Séquence 1 – Notions transversales de programmation

Objectifs

1. Python : Rappels - Fonctions et appels de fonction
2. Modularité - Utilisation de bibliothèques ou des API
3. Exploiter la documentation.
4. Mise au point de programmes (utiliser des jeux de tests)
5. Créer des modules simples et les documenter.

❶	Objectif : Python : Rappels - Fonctions et appels de fonction Modalités : Utilisation d'un IDE python sur Raspberry	☐	☐
❶	**1 Rappel - Deux commandes de base Python** **1.0.1 La commande print()** Cette commande permet d'afficher (imprimer) un message. Attention ! Ce message doit être mis entre guillemets : " ... "	☐	☐
❶	**A faire vous même 1.** • Ecrivez ceci : >>>print(Voici un premier message) • Que se passe-t-il ? Quel est le problème ? Comment le résoudre ? Ecrivez la bonne commande ..... • Ecrivez ceci : • >>>print("Affichage du deuxième message") • Tapez la commande pour afficher : Python c'est super ! • Essayez les touches flèches haut / flèches bas. Que se passe-t-il ?	☐	☐

1.0.2 La commande input()

Cette commande permet de demander à un « utilisateur » de saisir un texte.

A faire vous même 2.

- Ecrivez ceci :

```
>>>input("Comment t'appelle-tu ?")
```

- Que se passe-t-il ? Que faire ?

.....

- Ecrivez ceci :

```
>>>input("Quel âge as-tu ?")
```

- Tapez la commande pour afficher la question : Quelle matière préfère-tu ?
- Essayez les touches flèches haut / flèches bas. Que se passe-t-il ?

.....

2 Notion de variable en Python

2.0 Rappel - Affectation de variable

A faire vous même 3.

- Ecrivez ceci :

```
>>>toto
```

- Que se passe-t-il ?

.....

- Ecrivez ceci :

```
>>>toto=2005
>>>toto
```

- La variable toto est créé et contient la valeur 2005
- Ecrivez ceci :

```
>>>toto=1998
>>>toto
```

- La variable toto contient une autre valeur
- Créez une variable date_coupe_du_monde en lui affectant 2018
- Créez une variable poids_de_mon_chat en lui affectant 6.125
- Créez une variable nom_de_mon_chien en lui affectant Médor (Attention ! Il faut encadrer la valeur avec des guillemets !)

- Choisissez un nom de variable et affectez-lui une valeur
- Trouvez une commande Python qui affiche/imprime une variable déjà créé

2.1 Variable et commande input()

Nous pouvons combiner l'affectation de variable et la commande input.

A faire vous même 4.

- Ecrivez ceci :

```
>>>prenom=input("Quel est votre prénom")
>>>prenom
```

- Que se passe-t-il ?

.....

- Ecrivez une commande permettant de demander la couleur préférée et de la mettre dans une variable couleur
- Ecrivez une commande permettant de demander l'âge et de la mettre dans une variable age
- Testez les commandes suivantes :

```
>>>lycee="Les Cordeliers"
>>>print("Mon lycée c'est ", lycee)
```

- De la même façon, trouvez une commande qui permette d'afficher une phrase (que vous inventerez) qui intègre les 3 variables (prenom, couleur et age)

2.2 Aller plus loin

- Pas de déclaration de type (Java, C/C++, C#, ...)

```
>>>symbole = 10 * 20
>>>symbole
200
>>>type(symbole)
<class 'int'>
>>>assert type(symbole) == int # Rien ne se passe l'affirmation, l'assertion est vraie
>>>assert type(symbole)==float # Message d'erreur
...
AssertionError
```

- *original* est un symbole référençant un objet de type liste

```
>>>original = [0, 1]
```

- *alias* référence le même objet que *original*

```
>>>alias = original
```

- La preuve

```
>>>alias[0] = 10
>>>assert original == [10, 1]
>>>assert alias is original # Identité d'objet
```

- Affectations multiples

```
>>>a= b = 1 + 2
```

... est équivalent à ...

```
>>>b= 1 + 2
>>>a= b
```

- Affectations multiples bis

```
>>>a, b = 1, 2
>>>a, b = (1, 2)
```

... sont équivalents à ...

```
>>>a = 1
>>>b = 2
```

- Astuce : la substitution devaleurs

```
>>>a, b = b, a
```

- Affectation conditionnelle

```
>>>b = 1
>>>a = 5 if b > 2 else 10
>>>assert a == 10
```

❶ ## 2.3 Rappels sur les booléens ☐ ☐

```
>>>vrai, faux = True, False
>>>assert vrai or faux == True
>>>assert vrai and faux == False
>>>assert not vrai == False
>>>assert not false == True
```

❶ ## 2.4 Rappels sur les chaines de caractères ☐ ☐

```
>>>chaine = 'simple' # Avec apostrophe
>>>chaine = "double" # Avec guillemets
>>>multiligne = u"""Cette chaîne unicode
s'étend sur plusieurs
```

lignes.""" # Peut aussi être encadrée avec '''
>>>jointure_automatique = (
"ligne1"
"ligne2"
)
>>>assert jointure_automatique == "ligne1ligne2"
>>>assert type(chaine) is str # Types de base des chaines

- Caractères spéciaux avec \xxx

>>>chaine = """\
ignore""" # Le saut de ligne est ignoré
>>>assert chaine == "ignore"
>>>chaine = "\"" # Guillemets
>>>chaine = '\' # Apostrophe
>>>chaine = "\r\n\t" # Carriage return, line feed, tab
>>>chaine = "\\" # Le caractère "\" lui-même
chaine = r"chaine \ brute" # Les "\" font partie de la chaine

- Opérations sur les chaines

>>>assert "foo" + "bar" == "foobar" # Concaténation
>>>assert "foo" * 3 == "foofoofoo"
>>>assert "ab" in "zabc" # Appartenance
>>>assert "xy" not in "zabc" # Non appartenance
>>>assert "abcd"[0] == "a" # N-ième item d'une chaine, partant de 0
>>>assert "abcd"[-1] == "d" # N-ième item à droite d'une chaine, partant de -1

- Slices

>>>chaine = "abcdefgh"
>>>assert chaine[1:] == "bcdefgh" #Troncature à gauche
>>>assert chaine[:] == chaine #Ce qui permet de faire une copie
>>>assert chaine[:-2] == "abcdef" #Troncature à droite
>>>assert chaine[::2] == "aceg" #Extraction par pas de 2
>>>assert chaine [2:-2:2] == "ce" #Du 3ème à l'avant-avant dernier par pas de 2

3 Les séquences - for ... in range():

Quand on veut répéter une commande un certain nombre de fois (3 fois 7 fois, 632 fois, ...), on utilise une séquence.

A faire vous même 5.

- Ecrivez ceci sans vous trompez :

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

- Que se passe-t-il ?

.....

- Ecrivez ceci sans vous trompez :

```
>>>for i in range(5) :
... print("A la ", i)
```

- Que se passe-t-il ?

.....

4 Les scripts

La fenêtre avec les >>> s'appelle la *console*.

La console est intéressante quand on veut tester des commandes simples.

Dès qu'il faut enchaîner plusieurs commandes, il vaut mieux changer de sous-fenêtre et aller à droite pour écrire un *script*.

A faire vous même 6.

- Dans la fenêtre script écrivez une deuxième fois :

```
# -*- coding: UTF-8 -*-
print("Début du script")
for i in range(5) :
    print("A la ", i)
print("Fin du script")
```

- ATTENTION ! METTEZ 4 ESPACES AU DEBUT DE LA 3e LIGNE !
- Enregistrez ce script quelque part
- Exécutez-le
- A noter : La 1ère ligne permet simplement d'utiliser les caractères accentués (é, è, ë, ê, à, ...). Elle est à ajouter à chaque début de script.

A faire vous même 7.

- Ecrivez un script Python qui réalise les choses suivantes :
  - Afficher un message : Début de programme
  - Demander à un utilisateur : Jusqu'à combien dois-je compter ?
  - Afficher le comptage (Et de 1. Et de 2. Et de 3 ...)
  - Afficher un message : Fin de programme

5 Calculer avec Python

- Quotient seuil ou quotient entier de division euclidienne :

```
>>>27 // 7
3
```

- Modulo ou reste de division euclidienne :

```
>>>27 % 7
6
```

- Puissance :

```
>>>5 ** 3
125
```

A faire vous même 8.

- Écrivez un programme qui affiche les 20 premiers termes de la table de multiplication par 7.

A faire vous même 9.

- Écrivez un programme qui calcule le volume d'un parallélépipède rectangle dont on demande au départ la largeur, la hauteur et la profondeur.

  A noter : la fonction int() convertit une donnée saisie en entier :

```
>>>int('21')
21
```

A faire vous même 10. A faire hors classe

Écrivez un programme qui convertisse un nombre entier de secondes fourni au départ, en un nombre d'années, de mois, de jours, de minutes et de secondes. (Utilisez l'opérateur modulo : %).

6 Comparez deux valeurs

• Inférieur	• Supérieur

```
>>> a = 1
>>> b = 2
>>> a < b
True
```

- Inférieur ou égal

```
>>> a = 1
>>> b = 2
>>> 2*a <= b
True
```

- Egalité de valeur

```
>>> a = 1
>>> b = 2
>>> 2*a == b
True
```

```
>>> a = 1
>>> b = 2
>>> a > b
False
```

- Supérieur ou égal

```
>>> a = 1
>>> b = 2
>>> 2*a >= b
True
```

- Différence de valeur

```
>>> a = 1
>>> b = 2
>>> 2*a  != b
False
```

7 Instructions conditionnelles

A faire vous même 11.

- Dans la fenêtre script écrivez :

```
# -*- coding: UTF-8 -*-
nombre1 = 55
nombre2=input("Quel est votre nombre  ?")
nombre2=int(nombre2)
if nombre2 > nombre1 :
    print("Votre nombre est plus grand que le nombre du programme")
elif nombre2 < nombre1 :
    print("Votre nombre est plus petit que le nombre du programme")
else :
    print("Bravo  ! Vous avez trouvé  !")
```

- Attention ! Les lignes 5, 7 et 9 on 4 espaces en début de ligne
- Testez ce script
- Si vous avez compris, vous passez à la suite. Sinon vous appelez le professeur.

A faire vous même 12.

- L'ordi joue un dé. Le joueur joue un dé aussi. Le script dit qui est le vainqueur.
- Dans la fenêtre script écrivez ce début de script :

```
from random import randint
valeur_de1=randint(1,6)
```

- Compléter le script avec :
  - Afficher la valeur du dé de l'ordi
  - Lancer et afficher la valeur du dé du joueur
  - Comparer les 2 valeurs
  - Imprimer qui a gagné

❶

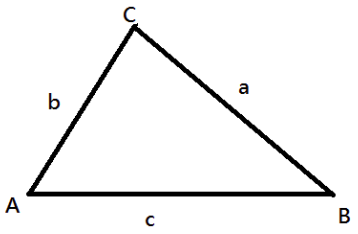
A faire vous même 13. A faire à la maison

- Écrivez un programme qui calcule les 50 premiers termes de la table de multiplication par 13, mais n' affiche que ceux qui sont des multiples de 7.

❶

A faire vous même 14. A faire à la maison

ABC est un triangle dont on connaît les longueurs en cm, des côtés.
Écrire le script python d'une fonction rectangleOuPas(a, b, c) qui donne pour résultat True (vrai) si le triangle est rectangle ou False (faux) sinon.



❶

8 Affectation augmentée

- Exemple simple :

  « a += 1 » est équivalent à « a = a + 1 »

```
>>>a = 15
>>>a += 1
>>>a
16
```

- Opérateurs compatibles : + - * /

```
>>>a = 15
>>>a *= 2
>>>a
30
```

❶

9 Boucles non bornées - while

Une boucle non bornée ne s'arrête que quand une condition est remplie.

A faire vous même 1.

- Dans la fenêtre script écrivez

```
# -*- coding: UTF-8 -*-
choix = "o"
nombre=1
while choix=="o" :
    nombre = nombre *2
    print("Puissance de 2 : ", nombre)
    choix = input("Tapez o pour continuer ")
print("AU REVOIR !")
```

- A noter : Se trouvent dans la boucle les 3 commandes qui sont décalés de 4 vers la droite

10 IMPORTANT ! L'indentation dans Python

L'indentation, c'est-à-dire le décalage vers la droite, des lignes est fondamental en Python. C'est cela qui détermine les blocs de commandes qui s'applique à tel ou tel instruction conditionnelle ou qui appartient à telle ou telle boucle.

11 Les fonctions

Il est possible de mettre des instructions que l'on utilise plusieurs fois dans une fonction. Une fonction est définie avec le mot-clé : **def**

11.0 Les fonctions mathématiques

Les fonctions python sont définies grâce au mot clé *def*.
Chaque fonction prend des paramètres (que l'on retrouve dans les parenthèses et retourne des valeur (juste après le mot-clé *return*).

A faire vous même 2.

F est la fonction définie sur $\mathbb{R}$ par :

$$\begin{cases} si\ x \leqslant 0\ alors\ f(x) = x^2 \\ si\ 0 < x \leqslant 1\ alors\ f(x) = x \\ si\ 1 < x\ alors\ f(x) = -2x + 3 \end{cases}$$

- Écrivez le programme python d'une fonction f qui donne pour résultat f(x) suivant la valeur de x.

```
premiere_fonction.py
def f(x) :
    if x<=0 :
        resultat=x**2
    elif x<=1 :
        resultat=x
    else :
        resultat=-2*x+3
    return resultat
```

- Puis testez

```
>>>from premiere_fonction import f
```

```
>>>toto=f(5)
>>>toto
-7
>>>toto=f(0.5)
>>>toto
0.5
```

❶ **A faire vous même 3.**

- Sur une droite graduée, A et B sont deux points d'abscisses a et b.

  a) Exprimez la distance AB en fonction de a et b

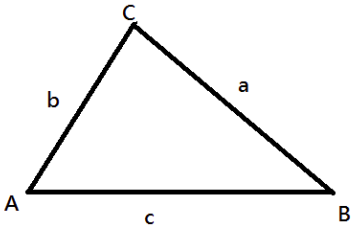
  b) Écrivez la fonction python d'une fonction dictanceDeuxPoints(a,b) qui donne pour résultat la distance AB

❶ **A faire vous même 4.**

Même chose que l'exercice précédent mais dans le plan avec A(xa, ya) et B(xb, yb)

❶ **A faire vous même 5.**

ABC est un triangle dont on connaît les longueurs en cm, des côtés.
Écrire le script python d'une fonction rectangleOuPas(a, b, c) qui donne pour résultat True (vrai) si le triangle est rectangle ou False (faux) sinon.



❶ ## 11.1 Précision sur les retours de fonctions

```
def returnValeurSimple():
    return 5
```

```
>>>resultat = returnValeurSimple()
>>>assert resultat == 5
```

```
def returnMultiple():
    return 2, 3
```

```
>>>resultat = returnMultiple()
>>>assert resultat == (2, 3) # Un tuple
>>>resultat1, resultat2 = returnMultiple()
>>>assert resultat1 == 2 # "unpack" du tuple retourné
>>>assert resultat2 == 3
```

- Les paramètres sont transmis par référence

```
def ajouteCleValeur(un_dico, cle, valeur):
```

```
    un_dico[cle] = valeur
    return
```

```
>>>dico = {}
>>>ajouteCleValeur(dico, 'un', 1)
>>>assert dico['un'] == 1
```

- Il y a un dictionnaire de toutes les variables et leurs valeurs

```
def voirLocales():
    dummy = 0
    assert 'dummy' in locals()
    print(locals())
```

```
>>>voirLocales()
```

11.2 Précisions sur les arguments de fonction

- Plusieurs arguments pour une fonction. On doit fournir tous les arguments positionnels

```
def foo(arg1, arg2):
    """Arguments positionnels"""
    return arg1 + arg2
```

```
>>>assert foo(2, 3) == 5
```

- Arguments nommés (qui ont une valeur par défaut) sont optionnels. Ils sont toujours après les positionnels

```
def foo(arg1, arg2=2, arg3=3):
    """Argument nommé **"""
    return arg1 + arg2 + arg3
```

```
>>>assert foo(2)== 7 #Tous les arguments nommés = par défaut
>>>assert foo(2,3) == 8 #2 pour le second argument
>>>assert foo(2,arg2=3) == 8 #Equivalent ci-dessus
>>>assert foo(2,arg3=4) == 8 #On utilise arg2 par défaut, on doit nommer arg3
```

11.3 Fonctions imbriquées

Il peut y avoir une fonction à l'intérieur d'une fonction.

```
def outer(param_o):
    locale_o = 0
    def inner(param_i):
        param_o = 4
        locale_o = 1
        return
    inner(param_o)
    return
```

```
>>>outer(6)
```

11.4 La fonction anonyme « lambda » (hors

programme)

- La fonction classique add2, parce qu'elle est très simple, peut se traduire par la fonction lambda add1

```
def add2(x, y):
return x + y
```

```
>>>add1 = lambda x, y: x + y # Pour les fonctions en une ligne
>>>assert add1(2, 3) == 5 # Sans surprise
```

11.5 Fonctions « builtin »

- 80 fonctions « builtin »
- Ne réinventez pas l'eau chaude
- Opérations les plus courantes sur les types standard
- len(objet) : nombre de sous-éléments d'un objet multivalué
- globals(), locals() : dictionnaires des variables globales et locales
- int(objet), float(objet), hash(objet) : conversions
- help(objet) : doc d'API de l'objet dans la console interactive
- list(objet), tuple(objet) : conversions d'objets multivalués
- etc...
- Toutes les fonctions « builtin »:

http://docs.python.org/library/functions.html

A faire vous même 6.

Programmez différentes fonctions :

- Fonction maximum(n1,n2,n3) qui renvoie le plus grand de 3 nombres n1,n2, n3
- Fonction aireDisque avec un seul argument : rayon
- Fonction volumeCylindre avec 2 arguments : rayon et hauteur
- Fonction volumeSphere
- Fonctions aireRectangle
- Fonction volumePrismeDroit que l'on puisse appelé avec 1, 2 ou 3 arguments

Objectif :

- Modularité - Utilisation de bibliothèques ou des API
- Exploiter la documentation.

Modalités : Utilisation d'un IDE python sur Raspberry

❷

12 Les bibliothèques python

12.0 Modules « builtin »

A NOTER : L'ensemble des informations officielles sur python se trouve sous www.python.org

A NOTER : Pour les allergiques à l'anglais, le site de l'association francophone python : www.afpy.org

Sans avoir à télécharger, ni installer quoi que ce soit, vous avez accès à des modules « builtin » ou « construit avec ». La liste est là : https://docs.python.org/fr/3/py-modindex.html

☐ ☐

❷

A faire vous même 1.

La fonction racine carré n'est pas une fonction « builtin ». Il faut aller la chercher dans le module math.

- Dans la console python testez ces lignes :

```
>>> from math import sqrt
>>> sqrt(9)
3
```

- Une autre façon de faire :

```
>>> import math
>>> math.sqrt(9)
3
```

- Ou en utilisant un alias :

```
>>> import math as m
>>> m.sqrt(9)
3
```

☐ ☐

❷

A faire vous même 2.

S'il est bien fait, un module python intègre sa propre documentation.

- Dans la console python testez ces lignes :

```
>>> import math
>>> help(math)
```

- Vous avez la liste de toutes les fonctions disponibles dans ce module avec une courte explication
- Cherchez comment récupérer une valeur approchée du nombre réel pi
- Testez la fonction sinus
- Testez une autre fonction de votre choix

☐ ☐

❷

A faire vous même 3.

L'aide peut être récupérée et lue via des pages web. Dans un shell linux tapez :

```
$ python -m pydoc -p 1234
```

Vous venez de lancer un mini-serveur web

- Consultez la documentation et testez le module random
- Ecrivez un petit script qui simule le tirage du loto

☐ ☐

❷

- Pour le plus rapides et/ou les plus motivés, voici une liste de modules intéressants :
  - time, datetime, calendar : gestion de l'heure et de la date
  - sys : adaptation de Python au système hôte. Un peu « fourre tout »
  - os : se substitue à certaines commandes linux
  - os.path : manipulation de chemin de fichiers
  - stat : informations sur les fichiers (sécurité, dates, UID, ...)
  - tempfile : fichiers et répertoires temporaires
  - glob, fnmatch : recherche et correspondance des fichiers avec patterns Unix (« *.py? », ...)
  - shutil, filecmp : déplacement, copie, comparaison de fichiers et répertoires

☐ ☐

❷

12.1 Modules pouvant être installés avec pip

La communauté des développeurs de python se sont organisés pour mettre en ligne leurs modules. Tout est sur le site de PyPI (Python Package Index) : https://pypi.org/

Comme l'installation de paquets linux, il existe un outil pour télécharger et installer automatiquement un de ces modules : pip

Vous pouvez le faire dans un shell linux :

```
$ pip install pillow #installe le module pil pour
manipulation d'image
```

Ou alors directement dans la console python :

```
>>>!pip install pillow
```

☐ ☐

❷

A faire vous même 4.

- Installez le module pillow (ou pil)
- Modifiez une image de votre choix (faites une rotation, une symétrie, ...)

☐ ☐

❷ **A faire vous même 5. (pour les rapides)**

- Installez le module numpy
- Testez ses possibilités

☐ ☐

❷ Un article sur les modules python qui peuvent être utiliser en sciences :
https://linuxfr.org/news/python-pour-les-sciences-une-presentation

☐ ☐

❷

12.2 Construire soi-même ses propres modules

12.2.1 Les Modules

Tous les fonctions, constantes et objets peuvent être enregistrés dans un fichier python (.py). Si ce fichier se trouve dans un répertoire déclaré dans le PYTHONPATH alors il est possible de faire un import.

A NOTER :

Quand un module est exécuté ou importé, son corps de programme est automatiquement exécuté. Pour éviter que ce corps de programme soit exécuté à l' import, il faut commencer par écrire :

```
if __name__ == '__main__':
    #Ici le corps du programme
```

☐ ☐

❷ **A faire vous même 6.**

- Construisez un module (nommée aires_et_volumes) de fonctions calculant les aires (carré, rectangle, disque, triangle, trapèze, ...)
- Ajoutez-y les calculs de volumes si possible s'appuyant sur un calcul d'aire (prisme droit, cylindre, sphère, cône, pyramide, ...)
- Ecrivez un corps de programme proposant un menu permettant d'exécuter les fonctions :

  Que voulez vous faire ?

  0- Pour quitter

  1- Pour ...

  2- Pour ...

☐ ☐

❷

12.2.2 La documentation

Chaque fonction peut être accompagné d'une docstring (""" ... """). Cette docstring fournit une documentation ou aide en ligne qu'il est possible d'afficher avec la commande help().

Pour les fonctions, souvent la docstring présente les paramètres d'entrée nécessaires et les valeurs retournées par la fonction.

Il est même possible de rédiger une docstring de présentation

☐ ☐

globale du module.
A NOTER : Le module sphinx permet d'obtenir une documentation style web détaillé.

❷ **A faire vous même 7.**

- Pour votre module aires_et_volumes, rédigez votre documentation (module et chaque fonction individuellement).
- Installez le module sphinx
- Renseignez-vous sur la manière d'écrire la documentation
- Rédigez votre documentation
- Testez sphinx et visualisez votre documentation

❸ Objectif :

- Mise au point de programmes (utiliser des jeux de tests)

Modalités : Utilisation d'un IDE python sur Raspberry

❸

13 Jeux de tests d'une fonction

Il est possible de créer des tests (avec des cas plus ou moins piègeux) afin d'être certain que la fonction se comporte de manière attendue.
Dans un deuxième temps, quand il y a des modifications dans le code, de vérifier qu'il n'y pas de régression.
Plus radical, certains préfèrent écrire d'abord les jeux de tests et ensuite le module est développé. On parle de "développement dirigé par les tests" ou "Tests Driven Development" (TTD).
Très souvent, ces tests sont faits avec des affirmations/assertions et la commande *assert*.

13.0 Jeux de test dans le corps de programme

Exemple :

```
def add2(x, y):
    return x + y

if __name__ == '__main__':
    assert add2(5,7)==12
    assert add2(0,1)==1
    assert add2(4.5, 5.6)==10.1
```

Dans ce cas, le jeux de tests est déroulé à l'éxécution du programme mais n'est pas lancé à l'import de la fonction.

	A NOTER : Il existe un module python pour faciliter la rédaction de jeux de tests : unittest - https://docs.python.org/2/library/unittest.html Mais pour pouvoir l'utiliser nous avons besoin de connaitre les principes de la Programmation Orientée Objet (POO) que nous verrons très bientôt.		

❸	**13.1 Jeux de test dans la documentation**
	Une documentation donne souvent des exemples de fonctionnement de telle ou telle fonction.
	Le module doctest - https://docs.python.org/2/library/doctest.html permet de s'appuyer sur ces exemples pour créer automatiquement un jeu de test.
	A NOTER : L'utilisation de l'invit de commande python >>> comme si la console python était utilisée.
	Exemple :
	`def add2(x, y):`
	`  """Fonction addition de deux nombre.`
	`  Prend en paramètres : Deux nombres`
	`  Retourne un nombre`
	`  >>> add2(5,7)`
	`  12`
	`  >>> add2(4.5,5.6)`
	`  10.1"""`
	`  return x + y`
	
	`if __name__ == '__main__':`
	`  import doctest`
	`  doctest.testmethod()`
	Dans ce cas, le jeux de tests est déroulé à l'éxécution du programme mais n'est pas lancé à l'import de la fonction.
		☐	☐

❸	**A faire vous même 1.** • Reprenez le fichier premiere_fonction.py et ajoutez un jeu de tests dans la documentation de la fonction f. • Modifiez le corps de programme pour dérouler ce jeux de tests.	☐	☐
❸	**A faire vous même 2.** • Pour votre module aires_et_volumes, modifiez la documentation pour y inclure des jeux de test à chaque fonction.	☐	☐

❶❷❸❹❺❻❼❽❾