THÈME 6:
FONCTIONS (AVANCÉ)

Notions du thème :

- Fonctions:
  - Plusieurs valeurs de retour
  - Docstring
  - Arguments optionnels
- Le mot-clé **`None`**

Plusieurs valeurs de retour: Un exemple

```
def division(a,b) :
    # renvoie le quotient et le reste
    # de la division de a par b
    quotient=a//b
    reste= a%b
    return quotient, reste

# programme principal
q,r = division(22,5)
print("q=", q, "et r=", r)
```

Plusieurs valeurs de retour

Syntaxe (dans le corps de la fonction):

`return` valeur1, valeur2, … , valeurN

Pour récupérer toutes les valeurs de retour lors d'un appel:

```
var1, var2,…, varN = nom_fonction(arguments)
```

Docstring: Un exemple

```
def division(a,b) :
      """ Renvoie le quotient et le reste
       de la division de a par b """
      quotient=a//b
      reste= a%b
      return quotient, reste
```

Dans l'interpréteur (ou dans un programme):

```
>>> help(division)
Help on function division in module __main__:

division(a, b)
    Renvoie le quotient et le reste
    de la division de a par b
```

Docstring

Une **docstring** est une **chaîne de caractères** (encadrée par des triple guillemets) placée au tout **début d'une fonction**, qui permet de **décrire** la fonction.

```
def nom_fonction(argument1, argument2, …):
    """ docstring
    """
    instructions de la fonction
```

On peut l'afficher grâce à **`help(`**`nom_fonction`**`)`**:

```
Help on function nom_fonction in module:
nom_fonction(argument1, argument2, …)
    docstring de la fonction affichée ici
```

/!\ Pas de parenthèses après `nom_fonction` dans `help(…)`

Docstring sur une fonction existante

```
>>> import random
>>> help(random.randint)
Help on method randint in module random:

randint(a, b) method of random.Random instance
    Return random integer in range [a, b],
including both end points.
```

Définir une fonction avec des arguments optionnels: exemple

La carte de MisterPizza comporte de multiples saveurs de pizzas, chacune pouvant être commandée en taille normale au prix de 9€, ou en taille maxi au prix de 12€. La très grande majorité des clients choisit des pizzas de taille normale.

```
def affiche_pizza(saveur, taille="normale"):
    """ Affiche saveur, taille et prix de la pizza
    """
    print("Pizza", saveur, "taille:", taille)
    if taille=="normale":
      prix=9
    elif taille=="maxi":
      prix=12
    print("Prix", prix, "euros.")
```

➔`taille` est un argument optionnel ayant comme **valeur par défaut** `"normale"`.

Définir une fonction avec des arguments optionnels

Pour rendre un argument **optionnel** lors de la définition d'une fonction, il faut ajouter après le nom de l'argument le signe = suivi de la valeur par défaut.

```
def nom_fonc(arg1, arg_opt=valeur_par_defaut):
    instructions
```

➔ `arg_opt` ***est*** un argument optionnel de la fonction

Mais `arg1` ***n'est pas*** un argument optionnel de la fonction

Appeler une fonction avec des arguments optionnels

```
>>> affiche_pizza("4 fromages")
Pizza 4 fromages taille: normale
Prix 9 euros.
>>> affiche_pizza("4 fromages", "maxi")
Pizza 4 fromages taille: maxi
Prix 12 euros.
>>> affiche_pizza("Reine", "normale")
Pizza Reine taille: normale
Prix 9 euros.
```

Help et Docstring sur une fonction avec arguments optionnels

```
>>> help(affiche_pizza)
Help on function affiche_pizza in module __main__:

affiche_pizza(saveur, taille='normale')
    Affiche saveur, taille et prix de la pizza
```

Plusieurs arguments optionnels

MisterPizza souhaite parfois afficher le prix en Francs (lorsque le client est âgé, mais il s'agit d'une situation peu fréquente).

```
def affiche_pizza(saveur, taille="normale", afficheF=False):
    """ Affiche saveur, taille et prix de la pizza
    """
    print("Pizza", saveur, ", taille: ", taille)
    if taille=="normale":
        prix=9
    elif taille=="maxi":
        prix=12
    if afficheF:
        prixFrancs=round(prix*6.55957, 2)
        print("Prix", prix, "euros (", prixFrancs, " F).")
    else:
        print("Prix", prix, "euros.")
```

Appel avec plusieurs arguments optionnels

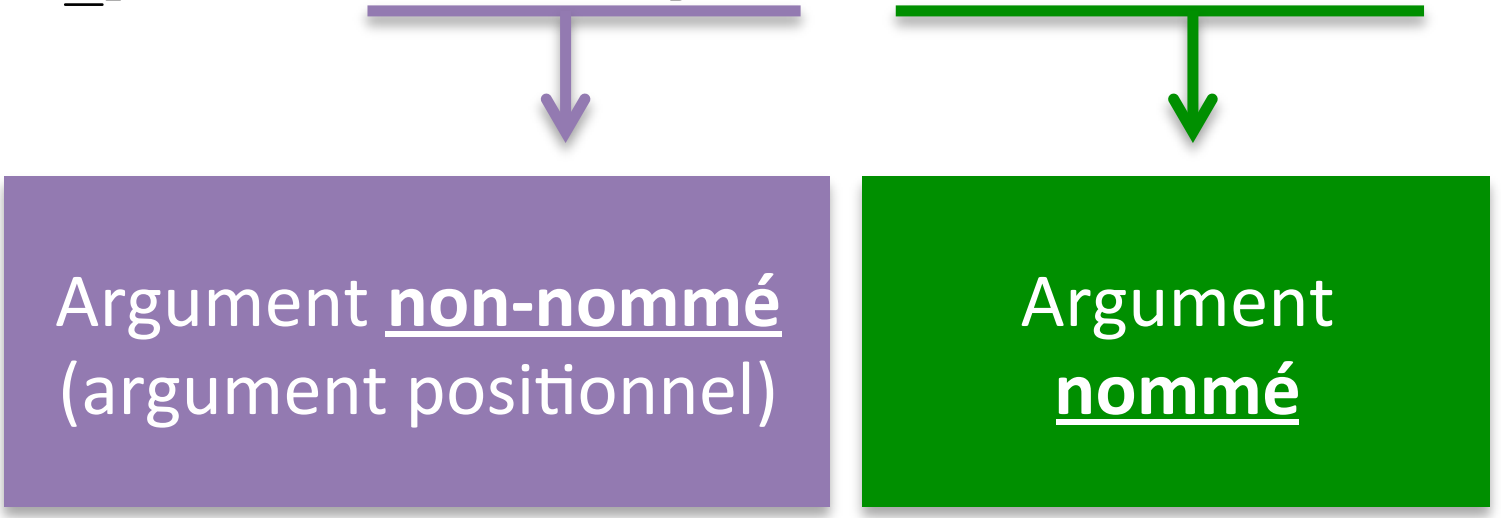
```
>>> affiche_pizza("4 fromages")
Pizza 4 fromages taille: normale
Prix 9 euros.

>>> affiche_pizza("4 fromages", "maxi")
Pizza 4 fromages taille: maxi
Prix 12 euros.

>>> affiche_pizza("Reine", "maxi", True)
Pizza Reine , taille:  maxi
Prix 12 euros ( 78.71  francs).

>>> affiche_pizza("4 saisons", True)
Erreur car True est pris pour la taille (2eme arg.)
```

Comment faire pour préciser le 3ème argument mais pas le 2ème?

Arguments nommés

Lors d'un appel de fonction, on peut préciser le nom de l'argument concerné :

```
>>> affiche_pizza("4 fromages", afficheF=True)
```



Les arguments non-nommés doivent toujours être **tous avant** les arguments nommés dans l'appel.
L'ordre des arguments nommés peut être fait selon votre préférence.

Appel de fonction

Arguments non-nommés puis Arguments nommés

```
>>> affiche_pizza("4 fromages", afficheF=True)
Pizza 4 fromages , taille:  normale
Prix 9 euros ( 59.04  francs).

>>> affiche_pizza("4 fromages", taille="maxi", afficheF=True)
Pizza 4 fromages , taille:  maxi
Prix 12 euros ( 78.71  francs).

>>> affiche_pizza("Chorizo", taille="maxi", True)
Erreur: il y a un argument non-nommé après un argument nommé

>>> affiche_pizza("Reine", afficheF=True, taille="maxi")
Pizza Reine , taille:  maxi
Prix 12 euros ( 78.71  francs).

>>> affiche_pizza("Chorizo", True, taille="maxi")
Erreur car taille est définie deux fois (True est pris pour
taille car 2eme argument non-nommé)
```

Arguments optionnels de print

En fait, nous avions déjà rencontré une fonction avec des arguments optionnels: `print`

```
>>> print("Mon age est", 18)
>>> print("Mon age est", 18, sep="égal à")
>>> print("Mon age est", 18, end=".")
>>> print("Mon age est", 18, sep=":", end=".")
```

`sep` et `end` sont des arguments optionnels de `print`. Par défaut, `sep` vaut `"  "` (espace) et `end` vaut `"\n"` (retour à la ligne).

Note: `sep` et `end` doivent toujours être nommés car la fonction `print` a un nombre variable d'arguments.

Le mot-clé **None**

Il existe une valeur constante en Python qui s'appelle **None**. Cela correspond à "rien", "aucune".

Lorsqu'une fonction n'a pas d'instruction `return`, elle renvoie la valeur `None`.

```
def dit_bonjour():
    print("Bonjour!")
    print("Bienvenue")
    # pas de return

# prog. Principal
test=dit_bonjour()
print("Test vaut", test)
```

Affichage lorsque l'on lance le module:

```
Bonjour!
Bienvenue.
Test vaut None
```

Le mot-clé **None**

Le mot-clé `None` peut aussi servir à initialiser une variable lorsque l'on ne sait pas encore précisément quelle valeur on souhaite lui attribuer.

```
reponse=None # on n'a pas encore de reponse
while reponse!="non":
    x=float(input("Veuillez entrer un nombre:"))
    print("Le carré de ce nombre est:", x*x)
    reponse=input("Voulez-vous recommencer?")
print("Terminé")
```

/!\ None n'est pas une chaîne de caractères en Python, donc il ne faut pas de guillemets.

- Ces slides ont été réalisés par:
  - Amir Charif
  - Lydie Du Bousquet
  - Aurélie Lagoutte
  - Julie Peyre
- Leur contenu est placé sous les termes de la licence **Creative Commons CC BY-NC-SA**