Exo 1 : Ecrire un programme qui additionne.

SOMMES

Entier A,B

Flottant C,D

fonction addition

Début

A <- 1

B <- 6

C <- 0.4

D <- A+B+C

écrire ('La somme de A+B+C est' , D)

Fin

Exo 2 : nous avons A = 10 et B=0.33

demander à l'utilisateur d'insérer une valeur {Lire A}

si la valeur données par l'utilisateur est 10 alors le programme répond 'c'est équivalent à A'

sinon il répond 'équivalent à B'

TEST Sans vérification

Entier A

Flottant B

fonction check

Début

A <- 10

B <- 0.33

Lire D

Si <D=10> Alors default

écrire ("c'est équivalent à A")

Sinon default

écrire ("c'est équivalent à B")

fin

TEST Avec Vérification

Entier A

Flottant B

fonction check

Début

A <- 10

B <- 0.33

Tant que <D!=A ou D!= B> (vérification)

début tant que

Lire D

fin tant que

Si <D=10> Alors default

écrire ("c'est équivalent à A")

Sinon default

écrire ("c'est équivalent à B")

Fin

Exo : Paire

Entier i

Fonction paire

Début

Pour (i { 0 : 1000 pas 2 ) a

lors écrire (i)

2 ème façon de faire

Pour (i { 0 : 1000) alors

Si i mod 2==0 alors écrire (i)

Exo : impair

Entier i

Fonction nombre impair

Début

Pour (i {1 ; 1000 pas 2 ) alors écrire (i)

NB PAIR

Entier i

fonction writePair

début

Pour i dans {0...1000}:default

Si <i mod 2 = 0> Alors default

Ecrire(i)

Fin

NB IMPAIR

Entier i

fonction writeImpair

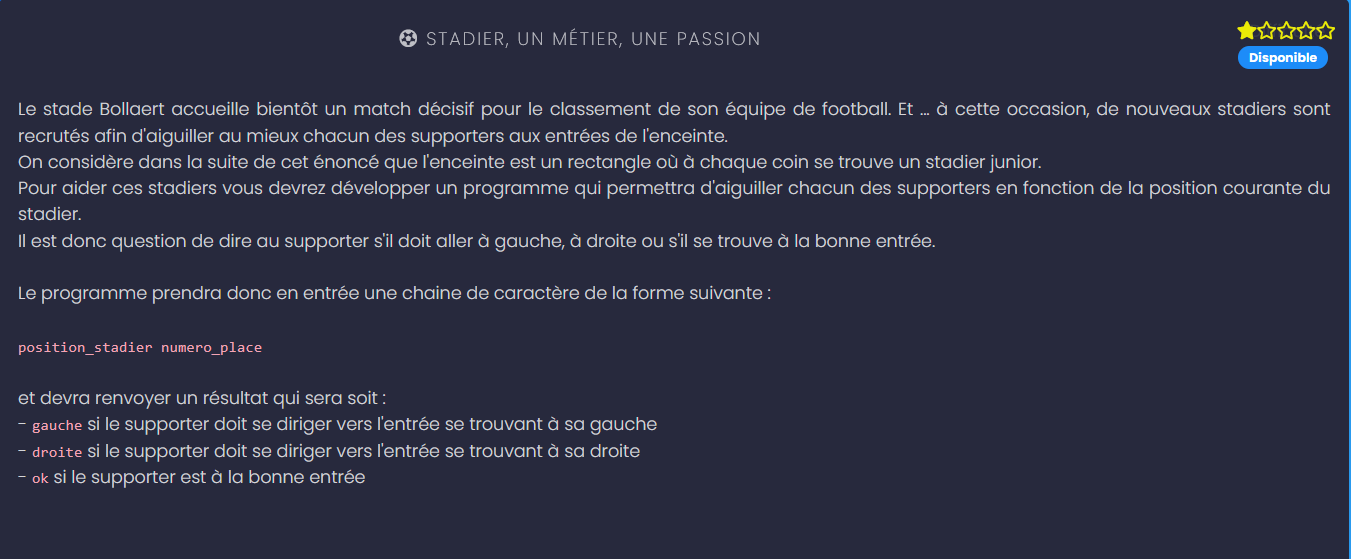
début

Pour i dans {0...1000}:default

Si <i mod 2 != 0> Alors default

Ecrire(i)

Fin



Pseudo code JS

NB Stadier const stadierRequest = () => {

Entier numberStadier return parseInt(prompt(stadierRequestMessage));} ;

fonction demandeStadier

début

écrire("Votre numéro Stadier?")

lire numberStadier

fin

NB Supporter const supporterRequest = () => {

Entier numberSupporter return parseInt(prompt(supporterRequestMessage));} ;

fonction demandeSupporter

début

ecrire("Numéro de place?")

lire numberSupporter

fin

Position\_stadier

Num\_place

If num\_place >= ( position\_stadier x10 )-9 et num \_place < position\_stadier\*10) {alert («   ok » ) }

If ( num\_place > 30 et position\_stadier ===1 ) { alert (« Gauche ») ; }

If ( num\_place < 11 et position\_stadier ===2 ) {alert (« Gauche ») ; }

If ( num\_place < 21 et position\_stadier ===3 ) {alert (« Gauche ») ; }

If ( num\_place < 31 et position\_stadier ===4 ) {alert (« Gauche ») ; }

If ( Num\_place ) Position\_stadier \*10 || ( num\_place < position\_stadier \*10-9 et num-place <position\_stadier\*(0)){ alert (« droite » ) ; }

Switch (true) { case Num\_place > (position\_stadier\*10) -9 et num\_place < position\_stadier\*10 ) {alert (« ok ») ;

Break

Défault :

Alert (« error ») ;

Break

Vérifications de la position des supporters

Const checkpositionsupporter =() => {

Let position\_stadier = stadierRequest () ;

While (isNaN (position\_stadier ) ) {

Position\_stadier = stadierRequest () ; }

While ( position\_stadier < 1 || position\_stadier > 4 ) {

Position\_stadier = stadierRequest () ; }

Let Num\_place = supporterRequest () ;

While (IsNan (num\_place ) ) {

Num\_place = supporterRequest () ; }

While ( Num\_place < 1 || Num\_place > 40 ) {

Num\_place = supporterRequest () ; }

IndicatePosition ( position\_stadier, Num\_place ) ; }

Exo 1 : En pseudo Code, faite un algo qui demande à l’utilisateur une valeur entre 1 et 3, redemandez si il ne respecte pas l’intervalle

Intervalle

ENTIER NB

Début

Ecrire ‘ Veuillez entrez un nombre entre 1 et 3 ‘

Lire NB

tant que nb >3 || nb < 1 alors défaut

Ecrire ‘veuillez entrez un nombre entre 1 et 3 ‘

Lire NB

Fin

Correction Alex

checkIntervalle

Entier N

fonction check

Debut

N ← 0

Ecrire "Entrez un nombre entre 1 et 3"

TantQue N < 1 ou N > 3

Lire N

Si N < 1 ou N > 3 Alors

Ecrire "Saisie erronée. Recommencez”

FinSi

FinTantQue

Fin

Exo 2 : Ecrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu’à ce que la

réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : « Plus

petit ! », et inversement, « Plus grand ! » si le nombre est inférieur à 10.

Intervalle 2

NB, coup Entier

Début

Ecrire « devine le nombre, tu as 5 chances »

Pour coup { 0….5 ] : défault

Si coup > 5 alors

Ecrire « Perdu »

Si i =< 5

Ecrire « Félicitation »

Lire NB

Si nb < 10 alors défault

Ecrire trop petit !

Si nb > 20 alors défault

Ecrire trop grand

FinSi

Fin

Correction Exo 2 (Alex)

Variable N en Entier

Debut

N ← 0

Ecrire "Entrez un nombre entre 10 et 20"

TantQue N < 10 ou N > 20

Lire N

Si N < 10 Alors

Ecrire "Plus grand !"

SinonSi N > 20 Alors

Ecrire "Plus petit !"

FinSi

FinTantQue

Fin

Exo 3 : Ecrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

1er possibilité

Variables N, i en Entier

Debut

Ecrire "Entrez un nombre : "

Lire N

Stop ← N+10

Ecrire "Les 10 nombres suivants sont : "

TantQue N < Stop

N ← N+1

Ecrire N

FinTantQue

Fin

2eme possibilité

Variables N, i en Entier

Debut

Ecrire "Entrez un nombre : "

Lire N

i ← 0

Ecrire "Les 10 nombres suivants sont : "

TantQue i < 10

i ← i + 1

Ecrire N + i

FinTantQue

Fin

3 eme possibilité

Variables N en Entier

Debut

Ecrire "Entrez un nombre : "

Lire N

Ecrire "Les 10 nombres suivants sont : "

Pour i dans {1...10}: default

Ecrire N + i

FinPour

Fin

4) Ecrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre.

Variables N, i en Entier

Debut

Ecrire "Entrez un nombre : "

Lire N

Ecrire "La table de multiplication de ce nombre est : "

Pour i dans {1...10}: default

Ecrire (N, " x ", i, " = ", n\*i)

i Suivant

Fin

5) Ecrire un algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu’à ce nombre. Par exemple, si l’on entre 5, le programme doit calculer (1 + 2 + 3 + 4 + 5 = 15)

ps: on souhaite afficher uniquement le résultat, pas la décomposition du calcul

Variables N, i, Som en Entier

Debut

Ecrire "Entrez un nombre : "

Lire N

Som ← 0

Pour i dans {1...N}: default

Som ← Som + i

Fin Pour

Ecrire "La somme est : ", Som

Fin

6) Ecrire un algorithme qui demande un nombre de départ, et qui calcule sa factorielle.

NB : la factorielle de 8, notée 8 !, vaut

1 x 2 x 3 x 4 x 5 x 6 x 7 x 8

Variables N, i, F en Entier

Debut

Ecrire "Entrez un nombre : "

Lire N

F ← 1

Pour i dans {2 ... N}

F ← F \* i

Fin pour

Ecrire "La factorielle est : ", F

Fin

7)

Exo8) Réécrire l’algorithme précédent, mais cette fois-ci on ne connaît pas d’avance combien l’utilisateur souhaite saisir de nombres. La saisie des nombres s’arrête lorsque l’utilisateur entre un zéro.

Variables N, i, PG, IPG en Entier

Debut

N ← 1

i ← 0

PG ← 0

TantQue N <> 0

Ecrire "Entrez un nombre : "

Lire N

i ← i + 1

Si i = 1 ou N > PG Alors

PG ← N

IPG ← i

FinSi

FinTantQue

Ecrire "Le nombre le plus grand était : ", PG

Ecrire "Il a été saisi en position numéro ", IPG

Fin

9) Écrire un algorithme qui permette de connaître ses chances de gagner au tiercé, quarté, quinté et autres impôts volontaires.

On demande à l’utilisateur le nombre de chevaux partants, et le nombre de chevaux joués. Les deux messages affichés devront être :

Dans l’ordre : une chance sur X de gagner

Dans le désordre : une chance sur Y de gagner

X et Y nous sont donnés par la formule suivante, si n est le nombre de chevaux partants et p le nombre de chevaux joués (on rappelle que le signe ! signifie "factorielle", comme dans l'exercice 6)

X = n ! / (n - p) !

Y = n ! / (p ! \* (n – p) !)

NB : cet algorithme peut être écrit d’une manière simple, mais relativement peu performante. Ses performances peuvent être singulièrement augmentées par une petite astuce. Vous commencerez par écrire la manière la plus simple, puis vous identifierez le problème, et écrirez une deuxième version permettant de le résoudre.

Variables N, P, i, Numé, Déno1, Déno2 en Entier

Debut Ecrire "Entrez le nombre de chevaux partants : "

Lire N

Ecrire "Entrez le nombre de chevaux joués : "

Lire P

Numé ← 1

Pour i dans {2...N}: default

Numé ← Numé \* i

i Suivant

Déno1 ← 1

Pour i dans {2...N - P}: default

Déno1 ← Déno1 \* i

i Suivant

Déno2 ← 1

Pour i dans {2...p}: default

Déno2 ← Déno2 \* i

i Suivant

Ecrire "Dans l’ordre, une chance sur ", Numé / Déno1

Ecrire "Dans le désordre, une sur ", Numé / (Déno1 \* Déno2)

Fin

Cette version, formellement juste, comporte tout de même deux faiblesses.

La première, et la plus grave, concerne la manière dont elle calcule le résultat final. Celui-ci est le quotient d'un nombre par un autre ; or, ces nombres auront rapidement tendance à être très grands. En calculant, comme on le fait ici, d'abord le numérateur, puis ensuite le dénominateur, on prend le risque de demander à la machine de stocker des nombres trop grands pour qu'elle soit capable de les coder. C'est d'autant plus bête que rien ne nous oblige à procéder ainsi : on n'est pas obligé de passer par la division de deux très grands nombres pour obtenir le résultat voulu.

La deuxième remarque est qu'on a programmé ici trois boucles successives. Or, en y regardant bien, on peut voir qu'après simplification de la formule, ces trois boucles comportent le même nombre de tours ! (si vous ne me croyez pas, écrivez un exemple de calcul et biffez les nombres identiques au numérateur et au dénominateur). Ce triple calcul (ces trois boucles) peut donc être ramené(es) à un(e) seul(e). Et voilà le travail, qui est non seulement bien plus court, mais aussi plus performant :

Variables N, P, i, A, B en Numérique

Debut

Ecrire "Entrez le nombre de chevaux partants : "

Lire N

Ecrire "Entrez le nombre de chevaux joués : "

Lire P

A ← 1

B ← 1

Pour i dans {1...P}: default

A ← A \* (i + N - P)

B ← B \* i

i Suivant

Ecrire "Dans l’ordre, une chance sur ", A

Ecrire "Dans le désordre, une chance sur ", A / B (FIN)