

Séance 2 : BOUCLES WHILE ET RÉCURSIONS

L1 – Université Côte d’Azur

Exercice 1 – Factorielles (★)

Écrivez une fonction `fact(n)` qui renvoie $n!$, la factorielle de n :

1. en utilisant une récurrence ;
2. en utilisant une boucle `while`.

Exercice 2 – Affichage de factorielles (★)

1. En utilisant la fonction `fact(n)` de l’exercice précédent, écrivez une fonction `affiche_facts(n)` qui affiche sur n lignes les entiers $1!, 2!, 3!, \dots, n!$.
2. Combien de multiplications sont effectuées par votre code lorsque vous exécutez `affiche_facts(n)` ?
3. Proposez une solution `affiche_facts_efficace(n)` utilisant moins de multiplications.

Exercice 3 – Logarithme entier (★)

Soit n un entier positif. On appelle *logarithme entier* de n l’entier $le(n)$ correspondant au nombre de fois où il faut diviser n par deux avant d’atteindre 1 ou 0. Par exemple, $le(5) = 1 + le(2) = 1 + (1 + le(1)) = 2$.

1. Calculez à la main $le(3)$, $le(5)$, et $le(11)$ puis écrivez les tests correspondants avec `assert`.
2. Écrivez la fonction récursive `le(n)` qui renvoie le logarithme entier de n .

Exercice 4 – Partie entière de la racine carrée (★)

Soit n un entier positif. La partie entière de la racine carrée de n , notée $\lfloor \sqrt{n} \rfloor$, est le plus grand entier k tel que $k^2 \leq n$. Écrivez une fonction `racine_entiere(n)` qui renvoie la partie entière de la racine carrée de n .

Complément sur la fonction print

Par défaut, la fonction `print` affiche un caractère `␣` (espace) entre chacun de ses arguments et termine en allant à la ligne à la fin. Deux paramètres optionnels de la fonction `print` permettent de modifier ce comportement par défaut : `sep` et `end`. Par exemple, le programme :

```
1 print('Comportement', 'par', 'default.', sep=' ', end='\n')
2 print('Sur', 'une', 'seule', 'ligne', 'avec', 'des', 'tirets.', sep='-', end='')
3 print('Ancienne ligne.\n', 'Nouvelle ligne', '.')
```

affiche

```
1 Comportement par default.
2 Sur-une-seule-ligne-avec-des-tirets.Ancienne ligne.
3 Nouvelle ligne .
```

Le caractère `'\n'` est un caractère de contrôle qui lorsqu’il est affiché provoque un passage à la ligne. Dans la deuxième instruction `print`, on a remplacé le caractère de fin `'\n'` par `' '` : `print` ne va pas à la ligne.

Complément sur les chaînes de caractères

On peut convertir un entier en chaîne de caractères : `str(42) == '42'`. Réciproquement, on peut convertir une chaîne de caractères en entier : `int('42') == 42`. Pour concaténer deux chaînes de caractères, on utilise l’opérateur `+`. On a par exemple `'caram' + 'bolage' == 'carambolage'`; ce `+` n’a rien à voir avec l’addition sur les entiers : `'2' + '3'` vaut `'23'`, mais pas `'3'+'2'`.

Exercice 5 – Frise (*- **)

1. Écrire une fonction `frise(n)` qui affiche une frise de longueur `n` alternant deux symboles `#` et `*`.

```

1 >>> frise(0)
2
3 >>> frise(1)
4 #
5 >>> frise(5)
6 #####
7 >>> frise(10)
8 #####*#####
    
```

2. Peut-on tester une telle fonction avec `assert`?

Exercice 6 – Écriture binaire (**- * * *)

1. Écrivez une fonction `affiche_calcul_binaire(n)` qui affiche le calcul de la représentation binaire de `n`, de sorte que l’on peut lire *verticalement* la représentation en binaire de `n`. Par exemple, pour $13 = (1101)_2$, on obtiendra dans la console :

```

1 >>> affiche_calcul_binaire(13)
2 1 car 13 = 2 × 6 + 1
3 0 car 6 = 2 × 3 + 0
4 1 car 3 = 2 × 1 + 1
5 1
    
```

2. Écrivez une fonction récursive `affiche_binaire(n)` qui affiche l’écriture binaire de `n` dans le sens de lecture **usuel**. Par exemple, on doit obtenir :

```

1 >>> affiche_binaire(13)
2 1101
    
```

3. Écrivez cette fonction en utilisant une boucle `while`.

Exercice 7 – Décomposition en facteurs premiers (* * *)

1. Écrivez une fonction `affiche_facteurs(n)` qui affiche la liste des facteurs premiers avec multiplicité de `n`. Par exemple, `affiche_facteurs(1176)` affichera :

```

1 >>> affiche_facteurs(1176)
2 2**3 3**1 7**2
    
```

Astuce : on pourra définir des sous-fonctions pour rendre le programme plus lisible.

2. Écrire la même fonction mais en gégrant proprement l’affichage des `+` :

```

1 >>> affiche_facteurs(1176)
2 2**3 * 3**1 * 7**2
    
```