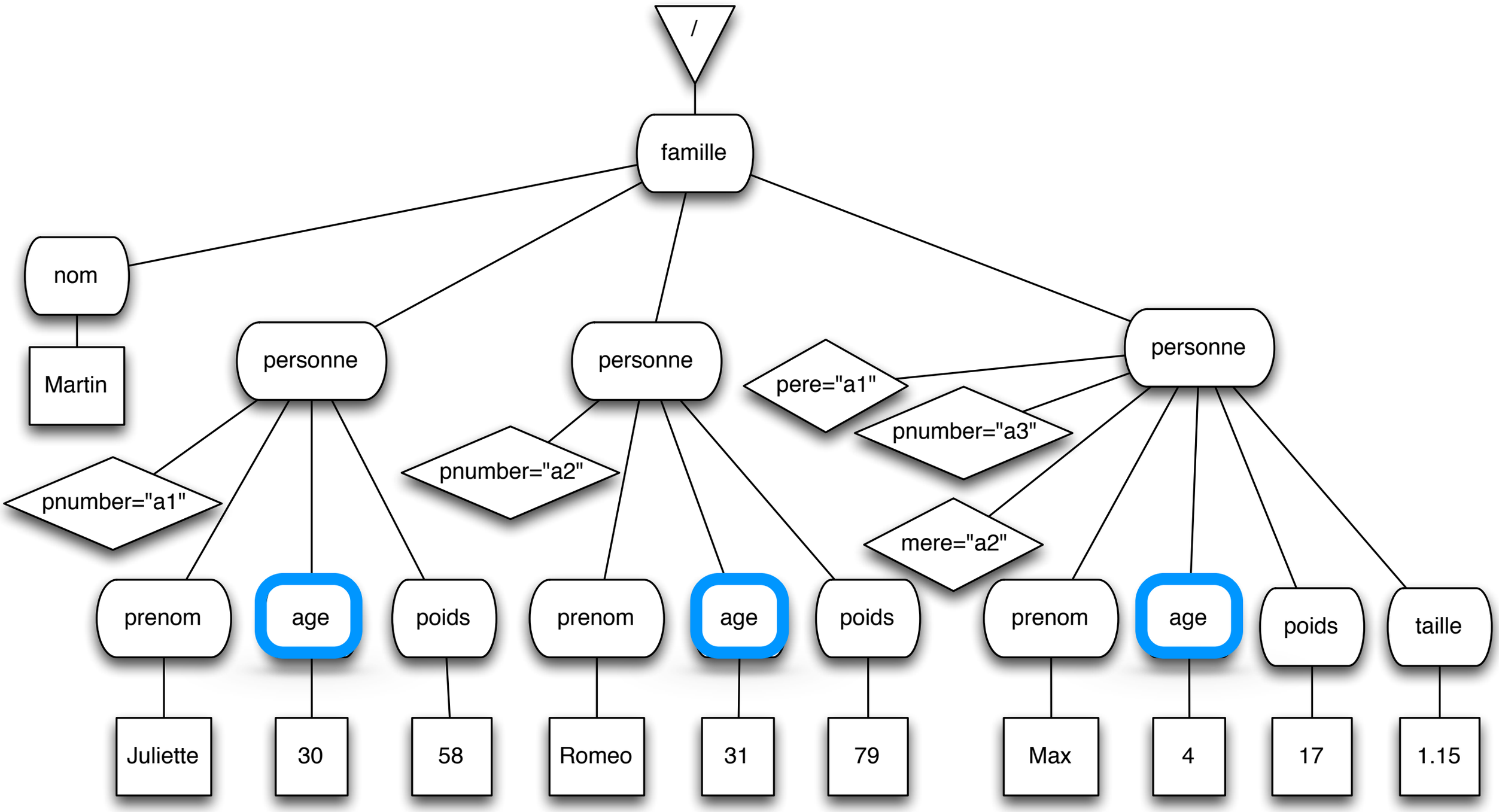
```
/descendant::personne/child::age/child::text()
```



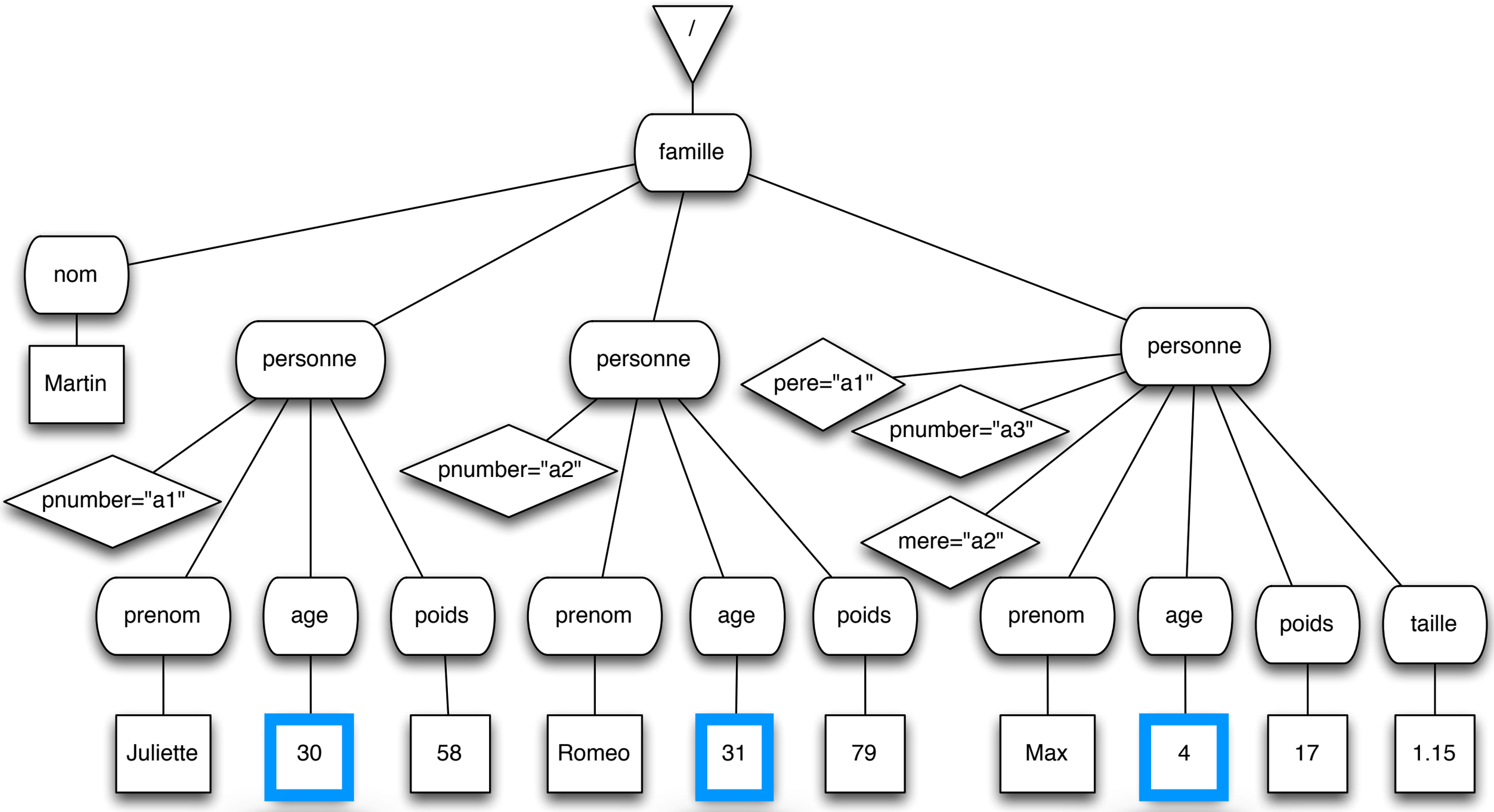
```
/descendant::personne/child::age/child::text()
```



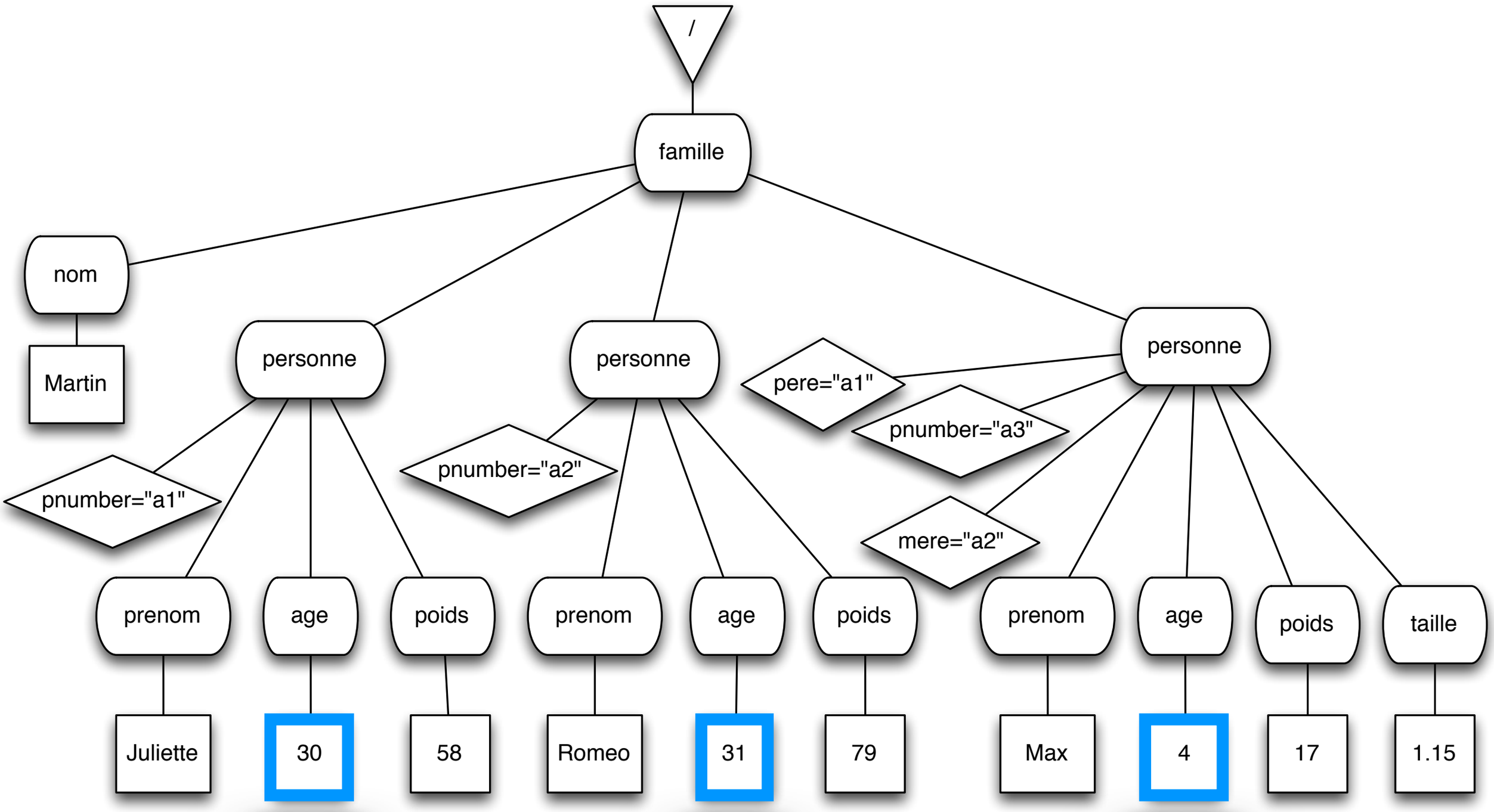
```
/descendant::personne/child::age/child::text()
```



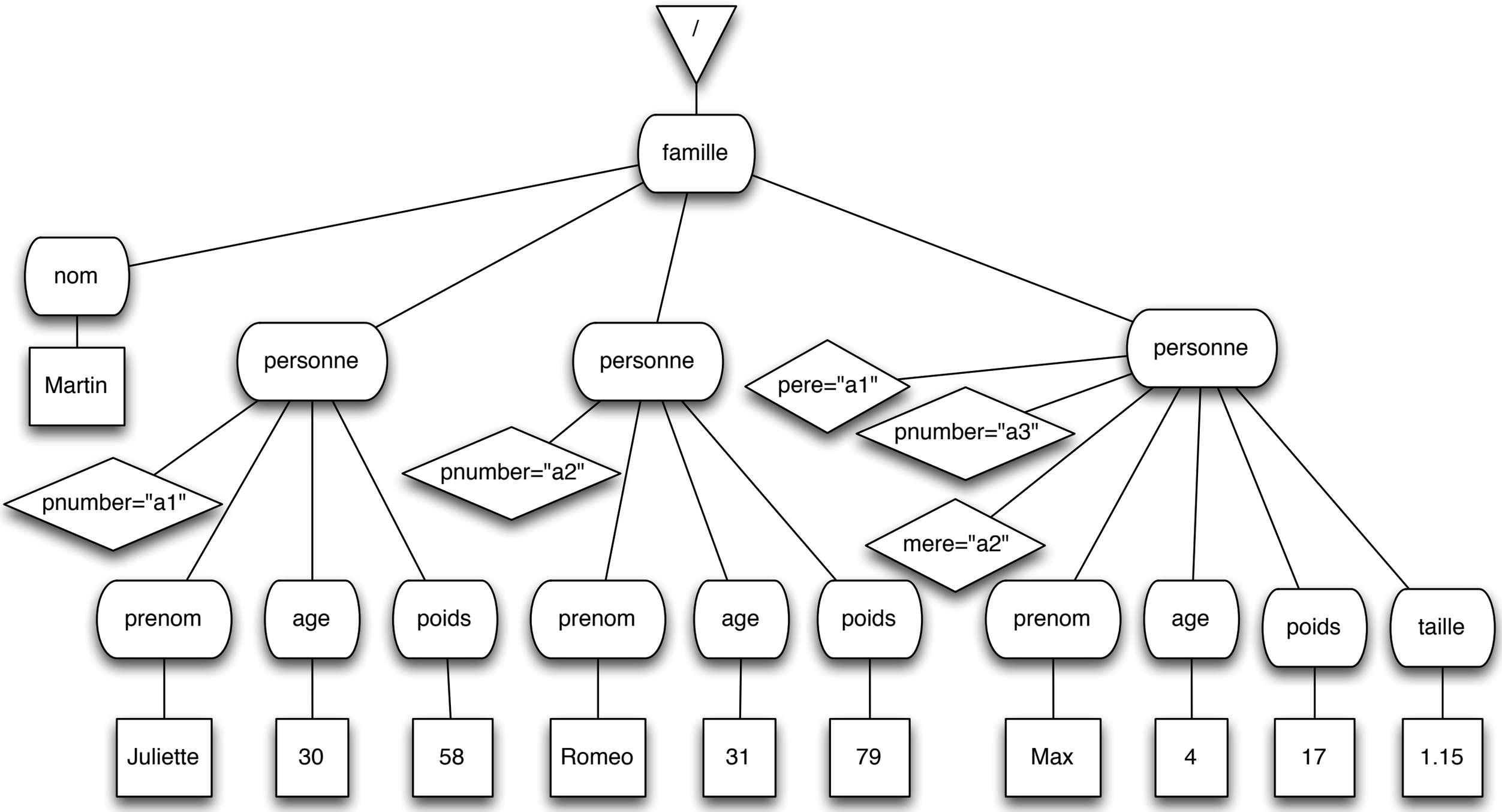
```
/descendant::personne/child::age/child::text()
```



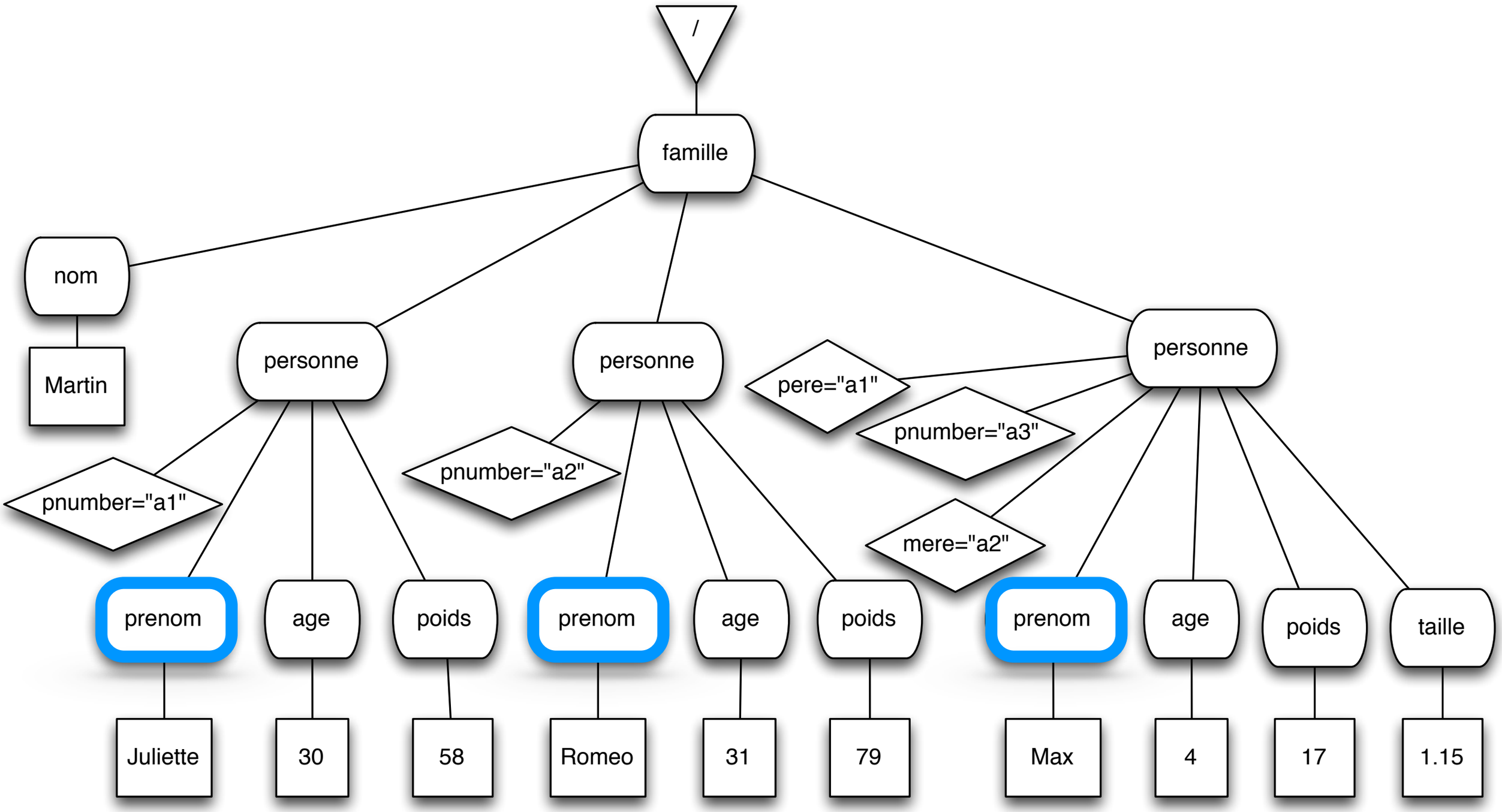
```
/descendant::personne/child::age/child::text()
```



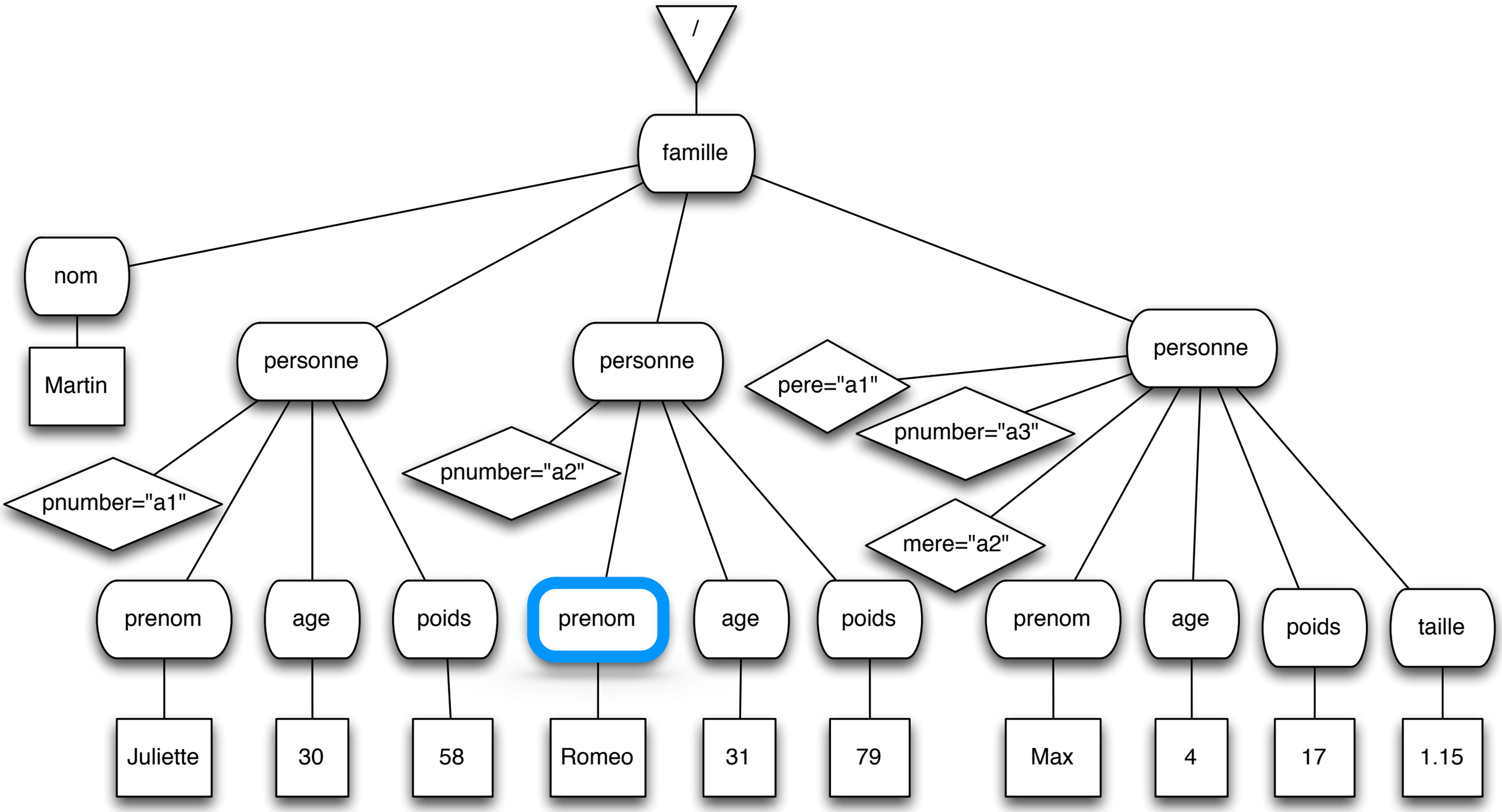
```
/descendant::prenom[child::text()=”Romeo“]/parent::*/attribute::pnumber
```



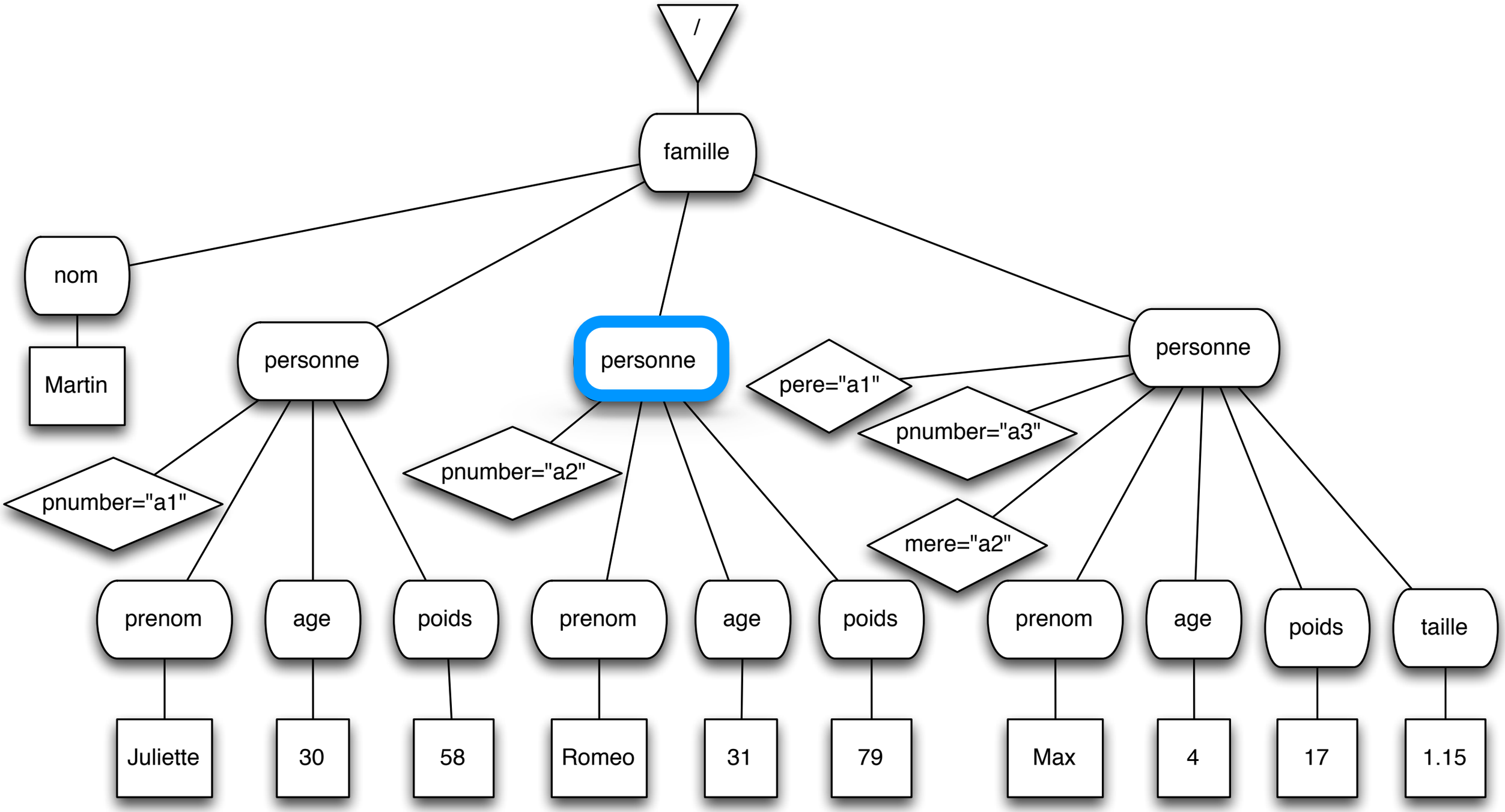
```
/descendant::prenom[child::text()="Romeo"]/parent::*/attribute::pnumber
```



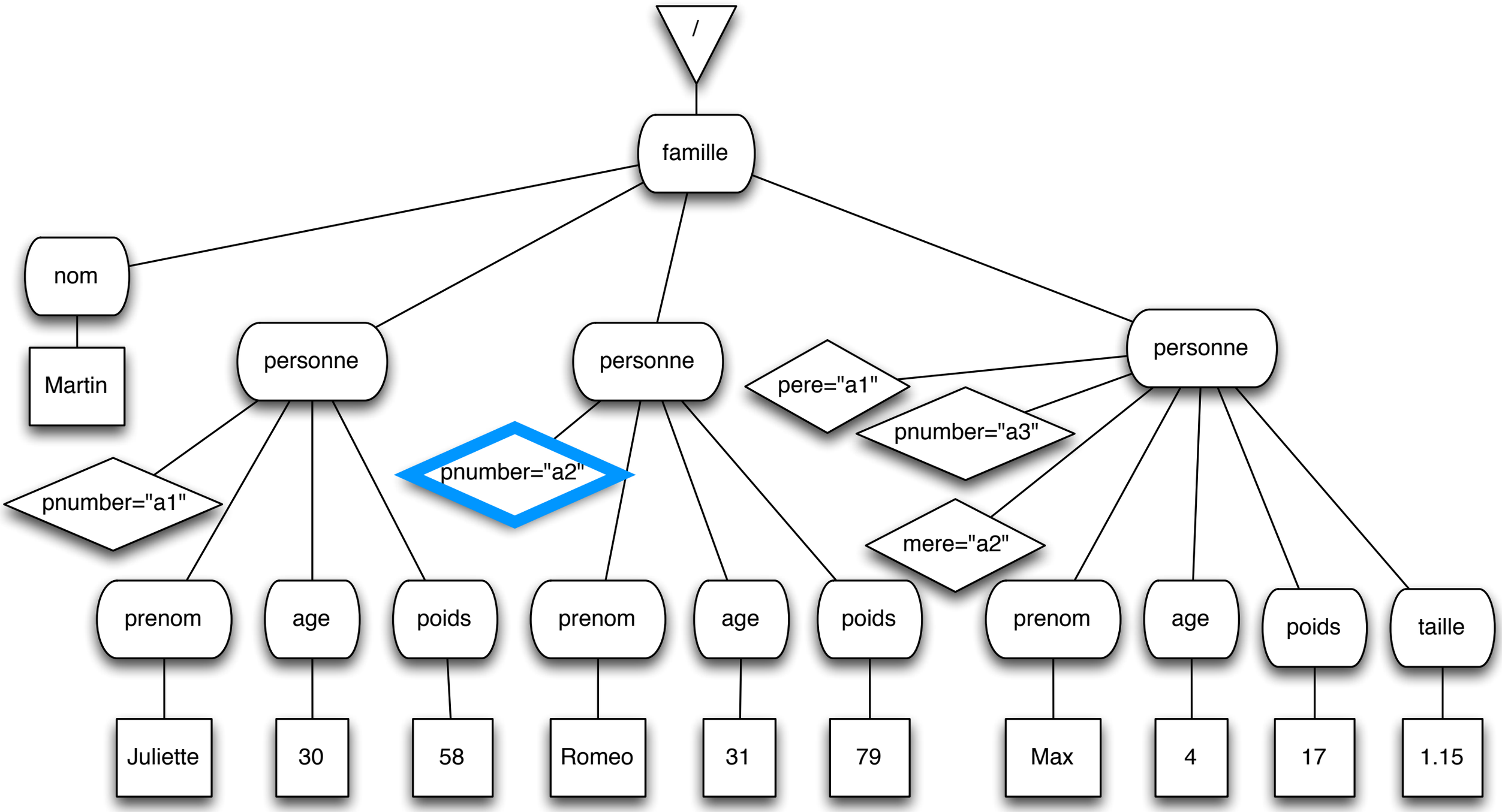
`/descendant::prenom[child::text()="Romeo"]``/parent::*/attribute::pnumber`



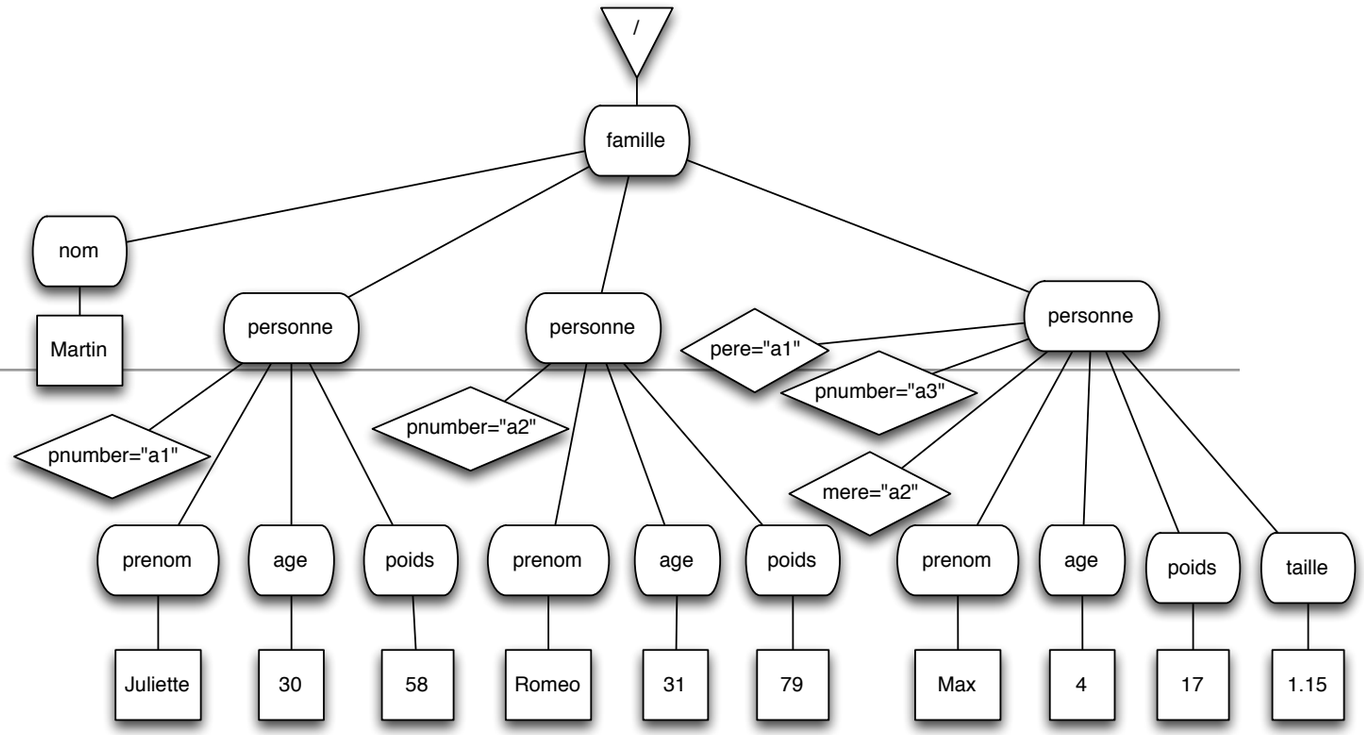
`/descendant::prenom[child::text()="Romeo"]/parent::*``/attribute::pnumber`



```
/descendant::prenom[child::text()="Romeo"]/parent::*/attribute::pnumber
```



Exemples

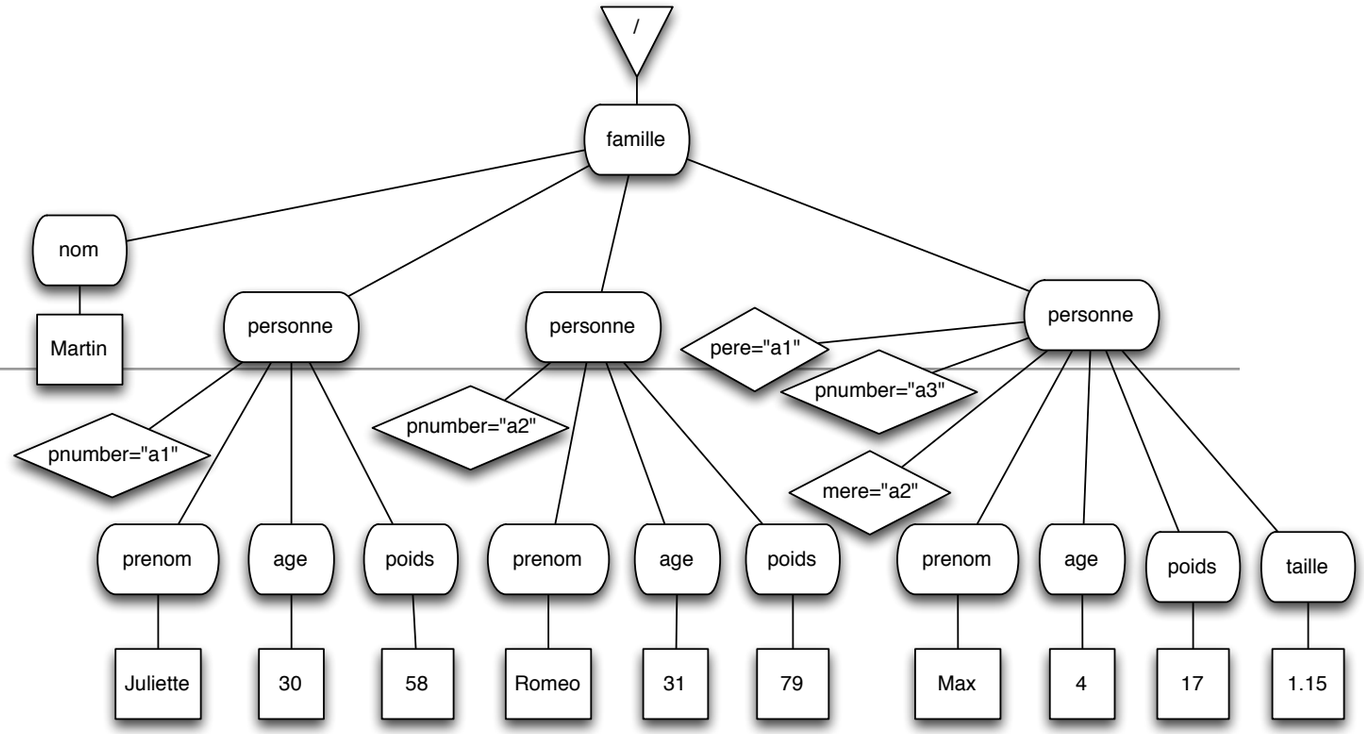


- `/child::famille/descendant::prenom/child::text()` retourne la liste des prénoms de personnes (sous la forme d'une liste de nœuds de type texte)

- `/descendant::nom/following-sibling::*` retourne la liste des 3 éléments personne.

- `/descendant::*` retourne la liste de tous les éléments du document (dans l'ordre du document)

Exemples



`/descendant::prenom[position()=2]` le deuxième élément prénom dans l'ordre du document.

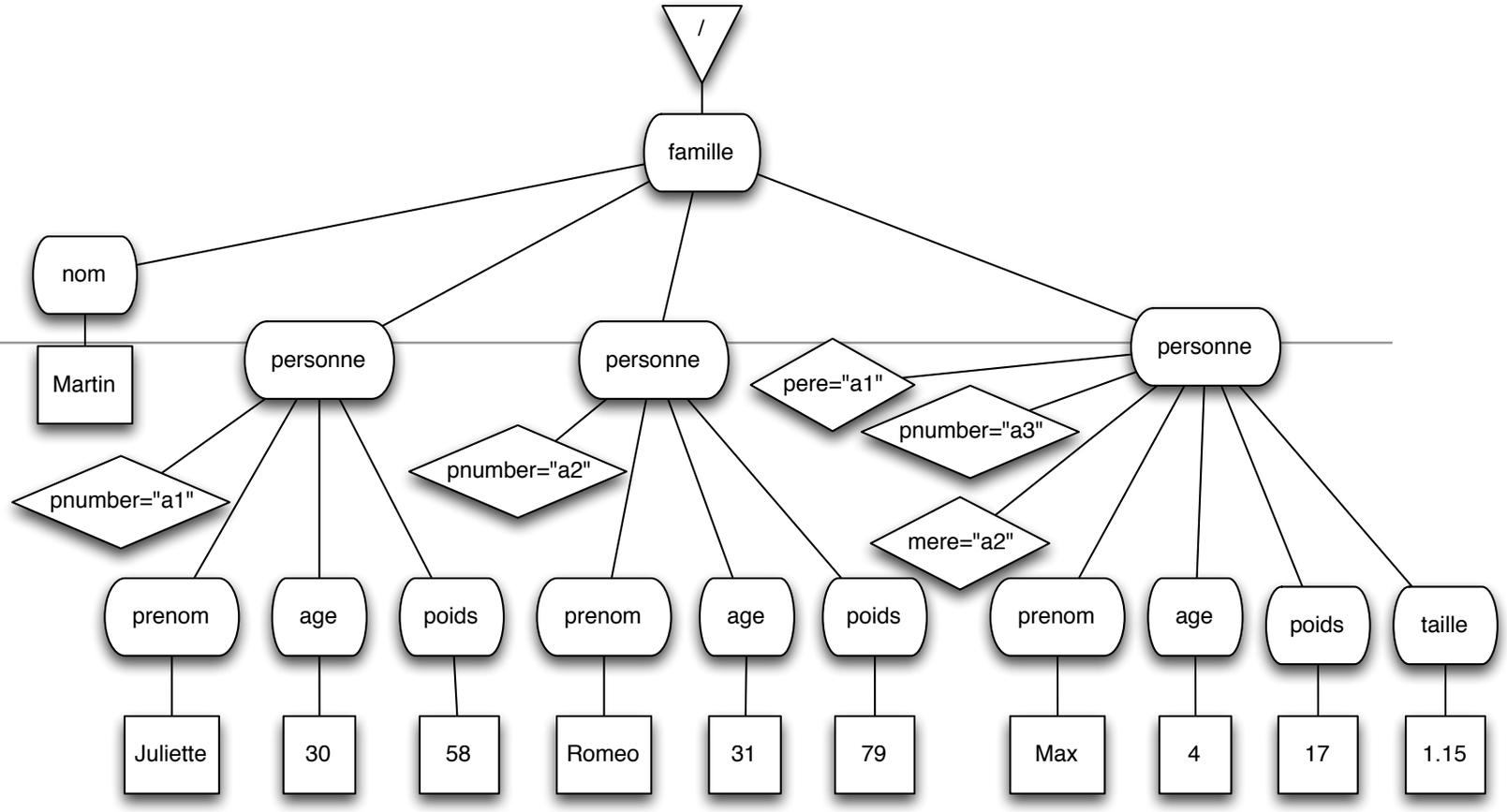
`/descendant::*/child::*[position()=2]` la liste des éléments qui sont deuxièmes fils (c'est la liste composée du 1er élément `personne` (dans l'ordre du document) et des éléments `age`).

Syntaxe abrégée

Syntaxe inspirée des systèmes de fichiers.

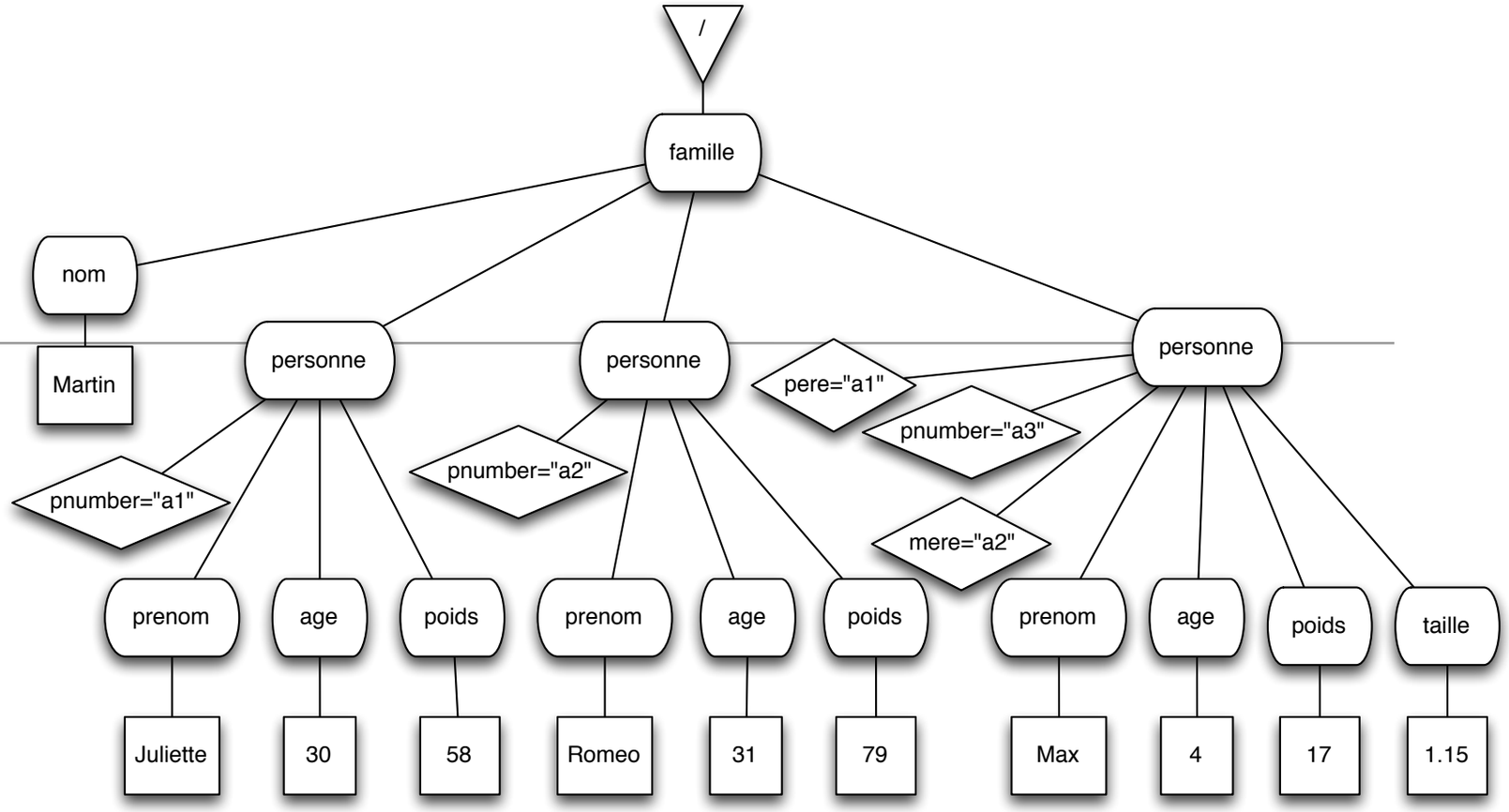
- `child::` peut être omis
- `attribute::` peut être remplacé par `@`
- `descendant-or-self::node()/child::` peut être remplacé par `//`
- `descendant-or-self::node()/attribute::` peut être remplacé par `//@`
- `self::node()` peut être remplacé par un `.`
- `parent::node()` peut être remplacé par `..`
- `[position()=x]` peut s'écrire `[x]`

Exemples



- `//*` sélectionne tous les éléments
- `//text()` sélectionne tous les nœuds texte
- `//@*` sélectionne tous les attributs de tous les éléments
- `//nom/..` sélectionne l'élément `famille`.
- `//personne/@pnumber` sélectionne tous les attributs `pnumber` des éléments `personne`.
- `//personne/../../famille/personne/@mere` sélectionne tous les attributs `mère` des éléments `personne`

Exemples



- `//famille/personne[2]` sélectionne le deuxième fils `personne` de l'élément `famille`.
- `//personne[@pnumber="a2"]` sélectionne les éléments `personne` dont l'attribut `pnumber` vaut `"a2"`.
- `//personne[.//@pnumber="a2"]` même résultat que la requête précédente.
- `//personne[//@pnumber="a2"]` sélectionne tous les éléments `personne` car `//@pnumber` est évalué à partir de la racine.

Sémantique existentielle des comparaisons

On dispose des comparaisons < , > , <= , >= , != , = entre expressions XPath (donc en particuliers entre listes d'items) avec une sémantique existentielle : `(1,2) = (2,3)` vaut vrai car il existe un élément commun entre les 2 séquences.

(Donc l'égalité n'est pas transitive ! On a (1 , 2) = (2 , 3) et (2 , 3) = (3 , 4), pourtant (1 , 2) = (3 , 4) vaut faux.)

Sémantique existentielle des comparaisons

```
//livre[auteur = editeur]
```

```
<?xml version="1.0" encoding="UTF-8"?>
<livres>
 <livre>
   <titre>Bonjour chez vous</titre>
   <auteur>Patrick McGoohan</auteur>
   <editeur>Patrick McGoohan</editeur>
 </livre>
</livres>
```

Sémantique existentielle des comparaisons

```
//livre[auteur = editeur]
```

```
<?xml version="1.0" encoding="UTF-8"?>
<livres>
 <livre>
   <titre>Bonjour chez vous</titre>
   <auteur>John Smith</auteur>
   <auteur>Patrick McGoohan</auteur>
   <editeur>James Kaith</editeur>
   <editeur>Patrick McGoohan</editeur>
 </livre>
</livres>
```

Sémantique existentielle des comparaisons

```
//livre[auteur = editeur]
```

```
<?xml version="1.0" encoding="UTF-8"?>
<livres>
 <livre>
   <titre>Bonjour chez vous</titre>
   <auteur>John Smith</auteur>
   <auteur>Will Stuart</auteur>
   <editeur>James Kaith</editeur>
   <editeur>Patrick McGoohan</editeur>
 </livre>
</livres>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>63</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>65</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>25</age>
      <poids>73</poids>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>18</age>
      <poids>54</poids>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
   <homme id="5">
      <prenom>Paul</prenom>
      <age>5</age>
      <poids>10</poids>
      <pere>3</pere>
   </homme>
</famille>
```

Le prénom des personnes les plus lourdes

```
/famille/*[not(poids < //poids)]/prenom/text()
```

Romeo

~~`/famille/*[poids >= //poids]/prenom/text()`~~

Juliette
Romeo
Max
Marie
Paul

```xml
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>63</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>65</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>25</age>
      <poids>73</poids>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>18</age>
      <poids>54</poids>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
   <homme id="5">
      <prenom>Paul</prenom>
      <age>5</age>
      <poids>10</poids>
      <pere>3</pere>
   </homme>
</famille>
```

Les id des hommes qui ne sont pas pères

~~//homme[@id != //pere]/@id~~

2

3

5

```
//homme[not(@id=//pere)]/@id
```

5

Séquences

Construction d'une séquence : Un moyen de construire une séquence est d'utiliser la virgule comme opérateur de concaténation. L'opérateur `to` permet de définir un intervalle.

- `(10, 1, 2, 3, 4)` séquence de 5 entiers
- `(10, (1, 2), (), (3, 4))` même séquence que précédemment
- `(10, 1 to 4)` toujours la même séquence.

Séquences

Expressions de chemin

- `//famille/*[age > 20]/(prenom,age)` séquence avec tous les fils `prenom` et `age` des éléments `personne` sélectionnés.

```
<prenom>Juliette</prenom>
<age>63</age>
<prenom>Romeo</prenom>
<age>65</age>
<prenom>Max</prenom>
<age>25</age>
```

- `(21 to 29)[5]` l'entier `25`.
- `//personne[position()=(1 to 2)]` le premier et le second élément `personne` du document.

Séquences

Opérateurs sur les séquences de nœuds :

- `union`,
- `intersect`,
- `except`.

Opérateur d'union

On peut faire l'union des résultats de plusieurs expressions XPath à l'aide de l'opérateur | (XPath 1.0) ou `union (XPath2.°)`.

```
//personne/@mere | //personne/@pere
```

sélectionne tous les attributs `pere` et tous les attributs `mere` des éléments `personne`.

Exercice : Utiliser cet opérateur d'union pour construire une expression XPath pouvant servir à tester si un nœud (le nœud contexte) est un attribut ou pas.

```
count(.|../@*) = count(../@*)
```

Opérateur d'union

```xml
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>30</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>31</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>4</age>
      <poids>12</poids>
      <taille>1.25</taille>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>3</age>
      <poids>18</poids>
      <taille>1.10</taille>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
</famille>
```

```
//age union //poids
```

```
<age>30</age>
<poids>58</poids>
<age>31</age>
<poids>97</poids>
<age>4</age>
<poids>12</poids>
<age>3</age>
<poids>18</poids>
```

Opérateur d'union

```xml
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>30</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>31</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>4</age>
      <poids>12</poids>
      <taille>1.25</taille>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>3</age>
      <poids>18</poids>
      <taille>1.10</taille>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
</famille>
```

```
//age union //age[name(..) eq "femme"]
```

```xml
<age>30</age>
<age>31</age>
<age>4</age>
<age>3</age>
```

```xml
<age>30</age>
<age>3</age>
```

```xml
<age>30</age>
<age>31</age>
<age>4</age>
<age>3</age>
```

Opérateur d'union

```xml
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>30</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>31</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>3</age>
      <poids>12</poids>
      <taille>1.25</taille>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>3</age>
      <poids>18</poids>
      <taille>1.10</taille>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
</famille>
```

```
//age union //age[name(..) eq "femme"]
```

```xml
<age>30</age>
<age>31</age>
<age>3</age>
<age>3</age>
```

```xml
<age>30</age>
<age>3</age>
```

```xml
<age>30</age>
<age>31</age>
<age>3</age>
<age>3</age>
```

Opérateur d'union

```
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>30</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>31</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>3</age>
      <poids>12</poids>
      <taille>1.25</taille>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>3</age>
      <poids>18</poids>
      <taille>1.10</taille>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
</famille>
```

```
//age[name(..) eq "homme"]
                  union
                     //age[name(..) eq "femme"]
```

```
<age>31</age>
<age>3</age>
```

```
<age>30</age>
<age>3</age>
```

```
<age>30</age>
<age>31</age>
<age>3</age>
<age>3</age>
```

Opérateur d'union

```
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>30</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>31</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>4</age>
      <poids>12</poids>
      <taille>1.25</taille>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>3</age>
      <poids>18</poids>
      <taille>1.10</taille>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
</famille>
```

```
(<A/> , <B/>) union (<C/> , <A/>)

<A/>
<B/>
<C/>
<A/>
```

Opérateur d'intersection

```
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>30</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>31</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>4</age>
      <poids>12</poids>
      <taille>1.25</taille>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>3</age>
      <poids>18</poids>
      <taille>1.10</taille>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
</famille>
```

```
//age[following-sibling::mere]

intersect

//femme/age
```

```
<age>3</age>
```

Opérateur de différence

```xml
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>30</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>31</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>4</age>
      <poids>12</poids>
      <taille>1.25</taille>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>3</age>
      <poids>18</poids>
      <taille>1.10</taille>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
</famille>
```

```
//age[following-sibling::mere]

except

//femme/age
```

```
<age>4</age>
```

Plein de fonctions!

Voir la fiche synthétique

http://www.mulberrytech.com/quickref/functions.pdf

```xml
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>63</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>65</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>25</age>
      <poids>73</poids>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>18</age>
      <poids>54</poids>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
   <homme id="5">
      <prenom>Max</prenom>
      <age>5</age>
      <poids>10</poids>
      <pere>3</pere>
   </homme>
</famille>
```

Tous les prénoms sans doublon

```
distinct-values(doc("famille.xml")//prenom)
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>50</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>50</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>25</age>
      <poids>73</poids>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>5</age>
      <poids>54</poids>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
   <homme id="5">
      <prenom>Max</prenom>
      <age>3</age>
      <poids>10</poids>
      <pere>3</pere>
   </homme>
</famille>
```

Quelle expression XPath pour obtenir le résultat suivant ?

```xml
<age>50</age>
<age>25</age>
<age>5</age>
<age>3</age>
```

distinct-values(doc(“famille.xml“)//age) X

50
25
5
3

doc(“famille.xml“)//age

```xml
<age>50</age>
<age>50</age>
<age>25</age>
<age>5</age>
<age>3</age>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>50</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>50</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>25</age>
      <poids>73</poids>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>5</age>
      <poids>54</poids>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
   <homme id="5">
      <prenom>Max</prenom>
      <age>3</age>
      <poids>10</poids>
      <pere>3</pere>
   </homme>
</famille>
```

Quelle expression XPath pour obtenir le résultat suivant ?

```xml
<age>50</age>
<age>25</age>
<age>5</age>
<age>3</age>
```

```
distinct-values(doc("famille.xml")//age)
```
X

```
//age[text()!=following::age/text()]
```
X

~~<age>50</age>~~
~~<age>50</age>~~
~~<age>25</age>~~
~~<age>5</age>~~

```xml
<?xml version="1.0" encoding="UTF-8"?>
<famille id="MARTIN">
   <femme id="1">
      <prenom>Juliette</prenom>
      <age>50</age>
      <poids>58</poids>
   </femme>
   <homme id="2">
      <prenom>Romeo</prenom>
      <age>50</age>
      <poids>97</poids>
   </homme>
   <homme id="3">
      <prenom>Max</prenom>
      <age>25</age>
      <poids>73</poids>
      <pere>2</pere>
      <mere>1</mere>
   </homme>
   <femme id="4">
      <prenom>Marie</prenom>
      <age>5</age>
      <poids>54</poids>
      <pere>2</pere>
      <mere>1</mere>
   </femme>
   <homme id="5">
      <prenom>Max</prenom>
      <age>3</age>
      <poids>10</poids>
      <pere>3</pere>
   </homme>
</famille>
```

NOK

OK

OK

OK

Quelle expression XPath pour obtenir le résultat suivant ?

```xml
<age>50</age>
<age>25</age>
<age>5</age>
<age>3</age>
```

distinct-values(doc("famille.xml")//age) X

//age[text()!=following::age/text()] X

//age[not(text()=following::age/text())] OK