

L1-MPCE2I L1-BBTE L1-ECO-MATHS	Informatique - Travaux Pratiques	TP2
--------------------------------------	----------------------------------	-----

### Objectifs du TP

- Les variables, les expressions, les entrées-sorties et les conditions (oui encore !)
- Les exceptions
- Les fonctions

### Exercice 1 : définir et exécuter une fonction

Lancez le logiciel IDLE puis, dans une fenêtre d'édition, saisissez le programme suivant :

```
# -*- coding: utf-8 -*-

def hello_world(name):
    print('My name is', name)
```

Chargez le fichier dans l'interpréteur avec le raccourci « F5 » ou via le menu « Run / Run module » de la fenêtre d'édition, puis testez la fonction dans l'interpréteur comme illustré dans l'image suivante :

```
Python 3.4.3 Shell
>>> ===== RESTART =====
>>>
>>> hello_world('Bond')
My name is Bond
>>> hello_world('Ulysse')
My name is Ulysse
>>> hello_world('toto')
My name is toto
>>>
```

Modifiez la fonction pour qu'elle corresponde au fonctionnement attendu et illustré dans l'image suivante :

```
Python 3.4.3 Shell
>>> ===== RESTART =====
>>>
>>> hello_world('Bond', 'James')
My name is Bond
James Bond
'Agent 007'
>>> agent = hello_world('Bond', 'James')
My name is Bond
James Bond
>>> print(agent)
Agent 007
>>>
```

### Exercice 2 : ma première fonction

Dans une fenêtre d'édition, écrivez une fonction nommée *moyenne(...)* qui calcul la moyenne de trois valeurs entières. La fonction devra gérer le cas où un (ou plusieurs) des arguments n'est pas convertible en entier comme illustré par l'image suivante :

```

Python 3.4.3 Shell
>>> ===== RESTART =====
>>>
>>> moyenne(10,12,13.5)
11.666666666666667
>>> moyenne(5,8,6)
6.333333333333333
>>> moyenne(4,3,'toto')
Erreur : vous n'avez pas saisi un entier
>>> moy = moyenne(90,85,55)
>>> print(moy)
76.66666666666667
>>> a = 63
>>> b = 56
>>> c = 23.56
>>> d = moyenne(a, b, c)
>>> print(d)
47.333333333333336
>>>
Ln: 38 Col: 4

```

### Exercice 3 : joue avec l'ordinateur

Lancez le logiciel IDLE puis, dans une fenêtre d'édition, écrivez une version par ordinateur du fameux jeu « Pile ou face ? » dans lequel l'ordinateur tire au hasard et l'utilisateur choisit un côté de la pièce « virtuelle ». La fonction appelée *jpof()* devra fonctionner comme dans l'exemple de l'image suivante :

```

Python 3.4.3 Shell
>>>
>>> jpof()
Pile (p) ou face (f) ? p
Gagné, c'est pile !
>>> jpof()
Pile (p) ou face (f) ? f
Perdu, c'est pile !
>>> jpof()
Pile (p) ou face (f) ? g
C'est pile : p ou face : f !
>>>
Ln: 40 Col: 4

```

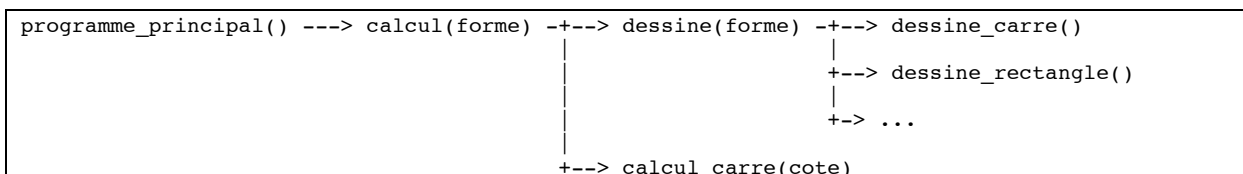
La fonction devra contrôler la saisie en vérifiant la validité ('p' ou 'f' uniquement).

Remarque : vous utiliserez la fonction *randint(a, b)* de la librairie *random* qui retourne une valeur aléatoire comprise dans l'intervalle  $[a, b]$  (*a* et *b* étant des entiers).

### Exercice 4 : une fonction qui appelle une fonction qui appelle des fonctions qui...

Le but de l'exercice est d'écrire un ensemble de fonctions dans le but d'établir un programme de calcul de surface géométrique plane. Une fonction principale fera appel à une autre fonction qui elle-même fera appel à d'autres fonctions et ainsi de suite.

La structure d'appel est la suivante :



```

|
+--> calcul_rectangle(cote1, cote2)
|
+--> calcul_parallelogramme(base, hauteur)
|
+--> ...

```

Les fonctions sont définies de la manière suivante :

```

programme_principal() : affiche un menu, vérifie la saisie puis lance le calcul

calcul(forme) : affiche la forme puis délègue le calcul de la surface à une autre fonction
après saisie des informations nécessaires pour ce calcul

dessine(forme) : affiche la forme donnée en argument

calcul_carre(cote) : calcul la surface d'un carré et retourne la valeur

dessine_carre() : affiche un carré

calcul_rectangle(cote1, cote2) : calcul la surface d'un rectangle et retourne la valeur

dessine_rectangle() : affiche un rectangle

etc.

```

**On se limitera dans un premier temps aux trois surfaces suivantes : carré, rectangle et trapèze.** Si vous avez du temps en fin de TP, vous pourrez ajouter d'autres surfaces : parallélogramme, losange, triangle, cercle et ellipse.

Le programme devra fonctionner comme dans les exemples des images suivantes :

```

Python 3.4.3 Shell
>>> programme_principal()
Choisissez un calcul de surface :
- carré.....c
- rectangle.....r
- parallélogramme.....p
- losange.....l
- trapèze.....tra
- triangle.....tri
> tre
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    programme_principal()
  File "/Users/lemeunie/Desktop/exercice_2_py3.py", line 188, in programme_principal
    raise Exception('Erreur : surface inconnue')
Exception: Erreur : surface inconnue
>>> programme_principal()
Choisissez un calcul de surface :
- carré.....c
- rectangle.....r
- parallélogramme.....p
- losange.....l
- trapèze.....tra
- triangle.....tri
> c
côté
côté > t
Erreur : seuls les entiers ou les réels sont acceptés
>>>
Ln: 82 Col: 4

```

Dans l'image précédente on voit que l'utilisateur doit saisir une surface connue sinon une exception est levée et le programme s'arrête. Il faut aussi contrôler la saisie des dimensions qui doivent être des entiers ou des réels.



```

Python 3.4.3 Shell
>>> ===== RESTART =====
>>>
>>> pgcd(10,2)
2
>>> pgcd(2,10)
2
>>> pgcd(1426,48)
2
>>> pgcd(123567, 123)
3
>>> pgcd(123,123567)
3
>>> pgcd(45986236598, 4569)
1
>>>
Ln: 34 Col: 4

```

Rappel : le plus grand diviseur commun de deux entiers naturels  $a$  et  $b$  est noté  $pgcd(a, b)$ . Le  $pgcd(a,b)$  divise  $a$  et divise  $b$ . Le  $pgcd$  est défini mathématiquement ainsi :

$$\text{Soit } a = bq + r \text{ (} r \text{ est le reste de la division euclidienne tel que } r = a \bmod(b))$$

$$pgcd(a, 0) = a \quad \text{et} \quad pgcd(a, b) = pgcd(b, r)$$

### Exercice 6 : temps d'exécution et fonction récursive de la suite de Fibonacci

- a) Mesurer le temps d'exécution d'une fonction ou d'un programme est quelque fois important car il permet de comparer différentes versions de codage d'un même algorithme ou simplement de connaître le temps d'attente possible lors de l'exécution d'une fonction.

Dans une fenêtre d'édition, saisissez le code suivant :

```

# coding: utf-8

from time import *

def duree_print(chaine):
    debut = time()
    print(chaine)
    fin = time()
    print('print avec "', chaine, '" en', fin - debut, 'secondes')

```

Testez la fonction `duree_print(...)` avec différents arguments :

```

Python 3.4.3 Shell
>>> ===== RESTART =====
>>>
>>> duree_print('toto')
toto
print avec " toto " en 0.11032390594482422 secondes
>>> duree_print('toto et titi et tata')
toto et titi et tata
print avec " toto et titi et tata " en 0.10218191146850586 secondes
>>> duree_print('toto et titi et tata et tutu et tete et lulu et lolo et rara et roro')
toto et titi et tata et tutu et tete et lulu et lolo et rara et roro
print avec " toto et titi et tata et tutu et tete et lulu et lolo et rara et roro " en 0.1101830005645752 secondes
>>>
Ln: 1727 Col: 4

```

- b) Ecrivez une fonction récursive qui calcul le  $n$ ème terme de la suite de Fibonacci et étudiez son temps d'exécution. La fonction appelée `fibonacci(...)` et la fonction `duree_fibonacci(...)` devront fonctionner comme dans l'exemple de l'image suivante :

```
Python 3.4.3 Shell
>>> ===== RESTART =====
>>>
>>> fibonacci(6)
8
>>> duree_fibonacci(10)
fibonacci avec " 10 " vaut 55 en 5.1975250244140625e-05 secondes
>>> duree_fibonacci(20)
fibonacci avec " 20 " vaut 6765 en 0.004933834075927734 secondes
>>> duree_fibonacci(30)
fibonacci avec " 30 " vaut 832040 en 0.5658049583435059 secondes
>>> duree_fibonacci(40)
fibonacci avec " 40 " vaut 102334155 en 73.86328887939453 secondes
>>>
```

Rappel : la suite de Fibonacci est la suite d'entier définie de la manière suivante

$$F(0) = 0, F(1) = 1 \text{ et } F(n) = F(n-1) + F(n-2)$$

Attention : ce n'est pas la même fonction qu'en TD !