

Université Polytechnique Hauts-de-France

INSA Hauts-de-France

Année 2022 - 2023

**M1 CDSI - O7CYASBD**  
**SECURITE DES BASES DE DONNEES**

**TP n°1 :**  
**Prise en main de Remix et Solidity**



Marwane AYaida (marwane.ayaida@uphf.fr)

# 1 Objectifs du TP

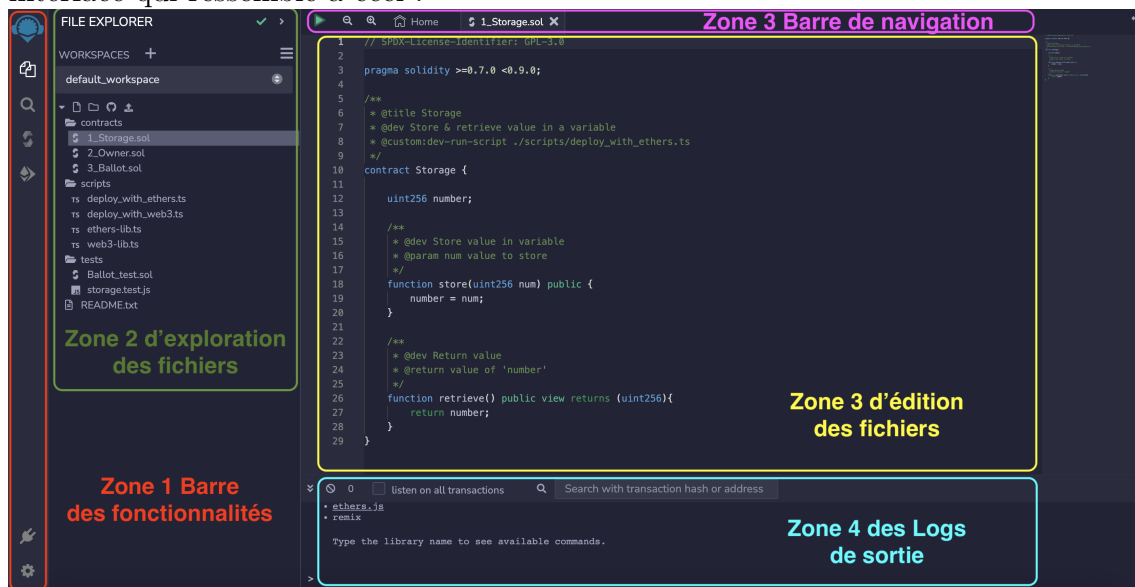
L'objectif de ce TP est de prendre en main l'outil de codage, déploiement et test des Smart Contracts dans Ethereum, à savoir **Remix**. C'est un outil gratuit accessible en ligne sans aucune utilisation, ce qui rend son utilisation assez aisée. Par la suite, nous nous intéresserons de plus près au langage de référence pour le développement de Smart Contract dans Ethereum, à savoir le langage **Solidity**. Nous allons voir des exemples d'utilisation des données de base de ce langage ainsi que des données plus complexes qui sont nécessaires à la création de Smart Contracts utiles. Nous verrons aussi comment le temps est utilisé pour la validation des transactions avec **Solidity**.

## 2 Prise en main de l'IDE Remix

Dans cette partie du TP, nous allons explorer les différentes fonctionnalités de l'IDE **Remix** qui permet de développer et de tester les Smart Contracts pour la Blockchain Ethereum.

► **2.1** Connectez-vous sur le site de l'IDE **Remix** : <https://remix.ethereum.org>.

Sur cette interface, vous pourrez tester l'utilisation de cet IDE. Vous arrivez sur une interface qui ressemble à ceci :



Les différentes zones sont décrites ci-dessous :

- Zone 1 : La barre des fonctionnalités permet de changer les différentes vues selon la fonctionnalité sélectionnée, ainsi :



permet d'explorer les fichiers du workspace courant.





permet de rechercher des chaînes de caractères dans les fichiers.



permet de compiler et de voir les options de compilation.



permet de tester le déploiement des Smart Contracts.

- Zone 2 : représente la zone de détail la fonctionnalité sélectionnée ci-dessous, ici on voit l'exemple d'exploration des fichiers.
- Zone 3 : représente la zone de navigation entre les fichiers ouverts, contient aussi l'icône "Home"  et un raccourci pour la compilation .
- Zone 4 : représente la zone de sortie des Logs du système.



► **2.2** Ouvrez l'exemple donné dans l'IDE **Remix** qui s'appelle "1\_Storage.sol" :

```

1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.7.0 <0.9.0;
4
5 /**
6  * @title Storage
7  * @dev Store & retrieve value in a variable
8  * @custom:dev-run-script ./scripts/deploy_with_ethers.ts
9  */
10 contract Storage {
11
12     uint256 number;
13
14     /**
15      * @dev Store value in variable
16      * @param num value to store
17      */
18     function store(uint256 num) public {
19         number = num;
20     }
21
22     /**
23      * @dev Return value
24      * @return value of 'number'
25      */
26     function retrieve() public view returns (uint256){
27         return number;
28     }
29 }

```

► **2.3** Regardez et déduisez la structure d'un Smart Contract écrit en **Solidity**.

► **2.4** Compilez le programme en utilisant le bouton "Compile"  et regardez ce que cela donne dans le menu "Solidity Compiler" .

► **2.5** Maintenant, déployons l'exemple sur une Blockchain Ethereum. Cliquez sur


le menu "*Deploy and run transactions*" . Cliquez ensuite sur le bouton "*Deploy*". Cela vous génère un contrat déployé en dessous du sous-menu "*Deployed Contracts*".

► **2.6** Dépilez le Smart Contract et testez les implémentations des fonctions "*retrieve*" et "*store*".

► **2.7** Modifiez ce programme afin d'ajouter deux fonctions "*increment*" et "*decrement*" qui permettent respectivement d'incrémenter et de décrémenter la valeur de la variable "*number*" d'un entier  $n$ .

► **2.8** Compilez, déployez et testez cette nouvelle version.

► **2.9** Maintenant, nous souhaitons déployer le même Smart Contract sur d'autres noeuds mineurs. Pour cela, copiez l'adresse unique du Smart Contract et ensuite collez la dans le champ "*Load contract from Address*" et enfin cliquez sur le bouton "*At Address*". Enfin, testez la modification de l'entier sur un des Smart Contracts et vérifiez que cela s'applique bien sur l'autre aussi.

► **2.10** Si vous voulez voir les détails des fichiers générés par la compilation, cliquez sur le menu "*Solidity Compiler*"  et puis le bouton "*Compilation Details*". Expliquez les différents champs et fichiers générés.

### 3 Prise en main du langage Solidity

Dans cette partie du TP, nous allons explorer le langage **Solidity** qui est utilisé comme langage de référence pour le développement de Smart Contracts à travers des exemples simples.

Nous souhaitons développer un Smart Contract pour gérer des enchères. Dont voici les détails :

- Nom du script : *Enchere.sol*
- Attributs :
  - *nom* : chaîne de caractères représentant le nom de l'enchérisseur, attribut publique, valeur par défaut "*UPHF*".
  - *montantEnchere* : entier représentant le montant de l'enchère actuelle, attribut publique, valeur par défaut 20000.
  - *eligible* : booléen qui indique si l'enchère est possible ou non, attribut publique.
  - *minEnchere* : entier constant représentant le montant minimum d'une enchère pour son acceptation, valeur par défaut 1000.
- Fonctions :
  - *setNom* : prend une chaîne de caractères en entrée et le répercute sur le nom de l'enchérisseur.

- *setMontantEnchere* : prend un entier en entrée et le répercute sur le montant de l'enchère.
- *determinerEligibilite* : teste si l'enchère actuelle est acceptable (supérieure à l'enchère minimale) ou non et le répercute sur le booléen "*eligible*".

► **3.1** Codez ce Smart Contract sur l'IDE ***Remix***.

► **3.2** Compilez et déployez ce Smart Contract. Puis, testez cette séquence :

1. Consulter les valeurs des attributs "*nom*", "*montantEnchere*" et "*eligible*".
2. Changez la valeur de l'attribut "*nom*" avec votre nom à vous et testez que cela a bien changé.
3. Changez la valeur de l'attribut "*montantEnchere*" à 300 et déterminer l'éligibilité de la transaction en appelant la fonction "*determinerEligibilite*".
4. Testez la valeur de l'attribut "*eligible*", est-ce logique ?
5. Changez la valeur de l'attribut "*montantEnchere*" à 5000 et déterminer l'éligibilité de la transaction en appelant la fonction "*determinerEligibilite*".
6. Testez la valeur de l'attribut "*eligible*", est-ce logique ?
7. Refaites d'autres tests pour valider votre Smart Contract.

## 4 Les types de données spécifiques dans Solidity

Dans cette partie du TP, nous allons voir des exemples de types de données spécifiques tels que "*address*", "*mapping*" et "*msg*", ainsi que leurs utilisations.

► **4.1** Chargez l'exemple de script ***Solidity*** "*Coin.sol*" (disponible sur Moodle) dans l'IDE ***Remix*** :

```

1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.7.0 <0.9.0;
4
5 /**
6  * @title Coin
7  * @dev Mint and send Coins
8  * @custom:dev-run-script ./scripts/deploy_with_ethers.ts
9  */
10 contract Coin {
11
12     // The keyword "public" makes those variables
13     // readable from outside.
14     address public minter;
15     mapping (address => uint) public balances;
```

```

16
17 // Events allow light clients to react on
18 // changes efficiently.
19 event Sent(address from, address to, uint amount);
20
21 // This is the constructor whose code is
22 // run only when the contract is created.
23 constructor() {
24     minter = msg.sender;
25 }
26
27 /**
28  * @dev Mint Coins for the Minter
29  * @param receiver address of the Minter
30  * @param amount the amount to be mint
31  */
32 function mint(address receiver, uint amount) public {
33     if (msg.sender != minter) return;
34     balances[receiver] += amount;
35 }
36
37 /**
38  * @dev Send Coins between Addresses
39  * @param receiver address of the receiver
40  * @param amount the amount to be sent to the receiver
41  */
42 function send(address receiver, uint amount) public {
43     if (balances[msg.sender] < amount) return;
44     balances[msg.sender] -= amount;
45     balances[receiver] += amount;
46     emit Sent(msg.sender, receiver, amount);
47 }
48 }

```

Ce script permet de générer de la cryptomonnaie grâce à la fonction "*mint*" afin d'alimenter la balance du compte "créateur de monnaie" et d'échanger de la cryptomonnaie avec la méthode "*send*" entre comptes si la balance du compte émetteur le permet.

Pour une meilleure compréhension du langage ***Solidity***, voici quelques informations de plus sur ce script :

- La variable "*address*" : c'est un type de donnée composite spécial défini dans ***Solidity*** d'une taille de 20 Octets qui permet de référencer un Smart Contract. La structure des données d'adresse contient également le solde du

compte en Wei. `"address.balance (uint256)"` Elle permet également le transfert de valeur vers une adresse spécifique `"address.transfer (amount uint256)"`.

- Le `"mapping"` : c'est une structure de données très polyvalente qui ressemble à une paire de clés/valeurs, elle peut également être considérée comme une table de hachage. La clé est généralement un hachage sécurisé d'un type de données **Solidity** simple tel que l'adresse et la valeur dans la paire clé-valeur qui peut être n'importe quel type arbitraire. Ici, nous illustrons l'idée du concept de `"mapping"` avec deux exemples. Nous pouvons utiliser le numéro de téléphone pour mapper des noms : `"mapping (uint => string) phoneToName;"`. C'est un exemple d'une fonction courante sur notre téléphone. Comme deuxième exemple, vous pouvez avoir une structure de toutes les données du client et utiliser le `"mapping"` pour mapper une adresse de compte aux données client, comme indiqué ici :

```
"struct customer {uint idNum; string name; uint bidAmount;}" et  
"mapping (address => customer) custData;"
```

- L'objet `"msg"` : c'est un type de données complexe spécifique au Smart Contract. Il représente l'appel qui peut être utilisé pour invoquer une fonction d'un Smart Contract. Il supporte de nombreux attributs dont deux qui nous intéressent ici : `"msg.sender"` représentant l'adresse de l'expéditeur et `"msg.value"` qui contient la valeur en Wei envoyée par l'expéditeur. Vous pouvez toujours rechercher d'autres détails pour l'utilisation de l'objet `"msg"` dans la documentation **Solidity**.

► **4.2** Testez la création de cryptomonnaie et de son transfert depuis le compte "créateur de monnaie" jusqu'à un autre compte. Attention, ici on parle bien de compte extérieurs (EOA) comme vu en cours et non pas les comptes internes utilisés pour identifier les Smart Contracts.

► **4.3** Maintenant, testez le transfert entre deux comptes autres que celui du "créateur de monnaie".

## 5 Les structures de données dans Solidity

Dans cette partie du TP, nous allons voir des exemples d'utilisation des structures de données, l'énumérateur `"enum"` et comment la notion du temps est gérée dans le langage **Solidity**.

► **5.1** Chargez l'exemple de script **Solidity** `"Ballot.sol"` (disponible sur Moodle) dans l'IDE **Remix** :

```
1 // SPDX-License-Identifier: GPL-3.0
```

```

2
3 pragma solidity >=0.7.0 <0.9.0;
4
5 /**
6  * @title Ballot
7  * @dev Ballot for voting
8  * @custom:dev-run-script ./scripts/deploy_with_ethers.ts
9  */
10 contract Ballot {
11
12     struct Voter {
13         uint weight;
14         bool voted;
15         uint8 vote;
16     }
17     struct Proposal {
18         uint voteCount;
19     }
20
21     address chairperson;
22     mapping(address => Voter) voters;
23     Proposal[] proposals;
24
25
26     /// Create a new ballot with $(_numProposals) different
27     proposals.
28     /**
29     * @dev Constructor of the Ballot
30     * @param _numProposals number of the different proposals
31     */
32     constructor(uint8 _numProposals) {
33         chairperson = msg.sender;
34         voters[chairperson].weight = 2;
35
36         // For each of the provided proposal,
37         // create a new proposal object and add it
38         // to the end of the array.
39         for (uint i = 0; i < _numProposals; i++) {
40             // 'Proposal({...})' creates a temporary
41             // Proposal object and 'proposals.push(...)'
42             // appends it to the end of 'proposals'.
43             proposals.push(Proposal({
44                 voteCount: 0
45             }));
46         }
47     }
48 }

```



```

46     }
47
48     /// Give $(toVoter) the right to vote on this ballot.
49     /// May only be called by $(chairperson).
50     /**
51     * @dev Register the voter by the chairperson
52     * @param toVoter address of the voter to be registered
53     */
54     function register(address toVoter) public {
55         if (msg.sender != chairperson || voters[toVoter].voted)
56             return;
57         voters[toVoter].weight = 1;
58         voters[toVoter].voted = false;
59     }
60
61     /// Give a single vote to proposal $(toProposal).
62     /**
63     * @dev Vote one time for each voter
64     * @param toProposal the number of the proposal to be voted
65     */
66     function vote(uint8 toProposal) public {
67         Voter storage sender = voters[msg.sender];
68         if (sender.voted || toProposal >= proposals.length) return;
69         sender.voted = true;
70         sender.vote = toProposal;
71         proposals[toProposal].voteCount += sender.weight;
72     }
73
74     /**
75     * @dev Compute the proposal which wins the vote
76     * @return _winningProposal the winning vote!
77     */
78     function winningProposal() public view returns (uint8
79         _winningProposal) {
80         uint256 winningVoteCount = 0;
81         for (uint8 prop = 0; prop < proposals.length; prop++)
82             if (proposals[prop].voteCount > winningVoteCount) {
83                 winningVoteCount = proposals[prop].voteCount;
84                 _winningProposal = prop;
85             }
86     }
87 }

```

Ce script permet de mettre en oeuvre un Smart Contract qui gère un vote dont les règles sont :

1. La créateur du vote est le responsable et de ce fait il obtient ainsi un poids de deux points pour son vote.
2. Les autres obtiennent une pondération d'un point uniquement pour leur vote.
3. Chaque électeur doit d'abord être inscrit par le responsable du vote avant de pouvoir voter.
4. Tout le monde ne peut voter qu'une seule fois.

► **5.2** Observez l'utilisation des structures *"Voter"* et *"Proposal"*. On fait appel à un élément de la structure en utilisant l'opérateur ".", par exemple *"Voter.weight"*.

► **5.3** Déployez ce Smart Contract en affectant, par exemple, à la variable *"\_\_numProposals"* la valeur 3.

► **5.4** Testez les résultats des votes avec plusieurs configurations en intégrant d'autres contrats votants en plus du responsable du vote.

Regardons maintenant un autre exemple à base de temps et qui utilise le type *"enum"*.

► **5.5** Chargez l'exemple de script ***Solidity*** *"StateTrans.sol"* (disponible sur Moodle) dans l'IDE ***Remix*** :

```

1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.7.0 <0.9.0;
4
5 /**
6  * @title StateTrans
7  * @dev State transaction based on time
8  * @custom:dev-run-script ./scripts/deploy_with_ethers.ts
9  */
10 contract StateTrans {
11
12     enum Stage {Init, Reg, Vote, Done}
13     Stage public stage;
14     uint startTime;
15     uint public timeNow;
16
17     /**
18      * @dev Constructor of the StateTrans
19      */
20     constructor() {
21         stage = Stage.Init;
22         startTime = block.timestamp;
23     }
24

```

```

25     //Assuming the Stage change has to be enacted APPROX every 10
        seconds
26     //timeNow variable is defined for understanding the process,
        you can simply use
27     // "block.timestamp" (the current time) Solidity defined
        variable
28     // Of course, time duration for the Stages may depend on your
        application
29     //10 seconds is set to illustrate the working
30     /**
31      * @dev Advances the system state if the time is elapsed
32      */
33     function advanceState () public {
34         timeNow = block.timestamp;
35         if (timeNow > (startTime + 10 seconds)) {
36             startTime = timeNow;
37             if (stage == Stage.Init) {stage = Stage.Reg; return;}
38             if (stage == Stage.Reg) {stage = Stage.Vote; return;}
39             if (stage == Stage.Vote) {stage = Stage.Done; return;}
40             return;
41         }
42     }
43 }

```

Commençons par discuter des unités de temps en **Solidity**. Dans une application Blockchain, tous les participants et les nœuds doivent se synchroniser sur un temps universel. À cette fin, la Blockchain Ethereum inclue un serveur de temps qui indique l'heure "Epoch Unix" ou l'heure depuis le 1er janvier 1970 en secondes. vous pourrez consulter ce lien pour un exemple de conversion de la date réelle du jour en heure "Unix Epoch". Cette heure est utilisée pour horodater l'heure du bloc. Lorsqu'un bloc est ajouté à la Blockchain, toutes les transactions confirmées par le bloc référencent le même temps que leur temps de confirmation. Une variable appelée "*block.timestamp*" définie par **Solidity**, renvoie l'horodatage de la création du bloc. Cette variable est souvent utilisée pour évaluer les conditions liées au temps. En d'autres termes, la variable "*block.timestamp*" dans une fonction n'est pas l'heure à laquelle la transaction de la fonction a été lancée, mais c'est l'heure à laquelle elle a été confirmée.

D'un autre côté, le type de données "*enum*" ou énumérateur, permet de lister des types de données définis par l'utilisateur avec un ensemble limité de valeurs significatives. Il est principalement utilisé pour un usage interne et il n'est pas pris en charge actuellement au niveau de l'ABI de **Solidity**. Cependant, il sert un objectif important de définition des états ou des phases d'un Smart Contract. Rappelez-vous

que les Smart Contracts représentent des projets de contrats qui peuvent passer par différentes états en fonction du temps ou des conditions d'entrée.

► **5.6** Déployez et testez le bon déroulement du contrat.

► **5.7** Copiez les script "*Ballot.sol*" en un nouveau script, intitulé "*BallotWithStages.sol*", qui permet d'utiliser la transition d'état implémentée dans le script donné comme exemple "*StateTrans.sol*" pour que le vote soit possible juste 10 secondes après son lancement sinon il n'est plus possible de voter pour les clients qui souhaitent le faire.

► **5.8** Déployez et testez le bon déroulement du contrat.

## 6 Les tests et validation dans Solidity

Dans cette partie du TP, nous allons voir comment se fait la validation et les tests des Smarts Contracts en utilisant les "*modifiers*" des fonctions, le "*revert*" (annulation) des transactions et les instructions "*require*" et "*assert*" ainsi que leurs utilisations. **Solidity** dispose d'une fonction "*revert*" qui entraîne une exception de retour d'état (retour à l'état initiale du contrat et des variables d'états). Cette gestion des exceptions annulera toutes les modifications apportées à l'état dans l'appel en cours et inverse la transaction et signale également une erreur à l'appelant.

Nous allons introduire un modificateur ou "*modifier*" de fonction avec l'étape requise pour le bon déroulement du vote de l'exemple précédent comme paramètre. Tout d'abord, nous allons définir un "*modifier*" de fonction pour l'accès selon les étapes du vote. Par la suite, nous allons ajouter le modificateur à l'en-tête des différentes fonctions. Ainsi, nous allons ajouter le modificateur avec le paramètre de l'étape requise. Le modificateur fait appel à l'instruction "*require*" qui permet de faire le test de l'étape actuelle. Si l'étape correspond, la fonction est exécutée et la transaction est validée. Sinon, rien n'est exécuté et la transaction est annulée. Voici le code du modificateur :

```
1 modifier validStage(Stage reqStage)
2 { require(stage == reqStage);
3     _;
4 }
```

► **6.1** Pour visualiser le code du Smart Contract intégrant le modificateur, chargez l'exemple de script **Solidity** "*BallotWithModifiers.sol*" (disponible sur Moodle) dans l'IDE **Remix** :

```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.7.0 <0.9.0;
4
```

```

5  /**
6   * @title BallotWithStages
7   * @dev Ballot for voting with stages and modifiers
8   * @custom:dev-run-script ./scripts/deploy_with_ethers.ts
9   */
10 contract Ballot {
11
12     struct Voter {
13         uint weight;
14         bool voted;
15         uint8 vote;
16         //address delegate;
17     }
18     struct Proposal {
19         uint voteCount;
20     }
21     enum Stage {Init,Reg, Vote, Done}
22     Stage public stage = Stage.Init;
23
24     address chairperson;
25     mapping(address => Voter) voters;
26     Proposal[] proposals;
27
28     event votingCompleted();
29
30     uint startTime;
31     //modifiers
32     modifier validStage(Stage reqStage)
33     { require(stage == reqStage);
34       _;
35     }
36
37
38     /// Create a new ballot with $(_numProposals) different
39     proposals.
40     /**
41     * @dev Constructor of the Ballot
42     * @param _numProposals number of the different proposals
43     */
44     constructor(uint8 _numProposals) {
45         chairperson = msg.sender;
46         voters[chairperson].weight = 2; // weight is 2 for testing
47         purposes
48         for (uint i = 0; i < _numProposals; i++) {
49             proposals.push(Proposal({

```

```

48         voteCount: 0
49     }));
50 }
51 stage = Stage.Reg;
52 startTime = block.timestamp;
53 }
54
55 /// Give $(toVoter) the right to vote on this ballot.
56 /// May only be called by $(chairperson).
57 /**
58  * @dev Register the voter by the chairperson
59  * @param toVoter address of the voter to be registered
60  */
61 function register(address toVoter) public validStage(Stage.Reg)
62 {
63     //if (stage != Stage.Reg) {return;}
64     if (msg.sender != chairperson || voters[toVoter].voted)
65         return;
66     voters[toVoter].weight = 1;
67     voters[toVoter].voted = false;
68     if (block.timestamp > (startTime+ 10 seconds)) {stage =
69         Stage.Vote; }
70 }
71
72 /// Give a single vote to proposal $(toProposal).
73 /**
74  * @dev Vote one time for each voter
75  * @param toProposal the number of the proposal to be voted
76  */
77 function vote(uint8 toProposal) public validStage(Stage.Vote)
78 {
79     // if (stage != Stage.Vote) {return;}
80     Voter storage sender = voters[msg.sender];
81     if (sender.voted || toProposal >= proposals.length) return;
82     sender.voted = true;
83     sender.vote = toProposal;
84     proposals[toProposal].voteCount += sender.weight;
85     if (block.timestamp > (startTime+ 10 seconds)) {stage =
86         Stage.Done; emit votingCompleted();}
87 }
88
89 /**
90  * @dev Compute the proposal which wins the vote
91  * @return _winningProposal the winning vote!
92  */

```

```

88     function winningProposal() public validStage(Stage.Done) view
      returns (uint8 _winningProposal) {
89         //if(stage != Stage.Done) {return;}
90         uint256 winningVoteCount = 0;
91         for (uint8 prop = 0; prop < proposals.length; prop++)
92             if (proposals[prop].voteCount > winningVoteCount) {
93                 winningVoteCount = proposals[prop].voteCount;
94                 _winningProposal = prop;
95             }
96         assert (winningVoteCount > 0);
97     }
98 }

```

► **6.2** Visualisez le code du modificateur "*validStage(Stage reqStage)*". Interprétez ce code et expliquez comment fonctionne-t-il ?

► **6.3** Expliquez la syntaxe utilisée pour intégrer les modificateurs dans les fonctions du vote. C'est quoi l'intérêt des modificateurs par rapport aux tests simples utilisés dans la version précédente ?

► **6.4** C'est quoi le type et l'utilité de l'objet "*votingCompleted*" et à quel moment est-il utilisé ?

► **6.5** Déployez et testez le bon déroulement du contrat.

► **6.6** À la fin du temps du vote et après l'affichage du résultat, essayez de voter. Que se passe-t-il cette fois-ci ? Regardez ce qui est affiché du côté du Log.

► **6.7** Provoquez d'autres situations du même type.

► **6.8** Quel est l'intérêt de cette ligne de commande ?

```

1  assert(winningVoteCount > 0);

```

► **6.9** Juste pour ce test, changez l'en-tête de la méthode "*winningProposal()*" pour l'activer avec l'étape "*Stage.Reg*" :

```

1  function winningProposal() public validStage(Stage.Reg) view
    returns (uint8 _winningProposal) {

```

► **6.10** Redéployez de nouveau le Smart Contract et testez le résultat du vote directement sans qu'il y ait aucun votant. Que remarquez-vous ?

► **6.11** Remettez la bonne en-tête de la méthode "*winningProposal()*" pour l'activer avec l'étape "*Stage.Done*" :

```

1  function winningProposal() public validStage(Stage.Done) view
    returns (uint8 _winningProposal) {

```

► **6.12** Refaites d'autres tests pour mieux comprendre les différents éléments que contient cet exemple.