LES FONCTIONS (et gestion des erreurs) NSI 04

1. LA GESTION DES ERREURS:

✔ **ACTIVITE:** Créez le fichier *erreurs.py* puis enregistrez-le dans le répertoire THONNY. Commençons par inviter l´utilisateur à entrer un entier...

```
x=int(input("Entier: "))
```

A l´exécution du programme, entrons autre chose qu´un entier…

```
    x=int(input("Entier: "))
ValueError: invalid literal for int() with base 10: ''
```

Extrait de la documentation officielle de PYTHON:

> A **TypeError** occurs when an operation or function is applied to an object of inappropriate type (exemple déjà rencontré: "3"+3).
> **ValueError** is thrown when a function's argument is of an inappropriate type (c´est le cas ici).

Impossible de concevoir un programme qui n´anticipe pas les erreurs de saisie (**ValueError**). PYTHON, comme les autres langages permet de gérer les erreurs avec les blocs **try** et **except** (**try** et **catch** dans les autres langages):

```
# syntaxe:
try:
  instructions
except ValueError:
  instructions en cas
  d´erreur sur le type de
  variable dans le bloc try
```

```
# tapez ce code dans erreurs.py
try:
  x=int(input("Entier: "))
except ValueError:
  print("Pas un entier...")

L´erreur est maintenant prise en
compte par le programme
```

C´est mieux, le programme est quitté sans erreur mais quitté quand même! Comment forcer l´utilisateur à saisir une entrée du type attendu? Une solution s´impose: la boucle **while**.

```
# tapez et testez ce code dans erreurs.py
while True:   # cette condition est toujours réalisée!
    try:
        x=int(input("Entier: "))
        break # tout va bien, on peut sortir de la boucle
    except ValueError:
        print("Ce n´est pas un entier, recommencez...")
print("Sortie de boucle, l´entier saisi vaut ",x)
```

✔ AIDE MEMOIRE, GESTION DES ERREURS:

QUELQUES TYPES D´ERREURS	
Type d´erreur :	exemples:

ValueError	>>>
	
	>>> from math **import** *
	>>> .....

ZeroDivisionError	>>>
TypeError	>>> .....

SYNTAXE DES BLOCS TRY ET EXCEPT	
Pour:	on tape en Python:
exécuter des instructions en cas d´erreur	
forcer le programme à recommencer **tant que** l´erreur n´est pas levée (raised)	

2. LES FONCTIONS:

Pour arrondir le nombre π on peut procéder ainsi:

- Multiplier π par 10 à la puissance du nombre de décimales désiré puis ajouter 0,5
- Ne garder que la partie entière de ce nombre
- Diviser cette partie entière par 10 à la puissance du nombre de décimales désiré

Pour arrondir le nombre π deux chiffres après la virgule on peut donc taper le code suivant (après importation de pi du module math): **`int(pi*10**2+0.5)/(10**2)`**

Téléchargez le fichier *arrondis.py* sur le site, placez-le dans le dossier où vous stockez tout votre matériel de cours puis ouvrez-le avec anaconda-spyder ou un éditeur de texte avant de copier le programme dans repl.it.

✔ **ACTIVITE:** Observez attentivement la syntaxe de la déclaration des trois premières fonctions. Dans la zone consacrée au corps du programme, testons successivement ces différents appels de fonctions (**`\u03c0`** est le caractère π en Unicode UTF-8).

```
arrondir_pi()
```

```
a=arrondir_pi_2(5)
print("\u03c0=",a)
```

```
a=arrondir_pi_3(5)
print("\u03c0=",a)
```

La dernière fonction doit nous fournir l´arrondi **de n´importe quel nombre** (*float*) avec **le nombre de décimales** voulu (*int*) passés tous deux en paramètres. Par exemple, l´appel/affectation de la fonction dans le corps du programme: a=**arrondir_nombre**(5.765832,3) doit afficher après **print**("arrondi:",a): **`arrondi:`** `5.766` dans la console.

a) Terminez dans le fichier *arrondis.py* l´écriture de la fonction **`arrondir_nombre`** puis testez si son appel/ affectation puis son affichage fonctionnent.
b) Modifiez le corps du programme de manière à ce que ce soit l´utilisateur qui saisisse lui même le nombre à arrondir et le nombre de décimales désiré. Testez...
c) Coupez cette partie du code (le corps du programme) du fichier *arrondis.py* puis enregistrez. Créez un nouveau fichier *principal.py* dans le même répertoire puis tapez:

```
from arrondis import arrondir_nombre

# Collez ici le code du b)
```

Si tout fonctionne, vous venez de créer votre premier module!

✔ AIDE MEMOIRE, FONCTIONS:

SYNTAXE DES DECLARATIONS ET APPELS DE FONCTIONS	
Déclaration et appel d´une fonction sans argument en entrée ni valeur retournée en sortie (on parle de procédure).	
Déclaration et appel d´une fonction sans argument en entrée avec valeur retournée en sortie. *ma_fonction() est maintenant une variable de type int, float, str, bool...*	
Déclaration d´une autre fonction avec arguments (arg) en entrée puis appel avec paramètres (param) correspondants. *Les paramètres et arguments peuvent être de type int, float, str, bool..*	
Appeler une fonction définie dans un autre programme (module): ***fichier.py***	

QUELQUES REGLES A CONNAITRE

+ Le programme ignore la déclaration d´une fonction (définie avec le mot clé **def**) tant qu´elle n´est pas appelée.
+ En PYTHON, les fonctions se déclarent avant leur appel.
+ L´instruction **return** permet à la fonction d´être utilisée comme variable de type, *int, float, str, bool...*
+ Dans le corps d´une fonction, on peut appeler une autre fonction.
+ Une fonction peut être passée en paramètre d´une autre fonction.
+ Une variable définie à l´intérieur du corps d´une fonction est **locale**, elle est invisible du corps du programme.