GRANDE ÉCOLE D'INGÉNIEURS

UE Informatique
Étape 2 - Les planchettes

Olivier Beaudoux (olivier.beaudoux@eseo.fr)

Semestre 4

Table des matières

1 Le concept de type abstrait

Le document *Elements.txt* donne une vue d'ensemble des éléments entrant en jeu dans l'application « Pytatra ! ». Chaque type d'élément sera implémenté par un *type abstrait*. Les types abstraits sont les ancêtres des *classes* des langages orientés objet, mais ils sont plus simples à appréhender : leur étude permet de passer plus facilement de la programmation centrée sur les fonctions à une programmation centrée sur les classes.

1.1 Utilisation de tuples

Les *tuples* permettent de représenter des données structurées *immuables* (non modifiables). Ils sont donc utilisés pour définir des type abstraits représentant des éléments immuables, au même titre que la classe *String* représente des chaînes de caractères non modifiables.

L'exemple du type abstrait *Vecteur* illustre l'utilisation d'un tuple pour représenter les coordonnées immuables d'un vecteur.

Spécification

La spécification du type abstrait *Vecteur* est la suivante :

***cree(x, y)* :** retourne un tuple représentant un vecteur de coordonnées *x* et *y*. Les coordonnées peuvent être des nombres entiers ou à virgule flottante.

***x(vecteur)* :** retourne la coordonnée *x* du *vecteur* (propriété en lecture seule).

***y(vecteur)* :** retourne la coordonnée *y* du *vecteur* (propriété en lecture seule).

***norme(vecteur)* :** calcule et retourne la *norme* [1] du *vecteur* (propriété en lecture seule).

***somme(vecteur1, vecteur2)* :** calcule et retourne le vecteur *somme* [2] de *vecteur1* et de *vecteur2*.

Travail à faire

Étape 2.1.1

- Réviser la notion de tuple en Python à l'aide du tutoriel du W3Schools.
- Télécharger depuis le campus l'archive « Etape-2.zip » et en extraire les fichiers dans le répertoire « S4-Informatique ».
- Ouvrir une console et se déplacer dans le répertoire « S4-Informatique/Etape-2/Geometrie ».
- Ouvrir les fichiers « Vecteur.py » et « TestVecteur.py » avec votre éditeur texte préféré.
- Analyser le code du module « TestVecteur.py ».
- Implémenter les différentes fonctions décrites telles que spécifiées dans la section précédente.

1. $\|\overrightarrow{v}\| = \sqrt{x^2 + y^2}$ avec $\overrightarrow{v} = x\,\overrightarrow{i} + y\,\overrightarrow{j}$
2. $\overrightarrow{v_1} + \overrightarrow{v_2} = (x_1 + x_2)\,\overrightarrow{i} + (y_1 + y_2)\,\overrightarrow{j}$ avec $\overrightarrow{v_{i=1..2}} = x_i\,\overrightarrow{i} + y_i\,\overrightarrow{j}$

- Tester et mettre au point cette implémentation en exécutant le module « TestVecteur.py ».

Bien comprendre :

- **la fonction cree qui joue le rôle de constructeur ;**
- **le tuple utilisé comme structure de données immuable ;**
- **les propriétés x, y et norme définies en lecture seule.**
- **la fonction somme qui ne modifie PAS les coordonnées des deux vecteurs.**

1.2 Utilisation de dictionnaires

Contrairement aux tuples, les dictionnaires permettent de représenter des données structurées *mutables* (modifiables). Ils sont donc utilisés pour définir des type abstraits représentant des éléments mutables.

L'exemple du type abstrait *Point* illustre l'utilisation d'un dictionnaire pour représenter les coordonnées modifiables d'un point.

Spécification

La spécification du type abstrait *Point* est la suivante :

cree(x, y) : retourne un dictionnaire représentant un point d'abscisse *x* et d'ordonnée *y*. Ces coordonnées peuvent être des nombres entiers ou à virgule flottante.

x(point, valeur=None) : modifie d'abscisse *x* du *point* en lui donnant la nouvelle *valeur*, dans le cas où cette valeur est définie (propriété en lecture et écriture) ; retourne l'abscisse *x* du *point*.

y(point, valeur=None) : idem pour l'ordonnée *y*.

vecteur(point) : retourne un *vecteur* de même coordonnées que le *point*.

Travail à faire

Étape 2.1.2

- Réviser la notion de dictionnaire en Python à l'aide du tutoriel du W3Schools.
- Ouvrir les fichiers *Point.py* et *TestPoint.py* et analyser le code du module *TestPoint.py*.
- Implémenter les différentes fonctions décrites telles que spécifiées dans la section précédente.
- Tester et mettre au point cette implémentation en exécutant le module *TestPoint.py*.

Bien comprendre :

- la fonction cree qui joue toujours le rôle de constructeur ;
- le dictionnaire utilisé comme structure de données mutables ;
- les propriétés x et y définies en lecture-écriture ;
- la fonction deplace qui modifie les coordonnées du point.

1.3 Utilisation de listes

Les listes permettent de stocker des *collections* d'éléments ordonnés et accessibles par un indice. Elles sont donc particulièrement utiles pour les types abstraits mettant en jeu des collections d'éléments, telles que les pioches de joueurs et la pile de planchettes.

L'exemple du type abstrait *Polygone* illustre l'utilisation d'une liste pour stocker les points du polygone, conjointement à un tuple et un dictionnaire.

Spécification

La spécification du type abstrait *Polygone* est la suivante :

cree(x1, y1, x2, y2, couleur, epaisseur) : retourne un tuple représentant un polygone formé d'une liste de points. Ce polygone est initialement un simple segment de droite : la liste est donc initialement constituée des deux points de coordonnées respectives (*x1*, *y1*) et (*x2*, *y2*) et représentant les extrémités du segment. Les traits formant le polygone ont une *couleur* et une *épaisseur* initiales données.

points(polygone) : retourne la liste des *points* formant de *polygone* (propriété en lecture seule).

couleur(polygone, valeur=None) : modifie la *couleur* du *polygone* en lui donnant la nouvelle *valeur*, dans le cas où cette valeur est définie ; retourne la *couleur* du *polygone*. La *couleur* est représentée par une chaîne de caractères définissant son nom.

epaisseur(polygone, valeur=None) : modifie l'*epaisseur* du *polygone* en lui donnant la nouvelle *valeur*, dans le cas où cette valeur est définie ; retourne l'*epaisseur* du *polygone*. L'*epaisseur* est représentée par nombre entier ou à virgule flottante.

ajoute(polygone, x, y) : ajoute un nouveau point de coordonnées *x* et *y* à la fin de la liste des *points* du *polygone*.

Travail à faire

Étape 2.1.3

- Réviser la notion de liste en Python à l'aide du tutoriel du W3Schools.
- Ouvrir les fichiers « Polygone.py » et « TestPolygone.py » et analyser le code du module « TestPolygone.py ».
- Implémenter les différentes fonctions décrites dans la section précédente et mettre au point cette implémentation en exécutant le module « TestPolygone.py ».

Bien comprendre :

- **la fonction cree utilise un tuple comme structure de données principale, ce tuple contenant les structures de données secondaires liste et dictionnaire ;**
- **la propriété points est en lecture seule en tant que contenant[3] mais son contenu peut être modifié.**
- **l'implémentation de la fonction ajoute illustre que le contenu d'un contenant de type liste peut être modifié, sans avoir à reconstruire un nouveau contenant.**

2 Application aux planchettes

2.1 Définition des planchettes

Les planchettes sont représentées par le type abstrait *Planchette*. Ce type abstrait définit les caractéristiques géométriques d'une planchette de numéro donné, conformément à la description fournie dans les règles du jeu (voir l'étape 1).

Spécification

La spécification du type abstrait *Planchette* est la suivante :

***cree(longueur, marge)* :** retourne une structure de données qui représente une planchette avec sa *longueur* totale et la longueur de sa *marge*. Ces dimensions sont deux nombres entiers d'unité le *cm*.

***longueur(planchette)* :** retourne la *longueur* totale de la *planchette*.

***marge(planchette)* :** retourne la longueur de la *marge* de la *planchette*.

***numero(planchette)* :** retourne le numéro de la planchette (nombre entier) selon le format décrit dans la règles du jeu (étape 1).

Travail à faire

Étape 2.2.1

- Se placer dans le répertoire « S4-Informatique/Etape-2/Pytatra ».
- Ouvrir les fichiers « Planchette.py » et « TestPlanchette.py » et analyser le code du module « TestPlanchette.py ».
- Implémenter les différentes fonctions décrites dans la spécification précédente et mettre au point cette implémentation en exécutant le module « TestPlanchette.py ».

Discuter avec votre encadrant de la structure de données choisie pour représenter ce type abstrait.

3. Un tuple, un dictionnaire et une liste sont des *contenants*. Le *contenu* des listes et des dictionnaires sont modifiables, ce qui n'est pas la cas pour les tuples.

2.2 Fenêtre de l'application

L'application s'exécute dans une fenêtre principale représentée par le type abstrait *Fenetre*. Elle contient principalement un « canvas » (toile en français). La bibliothèque Tkinter est utilisée pour implémenter le type abstrait *Fenetre*.

Spécification

La spécification du type abstrait *Fenetre* est la suivante :

Titre : *constante donnant le titre de l'application (« Pytatra ! ») affiché dans la barre de titre de la fenêtre.*

cree(largeur, hauteur) : retourne une structure de données qui représente la fenêtre contenant une toile de *largeur* et *hauteur* données. Ces deux nombres sont des entiers d'unité le pixel.

toile(fenetre) : retourne la *toile* (ou « canvas » en anglais) de la *fenetre*.

largeur(fenetre) : retourne la *largeur* de la *fenetre*.

hauteur(fenetre) : retourne la *hauteur* de la *fenetre*.

tk(fenetre) : retourne l'objet Tkinter instance de la classe *Tk* et représentant la fenêtre Tk de l'application.

affiche(fenetre) : affiche la *fenetre* et exécute la boucle principale de l'objet Tk. Cette fonction se termine lorsque l'utilisateur ferme la *fenetre*.

Travail à faire

Étape 2.2.2

- Réviser l'utilisation de la classe Tk de la bibliothèque Tkinter.
- Réviser l'utilisation de la classe Canvas de la bibliothèque Tkinter.
- Compléter le fichier « Fenetre.py » pour la partie « Etape 2 » uniquement : fonction *cree*, accesseurs et fonction *affiche*.
- Tester ce code en exécutant le module « TestFenetre.py » et le mettre au point jusqu'à ce que les tests soient concluants.

Bien comprendre :

- **l'instanciation (appel des constructeurs) des classes Tk et Canvas de Tkinter ;**
- **la boucle principale du programme qui s'exécute à l'affichage de la fenêtre Tk.**

2.3 Représentation graphique des planchettes

Le type abstrait *Planchette* définit le *modèle* d'une planchette, indépendamment de toute représentation graphique. En informatique, il est indispensable de séparer les *préoccupations* et donc les métiers associés : les architectes logiciels définissent les modèles des objets manipulés et les designers définissent leurs représentations graphiques.

Les représentations graphiques sont appelées des « vues » en informatique, par opposition aux « modèles ». Le fichier « VuePlanchette.py » contiendra l'implémentation de la construction de la vue pour

nos planchettes. Du fait qu'il n'est pas attendu d'interaction sur les vues des planchettes, aucune structure de données n'est attendue : un simple module « VuePlanchette » suffit, complété par son module de test « TestVuePlanchette ».

Spécification

La spécification du module *VuePlanchette* est la suivante :

Facteur : constante représentant le *facteur* conversion du nombre de centimètres relatif aux dimensions de la planchettes en nombre de pixels relatif à l'affichage sur l'écran ;

pixels(cm) : retourne le nombre de pixels correspondant au nombre de *cm*.

dessine(fenetre, planchette, x0, y0) : dessine la *planchette* à l'intérieur de la toile de la *fenetre* et à partir des coordonnées *x0* et *y0* d'unité le pixel.

Travail à faire

La figure 1 précise les coordonnées à prendre en considération pour le tracé du dessin (fonction *dessine*) :

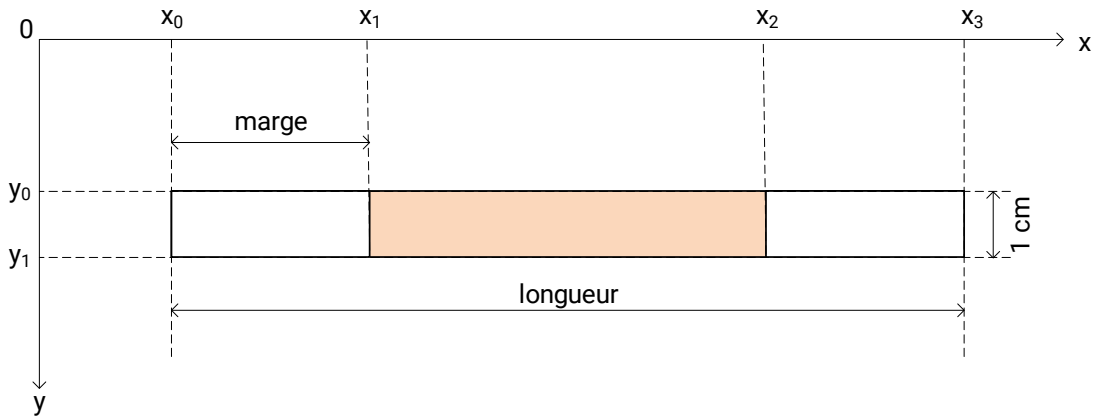

FIGURE 1 – Coordonnées des planchettes

Les axes (Ox) et (Oy) représentent le repère de la toile de la fenêtre principale ; son origine $(0, 0)$ est donc située au coin haut-gauche de l'écran. Leur unité est le pixel, d'où la nécessité de convertir les centimètres (longueur, marge et épaisseur de la planchette) en pixels. Les six coordonnées $x_{i=0..3}$ et $y_{j=0..1}$ permettent de caractériser les points essentiels de la représentation graphique des planchettes.

Étape 2.2.3

- Comprendre l'utilisation de la fonction create_rectangle du « canvas » Tkinter.
- Compléter le fichier « VuePlanchette.py ». La fonction *dessine* devra définir six variables locales représentant les six coordonnées $x_{i=0..3}$ et $y_{j=0..1}$.
- Tester ce code en exécutant le module « TestVuePlanchette.py ».

Bien comprendre :

- **l'utilisation du « canvas » Tkinter ;**
- **la géométrie des planchettes qu'il faudra également intégrer dans les représentations de la pile de planchettes et des pioches des joueurs ;**
- **le fait que « VuePlanchette.py » définisse un module et non pas un type abstrait.**