Correction du Devoir de Programmation

# Exercice 1

1. Type de retour : string (chaîne de caractères)

2. Tracés :

- inconnu(12,2) → "2\*2\*3"

- inconnu(17,2) → "17"

Rôle : retourne la décomposition en facteurs premiers.

3. Version itérative :

fonction decomposition(n):

début

resultat ← ""

d ← 2

tant que n > 1 faire

si n % d == 0 alors

si resultat ≠ "" alors

resultat ← resultat + "\*"

resultat ← resultat + str(d)

n ← n // d

sinon

d ← d + 1

retourner resultat

fin

# Exercice 2

1. Type : int

2. Tracés :

- inconnu(48,18) → 6

- inconnu(12,9) → 3

Rôle : calcule le PGCD (Euclide).

3. Itératif :

fonction pgcd(a, b):

début

tant que b ≠ 0 faire

temp ← b

b ← a % b

a ← temp

retourner a

fin

# Exercice 3

1. compression\_RLE(chaine):

fonction compression\_RLE(chaine):

début

resultat ← ""

compteur ← 1

pour i de 1 à longueur(chaine) - 1 faire

si chaine[i] == chaine[i-1] alors

compteur ← compteur + 1

sinon

resultat ← resultat + chaine[i-1] + str(compteur)

compteur ← 1

resultat ← resultat + chaine[-1] + str(compteur)

retourner resultat

fin

2. decompression\_RLE(compresse):

fonction decompression\_RLE(compresse):

début

resultat ← ""

i ← 0

tant que i < longueur(compresse) faire

lettre ← compresse[i]

i ← i + 1

nombre ← ""

tant que i < longueur(compresse) et compresse[i] est chiffre faire

nombre ← nombre + compresse[i]

i ← i + 1

resultat ← resultat + lettre \* entier(nombre)

retourner resultat

fin

# Exercice 4

fonction crypter\_texte(texte):

début

M ← creer\_matrice()

L ← longueur(texte)

resultat ← ""

pour chaque caractère c dans texte faire

si c == " " alors

val ← 32 \* L

sinon si c == "Z" ou c == "z" alors

val ← ASCII(c) \* L

sinon

maj ← majuscule(c)

trouvé ← faux

pour i de 0 à 4 faire

pour j de 0 à 4 faire

si M[i][j] == ASCII(maj) alors

val ← (i+1) \* (j+1) \* L + ASCII(maj)

trouvé ← vrai

quitter boucles

si non trouvé alors

val ← ASCII(c)

resultat ← resultat + chr(val % 256)

retourner resultat

fin

# Exercice 5

fonction calcul\_age(naissance):

année ← extraire\_année(naissance)

retourner 2025 - année

fonction generer\_email(nom, prenom):

nom ← majuscules(nom)

binaire ← ""

pour chaque lettre dans nom faire

si lettre == 'A' alors binaire ← binaire + "1010"

...

sinon binaire ← binaire + "1111"

nb\_1 ← compter '1' dans binaire

email ← prenom + "." + nom + str(nb\_1) + "@isig.tn"

retourner email

fonction generer\_numcart(nom, prenom, naissance):

a ← ASCII(première lettre du nom)

b ← ASCII(dernière lettre du prénom)

ppcm ← PPCM(a, b)

pgcd ← PGCD(a, b)

chiffre ← premier chiffre de naissance

retourner "132" + chiffre + str(ppcm) + str(pgcd)

fonction generer\_password(binaire\_nom, age):

tant que longueur(binaire\_nom) % 3 ≠ 0 faire

binaire\_nom ← binaire\_nom + "0"

resultat ← ""

pour i de 0 à longueur(binaire\_nom) - 1 par pas de 3 faire

groupe ← binaire\_nom[i:i+3]

val ← entier(groupe, 2)

resultat ← resultat + chr(val \* age)

retourner resultat

# Exercice 6

1. Ordre de récurrence : ordre 1 (dépend de Wn-1 seulement).

2. ApproxC(x):

fonction ApproxC(x):

début

epsilon ← 10^-6

W ← 1

n ← 1

boucle:

Wn ← W + (x / n) \* W

si abs(Wn - W) / W < epsilon alors

retourner Wn

W ← Wn

n ← n + 1

fin