Sébastien Hoarau
Thierry Massart
Jean Olgiati
Isabelle Poirier

Octobre 2020

Conditions d'utilisation du contenu du cours

CC-BY-SA : Sébastien Hoarau - Thierry Massart - Jean Olgiati - Isabelle Poirier

Attribution - Partage dans les Mêmes Conditions

Les contenus peuvent être partagés et adaptés, y compris dans un but commercial, sous réserve de créditer l'oeuvre originale et de partager l'oeuvre modifiée dans les mêmes conditions.

Table des matières

Avant Propos **I**

1 Votre projet **3**
1.1 Quête dans le château au sommet du Python des Neiges . . . 3
1.1.1 Information préliminaire . . . 3
1.1.2 Présentation du projet . . . 3
1.1.3 En pratique . . . 6
1.2 Détails des phases du projet . . . 9
1.2.1 Niveau 1 : construction et affichage du plan du château . . . 9
1.2.2 Niveau 2 : gestion des déplacements . . . 11
1.2.3 Niveau 3 : collecte d'objets dans le labyrinthe . . . 12
1.2.4 Niveau 4 : Le jeu escape game complet avec questions-réponses . . . 13
1.2.5 Remise et évaluation du projet . . . 14
1.3 Phase d'évaluation par les pairs . . . 17
1.3.1 Phase d'évaluation par les pairs . . . 17
1.4 Phase d'auto-évaluation du projet . . . 17
1.4.1 Phase d'auto-évaluation . . . 17

2 Projet donné lors des trois premières sessions du MOOC Apprendre à coder avec Python **19**
2.1 Le projet Vasarely . . . 19
2.1.1 Énoncé du projet . . . 19
2.1.2 En pratique . . . 21
2.1.3 Remise et évaluation du projet . . . 26
2.1.4 Galerie des oeuvres produites par le projet . . . 28

3 Annexes **31**

Index **31**

CC BY-SA 3.0 - 2020 - Sébastien Hoarau - Thierry Massart - Jean Olgiati - Isabelle Poirier

Avant Propos

Bienvenue dans la session 4 (2020-2021) de notre MOOC intitulé Apprendre à coder avec Python diffusé sur la plateforme FUN.

Cette nouvelle version du cours est le fruit d'un travail important que nous avons réalisé depuis la première session diffusée en 2019.

Nous avons apporté les améliorations ou ajouts suivants :

Contenu

Ce cours intègre désormais :

- un manuel comme support écrit pour accompagner le MOOC ;
- des sous-titres aux vidéos (dans la version en ligne), activables lors de la visualisation (attention : ne fonctionne pas avec le navigateur Safari) ;
- une Foire Aux Questions (FAQ) pour vous permettre de trouver une réponse aux questions fréquemment posées ;
- des résumés des vidéos du cours ;
- des énoncés de notre exerciseur UpyLaB, mieux structurés avec des exemples de ce qui est demandé ;
- un choix des exercices UpyLaB encore mieux adapté au cours ;

Par ailleurs, cette quatrième session vous propose un nouveau projet (« Quête au Château du sommet du Piton Rocheux ») où il vous sera demandé de construire et de jouer à votre jeu d'évasion (escape game) sur ordinateur.

Horaire de diffusion

Nous avons également aménagé l'horaire de diffusion pour vous permettre de suivre ce cours soit sur un semestre (septembre-décembre ou janvier-mai), soit sur toute l'année scolaire.

Nous vous invitons à vous inscrire dès à présent à cette nouvelle session pour bénéficier de l'ensemble des ressources.

Bon travail et bon amusement

Isabelle - Jean - Sébastien - Thierry

Septembre 2020

CC BY-SA 3.0 - 2020 - Sébastien Hoarau - Thierry Massart - Jean Olgiati - Isabelle Poirier

CHAPITRE 1

Votre projet

1.1 Quête dans le château au sommet du Python des Neiges

1.1.1 Information préliminaire

INFORMATION PRÉLIMINAIRE

Le projet que nous vous proposons dans ce MOOC est évalué via une évaluation entre pairs, suivie d'une auto-évaluation. Contrairement aux autres exercices, la réalisation du projet, si vous décidez de la faire, doit donc être synchronisée avec les autres apprenants pour permettre à cette évaluation de se dérouler harmonieusement.

Nous avons décidé de vous proposer 2 sessions pour réaliser votre projet. Une session d'automne (voir calendrier), et une session du printemps. Libre à vous de participer à l'une, l'autre ou même aux deux sessions, sachant que dans ce cas, la note maximale des deux notes qui vous seront attribuées sera retenue dans l'évaluation finale.

Si vous réalisez le parcours bleu ou le parcours rouge, et désirez participer à la session d'automne pour le projet, il est temps de réaliser ce dernier.

Ce module donne l'énoncé du projet que nous vous proposons dans le cadre de ce cours en ligne, ainsi que les procédures d'évaluation.

Note : Si vous ne comptez pas faire de projet, nous vous proposons de parcourir quand même cette section ; vous pourrez encore décider de ne pas faire de projet comme vous aviez prévu ou bien d'en faire une partie ou finalement de le faire soit en session d'automne ou plus tard à la session du printemps.

Le projet peut être réalisé dès que vous avez vu la matière du cours jusque et y compris la section 6.1, soit les instructions et fonctions Python, les séquences (chaînes de caractères, tuples et listes) et le début sur les dictionnaires et les ensembles.

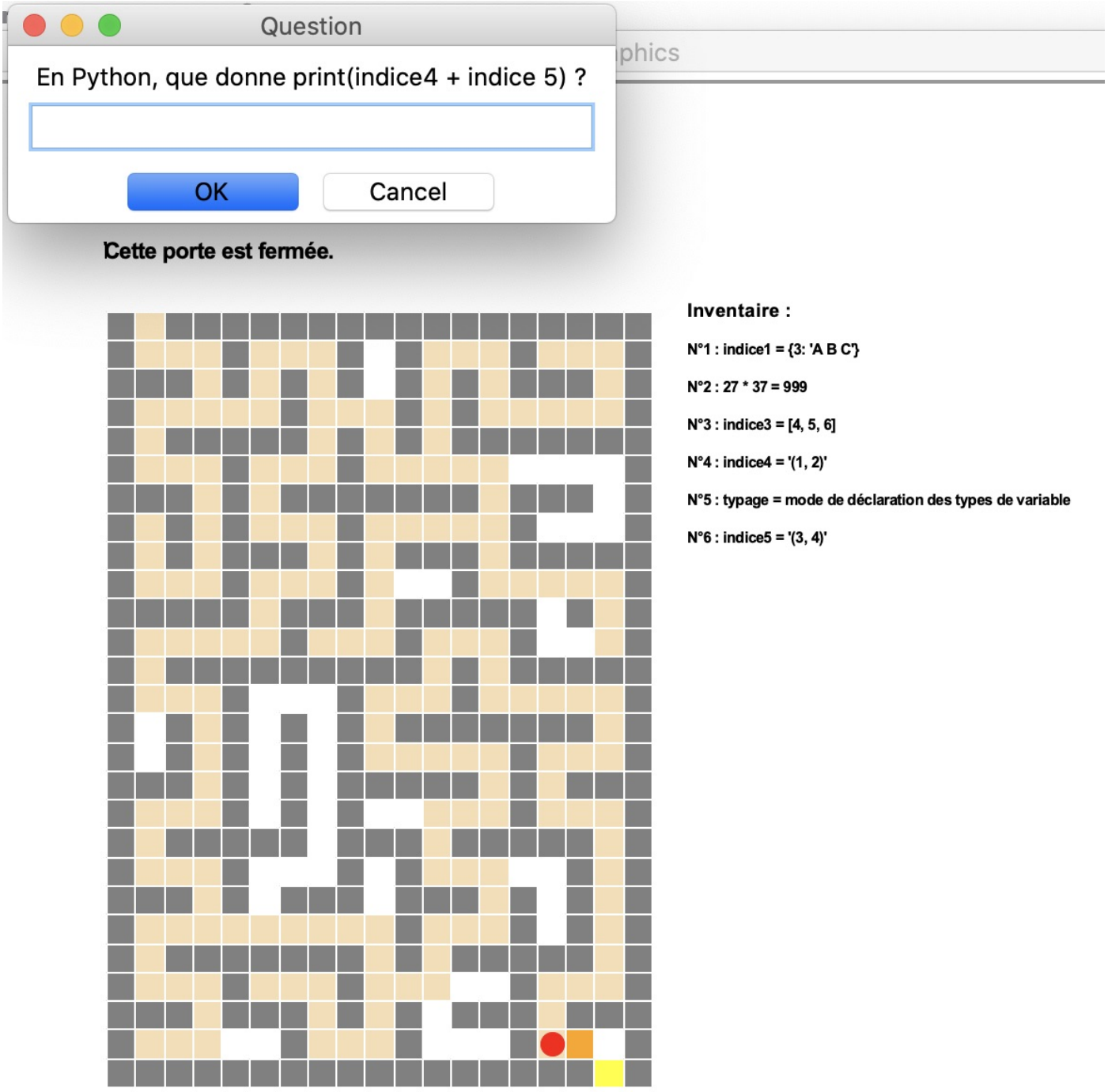
1.1.2 Présentation du projet

Lancelot entre dans le château au sommet du Python des Neiges, muni de son précieux sac de rangement et de sa torche fraîchement allumée aux feux de Beltane. Il doit trouver la statue de sainte Axerror, le chef-d'oeuvre de Gide de Rome, dit le « tyran malfaisant éternel ».

Heureusement, pour l'aider dans sa quête, Merlin, son maître, lui a fourni un plan minutieux des salles et des couloirs du château. Ce plan lui sera fort utile, vu l'énormité du bâtiment, tant par sa taille que par le nombre de ses pièces !

Avant de partir, Merlin lui a donné la clef de la porte d'entrée du château et lui a prodigué moults conseils, dont celui de bien garder tous les objets qu'il trouvera lors de sa quête : ceux-ci lui permettront de répondre aux diverses énigmes que ne manqueront pas de poser les gardes postés devant les portes à l'intérieur du château.

Merlin a affirmé à son disciple que, s'il procède avec intelligence, sa quête sera satisfaite.



Le jeu en une image

EN BREF

Ce projet, si vous le menez jusqu'au bout, va vous faire programmer un petit jeu du type *jeu d'évasion* (*escape game*) dans lequel le joueur commande au clavier les déplacements d'un personnage au sein d'un « château » représenté en plan. Le château est constitué de cases vides (pièces, couloirs), de murs, de portes, que le personnage ne pourra franchir qu'en répondant à des questions, d'objets à ramasser, qui l'aideront à trouver les réponses à ces questions et de la case de sortie / quête du château. Le but du jeu est d'atteindre cette dernière.

Le programme utilisera le module turtle comme interface graphique et comportera deux parties principales :

1. le tracé du plan du château,
2. la gestion du jeu sur le plan tracé (gestion des déplacements du personnage au sein du château, affichage des objets recueillis, gestion de l'ouverture des portes).

Les données nécessaires (plan du château, objets, portes) sont encodées dans 3 fichiers texte. Vous disposerez d'un jeu de fichiers de données que nous vous proposons et devrez réaliser un programme de jeu qui met en œuvre ces données. Vous pourrez ensuite si vous le souhaitez encoder votre propre château en préparant d'autres fichiers de données, qui tourneront avec le même programme. Votre château pourra aussi bien être du type labyrinthe (entièrement fait de couloirs étroits, sans portes) que du type escape game (un ensemble de pièces, entre lesquelles un personnage circule pour rassembler des objets lui permettant de répondre à des questions ou résoudre des énigmes).

Ce projet vous permettra d'écrire un code plus substantiel que ce que nous vous demandons ailleurs dans le cours, et va vous demander de manipuler de nombreux concepts vus tout au long du cours.

Il vous est demandé que votre programme puisse gérer divers jeux de données construits sur les mêmes principes que les nôtres. Cela permettra à chacun des participants de tester son programme sur les données proposées par d'autres.

NIVEAUX DE PROJET

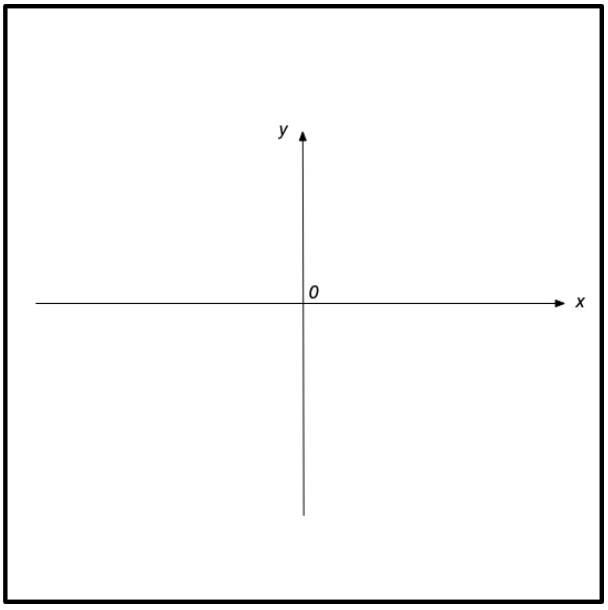
Dans le cadre de ce projet, 4 niveaux de jeux peuvent être réalisés correspondant à des difficultés croissantes :

- **Niveau 1 : construction et affichage du plan du château** Ce niveau consiste à tracer correctement le plan du château à partir du fichier de données. La gestion du jeu après affichage du plan ne fait pas partie de ce niveau.
- **Niveau 2 : en plus du niveau 1, la gestion des déplacements au clavier, sans portes ni objets à ramasser** Ce niveau consiste à gérer les déplacements d'un personnage dans le plan construit au niveau 1. Ici on suppose qu'aucun objet ni porte n'est présent dans le plan : leur gestion n'est pas prise en compte.
  - On déplace le personnage de case en case à l'aide des 4 flèches du clavier.
  - Le personnage ne doit pas pouvoir traverser les murs ni sortir du plan.
  - Si le personnage arrive sur la case quête / sortie du château, un message lui annonce qu'il a gagné.
  - **Option non demandée, si vous voulez aller plus loin :** les cases déjà parcourues par le personnage peuvent être affichées dans une couleur spécifique, de façon à ce que la trace de son parcours soit conservée.
- **Niveau 3 : niveaux 1 + 2 + gestion des objets à ramasser** Dans ce niveau, il y a dans le labyrinthe des objets que le joueur doit collecter. Outre le fichier contenant le plan, un second fichier de données contient la liste des objets et la case où chaque objet se trouve.
  - Les cases où se trouve un objet doivent apparaître dans une couleur spécifique.
  - Lorsque le joueur se déplace sur une case contenant un objet, un message lui signale qu'il a trouvé un objet, et ce dernier s'ajoute à l'inventaire affiché en permanence à côté du plan. La case où était l'objet devient vide, puisque l'objet a été ramassé.
  - **Option non demandée, si vous voulez aller plus loin :** on peut prévoir que la sortie du labyrinthe ne sera accessible qu'une fois tous les objets rassemblés.
- **Niveau 4 (escape game complet) : niveaux 1 + 2 + 3 + gestion des portes** Ce niveau consiste à ajouter dans la labyrinthe des portes que le joueur doit ouvrir en répondant à des questions. Le troisième fichier de données contient la liste des portes avec les questions et réponses associées.
  - Les cases où se trouve une porte doivent apparaître dans une couleur spécifique.
  - Lorsque le joueur tente d'accéder à une porte, la question associée lui est posée.
  - S'il répond correctement, la porte s'ouvre et il peut alors la franchir. La case de la porte apparaît alors comme une case vide.
  - S'il ne répond pas ou s'il donne une mauvaise réponse, la porte reste fermée et infranchissable.

1.1.3 En pratique

SYSTÈME DE COORDONNÉES

Le module turtle utilise un système de coordonnées dont l'origine `(0, 0)` est au centre le la fenêtre. L'axe des abscisses est orienté vers la droite, l'axe des ordonnées est orienté vers le haut.



Coordonnées des points dans une fenêtre turtle

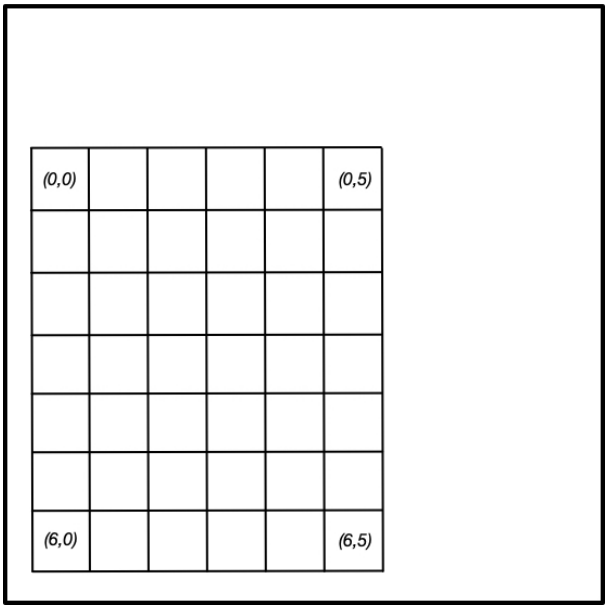
Dans ce qui suit, nous parlerons de *pixel turtle* pour parler de coordonnées dans la fenêtre turtle. Nous utiliserons les *pixels turtle* de `(-240, -240)` à `(240, 240)`.

Pour faciliter les discussions, nous vous demandons de numéroter les cases du plan du château :

— en numérotant les lignes de haut en bas et les colonnes de gauche à droite,

— en commençant la numérotation des lignes comme des colonnes à partir de 0 (convention habituelle de python).

La case (0, 0) est donc située en haut et à gauche du plan et la case `(0, 5)` est située 5 cases à sa droite. la case `(6, 5)` sera la 6e case depuis la gauche, sur la 7e ligne depuis le haut.



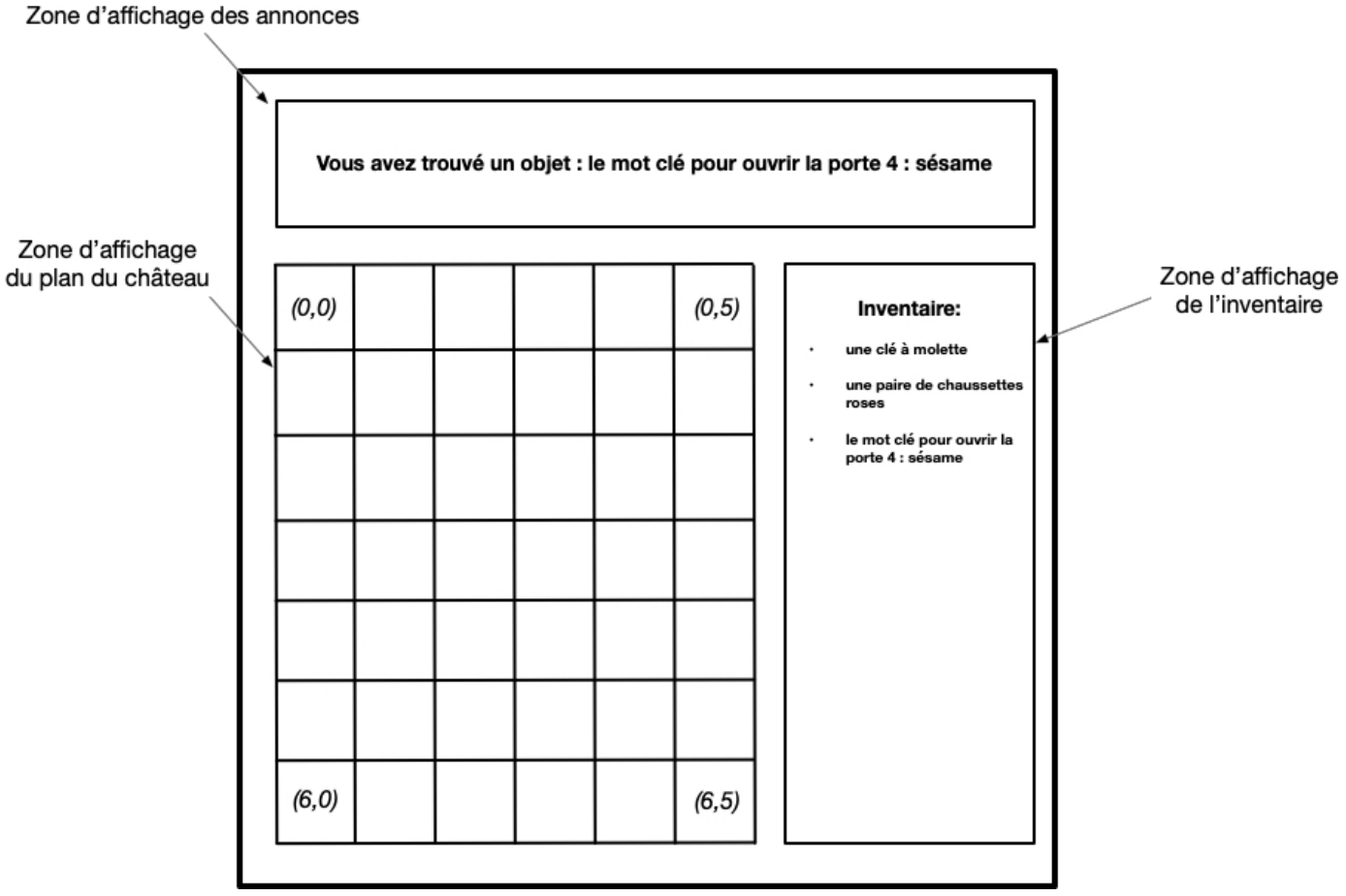
Coordonnées des carrés du plan du château

Point important : Lorsque vous calculerez les coordonnées nécessaires pour tracer une case, souvenez-vous que le numéro de ligne augmente quand l'ordonnée « y » (pour la fenêtre turtle) diminue.

OCCUPATION DES ZONES DE LA FENÊTRE TURTLE

Vous aurez besoin de distinguer 3 zones différentes dans la fenêtre turtle :

— un bandeau en haut pour l'affichage d'annonces, qui peut par exemple faire 40 *pixels turtle* de hauteur, que vous pourrez définir par les coordonnées en *pixels turtle* du début de la ligne de texte,
— en-dessous, une large zone à gauche pour l'affichage du plan, qu'il sera pratique de définir par les coordonnées de son coin inférieur gauche (abscisse et ordonnée minimales de la zone) et de son coin supérieur droit (abscisse et ordonnée maximales de la zone),
— la partie restant à droite sera la colonne d'affichage de l'inventaire, que vous pourrez définir par les coordonnées en *pixels turtle* du début de la première ligne de texte de l'inventaire.



Zones d'affichage du jeu dans la fenêtre turtle

DESCRIPTION DES FICHIERS DE DONNÉES

Les données correspondant à un jeu figurent dans 3 fichiers texte dont la structure est décrite ci-dessous.

a. Un fichier texte `plan_chateau.txt` qui contient les données du plan du château.
   Il comporte un certain nombre de lignes, toutes de la même longueur, représentant une ligne de cases du plan. Chaque ligne du fichier est une suite d'entiers séparés par des espaces, où chaque entier représente une case. Les valeurs de ces entiers codent chaque nature de case :
   — valeur 0 pour une case vide,
   — valeur 1 pour un mur (infranchissable),
   — valeur 2 pour la case de sortie/victoire,

— valeur 3 pour une porte qui sera franchissable en répondant à une question,
— valeur 4 pour une case contenant un objet à collecter.

b. Un fichier texte `objets.txt` qui contient une liste d'objets associés aux cases sur lesquelles on les trouve.
Chaque ligne sera du type :
`(x, y) , "objet"`
Ainsi, chaque ligne contient :
— un couple d'entiers positifs ou nuls (numéro de colonne, numéro de ligne) indiquant la case où se trouve l'objet,
— puis une virgule et une espace,
— puis une chaînes de caractères décrivant l'objet.
Par exemple :
`(12, 3), "un oreiller magique"`
signifiera que la case de coordonnées `(12, 3)` contient l'objet « un oreiller magique ».

c. Un fichier texte `portes.txt` qui contient une liste de questions/réponses associées aux portes.
Chaque ligne sera du type :
`(x, y) , ("question", "réponse")`
Ainsi, chaque ligne contient :
— un couple d'entiers positifs ou nuls (numéro de colonne, numéro de ligne) indiquant la case où se trouve la porte,
— puis une virgule et une espace,
— puis un couple de chaînes de caractères avec une question et une réponse.
Par exemple :
`(21, 12), ("Capitale de la Belgique ?", "Bruxelles")`
signifiera que pour franchir la porte située en case `(21, 12)`, il faudra répondre « Bruxelles » à la question « Capitale de la Belgique ? ».

CONSTANTES ET JEU DE DONNÉES TYPE

Pour vous aider à concevoir et à tester votre programme, un jeu de données comportant 3 fichiers types à téléchartger sur le format spécifié plus haut est donné ici :
— fichier `plan_chateau.txt` (https://upylab.ulb.ac.be/pub/Projets_MOOC/plan_chateau.txt)
— fichier `dico_objets.txt` (https://upylab.ulb.ac.be/pub/Projets_MOOC/dico_objets.txt)
— fichier ‘‘dico_portes.txt‘‘(https://upylab.ulb.ac.be/pub/Projets_MOOC/dico_portes.txt)
Il s'agit d'un escape game très simple avec des questions portant sur Python. Nous vous suggérons de placer ces 3 fichiers dans le même dossier que votre programme, pour pouvoir les appeler sans devoir décrire leur place dans l'arborescence.

Nous supposons aussi que la porte d'entrée du château est ouverte (elle est donc vue comme une case vide), qu'elle est en position `(0, 1)` et que le personnage s'y trouve initialement.

Un fichier `CONFIGS.py` ((https://upylab.ulb.ac.be/pub/Projets_MOOC/CONFIGS.py) vous est fourni : il donne les dimensions, couleurs et quelques constantes utilisées par notre programme de référence. Vous pouvez télécharger ce fichier et copier/coller ces définitions dans votre script ou mieux placer ce fichier `CONFIGS.py` dans le même répertoire que votre script et ajouter à ce dernier la ligne

```
from CONFIGS import *
```

dont l'effet en pratique sera le même qu'un copier/coller.

```
ZONE_PLAN_MINI = (-240, -240)  # Coin inférieur gauche de la zone d'affichage du plan
ZONE_PLAN_MAXI = (50, 200)  # Coin supérieur droit de la zone d'affichage du plan
POINT_AFFICHAGE_ANNONCES = (-240, 240)  # Point d'origine de l'affichage des annonces
POINT_AFFICHAGE_INVENTAIRE = (70, 210)  # Point d'origine de l'affichage de l'inventaire

# Les valeurs ci-dessous définissent les couleurs des cases du plan
COULEUR_CASES = 'white'
COULEUR_COULOIR = 'white'
COULEUR_MUR = 'grey'
```

(suite sur la page suivante)

(suite de la page précédente)

```
COULEUR_OBJECTIF = 'yellow'
COULEUR_PORTE = 'orange'
COULEUR_OBJET = 'green'
COULEUR_VUE = 'wheat'
COULEURS = [COULEUR_COULOIR, COULEUR_MUR, COULEUR_OBJECTIF, COULEUR_PORTE, \
            COULEUR_OBJET, COULEUR_VUE]
COULEUR_EXTERIEUR = 'white'

# Couleur et dimension du personnage
COULEUR_PERSONNAGE = 'red'
RATIO_PERSONNAGE = 0.9  # Rapport entre diamètre du personnage et dimension des cases
POSITION_DEPART = (0, 1)  # Porte d'entrée du château

# Désignation des fichiers de données à utiliser
fichier_plan = 'plan_chateau.txt'
fichier_questions = 'dico_questions.txt'
fichier_objets = 'dico_objets.txt'
```

Nous supposons donc que la fenêtre turtle est contenue dans un rectangle (de coin inférieur gauche en `(-240, -240)` et de coin supérieur droit `(240, 240)`)) et est composée de 3 zones :

— la zone *affichage du plan* (de coin inférieur gauche en `(-240, -240)` et de coin supérieur droit `(50, 200)`),
— la zone *annonces* avec les textes qui seront affichés en `(-240, 240)` (chaque annonce devra effacer la précédente),
— la zone *affichage de l'inventaire* (de coin **supérieur gauche** en `(70, 210)` et de coin **supérieur droit** en `(240, 210)`)

Pour faire simple, nous supposons également que le personnage démarre à la case de coordonnée (1,0)

1.2 Détails des phases du projet

Nous sommes conscients que ce projet, si vous voulez le faire dans sa totalité, est ambitieux surtout pour des programmeurs débutants.

Nous vous proposons donc, pour mener à bien votre projet, de le phaser dans sa réalisation ; de créer dans un premier temps (niveau 1) un embryon de projet, qui va grossir petit à petit (niveaux 2, 3) pour finalement englober la totalité de ce qui vous est demandé.

1.2.1 Niveau 1 : construction et affichage du plan du château

Le niveau 1 parle de lecture du plan à partir du fichier `plan_chateau.txt`, de la construction de la matrice correspondante pour stocker ce plan, et de son affichage grâce au module `turtle`.

Pour cela différentes fonctions doivent être programmées.

FONCTION DE LECTURE DU FICHIER CONTENANT LE PLAN

Vous devez écrire une première fonction `lire_matrice(fichier)`, qui recevra en argument le nom d'un fichier texte contenant le plan à tracer. Elle ouvrira ce fichier et renverra en sortie une matrice, c'est-à-dire une liste de listes, soit une liste dont chaque élément sera lui-même une liste représentant une ligne horizontale de cases du plan.

Prenons l'exemple de ce plan comprenant 4 lignes et 6 colonnes avec les cases vides blanches, des murs gris, un objet orange et une porte verte :

Plan d'un chateau très simple

Il est représenté par le fichier suivant, composé d'entiers séparés par des espaces et de retours à la ligne :

```
1 0 1 1 1 1
1 0 0 0 0 1
1 0 4 0 0 1
1 1 1 3 1 1
```

La fonction de lecture du fichier renverra la liste de listes suivante :

```
[[1, 0, 1, 1, 1, 1], [1, 0, 0, 0, 0, 1], [1, 0, 4, 0, 0, 1], [1, 1, 1, 3, 1, 1]]
```

FONCTIONS D'AFFICHAGE DU PLAN

Pour tracer le plan du château à partir de la matrice obtenue, vous écrirez une fonction `afficher_plan(matrice)` utilisant le module `turtle` ; cette fonction recevra en entrée la matrice telle que décrite ci-dessus.

Pour réaliser cette fonction, nous vous conseillons de suivre les étapes suivantes :

1. Commencer par une fonction `calculer_pas(matrice)` qui calcule la dimension à donner aux cases pour que le plan tienne dans la zone de la fenêtre `turtle` que vous avez définie : diviser la largeur et la hauteur de la zone en question par le nombre de cases qu'elle doit accueillir, retenir la plus faible de ces deux valeurs.

   Par exemple, pour une zone turtle d'affichage pour le plan de 290 (-240 à 50) de large et 440 (-240 à 200) de haut, si le plan fait 20 cases en largeur et 44 cases en hauteur, les carrés auront une taille de 10 pour ne pas sortir de la zone (ici le souci, si la taille est supérieure à 10, sera la hauteur du plan).
2. Définir une fonction `coordonnees(case, pas)` qui calcule les coordonnées en *pixels turtle* du coin inférieur gauche d'une case définie par ses coordonnées (numéros de ligne et de colonne). Souvenez-vous que les lignes sont numérotées de haut en bas, alors que l'axe des coordonnées verticales va de bas en haut. Par exemple : avec un château de 44 lignes (lignes 0 à 43), la case inférieure gauche du château (c'est-à-dire la case `(43, 0)`) pour une sous-fenêtre turtle allant de `(-240, -240)` à `(50, 200)` vaudra `(-240, -240)`.
3. Définir une fonction `tracer_carre(dimension)`, traçant un carré dont la dimension en *pixels turtle* est donnée en argument.
4. Définir une fonction `tracer_case(case, couleur, pas)`, recevant en arguments un couple de coordonnées en indice dans la matrice contenant le plan, une couleur, et un pas (taille d'un côté) et qui va appeler la fonction `tracer_carre` pour tracer un carré d'une certaine couleur et taille à un certain endroit.
5. Définir enfin une fonction `afficher_plan(matrice)`, qui va appeler la fonction `tracer_case` pour chaque ligne et chaque colonne du plan, par deux boucles imbriquées. Le principe est : pour chaque élément ligne de la matrice, pour chaque élément colonne de cet élément ligne, tracer une case à l'emplacement correspondant, dans une couleur correspondant à ce que dit la matrice.

CORPS DU PROGRAMME AU NIVEAU 1

Il restera alors à écrire la suite d'instructions qui va :

— fixer les valeurs nécessaires (noms des fichiers de données, couleurs des divers types de cases, points définissant les différentes zones de l'écran),
— appeler les différentes fonctions nécessaires pour créer la matrice et tracer le plan,

et à réaliser un premier script complet, contenant l'ensemble des éléments (docstrings, import, définitions des constantes, des fonctions et corps du programme) tel qu'expliqué dans le manuel des bonnes pratiques

1.2.2 Niveau 2 : gestion des déplacements

Bravo si vous avez terminé ce niveau 1. Vous pouvez passer au niveau 2. Ici, nous vous demandons d'ajouter à votre programme la gestion du déplacement d'un personnage.

Le personnage sera représenté par un petit rond (fonction prédéfinie turtle.dot). Il pourra se déplacer case par case dans les 4 directions (haut, bas, droite, gauche).

— Lorsque le personnage tentera un déplacement vers une case de mur ou vers l'extérieur du plan, rien ne se produira.
— Lorsqu'il tentera un déplacement vers une case vide, il avancera d'une case (en pratique, il s'agira de retracer la case de départ pour effacer le personnage, et de replacer ce dernier sur la case de destination).

Pour réaliser cette partie du code nous vous suggérons les parties suivantes :

FONCTION DE DÉPLACEMENT DU PERSONNAGE

Nous vous suggérons de définir une fonction principale de gestion des déplacements, qui sera assez simple pour le niveau 2, et que vous viendrez compléter si vous passez aux niveaux 3 (gestion des objets) et 4 (gestion des portes).

Pour le niveau 2, donc, nous avons besoin d'une fonction `deplacer(matrice, position, mouvement)` qui reçoit en arguments : - `matrice` : le plan du château, - `position` : un couple définissant la position où se trouve le personnage, - `mouvement` : un couple définissant le mouvement demandé par le joueur.

Pour l'instant (niveau 2) cette fonction aura les effets suivants :

— Si le mouvement souhaité fait sortir le personnage des limites du plan, rien ne se passera.
— Si le mouvement souhaité mène le personnage sur un mur, rien ne se passera.
— Si le plan que vous utilisez comprend des objets ou des portes, vous traiterez les objets comme des cases vides et les portes comme des cases de mur.

Note : Au niveau 2, le traitement des portes et des objets trouvés ne sont pas traités, et sont « vus » comme des murs et des cases vides. Ces éléments seront traités correctement aux niveaux 3 et 4, expliqués plus bas.

SAISIE DES DÉPLACEMENTS

Pour gérer la commande du personnage au clavier, nous allons utiliser de la programmation événementielle. Le principe est d'associer à un événement un certain traitement. Ici, il faudra associer au fait de pousser sur une touche du clavier (les flèches de votre clavier) un traitement qui gère le déplacement associé de votre personnage.

Pour ceci, nous allons utiliser des fonctions `turtle` qui n'ont pas été vues dans le MOOC, et nous vous donnons donc ci-dessous les portions de code nécessaires :

Pour faire avancer le personnage grâce aux touches du clavier

1. Nous allons utiliser :

— la fonction `turtle.listen()` pour demander à Python « d'écouter » ce qui se passe sur le clavier,
— la fonction `turtle.onkeypress()` pour associer une touche du clavier au déclenchement d'une fonction (notons que cette fonction ne peut avoir de paramètres),

— la fonction `turtle.mainloop()` pour placer le programme en position d'attente d'une action.

Vous placerez donc, dans la partie traitement (corps) de votre programme, le bloc suivant, qui associe aux 4 flèches du clavier (qui sont identifiées dans `turtle` avec les noms (chaînes de caractères) `"Left"` (flèche vers la gauche), `"Right"` (flèche vers la droite), `"Up"` (flèche vers le haut) et `"Down"` (flèche vers le bas)) les 4 fonctions respectives `deplacer_gauche`, `deplacer_droite`, `deplacer_haut` et `deplacer_bas`. Nous verrons au point suivant que, dans notre programme, il faudra définir chacune de ces 4 fonctions pour qu'elle déclenche le traitement requis chaque fois que la touche clavier correspondante est enfoncée. Ainsi, lorsqu'une des touches de flèche sera pressée (« onkeypress »), la fonction associée sera appelée.

```
turtle.listen()    # Déclenche l'écoute du clavier
turtle.onkeypress(deplacer_gauche, "Left")    # Associe à la touche Left une fonction␣
↪appelée deplacer_gauche
turtle.onkeypress(deplacer_droite, "Right")
turtle.onkeypress(deplacer_haut, "Up")
turtle.onkeypress(deplacer_bas, "Down")
turtle.mainloop()    # Place le programme en position d'attente d'une action du␣
↪joueur
```

2. Vous devrez alors définir les 4 fonctions (`deplacer_gauche()`, `deplacer_droite()`, `deplacer_haut()`, `deplacer_bas()`) que déclenchent les 4 flèches du clavier. Mais si le joueur appuie de manière répétée et rapide sur une touche, ou en continu en gardant le doigt enfoncé, la fonction associée risque d'être lancée plusieurs fois en parallèle, ce qui poserait de grandes difficultés. Aussi, nous allons encadrer le bloc d'instructions de la fonction par des instructions visant à désactiver provisoirement la touche concernée. Pour cela, nous allons associer provisoirement la touche à la valeur `None` qui représente une absence de valeur (et donc de traitement associé).

   Vous placerez donc dans votre code une définition de fonction du type qui suit, en ce qui concerne la flèche gauche, et des fonctions similaires pour les 3 autres flèches :

```
def deplacer_gauche():
    turtle.onkeypress(None, "Left")    # Désactive la touche Left
    ... # traitement associé à la flèche gauche appuyée par le joueur
    turtle.onkeypress(deplacer_gauche, "Left")    # Réassocie la touche Left à la␣
↪fonction deplacer_gauche
```

3. Il reste un point auquel il faut veiller sur le fonctionnement de la fonction `turtle.listen()`.

   Nous aurons besoin, pour le niveau 4, que le programme pose une question au joueur lorsque celui-ci veut franchir une porte. Pour cela, nous vous proposerons d'utiliser la fonction `turtle.textinput()` qui affiche une fenêtre de saisie puis attend que le joueur y entre une chaîne de caractères. Cette fenêtre de saisie lance en fait son propre `turtle.listen()` associé à la fenêtre de saisie, ce qui interrompt le `turtle.listen()` que nous avons lancé auparavant. Il faut donc, après l'utilisation de `turtle.textinput()`, relancer le `turtle.listen()`.

   En pratique, retenez ceci : si vous utilisez la fonction `turtle.textinput()`, vous devrez placer une nouvelle ligne `turtle.listen()` juste après, pour que le programme continue à détecter l'appui sur les touches du clavier.

```
reponse = turtle.textinput("Question", dico_portes[case][0])
turtle.listen()
```

1.2.3 Niveau 3 : collecte d'objets dans le labyrinthe

Pour le niveau 3, nous vous demandons d'ajouter au programme tel que rédigé au niveau 2 la gestion des objets, présents dans le château, que le personnage collecte quand il passe sur la case correspondante. Pour cela les éléments supplémentaires suivants devront être codés.

FONCTION DE LECTURE DES FICHIERS CONTENANT LES OBJETS

Vous aurez besoin d'une fonction `creer_dictionnaire_des_objets(fichier_des_objets)` créant un dictionnaire d'objets à partir du fichier correspondant, présenté plus haut. Cette fonction recevra en argument le nom du

`fichier_des_objets`, et renverra un dictionnaire comportant :
- en clefs les couples `(colonne, ligne)` désignant les cases où se trouvent les objets,
- en valeurs les chaînes de caractères correspondant aux objets.

Ainsi le fichier d'objets suivant :

```
(12, 3), "un oreiller"
(3, 15), "une paire de ciseaux"
```

donnera le dictionnaire :

```
{(12,3): "un oreiller", (3, 15): "une paire de ciseaux"}
```

Note :

La lecture des données reçues du fichier des objets (et on verra plus loin, aussi le fichiers des portes) peut être facilitée par l'utilisation de verbe `eval`. Par exemple, ayant une chaîne de caractères `chaine = '(1, 2), "bonjour"'`, après l'instruction :

```
a, b = eval(chaine)
```

`a` vaudra `(1, 2)` et `b` vaudra `"bonjour"`.

COMPLÉMENT À LA FONCTION DE DÉPLACEMENT DU PERSONNAGE

Au niveau 2, vous avez défini une fonction gérant le déplacement, qui examine si le personnage est envoyé vers une case vide ou vers un mur. Vous devez maintenant ajouter un autre cas à cette fonction, celui où le personnage est envoyé vers une case contenant un objet, et donc définir une fonction `ramasser_objet` :
- L'objet disparaîtra de la case (à la fois dans le plan et à l'affichage), qui devra donc prendre la couleur des cases vides.
- Le personnage avancera sur la case demandée.
- Une annonce du type « Vous avez trouvé : une clef à molette » s'affichera dans le bandeau d'affichage des annonces.
- L'objet s'ajoutera à l'inventaire des objets collectés affiché dans la colonne d'affichage de l'inventaire.

Pour cela, vous aurez sans doute besoin de passer de nouveaux paramètres à la fonction gérant le déplacement : la matrice contenant le plan (puisque vous serez amené à modifier la nature de la case contenant l'objet lorsque celui-ci sera ramassé) et l'ensemble contenant l'inventaire des objets ramassés (puisque l'objet sera ajouté à cet ensemble).

1.2.4 Niveau 4 : Le jeu escape game complet avec questions-réponses

Le niveau 4 demande de réaliser le jeu complet tel qu'annoncé en début d'énoncé. Pour cela les éléments supplémentaires suivants devront être codés.

FONCTION DE LECTURE DES FICHIERS CONTENANT LES QUESTIONS/RÉPONSES

Vous aurez besoin de lire le fichier contenant les questions-réponses associées aux portes. Cela peut être réalisé par la fonction de lecture du fichier des objets (`creer_dictionnaire_des_objets`) que vous aurez écrite au niveau précédent, puisque la structure des fichiers est presque la même.

Ainsi le fichier de questions/réponses suivant :

```
(12, 3) ("3 + 2 = ?", "5")
(3, 15) ("Quel était le prénom d'Henri IV ?", "Henri")
```

donnera le dictionnaire :

```
{(12,3): ("3 + 2 = ?", "5"),
 (3, 15): ("Quel était le prénom d'Henri IV ?", "Henri")}
```

COMPLÉMENT À LA FONCTION DE DÉPLACEMENT DU PERSONNAGE

Vous allez maintenant devoir ajouter le cas où la case de destination souhaitée est une porte. Vous aurez besoin d'une fonction `poser_question(matrice, case, mouvement)` pour :

- afficher dans le bandeau d'annonces `Cette porte est fermée.`
- poser au joueur la question correspondant à l'emplacement de la porte et saisir sa réponse,
- si la réponse est bonne, remplacer la porte par une case vide, afficher dans le bandeau d'annonce que la porte s'ouvre, et avancer le personnage,
- si la réponse est mauvaise, l'annoncer et ne pas déplacer le personnage.

Pour poser une question, vous pouvez utiliser la fonction `turtle.textinput()`. Elle prend en argument 2 chaînes de caractères, une qui sera le titre de la fenêtre d'input (par exemple « Question ») et une autre qui sera le texte affiché dans la fenêtre d'input (la question associée à la porte).

Comme cela a été signalé plus haut, utiliser la fonction `turtle.textinput()` interrompt l'écoute du clavier déclenchée par `turtle.listen()`, et vous devrez placer sur la ligne suivante une nouvelle instruction `turtle.listen()` pour recommencer à surveiller le clavier.

1.2.5 Remise et évaluation du projet

CONSIGNES À RESPECTER POUR LE PROJET

Consignes sur le contenu du projet

Les consignes décrites dans les sous-sections précédentes sont à respecter scrupuleusement ; relisez-les attentivement avant la remise !

Echéances

Consultez le calendrier pour connaître les dates limites

- de remise de votre projet ainsi que
- celle du travail d'évaluation des projets de vos pairs.

Attention

Si vous avez choisi de réaliser votre projet pour la session d'automne, vous devez respecter les échéances de cette session (soumission, évaluation des projets des pairs, seconde soumission, auto-évaluation). Dans le cas contraire, la session de printemps sera votre dernière possibilité, de nouveau en respectant les différentes échéances.

Notez que si vous n'avez pas soumis à temps votre projet pour la première phase d'évaluation par les pairs, il vous sera quand même possible de soumettre votre travail pour la phase suivante d'auto-évaluation, mais alors vous vous noterez sur 24 points et non sur les 48 points possibles pour le projet.

Modalités de remise

La procédure de remise est expliquée à la section suivante.

MAIS COMMENT ÉVALUER UN PROJET ?

De façon binaire nous pourrions dire qu'un programme informatique est bon s'il fait ce qu'on lui demande et sinon il est incorrect. Ce serait un peu court surtout après avoir parlé de toutes les règles de bonnes pratiques que nous avons vues.

En fait, un projet tel que le projet Château est évalué selon plusieurs critères de correction (programme correct, qui fait bien son travail) et de style (code qui suit les règles de bonnes pratiques).

Donnons ici des recommandations pour évaluer un projet de programmation niveau débutant.

Principes de base

La réalisation d'un projet informatique comprend plusieurs aspects qui font l'objet de choix qui doivent généralement être justifiés, ou de mise en application, généralement sous forme de programme structuré.

Tout d'abord, il est important que l'évaluation soit le résultat d'une notation

— positive : qui consiste à donner des points quand certains aspects sont présents dans la solution, et
— négative : qui consiste à enlever des points sur certains aspects qui sont absents ou incorrects.

Il convient aussi in fine d'avoir une évaluation globale où un projet qui *fonctionne* se voit *naturellement* attribuer des points même si plusieurs problèmes existent dans la réalisation.

Le mieux pour obtenir une telle note qui est le fruit d'une évaluation **négative et positive**, est de découper la note finale en sous notes, qui chacune évalue un aspect, qui de même fait l'objet d'une évaluation *négative et positive* avec évaluation globale.

Proposition

Ci-dessous la découpe de la note à donner au projet, que nous vous demandons d'appliquer dans vos évaluations pour obtenir une notation uniforme pour tous. Le nombre de points pour chaque catégorie est proposée pour une note finale sur 24 points. Comme il s'agit d'un projet pour débutant, les structures de données et les méthodes pour résoudre les problèmes sont données et ne nécessitent pas de recherche pour l'apprenant ; aucune note n'est donc donnée pour ces deux aspects.

Grille de notation du projet (sur 24 points)

— **Découpe (4 points)** [La découpe du code global avec commentaires] initiaux, et respect de l'ordre des parties : import des modules externes, suivi des définitions des constantes globales, suivi des définitions des fonctions et enfin, code global.
  — 4 points si les quatre parties sont présentes et l'ordre est respecté ;
  — 3 points si une des quatre parties manque ou est dans le mauvais ordre ;
  — 2 points si deux des quatre parties manquent ou sont dans le mauvais ordre ;
  — 1 points si trois des quatre parties manquent ou sont dans le mauvais ordre ;
  — 0 point si rien n'est satisfaisant.
— **Structuration en fonctions (3 points)** [La structuration en] fonctions de bonne taille (25 lignes maximum) et cohérentes (pas de fonctions définies mais non appelées par exemple) est réalisée :
  — 3 points si la structuration est satisfaisante ;
  — 2 points si des fonctions sont définies, mais certaines sont incohérentes soit sont trop longues ;
  — 1 points si des fonctions sont définies, mais certaines sont incohérentes et d'autres sont trop longues ;
  — 0 point si aucune fonction n'est définie ou une certaine structuration en fonction existe mais ne respecte pas du tout les règles de bonnes pratiques.
— **Bonnes pratiques (4 points)** [Les bonnes pratiques sont] respectées en matière d'utilisation des noms des constantes, variables, fonctions. . . , de l'indentation. . . :
  — 4 points si toutes les bonnes pratiques sont respectées ;
  — 3 points si les noms des constantes, variables et fonctions respectent les bonnes pratiques, mais l'indentation (4 caractères par incrément) n'est pas respectée ;
  — 2 points si les règles sont globalement repectées à quelques exceptions près à la fois au niveau indentations et noms de constantes, variables ou fonctions ;
  — 0 point si les règles de bonnes pratiques en matière d'encodage ne sont globalement pas respectées.
— **Commentaires (4 points)** [Commentaires et docstrings sont bien] présents : le docstring initial possède les informations suivantes : identité de l'auteur, date, ce que fait le programme, les fonctions possèdent un docstring décrivant les paramètres, ce que fait la fonction aux paramètres (par défaut, ils ne sont pas modifiés) et le résultat de la fonction, des commentaires pertinents existent quand c'est utile à la compréhension
  — 4 points si commentaires et docstrings sont présent et respectent les règles de bonnes pratiques ;
  — 3 points si un des éléments du docstring initial ou des commentaires est manquant ou non satisfaisant ;
  — 2 points si le docstring initial et les commentaires sont manquants ou non satisfaisants, mais que les fonctions ont des docstrings corrects ;
  — 0 point si les trois éléments docstring initial, commentaires et docstrings de fonctions sont manquants ou non satisfaisants.
— **Consignes (2 points)** : (voir énoncé du projet)
  — 2 points si toutes les consignes du projet sont respectées (niveau 4 de l'énoncé atteint) ;
  — 1 point si une bonne partie des consignes est respectée (niveau 2 ou 3 atteint) ;
  — 0 point si peu de consignes sont respectées (le niveau 1 n'a pas été dépassé).
— **Résultat (7 points)** : Le programme fonctionne correctement.
  — 7 points si le projet complet fonctionne toujours correctement (niveau 4 de l'énoncé) ;
  — 6 points si le programme donne les bons résultats à quelques exceptions prêt (niveau 3 atteint) ;
  — 4 points si certaines parties du projet ne sont pas (bien) réalisées (niveau 2 atteint) ;
  — 2 points si une partie du projet seulement est réalisée (niveau 1 atteint) ;
  — 0 point si le programme ne s'exécute pas sans échec ou si aucune partie ne fonctionne.

Explication sur l'évaluation

En plus des points attribués, il est essentiel pour l'évaluateur d'expliquer pourquoi telle ou telle note a été donnée, pour que l'auteur du projet puisse comprendre les notes données et puisse s'améliorer les fois suivantes.

ET COMMENT ALLEZ-VOUS ÊTRE ÉVALUÉ SUR VOTRE PROJET ?

Dans le cadre de ce cours, comme annoncé, l'évaluation de votre projet, se fera en deux temps :
- d'une part par une évaluation par les pairs
- d'autre part, par une auto-évaluation de votre projet après éventuelles améliorations.

En pratique, la remise du projet et son évaluation seront faites en **6 étapes** :

1. Après l'avoir réalisé, vous soumettrez votre projet.
2. Vous devrez évaluer trois projets d'autres étudiants choisis au hasard.
3. Vous recevrez les notes et commentaires sur votre propre projet venant d'au moins deux autres de vos pairs (deux autres apprenants).
4. À la lumière des commentaires donnés par vos pairs, vous corrigerez et soumettrez une version améliorée de votre projet.
5. Vous réaliserez une auto-évaluation, c'est-à-dire évaluerez votre propre projet.

Le document consignes pour la notation, téléchargeable ici reprend les explications et détaille chaque notation et demande de commenter chaque note donnée.

La note finale de votre travail sera la somme des évaluations par les pairs (24 points) et de votre auto-évaluation (24 points).

Les détails pratiques sur les évaluations seront donnés plus loin.

Temps et délais pour réaliser le projet

L'estimation du temps moyen que va vous mettre le projet Château est très variable. La matière vue jusqu'ici suffit pour sa réalisation. Vous avez plusieurs jours pour sa remise. Libre à vous de mener en parallèle la suite du module 6 avec la réalisation du projet ou plutôt de postposer la suite de l'apprentissage du module 6 et de vous atteler dès maintenant au projet.

DÉTAILS PRATIQUES SUR L'ÉVALUATION

Les paragraphes qui suivent vous permettent de réaliser :
- d'une part l'évaluation des projets de trois de vos pairs
- et d'autre part, quand vous aurez reçu deux évaluations de votre propre projet (d'autres pairs), une auto-évaluation de votre projet.

DERNIERS CONSEILS AVANT DE FAIRE LES ÉVALUATIONS

Vous et tous les autres apprenants débutez en programmation. L'évaluation par les pairs avec les commentaires qui l'accompagnent ainsi que l'auto-évaluation peuvent être très utiles pour votre apprentissage.

Pour que cela soit le cas, lors des différentes phases de notation, essayez d'être :
- objectif dans vos évaluations,
- clair et précis dans vos commentaires,
- mais toujours courtois et bienveillant

comme vous voudriez que les autres le soient pour vous-même.

Bon travail !

1.3 Phase d'évaluation par les pairs

1.3.1 Phase d'évaluation par les pairs

Si vous comptez réaliser votre projet pour la session d'automne, cette section et la section suivante contient les procédures pour soumettre votre, projet, réaliser et rapporter vos évaluations sur les projets des pairs que nous vous demandons d'effectuer soumettre à nouveau votre projet et finalement l'auto-évaluer.

MENU DE CETTE SOUS-SECTION

Cette sous-section contient trois parties qui vont vous demander des actions :

1) D'abord vous allez soumettre votre projet éventuellement accompagné d'un lien vers l'image d'un résultat de votre programme.
2) Ensuite vous allez évaluer le projet de trois de vos pairs (ou plus si demandé et si vous le voulez bien) et remettre des notes et vos commentaires les plus précis possible.
3) Enfin dans cette partie, vous allez recevoir la note des évaluations d'au moins deux de vos pairs avec les détails des notes et les commentaires associés.

24 points seront en jeu dans cette partie.

La suite (resoumission de votre projet éventuellement amendé et auto-évaluation) est l'objet de la sous-section qui suivra.

Note : Voir le formulaire d'évaluation du projet par les pairs dans le cours en ligne

---

..only : : html

Attention à l'horaire

Certains d'entre vous retiennent la date d'échéance indiquée dans la barre latérale sous l'intitulé de cette section. **Attention, regardez bien le calendrier. Cette date correspond à la fin de l'évaluation par les pairs. L'échéance pour la soumission de votre projet est fixée 15 jours avant cette date !**

1.4 Phase d'auto-évaluation du projet

1.4.1 Phase d'auto-évaluation

MENU DE CETTE SOUS-SECTION

Cette sous-section contient trois parties qui vont vous demander des actions :

1) D'abord vous allez soumettre une nouvelle fois votre projet éventuellement amendé suite à la lecture des notes et commentaires de vos pairs à nouveau éventuellement accompagné d'un lien vers l'image d'un résultat de votre programme.
2) Ensuite vous allez évaluer votre propre projet que vous venez de soumettre et remettre des notes et vos commentaires sur ce dernier.
3) Enfin, vous allez recevoir la note de votre auto-évaluation.

À nouveau, 24 points seront en jeu dans cette partie, ce qui totalisera 48 points pour la réalisation de votre projet.

Note : Voir le formulaire d'auto-évaluation du projet dans le cours en ligne

---

CHAPITRE 2

Projet donné lors des trois premières sessions du MOOC Apprendre à coder avec Python

2.1 Le projet Vasarely

2.1.1 Énoncé du projet

LE PROJET : INFORMATION PRÉLIMINAIRE

Nous donnons ici l'énoncé du projet que nous avons proposé dans le cadre du cours en ligne (FUN-MOOC) « Apprendre à coder avec Python », ainsi que les procédures d'évaluation proposées aux apprenants dans le cadre d'une évaluation par les pairs.

Le projet couvre les instructions et fonctions Python ainsi que le BA-ba des séquences (chaînes de caractères et tuples) ce qui correspond à la matière du MOOC jusque et y compris la section 5.1.

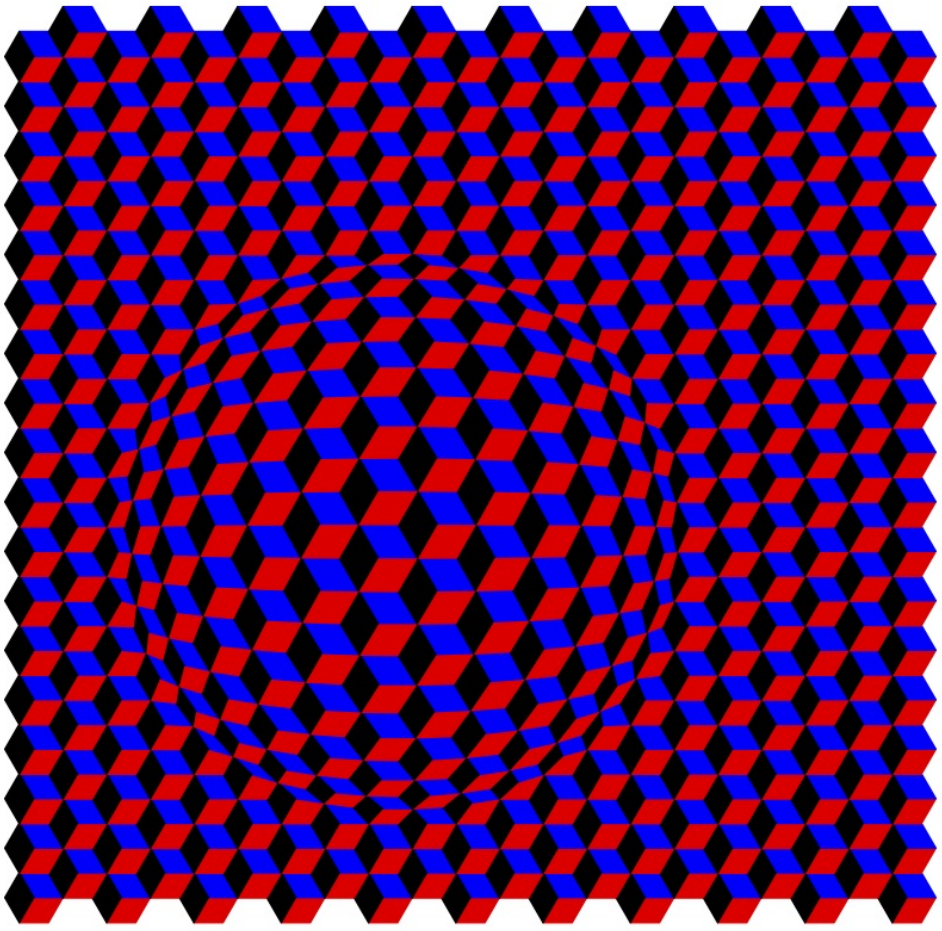
PROJET VASARELY (OP ART)

Dans ce qui suit, nous donnons l'énoncé de ce projet, baptisé « projet Vasarely » en référence à l'artiste Victor Vasarely qui a popularisé cette tendance. (Pour en savoir plus sur cet artiste, n'hésitez pas à consulter la page dédiée suivante : Victor Vasarely).

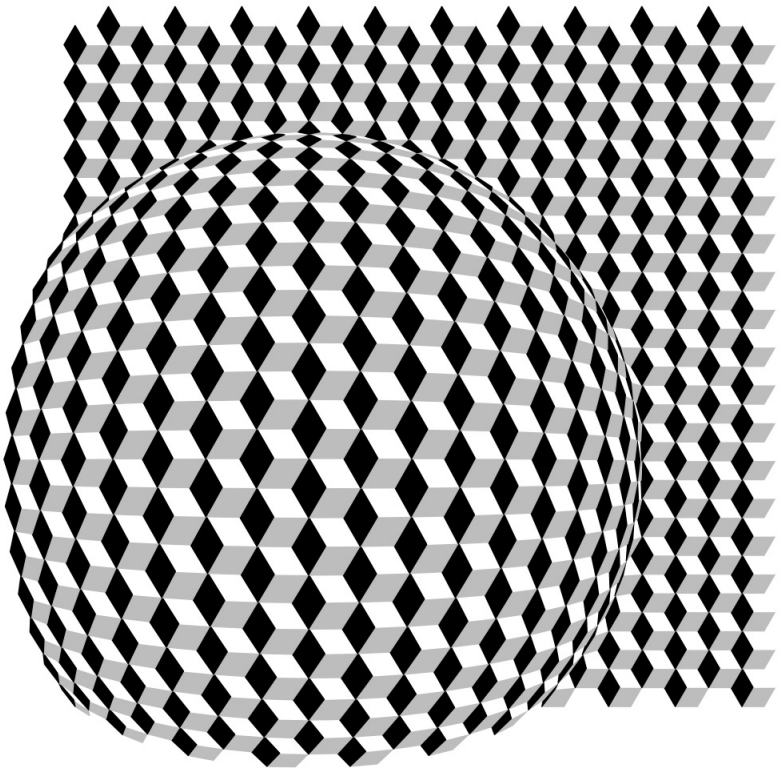
ÉNONCÉ DU PROJET VASARELY

Brève description du projet

Il vous est demandé de réaliser un programme en Python produisant des tableaux d'art optique comme représentés par les figures ci-dessous avec différentes couleurs et déformations :



Exemple de pavage déformé (bleu, rouge, noir)



Autre exemple de pavage déformé (noir, blanc, gris)

Ces tableaux représentent des pavages hexagonaux, vus d'en haut, formés avec des losanges de couleurs différentes, déformés par une boule. Votre programme utilisera le module `turtle`, présenté au module 2 de ce cours, pour peindre vos tableaux.

2.1.2 En pratique

LES CONSIGNES POUR VOTRE PROJET

Nous travaillons avec des coordonnées et distances telles que manipulées par les différentes fonctions de `turtle` (`up`, `begin_fill`, `goto`, ...). En particulier nous demandons de tracer, avec turtle, la vue de dessus d'une figure en 3 dimensions et donc de manipuler tantôt des coordonnées en 3 dimensions sous forme de triplet (x, y, z), tantôt en 2 dimensions sous forme de couple (x, y).

Nous vous demandons de réaliser quatre étapes.

ÉTAPE 1

En supposant que turtle est déjà importé, nous vous demandons d'écrire une fonction

```
hexagone(point, longueur, col, centre, rayon)
```

qui reçoit :

- le point `point` sous forme d'un triple (tuple de trois composantes) donnant la valeur des trois coordonnées du centre avant déformation de l'hexagone à peindre,
- la distance (avant déformation) `longueur` entre le centre et n'importe quel sommet de l'hexagone,
- le tuple `col` contenant les trois couleurs `(col1, col2, col3)` qui vont être utilisées pour dessiner les hexagones,
- le point `centre` sous forme de triple `(c_x, c_y, c_z)` qui donne le centre de la sphère de déformation,
- et le `rayon` de la sphère de déformation.

La fonction peint un hexagone déformé en traçant des lignes droites entre le centre et les extrémités dont la position est calculée avec la fonction `deformation`.

`col1` représente la couleur du pavé Nord-Est (en haut à droite), `col2` représente la couleur du pavé Ouest (à gauche) et `col3` représente la couleur du pavé Sud-Est (en bas à droite).

Note : Pour tester votre fonction, vous pouvez dans un premier temps définir la fonction `deformation` comme suit :

```
def deformation(point, centre, rayon):
    """renvoie le point sans le modifier"""
    return point
```

qui renvoie le point `point` sans le modifier.

ÉTAPE 2

Écrire ou télécharger la fonction `deformation(point, centre, rayon)` avec :
- `point` (`tuple` à 3 composantes `(p_x, p_y, p_z)`) : le point à déformer
- `centre` (`tuple (c_x, c_y, c_z)`) : le point central de la sphère de déformation
- `rayon` (`float`) : le rayon de la sphère de déformation

et sachant qu'elle renvoie le point `(x2, y2, z2)` après déformation.

Si vous décidez d'utiliser notre version de la fonction deformation, vous pouvez la copier / coller dans votre code à l'endroit adéquat (voir règles de bonnes pratiques).

```
from math import pi, sin, cos, sqrt, acos, asin, atan2

def deformation(p, centre, rayon):
    """ Calcul des coordonnées d'un point suite à la déformation engendrée par la sphère␣
↪émergeante
        Entrées :
          p : coordonnées (x, y, z) du point du dalage à tracer (z = 0) AVANT déformation
          centre : coordonnées (X0, Y0, Z0) du centre de la sphère
          rayon : rayon de la sphère
        Sorties : coordonnées (xprim, yprim, zprim) du point du dallage à tracer APRÈS␣
↪déformation
    """
    x, y, z = p
    xprim, yprim, zprim = x, y, z
    xc, yc, zc = centre
    if rayon**2 > zc**2:
        zc = zc if zc <= 0 else -zc
        r = sqrt(
            (x - xc) ** 2 + (y - yc) ** 2)                    # distance horizontale depuis le␣
↪point à dessiner jusqu'à l'axe de la sphère
        rayon_emerge = sqrt(rayon ** 2 - zc ** 2)             # rayon de la partie émergée de la␣
↪sphère
        rprim = rayon * sin(acos(-zc / rayon) * r / rayon_emerge)
        if 0 < r <= rayon_emerge:                          # calcul de la déformation dans les autres cas
            xprim = xc + (x - xc) * rprim / r                 # les nouvelles coordonnées sont␣
↪proportionnelles aux anciennes
            yprim = yc + (y - yc) * rprim / r
        if r <= rayon_emerge:
            beta = asin(rprim / rayon)
            zprim = zc + rayon * cos(beta)
            if centre[2] > 0:
                zprim = -zprim
```

(suite sur la page suivante)

(suite de la page précédente)

```
    return (xprim, yprim, zprim)

if __name__ == "__main__": # code de test
    for i in range(-150,150,50):
        for j in range(-150,150,50):
            print(deformation((i,j,0), (0,0,100), 100))
        print()
```

Ou vous pouvez importer le fichier `module_deformation.py` en cliquant sur le lien ci-dessous pour ensuite ouvrir le fichier `module_deformation.py` et copier / coller la définition de la fonction `deformation` dans votre code du projet Vasarely.

Note : Une alternative au copier / coller serait de faire un import de la fonction.

Pour ceux qui veulent essayer de définir `deformation` eux-mêmes (**attention : la théorie et le code sont beaucoup plus difficiles** que tout ce que nous avons fait jusqu'à présent) :

voici les explications de base en cliquant ici

et voici quelques conseils ou remarques :

— Vous pouvez utiliser la librairie `math`.
— Les calculs se font sur des points en trois dimensions.
— Le pavage avant déformation se trouve sur le plan avec z = 0.

Note : Notons que nous nous simplifions le problème lors du tracé des hexagones déformés : en effet après calculs des déplacements des points extrémités grâce à la fonction `deformation`, le programme tracera des lignes droites (et non pas des lignes déformées) entre ces extrémités. Ne vous en faites pas : l'effet final n'en est que peu affecté.

Conseil : pour tracer les hexagones utilisez la méthode `goto(x,y)` de `turtle` plutôt que `forward`.

Exemple d'hexagone avant et après transformation

Exemple de pavé hexagonal non déformé

hexagone de centre (0, 0, 0) de longueur d'arête avant déformation 30 avec déformation de centre (-50, -50, 0) et de rayon 90 (en rose, les contours du pavé avant déformation)

ÉTAPE 3

Écrire une fonction :

`pavage(inf_gauche, sup_droit, longueur, col, centre, rayon)`

qui reçoit :

- des valeurs entières `inf_gauche`, `sup_droit` précisant les coordonnées d'une fenêtre dont le coin inférieur gauche est `(inf_gauche, inf_gauche)` et le coin supérieur droit est `(sup_droit, sup_droit)` sachant que l'on ne prend en compte que les axes x et y, la hauteur du pavage avant transformation étant égale à 0 ;
- la `longueur` entre le centre et n'importe quel sommet de chaque hexagone avant déformation ;
- les trois couleurs sous forme de triple `col = (col1, col2, col3)` ;
- le `centre` de la déformation utile pour la fonction `deformation` (`centre = (x_c, y_c, z_c)` ;
- le `rayon` de la sphère de déformation utile pour la fonction `deformation`.

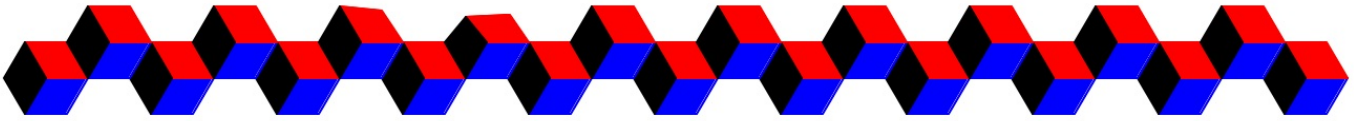
La fonction `pavage` peint les hexagones déformés dont les centres, **avant déformation**, se trouvent à l'intérieur de la fenêtre (bords inclus) avec l'hexagone en bas à gauche, avant déformation, centré sur le point `(inf_gauche, inf_gauche)`. Pour cela, elle utilise la fonction `hexagone`.

La séquence de couleurs est la même pour tous les pavés du pavage (tous les losanges en haut à droite du pavé sont de `col1`, à gauche, de `col2` et en bas à droite, de `col3`).

Que vous définissiez vous-même la fonction `deformation` ou que vous preniez la nôtre, de toute façon, n'oubliez pas que `turtle` ne tient compte que des deux premières dimensions, la valeur de z des points n'étant pas envoyée par exemple aux `goto(x,y)`.

Les figures suivantes montrent comment votre fonction pavage peut agencer les dessins des pavés, ligne par ligne.

Première ligne inférieure



Deux premières lignes inférieures peintes

Les couleurs possibles sont données par la spécification de couleur Tk, voir : Couleurs et Couleurs2.

ÉTAPE 4

Enfin, écrire un code principal qui demande à l'utilisateur les paramètres de l'exécution et qui appelle la fonction `pavage` pour réaliser le résultat.

Les paramètres seront donnés comme suit :

- `inf_gauche` : (valeur entière) donnant les coordonnées `(inf_gauche, inf_gauche)` du bord inférieur gauche de la fenêtre de visualisation ;
- `sup_droit` : (valeur entière) donnant les coordonnées `(sup_droit, sup_droit)` du bord supérieur droit de la fenêtre de visualisation ;
- `longueur` : (valeur entière) longueur d'un segment de pavé (avant déformation) ;
- `col` : (trois chaîne de caractères) donnant les trois couleurs des pavés ;

— `centre` : (trois entiers) donnant les coordonnées du centre de la sphère déformante ;
— `r` : (entier) donnant le rayon de la sphère déformante.

Par exemple les entrées de votre programme seront :

```
-305
305
20
blue
black
red
-50
-50
-50
200
```

Contrairement aux exercices UpyLaB, ici vos instructions *input*' peuvent afficher le texte qu'elles désirent ; par exemple une exécution pourrait donner l'affichage sur la console suivant :

```
Coin inférieur gauche (val, val) : -305
Coin supérieur drout  (val, val) : 305
Longueur d'une arrête : 20
Couleur 1 : blue
Couleur 2 : black
Couleur 3 : red
Abscisse du centre du cercle : -50
Ordonnée du centre du cercle : -50
Hauteur du centre du cercle : -50
Rayon du cercle : 200
```

Pour vous permettre de montrer l'image produite par votre programme, après la réalisation du travail, soit vous capturez l'image résultat, soit votre programme demandera à `turtle` de sauver le résultat dans un fichier. Pour cela, votre programme pourra exécuter en fin de programme les deux instructions :

```
turtle.getcanvas().postscript(file="pavage.eps")
turtle.done()
```

où ici le nom du fichier est `"pavage.eps"`. Notez que le seul format de fichier possible pour turtle est postscript (suffixe eps). Si, après l'exécution du programme, vous désirez un autre format pour le fichier résultat (jpeg ou pdf par exemple) il vous suffira de faire traduire le résultat, par exemple en utilisant une traduction en ligne (avec votre navigateur web, cherchez **online translation eps to jpg** par exemple).

Au terme de votre projet, si vous le désirez, un exemple de résultat de votre code, avec les couleurs et déformations que vous désirez, pourra être publié et visible par tous. Utilisez le **forum du Projet** pour toute question concernant l'énoncé.

EXEMPLES DE RÉSULTATS

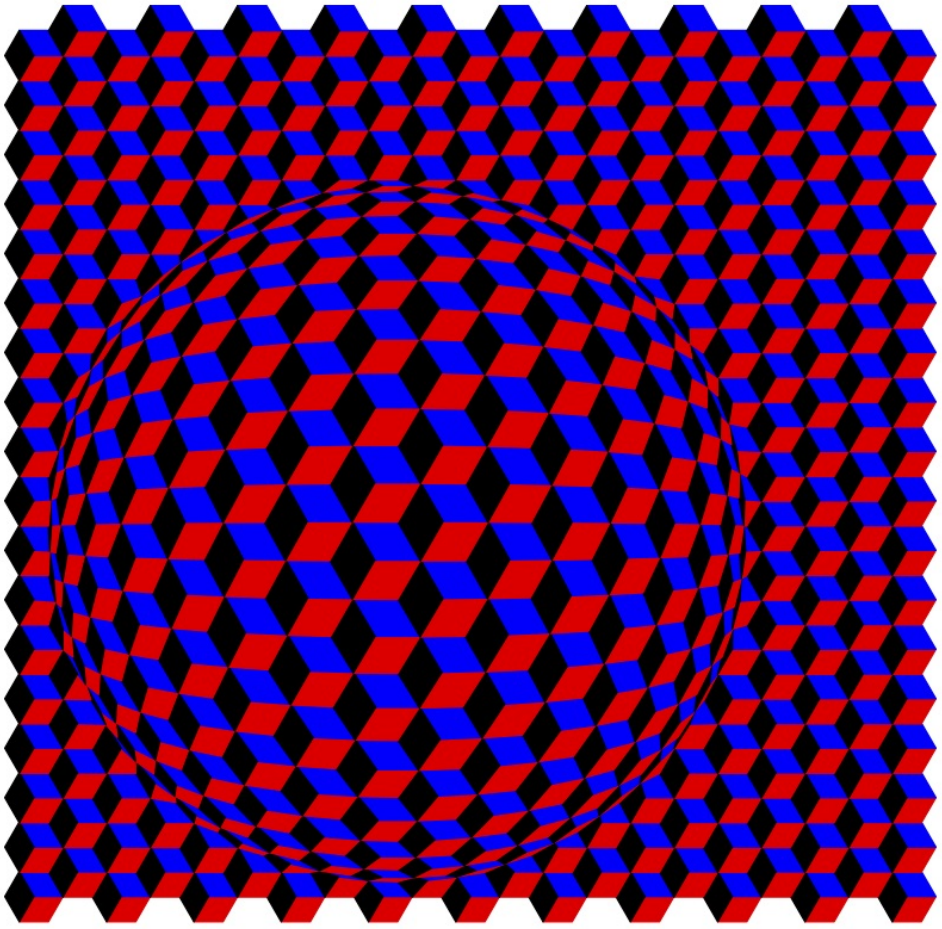
Le tableau au début de l'énoncé a été généré avec les paramètres
— `inf_gauche = -305`
— `sup_droit = 305`
— `longueur = 20`
— `col1 = 'blue'`
— `col2 = 'black'`
— `col3 = 'red'`
— `centre = (-50, -50, -50)`
— `r = 200`

Le second exemple en début d'énoncé est produit avec les mêmes paramètres, mis à part :
— `(col1, col2, col3) = ('white', 'black', 'grey')`
— `centre = (-100, -100, 0)` et

— `rayon = 300`

L'exemple ci-dessous a les mêmes paramètres que celui donné au début, mis à part :

— `centre = (-50, -50, 0)` et
— `rayon = 240`



pavage avec `centre = (-50, -50, 0)` et `rayon = 240`

2.1.3 Remise et évaluation du projet

MAIS COMMENT ÉVALUER UN PROJET ?

De façon binaire nous pourrions dire qu'un programme informatique est bon s'il fait ce qu'on lui demande et sinon il est incorrect. Ce serait un peu court surtout après avoir parlé de toutes les règles de bonnes pratiques que nous avons vues.

En fait, un projet tel que le projet Vasarely est évalué selon plusieurs critères de correction (programme correct, qui fait bien son travail) et de style (code qui suit les règles de bonnes pratiques).

Donnons ici des recommandations pour évaluer un projet de programmation niveau débutant.

Principes de base

La réalisation d'un projet informatique comprend plusieurs aspects qui font l'objet de choix qui doivent généralement être justifiés, ou de mise en application, généralement sous forme de programme structuré.

Tout d'abord, il est important que l'évaluation soit le résultat d'une notation

— positive : qui consiste à donner des points quand certains aspects sont présents dans la solution, et
— négative : qui consiste à enlever des points sur certains aspects qui sont absents ou incorrects.

Il convient aussi in fine d'avoir une évaluation globale où un projet qui *fonctionne* se voit *naturellement* attribuer des points même si plusieurs problèmes existent dans la réalisation.

Le mieux pour obtenir une telle note qui est le fruit d'une évaluation **négative et positive**, est de découper la note finale en sous notes, qui chacune évalue un aspect, qui de même fait l'objet d'une évaluation *négative et positive* avec évaluation globale.

Proposition

Ci-dessous la découpe de la note à donner au projet, que nous vous demandons d'appliquer dans vos évaluations pour obtenir une notation uniforme pour tous. Le nombre de points pour chaque catégorie est proposée pour une note finale sur 24 points. Comme il s'agit d'un projet pour débutant, les structures de données et les méthodes pour résoudre les problèmes sont données et ne nécessitent pas de recherche pour l'apprenant ; aucune note n'est donc donnée pour ces deux aspects.

Grille de notation du projet (sur 24 points en référence au 24 points obtenu dans le MOOC pour cette partie)

- **Découpe (4 points)** [La découpe du code global avec commentaires] initiaux, et respect de l'ordre des parties : import des modules externes, suivi des définitions des constantes globales, suivi des définitions des fonctions et enfin, code global.
  - 4 points si les quatre parties sont présentes et l'ordre est respecté ;
  - 3 points si une des quatre parties manque ou est dans le mauvais ordre ;
  - 2 points si deux des quatre parties manquent ou sont dans le mauvais ordre ;
  - 1 points si trois des quatre parties manquent ou sont dans le mauvais ordre ;
  - 0 point si rien n'est satisfaisant.
- **Structuration en fonctions (3 points)** [La structuration en] fonctions de bonne taille (25 lignes maximum) et cohérentes (pas de fonctions définies mais non appelées par exemple) est réalisée :
  - 3 points si la structuration est satisfaisante ;
  - 2 points si des fonctions sont définies, mais certaines sont incohérentes soit sont trop longues ;
  - 1 points si des fonctions sont définies, mais certaines sont incohérentes et d'autres sont trop longues ;
  - 0 point si aucune fonction n'est définie ou une certaine structuration en fonction existe mais ne respecte pas du tout les règles de bonnes pratiques.
- **Bonnes pratiques (4 points)** [Les bonnes pratiques sont] respectées en matière d'utilisation des noms des constantes, variables, fonctions. . . , de l'indentation. . . :
  - 4 points si toutes les bonnes pratiques sont respectées ;
  - 3 points si les noms des constantes, variables et fonctions respectent les bonnes pratiques, mais l'indentation (4 caractères par incrément) n'est pas respectée ;
  - 2 points si les règles sont globalement repectées à quelques exceptions près à la fois au niveau indentations et noms de constantes, variables ou fonctions ;
  - 0 point si les règles de bonnes pratiques en matière d'encodage ne sont globalement pas respectées.
- **Commentaires (4 points)** [Commentaires et docstrings sont bien] présents : le docstring initial possède les informations suivantes : identité de l'auteur, date, ce que fait le programme, les fonctions possèdent un docstring décrivant les paramètres, ce que fait la fonction aux paramètres (par défaut, ils ne sont pas modifiés) et le résultat de la fonction, des commentaires pertinents existent quand c'est utile à la compréhension
  - 4 points si commentaires et docstrings sont présent et respectent les règles de bonnes pratiques ;
  - 3 points si un des éléments du docstring initial ou des commentaires est manquant ou non satisfaisant ;
  - 2 points si le docstring initial et les commentaires sont manquants ou non satisfaisants, mais que les fonctions ont des docstrings corrects ;
  - 0 point si les trois éléments docstring initial, commentaires et docstrings de fonctions sont manquants ou non satisfaisants.
- **Consignes (2 points)** : (voir énoncé du projet)
  - 2 points si les consignes du projet sont respectées ;
  - 1 point si la moitié des consignes est respectée ;
  - 0 point si auncune consigne n'est respectée.
- **Résultat (7 points)** : Le programme fonctionne correctement.
  - 7 points si c'est le cas ;
  - 6 points si le programme donne les bons résultats sur la plupart des jeux de données ;
  - 4 points si le programme donne le bon résultat sur au moins un jeu de données ;
  - 2 points si le programme s'exécute sans échec sur au moins un jeu de données ;
  - 0 point si le programme ne s'exécute pas sans échec.

Explication sur l'évaluation

En plus des points attribués, il est essentiel pour l'évaluateur d'expliquer pourquoi telle ou telle note a été donnée, pour que l'auteur du projet puisse comprendre les notes données et puisse s'améliorer les fois suivantes.

ET COMMENT ETAIT ÉVALUÉ UN PROJET ?

Dans le cadre des trois premières sessions du cours « Apprendre à Coder avec Python », l'évaluation de chaque projet, se faisait en deux temps :
- d'une part par une évaluation par les pairs
- d'autre part, par une auto-évaluation du projet après éventuelles améliorations.

En pratique, la remise du projet et son évaluation étaient faites en **6 étapes** :

1. Après l'avoir réalisé, une soumission du projet éventuellement avec un fichier supplémentaire montrant un résultat.
2. Chaque apprenant devait évaluer trois projets d'autres étudiants choisis au hasard.
3. Chacun recevrait les notes et commentaires sur son propre projet venant d'au moins deux autres de vos pairs (deux autres apprenants).
4. À la lumière des commentaires donnés par les pairs, chacun corrigeait et soumettait une version améliorée de son projet.
5. et réalisait une auto-évaluation, c'est-à-dire chacun évaluait son propre projet.

Le document consignes pour la notation, téléchargeable ici reprend les explications et détaille chaque notation et demande de commenter chaque note donnée.

La note finale du travail sera la somme des évaluations par les pairs (24 points) et de votre auto-évaluation (24 points).

Temps et délais pour réaliser le projet

L'estimation du temps moyen que va vous mettre le projet Vasarely (en supposant que vous utilisez la fonction `deformation` qui vous est fournie), est très variable de quelque heures à plusieurs dizaines d'heures. réalisation.

DÉTAILS PRATIQUES SUR L'ÉVALUATION

Les paragraphes qui suivent vous permettent de réaliser :
- d'une part l'évaluation des projets de trois de vos pairs
- et d'autre part, quand vous aurez reçu deux évaluations de votre propre projet (d'autres pairs), une auto-évaluation de votre projet.

DERNIERS CONSEILS AVANT DE FAIRE LES ÉVALUATIONS

L'évaluation par les pairs avec les commentaires qui l'accompagnent ainsi que l'auto-évaluation peuvent être très utiles pour l'apprentissage de la programmation,.

Pour que cela soit le cas, lors des différentes phases de notation, essayez d'être :
- objectif dans vos évaluations,
- clairs et précis dans vos commentaires,
- mais toujours courtois et bienveillants

comme vous voudriez que les autres le soient pour vous-même.

Bon travail !

2.1.4 Galerie des oeuvres produites par le projet

RAPPEL DU CONTEXTE

Lors des 3 premières sessions du MOOC « Apprendre à coder avec Python » le projet consistait à réaliser
- une oeuvre d'art géométrique (Op Art)
- sous la forme d'un pavage hexagonal déformé par une sphère.

Les liens ci-dessous est un recueil des résultats lors des 3 sessions où ce projet a été proposé, finis ou en cours, parfois non conformes à l'énoncé mais pouvant donner un effet intéressant, ou parfois beaucoup plus sophistiqués que ce qui était demandé, au gré de l'imagination et de l'expertise de chacun.

Grand merci à toutes et tous pour le plaisir que vous nous avez donné à la vision de ces réalisations !

Isabelle, Sébastien, Thierry et L'équipe Pédagogique

GALERIE DES OEUVRES

— Les oeuvres de la session 3
— Les oeuvres de la session 2
— Les oeuvres de la session 1

CHAPITRE 3

Annexes

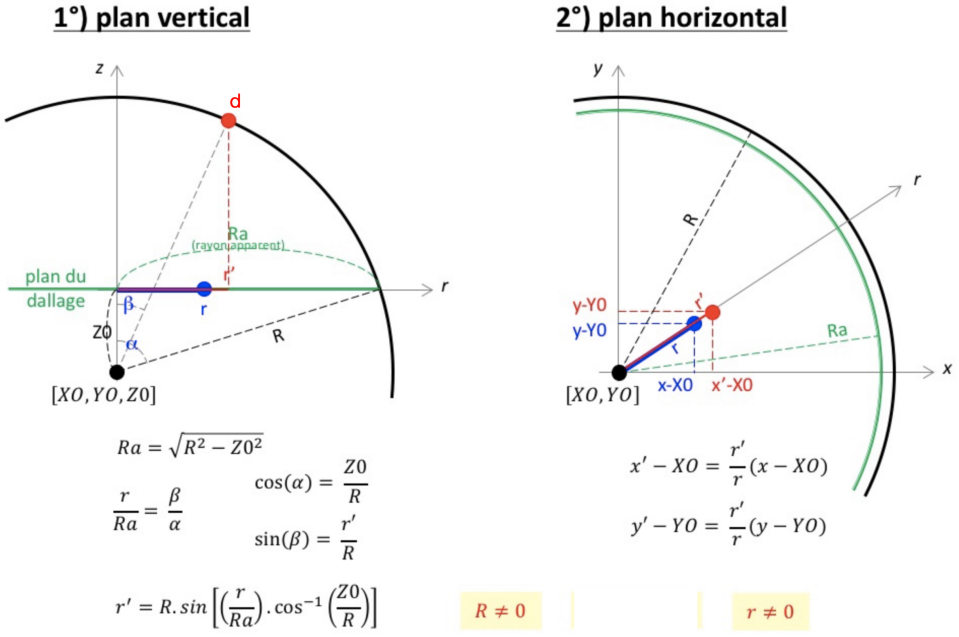
— Explications sur la déformation du projet Vasarely
— Aide-mémoire
— Petit manuel des bonnes pratiques Python

Projet Vasarely - Explications de la déformation

Crédit

La méthode et les schémas explicatifs sont de Thierry Duquenoÿ.

Explications



Pour les calculs, les points du pavage sont donnés avec des coordonnées en trois dimensions (x, y, z).
Le pavage sans déformation se trouve sur le plan avec $z = 0$ (plan x, y). Une sphère est définie par son centre et son rayon. Cette sphère va définir la déformation.
L'idée de base est que la surface déformée va se coller sur la sphère. Tous les points jusqu'aux bords de la déformation ne subissent aucune déformation. Le point au centre de la déformation va changer de hauteur (coordonnée z), mais ne changera pas non plus de coordonnées x et y. Par contre tous les autres points, à l'intérieur de la sphère vont avoir leurs trois coordonnées modifiées par la déformation. Le schéma ci-dessous explique comment les nouvelles coordonnées d'un point déformé seront calculées.
Nous supposons que le point avant déformation est à l'intérieur de la sphère de déformation et que le centre de déformation est sous le plan $z = 0$. Si le centre est au dessus, il suffit d'inverser le signe de z' après les calculs.
Regardons d'abord le plan vertical.

Sur le schéma, le centre de la déformation est en coordonnées $(X0, Y0, Z0)$ et le rayon vaut R.
Le rayon apparant Ra correspond au rayon de la calote de sphère apparent : d'après Pythagore, $Ra = \sqrt{R^2 - Z0^2}$.
Le point avant déformation (x, y, z) est tel que le rapport entre sa distance r entre au point $x = y = z = 0$ et la valeur Ra doit être égale après déformation au rapport entre l'angle β formé par l'axe $x = y = 0$ et le point d après déformation et l'angle α formé par l'axe $x = y = 0$ et le segment passant par le centre de la sphère $(X0, Y0, Z0)$ et par le point d'intersection entre le plan $z = 0$ et la sphère.
Ayant r', la distance entre la projection du point $d = (x', y', z')$ sur le plan $z = 0$;
Nous avons :

$$\frac{r}{Ra} = \frac{\beta}{\alpha}$$
$$cos(\alpha) = \frac{Z0}{R}$$
$$sin(\beta) = \frac{r'}{R}$$
$$r' = R.sin((\frac{r}{Ra}).cos^{-1}(\frac{Z0}{R}))$$

On peut donc voir avec le plan horizontal que :

$$x' = X0 + \frac{r'}{r}(x - X0)$$
$$y' = Y0 + \frac{r'}{r}(y - Y0)$$

Et à nouveau avec le plan vertical :

$$\beta = sin^{-1}(\frac{r'}{R})$$
$$z' = Z0 + R.cos(\beta)$$

Aide-mémoire PYTHON3 - v3.8.1

Fonctions prédéfinies

- `abs(x)` : valeur absolue de x
- `int(x)` : valeur x convertie en entier
- `float(x)` : valeur x convertie en réel
- `str(x)` : valeur x (int ou float), convertie en str
- `list(x)` : valeur x convertie en liste
- `tuple(x)` : valeur x convertie en tuple
- `dict(x)` : séquence de couples x convertie en dictionnaire
- `set(x)` : x converti en ensemble
- `help(x)` : aide sur x
- `dir(x)` : liste des attributs de x
- `type(x)` : type de x
- `print(...)` : imprime
- `input(x)` : imprime le string x et lit le string qui est introduit
- `round(x [,ndigits])` : valeur arrondie du float x à ndigits chiffres (par défaut 0)
- `range([start], stop, [step])` : retourne une suite d'entiers
- `sorted(s)` : retourne une liste avec les éléments de s triés

Gather, scatter et keyword arguments

- `def fun(*args)` : *gather* des arguments en un tuple args
- `fun(*s)` : *scatter de la séquence s lors de l'appel

Opérations et méthodes sur les séquences (str, list, tuples)

- `len(s)` : longueur de la séquence s
- `min(s), max(s)` : élément minimum, maximum
- `sum(s)` : (ne fonctionne pas pour les string) : somme de tous les éléments de la séquence s (valeur numérique)
- `s.index(value, [start, [stop]])` : premier indice de value dans s[start :stop]
- `s.count(sub [,start [,end]])` : nombre d'occurrences sans chevauchement de sub dans `s[start:end]`
- `enumerate(s)` : construit une séquence de couples dont le ième élément (à partir de 0) vaut le couple (i, s[i])
- `zip (a,b), zip(a,b,c)`, ... : construit une séquence de couples, resp. triples, ..., dont le ième élément reprend le ième élément de chaque séquence a, b [,c]

Méthodes sur les chaînes de caractères (str)

- `s.lower(), s.upper()` : string avec caractères en minuscules respectivement en majuscules
- `s.islower()`, `s.isdigit()`, `s.isalnum()`, `s.isalpha()`, `s.isupper()` : vrai si s n'est pas vide et n'a (respectivement) que des minuscules, des chiffres, des car. alphanumériques, alphabétiques, majuscules
- `s.find(sub [,start [,end]])` : premier indice de s où le sous string sub est trouvé dans `s[start:end]`
- `s.replace( old, new[, co])` : copie de s en remplaçant toutes les (ou les co premières) occurrences de old par new.
- `s.format(...)` : copie de s après formatage
- `s.capitalize()` : copie de s avec première lettre en majuscule
- `s.strip()` : copie de s en retirant les *blancs* en début et fin
- `s.join(t)` : crée un str qui est le résultat de la concaténation des éléments de la séquence de str t chacun séparé par le str s
- `s.split([sep [,maxsplit])` : renvoie une liste d'éléments séparés dans s par le caractère sep (par défaut *blanc*) ; au maximum maxsplit séparations sont faites (par défaut l'infini)

CC BY SA

Opérateurs et méthodes sur les listes

- `s.append(v)` : ajoute un élément valant v à la fin de la liste
- `s.extend(s2)` : rajoute à s tous les éléments de la liste s2
- `s.insert(i,v)` : insère l'objet v à l'indice i
- `s.pop([i])` : supprime l'élément d'indice i de la liste (par défaut le dernier) et retourne la valeur de l'élément supprimé
- `s.remove(v)` : supprime la première valeur v dans s
- `s.reverse()` : renverse l'ordre des éléments de la liste, le premier et le dernier élément échangent leurs places, ...
- `s.sort(key=None, reverse=False)` : trie s en place
- `s.copy()` : *shallow* copie superficielle de s
- `del s[i], del s[i:j]` : supprime un ou des éléments de s

Méthodes sur les dictionnaires (dict)

- `d.clear()` : supprime tous les éléments de d
- `d.copy()` : *shallow* copie de d
- `{}.fromkeys(s,v)` : crée un dict avec les clés de s et la valeur v
- `d.get(k [,v])` : renvoie la valeur d[k] si elle existe v sinon
- `d.items()` : liste des items (k,v) de d
- `d.keys()` : liste des clés
- `d.pop(k [,v])` : enlève d[k] s'il existe et renvoie sa valeur ou v sinon
- `d.popitem()` : supprime un item arbitraire (k,v) et retourne l'item sous forme de tuple
- `d.setdefault(k [,v])` : d[k] si elle existe sinon v et rajoute d[k]=v
- `d.update(s)` : s est une liste de tuples que l'on rajoute à d
- `d.values()` : liste des valeurs de d
- `del d[k]` : supprime l'élément de clé k de d

Méthodes sur les ensembles (set)

- `s = set(v)` : initialise s : un set contenant les valeurs de v
- `s.add(v)` : ajoute l'élément v au set s (ne fait rien s'il y est déjà)
- `s.clear()` et `s.copy()` : idem dictionnaires
- `s.remove(v)` : supprime l'élément v du set (erreur si v n'est pas présent dans s)
- `s.discard(v)` : si v existe dans s, le supprime
- `s.pop()` : supprime et renvoie un élément arbitraire de s

Modules

- `math` : accès aux constantes et fonctions mathématiques (`pi`, `sin()`, `sqrt(x)`, `exp(x)`,`floor(x)` (valeur plancher), `ceil(x)` (valeur plafond), ...) : exemple : `math.ceil(x)`
- `copy: copy(s), deepcopy(s)` : *shallow* et *deepcopy* de s

Méthodes sur les fichiers

- `f = open('fichier')` : ouvre 'fichier' en lecture (autre paramètres possibles : 'w'(en écriture), 'a'(en écriture avec ajout), encoding ='utf-8' : encodage UTF-8)
- `with open('fichier'...) as f` : ouvre 'fichier' pour traitement à l'intérieur du with
- `for ligne in open('fichier'...)` : ouvre et traite chaque ligne de 'fichier' et le ferme à la fin du for
- `f.read()` : retourne le contenu du fichier texte f
- `f.readline()` : lit une ligne
- `f.readlines()` : renvoie la liste des lignes de f
- `f.write(s)` : écrit la chaîne de caractères s dans le fichier f
- `f.close()` : ferme f

23.10.2020

Quelques bonnes pratiques de programmation (Python)

Traduction d'un problème en programme

1. *Analysez le problème*

- Identifiez clairement ce que sont les **données fournies**, ainsi que les **résultats et types** attendus à l'issue du traitement.
- Formalisez une **démarche générale de résolution** par une séquence d'opérations simples.
- Vérifiez que vous **envisagez tous les cas** de figure (en particuliers les cas "limites").

2. *Découpez votre problème en fonctions*

- Chaque fonction doit réaliser ***une* tâche** clairement identifiée.
- Limitez les fonctions à **25 lignes** maximum, sauf dans des cas exceptionnels.
- **Evitez la redondance** dans le code (copier/coller). Si cela arrive, c'est qu'il manque soit une fonction, soit une boucle, soit que des tests conditionnels peuvent être regroupés.
- N'utilisez **pas de variables globales**.
- Veillez à ce que tous les paramètres et variables d'une fonction soient **utilisés** dans cette fonction.
- Pour une fonction qui renvoie un résultat, organisez le code pour qu'il ne contienne qu'**un seul** `return`, placé comme dernière instruction de la fonction. Pour une fonction qui ne renvoie pas de résultat, ne mettez pas de `return` (il y en aura un implicitement à la fin de l'exécution de la fonction).
- Ne modifiez pas les *paramètres*.□ Exemple: Si vous recevez une borne inférieure `first` et une supérieure `last` et que vous devez itérer de la première à la dernière, n'incrémentez pas `first` dans la boucle, car la signification n'en serait plus claire ; créez plutôt une variable locale `pos` initialisée à `first`.
- Sauf si la fonction a comme but de modifier la (structure de) données reçue en paramètre ; dans ce cas la fonction ne renvoie pas de valeur.

3. *Testez le code au fur et à mesure du développement*

- Créez des **scénarios de test**, pour lesquels vous choisissez les données fournies et vous vérifiez que le résultat de la fonction est conforme à ce que vous attendez.
- Vérifiez les cas particuliers et les *conditions aux limites*. □ Exemples: Pour le calcul d'une racine carrée, que se passe-t-il lorsque le paramètre est un nombre négatif ?

Programmation

1. *Style de programmation*

- N'utilisez pas les instructions `break` ou `continue`
- Utilisez la forme raccourcie `if(is_leap_year(2008))` plutôt que la forme équivalente `if(is_leap_year(2008)==true)`
- Utilisez la forme `return <expression booléenne>` plutôt que la forme équivalente

```
if <expression booléenne>:
  res = True
else:
  res = False
return res
```

- N'exécutez pas plusieurs fois une fonction alors qu'une exécution suffit en retenant le résultat.
- Précisez le domaine de validité des paramètres.

Programmation (suite)

2. *Quelques erreurs classiques*

- Vous essayez d'utiliser une variable avant de l'avoir **initialisée**.
- L'**alignement des blocs** de code n'est pas respecté.
- Vous oubliez de fermer un fichier que vous avez ouvert.

Nommage de variables, fonctions, *etc.*

1. *Utilisez une convention de nommage*

`joined_lower`	pour les **variables** (attributs), et **fonctions** (méthodes)
`ALL_CAPS`	pour les **constantes**

2. *Choisissez bien les noms*

- Donnez des noms de variables qui expriment leur contenu, des noms de fonctions qui expriment ce qu'elles font (cf. règles de nommage ci-dessus).
- Évitez les noms trop proches les uns des autres.
- Utilisez aussi systématiquement que possible les mêmes genres de noms de variables.
  □ Exemples: `i`, `j`, `k` pour des indices, `x`, `y`, `z` pour les coordonnées, `max_length` pour une variable, `is_even()` pour une fonction, etc.

Style et documentation du code

1. *Soignez la clarté de votre code*

... c'est la première source de documentation.

- Utilisez les `docstrings` dans chaque fonction pour :
  - brièvement décrire ce que fait la fonction, **pas comment elle le fait**, et préciser ses entrées et sorties.
  - décrire les arguments des fonctions.
- Soignez les indentations (2 à 4 espaces chacune) et la gestion des espaces et des lignes blanches (deux lignes blanches avant et entre chacune des définitions de fonction globales ; une ligne blanche pour mettre en évidence une nouvelle partie dans une fonction),
- Il faut commenter le code à bon escient et avec parcimonie. Évitez d'indiquer le *fonctionnement* du code dans les commentaires. □ Exemples: Avant l'instruction "`for car in line:`", ne pas indiquer qu'on va boucler sur tous les caractères de la `line`...
- Évitez de paraphraser le code. N'utilisez les commentaires que lorsque la fonction d'un bout de code est difficile à comprendre.

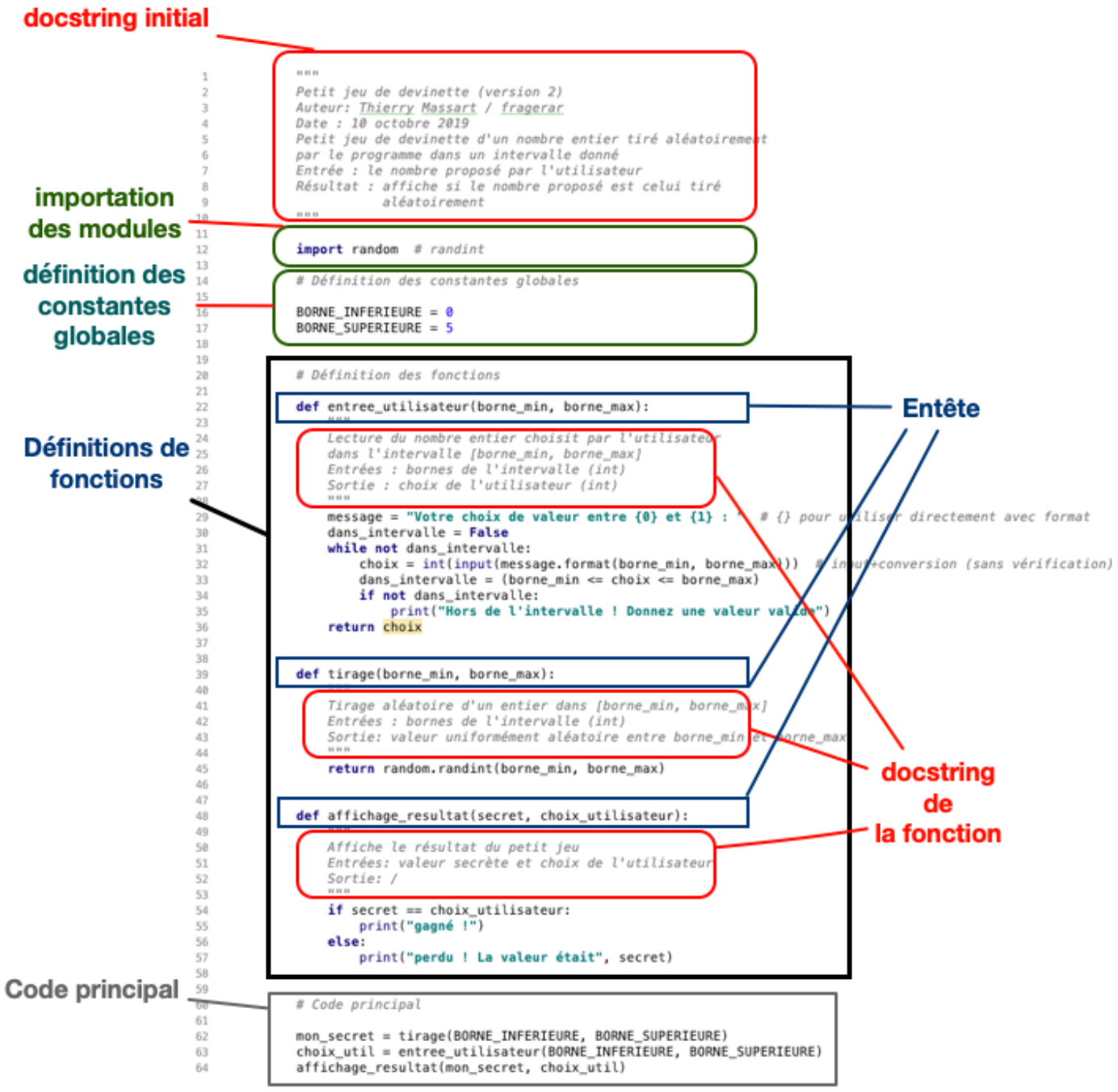
Structure d'un programme Python

1. *Voir verso*

BY SA

Quelques bonnes pratiques de programmation (Python)

Structure d'un programme Python



Index

évaluer le projet, 14, 26

aide-mémoire, 31

Bonnes pratiques Python, 31

déformation du projet Vasarely, 31

escape game, 3

jeu d'évasion, 3

projet Chateau, 3

À propos du cours

Vous avez un ordinateur, désirez apprendre à coder et êtes totalement ou partiellement débutant dans le domaine; vous êtes étudiant, professeur ou simplement une personne qui sente l'envie ou le besoin d'apprendre la programmation de base; ce cours utilise Python 3 comme clé pour vous ouvrir la porte de cette connaissance informatique.

Ce cours est orienté vers la pratique, et propose un matériel abondant pour couvrir l'apprentissage de la programmation de base, d'une part en montrant et expliquant les concepts grâce à de nombreuses capsules vidéo courtes et des explications simples, et d'autre part en vous demandant de mettre ces concepts en pratique d'abord de façon guidée et ensuite autonome. Plusieurs quiz, un projet individuel, et de nombreux exercices à réaliser et validés automatiquement avec notre outil UpyLaB intégré au cours, vous permettent de polir et ensuite de valider votre apprentissage.

Format du cours

Le cours s'étale sur 9 semaines et propose 3 parcours d'apprentissage et correspond à un travail hebdomadaire de 6 à 12 heures avec un projet évalué par les pairs. Si vous ne pouvez y consacrer 6h par semaine un parcours à allure libre est possible, mais sans projet ni attestation finale délivrée par FUN.

Les enseignants

SÉBASTIEN HOARAU

Sébastien Hoarau est maître de conférences à l'Université de la Réunion (UR) où, depuis plus de 20 ans, il enseigne la programmation aux étudiants de première année scientifique. Sa pédagogie est axée sur la pratique : quiz en cours, projets, utilisation de plateformes ludiques.

THIERRY MASSART

Thierry Massart est professeur à l'Université Libre de Bruxelles (ULB) où, depuis plus de 25 ans, il enseigne la programmation principalement aux étudiants de Sciences Informatique et de l'école Polytechnique de l'ULB. Il leur propose une pédagogie active orientée vers la pratique.

JEAN OLGIATI

Formé en ingénierie, graphisme et pédagogie, Jean travaille à l'Université libre de Bruxelles et à la Scientothèque, association qui promeut l'égalité des chances par les sciences. Il participe à des projets de soutien aux enseignants qui investissent le champ du numérique, et anime auprès de jeunes un atelier sur la programmation et la robotique.. Il a découvert Python grâce au présent MOOC et a contribué à définir le projet de fin de cours.

ISABELLE POIRIER

Isabelle est professeur agrégé de mathématiques et enseigne celles-ci depuis plus de 15 ans en établissement secondaire dans le sud de la France. Autodidacte en programmation (en particulier grâce à sa participation à plusieurs MOOC), elle met à profit son sens de la pédagogie et sa propre expérience pour venir en aide aux apprenants sur le forum et améliorer l'approche didactique proposée au bénéfice d'un meilleur apprentissage pour tous.