TUTORIEL DART
SESSION 01 LES BASES

" JavaScript tel qu'il est aujourd'hui n'est pas une solution viable à long terme et quelque chose doit changer ", ce quelque chose c'est la naissance d'un langage de programmation objet orienté Web appelé DART.

Introduction

Dart est un langage de programmation structuré **pour le web** purement **orienté objet**, les objets qu'il manipule peuvent être des classes ou des interfaces (que nous verrons plus loin). Si vous êtes habitué à développer en Java, vous n'aurez aucun problème avec ce langage assez prometteur.

Objectifs de Dart

1. Créer un langage structuré mais flexible pour la programmation Web.
2. Rendre DART familier et naturel pour les programmeurs, et donc facile à assimiler.
3. S'assurer que DART fournit de très hautes performances sur tous les navigateurs et les environnements modernes, des appareils mobiles aux fermes de serveurs.

TUTORIEL DART
SESSION 01 LES BASES

Mise en marche

Il y a plusieurs méthodes permettant d'"exécuter" un script Dart en attendant que les navigateurs supportent pleinement le langage.

1. Par Installation: dart_bin

- Windows. (*attention dart_bin n'est pas stable sur WIndows XP*)
- Linux (Ubuntu 11.04 32bits).
- Mac OS X.

Les fichiers Dart ont l'extension ".dart" et peuvent être exécutés en lançant la commande:

```
$> ./dart_bin [options] monscript.dart
```

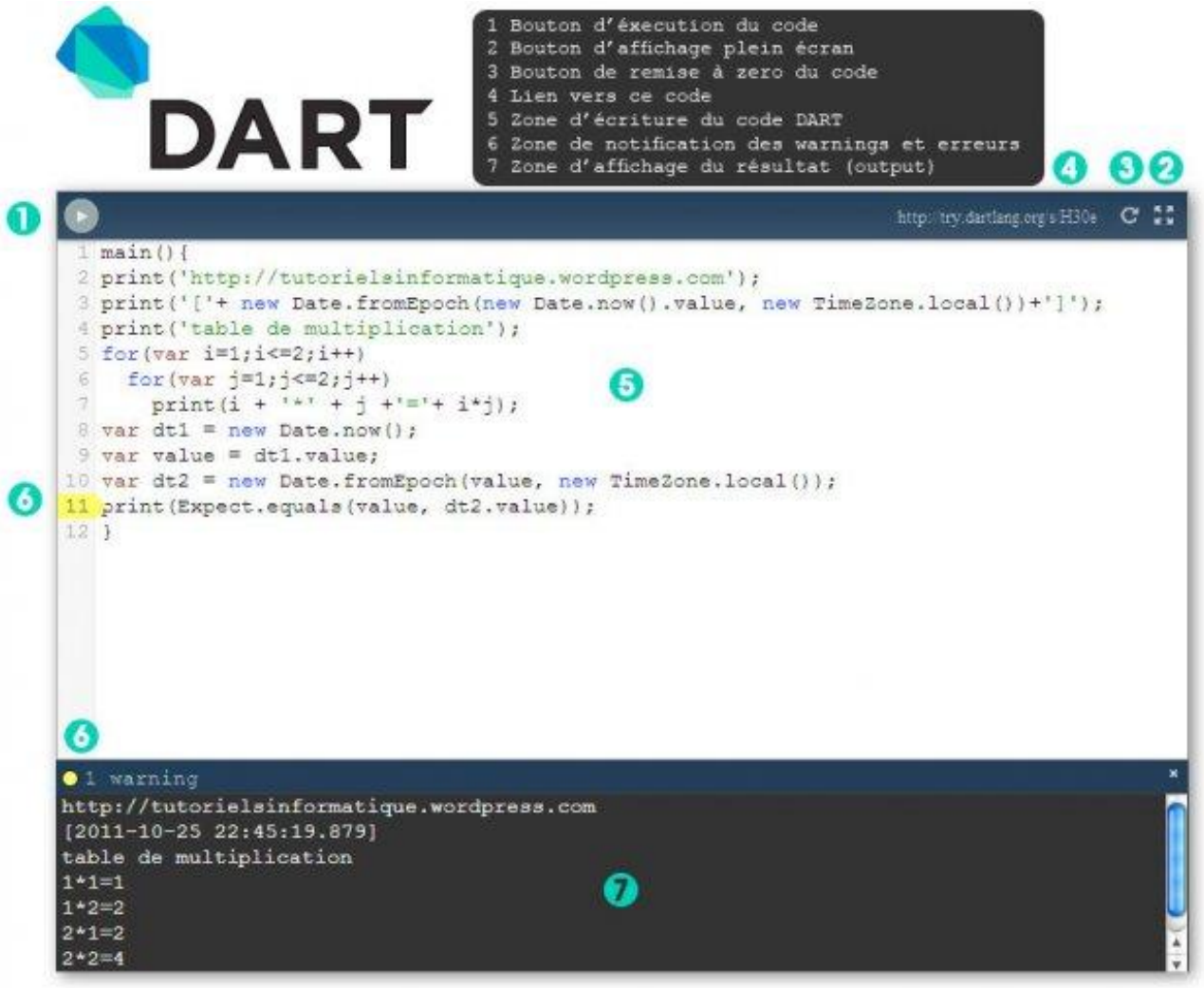
Voici les options possibles:

Options du compilateur	
–batch	Mode " Batch " pour les tests unitaires
–check-only	Ne fait que de l'analyse, sans génération de résultat
–expose_core_impl	Import automatique de dart:coreimpl library
–warn_no_such_type	traite les détections de test de type comme un avertissement et non une erreur fatale
–enable_type_checks	Génère le runtime (moteur d'exécution) test de type
–disable-type-optimizations	Désactive l'optimisation des types (for debugging)
–documentation-lib	Génère la documentation pour une bibliothèque donnée
–documentation-out	Répertoire pour la librairie
–generate-documentation	Génération du document de la source
–generate-isolate-stubs	Vérification d'état isolé
–generate_source_maps	Généré la carte des sources
–human-readable-output	Ecriture en langage humain compréhensible
–ignore-unrecognized-flags	Ignore les attributs inconnus
–isolate-stub-out	Fichier qui reçoit la vérification d'état
–jvm-metrics-detail	Affichage de données sommaire (verbose/stats)
–jvm-metrics-format	Modifie l'affichage des métriques
–jvm-metrics-type	Liste des sous argument du contrôle de l'affichage:+ " all: (par défaut) tout type de statistique + " gc: montre les collections de statistique " garbage " + " mem: montre des statistiques sur l'état de la mémoire + " jit: montre les statistique " jit "
–noincremental	Désactive la compilation incrémentale
Optimisation du Javascript généré	

–optimize	Produit un code optimisé
–out	Généré le code JavaScript dans un fichier
–help	Affiche le message d'aide
–jvm-metrics	Affiche les données JVP en fin de compilation
–metrics	Affiche les données de compilation
–fatal-type-errors	Considérer les erreurs de types comme fatale
–fatal-warnings	Traite les avertissement de non typage comme fatale
Options acceptées par le DartRunner	
–compile-only	Compilation sans exécution
–verbose	Affichage des données de diagnostic
–prof	Active l'analyse
–rhino	Utiliser Rhino comme interpréteur JavaScript

2. Compilation en ligne: le "Dartboard"

Il suffit d'aller sur try.dartlang.org, l'interface est simple, c'est celle que nous utiliserons pour ces sessions, elle est composée de 7 parties:



TUTORIEL DART
SESSION 01 LES BASES

Règles de base

Analyse du code

Tout script Dart ne peut être lancé que s'il passe par des étapes préliminaires d'analyse et de gestion parallèle des erreurs. Les étapes d'analyses sont les suivantes:

- **Analyse lexicale (lexer ou scanner)** – elle reconnait les caractères (ou chaines de caractères) d'entrée (les identificateurs, les mots réservés [while, print...], les opérateurs arithmétique et les affectations) et produit un flux d'unités lexicales (ou lexème) que vous reconnaitrait sur Dart par le terme de ***tokens***. ces chaines de caractères sont délimitées par les séparateurs suivant:

    - les espaces et tabulations (règle dite " skip " ou chaine ignorée)
    - les sauts de lignes (règle dite " skip " ou chaine ignorée)
    - les commentaires (règle dite " special_token ")

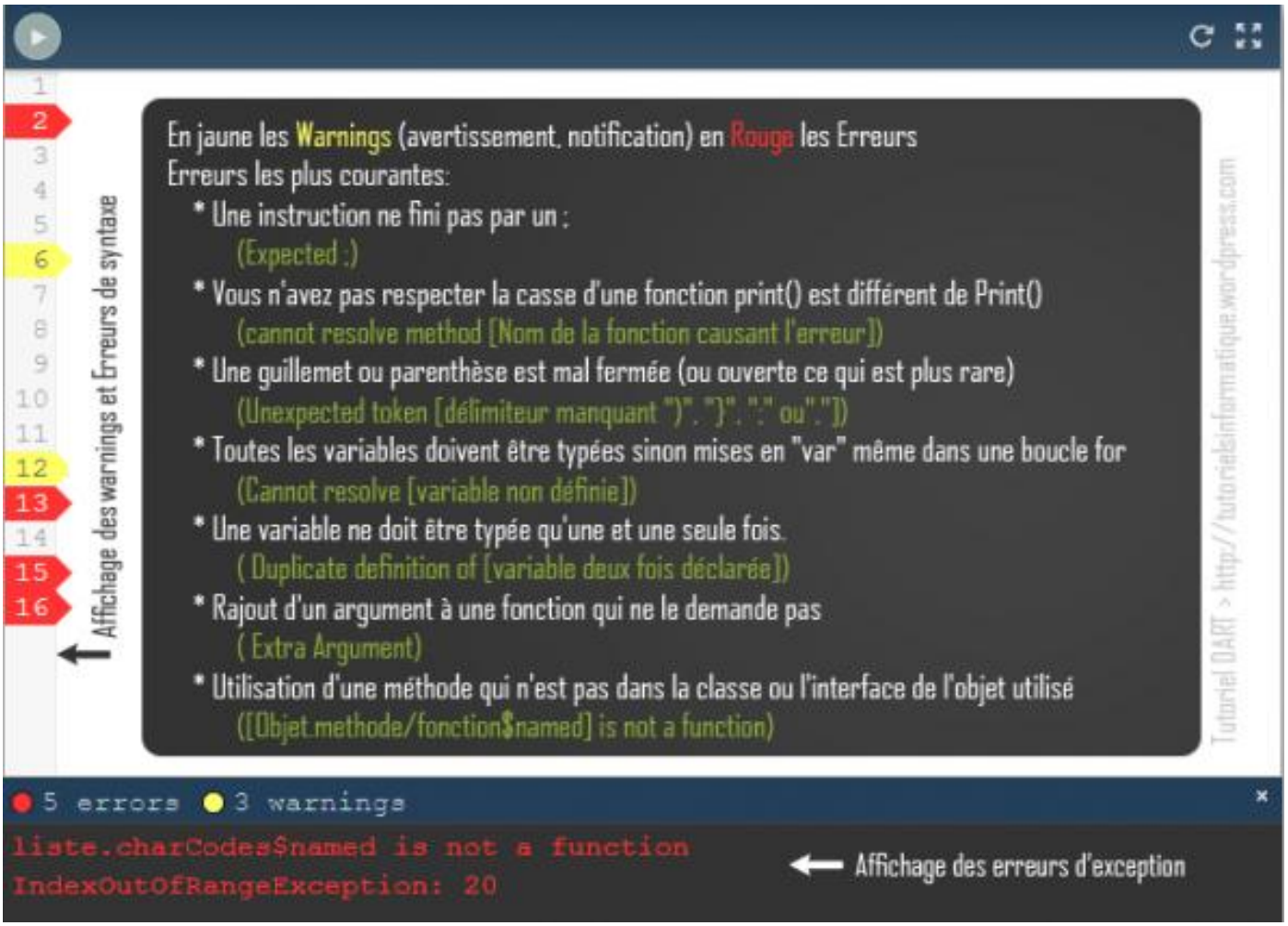
Exemple de quelques token Dart:

LPAREN	(
RPAREN	)
LBRACE	{
RBRACE	}
COLON	:
SEMICOLON	;
ASSIGN	=
ASSIGN_ADD	+=
OR	||
AND	&&
ADD	+
SUB	-
MUL	*
EQ	==
NE	!=
LT	<
GT	>
NOT	!
INC	++
CATCH	catch
FINAL	final
FOR	for
IF	if
TRY	try
VAR	var
VOID	void

- **Analyse syntaxique (parser)** – elle s'occupe de l'ordonnancement des unités lexicales, cela consiste à prendre ce flux de lexème (ayant une structure linéaire) et à le transformer en un arbre de syntaxe abstraite plus simple à interprété. Un arbre est composé de nœud, chaque nœud est une opération et chaque " fils " d'un nœud représente les arguments de cette opération. Cette arborescence incite a ce que l'interprétation d'un nœud ne se fasse qu'après avoir interprété les sous arbre de ce nœud.
- **Analyse sémantique** – elle vérifie la cohérence des instructions en se basant sur des règles qui garantiront que le script pourra s'exécuter normalement.

Gestion des erreurs

La gestion des Erreurs est particulièrement bien implantée sur Dartboard, comme le montre l'image en dessous. Dans la partie supérieure, on trouve les warning ou avertissements (en jaune) et les erreurs de pre-exécution (en rouge). Dans la partie inférieur on à l'affichage des erreurs plus profondes et imprévisibles. Ce sont des erreurs que l'on peut gérer manuellement (nous dédierons une session sur les options de compilation et la gestion des erreurs).



Notre premier script Dart Vierge

Comme en Java, Dart commence par la méthode main()

```
main(){
}
```

[Tester le code]

L'affichage à l'écran

*L'affichage sur Dart est réalisée par la fonction **print()**, fonction qui n'a qu'un seul argument (le contenu à affiché sur la sortie standard). L'élément à afficher doit être mis entre simple quote ou double quote, vous verrez qu'on peut faire des concaténations de chaine de caractère, utiliser des constantes mathématiques, insérer de l'interpolation d'opérations arithmétique, des fonctions ou utiliser un codage ascii tel que \n, \t et \uxxxx, l'utilisation des variables sera vu plus loin.*

```
main(){
print ('Bonjour le monde!');
print (" Hello the World! ");
print (" bienvenue " + " sur " + " tutorielsinformatique.wordpress.com ");
print ('nous verrons plus tard' + 'comment intégrer des variables' + " simplement ");
print ('je rajoute ici une tabulation \t voila 😀');
print ('Et la un saut de ligne \n et Hop! 😀');

print (" on peut faire plusieurs saut de ligne \n \n \n fin du message          " );
print (" Ligne 1 \nLigne2 \n Lign3 \u00A0 Ligne 4 \n Ligne 5");
print (" pour les caractère unicode on fait: (Bonjour) ");
print (" \u0042\u006f\u006e\u006a\u006f\u0075\u0072");
print (" Pour consulter tous les codes, visitez: ");
print (" http://www.unicode.org/charts/ ");
print (" on peut faire de l'interpolation entre accolades précédées d'un \$ ");
print ('on peut utiliser des fonctions en interpolation:');
print (" Nous sommes ${5+5-(8%5)} milliard d'habitant sur Terre ");
print ('ou en concaténation avec la chaine à afficher:');
print (" le nombre PI est égale à: "+ Math.PI);
print (" que l'on peut arrondir à: "+Math.PI.toStringAsFixed(2));
print (" on peut afficher toutes sortes de caractères: ");
print (" ☮ ✈ ♋ 웃 유 ☠ ☯ ♥ ✌ ✖ ☢ ☣ 🍸 🀰 ❁ ♨ ❖ ❤ ❥ ❦ ❧ ♡ ✗ ✘ ⊗ ♒ Ω ♦ ♠ ♥ ♣ ♢ ♤ ♡ ♧ ✦ ✧ ♕ ♕ ♕ ♛ ★ ☆
 ✯ ✭ ✍ ☾ ☽ ☼ ☀ ☘ ☂ ☃ ☻ ☺ 🏛 ♪ ♫ ♫ ✄ ✂ ✆ ✉ ∞ ♂ ♀ ☿ ▲ ▼ △ ▽ ◆ ◇ ◕ ◔ ◡ ϟ ᘓ 回 ₪ ✓ ✔ ✕ ☥ ☨ ☧ ☦ ☩ C·
 ⍟ ⍦ ⛐ ™ © ® ¿¡ №⇨ ❝ ❞ ∃ ⊥ ∀ Ξ ∞ Σ ⊓ ⌥ ⌘ 文 Ψ ");
}
```

[Tester le code]

```
main(){
print(" \u00ce\u00f1\u0163\u00e9r "
+ " \u00f1\u00e5\u0163\u0129\u014d\u00f1\u01fb\u013c\u00ee\u015f "
+ " \u00e5\u0163\u00ee\u1edd\u019e ");
```

```
print(" Îñţérñåţĩōñåļîşåţîờn ");
}
```

[Tester de l'internationalisation]

Les commentaires

Les lignes de commentaires sont des lignes que l'on rajoute pour expliquer son code, pour marquer un endroit, pour rajouter des informations de licence (cc). Sur Dart comme sur JavaScript tout ce qui est après deux slash " // " est considéré comme un commentaire et tout ce qui est entre / et */ est considéré comme un bloc de commentaires. Ils font partie dans la compilation / interprétation des éléments qui seront omis par l'analyse syntaxique.*

```
// tutorielsinformatique.wordpress.com
// les lignes 3, 5, 6, 7, 8 et 10 ne seront pas affichées
// ce sont soit des lignes de commentaires
// soit un bloc commenté
main(){
print('ligne n°1 affichée');
print('ligne n°2 affichée');
// print('ligne n°3 non affichée');
print('ligne n°4 affichée');
/* print('groupe de lignes (n°5) non affichée');
print('groupe de lignes (n°6)non affichée');
print('groupe de lignes (n°7)non affichée');
print('groupe de lignes (n°8)non affichée');*/
print('ligne n°9 affichée');// print('ligne n°10 non affichée');
}
```

[Tester le code]

Les variables

- *On peut déclarer les variables tout comme en JavaScript via le terme " **var** " (qui donne le type implicit any) ou en donnant le type exact (**num, int, double, bool, String**)*
- *Ou alors en définissant explicitement la variable par exemple: var i=1; et var j= "1"; la variable (i) est un entier tandis que la variable (j) est une chaine de caractère String (S en majuscule).*
- *Une variable ne peut être déclaré (je parle du type et non de la valeur) qu'une et une seule fois!*
- *Une mauvaise définition de type de variable crée généralement un avertissement (warning) sans pour autant que la compilation ou l'exécution du code soit stoppée.*
- *Une variable de type **final**, ne pas être initialisé que lors de sa déclaration.*
- *Une variable non initialisée est égale à **null**.*

```
main(){
var text0 = "DART";
String text1 = "DART";
int text2 = "DART";
print(text0);
print(text1);
print(text2);
}
```

En exécutant ce code on obtient un warning disant qu'on ne peut pas assigner de texte dans la variable text2 qui est un entier

[Tester le code]

Remarque: la fonction ***print*** n'accepte qu'un seul argument (donc pas de virgule de séparation d'arguments sinon cela causera le warning " extra argument ") [Tester le code]

Le code suivant montre que Dart est souple au niveau des types mais il faut quand même faire attention àBenfarhat ne pas trop se laisser aller en donnant n'importe quel type ce qui pourrait donner un résultat différent de celui escompté.

```
main(){
int a=1,b=2;
String c="1",d="2";
print (a+b);
print (c+d);
}
```

[Tester le code]

Rappelez vous qu'une variable marquée final doit absolument être initialisée lors de la déclaration !

```
main() {
// la variable i peut être initialisé plus tard
int i;
i=1;
// la variable j (final) doit être initialisé lors de la déclaration
final j;
j=1;
print (" $i $j ");
}
```

[Tester l'exemple]

Affichage d'un texte

Elle s'opère grâce à la fonction " print "! Vous remarquerez que chaque instruction finit par un point-virgule " ; "

```
main(){
var blog="tutorielsinformatique.wordpress.com";
print('ceci est une ligne');
print('ceci est une autre ligne');
print ('bienvenue sur '+blog);
}
```

[Tester le code]

Plus loin avec les variables String

Un triple " double quote " (ou triple quotes simple également) permet de délimiter une variable texte sur plusieurs lignes. Vous constaterez dans l'exemple ci dessous que l'on peut même intégrer une autre variable à l'intérieur (ce n'est pas une variable dynamique!!! le contenu de la variable multi-lignes est affecté une seule fois).

```
main(){
String nom= "Benfarhat elyes ";
var message = "  "" Bonjour $nom
+————————————————————————+
| Vous êtes invité à vous présenter                  |
| .. à nos locaux  pour faire le point sur la mise en place |
| d'applications Dart                                |
+————————————————La Direction———+
"  "";
nom= "Thomas Ben ";
print(message);
}
```

[Tester le code]

Interpolation et évaluation d'une variable

En condition normal les variables ne sont pas précédés d'un dollar '$' sauf à l'intérieur d'une chaine de caractère pour permuter avec le contenu de la variable. Il peut être intéressant de rajouter des guillemets si l'on désire avant interpolation faire une évaluation de la valeur de ce qui est entre guillemet. Ce qui est l'équivalent en JavaScript de ces lignes de code

```
var str = " 8 + 12)

document.write(eval(str))
```

Dans cette exemple j'ai mis plusieurs combinaisons qui je l'espère seront claire quant à l'utilisation du dollar et de l'importance du " *typage* " lors d'une évaluation.

```
main(){
int age=36;
String text="36";
print("Cette année j'ai "+age+" ans");
print("Cette année j'ai age ans");
print("Cette année j'ai $age ans");
print("L'année prochaine j'aurais $age +1 ans");
print("L'année prochaine j'aurais ${age +1} ans");
print("L'année prochaine j'aurais ${text +1} ans");
}
```

[Tester le code]

Appel d'une fonction sans argument

la fonction est définie avant main()

```
int mafonction(){
print('message écrit dans la fonction "mafonction"');
}
main(){
mafonction();
}
```

[Tester le code]

Appel d'une fonction avec argument

```
int mafonction(final nom){
print('bonjour '+nom+'!');
}

main(){
mafonction('toi');
mafonction('vous');
mafonction('dart');
}
```

[Tester le code]

La suite des sessions …

Nous avons vu la première partie de plusieurs sessions de formation DART. Dart est un nouveau langage qui gagne à être connu, très prometteur, il attire vers le web les développeurs java et offre beaucoup d'espérance si ses promesses sont tenus, Dart est en constante évolutions (voir les nouveautés), il ne peut que nous surprendre

Dans nos prochaines sessions nous verrons les Structures de contrôles Dart, la manipulation des nombres (fonctions mathématiques, trigonométrique, de conversion de base), la manipulation des chaines de caractères (concaténation, transformation, recherche), la manipulation des dates, Les interfaces List<E>, Set<E> et Queue<E>, les interfaces de manipulation de données composées Map<K,V>, la gestion des Exceptions, Les Expressions Régulières…

Pour plus d'infos

- Site de Google Dart: http://www.dartlang.org/
- Site de dartboard: http://try.dartlang.org/
- Suivi de l'évolution du code de Dart: http://code.google.com/p/dart/