TP 2

Sauts Conditionnels

Exercice 1: Simulation de grande aiguille

Écrire une fonction `une_minute_en_plus(h,m)` qui calcule l'heure une minute après celle passée en paramètre sous forme de deux entiers que l'on suppose cohérents. Voici quelques exemples :

- `une_minute_en_plus(14,32)` renvoie (14,33)
- `une_minute_en_plus(18,59)` renvoie (19,0)

Ne pas oublier le cas de minuit.

Solution

```
def une_minute_en_plus(h,m):
    if m < 59:
        return (h,m+1)
    elif h < 24:
        return (h+1,0)
    else:
        return (0,0)
```

Exercice 2: Résolution d'équations

1) Écrire une fonction `afficher_solution_premier_degre(a,b)` qui prend en paramètre deux réels `a` et `b` et qui AFFICHE la solution de l'équation `a.x + b = 0`.
   Contrairement au TD, inclure le cas `a=0`. On pourra afficher à l'écran des phrases comme `"Pas de solutions"` ou `"Une infinité de solutions"`
2) Écrire une fonction `afficher_solution_deuxieme_degre(a,b,c)` qui prend en paramètre trois réels `a`, `b` et `c` qui AFFICHE la (les) solution(s) de l'équation $ax^2+bx+c=0$. Ne pas oublier le cas `a=0`. Penser à réutiliser la fonction `afficher_solution_premier_degre(a,b)`.

Solution

1)

```
def afficher_solution_premier_degre(a,b):
    if a!=0:
        print (-(b*1.0)/a)
    elif b == 0:
        print ("Une infinité de solutions")
    else:
        print ("Pas de solutions")
```

2)
```
def afficher_solution_deuxieme_degre(a,b,c):
    from math import sqrt
    if a==0:
        afficher_solution_premier_degre(b,c)
    else:
        discriminant = b*b - 4*a*c
            if discriminant > 0:
                print( (-b-sqrt(discriminant)*1.0)/(2*a) )
                print( (-b+sqrt(discriminant)*1.0)/(2*a) )
            elif discriminant == 0:
                print( (-b*1.0)/(2*a) )
            else:
                print("Pas de solutions")
```

Exercice 3: Faute aux copies

La service de reprographie propose les photocopies avec le tarif suivant : les 10 premières coûtent 20 centimes l'unité, les 20 suivantes coûtent 15 centimes l'unité, et au-delà de 30 le coût est de 10 centimes. Écrire une fonction `cout_photocopies(n)` qui calcule le prix en euros à payer pour `n` photocopies.

Solution

```
def cout_photocopies(n):
    if n <= 10:
        return 0.20*n
    elif n <= 30:
        return 2.0 + 0.15*(n-10)
    else:
        return 5.0 + 0.1*(n-30)
```

Exercice 4: Découverte de la commande input

1) Tester la commande `input`. On pourra jouer avec l'instruction suivante dans l'invite de commandes :

```
t = input("Veuillez taper quelque chose...")
```

Essayer de rentrer des réels, des flottants, n'importe quoi, des booléens, des chaînes de caractères, des variables... Que fait la commande `input ?`

2) Quel est le danger d'utiliser une telle commande sur des programmes à accès public ?

Exercice 5: Application à la résolution d'équations du second degré

Utiliser la commande `input` pour écrire une fonction `afficher_trinome()` (sans argument), où l'on demandera poliment à l'utilisateur de taper 3 réels a, b et c et où l'on affichera la (les) solution(s) de l'équation $ax^2 + bx + c = 0$.

Solution

```
def afficher_trinome():
    a = input("Veuillez taper le coefficient en x^2 : ")
    b = input("Veuillez taper le coefficient en x : ")
    c = input("Veuillez taper le coefficient constant : ")
    afficher_solution_deuxieme_degre(a,b,c)
```

Les boucles

Exercice 6: Compter de 1 à n

Écrire une fonction `compter_de_1_a(n)` qui affiche les entiers d `1` à `n`.

Solution

```
def compter_de_1_a(n):
    for i in range(1,n+1):
        print(i)
```

Exercice 7: Puissances de 2

Écrire une fonction `liste_puissances_2(n)` qui affiche les `n` premières puissances de 2.

Solution

```
def liste_puissances_2(n):
    p = 1
    for i in range(n):
         print(p)
         p = 2*p
```

Exercice 8: Somme des cubes

Écrire une fonction `somme_cubes(n)` qui calcule $\sum_{i=1}^{n} i^3$.

Solution

```
def somme_cubes(n):
    s = 0
    for i in range(1,n+1):
        s = s + i*i*i
    return s
```

Exercice 9: Somme des factorielles

Écrire une fonction `somme_factorielles(n)` qui calcule $\sum_{i=1}^{n} i!$.

Solution

```
def somme_factorielles(n):
    f = 1
    s = 0
    for i in range(1,n+1):
    f = f*i
        s = s + f
    return s
```

Exercice 10: Division euclidienne

Écrire deux fonctions qui calculent le quotient et le reste de la division euclidienne de deux entiers en utilisant uniquement les opérations d'addition et de soustraction.

Solution

```
def reste(a,b):
    while (a >= b):
        a = a-b
    return a


def quotient(a,b):
    q=0
    while (a >= b):
        a = a - b
        q = q + 1
    return q
```

Exercice 11: PGCD et PPCM

Écrire deux fonctions qui calculent le PGCD et le PPCM de deux entiers.

Solution

```
def pgcd(a,b):
    while (b !=0) :
        r = a % b
        a = b
        b = r
    return a

def ppcm(a,b):
    return a*b/pgcd(a,b)
```

Exercice 12: Moyenne

Écrire une fonction qui demande à l'utilisateur de taper au clavier des réels et qui calcule et affiche au fur et à mesure la moyenne des réels qui ont été tapés. Le programme s'arrêtera dés que l'utilisateur entrera un nombre négatif.

Solution

```
def moyenne_en_temps_reel(a,b):
    s = 0
    n = 0
    x = input( "Veuillez rentrer un nombre : " )
    while (x >= 0) :
        s = s + x
        n = n + 1
        print((s*1.0)/n)
```

Exercice 13: Quel est le nombre que j'ai en tête ?

Implémentez le jeu suivant :

Le joueur doit essayer de trouver un entier compris entre 1 et 100 que le programme tirera au hasard. (On utilisera le paquet `random` où se trouve la fonction `randint(a,b)` qui renvoie un entier au hasard entre `a` et `b`.)

Avant de commencer la partie, le joueur choisit un contrat `c` qui est le total d'essais qu'il est possible d'effectuer. Une fois le contrat choisi, la partie commence. Le joueur a donc `c` propositions possibles. A chaque proposition, l'ordinateur donne une indication en disant si la solution est plus petite ou plus grande que le nombre proposé par le joueur. La partie se termine si le joueur a trouvé le nombre caché (il a gagné) ou s'il a utilisé toutes les propositions possibles (il a perdu).

Vous prendrez bien soin d'utiliser des fonctions pour rendre le code le plus lisible possible.

Solution

```
import random

def demander_solution( min, max, nb_essai, contrat ):
    print( "Il vous reste " + str( contrat-nb_essai ) + " essais." )
    solution = input(
        "Veuillez donner un entier entre " + str(min) + " et " +
        str(max) + ". "
    )
    while( not( type(solution) is int ) ):
        solution = input("Vous devez donnez un entier. ")
    return solution

def demander_contrat():
    contrat = input("Combien d'essai voulez-vous avoir ? ")
    while( not( type(contrat) is int ) or ( contrat <= 0 ) ):
        contrat = input(
            "Vous devez donnez un entier strictement positif. "
        )
    return contrat

def construire_solution( min, max ):
```

```
    return random.randint( min, max )

def l_essai_est_valide( essai, solution ):
    return essai == solution

def il_reste_des_essais( nb_essai, contrat ):
    return nb_essai < contrat

def le_jeu_continu( nb_essai, contrat, essai, solution ):
    return (
        not( l_essai_est_valide( essai, solution ) ) and
        il_reste_des_essais( nb_essai, contrat )
    )

def analyser_resultat( nb_essai, contrat, essai, solution ):
    if l_essai_est_valide( essai, solution ) :
        print(
            "Vous avez trouvé la solution dans les temps ! "
            "Vous avez gagné."
        )
    else:
        print(
            "C'est perdu, vous n'avez plus d'essai et vous n'avez "
            "pas trouvé la bonne réponse."
        )

def donner_indication( essai, solution ):
    if essai == None:
        return
    if( essai < solution  ):
        print("Le nombre à trouver est plus grand.")
    else:
        print("Le nombre à trouver est plus petit.")

def trouver_le_nombre( min=1, max=100 ):
    contrat = demander_contrat()
    solution = construire_solution( min, max )
    nb_essai = 0
    essai = None
    while( le_jeu_continu( nb_essai, contrat, essai, solution ) ):
        donner_indication( essai, solution )
        essai = demander_solution( min, max, nb_essai, contrat  )
        nb_essai = nb_essai + 1
    analyser_resultat( nb_essai, contrat, essai, solution )

trouver_le_nombre()
```