Bases de Python (II)

Conditions, prédicats, graphisme

Crédits : Christian Vincent, Sorbonne Université

Sauvegardez dans un même fichier texte (dans un éditeur de texte comme Notepad++ sous Windows, Sublime Text sous Mac ou Gedit sous Linux) vos réponses à chacun des exercices. Vous m'enverrez ce fichier pour lundi 17 septembre (gael.lejeune@sorbonne-universite.fr).

Le format sera le suivant: une ligne avec "Exercice" + le numéro de l'exercice, votre code pour l'exercice puis 2 lignes vides.

Quand vous utilisez **IPython Notebook**, il est très utile d'apprendre les "raccourcis clavier" dont vous obtenez la liste par le menu "**Help** > **Keyboard shortcuts**".

Attention : quand on ouvre à nouveau un fichier sous Notebook, les cellules ne sont pas exécutées.

Rappel :

tapez sur **Ctrl + entrée** (**exécute** la cellule)

tapez sur **Alt + entrée** (**exécute** la cellule et en **crée** une nouvelle)

Sommaire du TD

1 Contrôle du flux d'instructions

1.1 Contrôle du flux avec if

1.1.1 Généralités

Sous le groupe de mots "Contrôle du flux d'instructions" se cache en fait ce qu'on appelle aussi "une structure conditionnelle". Jusqu'ici les programmes vus en TD se déroulaient de façon assez linéaire et ne prenait aucune "décision" selon un critère donné.

Le programme suivait un seul chemin sans prendre de décision seul... (ou presque)

On peut être quelquefois amené à faire un choix parmi des options proposées. Supposons que vous cherchiez à deviner un nombre proposé par un programme, nombre entre 1 et 100. Vous entrez une proposition, par exemple 65.

Le programme doit alors vous répondre "Trop grand" si le nombre proposé est plus grand que celui à deviner et "Trop petit" sinon (ou Bravo si trouvé !).

La possibilité dans un programme de faire un choix est obtenu avec l'instruction ***if***

Entrez et exécutez les lignes suivantes

```
a = 7
if a > 0:
    print(a, "est positif")
```

Remarquez que a n'est écrit à l'écran que si $a > 0$, et c'est le cas... Remplacez a par -5 et relancez le programme. Il n'affiche rien ! Normal !

1.1.2 Exercices

Exercice 1:
Écrivez un programme qui affecte aux variables a, b, c respectivement les valeurs 2, -4, 5 et qui les traite une par une de la façon suivante : il les affiche si elles sont positives sous la forme par exemple "a = 2 qui est positif", etc.

Exercice 2:
Écrivez un programme qui demande d'entrer un nombre n au clavier et qui écrit ce nombre s'il est divisible par 2 (rappelez vous de l'opérateur %). Attention, quand on entre des données avec l'instruction **input** vous avez une chaîne de caractères. Pour faire des opérations mathématiques il faut transformer ces données en **nombre**. Par exemple, pour transformer la variable a qui contient le texte "123", en le nombre 123, il faut écrire **int(a)** (rappel).

```
a = "123"     # ici a est du texte
print(type(a))
a = int(a)    # ici a est devenu un nombre
print(type(a))
```

Améliorons le programme précédemment proposé ...

Entrez les lignes suivantes et exécutez plusieurs fois ce programme en faisant varier a

```
# indique si a est positif ou non
a = input("Entre un nombre non nul :")
a = int(a)  # a devient un entier
if a > 0:
    print(a,"est positif")
if a < 0:
    print(a,"est negatif")
```

Relancez plusieurs fois le pogramme et entrez diffrents types de nombre et remarquez comment le programme prend lui-même la décision d'afficher *... est positif* ou *... est négatif.*

Exercice 3:
On peut demander au programme de traiter le cas où le nombre entré est nul en sermonnant l'utilisateur avec la phrase "Le nombre doit être non nul !!!". Ajoutez en conséquence les lignes nécessaires au programme précédent.

Autre exemple : (lancez plusieurs fois le programme)

```
# majeur ou pas ?
age = int(input("Quel est ton age ? : "))
if age >= 18:        #age suffisant
    print("Tu es majeur")
if age < 18:         #age insuffisant
    print("Tu es mineur")
```

1.1.3 Résumé

En résumé : la syntaxe de l'instruction *if* est donc :

```
if condition:        # se termine par :
    instruction1     # 1e ligne du bloc
    instruction2     # 2e ligne du bloc
    ...
    instructionX     # 3e ligne du bloc
Autre instruction (hors du bloc)
```

les instructions 1, 2 et suivantes du **bloc** ne sont exécutées que si la **condition** est réalisée. Comme pour **while**, **NE PAS OUBLIER LES 2 POINTS ":" EN FIN DE LIGNE DU IF**.

1.2 Contrôle du flux avec if et else

1.2.1 Généralités

On peut raccourcir les deux choix successifs de l'exemple précédent (avec *Majeur* ou *Mineur*) de la façon suivante, grâce aux mots clés **if** et **else**,

if = si et **else = sinon**.

La syntaxe de l'instruction **if ... else:** est la suivante :

```
if condition:
    instruction1
    instruction2
    ...
    instructionX
else:            # se termine aussi par :
    instruction1
    instruction2
    ...
    instructionY
```

Entrez et exécutez plusieurs fois les lignes suivantes

```
a = int(input("Entre un nombre :"))
if a > 0: #si a est positif
   print(a, "est strictement positif")
else:              #sinon
   print(a, "est negatif")
```

1.2.2 Exercice

Exercice 4:
Écrire un programme qui demande un nombre n et qui écrit "pair" si c'est le cas et "impair" sinon. On rappelle que a est pair se traduit par "le reste de la division entière est zéro" : $a\%2 == 0$

1.3 Contrôle du flux avec if, elif et else

1.3.1 Généralités

Il est clair que dans l'exemple précédent, on a oublié de traiter le cas où l'utilisateur entre 0. On décompose désormais le problème de la façon suivante grâce au nouveau mot clé **elif** :

si $a > 0$, alors a est strictement positif

sinon si $a < 0$, alors a est strictement négatif

sinon a est nul

Pour "traduire sinon si, on utilise **elif** (contraction de else et de if).

Entrez et exécutez les lignes suivantes (n'entrez pas les commentaires)

```
a = int(input("Entre un nombre :"))
if a > 0:           #si a est strictement positif
    print(a, "est strictement positif")
elif a < 0: #sinon si a est strictement negatif
    print(a, "est strictement negatif")
else:               #sinon a est nul
    print("Le nombre est nul")
```

Ce qui suit **else** est exécuté obligatoirement si rien de ce qui précède n'a été vérifié.

1.3.2 Exercices

Exercice 5:
Écrivez (et testez) un programme qui demande un âge. Si l'âge est inférieur à 8 ans, il écrit "Dans les petits", si l'âge est entre 8 et 16 ans (8 et 16 compris), il écrit 'Dans les moyens" et si l'âge est supérieur à 16 il écrit "Dans les grands" (astuce : commencez par les petits et les grands).

Exercice 6:
Écrivez un programme qui demande le sexe (F pour femme et H pour homme) et la situation matrimoniale (M pour marié et C pour célibataire) et qui écrit "Bonjour Monsieur ou Madame ou Mademoiselle..." selon le cas (astuce : l'homme marié ou pas est toujours monsieur, donc à traiter d'abord).

Exercice 7:
Écrivez un programme qui demande la note obtenue à un examen. Ensuite il écrit "TB" si la note est supérieure ou égale à 16, écrit "B" si la note est comprise entre 14 et inférieure à 16, "AB" si la note est comprise entre 12 et inférieure à 14, "Passable" si la note est comprise entre 10 et inférieure à 12, et "Refus" si la note est inférieure à 10.

2 Prédicats et opérateurs booléens

2.1 Notion de prédicat

2.1.1 Définition

Vous avez déjà rencontré des opérateurs de comparaison dans la fiche précédente.

Opérateurs	Significations
$<$	Strictement supérieur à
$>$	Strictement inférieur à
$<=$	Inférieur ou égal à
$>=$	Supérieur ou égal à
$==$	Égal à
$!=$	Différent de

Nous les avons utilisé lors des boucles avec **while** et lors des expressions conditionnelles avec **if** et **elif**. Par exemple, le programme suivant

```
a = −2
while a <= 6:
    print(a ** 2,end=" ")
    a = a + 1
```

s'exécute *tant que* a est plus petit ou égal à 6. Il affiche les carrés successifs de a entier, variant de -2 à 6 sur une ligne.

Autre exemple

```
a = 54
if a % 3 == 0:
    print(a, "est divisible par 3")
```

La valeur de a est affichée si a est (c'est le cas ici car $54 = 18 \times 3$) divisible par 3.

Les conditions qui se trouvent entre **while** et les deux points **:**

ou

les conditions qui se trouvent entre **if** et les deux points **:**
sont appelés des ***prédicats***.
Ces prédicats sont **vrais** ou **faux** selon le cas.

2.1.2 Vérification manuelle de prédicat

Vous pouvez tester directement ces prédicats dans une cellule.

Essayez les prédicats suivants et observez ce que *Python* renvoie.

```
5 < 13
```

```
3 * 2 == 7
```

```
"a" < "z"
```

```
"azur" > "zythum"
```

Notez la syntaxe et la casse utilisée : *True* et *False*

Essayez ensuite...

```
a = 5
print(a < 7)
print(a != 8)
print(a == 6)
```

Exécutez

```
age = 14
if age < 18:
    print("mineur")
```

Notez que cette syntaxe donne le même résultat que :

```
age = 14
if (age < 18) ==True:#les parenthèses donnent la portée du "=="
    print("mineur")
```

Mais aussi que

```
age = 14
if (age > 18) ==False:
    print("mineur")
```

Observez la différence avec :

```
age = 14
if age > 18 ==True:
    print("mineur")
```

Exécutez

```
age = 14
if age < 18 == True:    # signifie : "age < 18" est vraie
    print("mineur")
```

Ces deux programmes sont totalement équivalents (sauf en longueur...).

Une variable peut être aussi **booléenne**, dans ce cas elle ne prend que 2 valeurs : **True** ou **False**

Exécutez (sans les commentaires, ce qui est précédé de %)

```
statut = False      # statut est une variable booléenne
age = int(input("Donne ton age ? : "))
if age >= 18:
    statut = True
if statut:  # si statut est True, Majeur est écrit sinon Mineur
    print("Majeur")
else:
    print("Mineur")
```

Ici, l'écriture $if\ statut$ est un raccourci souvent utilisé
elle signifie $if\ statut == True$.

2.1.3 Utilisation en programmation

Exercice 8:
Écrivez un programme qui teste si un nombre entier entré au clavier est pair ou impair et l'écrit (ex : on entre 5 et le programme écrit "5 est impair".

Exercice 9:
Écrivez un programme qui demande un âge et qui renvoie "Précédent millénaire" si la personne est née avant 2000 et "Millénaire actuel" si la personne est née en 2000 ou après.

Exercice 10:
Écrivez un programme qui demande un âge. Si l'âge est supérieur ou égal à 18, la variable booléenne *conducteur* sera vraie sinon elle est fausse. Le programme écrit alors cette variable.

2.2 Les mots-clés and, or et not

2.2.1 Utilisation du and

Il arrive souvent que des conditions doivent tester plusieurs prédicats, par exemple quand l'on cherche à vérifier si une variable quelconque, de type entier par exemple, se trouve dans un intervalle précis (c'est-à-dire comprise entre deux nombres).

Prenez l'exemple d'un jeu de fléchettes sur une cible circulaire. Si la distance au centre est inférieure ou égale à 5 cm, on marque 30 points, si la distance au centre est comprise entre 5 cm et 10 cm (y compris), on marque 20 points, si la distance au centre est comprise entre 10 cm et 15 cm (y compris) on marque 10 points et 0 au-delà.

On peut faire le programme correspondant ainsi (Exécutez-le)

```
    # jeu de fléchettes
    d = int(input("Distance au centre ?"))
# on demande la distance de la flèchette au centre
    if d <= 5:              # la distance est infèrieure ou égale
        point = 30
    if d > 5 and d <= 10: # deux conditions
        point=20
    if d > 10 and d <= 15: # deux conditions
        point=10
    else:                   # sinon...
        point = 0
```

```
print(point)          # on affiche le nombre de point
```

Le mot-clé **and** a pour but de relier deux conditions. la ligne

$$if\ d > 5\ and\ d <= 10 :$$

signifie que **si** (d est supérieure à 5) **et** (inférieure ou égale à 10), **alors**...

2.2.2 Exercices

Exercice 11:
Écrivez un programme qui demande un nombre entier entre 0 et 100. Si le nombre est divisible par 2 et 3, il s'affiche "multiple de 6". Si le nombre est divisible par 2 et 5, il s'affiche "multiple de 10" et sinon il affiche "N'est pas multiple de 6 ou de 10".

Testez ensuite votre programme avec diverses valeurs.

Autre situation : prenez l'exemple d'un jeu de dé (1 seul dé) avec le jeu suivant où si l'on obtient 1 ou 6, on gagne 10 euros, si on obtient 2 ou 5 on gagne 5 euros et si on obtient 3 ou 4, on perd 20 euros.

Tapez et exécutez le programme correspondant à cette situation.

```
# jeu de dés
res = int(input("Resultat du jet :"))
if res == 1 or res == 6:    # on obtient 1 ou 6
    gain = 10
if res == 2 or res == 5:    # on obtient 2 ou 5
    gain = 5
if res == 3 or res == 4:    # on obtient 3 ou 4
    gain = -20
print(gain)          # on affiche le gain
```

2.2.3 Utilisation du or

Le mot-clé **or** a pour but de scinder deux conditions. la ligne

$$if\ res == 1\ or\ res == 6 :$$

signifie que **si** (res est égal à 1) **ou** (res est égal à 6), **alors**...

Le "**ou**" en programmation est un "**ou inclusif**", comme en mathématique, c'est-à-dire que les conditions qui doivent être vérifiées sont la première, la deuxième ou les deux à la fois.

Par exemple le code suivant affiche "GAGNE !" si l'entier n est divisible par 3 ou s'il est divisible par 5. "GAGNE !" sera affiché pour $n = 6$, pour $n = 10$, pour $n = 15$ mais pas pour $n = 14$.

```
if n % 3 == 0 or n % 5 == 0:
    print("GAGNE !")
```

2.2.4 Exercices

Exercice 12:
Écrivez un programme qui demande à l'utilisateur le résultat du lancer de deux pièces (avec l'instruction **input**) et qui affiche le gain obtenu en lançant deux pièces selon la règle suivante : si on obtient 2 "pile" ou 2 "face", on gagne 10 euros et si on obtient deux faces différentes, on perd 5 euros.

Le mot-clé **not** permet de prendre la négation d'une proposition (vraie ou fausse d'ailleurs). Exécutez le programme suivant

```
# not dans un programme
majeur = False
age = int(input("Entre un age :"))
if age >= 18:                    # si age>=18 majeur est vrai
    majeur = True
    print("il est majeur")
if not majeur == True:           # si majeur n'est pas vrai
    print("il est mineur")
```

On peut bien sûr, et grâce au parenthésage, combiner les *and* avec des *or* et des *not*. Attention toutefois à la lisibilité du code obtenu, il vaut mieux parfois séparer en plusieurs lignes.

Qu'affiche le programme suivant ? réfléchissez-y avant de l'essayer.

```
a = 3  # affiche-t-on 1, 2 ou 3 ?
if a > 2 and a < 3:
    print(1)
if a == 2 or (a > 1 and a < 6):
    print(2)
if not a > 5 and a < 6:
    print(3)
```

Exercice 13:
En athlétisme, les vétérans sont classés en catégories.

Vétéran 1 : de 40 à 49 ans

Vétéran 2 : de 50 à 59 ans
Vétéran 3 : à partir de 60 ans
Vétéran 4 : à partir de 70 ans
Écrivez un programme qui prend l'âge en entrée et qui renvoie la catégorie en sortie (Si l'âge est inférieur à 40, il affiche "Jeune") en n'utilisant qu'un seul *print()*.

Exercice 14:
On lance 2 dés tétraédriques (c'est-à-dire à 4 faces numérotées de 1 à 4), l'un rouge et l'autre vert. Si on obtient un double (2 fois le même numéro) on gagne 30 euros, si on obtient 4 au dé rouge et 1, 2 ou 3 au dé vert on gagne 5 euros, dans tous les autres cas on perd 2 euros. Écrivez un programme qui prend les valeurs des deux dés en entrée, et en sortie affiche le gain.
Essayez

```
a = 0
if a:
    print("0 est donc vrai ?")      # ne s'affiche pas
a = 1
if a:
    print("1 est vrai !")
```

La chaîne de caractère vide "" est considérée comme *False*, tout autre chaîne est *True*.
Essayez

```
a = "lapin"
if a:
    print("C'est un", a)
a = ""
if a:
    print("Pas de lapin !") # ne s'affiche pas
```

Si vous voulez progresser il est intéressant de bien comprendre le programme du jeu "Devine un nombre" ci-dessous.
Il combine boucles (*while*), des conditions (*if*, *else*), ... Il fait aussi appel aux résultats précédents :

```
import random # bibliotheque (library) pour le tirage au hasard
print("===== Devine un nombre =====")
rep = 1       # rep vaut d'abord 1 donc est True
while rep:    # tant que rep vaut "oui", on joue
    rep = input("Veux-tu jouer ? (oui/non)")  # demande
    if rep == "non":
        break    # on sort de la boucle while principale
    nb = random.randint(0,100)
    while rep:
        prop = int(input("Propose un nombre ?"))
        if prop == nb:
            print("Bravo !!!")
            break    # on sort du dernier while
        if prop > nb:
            print("Trop grand")
        else:
            print("Trop petit")
print("A bientôt")
```

3 Graphisme

3.1 Introduction

Pour s'amuser un peu avec les boucles, les expressions conditionnelles et le graphisme, ou tout simplement pour dessiner, une bibliothèque simpliste mais très formatrice est présente dans Python. Il suffit simplement de l'appeler pour pouvoir en disposer.

En début de programme, on place toujours la ligne suivante

from turtle import *

qui signifie "j'importe (*import*) toutes les fonctions (*) de la bibliothèque *turtle*.

Par exemple si vous ne voulez importer que la fonction *racine* de la bibliothèque *math*, vous écrivez **from math import sqrt**

```
from math import sqrt
sqrt(4)
```

Pour tout connaître sur cette bibliothèque, rendez-vous à l'adresse suivante :

https://docs.python.org/3/library/turtle.html. Pour *turtle*, il s'agit d'une "tortue" (petit triangle) qui peut se déplacer en laissant derrière elle une trace (mais elle peut faire des sauts). On peut la faire avancer, reculer, tourner, sauter à une position, réagir avec la souris ou le clavier, disparaître, apparaître, ...

La tortue peut réagir à des événements, peut aussi changer sa couleur de trace et remplir en couleur certaines zones. Avant de programmer vous-mêmes cette tortue, entrez et exécutez les lignes suivantes (exemple du site officiel) où *color*, *forward*, *left*, etc. sont des fonctions importées de *turtle*.

NB: turtle va ouvrir une nouvelle fenêtre, si elle ne s'affiche pas directement, vérifiez dans votre barre des tâches qu'elle n'est pas tout simplement "cachée".

```
from turtle import *
color('red', 'yellow')
begin_fill()
while True:
    forward(200)
    left(170)
    if abs(pos()) < 1:
        break
end_fill()
done()
```

Commandes	**Explications**	**Raccourcis**
reset()	Efface tout et replace la tortue au centre	
goto(x,y)	Place la tortue au point de coordonnées x et y	
forward(d)	Avance de d	fd(d)
backward(d)	Recule de d	bk(d)
penup()	Lève la tortue	pu() ou up()
pendown()	Repose la tortue	pd() ou down()
color(c)	Couleur du trait laissé	
left(a)	Tourne à gauche de a degrés	lt(a)
right(a)	Tourne à droite de a degrés	rt(a)
width(e)	Épaisseur e du trait	pensize()
fill(1)	Remplissage d'une suite de traits "fermés"	
write(texte)	Écrit du *texte* à l'endroit où se trouve la tortue	
speed()	Vitesse de la tortue mini=1 -¿ 1 et 0=max	

3.2 Figures élémentaires

3.2.1 Bien démarrer...

La tortue part toujours du centre de la fenêtre graphique, à moins de lui préciser les coordonnées d'un autre départ. Les coordonnées du centre de la fenêtre qui s'ouvre sont $(0;0)$. Chaque point de la fenêtre graphique à une abscisse et une ordonnée qui sont **relatives** à ce point (positive ou négative).

Entrez et exécutez les lignes suivantes qui place la tortue en $(-150,-100)$:

```
from turtle import *
goto(-150,-100)      # pour se rendre en (-150,-100)
done()
```

La dernière ligne signifie au système que l'on a fini d'écrire dans la fenêtre graphique.

3.2.2 Tracer un carré

(pensez à fermer la fenêtre précédente) Modifiez les lignes précédentes pour venir vous positionner correctement, sans laisser de "trace" (comme ci-dessous).

```
from turtle import *
up()                 # on lève la tortue
goto(-150,-100)      # pour se rendre en (-150,-100)
down()               # on repose la tortue
done()
```

Vous devez être bien positionné ! Fermez la fenêtre, reprenez les lignes précédentes que vous complétez par ce qui suit sans oublier *done()* !

Carré de côté 300

```
fd(300)              # avance=forward de 300 points
lt(90)               # à gauche=left de 90 degres
fd(300)              # avance de 300 points
lt(90)               # à gauche de 90 degres
fd(300)              # avance de 300 points
lt(90)               # à gauche de 90 degres
fd(300)              # avance de 300 points
done()
```

Remarquez que vous faites plusieurs fois la même chose (on avance et on tourne). C'est dommage, il y a les boucles pour cela. Entrez et exécutez les

lignes suivantes (ne changez que ce qui est nécessaire des lignes précédentes).

```
from turtle import *
up()
goto(-150,-100)
down()
nbcote = 1
while nbcote <= 4:  # le carré a 4 côtés
    fd(300)
    lt(90)
    nbcote = nbcote + 1
done()      # en dehors de la boucle, indentation !
```

Essayez cette fois une sorte de spirale. Vous allez reproduire plusieurs fois le "carré" en diminuant la longueur du côté à chaque "virage". Pour cela on va initialiser la variable *cote* à 300 et diminuer de 2 cette longueur à chaque passage dans la boucle que l'on va exécuter 32 fois. Exécutez et observez les lignes suivantes.

```
from turtle import *
up()
goto(-150,-100)
down()
nbcote = 1
cote = 300
while nbcote <= 32:  # pour 32 côtés
    fd(cote)
    lt(90)
    cote = cote - 2 # après chaque trait, on enlève 2
    nbcote = nbcote + 1
done()
```

Faites varier les différents paramètres afin de vérifier l'effet obtenu (augmentez le nombre de passage de la boucle). Vous pouvez accélérer ou ralentir la tortue en entrant

```
speed(0)      # vitesse maxi
#à savoir : speed(10) rapide, speed(6) moyenne, speed(1)
```

Cette instruction est à placer vers le début du programme pour que l'effet soit pris en compte dans le tracé.

3.2.3 Exercices

Exercice 15:
Reproduire le même procédé pour (départ toujours en (-150,-100))

1. d'abord faire un rectangle de longueur 400 et de largeur 300
2. construire une sorte de spirale en diminuant la longueur et la largeur de 4 à chaque boucle.

Exercice 16:
On va abandonner les angles droits (90) et introduire la variable *angle* que vous initialisez à 70 par exemple (essayez ensuite avec $angle = 80$, puis $angle = 60$). Donc la commande $lt(90)$ sera remplacée par $left(angle)$ dans le programme qui construisait une spirale à partir du carré. Exécutez et observez...

Exercice 17:
Tracez un triangle équilatéral de cote = 200 (angle = 120 pour lt(angle)). En vous inspirant de l'exercice qui vise à tracer une spirale à partir du carré, décrémentez la longueur de 4 à chaque boucle et augmentez le nombre de coté de 1 en restant inférieur ou égal à 100.

Exercice 18:
En vous aidant des bibliothèques **math** (pour la racine carré) et **turtle**, écrivez un programme qui soit capable d'afficher une petite maison comme cela

La maison est constituée d'un carré de longueur a et d'un triangle isocèle rectangle dont un coté mesure $\frac{a\sqrt{2}}{2}$, les angles à la base sont de 45 degrés. On laisse à l'utilisateur le soin de préciser la longueur du coté a et l'emplacement de la maison (par exemple les coordonnées du point inférieur gauche de la maison).

RAPPEL : envoyez vos réponses aux exercices de cette fiche pour lundi 17 septembre (gael.lejeune@sorbonne-universite.fr).