Cours n°2
30 septembre 2020

Python :
retour sur les tests ,
premières boucles

Rappel de l'exercice sur Zorglub:

Les habitants de Zorglub paient l'impôt selon les règles suivantes :

· les hommes de plus de 20 ans paient l'impôt

· les femmes paient l'impôt si elles ont entre 18 et 35 ans

· les autres ne paient pas d'impôt

Le programme demandera donc l'âge et le sexe du Zorglubien, et se prononcera donc ensuite sur le fait que l'habitant est imposable.

Une correction

```
s=input("homme ou femme?  ")
a=int(input ("votre age?   "))  # ne pas oublier int
if s=="homme":
    if a> 20:
        print ("impot à payer")
    else:
        print ("pas d'impot")
elif s=="femme":
        if 18<a and a<35:
            print ("impot à payer")
        else:
            print ("pas d'impot")
Else:  # ces 2 dernières lignes si on veut traiter les erreurs de saisie sur homme/femme
        print ("je n'ai pas compris votre réponse”)
```

Une autre version

On peut raccourcir l'écriture and et or

```
s=input("homme ou femme?  ")
a= int(input ("votre age?   "))
if (s=="homme" and a>20) or (s=="femme" and 18<a and a<35):
    print ("impot à payer")
elif s!="homme" and s!="femme":
    print ("je n'ai pas compris votre réponse")
else:
    print ("pas d'impot")
```

Là il faut des parenthèses pour les conditions car sinon ce n'est pas compréhensible

utilisation d'autres tests

On peut aussi utiliser des opérateurs et composer avec des tests : par exemple

```
if a%2 != 0:
    print (“a n'est pas pair”)

if a**2==b:
    print ('a est racine de b')
```

Remarque importante

Attention:

on utilise les tests directement:

```
if test :
```

et surtout pas en utilisant une syntaxe du genre:

```
if test==True :
```

Important: ordre des conditions:

Pour la résolution de l'équation ax+b=0

```
print ("resolution de ax+b=0")
a=float(input ("valeur de a ?"))
b=float(input ("valeur de b ?"))
if a!=0:
    print ((- b)/a)
elif b==0:  # là on sait que a=0
    print ("tout réel est solution")
else:  # là on sait que a=0 et b!= 0 pas besoin de test
    print ("pas de solution")
```

Important: ordre des conditions:

Autre possibilité d'écriture du programme pour la résolution de l'équation ax+b=0

```
print ("resolution de ax+b=0")
a=float(input ("valeur de a ?"))
b=float(input ("valeur de b ?"))
if a==0 and b==0:
    print ("tout réel est solution")
elif a==0:  # on sait que b!=0
    print ("pas de solution" )
else:       # on sait que a!=0
     print ((-b)/a)
```

Important: ordre des conditions:

Retour sur le triangle/tuple

```
if a+b+c!=180:
    print("pas un triangle")
elif a==b or a==b or b==c:
    print("isocèle")
elif a==b==c:
    print("équilatéral")
```

Là c'est faux!

Pas de conditions inutiles grâce à un ordre bien choisi

```
if a+b+c!=180 :
    print("pas un triangle")
elif 0 in triangle:
    print("triangle plat")
elif a==b==c:
    print("équilatéral")
elif a==b or a==b or b==c and 90 in triangle:
    print("isocèle rectangle")
elif a==b or a==b or b==c :
    print("isocèle ")
elif 90 in triangle:
    print('rectangle')
else:
    print("quelconque")
```

Retour sur les strings: les slices

sous-chaînes : à partir d'une chaîne x, on peut extraire une "tranche" de cette chaîne (une sous-chaîne) :

- x[n:p] renvoie la tranche de x comprise entre les indices n (inclus) et p (exclus).

On peut ne pas mettre les index:

- x[:p] du début à p exclus
- x[n:] de n inclus à la fin de la chaîne
- x[:] copie de la chaîne

Exemple pour la date: si d="30/09/2020"
d[:2] donne "30", d[3:5] donne "09" , d[6:] donne "2020"

Cours 2: première boucle en python

Des boucles en python

En python nous verrons 2 manières d'écrire des boucles :

→ for

C'est un parcours d'une séquence (chaîne, tuple, liste, fichier, dictionnaire, séquence d'entiers....)

→ while

C'est la boucle “tant que”: on peut tout écrire avec mais le for est parfois plus rapide à mettre en place

for

On aura des boucles du style:
pour i entre 1 et n   faire.....
pour x dans le tuple faire....
pour x dans la chaîne faire....

(et plus tard dans un dictionnaire, un fichier ouvert en lecture, un tableau....)

for <truc> in <sequence>:
    <bloc instructions1> # à chaque étape de la boucle
<bloc instructions2> # quand la boucle est finie

attention au : et à l'indentation

in

in permet parcourir une séquence ou de tester l'appartenance d'un élément à un tuple, à une chaîne....

```
>>>4 in (3,4,5):
True
>>> "4" in "56745"
True
```

On peut tester l'inclusion d'une sous-chaîne, mais pas d'un sous-tuple

```
>>> "34" in "346243"
True
>>> (2,3) in (1,2,3,4)  # le sous-tuple est inclus
False                   # mais il n'appartient pas
>>> (2,3) in (1,(2,3),4) # là il appartient
True
```

Remarque:

pour tester que x==3 ou x==4 ou x==5 ou x==6,
on peut utiliser: x in(3,4,5,6)

Nombre de jours d'un mois (hors année bissextile):

```
if m in (1,3,5,7,8,10,12):
    print (31)
elif m==2:
    print(28)
else:
   print 30
```

Exemples de boucles

→ parcourir un tuple de nombres et compter le nombre d'éléments

```
t=(1,3,5,7,9)
res=0     # on initialise un compteur
for x in t:
   res=res+1                  # on peut écrire res+=1  aussi
print (res)
```

Remarque: len(t) donne le même résultat

→ parcourir un tuple de nombres et faire la somme des éléments

```
t=(1,3,5,7,9)
som=0     # on initialise un compteur
for x in t:
   som= som+x                # on peut aussi écrire som+=x
print (som)
```

remarque

Dans tous les cas l'instruction
x+=truc
revient à x=x+truc

Attention:

```
>>> x=1
>>> x+
SyntaxError: invalid syntax
>>> x++              # pas possible en python
SyntaxError: invalid syntax
>>> x+=1   # ne renvoie rien mais x est modifié
>>> x
2
```

Encore des exemples

→ Parcourir une chaîne et compter le nombre d'espaces

```
esp=0
ch=" python est un langage formidable     "
for x in ch:
    if x==' ':
        esp+=1              # si x n'est pas ' ' on ne fait rien la boucle continue
print(esp)
```

Encore des exemples

→ ré-écrire une chaine entrecoupée d'étoiles

```
ch=”bonjour”
res="*"
for x in ch:
    res+= x+"*"
print (res)
```

Ce programme affichera:

*b*o*n*j*o*u*r*

Remarque: initialiser res à “*” permet d'avoir toutes les étoiles y compris celle de départ

remarque

quelle est la différence entre les 2 programmes:

```
ch=”bonjour”
res="*"
for x in ch:
    res+= x+"*"
print (res)
```

```
ch=”bonjour”
res="*"
for x in ch:
    res+= x+"*"
print (res)
```

Dans les deux cas à la fin, res vaut la même chose.

```
>>> res
'*b*o*n*j*o*u*r*'
```

Mais les affichages n'auront pas été les mêmes

```
>>>
*b*o*n*j*o*u*r*
```

```
>>>
*b*
*b*o*
*b*o*n*
*b*o*n*j*
*b*o*n*j*o*
*b*o*n*j*o*u*
*b*o*n*j*o*u*r*
```

Ce qui montre comment la boucle fonctionne

Boucle sur les entiers

Si on veut par exemple afficher tous les entiers entre 1 et 8

On peut faire for i in (1,2,3,4,5,6,7,8) mais on ne peut pas donner une instruction from i=1 to i=8.
On a besoin d'une **séquence** pour que for puisse la parcourir

C'est range qui va “fabriquer” cette séquence.

qui **n'est pas vraiment créée** mais on peut la parcourir.

range peut avoir 1, 2 ou 3 arguments

Range

range(n) crée la séquence des entiers de 0 à n (n exclus donc jusqu'à n-1 compris) ce qui donne n entiers

range(p,n) crée la séquence des entiers de p (inclus) à n (n exclus)

range(p,n,step) crée la séquence des entiers à partir de p (inclus) en rajoutant step à chaque fois jusqu'à atteindre ou dépasser n (n exclus)

Range et for

```
for x in range(4):
    print(x)
```

va afficher

0
1
2
3

Par défaut print va à la ligne à la fin de l'affichage

Range et for

```
for x in range(2,4):
    print(x)
```

va afficher

2
3

Range et for

```
for x in range(2,13,3):
    print(x)
```

va afficher

2
5
8
11

Remarques

```
for x in range(2,13,3):
    print(x,end='*')
```

va afficher

2*5*8*11*

Le paramètre end qui vaut par defaut '\n' (saut de ligne) permet d'écrire tous les affichages d'une boucle sur une seule ligne (avec end='')

Attention : ne pas confondre sep et end qui sont 2 paramètres possibles de print

Calculs de sommes

Exemple: on veut faire la somme des carrés des entiers de 1 à 10 inclus:

```
n=10
som=0
for p in range(1,n+1): # n+1 si on veut que n soit inclus
    som=som+p*p      # ou alors som+=p**2
print(som)
```

Calculs de sommes

Exemple: on considère la suite définie par

$u_0=1$

$u_{n+1}=3u_n+2$

Si on affecte u=1
Puis si on fait u=3*u+1 un certain nombre de fois alors u prend successivement les valeurs de u1, u2, u3 ….

D'où le programme:

```
u=1
n=10
for i in range(n):
    u=3*u+1
    print ("u = " ,u)  # on met un print à chaque étape
                        # pour “voir” fonctionner la boucle
```

Calculs de sommes

On obtient l'affichage:

```
u =  4
u =  13
u =  40
u =  121
u =  364
u =  1093
u =  3280
u =  9841
u =  29524
u =  88573    # qui correspond bien à u10
```

Calculs de sommes

Si on ne veut afficher que le résultat il suffit de modifier le programme:

```
u=1
n=10
for i in range(n):
    u=3*u+1
print ("u = " ,u)
```

Et on obtiendra simplement l'affichage:

```
u =  88573
```

En affichant p, on aura aussi l'affichage du numéro du terme.

Encore des exemples

On veut fabriquer un tuple comprenant les entiers compris entre début (inclus) et fin (exclus)

```
t=()
deb=0
fin=10
for i in range(deb,fin):
    t=t+(i,)  # ne pas oublier la virgule
print(t)
```

Remarque: additionner 2 tuples permet de rajouter l'élément i à la fin du tuple.

Il n'est pas possible de faire t+i
Attention c'est t+(i,) pas t+(i): il faut additionner 2 tuples

remarques

→ ne pas oublier la possibilité d'utiliser input pour obtenir les données à traiter

Exemple: on veut le script suivant

```
>>>
debut du tuple 1
fin du tuple 5
(1, 2, 3, 4)
```

```
t=()
deb=int(input("debut du tuple"))
fin=int(input("fin du tuple"))
for i in range(deb,fin):
    t=t+(i,)
print(t)
```

Encore des exemples

On a parcouru une chaine pour compter les espaces. On voudrait maintenant garder une trace des ***positions*** de ces espaces. On va donc mettre les indices dans un tuple pour les garder.

Comme on a besoin des indices on va parcourir la chaîne par les indices non pas par les caractères

Le programme commencera par:

```
ch="python est un langage formidable"
for i in range(len(ch)):
```

Encore des exemples

D'où le programme:

```
ch="python est un langage formidable"
t=()
for i in range(len(ch)):          # i varie entre 0 et len(ch)-1
    if ch[i]==' ':
        t=t+(i,)
print(t)
```

qui affichera

(6, 10, 13, 21)

Encore des exemples

On a un tuple de nombres dont on veut faire la somme deux par deux.

Ici le ***for x in t*** ne peut pas fonctionner car il faut additionner x et le nombre qui le suit. On va donc utiliser un parcours par indices.

```
t=(3,4,2,2,6,3,6,6,5,7)
tsom=()
for i in range(len(t)-1):  # attention
    tsom+=(t[i]+t[i+1],)
```

fin de la boucle:
t[i+1] doit exister
Donc i+1<len(t) car le dernier est indexé par len(t)-1

Encore des exemples

Si on veut faire plutôt des sommes par couples :

```
t2som=()
for i in range(0,len(t)-1,2): # on va de 2 en 2
    t2som+=(t[i]+t[i+1],)
```

```
>>> t
(3, 4, 2, 2, 6, 3, 6, 6, 5, 7)
>>> tsom
(7, 6, 4, 8, 9, 9, 12, 11, 12)
>>> t2som
(7, 4, 9, 12, 12)
```

Et avec des tuples de tuples?

→ Il n'est pas très compliqué de traiter de telles données

Exemple: on veut travailler avec des cartes style “UNO”
On aura 4 couleurs possibles (rouge, vert, jaune, bleu) et les hauteurs (1,2,3,4,5,6,7,8)

Une carte sera un tuple (hauteur, couleur) comme par exemple (2,'rouge') et un jeu sera un tuple de cartes comme par exemple jeuEx=((2,'rouge'),(4,'jaune'),(3,'rouge'))

```
>>>jeuEx[1]
(4,'jaune')
>>>jeuEx[1][0]
4
>>>jeuEx[1][1]
'jaune'
```

Et avec des tuples de tuples?

```
jeuEx=((2,'rouge'),(4,'jaune'),(3,'rouge'))
```

→ on peut alors compter les cartes rouges

```
cp=0
for carte in jeu:
    if carte[1]=='rouge':
        cp+=1
print(cp)
```

Affichera 2

Et avec des tuples de tuples?

```
jeuEx=((2,'rouge'),(4,'jaune'),(3,'rouge'))
```

→ ou les sélectionner

```
res=()
for carte in jeu:
    if carte[1]=='rouge':
        res+=(carte,)
print(res)
```

Affichera ((2,'rouge'),(3,'rouge'))

Et avec des tuples de tuples?

→ on peut alors compter les cartes rouges

```
cp=0
for carte in jeu:
    if carte[1]=='rouge':
        cp+=1
print(cp)
```

→ ou les sélectionner

```
res=()
for carte in jeu:
    if carte[1]=='rouge':
        res+=(carte,)
print(res)
```

Double boucle

Exemple : on a deux listes de prénoms et on veut écrire tous les couples possibles

```
for x in (“pierre”, “paul”, “eric”):
    for y in (“elsa”, “léa”, “anne”):
        print(“couple possible “, x, y)
```

Affichera

```
Couple possible pierre elsa
Couple possible pierre léa
Couple possible pierre anne
Couple possible paul elsa
Couple possible paul léa
Couple possible paul anne
Couple possible eric elsa
Couple possible eric léa
Couple possible eric anne
```

Un premier jeu !!!

Il s'agit de réaliser une version simple en mode “texte” d'un jeu de pierre feuille ciseau (Chi fou mi):

A chaque tour de jeu le joueur choisit pierre feuille ou ciseau ; l'ordinateur aussi. Si les deux ont choisi la même chose ils sont ex-aequo aucun point n'est marqué. Sinon le joueur marque un point s'il gagne et perd un point si c'est l'ordinateur qui gagne. On rappelle que la pierre l'emporte sur le ciseau qui l'emporte sur la feuille qui elle-même l'emporte sur la pierre.

On décide de jouer n fois de suite (en demandant n à l'utilisateur au départ) . On affichera les scores au fur et à mesure et le score final à la fin

Un peu d’aléatoire

Pour faire jouer l'ordinateur aléatoirement, on mettra

```
from random import randint
```

au début du programme

Puis

```
nb_ordi=randint(0,2)
```

pour avoir un entier aléatoire qui vaudra 0 1 ou 2
(voir cours suivant pour explications)

Exemple du déroulement d'une partie:

```
>>>
combien de tours de jeu? 4
votre choix    (tapez pierre feuille ou ciseau)pierre
l'ordinateur a choisi pierre
ex-aequo
score actuel vous avez 0 point(s)
votre choix    (tapez pierre feuille ou ciseau)feuille
l'ordinateur a choisi ciseau
le joueur perd un point
score actuel vous avez -1 point(s)
votre choix    (tapez pierre feuille ou ciseau)ciseau
l'ordinateur a choisi ciseau
ex-aequo
score actuel vous avez -1 point(s)
votre choix    (tapez pierre feuille ou ciseau)pierre
l'ordinateur a choisi ciseau
le joueur marque 1 point
score actuel vous avez 0 point(s)
score final  0
```

Comment structurer le programme?

initialisation

Choix du nombre de tours

Boucle
pour chaque tour:
  Choix par l'utilisateur
  Choix par l'ordinateur
  Calcul des points en fonction de choix et choix_ordi
  Affichage des scores

Affichage final

Ce qui va donner dans un premier temps la structure suivante:

```
pts=0   # initialisation

from random import randint   # voir cours 3

nb=int(input ("combien de tours de jeu?"))

for i in range(nb):  # boucle
    choix= ….
    choix_ordi=......
    print ("l'ordinateur a choisi "+ choix_ordi)

     Calcul des points en fonction de choix et choix_ordi

    print ("score actuel vous avez "+ str(pts)+ " point(s)")

print ("score final  " + str(pts))
```

Puis en complètant:

```
for i in range(nb):
    choix= input("votre choix    (tapez pierre feuille ou ciseau)")
    nb_ordi= randint(0,2)
    choix_ordi=("pierre", "feuille", "ciseau")[nb_ordi]
    print ("l'ordinateur a choisi "+ choix_ordi)
    if choix==choix_ordi:
        print ("ex-aequo")
    elif (choix== "pierre" and choix_ordi== "ciseau") or (choix == "ciseau"
and choix_ordi== "feuille") or (choix== "feuille" and choix_ordi == "pierre"):
        print ("le joueur marque 1 point")
        pts+=1
    else:
        print ("le joueur perd un point")
        pts-=1
    print ("score actuel vous avez "+ str(pts)+ " point(s)")
print ("score final  " + str(pts))
```

Ce programme fonctionne t-il bien?
Peut-on “planter” le programme en cours de jeu?

Les erreurs ne peuvent venir que des saisies par l'utilisateur

→ Si dans la demande du nombre de tours on ne tape pas un entier , le programme se plante (car on tente de faire int sur une chaîne qui n'est pas un entier) ; nous verrons la semaine prochaine comment faire

→ dans le cours du jeu si on tape autre chose que pierre feuille ou ciseau le programme ne se plante pas mais le joueur perd forcément le point
On pourra aussi éviter cela.

Que nous manque-t-il maintenant?

Les programmes sont tous déclenchés lors du “run” et on peut difficilement écrire des petits sous-programmes....

Ce sera très facile avec les fonctions dès la semaine prochaine.....

Remarque: grâce aux boucles on peut prévoir de faire recommencer un programme plusieurs fois de suite sans repasser par “run” .

À retenir et savoir utiliser

- → les tests, les connecteurs, booléens
- → les structures de contrôle (if, elif, else)
- → range
- → in
- → la boucle for
- → randint