



DÉVELOPPEMENT INFORMATIQUE LOGIQUE & PROGRAMMATION

LANGUAGE JAVASCRIPT

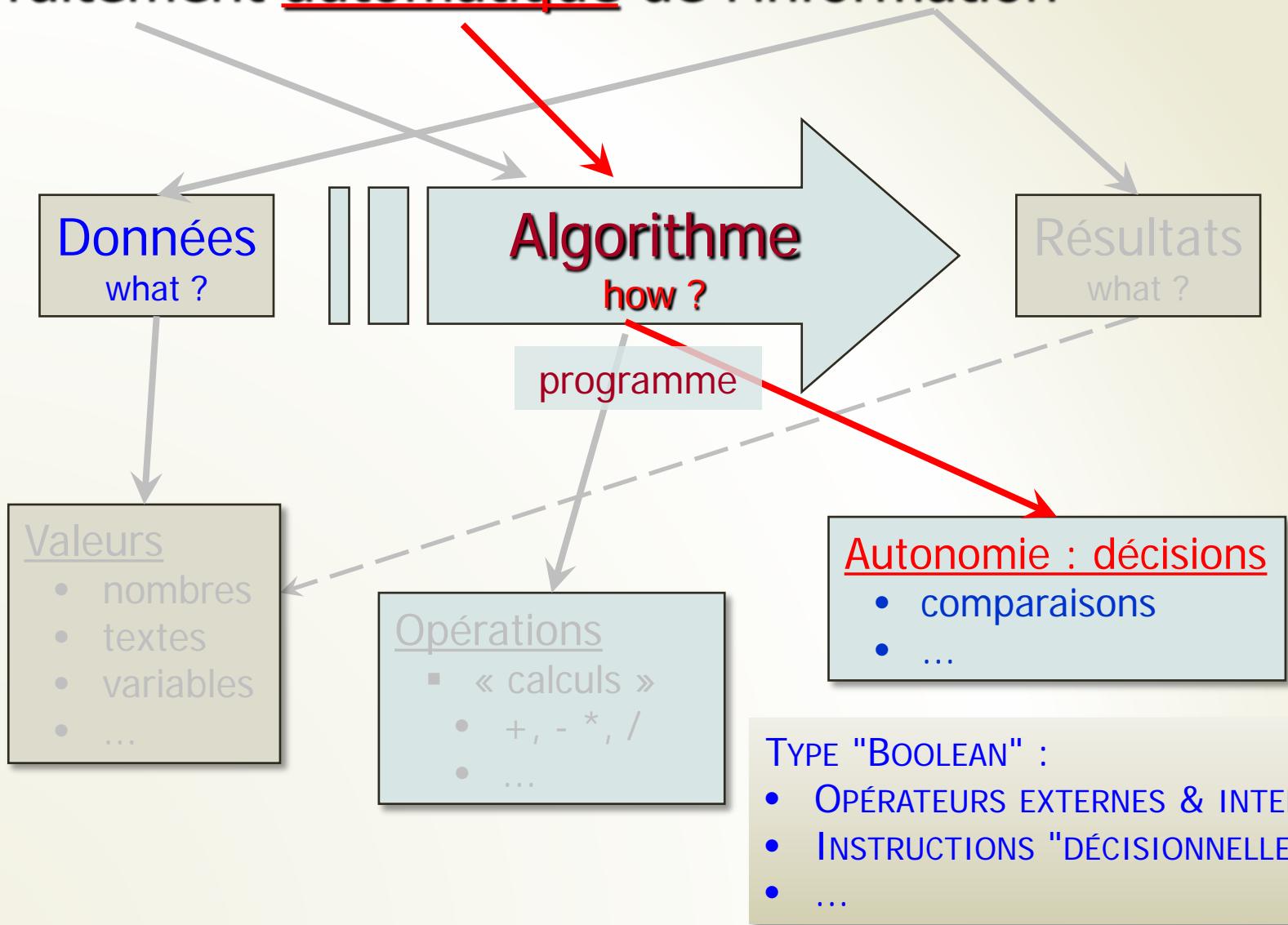
Y. DELVIGNE

CH. LAMBEAU

CONCEPTS FONDAMENTAUX :

TRAITEMENT
AUTOMATIQUE
DE L'INFORMATION

Traitement automatique de l'information



Bien percevoir le problème ! (diverses expressions)

- 10% de réduction à partir de 11 pièces !
- toute commande de plus de 10 pièces donne droit à une réduction de 10%
- une réduction de 10% est accordée à condition que la quantité commandée soit supérieure à 10
- si **quantité > 10** alors **réduction = 10%**

évaluation d'une condition
par comparaison de valeurs

action/affectation à exécuter
si la condition est vérifiée

```
var quantite, reduction;  
quantite = +prompt('quelle quantité ?');  
if (quantite > 10) { reduction = 0.1; }
```



RAPPEL : TYPE DE DONNÉES

- ENSEMBLE HOMOGÈNE DE VALEURS (UNIQUES)

1 est unique parmi les entiers

'A' est unique dans l'alphabet

"hello" est unique dans les chaînes de caractères

- EXISTENCE D'UN ORDRE (→ COMPARAISONS, CONDITIONS ...)

-2 < -1 < 0 < 1 < 2 ...

'A' < 'B' < 'C' ... 'A' < 'a' "papaye" < "pepsi"

- CET ORDRE EST 'TOTAL'

grâce à l'unicité : entre les valeurs, on a < (ou >)

L'OUTIL DÉCISIONNEL EST LA COMPARAISON DE VALEURS

→ OPÉRATEURS DE COMPARAISON (RELATIONNELS) :

STRICTEMENT INFÉRIEUR À <

INFÉRIEUR (OU ÉGAL) À <=

ÉGAL À ==

DIFFÉRENT DE !=

SUPÉRIEUR (OU ÉGAL) À >=

STRICTEMENT SUPÉRIEUR À >

parce que =
utilisé par
l'affectation

→ EXPRESSIONS DE COMPARAISON (RELATIONNELLES) :

(«valeur» «opérateur» «valeur»)

au sens large : - valeur littérale
 - contenu de variable
 - expression

(x < 10)
(x != y - 5)
(x == undefined)

LES COMPARAISONS N'ONT DU SENS QUE PARCE QUE LES TYPES DE DONNÉES SONT ORDONNÉS

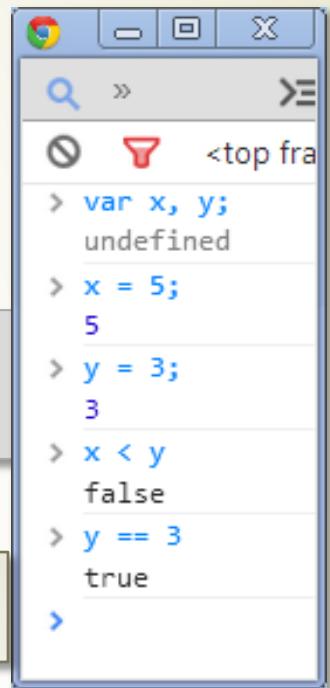
TOUTE COMPARAISON EST EN FAIT UNE QUESTION DONT LA RÉPONSE NE SAIT ÊTRE QUE **OUI** OU **NON** (**VRAI** OU **FAUX**)

➤ TYPE PRIMITIF "**boolean**" :

- ne contient que deux valeurs : **true** et **false**
- dispose d'opérateurs spécifiques

TOUTE EXPRESSION RELATIONNELLE EST ÉVALUÉE ET RENVOIE SOIT **true** SOIT **false**

pas de " "
ce n'est pas un string



```

var x, y;
undefined
> x = 5;
5
> y = 3;
3
> x < y
false
> y == 3
true
>

```

LES EXPRESSIONS MIXTES (MÉLANGEANT "NUMBER" ET "STRING") DE JAVASCRIPT PEUVENT POSER PROBLÈME

→ DEUX CONCEPTIONS DE L'ÉGALITÉ ET DE LA DIFFÉRENCE

- ÉGALITÉ 'EN VALEUR' : $10 == '10'$ (true)
- INÉGALITÉ 'EN VALEUR' : $'14.5' != 10$ (true)
 $'moi' == 15$ (false)

- LE 'STRING' EST CONVERTI EN 'NUMBER' AVANT COMPARAISON
- SI LE 'STRING' NE CONTIENT PAS UN LITTÉRAL NUMÉRIQUE,
LA COMPARAISON D'ÉGALITÉ S'ÉVALUE TOUJOURS À FALSE

chaleureusement recommandé actuellement

- ÉGALITÉ EN 'TYPE ET VALEUR' $10 === '10'$ (false)
- INÉGALITÉ EN 'TYPE ET VALEUR' $10 !== '10'$ (true)

ALGORITHMIQUE : LES INSTRUCTIONS DE CONTRÔLE

PERMETTENT DE PRENDRE UNE DÉCISION SUR BASE D'UNE CONDITION (EXPRESSION RELATIONNELLE)

C.-À-D. D'EXÉCUTER (OU PAS) UNE (OU PLUSIEURS) INSTRUCTIONS (SÉQUENCE)

INSTRUCTION CONDITIONNELLE

**SI «CONDITION» ALORS
«SÉQUENCE»
FINSI**

si la condition est évaluée **vrai**,
la séquence est exécutée

INSTRUCTION ALTERNATIVE

**SI «CONDITION» ALORS
«SÉQUENCE1»
SINON
«SÉQUENCE2»
FINSI**

si la condition est évaluée **vrai**,
la séquence1 est exécutée
si la condition est évaluée **faux**,
la séquence2 est exécutée

TRAITEMENT : AUTONOMIE - DÉCISION (6)

ALGORITHMIQUE :

INSTRUCTION CONDITIONNELLE

**SI «CONDITION» ALORS
«SÉQUENCE»
FINSI**

JAVASCRIPT :

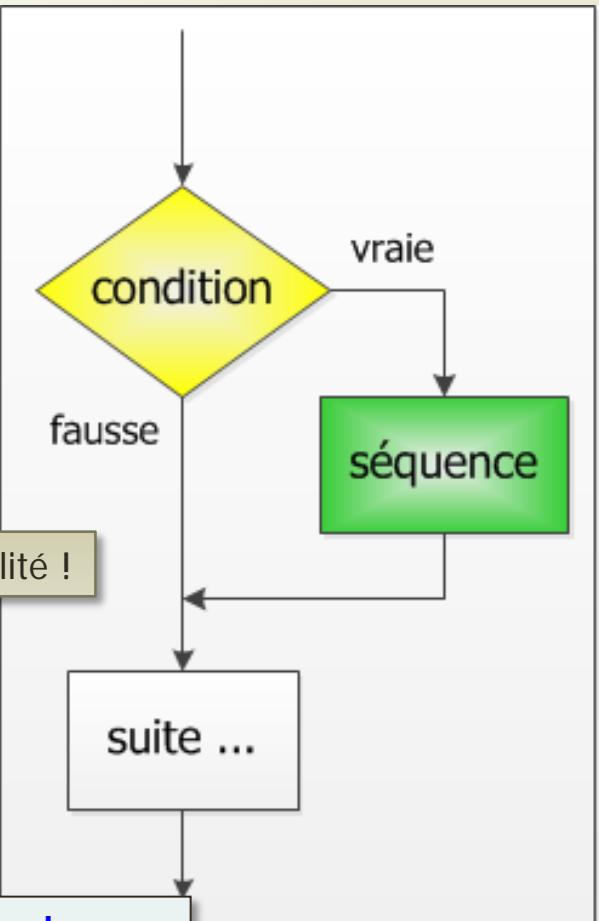
INSTRUCTION CONDITIONNELLE

```
if («condition») {
    «séquence»
}
```



indenter = lisibilité !

```
var prixUnitaire = 15.0, quantite, prix;
quantite = +prompt('quelle quantité ?');
if (quantite > 10)
    { prixUnitaire *= 0.9; }
prix = prixUnitaire * quantite;
```



TRAITEMENT : AUTONOMIE - DÉCISION (7)

ALGORITHMIQUE :

INSTRUCTION ALTERNATIVE

**SI «CONDITION» ALORS
 «SÉQUENCE1»
SINON
 «SÉQUENCE2»
FINSI**

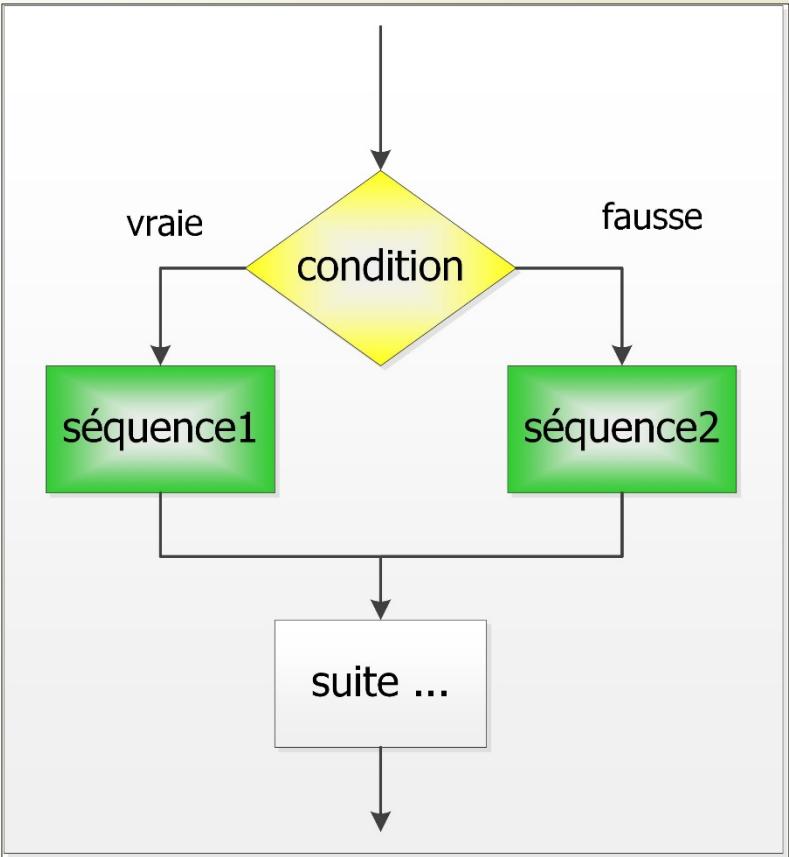
JAVASCRIPT :

INSTRUCTION ALTERNATIVE

```
if (<>condition</>) {
  <>séquence1</>
} else {
  <>séquence2</>
}
```



indenter = lisibilité !



TRAITEMENT : AUTONOMIE - DÉCISION (8)

JAVASCRIPT :

INSTRUCTION ALTERNATIVE

```
if («condition») {  
    «séquence1»  
} else {  
    «séquence2»  
}
```



```
var nbMsg; // nombre de messages  
  
nbMsg = prompt('combien de messages ?');  
  
if (nbMsg === '1') // ou (nbMsg == 1)  
    { alert('vous avez 1 message'); }  
else  
    { alert('vous avez ' + nbMsg + ' messages'); }
```

"TRUTHY" ET "FALSY"

LES VALEURS DES TYPES PRIMITIFS ONT TOUTES UN ÉQUIVALENT DE 'VALEUR DE VÉRITÉ' (**true/false**) LORSQU'ELLES APPARAISSENT EN TANT QU'EXPRESSION CONDITIONNELLE **if** (**<<value>>**) ...

type	valeur	vérité
"undefined"	undefined	false
"number"	0	false
"number"	!= 0	true
"number"	NaN	false
"number"	Infinity	true
"string"	""	false
"string"	!= ""	true

j'ai utilisé `alert()` parce que `console.log()` prend trop de place dans la boîte ☺

rouge = texte affiché

```

var v;
if (v) alert('def'); else alert('undef');

v = 0;
if (v) alert('non nul'); else alert('nul');

v = -3.5;
if (v) alert('non nul'); else alert('nul');

v = "";
if (v) alert('non vide'); else alert('vide');

v = "yop";
if (v) alert('non vide'); else alert('vide');

```

2 FONCTIONS BOOLÉENNES (QUI RENVOIENT TRUE/FALSE)

PRÉDÉFINIES DANS LE 'CORE' JAVASCRIPT

isFinite(«expression») : détermine si l'expression numérique est finie ou non

isNaN(«expression») : détermine si l'expression est numérique ou non

```
isFinite(25)                      // true
isFinite(111/0)                    // false
isFinite(undefined)                // false

isNaN(0)                          // => false
isNaN(0/0)                        // => true
isNaN(parseInt("3"))              // => false
isNaN(parseInt("hello"))           // => true
isNaN("3")                         // => false
isNaN("hello")                     // => true
isNaN(true)                        // => false
isNaN(undefined)                   // => true
```

1 FONCTION BOOLÉENNE (QUI RENVOIE TRUE/FALSE)

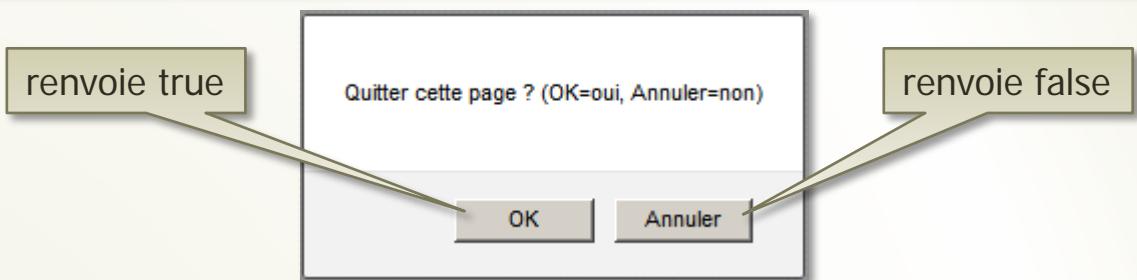
PRÉDÉFINIE DANS LE NAVIGATEUR ...

... COMME LE SONT ALERT() ET PROMPT()

confirm(«string») : pose une question 'oui/non' à l'utilisateur

beaucoup plus simple que d'utiliser prompt() et de devoir gérer la réponse

```
confirm('Quitter cette page ? (OK=oui, Annuler=non)')
```



```
var veutQuitter; // variable booleenne
veutQuitter = confirm('Quitter cette page ? (OK=oui, Annuler=non)');
if (veutQuitter) {
    // ...
} else {
    // ...
}
```

ON RÉSUME ?

➤ TYPES PRIMITIFS

- "number" + Infinite et NaN
- "string"
- "undefined" 1 valeur : undefined
- "boolean" 2 valeurs : true, false

➤ OPÉRATEURS & FONCTIONS

- "number" +, -, *, /, %, ()
- "string" +
- affectation =, +=, -=, *=, /=, %=, ++, --
- comparaison <, <=, >, >=,
==, !=, ===, !==
- "boolean" **isFinite()**, **isNaN()**, **confirm()**

CONDITIONNELLE

```
if («condition») {  
    «séquence»  
}
```

ALTERNATIVE

```
if («condition») {  
    «séquence1»  
} else {  
    «séquence2»  
}
```

LE DANGER DE **UNDEFINED !**

```
var prixUnitaire = 15.0, quantite, prix;
var tauxReduction;
quantite = +prompt('quelle quantité ?');
if (quantite > 10)
    { tauxReduction = 0.1; }
prix = prixUnitaire * (1 - tauxReduction) * quantite;
```

que vaut-il
si quantité <= 10 ??
undefined !
donc prix vaut
NaN!!

UNE RÈGLE ABSOLUE :

une variable ne peut apparaître comme rvalue que si elle a fait l'objet d'une affectation préalable (donc en tant que lvalue)

DIVERSES POSSIBILITÉS ...

```
if (quantite > 10)
    { tauxReduction = 0.1; }
else
    { tauxReduction = 0.0; }
```

```
tauxReduction = 0;
if (quantite > 10)
    { tauxReduction = 0.1; }
```

```
var tauxReduction = 0;

if (quantite > 10)
    { tauxReduction = 0.1; }
```

TRAITEMENT : AUTONOMIE - DÉCISION (13)

CONDITIONS "EN CASCADE" (VARIABLES MULTIVALUÉES) : LOURDEUR !

indentation = lisibilité ! ☺☺

TRAITEMENT : AUTONOMIE - DÉCISION (14)

CONDITIONS "EN CASCADE" (VARIABLE MULTIVALUÉE) : ALTERNATIVE

```

var jour, libelle;
jour = +prompt('numéro de jour (1..7) ?');
if (jour === 1)
{ libelle = 'lundi'; }
else if (jour === 2)
{ libelle = 'mardi'; }
else if (jour === 3)
{ libelle = 'mercredi'; }
else if (jour === 4)
{ libelle = 'jeudi'; }
else if (jour === 5)
{ libelle = 'vendredi'; }
else if (jour === 6)
{ libelle = 'samedi'; }
else if (jour === 7)
{ libelle = 'dimanche'; }
else
{ libelle = 'erreur'; }

```

même niveau pour marquer
le caractère multivalué de la variable

ALGORITHMIQUE

SI «CONDITION1» **ALORS**
 «SÉQUENCE1»
SINONSI «CONDITION2» **ALORS**
 «SÉQUENCE2»
SINONSI «CONDITION3» **ALORS**
 «SÉQUENCE3»
 ...
SINON
 «SÉQUENCEN»
FINSI

autre outil nettement plus 'simple'
dans un chapitre ultérieur :
switch()

TRAITEMENT : AUTONOMIE - DÉCISION (15)

CONDITIONS "IMBRIQUÉES" (PLUSIEURS VARIABLES IMPLIQUÉES)

```
var sexe, etatCivil, message = 'bonjour ';  
sexe = prompt('sexe (m/f) ?');  
etatCivil = prompt('état civil (c/m/d/v) ?');  
  
if (sexe === 'm')  
    { message += 'monsieur'; }  
else {  
    if (etatCivil === 'c')  
        { message += 'mademoiselle'; }  
    else  
        { message += 'madame'; }  
}
```

ici indenter
parce que
2 variables
distinctes

NÉCESSITÉ DE TRAITER DES CONDITIONS PLUS COMPLEXES

- COMBINAISON DE COMPARAISONS SIMPLES
- S'EXPRIMANT À L'AIDE DE TROIS OPÉRATEURS LOGIQUES :
OU, ET, NON

SI LE SEXE EST '*m*' OU '*M*' ...

SEXE = 'm' OU **SEXE = 'M'**

SI L'ÂGE EST ENTRE 18 ET 25 ANS ...

AGE >= 18 ET **AGE <= 25**

SI LA RÉPONSE N'EST PAS PARMI 'o', 'O', 'n', 'N' ...

REONSE <> 'o' ET **REONSE <> 'O'** ET ...

...

TRAITEMENT : AUTONOMIE - DÉCISION (17)

➤ TYPE PRIMITIF "boolean" :

- ne contient que deux valeurs : **true** et **false**
- dispose d'opérateurs internes spécifiques :
 - **||** (or, ou)
 - **&&** (and, et)
 - **!** (not, non)

internes :
les opérandes sont "boolean"
et le résultat est "boolean"

OU

a	b	a b
true	true	true
true	false	true
false	true	true
false	false	false

ET

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

NON

a	!a
true	false
false	true

TRAITEMENT : AUTONOMIE - DÉCISION (18)

EXEMPLES : OU (||)

```
var sexe, etatCivil, message = 'bonjour ';
```

```
sexe = prompt('sexe (m/f) ?');
```

```
etatCivil = prompt('état civil (c/m/d/v) ?');
```

```
if (sexe === 'm' || sexe === 'M') // masculin ?
```

```
 { message += 'monsieur'; }
```

```
else if (sexe === 'f' || sexe === 'F') // féminin ?
```

```
 { if (etatCivil === 'c' || etatCivil === 'C')
```

```
     { message += 'mademoiselle'; }
```

```
     else if (etatCivil === 'm' || etatCivil === 'M')
```

```
         { message += 'madame'; }
```

```
     // ...
```

```
 }
```

```
else
```

```
 { message = 'erreur de sexe !'; }
```

TRAITEMENT : AUTONOMIE - DÉCISION (19)

EXEMPLES : ET, ENTRE ... (&&)

```
var temperature, message;

temperature = prompt('combien de degrés ?');

if (temperature >= 18 && temperature <= 28) {
    message = 'ok !';
} else {
    message = 'nok !';
}
```

EXEMPLES : NON (!)

la négation est assez lourde en JS
 (double parenthèses)
 utiliser des équivalents (dia suivante)

```
var sexe, message;

sexe = prompt('quel sexe (m/f) ?');

if (!(sexe === 'm' || sexe === 'M')) {
    message = 'féminin';
} else {
    message = 'masculin';
}
```

TRAITEMENT : AUTONOMIE - DÉCISION (20)

ATTENTION À LA NÉGATION

opérateur	sa négation	l'équivalent
$(x == y)$	$(\!(x == y))$	$(x != y)$
$(x != y)$	$(\!(x != y))$	$(x == y)$
$(x < y)$	$(\!(x < y))$	$(x >= y)$
$(x <= y)$	$(\!(x <= y))$	$(x > y)$
$(x > y)$	$(\!(x > y))$	$(x <= y)$
$(x >= y)$	$(\!(x >= y))$	$(x < y)$

négation d'expressions relationnelles
(opérateurs de comparaison)

opérateur	sa négation	l'équivalent
$(c1 c2)$	$(\!(c1 c2))$	$(\!c1 \&& \!c2)$
$(c1 \&& c2)$	$(\!(c1 \&& c2))$	$(\!c1 \!c2)$

négation d'expressions logiques
(opérateurs logiques ET OU)

lois dues à de Morgan

TRAITEMENT : AUTONOMIE - DÉCISION (21)

VARIABLES BOOLEENNES

POUR QUOI FAIRE ?

- ÉLÉGANCE
- LISIBILITÉ

COMMENT FAIRE ?

- AFFECTER LE RÉSULTAT D'UNE EXPRESSION BOOLEENNE À UNE VARIABLE

```

var nombre, estPair;
var msg = 'le nombre est ';
nombre = parseInt(prompt('un nombre entier svp : '));
estPair = ((nombre % 2) === 0);
if (estPair)
{ alert(msg + 'pair'); }
else
{ alert(msg + 'impair'); }

```

bien 'sentir' le caractère booléen de la variable (choix du nom)

ne pas écrire

`if(estPair === true)`



on ne dit pas "s'il fait beau **est vrai** ..." mais "s'il fait beau ..."

TRAITEMENT : AUTONOMIE - DÉCISION (22)

CONDITIONS "EN CASCADE" : SWITCH()

POUR QUOI FAIRE ?

- REEMPLACER UNE CASCADE DE `IF () {} ELSE IF () {} ...`
- ÉLÉGANCE, EXPRESSIVITÉ, LISIBILITÉ, COMPACITÉ ...
- OUTIL PLUS PUISSANT QU'IL N'Y PARAÎT ... (MULTIVALEUÉ, MULTICONDITIONNÉ)

ALTERNATIVE MULTIPLE : JS

```
switch (<<expression>>) {
    case <<valeur1>> :
        <<sequence1>>; break;
    case <<valeur2>> :
        <<sequence2>>; break;
    ...
    default : <<séquenceN>>;
}
```

si l'expression est évaluée à valeur1
on exécute la séquence1 et on sort

sinon, si l'expression est évaluée à valeur2
on exécute la séquence2 et on sort

si aucune des valeurs ne correspond
on exécute la séquenceN



la comparaison du switch utilise ==

TRAITEMENT : AUTONOMIE - DÉCISION (23)

CONDITIONS "EN CASCADE" (VARIABLES MULTIVALUÉES) : SWITCH()

UTILISATION TYPIQUE : UNE VARIABLE ET DES CONSTANTES LITTÉRALES

```
var jour, libelle;
jour = prompt('numéro de jour (1..7) ?');
jour = parseInt(jour);
switch (jour) {
    case 1: libelle = 'lundi';
               break;
    case 2: libelle = 'mardi';
               break;
    case 3: libelle = 'mercredi';
               break;
    // etc ...
    case 7: libelle = 'dimanche';
               break;
    default: libelle = 'erreur';
}
```

EN ALGORITHMIQUE ...

SELONQUE «VARIABLE» VAUT

«VALEUR1» : «SÉQUENCE1»
 «VALEUR2» : «SÉQUENCE2»
 «VALEUR3» : «SÉQUENCE3»
 # ETC ...

SINON

«SÉQUENCEN»

FINSI

👉 la comparaison du switch utilise ===

TRAITEMENT : AUTONOMIE - DÉCISION (24)

CONDITIONS "EN CASCADE" (VARIABLES MULTIVALUÉES) : SWITCH()

LE RÔLE DU BREAK : UN TRAITEMENT IDENTIQUE POUR DIFFÉRENTES VALEURS

```
var jour, libelle;
jour = prompt('numéro de jour');
jour = parseInt(jour);
switch (jour) {
    case 1 :
    case 2 :
    case 3 :
    case 4 :
    case 5 : libelle = 'jour de semaine';
               break;
    case 6 :
    case 7 : libelle = 'week end';
               break;
    default : libelle = 'erreur';
}
```

entre 1 et 5 ⇔ une série de OU
`if (x==1 || x==2 || x==3 || x==4 || x==5)`

EN ALGORITHMIQUE ...

SELONQUE «VARIABLE» ENTRE
 «VAL1» ET «VAL2» : «SÉQUENCE1»
 «VAL3» ET «VAL4» : «SÉQUENCE2»
 # ETC ...
 SINON
 «SÉQUENCEN»
 FINSI

TRAITEMENT : AUTONOMIE - DÉCISION (25)

CONDITIONS "EN CASCADE" : SWITCH()

PENSER AUTREMENT ?

- VALEUR LITTÉRALE , VARIABLE, EXPRESSION, CONDITION
ONT UNE NATURE COMMUNE : VALEUR
- ➔ SWITCH() MULTICONDITIONNÉ

ALTERNATIVE MULTIPLE : JS

```
switch (<>expression>) {  
    case <>expression1> :  
        <>sequence1>; break;  
    case <>expression2> :  
        <>sequence2>; break;  
    ...  
    default : <>séquenceN>;  
}
```

et si on mettait ici
la constante true ...

... et ici une condition ???

EN ALGORITHMIQUE ...

SELON **QUE**

«CONDITION1» : «SÉQUENCE1»

«CONDITION2» : «SÉQUENCE2»

ETC ...

SINON

«SÉQUENCEN»

FINSI

TRAITEMENT : AUTONOMIE - DÉCISION (26)

CONDITIONS "EN CASCADE" : SWITCH() MULTICONDITIONNÉ

```

var jour, libelle;
jour = +prompt('numéro de jour (1..7) ?');
switch (true) {
    case (jour >= 1 && jour <= 5) :    libelle = 'jour de semaine';
                                            break;
    case (jour === 6 || jour === 7) : libelle = 'week end';
                                            break;
    default :                           libelle = 'erreur';
}
  
```

```

var cote, message;
cote = +prompt('quelle cote (0..20) ?');
switch (true) {
    case (cote > 20) : message = "erreur de cote !";
                         break;
    case (cote >= 14) : message = "bien !";
                         break;
    case (cote >= 10) : message = "pas trop mal";
                         break;
    case (cote >= 8)  : message = "encore un effort";
                         break;
    case (cote >= 0)  : message = "faut s'y mettre !";
                         break;
    default           : message = "erreur de cote";
}
  
```



ON RÉSUME ?

➤ TYPES PRIMITIFS

- "number"
- "string"
- "undefined" 1 valeur : undefined
- "boolean" 2 valeurs : true, false

➤ OPÉRATEURS & FONCTIONS

- "number" +, -, *, /, %, ()
- "string" +
- affectation =, +=, -=, *=, /=, %=, ++, --
- comparaison <, <=, >, >=,
 ==, !=, ===, !==
- "boolean" &&, ||, !
 isFinite(), isNaN(), confirm()

CONDITIONNELLE

```
if («condition») {  
  «séquence»  
}
```

ALTERNATIVE

```
if («condition») {  
  «séquence1»  
} else {  
  «séquence2»  
}
```