Chapitre 1

Initiation Python

Version avec preuves.

Table des matières

1 Les variables **2**
1.1 Définitions . . . 2
1.2 Types . . . 2
1.3 Opérations sur les variables . . . 3
1.4 Entrée et sortie . . . 4

2 Les conditionnelles **5**
2.1 Expression booléenne . . . 5
2.2 Conditionnelle . . . 6

3 Les boucles **6**
3.1 Liste "toute faite" . . . 6
3.2 Boucle inconditionnelle . . . 7
3.3 Boucle conditionnelle . . . 7

4 Les fonctions **8**
4.1 Return . . . 9
4.2 Variable locale . . . 9

1 Les variables

1.1 Définitions

Définition 1 :

Une variable, pour un algorithme donné, s'identifie à une case mémoire étiquetée. Elle porte donc un nom, son étiquette, et représente une valeur et elle permet de mettre en mémoire une valeur que l'on peut alors réutiliser.
En Python, la déclaration d'une variable et son initialisation (c.-à-d. la première valeur que l'on va stocker dedans) se fait en même temps.

Déclaration d'une variable en python :

```
x = 3
```

Remarque :
Il est souvent utile de mettre des noms de variables significatif.

Nom d'une variable :

```
nb_de_joueur = 3
nom_joueur_1 = "Pim"
nom_joueur_2 = "Pam"
nom_joueur_3 = "Pum"
```

1.2 Types

Définition 2 :

Le type d'une variable correspond à la nature de cette variable :

- Si cette variable est un nombre entier, son type sera `int`.
- Si cette variable est un nombre décimale, son type sera `float`.
- Si cette variable est une chaine de caractère, son type sera `str`.

Il existe beaucoup de type différents.

Exemple :

Quelques types Python :

```
>>> x = 3
>>> type(x)
<class 'int'>
>>> z = 2.7
>>> type(z)
<class 'float'>
>>> y = 2,7
>>> type(y)
<class 'tuple'>
>>> type('a')
```

```
<class 'str'>
>>> mot = "Bonjour"
>>> type(mot)
<class 'str'>
```

1.3 Opérations sur les variables

Les opérateurs sur les entiers et les flottants sont les opérateurs usuelles : + , - , * . Seule la division entière est notée : // (elle renvoie la partie entière de la division). Le reste de la division euclidienne se note % :

Opérateurs entiers et flottants :

```
>>> 5+3
8
>>> 5*7
35
>>> 3**4
81
>>> 15/2
7.5
>>> 15//2
7
>>> 15/4
3.75
>>> 15//4
3
>>> 15 % 4
3
>>> 1.1+1.2+1.3  # attention aux flottants
3.5999999999999996
```

Les chaines de caractère possèdent aussi des opérateurs :

Opérateurs chaines de caractères :

```
>>> mot_1 = "coucou"
>>> mot_2 = "salut"
>>> mot_1+mot_2  # concatenation
'coucousalut'
>>> mot_2 * 3
'salutsalutsalut'
```

Exercice 1 :

En suivant les procédures, compléter le tableau associant aux variables sa valeur :

trois étapes :

```
# etape 1
nb_1 = 8
nb_2 = -2
nb_3 = nb_1 + 2* nb_2
```

```
# etape 2
nb_1 = nb_1 + 2
nb_3 = nb_2 - nb_1
nb_2 = nb_3 // 5

# etape 3
nb_1 = (nb_1 + nb_3)//3
nb_2 = nb_2 // nb_1
nb_3 = nb_1 + nb_2 + nb_3
```

Étape	1	2	3
nb_1			
nb_2			
nb_3			

Correction

Étape	1	2	3
nb_1	8	10	-1
nb_2	-2	-3	3
nb_3	4	-12	-10

1.4 Entrée et sortie

- La fonction `input` permet de créer une interaction pour récupérer la valeur d'une variable. On peut l'argument d'une chaine de caractère, mais la fonction renvoie une chaine de caractère que l'on peut affecter à une variable :

Exemples :

```
>>> x = input("entrer un nombre : ")
entrer un nombre : 8
>>> y = input (" entrer un autre nombre ")
 entrer un autre nombre 7
>>> x+y
'87'
>>> x = int(input("entrer un nombre : "))
```

```
entrer un nombre : 8
>>> y = int(input("entrer␣un␣autre␣nombre␣:␣"))
entrer un autre nombre : 7
>>> x+y
15
```

- La fonction `print` permet l'affichage d'une chaine de caractère ou d'autres types.

Exemples :

```
>>> debut = "␣il␣faut␣"
>>> fin = "␣partaciper␣"
>>> milieu = "␣partaciper␣"
>>> fin = "␣au␣cours␣!"
>>> print( debut + milieu + fin )
 il faut  partaciper  au cours !
```

Exercice 2 :

1. Écrire un programme qui demande l'année de naissance et renvoie l'age sous la forme "Vous avez .... ans ".
2. Écrire un programme qui demande l'age et renvoie l'année de naissance.
3. Écrire un programme qui demande l'age et renvoie l'age en 2050.

2 Les conditionnelles

2.1 Expression booléenne

Définition 3 :

Une proposition est soit vraie, soit fausse.

Exemple 1 :

Si la variable A a pour valeur 12.

- P_1 : " $A \geqslant 3$ " est vraie.
- P_2 : " $A < 15$ " est vraie.
- P_3 : " $A = 5$ " est fausse.
- P_4 : " $A \leqslant 2$ " est fausse.

2.2 Conditionnelle

Définition 4 :

Une conditionnelle est une procédure qui sera effective si la proposition qui lui est associée est vraie. Il est possible de proposer une procédure alternative si la proposition est fausse.

Exemples :

```
>>> a = 8
>>> if (a>3) :
    a = 14
```

Dans l'exemple précédent, la variable `a` prend initialement la valeur 8. La proposition étant vraie, la valeur de `a` passe alors à 14.

Exemples :

```
>>> a = 8
>>> if (a<4) :
    a = 10
  else :
    a = 2
```

Dans l'exemple précédent, la variable `a` prend initialement la valeur 8. La proposition étant fausse, la valeur de `a` passe alors à 2.

Exercice 3 : Écrire une procédure qui demande à l'utilisateur d'entrer 3 entiers, et elle affiche le plus grand des 3.

Correction

```
un = int(input("Entrer le premier nb : "))
deux = int(input("Entrer le second nb : "))
trois = int(input("Entrer le troisieme nb : "))

if (un > deux) :
    if un > trois :
        print("le plus grand est : {} "+ str(un))
    else :
        print("le plus grand est : " + str(trois))
else :
    if deux > trois :
        print("le plus grand est : {} "+ str(deux))
    else :
        print("le plus grand est : " + str(trois))
```

3 Les boucles

3.1 Liste "toute faite"

Python dispose de la fonction `range` qui renvoie une liste d'entiers. Suivant les paramètres, le résultat diffère :

Exemples : Tester les différents appels :

```
>>> list(range(5))
>>> list(range(12))
```

L'appel `range(n)` renvoie la liste des entiers entre 0 et $n-1$.

```
>>> list(range(5,12))
>>> list(range(127,135))
```

L'appel `range(n,p)` renvoie la liste des entiers entre n et $p-1$.

```
>>> list(range(5,22,3))
>>> list(range(124,155,10))
```

L'appel `range(n,p,s)` renvoie la liste des entiers entre n et $p-1$ avec un pas de s.

3.2 Boucle inconditionnelle

Définition 5 :

Une boucle "Pour" (ou boucle inconditionnelle) sert à répéter une nombre de fois fini plusieurs procédures. En python, on se sert d'une liste et d'un élément qui parcours l'ensemble de la liste

Exemples :

```
>>> for i in range(3) :
  print("bonjour")

>>> for i in range(5) :
  print(i)
```

Exercice 4 : Écrire un ensemble de procédures qui demande à l'utilisateur d'entrer un entier et affiche la liste des 10 premiers multiples de cet entier.

Correction

```
"""
nb = int(input("entrer un nombre : "))


for i in range(10) :
    print(nb*i)

for i in range(0,10*nb,nb) :
    print(i)


mon_entier = 0

while ( mon_entier < 10 * nb ) :
    print(mon_entier)
    mon_entier = mon_entier + nb
"""
```

```
def ecrire_multiple (n) :
    compteur = 0
    while ( compteur <= 10) :
        print(compteur * n)
        compteur = compteur +1



def ma_fonction_2(x) :
    return ( 3*x+1)

def autre_fonction (x) :
    for i in range(25) :
        print("coucou")
        if i == x :
            return (i)
```

3.3 Boucle conditionnelle

Définition 6 :

Une boucle "Tant que" (ou boucle conditionnelle) est une boucle qui se répète tant que la condition donnée reste vrai. Si cette condition devient fausse, la boucle s'arrête.

Exemples :

```
>>> n = 5
>>> while (n < 12 ) :
  print('Bonjour')
  n = n+1
```

Exercice 5 : Écrire un ensemble de procédures qui demande à l'utilisateur d'entrer un entier et affiche la liste des 10 premiers multiples de cet entier, en utilisant une boucle conditionnelle.

Correction

```
entier = int(input("Entrer␣un␣entier␣"))

suite = 0

while suite <= 10 :
    print( suite * entier )

    suite = suite + 1
```

4 Les fonctions

Définition 7 :

Les fonctions algorithmiques permettent d'englober un ensemble de procédures et d'y inclure d'éventuels arguments. Pour créer une fonction, on utilise le mot clé `def`. Pour faire appel à la fonction, on la nomme avec les éventuelles arguments.

Exemples : Sans argument

```
def ecrire_multiple () :
  n = int(input("Entrer un nombre "))
  compteur = 0
  while (compteur <= 10) :
    print(compteur * n)
    compteur = compteur + 1
```

L'appel de la fonction se fait alors :

```
>>> ecrire_multiple ()
```

Exemples : Avec un argument

```
def ecrire_multiple (n) :
  compteur = 0
  while (compteur <= 10) :
    print(compteur * n)
    compteur = compteur + 1
```

L'appel de la fonction se fait alors :

```
>>> ecrire_multiple (5)
```

4.1 Return

Le but d'une fonction est le plus souvent de renvoyer une valeur. Cette procédure se fait avec l'appel `return`.

Exemples :

```
def ma_fonction (x) :
  return (3*x+1)
```

La fonction renvoie alors ce qui est après le `return`.

```
resultat =  ma_fonction (5)
```

Exercice 6 : Écrire la fonction `plus` qui prend en arguments un couple d'entiers et renvoie la somme des deux valeurs.

4.2 Variable locale

Définition 8 :

Une variable locale est une variable qui n'existe que dans la fonction.

Exemples :

```
s = 15
def ma_fonction ( n,p) :
  s = 5
  if n > p :
    s = 17
  return s
```

Après l'appel de la fonction, la variable global `s` reste inchangée.

Exemples : Avec gestion d'une variable globale.

```
s = 15
def ma_fonction ( n,p) :
  global s
  s = 5
  if n > p :
    s = 17
  return s
```

Après l'appel de la fonction, la variable global `s` a été changée