Epreuve d'Informatique
15 mai 2024

Ecrire directement sur les feuilles de l'énoncé. Rendre toutes les feuilles (même vierges) avec votre nom.
Documents autorisés : notes de cours et T.D., polys de cours.
Compétences AF1. Compétences méthodologiques et de conceptualisation (algorithmique)
Compétences AF2. Compétences en programmation
Les deux compétences sont évaluées dans chaque exercice sauf indiqué en début de l'énoncé.
On ne vous demande pas de mettre des docstings mais utiliser des noms de variables explicites !

Exercice I - Evaluation d'expressions arithmétiques simples

(environ 25 minutes)

AF1 – *Compétences méthodologiques et de conceptualisation (algorithmique)*
AF2 – *Compétences en programmation dans le langage de programmation Python (chaînes de caractères, listes, fonctions)*

On considère des expressions sous forme de chaînes de caractères décrivant des sommes de nombres entiers naturels.

Par exemple : "123+15+1", "123+5", "123+15", "123"

Ces chaînes ne comprennent que des **chiffres** et le **symbole '+**'.

Les chaînes suivantes sont considérées comme mal formées : "", "12+", "+12", "12++12"

Ces expressions représentent donc soit un entier naturel, soit une somme d'entiers naturels.

1) Chaîne valide

Écrire une **fonction** *est_valide* qui retourne *True* si la chaîne fournie en paramètre est valide, c'est-à-dire qu'elle ne contient que des caractères autorisés (**chiffres** et "**+**") et que les opérateurs "**+**" sont bien placés (ni au début, ni à la fin, ni en doublon) et *False* sinon. (entre 10 et 13 lignes)

Aide : on peut utiliser la méthode isdigit du module str

ch.isdigit() Retourne True si la chaîne ch ne contient que des chiffres, et False sinon.
Si la chaine est vide elle retourne False.

2) Evaluation

Écrire une **fonction** *evalue* qui évalue la valeur d'une expression de ce type. Si l'expression n'est pas bien formée, cette fonction retourne *False.* (environ 5 et 8 lignes)

Exemple : evalue("123+15+1") retourne 139

Indications :

- *on peut utiliser la méthode split*
- *rappel : la fonction int(ch) permet de convertir une chaîne en un entier int("462") → 462*

Epreuve d'Informatique
15 mai 2024

Exercice II - Données expérimentales

(Environ 20 minutes)

***AF2** – Compétences en programmation dans le langage de programmation Python (fichiers, fonctions)*

Lors d'une série d'expériences à diverses dates, les mesures sont répétées plusieurs fois. L'ensemble des mesures sont stockées dans un fichier de données nommé "*mesures.dat*".

Exemple, dans le fichier « mesures.dat » suivant, les lignes commençant par un "#" fournissent la date de l'expérience, les autres lignes fournissent la suite des mesures réalisées séparées par des espaces

```
# 25 janvier 2020
0.0051 0.0274 0.8883 0.5716 0.1246
# 26 mars 2020
1.4532 1.7710 1.7266 1.8425 1.3793
# 2 avril 2020
5.2299 5.7239 5.7531 5.2266 5.6840
```

On souhaite construire un fichier de données "*bilan.out*" fournissant pour chaque date les valeurs minimales et valeurs maximales de chaque série de mesures.

Exemple, Pour le fichier précédent, un extrait du le fichier obtenu "bilan.out" est fourni ci-dessous :

```
# 25 janvier 2020
min : 0.0051 max : 0.8883
# 26 mars 2020
min : 1.3793 max : 1.8425
# 2 avril 2020
min : 5.2266 max : 5.7531
```

Ecrire une **fonction** *bilan* qui prend en paramètre le nom du fichier des mesures et créé le fichier bilan.
Les lignes commençant par un "#" sont réécrites telles quelles. (*environ 20 minutes – entre 12 et 14 lignes*)
Indications : vous pouvez utiliser les méthodes min et max sur les listes

Epreuve d'Informatique
15 mai 2024

Exercice III - Bataille navale

(Environ 60 minutes)

AF1 – Compétences méthodologiques et de conceptualisation (algorithmique)
AF2 – Compétences en programmation dans le langage de programmation Python (fichiers, fonctions)

On souhaite réaliser le jeu de la bataille navale, on s'appuie sur la modélisation suivante :

- Le jeu est caractérisé par la taille du plateau de jeu (carré) et une liste de bateaux
- chaque bateau est caractérisé par : une liste d'éléments et un état général (intact=0, touché=1 ou coulé=2)
- un élément est caractérisé par sa position sur le plateau de jeu (ligne, colonne) et son état (intact=0 ou touché=1)

1) Classe Element

a) Définir la classe *Element* et son constructeur *__init__*. (*5 lignes*)

Le constructeur reçoit en **paramètres** la position de l'élément (numéro de ligne et numéro de colonne).
Les **attributs** à définir sont :
- La position de l'élément
- Son état qui est initialisé à 0

b) Écrire la méthode accesseur de l'état d'un élément. *(2 lignes)*

c) Définir la méthode *est_touché* qui reçoit en paramètre les coordonnées d'un coup joué et met à jour l'attribut *état* de l'élément à 1 si le coup touche l'élément. (*entre 2 et 4 lignes*)

2) *Classe Bateau*

a) Définir la classe *Bateau* et son constructeur *__init__*. (*environ 14 et 18 lignes*)
Le constructeur reçoit en **paramètres** :

- la longueur du bateau *lg*
- la position (l, c) du premier élément du tableau (à gauche ou en haut)
- la direction du bateau « h » pour horizontale ou « v » pour verticale

Les **attributs** à définir à partir de ces paramètres sont :

- la longueur du bateau
- une liste de *lg* instances de la classe *Element*
- un état initialisé à 0

b) Écrire la méthode accesseur de l'état d'un bateau. *(2 lignes)*

c) Écrire la méthode *__lt__* qui surcharge l'opérateur « < ». Un bateau est considéré comme plus petit qu'un autre si sa longueur est plus petite. (*entre 2 et 4 lignes*)

d) Écrire une méthode *est_touche* qui reçoit en paramètres les coordonnées d'un coup joué (numéro de ligne et numéro de colonne) et met à jour l'attribut *état* de **chaque élément** du bateau. Cette méthode s'appuiera sur la méthode *est_touche* de la classe *Element*. (*3 lignes*)

e) Écrire une méthode *maj_etat* qui met à jour l'attribut *état* du bateau à jour en fonction de l'état de ses éléments (intact=0, touché=1 ou coulé=2). Un bateau est coulé si tous ses éléments sont touchés. (*environ 10 ou 11 lignes*)

f) On a différents types de bateaux en particulier :
   - un croiseur est de taille 3
   - un sous-marin est de taille 2
     - un sous-marin a la possibilité de plonger, il a donc un attribut supplémentaire à *True* s'il est plongé et à False sinon. Au départ, un sous-marin est en surface.
     - Quand un sous-marin est en plongée, il ne peut pas être touché

   Définir les différentes sous-classes correspondant à ces types de bateaux. On demande ici de définir les constructeurs *__init__* mais également les éventuelles fonctions qu'il faut redéfinir. (*environ 10 lignes*)

3) Classe Bataille

a) Définir la classe *Bataille* et son constructeur *__init__*. Le constructeur reçoit ses attributs en paramètres : la dimension du plateau de jeu et la liste des bateaux. (*3 lignes*)

b) Définir une méthode *fin_partie* qui retourne *True* si tous les bateaux sont coulés et *False* sinon. (*environ 5 ou 6 lignes*)

Epreuve d'Informatique
15 mai 2024

Exercice IV - Coloriage de cartes

(Environ 15 minutes)

AF1 – *Compétences méthodologiques et de conceptualisation (algorithmique)*
AF2 *– Compétences en programmation dans le langage de programmation Python (fichiers, fonctions)*

1) Théorème des 4 couleurs

Nous disposons du dictionnaire *couleur_region* contenant pour chaque région la couleur qui lui est associée.

Exemple : dico_couleurs = {'C': 'green', 'B': 'blue', 'D': 'red', 'F': 'red', 'E': 'green', 'A': 'yellow'}

Ecrire une fonction *verif* qui retourne *True* si avec la solution proposée par le biais de ce dictionnaire passé en paramètre, le théorème des 4 couleurs est vérifié et *False* sinon. (*8 ou 9 lignes)*

Rappel : le théorème des 4 couleurs indique qu'il est possible, en n'utilisant que quatre couleurs différentes, de colorier n'importe quelle carte découpée en régions connexes.

Il s'agit donc ici de vérifier que 4 couleurs différentes au plus sont utilisées pour colorier la carte.

(**TSVP**)

2) Voisin commun

On dispose d'un dictionnaire associant à chaque région la liste de ses voisins.

Écrire une fonction *voisin_commun* qui pour un dictionnaire des voisins et deux régions fournies en paramètres retourne *True* si les deux régions ont un voisin commun et *False* sinon. (*entre 5 et 7 lignes*)