Atelier 8 : objets prédéfinis et documentation en ligne

Objectifs

- S'intéresser aux objets prédéfinis et aux méthodes disponibles
- Voir comment la documentation en ligne est présentée et l'exploiter
- Appliquer la matière vue dans le cadre d'exercices pratiques globaux

Note. Les énoncés qui suivent vous proposent d'utiliser des fonctionnalités qui n'ont pas été décrites au cours mais que vous devrez rechercher sur internet, apprendre par vous-mêmes et ensuite appliquer pour résoudre le problème posé. Toutes les informations nécessaires peuvent être trouvées sur le site de documentation de *Mozilla Developer Network* (MDN) dont l'adresse est la suivante.

https://developer.mozilla.org/en-US/docs/Web/JavaScript

Il en existe également une version française qui peut être pratique si vous avez des difficultés à comprendre certains passages mais, de manière générale, c'est plutôt intéressant de s'exercer à utiliser de la documentation technique en anglais.

Exercice 1 : Lecture de la documentation en ligne

Étape 1 : propriétés propres et propriétés héritées

Si un objet possède un prototype, il hérite des propriétés de ce dernier. Considérez l'exemple suivant.

```
let parent = { valeur : 12 };
let fils = Object.create(parent);
fils.nombre = 42;
```

L'objet `fils` permet d'accéder à (au moins) deux propriétés : `fils.nombre`, qui est une propriété propre à `fils` et `fils.valeur`, qui est une propriété héritée de son prototype `parent`.

Testez votre « google-fu » (capacité à effectuer des recherches fructueuses sur Google) en tentant de repérer les outils que Javascript propose pour identifier les propriétés propres à un objet.

Indice : il y a deux méthodes qui sont pertinentes (les réponses sont données dans l'étape suivante) ; en anglais, « propriétés propres à un objet » se traduirait par « own properties ».

Étape 2 : propriétés propres et propriétés héritées (ébauche de solution)

Pour vérifier que vous avez bien trouvé les deux méthodes en question, voici des liens vers les pages de description : http://goo.gl/z9RfJV et http://goo.gl/rBAZPy

Sur chacune de ces deux pages, lisez attentivement le titre rappelant le nom de méthode et le petit texte en-dessous du titre indiquant ce que fait la méthode.

Avez-vous remarqué une différence importante entre les deux titres ?

À quels objets sont rattachées chacune des deux propriétés ? (la réponse n'est pas la même pour les deux propriétés)

Voyez-vous en quoi cette différence implique que ces méthodes seront utilisées de manière différente ?

Pour tester votre compréhension, utilisez ces deux méthodes pour écrire

a) une commande Javascript qui va donner la liste des propriétés propres de `fils` (a priori, il n'y a que `nombre`) et
b) une commande Javascript qui va indiquer (par un booléen) si « `valeur` » est une propriété propre de `fils` (devrait indiquer `false`).

Si vous éprouvez des difficultés, consultez les sections « Syntax » des deux pages de descriptions (ou les exemples donnés plus bas) et observez bien les différences.

Étape 3 : retour sur Number

Recherchez la page de description de la fonction `Number` sur MDN (si vous ne la trouvez pas, voici son adresse : http://goo.gl/3GfWl8)

Lisez la section intitulée « Description ». Assurez-vous que vous comprenez bien chacune des phrases. Tout cela devrait vous évoquer des choses déjà vues dans les modules précédents.

Consultez les trois sections qui suivent, à savoir :

- « Properties », qui cite les attributs de l'objet `Number` ;
- « Methods », qui cite les méthodes de l'objet `Number` ;
- « Number instances », qui cite les méthodes de l'objet `Number.prototype`, c'est-à-dire les méthodes dont chacun des nombres va hériter.

Au sujet de la méthode `Number.parseFloat()`, on peut lire « The value is the same as `parseFloat()` of the global object. »

a) Vous souvenez-vous de qui est ce « global object » ?
b) Si `pi = "3.1415"`, citez trois manières différentes (3 syntaxes différentes) pour obtenir la valeur réelle de cette chaîne de caractères (les trois manières utilisent `parseFloat` et découlent de la phrase citée ci-dessus).

Étape 4 : printf("%.2f", prix) en Javascript

Toujours sur base de la page consacrée à `Number`, pouvez-vous trouver la méthode qui permettra de résoudre le problème suivant ?

Le tableau `catalogue` défini ci-dessous contient les descriptions de divers articles proposés à la vente. On voudrait afficher (dans la console par exemple), pour chaque article, son code et son prix en Euros. Comme il s'agit de prix en Euros, ceux-ci devraient être affichés avec 2 décimales (par exemple : 2.00€, 4.50€, 3.25€, 7.90€, 6.42€).

```
let catalogue = [
  {code: "AB714", prix: 2},
  {code: "ZU330", prix: 4.5},
  {code: "YO123", prix: 3.25},
  {code: "BU999", prix: 7.9},
  {code: "VO111", prix: 6.42}
];
```

Étape 5 : les deux isNaN

Toujours sur base de la page consacrée à `Number`, repérez la méthode `Number.isNaN` qui, selon sa description, « determines whether the passed value is NaN. »

Dans les modules précédents, on a utilisé une fonction « `isNaN` » différente de cette méthode portée par l'objet `Number`.

Utilisez la documentation en ligne pour déterminer les différences entre ces deux fonctions. Pour tester votre compréhension, citez une valeur pour laquelle ces deux fonctions donnent des réponses différentes.

Étape 6 : parseInt

Sur la page consacrée à `Number`, on évoque la méthode `Number.parseInt` qui, comme `parseFloat`, existe également sur l'objet global.

Examinez et exécutez le code suivant.

```
let nombres = ["3", "4", "5", "6"];
nombres.map(x => x + x);
```

Rien de trop étonnant pour le moment. Faites de même avec le code suivant.

```
nombres.map(x => parseInt(x));
```

Tout va toujours bien ? Mais peut-être vous dites-vous qu'il est inutile de créer une « nouvelle » fonction qui fait la même chose que `parseInt` et qu'on pourrait l'utiliser directement ? Testez donc le code suivant.

```
nombres.map(parseInt);
```

Pouvez-vous expliquer ce résultat ? (Si vous ne trouvez pas immédiatement, consultez aussi la page abordant la méthode `map`).

Exercice 2 : Arrays et strings en pagaille

Cet exercice utilise quelques-unes des méthodes prédéfinies sur les tableaux et sur les chaînes de caractères. Comme précédemment, référez-vous à la documentation en ligne sur MDN pour obtenir plus de détails quant aux spécifications de ces méthodes.

Étape 1 : distance

Examinez la définition de fonction suivante et déterminez ce qu'elle fait. Vérifiez ensuite votre réponse en utilisant les exemples cités.

```
function distance (tab,x) {
  return tab.lastIndexOf(x) - tab.indexOf(x);
}

distance([0,1,2,3,4,3,2,1,0], 4)
distance([0,1,2,3,4,3,2,1,0], 2)
```

Étape 2 : motsLexico

Faites de même avec la définition suivante.

```
function motsLexico (s) {
  return s.split(" ").sort();
}

motsLexico("le chien noir aboie toute la nuit");
motsLexico("toujours penser clean code");
```

Étape 3 : memeSigne

Et la fonction décrite ci-dessous ?

```
function memeSigne (tab) {
  return tab.every(function (x) { return x > 0; })
    || tab.every(function (x) { return x < 0; });
}

memeSigne([1,2,3,4,5,6]);
memeSigne([-2,-5,-7,-34,-1]);
memeSigne([1,-2,3,-4,5,-6]);
```

Étape 4 : carreDesPairs

Et cette fonction ?

```
function carreDesPairs (tab) {
  return tab.filter(function (x) { return x % 2 == 0; })
    .map(function (x) { return x * x; });
}
```

```
carreDesPairs([1,2,3,4,5]);
carreDesPairs([5,6,7]);
```

Étape 5 : plusLong

Une petite dernière ?

```
function plusLong (s) {
  function plusLongDeDeux (s1,s2) {
    return s1.length > s2.length ? s1 : s2;
  }
  return s.split(" ").reduce(plusLongDeDeux);
}

plusLong("le chien noir aboie toutes les nuits");
plusLong("toujours penser clean code");
```

Étape 6 : à vous de coder

En utilisant les méthodes présentées ci-dessus ou d'autres méthodes prédéfinies, définissez des fonctions répondant aux spécifications suivantes.

1. Une fonction `tousNamur(tab)` qui reçoit un tableau de codes postaux et qui indique s'ils correspondent bien tous à la province de Namur (c'est-à-dire s'ils sont bien tous compris entre 5000 et 5999 au sens large).
   Utilisez « every ».

```
tousNamur([5000,5500,5600,5200]) → true
tousNamur([1000,5600,4000]) → false
```

2. Une fonction `motsAvecInitiale(s,lettre)` qui reçoit une chaîne de caractères comportant plusieurs mots séparés par des virgules et une lettre et qui renvoie un tableau reprenant tous les mots de la chaîne qui commencent par la lettre en question.
   Utilisez « filter » et « startsWith ».

```
motsAvecInitiale("le,chien,est,un,caniche","c") → ["chien","caniche"]
motsAvecInitiale("trois,plus,quatre,égal,sept","h") → []
```

3. Une fonction `pascalCase(mots)` qui reçoit un tableau de mots et renvoie une unique chaîne de caractères obtenue en concaténant ces mots selon les règles du PascalCase (chaque mot commence par une majuscule et le reste du mot est en minuscules).
   Utilisez « map », « substring », « toLowerCase » et « toUpperCase ».

```
pascalCase(["age","CAPITAINE","naVIre"]) → "AgeCapitaineNavire"
pascalCase(["nb","jeux","PROMOTION","ajd"]) → "NbJeuxPromotionAjd"
```

Exercice 3 : Tri de tableaux

Cet exercice se base sur la méthode `Array.prototype.sort()` disponible sur tous les tableaux. Elle est décrite sur la page https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort .

Étape 1 : première approche de la méthode sort

Pour vous familiariser avec cette méthode, définissez tout d'abord un tableau de chaînes de caractères (des prénoms, des noms de villes, … ce que vous voulez) puis triez-le.

La description en haut de la page indique que cette méthode « sorts the elements of an array in place and returns the array. »

Que veut dire « in place » dans ce cas-ci ? (réponse : https://goo.gl/bfXUlr)

Calculez

```
[10, 42, 97, 50, 100].sort()
```

puis expliquez le résultat obtenu…

Étape 2 : fonction de comparaison

La méthode `sort` peut accepter un paramètre décrit sur la page web sous le nom de `compareFunction`. Lisez la description de ce paramètre optionnel dans la section « Parameters » et dans la section « Description » plus bas.

Évaluez le code suivant. Expliquez-en le résultat… pourquoi utiliser une fonction qui calcule la différence de deux nombres ?

```
function diff (x, y) { return x - y; }
[10, 42, 97, 50, 100].sort(diff);
```

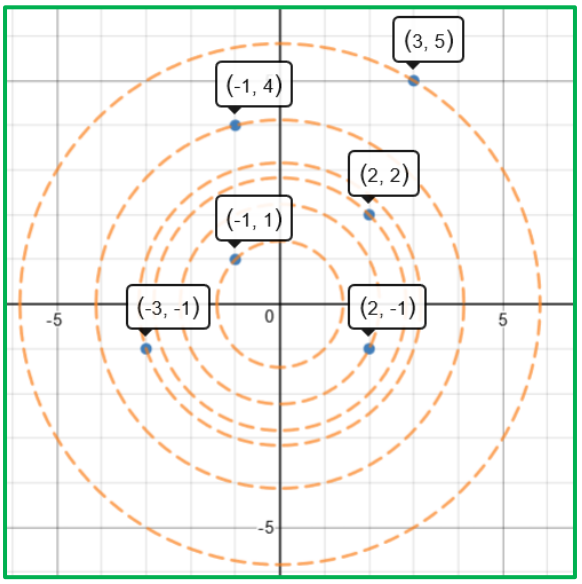
Étape 3 : tri de points

Le tableau ci-dessous reprend la liste des points qui sont représentés sur le graphique de la page suivante.

```
function Point (x, y) {
  this.x = x;
  this.y = y;
}
let points = [
  new Point(-3,-1),
  new Point(-1,4),
  new Point(-1,1),
  new Point(2,2),
  new Point(2,-1),
  new Point(3,-5)
];
```

Observez tout d'abord les affichages produits par les commandes et expressions suivantes (à entrer une par une).

```
points
console.log(points);
points + ""
```



Aucune de ces sorties n'est vraiment pratique à lire… pour améliorer la chose, définissez une méthode `toString()` sur les points. Utilisez le format d'affichage standard, à savoir (x, y) puis testez à nouveau les trois lignes de code données ci-dessus afin de déterminer la solution qui sera la plus pratique pour vos futurs tests.

Le but de cet exercice est de pouvoir trier le tableau de points grâce à une unique commande

```
points.sort(comparePoints);
```

qui réordonnera les points du plus proche de l'origine vers le plus éloigné de l'origine.

Définissez la fonction `comparePoints` qui permet d'obtenir ce résultat.

Note. La distance entre l'origine $(0,0)$ et un point (x, y) est $\sqrt{x^2 + y^2}$. Ceci dit, posez-vous également la question suivante : si on vous demande quel est le nombre le plus grand parmi $\sqrt{729}$ et $\sqrt{123}$, devez-vous vraiment calculer les racines carrées pour arriver à la réponse ?

Exercice 4 : Souffler les bougies

Le but de cet exercice est de modéliser une liste d'amis dont on retient le nom, le prénom, le sexe et la date de naissance, de la manipuler et de l'afficher sous diverses formes (1) en créant des objets Javascript et (2) en se servant des objets prédéfinis dans Javascript.

Étape 1 : structure des fichiers et fonction constructrice

Le projet sera organisé en deux fichiers :

- un document appelé `amis.html` qui correspondra à la page principale et
- un fichier `amis.js` qui contiendra une bonne partie du code Javascript (à savoir tout ce qui concerne les objets « Amis » ainsi que les diverses fonctions utilitaires nécessaires).

Contrairement aux laboratoires précédents, cet énoncé vous laisse relativement libres. Libres entre autres d'organiser votre code comme vous le voulez : à vous de déterminer quelles fonctions (et méthodes) annexes définir et comment structurer vos scripts. Pensez Clean Code, pensez « code facile à lire », pensez « code bien structuré », pensez « code facile à modifier ».

Dans un premier temps, concentrez-vous sur l'écriture d'une fonction constructrice `Ami` permettant de créer des objets « ami » comportant

- un nom,
- un prénom,
- un sexe stocké sous le format que vous trouvez le plus adéquat mais reçu par la fonction constructrice sous la forme d'une chaîne de caractère (« f » et « F » signifiant féminin, « m » et « M » signifiant masculin), et
- une date de naissance stockée sous la forme d'un tableau associatif (jour, mois, année retenus séparément) mais reçue par la fonction constructrice au format AAAAMMJJ.

Ainsi, si la variable `adelaide` correspond une certaine Adélaïde Berie, née le 30/10/1973, on aura

```
adelaide.nom → "Berie"
adelaide.prenom → "Adélaïde"
adelaide.sexe → (ce qui correspond à « féminin »)
adelaide.dateNaiss.jour → 30
adelaide.dateNaiss.mois → 10
adelaide.dateNaiss.annee → 1973
```

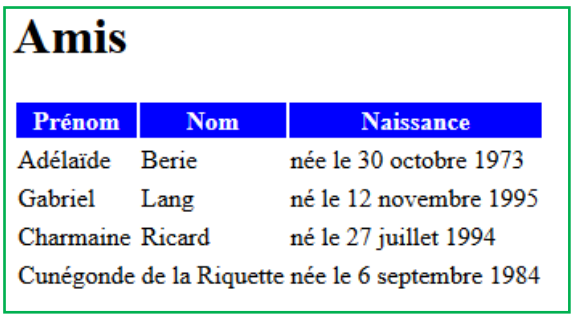
La fonction constructrice devra être compatible avec le script suivant, censé remplir un tableau appelé `amis`.

```
amis = [];
amis[0] = new Ami ("Adélaïde", "Berie", "F", 19731030);
amis[1] = new Ami ("Gabriel", "Lang", "m", 19951112);
amis[2] = new Ami ("Charmaine", "Ricard", "M", 19940727);
amis[3] = new Ami ("Cunégonde", "de la Riquette", "F", 19840906);
```

Ajoutez d'ailleurs les lignes de script reprises ci-dessus dans un script Javascript placé directement dans la partie « en-tête » du document HTML. Ce script initialisera la liste des amis à afficher (et sera complété au fil des étapes suivantes).

Étape 2 : premiers affichages

Dans un premier temps, ajoutez un titre (balise `h1`) « Amis » suivi d'un tableau dans le document HTML. Le tableau à produire devra correspondre à l'exemple donné ci-dessous.



Amis

Prénom	Nom	Naissance
Adélaïde	Berie	née le 30 octobre 1973
Gabriel	Lang	né le 12 novembre 1995
Charmaine	Ricard	né le 27 juillet 1994
Cunégonde de la Riquette		née le 6 septembre 1984

Pour produire le tableau, vous pouvez utiliser un script inséré dans la partie `body` du document HTML et l'instruction `document.write`.

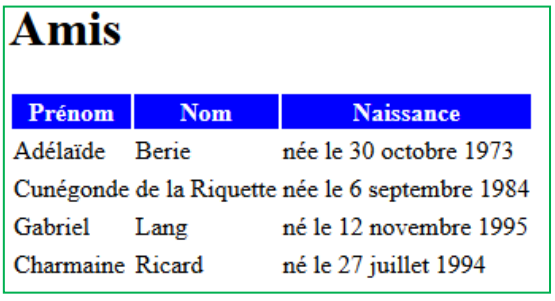
Quelques conseils/notes/observations :

- N'agglutinez pas tout votre code dans le script du document HTML. Pensez aux méthodes qui pourraient être définies pour les objets `Ami` et qui pourraient vous simplifier le travail.
- Parmi les objets prédéfinis en Java, il y a `Math`, qui contient diverses méthodes permettant de manipuler les nombres ; vous pourriez en avoir besoin.
- **Important** : N'oubliez pas de définir les méthodes partagées par tous les objets `Ami` dans leur prototype commun ! Ne les répétez pas dans chaque objet !

Étape 3 : tri alphabétique

Pour l'instant, les amis sont affichés dans l'ordre où ils sont encodés. Modifiez le script définissant le tableau `amis` en ajoutant une seule instruction à la fin du script, de manière à trier ce tableau selon les noms et prénoms des personnes. Pour ce faire, utilisez la méthode `Array.prototype.sort` (consultez sa documentation en ligne si nécessaire).

Vous devriez obtenir le résultat suivant.

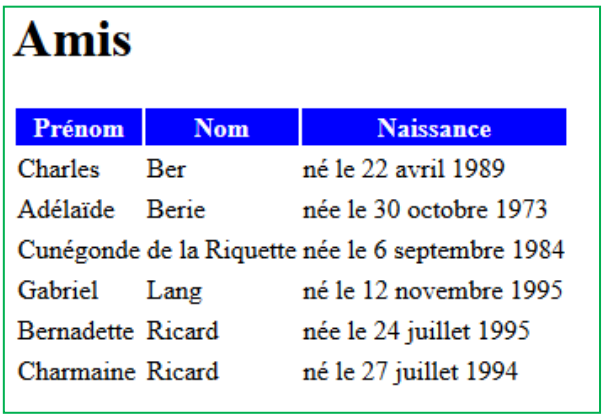


Amis

Prénom	Nom	Naissance
Adélaïde	Berie	née le 30 octobre 1973
Cunégonde de la Riquette		née le 6 septembre 1984
Gabriel	Lang	né le 12 novembre 1995
Charmaine	Ricard	né le 27 juillet 1994

Ajoutez les deux amis suivants et vérifiez que votre script produit bien le résultat attendu.

```
amis[4] = new Ami ("Bernadette", "Ricard", "f", 19950724);
amis[5] = new Ami ("Charles", "Ber", "m", 19890422);
```



Amis

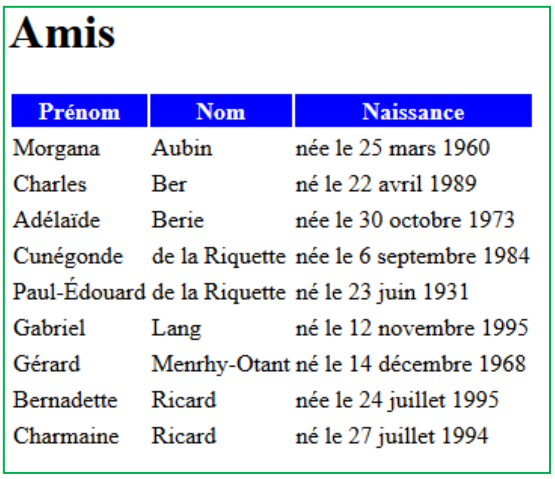
Prénom	Nom	Naissance
Charles	Ber	né le 22 avril 1989
Adélaïde	Berie	née le 30 octobre 1973
Cunégonde	de la Riquette	née le 6 septembre 1984
Gabriel	Lang	né le 12 novembre 1995
Bernadette	Ricard	née le 24 juillet 1995
Charmaine	Ricard	né le 27 juillet 1994

Étape 4 : autre format d'entrée

Afin de pouvoir récupérer certains amis qui avaient été encodés dans une ancienne liste, modifiez la fonction constructrice `Ami` afin qu'elle puisse être également appelée avec une unique chaîne de caractères reprenant le prénom et le nom (par exemple « Morgana Aubin »). Dans cette chaîne de caractères, on supposera que le prénom s'arrête juste avant le premier espace et que le nom commence juste après cet espace.

Pour vérifier le code de votre fonction constructrice surchargée, ajoutez les amis suivants.

```
amis[6] = new Ami ("Morgana Aubin", "F", 19600325);
amis[7] = new Ami ("Paul-Édouard de la Riquette", "m", 19310623);
amis[8] = new Ami ("Gérard Menrhy-Otant", "M", 19681214);
```



Amis

Prénom	Nom	Naissance
Morgana	Aubin	née le 25 mars 1960
Charles	Ber	né le 22 avril 1989
Adélaïde	Berie	née le 30 octobre 1973
Cunégonde	de la Riquette	née le 6 septembre 1984
Paul-Édouard	de la Riquette	né le 23 juin 1931
Gabriel	Lang	né le 12 novembre 1995
Gérard	Menrhy-Otant	né le 14 décembre 1968
Bernadette	Ricard	née le 24 juillet 1995
Charmaine	Ricard	né le 27 juillet 1994

Étape 5 : tri chronologique

Modifiez le tableau pour que les amis soient affichés non plus par ordre alphabétique sur leurs nom et prénom mais plutôt par ordre chronologique sur leur date d'anniversaire (on négligera l'année).

Pour ce faire, commencez par définir une fonction `compareDates(d1,d2)` qui compare deux dates (c'est-à-dire deux tableaux associatifs contenant des propriétés jour, mois et

annee) et renvoie -1 si `d1` vient avant `d2` dans l'année, 0 s'il s'agit de dates identiques et 1 si `d1` vient après `d2` dans l'année.

Utilisez-la ensuite pour créer une fonction `compareAmiChrono(ami1,ami2)` qui compare deux amis selon leur date d'anniversaire.

Amis

Prénom	Nom	Naissance
Morgana	Aubin	née le 25 mars 1960
Charles	Ber	né le 22 avril 1989
Paul-Édouard	de la Riquette	né le 23 juin 1931
Bernadette	Ricard	née le 24 juillet 1995
Charmaine	Ricard	né le 27 juillet 1994
Cunégonde	de la Riquette	née le 6 septembre 1984
Adélaïde	Berie	née le 30 octobre 1973
Gabriel	Lang	né le 12 novembre 1995
Gérard	Menrhy-Otant	né le 14 décembre 1968

Étape 6 : quelques colonnes de plus

Ajoutez trois colonnes aux tableaux. Ces colonnes seront intitulées « 2016 », « 2017 » et « 2018 » et indiqueront, pour chacune de ces trois années,

- l'âge que l'ami en question atteindra au moment de son anniversaire ;
- le jour qui correspond à son anniversaire (lundi, mardi, mercredi, jeudi, vendredi, samedi ou dimanche) (servez-vous des méthodes de l'objet prédéfini `Date`).

Amis

Prénom	Nom	Naissance	2016	2017	2018
Morgana	Aubin	née le 25 mars 1960	56 ans (vendredi)	57 ans (samedi)	58 ans (dimanche)
Charles	Ber	né le 22 avril 1989	27 ans (vendredi)	28 ans (samedi)	29 ans (dimanche)
Paul-Édouard	de la Riquette	né le 23 juin 1931	85 ans (jeudi)	86 ans (vendredi)	87 ans (samedi)
Bernadette	Ricard	née le 24 juillet 1995	21 ans (dimanche)	22 ans (lundi)	23 ans (mardi)
Charmaine	Ricard	né le 27 juillet 1994	22 ans (mercredi)	23 ans (jeudi)	24 ans (vendredi)
Cunégonde	de la Riquette	née le 6 septembre 1984	32 ans (mardi)	33 ans (mercredi)	34 ans (jeudi)
Adélaïde	Berie	née le 30 octobre 1973	43 ans (dimanche)	44 ans (lundi)	45 ans (mardi)
Gabriel	Lang	né le 12 novembre 1995	21 ans (samedi)	22 ans (dimanche)	23 ans (lundi)
Gérard	Menrhy-Otant	né le 14 décembre 1968	48 ans (mercredi)	49 ans (jeudi)	50 ans (vendredi)