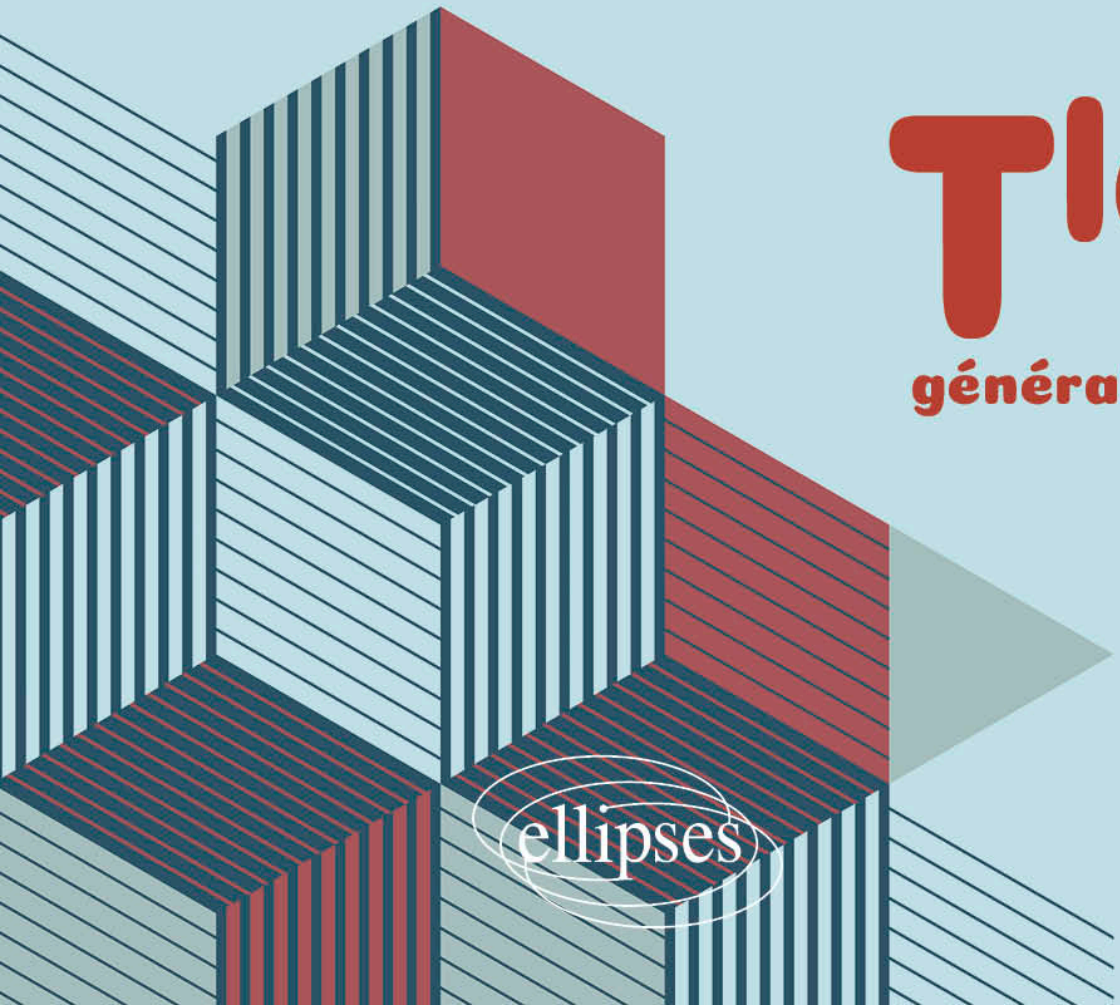
ANNALES

SPÉCIALITÉ NSI

Tle générale



ellipses

CHAPITRE 1
PROGRAMMATION ET ALGORITHMIQUE

Exercice 1 : Récursivité

POLYNÉSIE 2022, JOUR 1, EXERCICE N°1 (4 POINTS)

Cet exercice traite du thème « programmation », et principalement de la récursivité.

On rappelle qu'une chaîne de caractères peut être représentée en Python par un texte entre guillemets "" et que :

- la fonction `len` renvoie la longueur de la chaîne de caractères passée en paramètre ;
- si une variable `ch` désigne une chaîne de caractères, alors `ch[0]` renvoie son premier caractère, `ch[1]` le deuxième, etc. ;
- l'opérateur + permet de concaténer deux chaînes de caractères.

Exemples :

```
>>> texte= "bricot"
>>> len(texte)
6
>>> texte[0]
"b"
>>> texte[1]
"r"
>>> "a" + texte
"abricot"
```

On s'intéresse dans cet exercice à la construction de chaînes de caractères suivant certaines règles de construction.

Règle A : une chaîne est construite suivant la règle A dans les deux cas suivants :

- soit elle est égale à "a" ;
- soit elle est de la forme "a"+chaine+"a", où chaine est une chaîne de caractères construite suivant la règle A.

Règle B : une chaîne est construite suivant la règle B dans les deux cas suivants :

- soit elle est de la forme "b"+chaine+"b", où chaine est une chaîne de caractères construite suivant la règle A ;
- soit elle est de la forme `"b"+chaine+"b"`, où `chaine` est une chaîne de caractères construite suivant la règle B.

On a reproduit ci-dessous l'aide de la fonction `choice` du module `random`.

```
>>>from random import choice
>>>help(choice)
Help on method choice in module random:
choice(seq) method of random.Random instance
    Choose a random element from a non-empty sequence.
```

La fonction `A()` ci-dessous renvoie une chaîne de caractères construite suivant la règle A, en choisissant aléatoirement entre les deux cas de figure de cette règle.

```
def A():
    if choice([True, False]):
        return "a"
    else:
        return "a" + A() + "a"
```

1. Cette fonction est-elle récursive ? Justifier.

Définition

L'appel d'une fonction est la commande qui demande l'exécution de cette fonction. Ici il s'agit de la commande `A()`.

Une fonction est récursive si elle s'appelle elle-même à l'intérieur de sa propre définition.

Solution

La fonction `A` est une fonction récursive, car elle s'appelle elle-même à la dernière ligne : `return "a" + A() + "a"`.

2. La fonction `choice([True, False])` peut renvoyer `False` un très grand nombre de fois consécutives. Expliquer pourquoi ce cas de figure amènerait à une erreur d'exécution.

Rappel sur les capacités de stockage lors des appels récursifs

Lorsqu'on effectue les appels récursifs de la fonction `A()`, les calculs sont stockés dans un objet de type `pile`. Or dans une pile on place les nouveaux objets en haut de la pile et on a accès seulement au dernier objet placé. C'est seulement lorsqu'on obtient la valeur `True` pour la fonction `choice` que l'on connaît la valeur de `A()`, `"a"`, et qu'on peut accéder aux valeurs de la pile situées en dessous. Le problème c'est que la pile utilisée pour le stockage, la pile d'exécution, a une taille limitée. Lorsqu'on la dépasse, il y a une erreur de dépassement de la pile, `stack overflow`.

Solution

Si la fonction `choice` renvoie systématiquement `False`, les appels récursifs seraient trop nombreux (dépassement de la taille de la pile maximale fixée par Python), nous aurions alors une erreur d'exécution ('recursionError').

Dans la suite, on considère une deuxième version de la fonction A. À présent, la fonction prend en paramètre un entier `n` tel que, si la valeur de `n` est négative ou nulle, la fonction renvoie "a". Si la valeur de `n` est strictement positive, elle renvoie une chaîne de caractères construite suivant la règle A avec un `n` décrémenté de 1, en choisissant aléatoirement entre les deux cas de figure de cette règle.

```
def A(n):
    if ... or choice([True, False]) :
        return "a"
    else:
        return "a" + ... + "a"
```

3. Recopier sur la copie et compléter aux emplacements des points de suspension ... le code de cette nouvelle fonction A.

Analyse du code

La ligne qui suit la condition `if` est return `"a"`, et l'énoncé indique que, « si la valeur de `n` est négative ou nulle, la fonction renvoie "a" ». La condition est donc `n<=0` qui doit remplacer les …. La valeur alternative de la fonction, pour `n>0`, doit renvoyer « une chaîne de caractères construite suivant la règle A avec un `n` décrémenté de 1 », soit `A(n-1)`, qui doit remplacer les derniers …

Solution

```
def A(n):
    if n <= 0 or choice([True, False]) :
        return "a"
    else:
        return "a" + A(n-1) + "a"
```

4. Justifier le fait qu'un appel de la forme A(n) avec n un nombre entier positif inférieur à 50, termine toujours.

Solution

Dans le cas où la fonction `choice` renvoie systématiquement `False`, nous aurons 50 appels récursifs puisque pour le 51e appel de la fonction `A` nous aurons `n` qui sera égal à zéro, le `return "a"` sera donc obligatoirement exécuté (présence du `or` dans le `if`), ce qui entraînera la fin de l'exécution du programme.

On donne ci-après le code de la fonction récursive `B` qui prend en paramètre un entier `n` et qui renvoie une chaîne de caractères construite suivant la règle B.

```
def B(n):
    if n <= 0 or choice([True, False]):
        return "b" + A(n-1) + "b"
    else:
        return "b" + B(n-1) + "b"
```

On admet que :

- les appels `A(-1)` et `A(0)` renvoient la chaîne "a";
- l'appel `A(1)` renvoie la chaîne "a" ou la chaîne "aaa";
- l'appel `A(2)` renvoie la chaîne "a", la chaîne "aaa" ou la chaîne "aaaaa".

5. Donner toutes les chaînes possibles renvoyées par les appels `B(0)`, `B(1)` et `B(2)`.

Vers la solution

L'appel B(0) renvoie `"b" + A(-1) + "b"` et A(-1) renvoie `"a"`, donc B(0) renvoie `"bab"`.

L'appel B(1) renvoie `"bab"` si `choice` prend la valeur `True`, ou `"b" + B(0) + "b"`, soit "bbabb" si la fonction `choice` prend la valeur `False`, donc B(1) renvoie "bab" ou "bbabb".

L'appel B(2) renvoie "b" + A(1) + "b" si choice prend la valeur True ou "b" + B(1) + "b" dans le cas contraire. Comme A(1) renvoie "a" ou la chaîne "aaa", la première valeur de choice amène B(2) à prendre la valeur "bab" ou "baaab". D'après le résultat de la valeur de B(1), la deuxième valeur possible de choice amène B(2) à prendre les valeurs "bbabb" ou "bbbabbb". Ainsi les 4 valeurs possibles de B(2) sont "bab", "baaab", "bbabb" ou "bbbabbb".

Solution

B(0) renvoie `"bab"`, `B(1) renvoie` "bab" ou "bbabb" et B(2) renvoie "bab", "baaab", "bbabb" ou "bbbabbb".

On suppose maintenant qu'on dispose d'une fonction `raccourcir` qui prend comme paramètre une chaîne de caractères de longueur supérieure ou égale à 2, et renvoie la chaîne de caractères obtenue à partir de la chaîne initiale en lui ôtant le premier et le dernier caractère.

Par exemple :

```
>>> raccourcir("abricot")
"brico"
>>> raccourcir("ab")
""
```

6. Recopier sur la copie et compléter les points de suspension ... du code de la fonction `regleA` ci-dessous pour qu'elle renvoie `True` si la chaîne passée en paramètre est construite suivant la règle A, et `False` sinon.

```
def regleA(chaine):
    n = len(chaine)
    if n >= 2:
        return chaine[0] == "a" and chaine[n-1] == "a" and
regleA(...)
    else:
        return chaine == ...
```

Vers la solution

Il s'agit d'une fonction récursive puisqu'à la 5e ligne il y a `regleA(…)`.

Le renvoi (`return`) de la ligne 4 est un booléen qui correspond à la condition à la construction de la chaîne de caractères lorsque la longueur de la chaîne de caractères est supérieure ou égale à 2 (`n >= 2`). Dans ce cas, « elle est

de la forme "a"+chaine+"a", où chaine est une chaîne de caractères construite suivant la règle A ». Lorsqu'une chaîne a une longueur n >=2, et qu'on enlève le premier et le dernier caractères, la chaîne se situant entre ces deux caractères s'obtient grâce à la fonction raccourcir. Il s'agit de raccourcir(chaine). Il faut donc remplacer les … par raccourcir(chaine).

Le dernier renvoi de la fonction correspond au cas où la chaîne est de longueur 1. Dans ce cas la chaîne doit être "a".

Solution

```
def regleA(chaine):
    n = len(chaine)
    if n >= 2:
        return chaine[0] == "a" and chaine[n-1] == "a" and\
        regleA(raccourcir(chaine))
    else:
        return chaine == "a"
```

7. Écrire le code d'une fonction regleB, prenant en paramètre une chaîne de caractères et renvoyant True si la chaîne est construite suivant la règle B, et False sinon.

Vers la solution

On va écrire la fonction regleB sur le même modèle que la fonction regleA. Lorsque la chaîne est de longueur supérieure ou égale à 2, elle est construite suivant « la règle B dans les deux cas suivants :

- soit elle est de la forme "b"+chaine+"b", où chaine est une chaîne de caractères construite suivant la règle A ;
- soit elle est de la forme "b"+chaine+"b", où chaine est une chaîne de caractères construite suivant la règle B. »
- Dans ces deux cas, chaine entre les deux "b" s'obtient avec la fonction raccourcir, soit en suivant la règle A, soit en suivant la règle B.
- Sinon elle n'est pas construite suivant la règle B.

Solution

```
def regleB(chaine):
    n = len(chaine)
    if n >= 2:
        return chaine[0] == "b" and chaine[n-1] == "b" and
(regleA(raccourcir(chaine)) or regleB(raccourcir(chaine)))
    else:
        return False
```

ON RETROUVE LE THÈME RÉCURSIVITÉ DANS LES EXERCICES AUX PAGES SUIVANTES DES ANNALES 42, 88, 103, 111, 123, 132, 159, 173, 176, 208, 237, 245

Exercice 2 : Programmation orientée objet

MÉTROPOLE 2022, JOUR 1, EXERCICE N°5 (4 POINTS)

Cet exercice porte sur la Programmation Orientée Objet.

Les participants à un jeu de `LaserGame` sont répartis en équipes et s'affrontent dans ce jeu de tir, revêtus d'une veste à capteurs et munis d'une arme factice émettant des infrarouges.

Les ordinateurs embarqués dans ces vestes utilisent la programmation orientée objet pour modéliser les joueurs. La classe `Joueur` est définie comme suit :

```
class Joueur:
      def __init__(self, pseudo, identifiant, equipe):
            """ constructeur """
            self.pseudo = pseudo
            self.equipe = equipe
            self.id = identifiant
            self.nb_de_tirs_emis = 0
            self.liste_id_tirs_recus = []
            self.est_actif = True

      def tire(self):
            """méthode déclenchée par l'appui sur la gâchette"""
            if self.est_actif == True:
                  self.nb_de_tirs_emis = self.nb_de_tirs_emis + 1
      def est_determine(self):
            """méthode qui renvoie True si le joueur réalise un
grand nombre de tirs"""
            return self.nb_de_tirs_emis > 500
      def subit_un_tir(self, id_recu):
            """méthode déclenchée par les capteurs de la veste"""
            if self.est_actif == True:
                  self.est_actif = False
                  self.liste_id_tirs_recus.append(id_recu)
```

1. Parmi les instructions suivantes, recopier celle qui permet de déclarer un objet `joueur1`, instance de la classe `Joueur`, correspondant à un joueur dont le pseudo est "Sniper", dont l'identifiant est 319 et qui est intégré à l'équipe "A" :

Instruction 1 : joueur1 = ["Sniper", 319, "A"]
Instruction 2 : joueur1 = new Joueur["Sniper", 319, "A"]
Instruction 3 : joueur1 = Joueur("Sniper", 319, "A")
Instruction 4 : joueur1 = Joueur{"pseudo":"Sniper", "id":319, "equipe":"A"

Vers la solution

Pour déclarer une instance de classe en Python, on utilise le nom de la classe comme une fonction, ici `Joueur`, et on met comme arguments (entre

parenthèses) les arguments du constructeur, c'est-à-dire la fonction `__init__` de la classe, ici `(pseudo, identifiant, equipe)`.
La bonne réponse est donc l'instruction 3.

Solution

L'instruction correspondant à un joueur dont le pseudo est "Sniper", dont l'identifiant est 319 et qui est intégré à l'équipe "A" et l'instruction 3 :
joueur1 = Joueur("Sniper", 319, "A")

2. La méthode `subit_un_tir` réalise les actions suivantes :
Lorsqu'un joueur actif subit un tir capté par sa veste, l'identifiant du tireur est ajouté à l'attribut `liste_id_tirs_recus` et l'attribut `est_actif` prend la valeur `False` (le joueur est désactivé). Il doit alors revenir à son camp de base pour être de nouveau actif.

a) Écrire la méthode `redevenir_actif` qui rend à nouveau le joueur actif uniquement s'il était précédemment désactivé.

Rappel sur la programmation orientée objet

Une méthode d'un objet est une fonction de la classe objet qui va s'appliquer aux instances de l'objet.
Dans le code de la méthode, le premier argument sera toujours le mot-clé `self` qui désigne l'objet lui-même et qui va permettre d'avoir accès à tous les attributs de l'objet. Les autres arguments de la méthode s'écrivent à la suite comme lorsque l'on définit une fonction habituelle.
Pour désigner un attribut du joueur, on utilise le point. `self.attribut` désigne l'attribut pour ce joueur.

Vers la solution

L'en-tête de la méthode ne contient comme argument que le mot-clé self car il n'y a pas besoin d'arguments extérieurs aux attributs d'un joueur.
La seule chose que doit faire cette méthode c'est de passer l'attribut `est_actif` du joueur de `False` à `True`. L'accès à cet attribut s'effectue comme suit `self.est_actif`, d'où le code suivant qui s'insère dans la classe `Joueur`.

Solution

```
def redevenir_actif(self):
    if not self.est_actif :
        self.est_actif = True
```

b) Écrire la méthode `nb_de_tirs_recus` qui renvoie le nombre de tirs reçus par un joueur en utilisant son attribut `liste_id_tirs_recus`.

Vers la solution

L'attribut `liste_id_tirs_recus` permet de stocker dans une liste les id des joueurs qui ont touché le joueur par un tir. La longueur de cette liste permet de connaître le nombre de tirs reçus par le joueur. D'où le code suivant qui s'insère dans la classe `Joueur`.

Solution

```
def nb_de_tirs_recus(self):
    return len(self.liste_id_tirs_recus)
```

3. Lorsque la partie est terminée, les participants rejoignent leur camp de base respectif où un ordinateur, qui utilise la classe `Base`, récupère les données.

La classe `Base` est définie par :

- ses attributs :
  - `equipe` : nom de l'équipe (`str`). Par exemple, "A",
  - `liste_des_id_de_l_equipe` qui correspond à la liste (`list`) des identifiants connus des joueurs de l'équipe,
  - `score` : score (`int`) de l'équipe, dont la valeur initiale est 1000 ;
- ses méthodes :
  - `est_un_id_allie` qui renvoie `True` si l'identifiant passé en paramètre est un identifiant d'un joueur de l'équipe, `False` sinon,
  - `incremente_score` qui fait varier l'attribut score du nombre passé en paramètre,
  - `collecte_information` qui récupère les statistiques d'un participant passé en paramètre (instance de la classe Joueur) pour calculer le score de l'équipe.

```
def collecte_information(self,participant):
    if participant.equipe == self.equipe : # test 1
        for id in participant.liste_id_tirs_recus:
            if self.est_un_id_allie(id): # test 2
                self.incremente_score(-20)
            else:
                self.incremente_score(-10)
```

a) Indiquer le numéro du test (test 1 ou test 2) qui permet de vérifier qu'en fin de partie un participant égaré n'a pas rejoint par erreur la base adverse.

Vers la solution

`participant.equipe` restitue l'attribut `equipe` du participant passé en paramètre et `self.equipe` restitue l'attribut `equipe` de la base, ainsi le test 1 indique que le participant a rejoint la base adverse par erreur.

Solution

Test 1

b) Décrire comment varie quantitativement le score de la base lorsqu'un joueur de cette équipe a été touché par le tir d'un coéquipier.