algobase-training-course

Etape 4: Boucles, entrées et conditions

Question 1: saisie de l'utilisateur

Reprenons notre code précédent:

```
/// <reference path="fonctions.ts"/>
creerZoneDessin();
let nbrectangles:number=5;
let norectangle:number;
for(norectangle=0;norectangle<nbrectangles;norectangle++){
    dessinerRectangleRempli(100*norectangle,200,10,100,"blue");
}
```

Un des avantages d'avoir utilisé une boucle est que l'on peut (ici) afficher autant de rectangles que l'on veut, sans avoir à ré-écrire complètement notre code. Pour aller plus loin: nous allons demander au visiteur du site combien de rectangles il souhaite afficher. Nous utiliserons la fonction lireNombre. Celle-ci prend comme paramètre le message à afficher à l'utilisateur (une chaîne de caractères donc) et retourne le nombre saisi par l'utilisateur. Au lieu de

```
let nbrectangles:number=5;
```

indiquez

```
let nbrectangles:number;
nbrectangles=lireNombre("Combien de rectangles voulez-vous afficher ?");
```

Question 2: problème de place

En pratique, l'utilisateur peut rentrer le nombre qu'il veut. Il ne faut pas dessiner des rectangles en dehors de la zone de dessin. Nous allons donc:

1) calculer combien de rectangles rentrent dans la fenêtre (variable nbrectangles_max). Ce nombre vaut la largeur de la fenêtre (que l'on obtient avec la fonction largeurZone()) divisé par l'espace entre chaque rectangle (100).

2) nous ne dessinerons que les rectangles qui rentrent dans la fenêtre. Une solution consiste à ne pas executer l'instruction dessinerRectangleRempli si nbrectangles > nbrectangles_max. Cependant, la boucle est toujours parcourue (pour rien). Une alternative est donc de modifier le nombre de rectangles à afficher: soit c'est le

nombre rentré par l'utilisateur soit c'est le nombre maximum de rectangles. Nous allons donc modifier la valeur de nbrectangles avant de rentrer dans la boucle, à l'aide d'un test conditionnel (SI).

```
if(nbrectangles>nbrectangles_max)
{
    console.log("Valeur trop grande, nous ne tracerons que "+nbrectangles
    nbrectangles=nbrectangles_max;
}
```

i Le **+** correspond à la concaténation de chaînes de caractères

i A-t-on besoin de faire des tests si l'utilisateur rentre un nombre négatif ou nul **?** Non car nous ne rentrerons pas dans la boucle POUR **!**

Question 3: changement de couleur

Nous allons modifier le code de façon à ce que la couleur du rectangle soit choisie en fonction de la valeur saisie par l'utilisateur: quand le nombre est pair, les rectangles seront en bleu, en rouge dans l'autre cas. Définissez une variable de type string nommée couleur. Modifiez la ligne dessinerRectangleRempli en mettant couleur à la place de "blue". L'instruction pour assigner la couleur bleu à couleur est:

```
couleur="blue";
```

Cette instruction n'est pas toujours utilisée **=>** il faut utiliser un SI (test conditionnel). En Typescript: SI -> if

```
if(nbrectangles%2 == 0) {
    couleur="blue";
}
```

i **%** correspond au modulo, c'est à dire le reste de la division entière par le nombre qui suit. Les nombres pairs sont des multiples de **2** donc il ne reste rien de la division par **2**.

On n'a que deux situations: les pairs (bleu) et les impairs (rouge) donc l'instruction pour assigner la couleur rouge est utilisée dans le SINON

```
if(nbrectangles%2 == 0) {
    couleur="blue";
}
else {
    couleur="red";
}
```

Codez et testez.

Entrez la valeur 15. Est-ce que les rectangles sont en rouge ? Si ce n'est pas le cas: où avez-vous mis le SI ... SINON ? Si vous l'avez mis après le SI de la Question 2, l'erreur est là: vous choisissez la couleur après avoir modifié la valeur du nombre de rectangles à tracer (si cela dépassait de la fenêtre). Mettez le code entre la partie correspondant à la saisie par l'utilisateur et le SI de la question 2. Testez.

Question 4: changement de couleur (2)

Nous allons modifier le code: un seul rectangle demandé -> rose, deux rectangles -> vert, 3 rectangles -> rouge et, dans les autres cas, bleu. La couleur dépend toujours du nombre saisi par l'utilisateur mais il n'est plus question de pair ou impair. Comme auparavent, l'instruction couleur="pink" (couleur="green", couleur="red", ...) n'est pas toujours utilisé => SI. On a ici plusieurs cas possible => SI ... SINON SI ... SINON SI .... SINON

```
if(nbrectangles == 1) {
    couleur="pink";
}
else if(nbrectangles == 2) {
    couleur="green";
}
else if(nbrectangles == 3) {
    couleur="red";
}
else {
    couleur="blue";
}
```

Codez et testez.

Question 5: changement de couleur (3)

Nous changeons d'avis: la couleur des rectangles ne dépendra plus de ce qui est saisi par l'utilisateur mais de la position du rectangle: le 1er rectangle est rose, le deuxième vert, le troisième rouge et les autres bleus. La couleur va donc varier pour chaque rectangle: les tests condtionnels seront donc dans la boucle POUR et le test dépend de norectangle !

```
for(norectangle=0;norectangle<nbrectangles;norectangle++){
    if(norectangle == 0) {
        couleur="pink";
    }
    else if(norectangle == 1) {
        couleur="green";
```

```
    }
    else if(norectangle == 2) {
        couleur="red";
    }
    else {
        couleur="blue";
    }
    dessinerRectangleRempli(100*norectangle,200,10,100,couleur);
}
```

Remarque: la variable couleur est maintenant utilisée uniquement dans la boucle. On peut donc la définir dans la boucle

```
for(norectangle=0;norectangle<nbrectangles;norectangle++){
    let couleur:string;
    if(norectangle == 0) {
    // ...
    }
    dessinerRectangleRempli(100*norectangle,200,10,100,couleur);
}
```

Transcompilez et testez ! Ajoutez après la boucle l'instruction suivante:

```
console.log(couleur);
```

Essayez de transcompiler...

i La variable couleur a été définie à l'intérieur de la boucle. Cette variable n'est accessible que dans ce bloc d'instruction là (dans ce bloc et les sous-blocs) et donc pas en dehors du bloc ... comme par exemple pour notre console.log . C'est ce qu'on appelle la portée d'une variable. L'avantage est très simple: les variables définies dans un bloc d'instruction (notamment dans une fonction) ne sont utilisées que dans ce bloc, ce qui permet d'avoir des variables nommées similairement.

Question 6

Précédemment (Question 2), nous avons demandé à l'utilisateur de saisir un nombre puis nous affichions nbrectangles_max ou nbrectangles rectangles. Une alternative est de demander à l'utilisateur de saisir un nombre de rectangle "compatible" avec le nombre maximum. Modifiez votre code en indiquant à l'utilisateur combien de rectangles maximum il peut demander: 1) on calcule la valeur de nbrectangles_max d'abord 2) puis on demande à l'utilisateur:

```
let nbrectangles:number;
nbrectangles=lireNombre("Combien de rectangles voulez-vous afficher (infè
```

i Le **+** correspond à la concaténation de chaînes de caractères

Question 6 (suite)

Rien n'oblige l'utilisateur à réellement saisir un nombre infèrieur ou égal à nbrectangles_max. Pour l'obliger, il faudrait redemander à l'utilisateur de saisir le nombre s'il dépasse (ie. s'il est supérieur à nbrectangles_max). Puis lui redemander à nouveau si ce n'est toujours pas bon, etc...

On effectue donc plusieurs fois l'instruction "demander à l'utilisateur" et on ne sait pas à l'avance combien de fois => Boucle TANT QUE

- Instruction répétée:

  nbrectangles=lireNombre("Combien de rectangles voulez-vous afficher (infèrieur ou égal à "+nbrectangles_max+")?");

- Condition pour rester dans la boucle: le nombre saisi par l'utilisateur est plus grand que nbrectangles_max => (nbrectangles >=nbrectangles_max)

- Initialisation: il faut donner à nbrectangles une valeur qui permette de rentrer dans la boucle. On peut par exemple donner la valeur nbrectangles_max+1

- Pour éviter une boucle infinie, il faut mettre à jour la valeur de nbrectangles dans la boucle: c'est ce qui est fait avec nbrectangles=lireNombre...

Le code est donc:

```
nbrectangles=nbrectangles_max+1; // initialisation pour rentrer dans la b
while( nbrectangles >=nbrectangles_max )
{
    nbrectangles=lireNombre("Combien de rectangles voulez-vous afficher (
}
```

Lorsque la valeur de nbrectangles est valide, on sort de la boucle et on peut alors faire la suite du code (afficher les rectangles).

Question 6 (ter)

En TypeScript, les nombres sont des entiers (non naturels). L'utilisateur peut donc saisir un nombre infèrieur à 0. Nous considèrerons qu'il doit mettre 0 s'il n'en veut pas mais pas un nombre infèrieur. Il faut donc modifier la condition: on redemande à l'utilisateur tant que le nombre est soit plus petit que 0 soit plus grand que nbrectangles_max: => (nbrectangles<0) OU (nbrectangles >=nbrectangles_max)

En TypeScript l'opérateur logique entre booléens OU s'écrit ||

```
while( (nbrectangles<0) || (nbrectangles >=nbrectangles_max))
{
    nbrectangles=lireNombre("Combien de rectangles voulez-vous afficher (
}
```

Question 7

Nous souhaitons modifier le code précédent: l'utilisateur devra entrer l'espacement souhaité entre les rectangles (et non plus la valeur de 100). L'espacement devra être un nombre: plus grand que 0 et plus petit que 200. Comme cet espacement est utilisé pour calculer le nombre maximum de rectangles, cet espacement doit être spécifier avant. Comme précédemment, pour forcer l'utilisateur à saisir un nombre valide, il faudra répéter de demander à l'utilisateur si ce n'est pas bon. Le début du code est donc:

```
let espacement:number = -1;
```

- Ajoutez le code demandant à l'utilisateur de saisir l'espacement (dans une boucle TANT QUE).
- Modifiez le calcul du nombre maximum de rectangles
- Modifiez l'emplacement des rectangles dans la boucle POUR: espacement*norectangle au lieu de 100*norectangle

Nous allons rajouter une contrainte: l'espacement doit être un multiple de 10. Il faut donc continuer de demander si le modulo 10 du nombre saisi n'est pas nul. Ajoutez cela dans la condition du TANT QUE (il faut rajouter un nouveau OU) ... et spécifiez à l'utilisateur que ce doit être un multiple de 10.

Question 8: nettoyage du code

Il est important de faire un code durable. Rappel des règles de base: indentation, nommage explicite des variables, séparation des morceaux de code, commentaires

```
/// <reference path="fonctions.ts"/>
let espacement:number;
let nbrectangles_max:number;
let nbrectangles:number;
let norectangle:number;

creerZoneDessin();

// quel espacement
espacement=-1;// initialisation pour rentrer dans la boucle
while( (espacement<0) || (espacement >=200) || (espacement%10 !=0 ) )
{
    espacement=lireNombre("Combien voulez-vous d'espacement entre les rec
}

// combien de rectangles
nbrectangles_max=largeurZone()/espacement;
```

```
nbrectangles=nbrectangles_max+1; // initialisation pour rentrer dans la b
while( (nbrectangles<0) || (nbrectangles >=nbrectangles_max))
{
    nbrectangles=lireNombre("Combien de rectangles voulez-vous afficher (
}

// tracé des rectangles
for(norectangle=0;norectangle<nbrectangles;norectangle++){
    dessinerRectangleRempli(espacement*norectangle,200,10,100,"blue");
}
```