TP n° 5 : Instructions conditionnelles

1 Instructions conditionnelles

1.1 Motivations

Les *instructions conditionnelles* permettent de faire des choix dans un programme, c'est-à-dire d'altérer le déroulement du programme en fonction de la valeur de conditions. Elle correspondent aux mots-clés « *si* » (`if`), « *sinon, si* » (`elif`; contraction de `else if`) et « *sinon* » (`else`).

Par exemple, on peut vouloir traduire dans un programme le fait que s'il fait beau on choisit d'aller pique-niquer alors que s'il pleut on préfère aller au cinéma. Dans les autres cas (neige, grisaille, etc.), on reste chez soi.

PYTHON
```
if beau_temps:
    sortie = "Pique-nique"
elif pluie:
    sortie = "Cinéma"
else:
    sortie = "Aucune"
```

1.2 Syntaxe

La syntaxe générale est :

PYTHON
```
if condition1:
    # bloc1 qui s'exécute
    # si la condition1 est vérifié
elif condition2:
    # bloc2 qui s'exécute
    # si la condition1 n'est pas vérifiée
    # mais la condition2 l'est
elif condition3:
    # bloc3 qui s'exécute
    # si les condition 1 et 2 ne sont pas vérifiées
    # mais la condition3 l'est
...
...
else:
    # bloc qui s'exécute
    # si aucune condition n'est vérifiée
```

Les clauses `elif` et la clause `else` sont facultatives. Par exemple :

PYTHON
```
if 0 != 0:
    print("Le monde est absurde.")
```

Mais attention à ne pas faire n'importe quoi !

PYTHON
```
reponse = 42
if reponse % 5 == 0:
    question = "Proposer un multiple de 5."
elif reponse < 0:
    question = "Proposer un nombre positif."

print(question)
# NameError: name 'question' is not defined
```

1.3 Blocs d'indentation

✔ Comme pour les fonctions, les blocs sont introduits par un deux-points « : » et définis par l'*indentation*. Une indentation est constituée de quatre espaces.

PYZO affiche les guides d'indentations. Si vous avez souvent des erreurs, dans un premier temps, vous pouvez aussi afficher explicitement les espaces dans le menu *Affichage* > *Afficher les espaces*.

⚠ Au sein d'un bloc, le niveau d'indentation doit être le même !

PYTHON
```
if 0 == 0:
    print("Tout va bien en mathématiques")
   print("Mais pas en informatique")
# IndentationError: unindent does not match
# any outer indentation level
```

1.4 Instructions conditionnelles et fonctions

On peut utiliser un bloc d'instructions conditionnelles dans une fonction :

PYTHON
```
def partie_positive(x):
    """Partie positive d'un réel."""
    if x > 0:
        x_plus = x
    else:
        x_plus = 0.
    return x_plus
```

Remarquer que les affectations de la variable `x_plus` sont situées dans des blocs de deux niveaux d'indentation : un pour le corps de la fonction, puis un pour le corps de l'instruction conditionnelle, soit 8 espaces.

L'instruction `return` arrête l'exécution de la fonction et renvoie la valeur de retour correspondante. On peut donc aussi écrire [1] :

PYTHON
```
def partie_positive(x):
    """Partie positive d'un réel."""
    if x > 0:
        return x
    else:
        return 0.
```

Question 1

Définir une fonction `partie_negative` qui renvoie la partie négative x^- d'un réel x, c'est-à-dire x si $x > 0$ et 0 sinon.

Erreurs possibles (à retenir pour la suite) :

- `IndentationError: unindent does not match ...` : c'est explicitement un problème d'indentation !
- `SyntaxError: invalid syntax` : avez-vous oublié un deux-points ?
- `SyntaxError: invalid syntax` : avez-vous ajouté une condition dans la clause `else` ? Relisez votre programme à voix haute en remplaçant `else` par « *dans tous les autres cas* ».

Question 2

Que fait la fonction suivante ?

PYTHON
```
def mystere(x, y):
    """Mystère..."""
    if x >= y:
        return x
    else:
        return y
```

1. Cette construction est très classique en PYTHON et je vous encourage à l'utiliser, même si c'est contraire aux principes fondamentaux de la programmation structurée (une fonction doit posséder un seul point d'entrée et un seul point de sortie).

Question 3

Écrire une fonction `min2(a, b)` qui prend en arguments deux nombres `a` et `b` et qui renvoie leur minimum.

On demande d'écrire une suite d'instructions qui effectue la transformation suivante : on part d'un entier $n > 0$, s'il est pair, on le divise par deux, sinon on le multiplie par trois et on lui ajoute un. L'élève Marius propose la solution suivante :

PYTHON
```
if n % 2 == 0:
    n = n // 2
if n % 2 == 1:
    n = 3*n + 1
```

Question 4

Quelle est la valeur de n à la fin du programme si au départ $n = 2$? Quelle devrait être la valeur de n? Expliquer. .....
.....
.....
.....
.....

Question 5

Corriger ce programme pour qu'il effectue ce qui est demandé.

En répétant l'opération, on obtient une suite d'entiers positifs dont chacun ne dépend que de son prédécesseur, appelée suite de Syracuse de n (le terme initial). La conjecture de Syracuse (ou conjecture de Collatz, conjecture d'Ulam, conjecture tchèque ou encore problème $3x+1$), est l'hypothèse mathématique selon laquelle la suite de Syracuse de n'importe quel entier strictement positif atteint 1. En dépit de la simplicité de son énoncé, cette conjecture défie [2] depuis de nombreuses années les mathématiciens. Le grand mathématicien Paul ERDŐS a dit à propos de la conjecture de Syracuse : « *les mathématiques ne sont pas encore prêtes pour de tels problèmes* ».

2. Cette conjecture mobilisa tant les mathématiciens durant les années 1960, en pleine guerre froide, qu'une plaisanterie courut selon laquelle ce problème faisait partie d'un complot soviétique visant à ralentir la recherche américaine.

1.5 Test imbriqués

De même, on peut utiliser un bloc de test comme bloc d'instruction dans test. Par exemple, on peut ne pas vouloir continuer l'opération précédente sur la suite de Syracuse si on atteint 1.

PYTHON
```
def s(n):
    """Renvoie le terme suivant dans la suite de Syracuse,
       mais en stagnant en 1."""
    if n > 1:
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3*n + 1
    return n

print(s(s(s(s(s(5))))))
```

Dans la mesure du possible, il convient d'éviter les tests imbriqués car ils compliquent la lisibilité d'un programme. En particulier, il vaut mieux utiliser l'instruction `elif` qu'un `if` imbriqué dans un `else`.

Non

PYTHON
```
if x % 3 == 0:
    # Cas 0
else:
    if x % 3 == 1:
        # Cas 1
    else:
        # Cas 2
```

Oui

PYTHON
```
if x % 3 == 0:
    # Cas 0
elif x % 3 == 1:
    # Cas 1
else:
    # Cas 2
```

2 Exercices

EXERCICE 1

Écrire une fonction `division(a, b)` qui prend en arguments deux nombres `a` et `b` et qui renvoie la valeur `a / b` si elle existe et `None` sinon.

EXERCICE 2

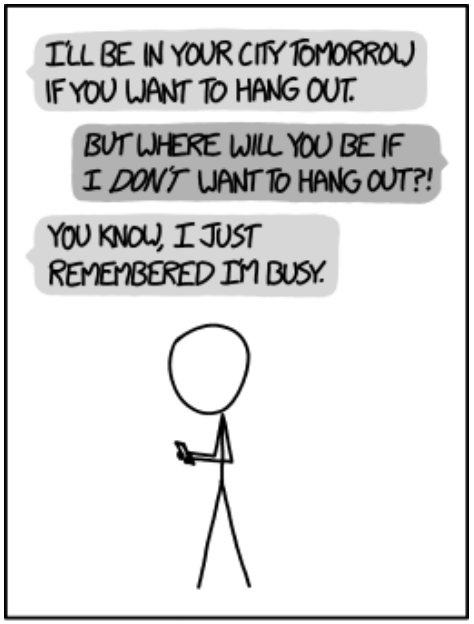
Écrire une fonction `valeur_absolue` qui renvoie la valeur absolue d'un nombre (sans utiliser la fonction `abs` bien sûr).

EXERCICE 3

Écrire une fonction `max3(a, b, c)` qui prend en arguments trois nombres `a`, `b` et `c` et qui renvoie leur maximum. Même question avec `min3`.

✔

Les fonctions `min` et `max` sont prédéfinies en PYTHON et peuvent prendre un nombre quelconque d'arguments. Vous pouvez les utiliser librement, sauf s'il vous semble que l'on demande de les implémenter.



WHY I TRY NOT TO BE PEDANTIC ABOUT CONDITIONALS.

https://xkcd.com/1652/