2de TD n°3 : utilisation de la calculatrice en mode « Programme »

Algorithmes et programmes : Un *algorithme* est un ensemble d'instructions structuré de manière à atteindre un but. Ces instructions manipulent des données (nombres, textes, etc.) : entrées, stockages en mémoire, calculs, affichages. Comme un algorithme est souvent exécuté par une machine (calculatrice, ordinateur, robot), il doit être traduit dans un langage que cette machine comprend : les *programmes* pour calculatrices *Casio* et *TI* utilisent une syntaxe dérivée du langage « Basic » (le Basic Casio est différent du Basic TI) tandis que les programmes de la calculatrice *Numworks* sont en « Python » (un autre langage de programmation, utilisé plutôt sur ordinateur).

Le mode « programmation » de la calculatrice :

Casio : On y entre avec la touche PRGM. Pour commencer un nouveau programme, sélectionner NEW (bouton F3), taper ensuite un nom pour le programme (ALGO1 par exemple) et EXE. Il ne reste plus qu'à écrire le programme. Pour modifier un programme existant, choisir EDIT dans le menu PRGM ; pour l'exécution, choisir EXE.

TI : On y entre avec la touche « prgm ». Pour commencer un nouveau programme, sélectionner NOUV, taper ensuite un nom pour le programme (ALGO1 par exemple), valider avec Entrer. Il ne reste plus qu'à écrire le programme. Pour modifier un programme existant, choisir EDIT dans le menu prgm ; pour l'exécution, taper Entrer deux fois.

Numworks : Faire la mise à jour (module Python indisponible sans cela). L'accès est simple depuis le menu « Accueil ». Il est précisé que c'est une version bêta (test) de Python qui est limitée et va bientôt être améliorée. On commence par éditer le programme (un seul à la fois, pas besoin donc de le nommer)

1) Entrées, affectations, sorties

	Algorithme	Basic Casio	Basic TI	Python Numworks
Affectation : instruction qui place une valeur en mémoire	A=A+1	A+1→A	A+1 $^{STO}\rightarrow$ A	a=a+1
Entrée (lecture) : affectation qui utilise la saisie de l'utilisateur	Saisir A	?→A	Input A	a=float(input())
Initialisation : première affectation pour une variable	A=0	0→A	0 $^{STO}\rightarrow$ A	a=0
Sortie (écriture) : instruction qui affiche une donnée.	Afficher A	A◢	Disp A	print(a)

Remarques : L'instruction A=A+1 signifie que l'on augmente de 1 la valeur de A. On note parfois cela A:=A+1 ou bien A←A+1 ou encore A+1→A. Nous avons choisi la notation la plus simple (celle de Python).
En Python, il faut convertir une entrée numérique (par défaut, une chaîne de caractères) en entier ou en flottant (nombre avec virgule). Utiliser l'instruction int() ou float() selon le but recherché. Dans la version actuelle, cette instruction n'est pas disponible. Taper l'initialisation souhaitée directement dans le programme (ex : A=0,39)

Avec ces instructions, on va écrire un algorithme qui calcule l'image y d'un nombre x (donnée saisie en entrée) par une fonction f (on va prendre ici $f(x)=\frac{1}{(5x-2)(2-x)}$) et qui affiche le résultat.

- Traduire l'algorithme simplissime ci-dessous dans le langage de votre calculatrice

Algorithme	Programme en Basic Casio	Programme en Basic TI	Programme en Python Numworks
Saisir A A=1/((5A-2)(2-A)) Afficher A	?→A 1÷((5A-2)×(2-A))→B B◢	Input A 1÷((5A-2)×(2-A)) $^{STO}\rightarrow$ B Disp B	a=0 [input non disponible] a=1÷((5×a-2)×(2-a)) print(a)

- Programmer cet algorithme sur votre calculatrice, puis tester le avec la valeur x=1 (on trouve $y=\frac{1}{3}$, soit ≈0,3333)
- Compléter le tableau de données

x	0	0,39	0,41	0,75	1	1,5	1,99	2,01	2,5	3
$f(x)$	-0,25	-12,42	12,578...	0,457...	0,333...	0,363...	12,578..	-12,422...	-0,190...	-0,076...

Commentaire 1 : Vous avez noté l'inversion entre l'écriture dans l'algorithme A=1÷((5A-2)×(2-A)), qui affecte dans la mémoire A le résultat du calcul 1÷((5A-2)×(2-A)), et l'écriture dans les programmes 1÷((5A-2)×(2-A))→A qui effectue la même chose (cette inversion n'est pas obligatoire dans tous les langages de programmation ; en Python par exemple, cette affectation s'écrit sans inversion A=1÷((5×A-2)×(2-A)).

Commentaire 2 : Ici, on peut utiliser une seule mémoire (il serait plus juste de dire une seule variable). On a choisit de mettre tout dans A. Mais on aurait pu choisir de mettre l'image dans une 2de mémoire, par exemple B. L'algorithme deviendrait « Saisir A ; B=1÷(5A-2)÷(2-A) ; Afficher B ».

- Programmer cet algorithme sur votre calculatrice, puis tester le avec la valeur x=1 (on trouve $y=\frac{1}{3}\approx 0{,}333$). Compléter le tableau de données

x	0	0,3	0,39	0,5	1	1,5	1,9	2,1	2,5	3
$f(x)$	-0,25	-1,176...	-12,42	1,333...	0,333...	0,363...	1,333...	-1,176...	-0,19	-0,08

Commentaire 3 : Rassurez-vous, votre calculatrice dispose de fonctions plus performantes pour obtenir un tableau de données (et aussi l'affichage graphique). Ce premier exercice était juste destiné à se familiariser avec le mode programme et l'affectation de valeurs dans une mémoire.

2) Boucles

Une boucle permet d'exécuter plusieurs fois une succession d'instructions. Il en existe deux sortes :

	Algorithme	Basic Casio	Basic TI	Python Numworks
Boucle « pour » : on fait un nombre fixé (N) de tours	Pour I allant de 1 à N : [instructions] fin de la boucle pour	For 1→I To N [instructions] Next	For (I,1,N) [instructions] End	For i in range(n) : [instructions] ↑ noter le décalage (indentation)
Boucle « tant que » : on fait des tours tant qu'une condition est vraie	Tant que I<N : [instructions] fin de la boucle while	While I<N [instructions] WEnd	While I<N [instructions] End	While i<n : [instructions] ↑ noter le décalage (indentation)

Remarques : En Python, il y a deux points à la fin de l'en-tête de boucle ; ensuite il faut une indentation (un seul espace suffit). Avantage : il n'y a pas de marque de fin de boucle, juste un retour sans indentation. La syntaxe Python range(n) correspond aux valeurs 0,1,2,...,(n−1). Si on veut aller de 1 à n (pas seulement n valeurs), il faut écrire range(1,n+1).

D'une façon générale, l'instruction « For I... » gère l'incrémentation de la variable I (elle augmente de 1 à chaque tour) ; l'instruction « While I... » ne fait pas cela. Si on utilise une variable I qui doit être incrémentée à chaque tour, il faut le faire dans les instructions (I=I+1).

a) L'algorithme suivant calcule et affiche la somme s des n premiers carrés d'entiers : $s=1^2+2^2+3^2+...+n^2$:

- Traduire l'algorithme ci-dessous dans le langage de votre calculatrice

Algorithme	Programme en Basic Casio	Programme en Basic TI	Python Numworks
Saisir N S=0 Pour I allant de 1 à N S=S+I² fin de la boucle « Pour » Afficher S	?→N 0→S For 1→I To N S+I^2→S Next S◢	Input N 0 STO→S For (I,1,N) S+I^2 STO→S End Disp S	n=100 s=0 for i in range (1,n+1) : s=s+i**2 print(s)

*Remarque : En Python, le carré se note i**2 (2 fois le signe de la multiplication)*

- Programmer cet algorithme sur votre calculatrice. Tester le programme avec la valeur n=5 (on trouve s=55). Calculer $S_{100}=1^2+2^2+3^2+...+100^2$: S_{100}=338 350 ; $S_{1000}=1^2+2^2+3^2+...+1000^2$: S_{1000}=333 833 500

b) L'algorithme suivant détermine la valeur de n à partir de laquelle la somme $s=1^2+2^2+...+n^2$ dépasse un nombre m donné (m étant une donnée saisie par l'utilisateur) et qui affiche cette valeur de n.

- Traduire l'algorithme ci-dessous dans le langage de votre calculatrice

Commentaire 1 : Ici, on ne sait pas combien de tours de boucle il faut faire. On doit donc utiliser une boucle « tant que » avec une condition (un test) qui est vraie au départ et qui, lorsqu'elle deviendra fausse, provoquera la sortie de boucle. Ici, la condition est S<M. Celle-ci est réalisée (vraie) dès lors qu'on entre une valeur de M>0. On entre alors dans la boucle, et au fur et à mesure que S est augmenté, on teste si la valeur de S vérifie toujours la condition. Lorsque S a dépassé M, on sort de la boucle et on affiche la valeur de I qui a provoqué cette sortie.

Algorithme	Programme en Basic Casio	Programme en Basic TI	Python Numworks

Saisir M
I=0
S=0
Tant que S<M
I=1+I
S=S+I²
fin de la boucle « tant que »
Afficher I
Afficher S
	?→M
	0→I
	0→S
	While S<M
	1+I→I
	S+I^2→S
	WhileEnd (ou WEnd)
	I◢
	S◢
		Input N
		0 STO→I
		0 STO→S
		While S<M
		1+I STO→I
		S+I^2 STO→S
		End
		Disp I
		Disp S
			m=50
			i=0
			s=0
			while s<m :
			  i=i+1
			  s=s+i**2
			print(i)
			print(s)

Commentaire 2 : Dans l'algorithme d'une boucle « tant que », on doit gérer l'initialisation et l'incrémentation du compteur I puisque la boucle « tant que » fonctionne sans compteur. Ce qu'il ne faut pas faire, c'est inverser, sans réfléchir, les instructions « I=1+I puis S=S+I² » en écrivant « S=S+I² puis I=1+I », ou alors il faudrait initialiser I à 1 et aussi, il faudrait aussi enlever 1 à la valeur finale affichée. L'ordre des instructions a une importance ici car on utilise I pour calculer S.

Commentaire 3 : On aurait pu appeler le compteur N au lieu de I puisque ici, on n'utilise pas ce nom de variable. La notation utilisée pour la puissance 2 n'est peut-être pas celle que vous avez utilisé (I² est possible sur les calculatrice) mais c'est la notation en ligne généralement admise des puissances. En Python, on note I**2.

Commentaire 4 : On peut afficher i et s avec l'instruction print(i,s) mais la sortie n'est pas très claire (cela affiche 5 55 pour le test). L'affichage choisi sépare les sorties sur deux lignes. Avec un Python complet on soignera davantage encore la sortie, par exemple en écrivant : print("au rang n={}, la somme s vaut : {} et dépasse {} pour la 1ere fois".format(i, s,m))

Tester le programme avec les valeurs m= 50 (on trouve n=5 et s=55).

Calculer n pour m= 10^6 : On trouve n=144 et s=1 005 720.

Notre programme affichant S, on peut le donner. Mais, dans la consigne, ce n'était pas demandé.

3) Instructions conditionnelles

	Algorithme	Basic Casio	Basic TI	Python Numworks
Si une condition (I<N) est vraie on exécute les instructions I_1, sinon (facultatif) on exécute les instructions I_2.	Si I<N alors I_1 sinon I_2 fin du « Si »	IF I<N Then I_1 Else I_2 IfEnd	IF I<N Then I_1 ELSE I_2 End	If i<n : I_1 else : I_2

a) Test simple

- Écrire l'algorithme qui demande d'entrer un nombre x (entre 0 et 10) et qui calcule l'aire du polygone restant (situation vue en cours). Traduire ensuite l'algorithme dans le langage de votre calculatrice.

Rappel : $A(x)=100-2x^2$ si $x\leq 5$ et $A(x)=100-2x^2+4(x-5)^2$ si $x>5$.

Algorithme	Programme en Basic Casio	Programme en Basic TI	Python Numworks

Saisir X
Si X≥5
Alors Y=100-2X²+4(X-5)²
Sinon Y=100-2X²
fin du Si
Afficher Y
	?→X
	If X≥5
	Then 100-2X²+4(X-5)²→Y
	Else 100-2X²→Y
	IfEnd
	Y◢
		Input X
		If X≥5
		Then 100-2X²+4(X-5)²→Y
		Else 100-2X²→Y
		End
		Disp Y
			Input x
			If x≥5 :
			  100-2*x**2+4*(x-5)**2→y
			else :
			  100-2*x**2→y
			print(y)

Commentaire 1 : Ici, on a utilisé deux mémoires : X et Y.
On aurait pu n'en utiliser qu'une seule, par exemple en écrivant :
« Saisir X ; Si X≥5 ; Alors X=100-2X²+4(X-5)² ; Sinon X=100-2X² ; fin du Si ; Afficher X »
Il faut se méfier de ne pas écraser une valeur qui serait utile. Ici, on choisit une affectation parmi deux, on peut perdre la valeur de X puisqu'elle n'est plus utile ensuite.

- Programmer cet algorithme sur votre calculatrice.
  Compléter le tableau de données de la fonction A. (Cela a été fait en cours)

x	0	1	2	3	4	5	6	7	8	9	10
$A(x)$	100	98	92	82	68	50	32	18	8	2	0

Remarque : Dans une boucle « While » ou une instruction conditionnelle, on utilise généralement un test simple, c'est-à-dire un test qui utilise un des symboles <, >, =, ≤, ≥ ou ≠. On peut associer des tests simples pour examiner des situations plus complexes :

- dans le test $x > 0$ et $i < 100$ (et s'écrit *and*), pour effectuer I_1, il faut que *les deux* conditions soient vraies
- dans le test $x > 0$ ou $x < -1$ (ou s'écrit *or*), pour effectuer I_1, il faut qu'*une au moins* des conditions soit vraie

b) Application

On veut calculer la somme $Z(n)= 1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\frac{1}{5}-....$ (somme des inverses des entiers successifs affectés alternativement d'un signe + ou d'un signe –) pour différentes valeurs de l'entier n.

- Écrire un algorithme qui utilise un test (le reste de la division euclidienne de n par 2 est noté n%2) et un qui n'utilise pas de test (penser à la fonction $n \mapsto (-1)^n$)
- Traduire un des deux algorithmes en programme pour votre calculatrice et déterminer Z(100).

L'algorithme proposé en cours utilise la fonction $n\mapsto(-1)^n$ qui prend pour valeur 1 ou -1 selon que n est pair ou impair. Ici, pour obtenir $\frac{-1}{2}$ quand n=2, il faut écrire $(-1)^{n+1}$ qui vaut alors $(-1)^3 = -1$ car $(-1)^2 = 1$.

- La façon la plus simple de traduire le calcul de $Z(n)$ en algorithme est donc :

Algorithme	Programme en Basic Casio	Programme en Basic TI	Python Numworks
Saisir N S=0 Pour I allant de 1 à N S=S+(-1)^(N+1)÷I fin de la boucle « Pour » Afficher S	?→N 0→S For 1→I To N S+(-1)^(I+1)÷I→S Next S◢	Input N 0 STO→S For (I,1,N) S+(-1)^(I+1)÷I STO→S End Disp S	n=100 s=0 for i in range (1,n+1) : s=s+(-1)**(i+1)/i print(s)

Déterminons Z(100) : on trouve 0,688172179310195 ; déterminons Z(1000) : on trouve 0,6926474305598223
Remarque : ce nombre s'approche de l'irrationnel ln2=0,6931471806... (logarithme népérien de 2)

- Avec un test de parité :

Algorithme	Programme en Basic Casio	Programme en Basic TI	Python Numworks
Saisir N S=0 Pour I allant de 1 à N Si N%2=0	?→N 0→S For 1→I To N If N%2=0	Input N 0 STO→S For (I,1,N) If N%2=0	n=100 s=0 for i in range (1,n+1) : if i%2==0 :

Alors S=S-1÷I Sinon S=S+1÷I fin de la boucle « Pour » Afficher S	Then S-1÷I→S Else S+1÷I→S IfEnd Next S◢	Then S-1÷I→S Else S+1÷I→S End End Disp S	s=s-1/i else : s=s+1/i print(s)

Commentaire 1 : Ici, on doit trouver la véritable syntaxe du test N%2=0 sur sa calculatrice. L'opération A%B signifie reste de la division euclidienne de A par B. Elle est écrite A%B en Python.
En Casio c'est plutôt Rmdr (*remainder*, dans le menu « option », « calc ») qui s'utilise comme % : 5 Rmdr 3= 5%3 = 2 (le reste de la division euclidienne de 5 par 3 est 2).
Sur TI ce sera REMAINDER (dividende,diviseur) donc REMAINDER (5,3) donne 2.
Sur d'autres calculatrices, ce sera noté Mod (*modulo*) et sur le tableur d'OpenOffice aussi, c'est Mod(dividende;diviseur) donc Mod(5;3) donne 2.
NB : Bien sûr, on peut aussi utiliser un autre moyen. Si Partent désigne la fonction partie entière (voir en cours), le reste de la division de X par Y vaut : X-Y×Partent(X/Y). Par exemple 5-3×Partent(5/3)=5-3*1=2
Commentaire 2 : En Python, le test d'une égalité s'effectue au moyen de deux =. L'affectation demande un seul = mais ici, si on écrit « if i%2=0 : » (avec un seul =) il y aura une erreur à l'exécution. Un test comme « if i=0 : » ne donnera pas une erreur mais l'affectation de 0 dans i, ce qui renvoie une valeur « correcte », comme si le test était vérifié... alors qu'il n'a pas été effectué.

- Une autre méthode : changer le signe d'une variable B à chaque passage dans la boucle.

Cette variable B est initialement égale à 1. Avec cette méthode astucieuse on a la solution la plus simple.

Algorithme	Programme en Basic Casio	Programme en Basic TI	Python Numworks
Saisir N S=0 B=1 Pour I allant de 1 à N S=S+B÷I B=-B fin de la boucle « Pour » Afficher S	?→N 0→S 1→B For 1→I To N S+B÷I→S -B→B Next S◢	Input N 0 ^{STO}→S 1 ^{STO}→B For (I,1,N) S+B÷I→S -B→B End Disp S	n=100 s=0 b=1 for i in range (1,n+1) : s=s+b/i b=-b print(s)

La morale de cette histoire : il n'y a pas une seule façon de faire.