Séance 1 : Variables, fonctions et conditions

L1 – Université Côte d'Azur

Exercice 1 — Variables et instructions

Exécutez manuellement le code ci-dessous et donnez à la fin de chaque ligne la valeur des variables.

```
a = 2
b = 10
c = b/a
a = 2*a
a = a+1
a == c
d = b % (a - c)
```

Exercice 2 — Fonctions simples

1. Écrivez la fonction `max(a,b)` qui renvoie le plus grand élément entre `a` et `b`. Écrivez trois tests avec `assert`
2. Écrivez la fonction `max4(a,b,c,d)` qui renvoie le plus grand élément entre les quatres arguments en utilisant la fonction `max`.
3. Même question que la précédente mais sans utiliser la fonction `max`.
4. On définit le signe d'un nombre comme étant positif si ce nombre est supérieur ou égal à zéro et négatif s'il est strictement inférieur à 0. Écrivez une fonction `même_signe(a,b)` qui renvoie un booléen indiquant si les deux nombres `a` et `b` ont le même signe.
5. Écrivez cinq tests avec `assert` pour la fonction `même_signe`

Exercice 3 — Courbes de fonctions

Tracez la courbe des fonctions ci-dessous.

```
def f(x) :
    if x <= 0 or x >= 1 :
        return 1
    else :
        return -1
```

```
def g(x) :
    if x <= 0 :
        return 1
    elif x <= 1 :
        return 0
    else :
        return 1
```

```
def h(x) :
    if x < 0 :
        return -x
    return x
```

```
def i(x) :
    return x
    if x < 0 :
        return -x
```

Exercice 4 — Le plus grand en valeur absolue

On souhaite écrire une fonction `max_abs(x,y)` qui **renvoie** le nombre le plus grand en valeur absolue. Par exemple on aura `max_abs(2,-3) == -3`. Si les deux nombres ont la même valeur absolue mais pas le même signe, la fonction renverra celui qui est positif. Par exemple, `max_abs(3,-3) == 3`.

1. Écrivez trois tests avec `assert`.

```
assert max_abs(3,-5)==-5
assert max_abs(-3,5)==5
assert max_abs(-4,4)==4
```

2. Écrivez le fonction `max_abs(x,y)`

```
def max_abs(x,y) :
    if abs(x) > abs(y) :
        return x
    elif abs(x) == abs(y) :
        if x > 0 :
            return x
        else : # abs(x)==abs(y) et x<=y
            return y
    else : # abs(x)<abs(y)
        return y
```

Autre solution :

```
def max_abs2(x,y) :
    if abs(x) > abs(y) :
        return x
    elif abs(x) == abs(y) and x > 0 :
        return x
    else :
        return y
```

3. Modifiez la fonction précédente en une fonction `afficher_max_abs(x,y,msg)` sans résultat qui **affiche** le message `msg` suivi du plus grand en valeur absolue. Par exemple, dans la console, on aura :

```
>>> afficher_max_abs(1,-3,'le plus grand en valeur absolue est')
le plus grand en valeur absolue est -3
```

```
def afficher_max_abs(x,y,msg) :
    if abs(x) > abs(y) or (abs(x) == abs(y) and x > 0) :
        print(msg , x)
    else :
        print(msg , y)
```

4. Écrivez la fonction `afficher_max_abs` en une seule ligne en appelant la fonction `max_abs`.

```
def afficher_max_abs(x,y,msg) :
    print(msg, max_abs(x,y))
```

Exercice 5 — Convertir l'heure

1. Écrivez une fonction `hms(n)` prenant un entier positif n représentant un nombre de secondes. L'effet de cette fonction sans résultat est l'affichage d'une ligne exprimant la conversion de n secondes en heures-minutes-secondes.

```
>>> hms(4567)
4567 ---> 1 heure(s) 16 minute(s) 7 seconde(s)
```

```python
def hms(n) :
    secondes = n%60
    minutes_dont_heures = n//60
    minutes = minutes_dont_heures%60
    heures  = minutes_dont_heures//60
    print(n,'--->',heures,'heure(s)', minutes,'minute(s)',secondes,'seconde(s)')
```

2. Modifier le programme pour gérer le « s » du pluriel.

```
>>> hms(4567)
4567 ---> 1 heure 16 minutes 7 secondes
>>> hms(4140)
4140 ---> 1 heure 9 minutes 0 seconde
```

```python
# L'astuce est d'ajouter une fonction auxilliaire.
def pluriel(mot,n):
    if n<=1:
        return mot
    else:
        return mot+"s"

def hms(n):
    secondes = n%60
    minutes_dont_heures = n//60
    minutes = minutes_dont_heures%60
    heures  = minutes_dont_heures//60

    h = pluriel("heure", heures)
    m = pluriel("minute", minutes)
    s = pluriel("seconde", secondes)
    print(n,'--->',heures,h , minutes,m ,secondes, s)
```

Exercice 6 — L'espion

On définit la fonction suivante :

```python
def spy() :
    print('My name is')
    # Bond, James Bond
    return 0 + 0 + 7
```

Qu'affiche la console si l'on saisit les expressions suivantes ? Expliquez.

1) `spy()`
2) `spy`
3) `spy() + spy()`
4) `max(spy() , spy())`
5) `spy() == 7 or spy() == 'My name is'`
6) `print(spy())`
7) `print(print(spy()))`

La première ligne est affichée, la seconde est retournée (c'est le résultat)

```
>>> spy()
My name is
7
```

Une fonction non évaluée a une valeur : son nom et son adresse en mémoire

```
>>> spy
<function spy at 0x7f5cf6436a20>
```

La fonction est appelée deux fois, elle affiche deux fois « My name is », puis le calcul renvoie 14.

```
>>> spy() + spy()
My name is
My name is
14
```

La fonction est appelée deux fois, elle affiche deux fois « My name is », puis le calcul renvoie 7.

```
>>> max(spy() , spy())
My name is
My name is
7
```

La fonction n'est appelée qu'une fois (évaluation paresseuse Cours 1 partie V). Le calcul n'affiche qu'une fois « My name is » puis renvoie `True`.

```
>>> spy() == 7 or spy() == 'My name is'
My name is
True
```

En apparence comme au 6.1, mais c'est le `print` qui déclenche l'affichage du 7, alors qu'à la question 1 c'était le shell qui avait affiché le résultat de l'expression. Rien n'est renvoyé (techniquement si : `None` est renvoyé, mais dans ce cas le shell *n'affiche rien).*

```
>>> print(spy())
My name is
7
```

La fonction `print` renvoie `None` ; le shell n'affiche pas le resultat, mais `print(None)` affiche `None`

```
>>> print(print(spy()))
My name is
7
None
```