Révisions python - fonctions à la main

BARBIER J.M. - LGT Dumezil - TNSI

1 Dérouler un appel de fonction

1.1 Rappel

Méthode de base (**fil**, **ardoise**, **sortie**) : cf. feuille *Révisions python - à la main*.

Nouveauté : **une ardoise par appel**.

1.2 Les règles en plus

- `def` ne fait qu'une chose : créer le nom. Le corps n'est **pas** exécuté.
- chaque appel → **nouvelle ardoise**, empilée sous la précédente, titrée `nomfonction()`
- on y écrit d'abord les **paramètres**, avec le nom de la **définition** (pas celui de l'appelant)
- une ligne spéciale `retour` : vaut `None` tant qu'on n'a pas croisé `return`
- le fil saute dans le corps, puis **revient** à la ligne de l'appel
- `return` → on remplit `retour`, on **barre l'ardoise entière**, on remonte
- les variables d'une fonction n'existent nulle part ailleurs
- variable lue mais jamais affectée dans la fonction → on va la chercher dans le programme principal (en **lecture seule**)
- appels imbriqués : le **plus intérieur** d'abord

1.3 Exemple

```
1  def double(n):
2      n = n * 2
3      return n
4
5  a = 5
6  b = double(a)
7  print(a, b)
```

1.3.1 Le fil

Étape	Ligne	Ce qui se passe
1	1	`double` existe (corps non exécuté)
2	5	création de a
3	6	appel `double(5)` → nouvelle ardoise

Étape	Ligne	Ce qui se passe
4	2	n change
5	3	`return` → `retour` = 10, ardoise `double()` ☒
6	6	retour à la ligne d'appel : b = 10
7	7	affichage

1.3.2 Les ardoises

Programme principal

Nom	Valeur
`double`	*(fonction)*
a	5
b	10

double() — vivante des étapes 3 à 5, barrée ensuite

Nom	Valeur
n	~~5~~ 10
`retour`	~~None~~ 10

1.3.3 La sortie

```
5 10
```

a ne change pas : n a reçu une **copie** de la valeur.

1.4 À retenir

- un appel = une ardoise ; un `return` = ardoise barrée
- pas de `return` → `retour` vaut `None`
- `print` affiche, `return` renvoie : ce n'est **pas** la même chose
- `int`, `float`, `str`, booléens : la fonction travaille sur une copie
- listes, dictionnaires : **même objet** → les modifications se voient dehors

> Vérifier ensuite avec PyMemViz (mode *ardoise*) ou pythontutor.

1.5 Les corrections

Chaque exercice a un lien **PyMemViz** (mode *ardoise*). À ouvrir **après** avoir déroulé sur papier :

- comparer son fil, son ardoise et sa sortie avec ceux de la machine
- lancer soi-même l'exécution (bouton **Lecture auto.**), pour voir les valeurs changer et les ardoises apparaître / disparaître en direct
- bouton **Éditer** pour modifier le code et rejouer

2 Exercices : appel et retour

2.1 Exercice 1

```
def calcul(a, b=4):
    b = b - a
    return b
print(calcul(1, 3))
print(calcul(a=2))
```

Fil + ardoises + sortie. Combien d'ardoises `calcul()` créées en tout ?

Correction : dérouler dans PyMemViz

2.2 Exercice 2

```
def ajoute(x, y):
    print(x, y)
    x += y
    return x
x, y = 5, 2
print(ajoute(x, y), x, y)
```

Ardoises + sortie. Pourquoi le `x` du programme principal ne change pas ?

Correction : dérouler dans PyMemViz

2.3 Exercice 3

```
def f1(n):
    print(n * 2)
def f2(n):
    return n * 2
a = f1(3)
b = f2(3)
print(a, b)
```

Sortie ? Pourquoi a ne vaut-il pas 6 ?

Correction : dérouler dans PyMemViz

3 Exercices : arguments

3.1 Exercice 4

```
def tarif(base, remise=0, majoration=1):
    return (base - remise) * majoration
print(tarif(100))
print(tarif(100, 20))
print(tarif(100, majoration=2))
print(tarif(majoration=3, base=10))
print(tarif(100, 20, 2))
```

Sortie ? Puis : pourquoi `tarif(remise=5, 100)` est-il une erreur ?

Correction : dérouler dans PyMemViz

3.2 Exercice 5

Écrire une fonction `distance(x1, y1, x2=0, y2=0)` qui renvoie la distance entre deux points ($\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$).

Rappel : `from math import sqrt` puis `sqrt(n)`, ou `import math` puis `math.sqrt(n)`.

Donner ensuite le résultat de :

- `distance(3, 4)`
- `distance(1, 1, 4, 5)`
- `distance(y2=3, x1=0, y1=0, x2=4)`

Correction : dérouler dans PyMemViz

4 Exercices : portée des variables

4.1 Exercice 6

```
n = 1
m = 2
def essai(m, p):
    n = 10
    print(n, m, p)
print(n, m)
essai(m=7, p=8)
print(n, m)
```

Deux ardoises + sortie. Quelles variables disparaissent à la fin de `essai` ?

Correction : dérouler dans PyMemViz

4.2 Exercice 7

```
seuil = 10
def verifie(x):
    return x > seuil
print(verifie(12))
print(verifie(3))
```

Sortie ? D'où vient `seuil` dans l'ardoise de `verifie()` ?

Correction : dérouler dans PyMemViz

4.3 Exercice 8

```
compteur = 0
def incremente():
    compteur = compteur + 1
    return compteur
print(incremente())
```

Ce programme plante. Pourquoi ? Le corriger **sans** utiliser `global`.

Voir le plantage dans PyMemViz

Correction : dérouler dans PyMemViz

5 Exercices : appels imbriqués

5.1 Exercice 9

```
def g(a):
    return a + 3
def f(b):
    return b * 2
print(f(g(f(1))))
```

Ordre des appels ? Sortie ? Combien d'ardoises vivantes en même temps, au maximum ?

Correction : dérouler dans PyMemViz

5.2 Exercice 10

```
def a(n):
    print("a", n)
    return n + 1
def b(n):
    print("b", n)
    return a(n) * 2
def c(n):
    print("c", n)
    return b(n) + a(n)
print(c(1))
```

Fil complet + sortie. Combien de fois a() est-elle appelée ?

Correction : dérouler dans PyMemViz

6 Exercices : fonctions et listes

6.1 Exercice 11

```
def ajoute_zero(t):
    t.append(0)
    return t
def remplace(t):
    t = [0, 0, 0]
    return t
liste = [1, 2]
ajoute_zero(liste)
print(liste)
remplace(liste)
print(liste)
```

Sortie ? Expliquer la différence entre `t.append(0)` et `t = [0, 0, 0]`.

Correction : dérouler dans PyMemViz

6.2 Exercice 12

```
def moyenne(notes):
    total = 0
    for n in notes:
        total += n
    return total / len(notes)
mes_notes = [12, 8, 16]
print(moyenne(mes_notes))
print(moyenne([20]))
print(mes_notes)
```

Ardoise de `moyenne()` pour le premier appel (`total` change à chaque tour) + sortie. Combien d'ardoises `moyenne()` en tout ?

Correction : dérouler dans PyMemViz

7 Exercices : une fonction est un objet

7.1 Exercice 13

```
def bip():
    print("bip")
def bop():
    print("bop")
son = bip
son()
def triple(f):
    f()
    f()
    f()
triple(bop)
triple(f=son)
print(type(bip))
```

Sortie ? Que contient exactement `son` ?

Correction : dérouler dans PyMemViz

8 Exercices : à écrire

8.1 Exercice 14

Écrire `hypotenuse(a, b)` qui renvoie la longueur de l'hypoténuse d'un triangle rectangle de côtés a et b.

Rappels :

- Pythagore : $c^2 = a^2 + b^2$, donc $c = \sqrt{a^2 + b^2}$
- racine carrée : `from math import sqrt` puis `sqrt(n)`, ou `import math` puis `math.sqrt(n)`
- puissance : `a ** 2`

Vérifier à la main avec `hypotenuse(6, 8)`.

Correction : dérouler dans PyMemViz

8.2 Exercice 15

Écrire `dans_intervalle(x, mini=0, maxi=20)` qui **renvoie** `True` ou `False` selon que x est dans `[mini, maxi]`.

Donner le résultat de `dans_intervalle(25)`, `dans_intervalle(25, 0, 100)` et `dans_intervalle(5, maxi=3)`.

Correction : dérouler dans PyMemViz

8.3 Exercice 16

Écrire `maximum(t)` qui renvoie le plus grand élément de la liste `t`, sans utiliser `max`. La fonction renvoie `None` si la liste est vide.

Dérouler l'ardoise pour `maximum([3, 9, 2])`.

Correction : dérouler dans PyMemViz