Travaux pratiques d'introduction 4

Les structures conditionnelles

Informatique tronc commun MPSI et PCSI

I Présentation

I.1 Les booléens

Les comparaisons et tests d'égalité sont des expressions ordinaires, qui comme les expressions arithmétiques produisent un résultat. Ce résultat est l'une des deux valeurs dites booléennes, l'une représentant le vrai (notée `True`) l'autre le faux (notée `False`).
Les opérateurs de comparaison sont :

- le test d'égalité qui s'écrit `==` et non `=`,
- la différence qui s'écrit `!=`,
- la comparaison qui peut s'écrire : `<`, `<=`, `>` ou `>=`.

De la même manière que l'ensemble des entiers vient avec des opérations arithmétiques, l'ensemble des booléens est associé à trois opérations booléennes permettant d'en combiner les valeurs :

- **`and`** : il faut que les deux opérateurs de comparaison soient évalués à `True`
- **`or`** : il faut qu'au moins un des deux opérateurs de comparaison soit évalué à `True`
- **`not`** : inverse le résultat.

Ces opérations booléennes permettent donc également de combiner plusieurs tests de comparaison et d'égalité dans une unique condition.
Remarque : lors de l'évaluation de `a or b` on sait que le résultat est vrai dès que la résultat de la proposition `a` est vérifiée. Python ne calculera pas le résultat de `b` dans ce cas. De même pour `a and b`, si `a` est faux Python renvoie `False` sans chercher à vérifier `b`. On parle d'évaluation paresseuse.

I.2 Structures conditionnelles

La structure conditionnelle la plus simple est composée par :

- l'instruction **`if`** suivi d'une expression booléenne qui se termine par `:`,
- un bloc d'instructions indenté, ces instructions ne seront exécutées que si l'expression booléenne est évaluée avec la valeur `True`

```
def bac(note) :
    if note >= 10 :
        return 'admis'
```

Si le test permet de discriminer entre deux blocs d'instructions alors la structure conditionnelle s'écrit de la manière suivante :

- l'instruction **`if`** suivi d'une expression booléenne puis de `:`,
- un bloc d'instructions indenté, ces instructions ne seront exécutées que si l'expression booléenne est évaluée avec la valeur `True`,

- l'instruction **`else`** au même niveau que **`if`** suivi de `:`,
- un nouveau bloc d'instructions indenté qui sera effectué si l'expression booléenne est évalué avec la valeur `False`.

```
def bac2(note) :
      if note >= 10 :
            return 'admis'
      else :
            return 'non admis'
```

Si il y a plus de deux cas, il faudra séparer ces différents cas.
La structure conditionnelle s'écrit alors :

- l'instruction **`if`** suivi d'une expression booléenne puis de `:`,
- un bloc d'instructions indenté, ces instructions ne seront exécutées que si l'expression booléenne est évaluée avec la valeur `True`,
- un certain nombre d'instructions **`elif`** au même niveau que **`if`** suivies d'une expression booléenne puis de `:`,
- pour chaque instruction un bloc indenté qui sera effectué si l'expression booléenne vaut `True` et les précédentes valent `False`,
- l'instruction **`else`** au même niveau que **`if`** suivi de `:`,
- un nouveau bloc d'instructions indenté qui sera effectué si toutes les expressions booléennes précédentes sont évaluées avec la valeur `False`

```
def mentionbac(note) :
      if note < 10 :
            return 'non admis'
      elif note < 12 :
            return 'admis sans mention'
      elif note < 14 :
            return 'admis mention assez bien'
      elif note < 16 :
            return 'admis mention bien'
      else :
            return 'admis mention très bien'
```

II Exercices

Exercice 1 - Valeur absolue
Écrire une fonction qui renvoie la valeur absolue d'un nombre x.

Exercice 2 - Signe d'un nombre
Écrire une fonction qui renvoie -1, 0 ou 1 selon que le nombre x en entrée est négatif, nul ou positif.

Exercice 3 - Suite
On définit une suite(un) par son premier terme u_0 et par la relation :

$$u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$$

Écrire une fonction `suite(u0, n)` qui calcule u_n.

Exercice 4 - Pair ou impair
Écrire une fonction pair(n) qui renvoie `True` si n est pair, `False` dans le cas contraire.
crire une fonction impair(n) qui renvoie `True` si n est impair, `False` dans le cas contraire.

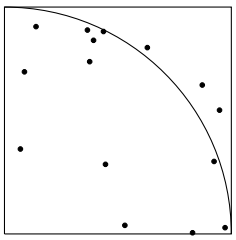
Exercice 5 - Année bissextile
Écrire une fonction `bissextile(annee)` qui renvoie `True` ou `False` selon que l'année donnée en entrée sous forme d'un entier est ou n'est pas bissextile.
Une année est dite bissextile si c'est un multiple de 4, sauf si c'est un multiple de 100. Toutefois si c'est un multiple de 400, alors elle est considérée comme bissextile.

Exercice 6 - Jours dans le mois
Écrire une fonction `jours(mois, annee)` qui renvoie le nombre de jours du mois donné pour son numéro (entre 1 et 12) , le paramètre année sert à déterminer le nombre de jours du mois de février (mois 2) selon que l'année est bissextile ou non.

Exercice 7 - Équation du second degré
Écrire une fonction `racinesReelles(a, b, c)` qui renvoie les racines réelles de l'équation du second degré $aX^2 + bX + c = 0$. La fonction devra renvoyer un triplet :

- $(2, r_1, r_2)$ si le polynôme admet deux racines réelles r_1 et r_2,
- $(1, r, r)$ si le polynôme admet une racine double r,
- $(0, 0, 0)$ si le polynôme n'admet pas de racine réelle.

Exercice 8 - Nombre premier
Un nombre $n \geq 2$ est **premier** s'il n'admet d'autre diviseur que 1 et lui-même c'est-à-dire si et seulement si il n'admet aucun diviseur entre 2 et $n - 1$.
Écrire une fonction premier(n) qui renvoie `True` ou `False` selon que n est premier ou non.



La méthode de Monte-Carlo permet d'approcher l'aire d'une surface incluse dans un rectangle. Pour cela on choisit n points aléatoirement dans le rectangle et on note r la proportion de ceux qui sont dans la surface L'aire de la surface sera approchée par $r.S$ où S est l'aire du rectangle. Dans le cas d'un quart de cercle inclus dans le carré unité, on calcule ainsi une valeur approchée de $\frac{\pi}{4}$. Dans l'exemple ci-contre on peut approcher π par $4.12/15 = 3{,}2$
La fonction `random()` du module `random` renvoie une valeur aléatoire choisie entre 0 et 1.

Exercice 9 - Méthode de Monte-Carlo
Écrire une fonction `approx_pi(n)` qui utilise cette méthode pour déterminer une valeur approchée de π en utilisant n points dans le carré unité.

III Compléments

Exercice 10
Écrire un programme qui demande trois longueurs à l'utilisateur (nombres entiers), indique si ces trois longueurs peuvent être les longueurs des trois côtés d'un triangle et, le cas échéant, s'il s'agit d'un triangle équilatéral, isocèle ou scalène (trois côtés de longueurs différentes). Indiquer également si le triangle est rectangle.

Pour demander quelque chose à l'utilisateur, on utilise l'instruction `input("demande :")`.

III.1 Triangles rectangles

On cherche les triangles rectangles dont les longueurs des côtés sont des entiers. Plus précisément on cherche les entiers a, b et c tels que $a \leq b$ et $a^2 + b^2 = c^2$. On dit que (a, b, c) est un triplet pythagoricien.

Exercice 11
Écrire une fonction `pythagoriciens(n)` qui renvoie le nombre de triplets pythogoriciens (a, b, c) avec $1 \leq a \leq n$, $1 \leq b \leq n$ et $1 \leq c \leq n$.

III.2 Bataille navale

On considère un jeu de bataille navale simplifié : il ne s'agit pas de couler le porte-avions mais plutôt une barque tenant sur une seule case repérée par ses coordonnées ligne0 et colonne0 qui sont des variables globales. On fait un tir sur une case repérée par ses coordonnées ligne et colonne. On veut alors le comportement suivant :

- si la barque est exactement sur la case considérée, le programme écrit le texte "Coulé"
- si le tir atteint la bonne ligne ou la bonne colonne, le programme écrit "En vue",
- Si le tir est totalement raté, le programme écrit "À l'eau".

On propose la fonction suivante :

```
def bataille(ligne, colonne) :
    if ligne == ligne0 or colonne == colonne0 :
        print("En vue")
    elif ligne ==ligne0 and colonne == colonne0 :
        print("Coulé")
    else :
        print("À l'eau")
```

Exercice 12
Pourquoi cette fonction n'est pas correcte ?
Proposer une fonction valide.

En général, à la bataille navale, un bateau n'est "En vue" que si la case visée est immédiatement voisine de celle du bateau.

Exercice 13
Modifier le programme de bataille navale afin de tenir compte de cette règle.

IV Solutions

Solution de l'exercice 1 - Valeur absolue

```
def val_abs(x) :
    if x < 0 :
        return -x
    else :
        return x
```

Solution de l'exercice 2 - Signe d'un nombre

```
def signe(x) :
    if x < 0 :
        return -1
    elif x ==0 :
        return 0
    else :
        return 1
```

Solution de l'exercice 3 - Suite

```
def suite(u0, n) :
    u = u0
    for i in range(n) :
        if n%2 == 0 :
            u = u//2
        else :
            u = 3*u + 1
    return u
```

Solution de l'exercice 4 - Pair ou impair

```
def pair(n) :
    if n%2 == 0 :
        return \lstinline{True}
    else :
        return \lstinline{False}

def impair(n) :
    if n%2 != 0 :
        return \lstinline{True}
    else :
        return \lstinline{False}
```

Solution de l'exercice 5 - Année bissextile

```
def bissextile(annee) :
    if annee%400 == 0 :
        return \lstinline{True}
    elif annee%100 == 0 :
        return \lstinline{False}
    elif annee%4 == 0 :
        return \lstinline{True}
    else :
        return \lstinline{False}
```

Solution de l'exercice 6 - Jours dans le mois

```
def jours(mois, annee) :
    """les mois sont donnés par leur numéro
       (entre 1 et 12)"""
    if mois == 2 :
        if bissextile(annee) :
            return 29
        else :
            return 28
    elif mois == 4 or mois ==  6 or mois == 9 or mois == 11 :
        return 30
    else :
        return 31
```

Solution de l'exercice 7 - Équation du second degré

```
def racinesReelles(a, b, c) :
    """entrées : les 3 coefficients du polynôme
                 aX**2 + bX + c = 0
        sortie : le nombre de racines réelles
                 et leurs valeurs"""
    delta = b**2 - 4*a*c
    if delta > 0 :
        return (2, (-b + delta)/(2*a), (-b - delta)/(2*a))
    elif delta ==0 :
        return (1, -b/(2*a), -b/(2*a))
    else :
        return (0, 0, 0)
```

Solution de l'exercice 8 - Nombre premier

```
def premier(n) :
    max = int(n**0.5)
    for i in range(2, max+1) :
        if n%i ==0 :
            return False
    return True
```

Solution de l'exercice 9 - Méthode de Monte-Carlo

```
from random import random

def approx_pi(n) :
    dedans = 0
    for i in range(n) :
        x = random()
        y = random()
        if x**2 + y**2 < 1 :
            dedans = dedans +1
    return (dedans/n)*4
```

Solution de l'exercice 10

Étant données les trois longueurs, il s'agit d'un triangle si aucune n'est strictement supérieure à la somme des deux autres.

```
a = int(input("entrer la première longueur :"))
b = int(input("entrer la deuxième longueur :"))
c = int(input("entrer la troisième longueur :"))

if a+b >= c and a+c >= b and b+c >= a :
    print ("ceci est un triangle")
    if a == b and a ==c :
        print("équilatéral")
    elif a ==b or a == c or b == c :
        print("isocèle")
    else :
        print("scalène")
    if a**2+b**2==c**2 or a**2+c**2==b**2 or b**2+c**2==a**2 :
        print("rectangle")
else :
    print("ceci n'est pas un triangle")
```

Solution de l'exercice 11

```
def pythagoriciens(n) :
    nombre = 0
    for a in range(1, n+1) :
        for b in range (1, n+1) :
            for c in range(2, n+1) :
                if a**2+ b**2 == c**2 :
                    nombre = nombre + 1
    return nombre
```

pour aller plus vite on peut définir une autre fonction

```
def pythagoriciens1(n) :
    nombre = 0
    for a in range(1, n+1) :
        for b in range(a, n+1) :
            r = (a**2 + b**2)**0.5
            if r == int(r) and r <= n :
                nombre = nombre + 1
    return nombre
```

Solution de l'exercice 12

Le programme ne peut jamais renvoyer "Coulé".

```
def bataille(ligne, colonne) :
    if ligne == ligne0 and colonne == colonne0 :
        print("Coulé")
    elif ligne ==ligne0 or colonne == colonne0 :
        print("En vue")
    else :
        print("À l'eau")
```

Solution de l'exercice 13

```
def bataille(ligne, colonne) :
    if ligne == ligne0 and colonne == colonne0 :
        print("Coulé")
    elif (abs(ligne - ligne0) + abs(colonne - colonne0) <= 1 :
        print("En vue")
    else :
        print("À l'eau")
```