# 1NSI_Fonctions_Specification_Tests

October 6, 2021

## 1 Préambule

Ce fichier est un notebook Python.

Il comporte deux types de cellules :

- les cellules d'édition dans lesquelles vous pouvez saisir du texte éventuellement enrichi de mises en formes ou de liens hypertextes avec la syntaxe du langage HTML simplifié qui s'appelle Markdown. Voir http://daringfireball.net/projects/markdown/ pour la syntaxe de Markdown.
- les cellules de code où l'on peut saisir du code Python3 puis le faire exécuter avec la combinaison de touches `CTRL + RETURN`

Une cellule peut être éditée de deux façons différentes :

- en mode *commande* lorsqu'on clique sur sa marge gauche qui est surlignée alors en bleu, on peut alors :
  - changer le type de la cellule en appuyant sur `m` pour passer en cellule Markdown ou sur `y` pour passer en cellule de code
  - insérer une cellule juste au-dessus en appuyant sur `a`
  - insérer une cellule juste en-dessous en appuyant sur `b`
  - couper la cellule en appuyant sur `x` etc …
- en mode *édition* lorsqu'on clique sur l'intérieur de la cellule.

L'aide complète sur les raccourcis claviers est accessible depuis le bouton `Help` dans la barre d'outils ci-dessus.

### 1.1 Exercice 1 Q1

L'Indice de Masse Corporelle se calcule par la formule IMC $= \frac{\text{masse}}{\text{taille}^2}$ où la masse est en kilogrammes et la taille en mètres. Un IMC est considéré comme normal s'il est compris entre 18,5 et 25. En dessous de $18,5$, la personne est en sous-poids et au-dessus de 25 elle est en sur-poids.

Écrire une fonction d'en-tête `imc(m, t)` qui renvoie la classification de l'IMC correspondant à une masse de `m` kilogrammes et une taille de `t` mètres : classe 0 pour sous-poids, 1 pour normal et 2 pour surpoids.

[ ]:
```python
def imc(m, t):
    """
    Renvoie la classification de l'IMC pour une taille t
    et une masse m:
    0 pour sous-poids
    1 pour normal
    2 pour surpoids


    Parameters:
    -----------
    m: int
    t: int

    Préconditions :
        0 <= m <= 200
        0 <= t < 2.5


    Returns:
    --------
    int
    """
    #préconditions
    assert 0 <= m <= 200 and 0 <= t < 2.5
    #à compléter


#Tests unitaires qui doivent être vérifiés par la fonction
assert imc(80, 1.5) == 2
assert imc(80, 1.8) == 1
assert imc(59, 1.8) == 0
```

Solution

Version 1 (avec structures conditionnelles imbriquées) :

```python
def imc(m, t):
    """
    Reenvoie la classification de l'IMC pour une taille t
    et une masse m:
    0 pour sous-poids
    1 pour normal
    2 pour surpoids

    Parameters:
    -----------
    m: int
```

```
    t: int

    Préconditions :
        0 <= m <= 200
        0 <= t < 2.5


    Returns:
    --------
    int
    """
    #préconditions
    assert 0 <= m <= 200 and 0 <= t < 2.5
    #à compléter
    mesure = m / t ** 2
    if mesure < 18.5:
        return 0
    else:
        if mesure <= 25:
            return 1
        else:
            return 2
```

Version 2 (avec `if ... elif ... else`) :

```
def imc(m, t):
    """
    Reenvoie la classification de l'IMC pour une taille t
    et une masse m:
    0 pour sous-poids
    1 pour normal
    2 pour surpoids

    Parameters:
    -----------
    m: int
    t: int

    Préconditions :
        0 <= m <= 200
        0 <= t < 2.5


    Returns:
    --------
    int
    """
    #préconditions
    assert 0 <= m <= 200 and 0 <= t < 2.5
```

```
    #à compléter
    mesure = m / t ** 2
    if mesure < 18.5:
        return 0
    elif mesure <= 25:
        return 1
    else:
        return 2
```

## 1.2 Exercice 1 Q2

### 1.2.1 Question 2) a)

Écrire une fonction`max2(a, b)` qui renvoie le maximum de deux nombre `a` et `b`.

[1]:
```
def max2(a, b):
    """
    Renvoie le maximum de deux entiers a et b

    Parameters:
    -----------
    a: int
    b: int

    Returns:
    --------
    int
    """
    #à compléter

#tests unitaires => à vous de les compléter en couvrant le plus de cas possible
assert max2(5, 5) == 5
assert max2(-10, 5) == 5
assert max2(-10, 0) == 0
```

Solution

Version 1 (avec structures conditionnelles imbriquées) :

```
def max2(a, b):
    """
    Renvoie le maximum de deux entiers a et b

    Parameters:
    -----------
    a: int
    b: int

    Returns:
    --------
```

```
    int
    """
    if a > b:
        return a
    else:
        return b
```

L'exécution d'une instruction `return` provoque une sortie de la fonction donc on peut omettre le `else`

```
def max2(a, b):
    """
    Renvoie le maximum de deux entiers a et b

    Parameters:
    -----------
    a: int
    b: int

    Returns:
    --------
    int
    """
    if a > b:
        return a
    return b
```

### 1.2.2 Question 2) b)

Écrire une fonction`max3(a, b)` qui renvoie le maximum de trois entiers `a`, `b` et `c`.

[3]:
```
def max3(a, b, c):
    """
    Renvoie le maximum de trois entiers a, b et c

    Parameters:
    -----------
    a: int
    b: int
    c:int

    Returns:
    --------
    int
    """
    #à compléter

#tests unitaires => à vous de les compléter en couvrant le plus de cas possible
assert max3(5, 5, 5) == 5
```

```
assert max3(5, -4, 3) == 5
assert max3(3, -4, 5) == 5
```

Solution

Version 1 (avec structures conditionnelles imbriquées) :

```
def max3(a, b, c):
    """
    Renvoie le maximum de trois entiers a, b et c

    Parameters:
    -----------
    a: int
    b: int
    c:int

    Returns:
    --------
    int
    """
    if a >= b:
        if a >= c:
            return a
        else:
            return c
    else:
        if b >= c:
            return b
        else:
            return c
```

On peut utiliser `max2(a, b)`.

```
def max3(a, b, c):
    """
    Renvoie le maximum de trois entiers a, b et c

    Parameters:
    -----------
    a: int
    b: int
    c:int

    Returns:
    --------
    int
    """
    return max2(a, max2(b, c))
```

## 1.3 Exercice 2

### 1.3.1 Question 1)

Écrire une fonction `aumoinsun(a,b,c)` qui renvoie un booléen indiquant si l'un au moins des entiers `a, b` ou `c` est positif

[7]:
```
def aumoinsun(a, b, c):
    """
    Renvoie un booléen indiquant si au moins un des trois entiers
    a,b ou c est positif ou nul

    Parameters:
    -----------
    a: int
    b: int
    c:int

    Returns:
    --------
    bool
    """
    #à compléter


#tests unitaires : à compléter pour qu'ils couvrent le plus de cas possibles
assert aumoinsun(-1, -1, 2)
assert aumoinsun(2, -1, -1)
assert not  aumoinsun(-1, -1, -1)
```

Solution

Version 1 avec code pas beau :

```
def aumoinsun(a, b, c):
    """
    Renvoie un booléen indiquant si au moins des trois entiers
    a,b ou c est positif ou nul

    Parameters:
    -----------
    a: int
    b: int
    c:int

    Returns:
    --------
    bool
    """
    if  a >= 0 or b >= 0 or c >= 0:
```

```
        return True
    else:
        return False
```

On n'écrira pas :

```
if condition:
    return True
else:
    return False
```

mais de façon plus élégante :

```
return condition
```

ce qui nous amène à :

```
def aumoinsun(a, b, c):
    """
    Renvoie un booléen indiquant si au moins des trois entiers
    a,b ou c est positif ou nul

    Parameters:
    -----------
    a: int
    b: int
    c:int

    Returns:
    --------
    bool
    """
    return  a >= 0 or b >= 0 or c >= 0
```

### 1.3.2 Question 2)

Écrire une fonction `tous(a,b,c)` qui renvoie un booléen indiquant si tous les entiers `a, b, c` sont positifs ou nuls.

[ ]:
```
def tous(a, b, c):
    """
    Renvoie un booléen indiquant les trois entiers
    a,b et c sont positifs ou nuls

    Parameters:
    -----------
    a: int
    b: int
    c:int

    Returns:
```

```
    --------
    bool
    """
    #à compléter


#tests unitaires : à compléter pour qu'ils couvrent le plus de cas possibles
assert not tous(-1, -1, 2)
assert not tous(2, -1, 3)
assert tous(2,2,2)
```

Solution

Version 1 avec code pas beau :

```
def tous(a, b, c):
    """
    Renvoie un booléen indiquant si les trois entiers
    a,b et c sont positifs ou nuls

    Parameters:
    -----------
    a: int
    b: int
    c:int

    Returns:
    --------
    bool
    """
    if   a >= 0 and b >= 0 and c >= 0:
        return True
    else:
        return False
```

On n'écrira pas :

```
if condition:
    return True
else:
    return False
```

mais de façon plus élégante :

```
return condition
```

ce qui nous amène à :

```
def tous(a, b, c):
    """
    Renvoie un booléen indiquant si les trois entiers
```

```
    a,b et c sont positifs ou nuls

    Parameters:
    -----------
    a: int
    b: int
    c:int

    Returns:
    --------
    bool
    """
    return a >= 0 and b >= 0 and c >= 0
```

### 1.3.3 Question 3)

Écrire une fonction `croissant(a,b,c)` qui renvoie un booléen indiquant si trois entiers `a, b, c` sont dans l'ordre croissant.

[ ]:
```
def croissant(a, b, c):
    """
    Renvoie un booléen indiquant si les trois entiers
    a,b et c sont dans l'ordre croissant.

    Parameters:
    -----------
    a: int
    b: int
    c:int

    Returns:
    --------
    bool
    """
    #à compléter


#tests unitaires
assert croissant(10, 10 , 10)
assert croissant(-10, -10 , -10)
assert croissant(-10, 0 , 10)
assert not croissant(10, 10, 9)
assert not croissant(10, 11, 9)
assert croissant(9, 11, 12)
assert not croissant(-9, -4, -5)
```

Solution

Version 1 avec code pas beau :

```
def croissant(a, b, c):
    """
    Renvoie un booléen indiquant les trois entiers
    a,b et c sont dans l'ordre croissant.

    Parameters:
    -----------
    a: int
    b: int
    c:int

    Returns:
    --------
    bool
    """
    if a <= b and b <= c:
        return True
    else:
        return False
```

On n'écrira pas :

```
if condition:
    return True
else:
    return False
```

mais de façon plus élégante :

```
return condition
```

ce qui nous amène à :

```
def croissant(a, b, c):
    """
    Renvoie un booléen indiquant les trois entiers
    a,b et c sont dans l'ordre croissant.

    Parameters:
    -----------
    a: int
    b: int
    c:int

    Returns:
    --------
    bool
    """
    return a <= b and b <= c
```

### 1.3.4 Question 4)

Une année est bissextile si elle est divisible par 400 ou si elle n'est pas divisible par 100 et qu'elle est divisible par 4. Écrire une fonction `bissextile(a)` qui renvoie un booléen indiquant si l'année `a` est bissextile.

[ ]:
```
def bissextile(a):
    """
    Renvoie un booléen indiquant si l'année a
    est bissextile.

    Parameters:
    -----------
    a: int

    Returns:
    --------
    bool
    """
    #à compléter

#Tests unitaires
assert bissextile(2020)
assert bissextile(2000)
assert not bissextile(1900)
assert not bissextile(2021)
```

Solution

```
def bissextile(a):
    """
    Renvoie un booléen indiquant si l'année a
    est bissextile.

    Parameters:
    -----------
    a: int

    Returns:
    --------
    bool
    """
    return a % 400 == 0 or (a % 100 != 0 and a % 4 == 0)
```

## 1.4 Entrainement 1

Écrire une fonction`mention(note)` qui prend en paramètre une note et renvoie la chaîne de caractères `'R'` si note < 10, `'A'` si 10 <= note <12, 'AB' si 12 <= note <14, 'B' si 14 <= note< 16 et 'TB' sinon.

[ ]:
```python
def mention(note):
    """
    Prend en paramètre une note et renvoie
    une chaîne de caractères :
    - 'R' si note < 10
    - 'A' si 10 <= note < 12
    - 'AB' si 12 <= note < 14
    - 'B' si 14 <= note < 16
    - 'TB' si 16 <= note

    Parameters:
    -----------
    note: int

    précondition    0 <= note <= 20

    Returns:
    --------
    str
    """
    #à compléter

#tests unitaires à compléter pour couvrir tous les cas possibles
assert mention(8)  == 'R'
assert mention(10)  == 'A'
assert mention(11)  == 'A'
```

Solution

```python
def mention(note):
    """Prend en paramètre une note et renvoie
    une chaîne de caractères :
    - 'R' si note < 10
    - 'A' si 10 <= note < 12
    - 'AB' si 12 <= note < 14
    - 'B' si 14 <= note < 16
    - 'TB' si 16 <= note

    Parameters:
    -----------
    note: int

    précondition    0 <= note <= 20

    Returns:
    --------
    str
    """
```

```
    if note < 10:
        return 'R'
    elif note < 12:
        return 'A'
    elif note < 14:
        return 'AB'
    elif note < 16:
        return 'B'
    else:
        return 'TB'
```

### 1.5 Exercice 3

[10]:
```
from random import randint

def sommeDe(n):
    """
    Renvoie la somme des résultats obtenus en lançant n
    dés à 6 faces :

    Parameters:
    -----------
    n: int

    précondition    0 <= n

    Returns:
    --------
    int
    """
    #à compléter


def urne():
    """
    Renvoie le numéro d'une boule choisie dans une urne
    contenant 5 boules de numéro 1, 3 boules de numéro 2,
    et deux boules de numéro3  :

    Parameters:
    -----------
    aucun


    Returns:
    --------
    int
```

```
    """
    #à compléter


#Pas de tests unitaires, fonctions aléatoires
```

Solution

```
def sommeDe(n):
    """
    Renvoie la somme des résultats obtenus en lançant n
    dés à 6 faces :

    Parameters:
    -----------
    n: int

    précondition    0 <= n

    Returns:
    --------
    int
    """
    #à compléter
    s = 0
    for k in range(n):
        de = randint(1, 6)
        s = s + de
    return s

def urne():
    """
    Renvoie le numéro d'une boule choisie dans une urne
    contenant 5 boules de numéro 1, 3 boules de numéro 2,
    et deux boules de numéro3  :

    Parameters:
    -----------
    aucun


    Returns:
    --------
    int
    """
    #à compléter
```

```
    choix = randint(1, 10)
    if choix <= 5:
        return 1
    elif choix <= 8:
        return 2
    else:
        return 3
```

### 1.6 Entrainement 2

- Écrire une fonction `moyenneDe(n)` qui renvoie la valeur moyenne des faces obtenues sur un échantillon de `n` lancers.
- Écrire une fonction `premier6()` qui renvoie le rang du premier 6 obtenu lorsqu'on lance successivement le dé.
- Écrire une fonction `tempsAttente(n)` qui renvoie le temps d'attente moyen du premier 6 sur un échantillon de `n` lancers.

[3]:
```
1 + 1
```

[3]: 2

Solution

```
from random import randint

def moyenneDe(n):
    """Renvoie la moyennes sur un échantillon de na lancers de dés"""
    s = 0
    for k in range(n):
        s = s + randint(1, 6)
    return s / n

def premier6():
    """Renvoie le rang du premier 6"""
    k = 1
    while randint(1, 6) != 6:
        k = k + 1
    return k

def tempsAttente(n):
    """Renvoie le temps d'attente moyen du premier 6
    sur un échantillon de n lancers"""
    t = 0
    for k in range(n):
        t = t + premier6()
    return t / n
```

## 1.7 Exercice 4

### 1.7.1 Question 1)

Écrire une fonction `spirale1(n)` qui permet de tracer une spirale constituée de `n` carrés déformés.

[ ]:
```
from turtle import *

def spirale1(n):
    """Trace une spirale constituée de n carrés déformés"""
    #à compléter


spirale1(10)
mainloop()
```

Solution

```
from turtle import *

def spirale1(n):
    """Trace une spirale constituée de n carrés déformés"""
    penup()
    goto(0,0)
    pendown()
    c = 5
    for i in range(n):
        for j in range(4):
            forward(c)
            c = c + 10
            left(90)


spirale1(10)
mainloop()
```

### 1.7.2 Question 2)

Écrire une fonction `spirale2(n, m)` qui permet de tracer une spirale constituée de `n` polygones déformés à `m` côtés.

[ ]:
```
from turtle import *

def spirale2(n, m):
    """Trace une spirale constituée de n polygones déformés
    à m cotés"""
    #à compléter


spirale2(4, 6)
```

```
mainloop()
```

Solution

```
from turtle import *

def spirale2(n, m):
    """Trace une spirale constituée de n polygones déformés
    à m cotés"""
    penup()
    goto(0,0)
    pendown()
    c = 5
    for i in range(n):
        for j in range(m):
            forward(c)
            c = c + 10
            left(360/m)



spirale2(4, 6)
mainloop()
```

### 1.8 Entrainement 3

Écrire une fonction `spirale3(n, m)` qui permet de tracer une spirale constituée de `n` polygones réguliers concentriques à `m` côtés.

[ ]:
```
from turtle import *
from math import sin, pi

def spirale3(n, m):
    shape('turtle')
    #à compléter


spirale3(4, 10)
mainloop()
```

Solution

```
from turtle import *
from math import sin, pi

def spirale3(n, m):
    shape('turtle')
    ecart = 10
    rayon = 20
```

```
    cote = 20
    for i in range(n):
        penup()
        goto(0,0)
        setheading(0)
        forward(rayon)
        pendown()
        #la difficulté est de calculer l'orientation de la tortue pour le tracé du premier cot
        setheading(90+180/m)
        cote = 2 * rayon * sin(pi/m)
        for j in range(m):
            forward(cote)
            left(360/m)
        rayon = rayon + ecart


spirale3(4, 10)
mainloop()
```

## 1.9 Exercice 5

### 1.9.1 Question 1

On décide de ranger des oeufs dans des boîtes de six. Programmer la fonction `nb_boites(n)` qui prend en argument un entier `n` correspondant à un nombre d'oeufs et renvoie le nombre de boîtes nécessaires pour ranger les oeufs.

On observera attentivement le jeu de tests et on fera quelques exemples à la main avant de commencer.

[11]:
```
def nb_boites(n):
    """
    Renvoie le nombre de boites de 6 oeufs nécessaires
    pour ranger  n oeufs

    Parameters:
    -----------
    n: int

    précondition    0 <= n

    Returns:
    --------
    int
    """
    #à compléter
```

```
#tests unitaires
assert nb_boites(8) == 2
assert nb_boites(3) == 1
assert nb_boites(6) == 1
assert nb_boites(38) == 7
assert nb_boites(600) == 100
assert nb_boites(601) == 101
assert nb_boites(0) == 0
```

Solution

```
def nb_boites(n):
    """
    Renvoie le nombre de boites de 6 oeufs nécessaires
    pour ranger  n oeufs

    Parameters:
    -----------
    n: int

    précondition    0 <= n

    Returns:
    --------
    int
    """
    if n%6 == 0:
        return n//6
    else:
        return n//6 + 1
```

### 1.9.2 Question 2

Programmer une fonction `est_pair(n:int)->bool` qui indique, en renvoyant True ou False, si un entier `n` est pair ou pas. Reformuler les tests unitaires qui sont exprimés maladroitement.

[ ]:
```
def est_pair(n):
    """
    Détermine si un entier est pair

    Parameters:
    -----------
    n: int

    Returns:
```

```
    --------
    bool
    """
    #à compléter



#Test unitaires (écriture maladroite, à reformuler)
assert est_pair(778) == True
assert est_pair(37)  == False
assert est_pair(-3)  == False
assert est_pair(0) == True
assert est_pair(-4) == True
```

Solution

```
def est_pair(n):
    """
    Détermine si un entier est pair

    Parameters:
    -----------
    n: int

    Returns:
    --------
    bool
    """
    return n%2 == 0



#Test unitaires (écriture maladroite, à reformuler)
assert est_pair(778)
assert not est_pair(37)
assert not est_pair(-3)
assert est_pair(0)
assert est_pair(-4)
```