UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Programmation Impérative - TD : Feuille 2

Exercice 1 – Affichage et Paramètres de Programme

Q1) Écrire un programme Python qui crée une variable `qui` et lui donne comme valeur la chaîne de caractères `Robin`. Puis affiche `Bonjour`, suivi de la valeur de `qui`, suivi d'un point d'exclamation. Quand on le lance, il doit donc afficher `Bonjour Robin !`.

Q2) Proposer une solution pour qu'il n'y aie pas d'espace entre le nom et le point d'exclamation.

Q3) En utilisant `import sys` puis les variables `sys.argv[0]` et `sys.argv[1]`, écrire un programme qui utilise le paramètres du programme pour remplir la valeur de `qui`, puis affiche la même chose que dans la première question, suivi de `(dit blabla.py)` mais avec `blabla.py` remplacé par le nom du programme Python. Par exemple, si le programme est écrit dans `hello.py` et qu'on le lance avec `python3 hello.py Kim`, il affiche `Bonjour Kim ! (dit « hello.py »)`

Exercice 2 – Racines de Polynômes

Les racines (valeurs de x pour lesquelles le polynôme vaut 0) d'un polynôme de degré 2, $ax^2 + bx + c$, sont définies par : $\frac{-b \pm \sqrt{\Delta}}{2a}$ avec $\Delta = b^2 - 4ac$

Q4) Écrire un programme Python qui crée trois variables `a`, `b` et `c`, et les initialisent avec les valeurs 1.523, 10.422 et 3.33333. Puis calcule et affiche les racines du polynôme $ax^2 + bx + c$, en supposant qu'elles existent.

Q5) Écrire une variante de votre programme pour qu'il reçoivent les valeurs de `a`, `b` et `c` en paramètres. Il affiche par exemple la même chose que précédemment si on le lance avec `python3 racines.py 1.523 10.422 3.33333`.

Q6) Modifiez votre programme pour qu'il affiche un message différent selon le nombre de racines (il y a 3 cas possibles).

Q7) Écrire une variante de votre programme pour qu'il affiche `a :` pour demander à l'utilisateur de choisir la valeur de `a`, puis de même pour `b` et pour `c`.

Exercice 3 – Années Bissextiles

Une année est bissextile si elle est un multiple de 4, sauf si elle est un multiple de 100, sauf si elle est un multiple de 400. Par exemple : 2008 est bissextile (divisible uniquement par 4); 1900 n'est pas bissextile, car divisible par 100 mais non divisible par 400; 2000 est bissextile car divisible par 400.

On rappelle que l'opérateur « modulo » (noté `%` en python) renvoi le reste de la division entière de deux nombres. Par exemple, `35 % 5` vaut `0` puisque `35` est divisible par `5`, et `39 % 5` vaut `4` (`39 = 4 + 5*7`).

Q8) Écrire un programme Python qui commence par créer une variable `an` avec pour valeur `1999`. Le programme doit raisonner sur la valeur de `an` et afficher `1999 est année bissextile` (si c'est le cas) et `1999 est une année normale` sinon. Défit : votre solution peut elle être simplifiée, par exemple en réduisant le nombre de `if`, de conditions, de ligne `print` ?

Q9) Écrire une variante de votre programme pour qu'il affiche `Veuillez taper une année :` pour demander à l'utilisateur de choisir la valeur de `an`.

Q10) Écrire une variante de votre programme qui prend un nombre quelconque de paramètres représentant des années, et pour chacune d'entre elle affiche une ligne qui dit si l'année est bissextile.

Synthèse de la bibliothèque `qtido`

Après avoir importé la bibliothèque `qtido` avec :

```
from qtido import *
```

Il est possible de créer une fenêtre avec la fonction `creer(w, h)`, par exemple :

```
f = creer(700, 500)
```

En supposant que `f` est une fenêtre graphique (retournée par la fonction `creer(…)`), la bibliothèque `qtido` fourni de nombreuses fonctions. Le premier paramètre est toujours la fenêtre concernée par l'opération. La bibliothèque accepte la plupart du temps des valeurs de coordonnées non entières (par exemple 10.5).

Fonctions relatives à la tortue

- `creer_tortue(f)` : Crée une tortue pour tracer dans la fenêtre `f`. Renvoie la tortue créée.
- `tortue_avance(t, d)` : Ordonne à la tortue d'avancer de `d` pixels.
- `tortue_droite(t, da)` : Ordonne à la tortue de tourner vers la droite d'un angle de `da` degrés.
- `tortue_gauche(t, da)` : Ordonne à la tortue de tourner vers la gauche d'un angle de `da` degrés.
- `tortue_stop(t)` : Ordonne à la tortue de lever le stylo (d'arrêter de tracer).
- `tortue_trace(t)` : Ordonne à la tortue d'abaisser le stylo (de recommencer à tracer).

Fonctions d'affichage simple

Les premières fonctions utiles pour générer un dessin avec des lignes et le sauvegarder.

- `effacer(f)` : Repeindre en noir toute la fenêtre.
- `couleur(f, r, g, b)` : Définir la couleur des tracés à partir de ce moment. `r`, `g`, `b` sont les quantités de rouge, vert et bleu, comprises entre 0 et 1. Par exemple, le noir est 0,0,0 (absence de chacune des couleurs), le blanc est 1,1,1 (quantité maximale des trois couleurs), le jaune est 1,1,0 (mélange rouge+vert, sans bleu), 1,0.5,0 est un orange (entre jaune et rouge)…
- `ligne(f, x1, y1, x2, y2)` : Trace un segment de (x1,y1) à (x2,y2).
- `exporter_image(f, nom_fichier)` : Sauvegarder la fenêtre sous forme d'un fichier image (par exemple 'toto.png').

Et d'autres fonctions d'affichage.

- `rectangle(f, x1, y1, x2, y2)` : Trace un rectangle dont un coin a pour coordonnées (x1,y1) et l'autre (x2,y2). Le second coin est exclu du tracé.
- `cadre(f, x1, y1, x2, y2)` : Comme `rectangle` mais trace uniquement le contour.
- `disque(f, cx, cy, r)` : Rempli un disque centré en (cx,cy) et de rayon `r`.
- `cercle(f, cx, cy, r)` : Comme `disque` mais trace uniquement le contour.
- `texte(f, g, b, taille, texte)` : Affiche la chaîne `texte` avec une taille de caractères de `taille` et avec le coin inférieur gauche de coordonnées (g, b).
- `texte_centre(f, cx, b, taille, texte)` : Comme `texte` mais le texte est centré horizontalement autour du point (cx, b).
- `epaisseur_du_trait(f, w)` : Définir l'épaisseur du stylo utilisé pour tracer les contours (cadre, cercle, ligne, etc.). L'épaisseur `w` est exprimée en nombre de pixels.

Et encore d'autres fonctions plus avancées.

- `polygone(f, liste_points)` :
- `polyligne(f, liste_points)` :
- `utiliser_transformation(f, tx, ty, sx=1, sy=1, r=0)` : Change la transformation utilisée pour le tracer. Tout les tracés auront une translation de (`tx`, `ty`), un étirement horizontal de `sx` et vertical de `sy`, et enfin une rotation de `r` degrés.
- `annuler_transformation(f)` : Remet la transformation à sa valeur par défaut. Cette fonction est aussi automatiquement appelée par `effacer(…)`.

Fonctions pour l'animation et la gestion de la fenêtre

- `est_fermee(f)` : Renvoie un booléen valant `True` si la fenêtre a déjà été fermée par l'utilisateur.
- `attendre_fermeture(f)` : Bloque et attend que l'utilisateur ferme la fenêtre.
- `attendre_pendant(f, ms)` : Bloque et attend pendant `ms` millisecondes.
- `re_afficher(f)` : Force le réaffichage (peut être utile avec des animations qui n'utilisent jamais `attendre_pendant` ou `attendre_evenement`).
- `reinitialiser_attendre_evenement(f, trigger=False)` :

Fonctions pour la gestion d'événements clavier et souris

- `attendre_evenement(f, ms)` : Comme `attendre_pendant` mais se débloque aussi si un événement (clavier, bouton, …) se produit. Dans le cas où il s'est produit un événement, la fonction `dernier_evenement` permet de le récupérer. Dans le cas où les `ms` millesecondes se sont écoulées, `dernier_evenement` renverra la valeur `None`.
- `dernier_evenement(f)` : Renvoi un identifiant du dernier événement s'étant produit (au dernier appel de `attendre_evenement`) ou `None` si aucun événement ne s'est produit.
- `les_touches_appuyees(f)` : Renvoi une liste des touches du clavier qui sont actuellement appuyées.

Fonctions pour l'utilisation de boutons, etc

Les « widgets » (boutons, champ textes, etc.), une fois ajoutés à une fenêtre, s'affiche automatiquement. Il faut donc ajouter un widget juste après avoir créer la fenêtre et non pas à chaque fois que l'on ré-affiche son contenu. Les fonctions d'ajout prennent des coordonnées (x1, y1, x2, y2) qui correspondent au rectangle que doit occuper le widget.

- `ajouter_bouton(f, ev, x1, y1, x2, y2, texte)` : Crée un bouton avec `texte` marqué dessus. Quand ce bouton est clické, l'événement `ev` sera émis.
- `ajouter_slider(f, ev , x1, y1, x2, y2, v_min, v_max)` : Ajoute durablement . `v_min` et `v_max` sont inclus, il y a donc `v_max - v_min + 1` valeurs possibles.
- `ajouter_champ_texte(f, ev , x1, y1, x2, y2)` : Crée un bouton avec `texte` marqué dessus. Quand le texte change, l'événement `ev` sera émis.
- `ajouter_zone_texte(f, ev , x1, y1, x2, y2)` : Comme le champ texte mais crée une zone où il est possible de taper plusieurs lignes.
- `supprime_widgets(f)` : Supprime tous les widgets de la fenêtre.

Quand un widget contient une valeur (tous sauf les boutons), il est possible d'accéder à la valeur ou de la modifier, en utilisant le nom de l'événement donner lors de la création du widget.

- `lire_slider(f, ev)` : Renvoie la valeur, sous forme d'un entier, du slider associé à l'événement `ev`.
- `lire_champ_texte(f, ev)` : Renvoie la valeur, sous forme d'une chaîne de caractères, du champ texte associé à l'événement `ev`.
- `lire_zone_texte(f, ev)` : Renvoie la valeur, sous forme d'une chaîne de caractères, de la zone de texte associée à l'événement `ev`.
- `changer_slider(f, ev, value)` : Change la valeur du slider associé à l'événement `ev`. Le paramètre `val` doit être un entier.
- `changer_champ_texte(f, ev, val)` : Change le contenu du champ texte associé à l'événement `ev`. Le paramètre `val` doit être une chaîne de caractères.
- `changer_zone_texte(f, ev, val)` : Change le contenu de la zone de texte associée à l'événement `ev`. Le paramètre `val` doit être une chaîne de caractères.