UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Outils Info. - TD : Feuille 2 + Référence qtido (verso)

Exercice 1 – Affichage et `input()` 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 ait pas d'espace entre le nom et le point d'exclamation.

Q3) En utilisant la fonction prédéfinie `input`, écrire un programme qui écrit à l'utilisateur `Entrez votre nom :` pour lui demander de taper au clavier la valeur de `qui`, puis affiche la même chose que dans la première question.

Q4) Écrire maintenant un programme qui demande à l'utilisateur son nom et le jour de la semaine et lui affiche par exemple `Bonjour Robin, je te souhaites un bon mardi.`.

Q5) Écrire un programme qui initialise deux variables, une à 27 et l'autre à 5. Le programme doit ensuite calculer le reste de la division entière du premier nombre par le second puis afficher (en utilisant les variables) : `Le reste de la division 27/5 est 2.`

Q6) Écrire une variante de ce programme qui demande à l'utilisateur de taper les valeurs des deux entiers (l'un après l'autre).

Exercice 2 – Racines (réelles) 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 $\Delta = b^2 - 4ac$

Q7) Écrire un programme Python qui crée trois variables `a`, `b` et `c`, et les initialise avec les valeurs 1.523, 10.42 et 3.33, puis calcule et affiche les deux racines du polynôme $ax^2 + bx + c$.

Q8) Écrire une variante de votre programme pour qu'il demande les valeurs de `a`, `b` et `c` à l'utilisateur. Il affiche alors la même chose que précédemment si l'utilisateur tape `1.523` puis `10.42` et `3.33`.

Q9) Modifiez votre programme pour qu'il affiche un message différent selon le nombre de racines (il y a 3 cas possibles).

Exercice 3 – Années Bissextiles

Q10) Une année est bissextile si elle est un multiple de 4. Écrire un programme Python qui commence par créer une variable `annee` avec pour valeur `2000`. Le programme doit raisonner sur la valeur de `annee` et afficher `2000 est année bissextile` (si c'est le cas) et `2000 est une année normale` sinon.

Q11) La règle était fausse : une année est bissextile si elle est un multiple de 4, sauf si elle est un multiple de 100. Corriger le programme pour appliquer cette nouvelle règle

Q12) La vraie règle est la suivante : 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. Corriger le programme.

Q13) Écrire un programme qui affiche, pour chaque année entre 0 et 2050, si elle est bissextile ou non.

Q14) Écrire un programme qui demande à l'utilisateur une année de `début` et une année de `fin` et affiche, pour chaque année entre `début` et `fin` (inclus), si elle est bissextile ou non.

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 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)` : Trace un polygone à partir d'une liste de points (x, y) (n-uplet ou liste à deux éléments).
- `polyligne(f, liste_points)` : Comme polygone mais en ne traçant que le contour.
- `grille_numpy(f, tab, pas, taille, x, y, couleur='-green +red')` : Affiche un tableau numpy 2D sous forme de carrés de couleur, le premier carré étant en centré en (x,y), les carrés ont la taille donnée et sont espacés avec un pas donné (par exemple, pas=taille). La couleur peut être une chaîne de caractères (par défaut `-green +red`) composée de parties séparées par des espaces ; chaque partie est composée d'un nom de couleur (`red`, `green`, `blue`) précédé d'un caractère (`+` pour que la quantité de couleur augmente avec la valeur de la case, `-` pour que la couleur suive l'inverse de la valeur, `1` pour que la couleur soit toujours à 1, `½` (copier ce caractère) pour que la valeur soit toujours à 0.5). Un autre possibilité pour `couleur` est de passer une fonction qui prend une valeur et renvoie un quadruplet de valeurs entre 0 et 1 `(r,g,b,a)` (ou a est l'opacité), par exemple, passer `niveau_gris` avec dans le programme `def niveau_gris(v): return (v,v,v,1)`.
- `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` est émis.
- `ajouter_slider(f, ev , x1, y1, x2, y2, v_min, v_max)` : Crée un slider (glissière) pour choisir une valeur entière entre `v_min` et `v_max` (inclus), il y a donc `v_max - v_min + 1` valeurs possibles. Quand la valeur change, l'événement `ev` est émis.
- `ajouter_champ_texte(f, ev , x1, y1, x2, y2)` : Crée un champ texte pré-rempli avec `texte`. Quand le texte change, l'événement `ev` est 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, val)` : Change la valeur du slider (glissière) 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.

Fonctions relatives à la tortue

- `t = creer_tortue(f)` : Crée une tortue pour tracer dans la fenêtre `f`. Renvoie la tortue créée (et la stocke ici dans `t`).
- `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).