

cours_2

January 29, 2024

1 Cours 2

Notions

Dans cette partie, on explorera: - les fonctions prédéfinies, - la création de fonctions, leurs entrées et leurs sorties - les blocs conditionnels simples - les blocs conditionnels imbriqués

2 Fonctions prédéfinies

En informatique, une fonction est une séquence d'instructions regroupées sous un nom spécifique, et conçue pour effectuer une tâche particulière ou renvoyer un résultat. Les fonctions permettent de structurer le code en le divisant en morceaux plus petits et réutilisables.

Exemple: print() La fonction `print()` qu'on a utilisé au dernier cours est une fonction *prédéfinie* (aussi appelées *built-in functions*), c'est-à-dire qu'elle est intégrée au langage de programmation et peuvent être utilisées en tout temps dans le cadre de ce langage. voici plusieurs autres fonctions prédéfinies dans Python: https://www.w3schools.com/python/python_ref_functions.asp

Dans la fonction, on doit fournir des **arguments** qui seront utilisés dans cette fonction.

Pour la fonction `print()`, on devait mettre les éléments qu'on souhaitait voir apparaître dans la console entre les parenthèses. On peut aussi fournir plusieurs autres arguments si désiré:

Definition and Usage

The `print()` function prints the specified message to the screen, or other standard output device.

The message can be a string, or any other object, the object will be converted into a string before written to the screen.

Syntax

```
print(object(s), sep=separator, end=end, file=file, flush=flush)
```

Parameter Values

Parameter	Description
<code>object(s)</code>	Any object, and as many as you like. Will be converted to string before printed
<code>sep='separator'</code>	Optional. Specify how to separate the objects, if there is more than one. Default is ' '
<code>end='end'</code>	Optional. Specify what to print at the end. Default is '\n' (line feed)
<code>file</code>	Optional. An object with a write method. Default is <code>sys.stdout</code>
<code>flush</code>	Optional. A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False

3 Construction de fonctions

En plus des fonctions prédéfinies, il faudra également créer des fonctions adaptés à nos besoins. Il s'agira de faire un *traitement* de certaines données, appelé "l'intrant" et de fournir une sortie appelée "extrant" (en anglais, on verrait *input* et *output*). Il existe aussi des fonctions qui ne fournissent aucune sortie, qui ne font que des tâches spécifiques.

3.0.1 Exemples d'intrants / extrants:

Exemple 1 : Calcul de la somme de deux nombres

Intrants : Deux nombres, x et y

Extrants : La somme des deux nombres, x + y

Exemple 2 : Vérification de la validité d'un mot de passe

Intrants : Un mot de passe entré par l'utilisateur

Extrants : Un message indiquant si le mot de passe est valide ou non

Exemple 3 : Conversion de Celsius en Fahrenheit

Intrants : Une température en Celsius

Extrants : La température équivalente en Fahrenheit

On est très habitué à ce concept, on y est exposé dans la vie de tous les jours!

3.0.2 Syntaxe

Pour créer une fonction, - On écrit les instructions dans la fenêtre principale - On utilise la mot-clé **def** suivi du nom de la fonction et des **paramètres** dont elle aura besoin (de 0 à n paramètres) Les paramètres sont des valeurs qui sont passées à la fonction. Les paramètres sont optionnels. Si la fonction ne prend pas de paramètres, on doit quand même mettre les parenthèses. Cela permet de distinguer une fonction d'une variable. Les paramètres sont séparés par des virgules. Les paramètres sont des **variables locales** à la fonction. Elles ne sont pas accessibles à l'extérieur de la fonction. - On met toujours le symbole "deux points" (:) après la parenthèse fermante de la fonction. - On doit faire un retour de ligne et garder une **indentation** par rapport à la déclaration de la fonction. Celle-ci est primordiale en Python. > Le nombre d'espaces standard est de 4 ([Src : PEP 8](#)). Il est aussi possible de mettre des tabulations. Cependant, il faut être cohérent. Il est proscrit de mélanger les espaces et les tabulations. - On écrit ensuite les **instructions** - On termine avec le mot clé **return** suivi de la variable qui contient l'élément à retourner:

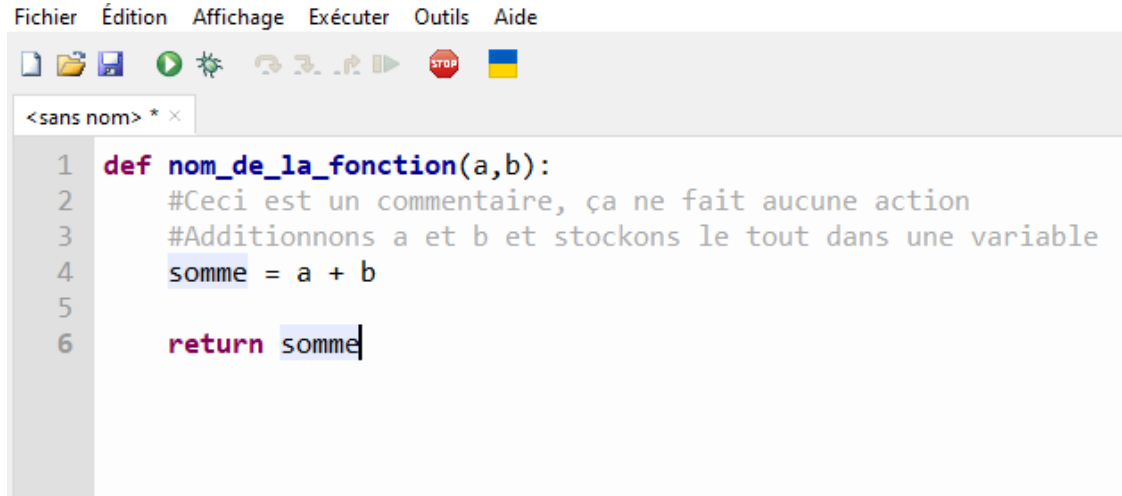
```
def nom_de_la_fonction([paramètres [, ...]]):  
    '''Documentation de la fonction (optionnel)'''  
    # Corps de la fonction  
    # ...  
    return valeur_de_retour # Optionnel
```

Note : Dans la littérature technique, les caractères [] indiquent que le contenu est optionnel. Dans la syntaxe ci-dessus, les paramètres sont optionnels. On peut donc écrire `def nom_de_la_fonction():` si la fonction ne prend pas de paramètres.

return est utilisé pour retourner une valeur à l'appelant. - **return** termine l'exécution de la fonction. - **return** peut retourner une valeur ou une expression. - **return** peut retourner plusieurs

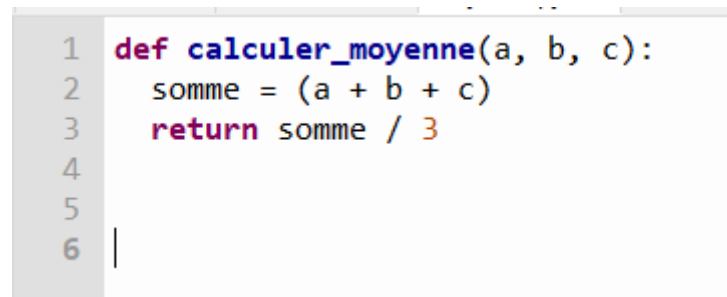
valeurs séparées par des virgules. - `return` peut être utilisé pour retourner une valeur avant la fin de la fonction. - `return` est optionnel. Si la fonction ne retourne pas de valeur, on peut omettre le `return`.

Exemple 1: sans opération dans le return



```
Fichier  Édition  Affichage  Exécuter  Outils  Aide
<sans nom> * x
1  def nom_de_la_fonction(a,b):
2      #Ceci est un commentaire, ça ne fait aucune action
3      #Additionnons a et b et stockons le tout dans une variable
4      somme = a + b
5
6      return somme
```

Exemple 2: avec opération dans le return



```
1  def calculer_moyenne(a, b, c):
2      somme = (a + b + c)
3      return somme / 3
4
5
6  |
```

3.0.3 Exercice 1:

Créer une fonction nommée *mon_hypot* qui prend en **paramètre** 2 éléments *a* et *b* et qui retourne la variable calculée *c* qui contient le calcul suivant:

$$c = \sqrt{(a^2 + b^2)}$$

3.0.4 Comment utiliser ma fonction?

Une fonction ne fait rien si elle n'est pas appelée!

Pour vérifier que la fonction fait bien le travail, il faut **appeler** cette fonction. En écrivant la fonction, on a créé les instructions, il faut maintenant les mettre en oeuvre. Dans la **fenêtre principale**, on appelle la fonction par son nom et on lui fournit des **arguments** (ceux-ci seront les paramètres de la fonction). Dans la fenêtre principale où est décrite la fonction, on entre:

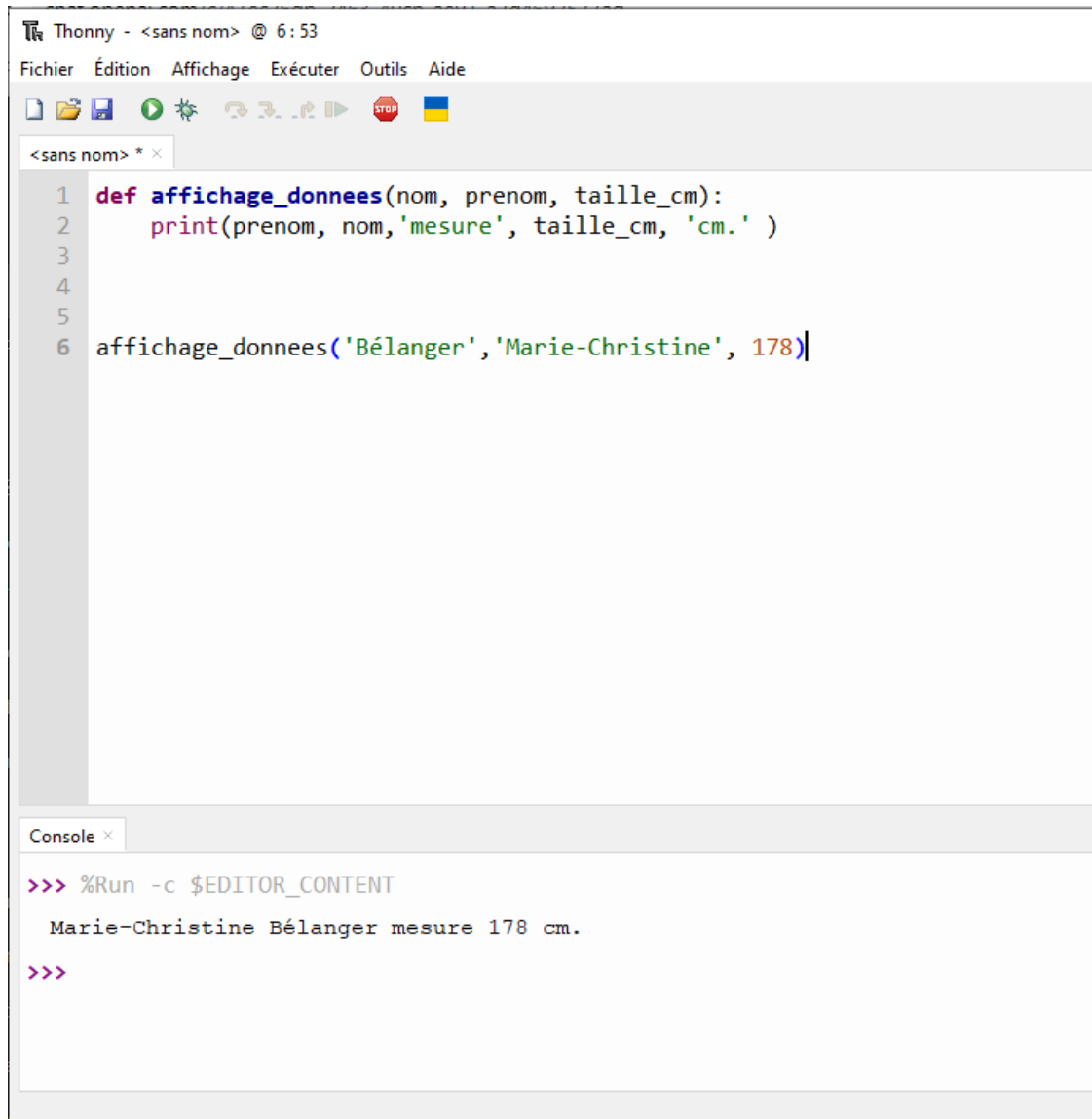
```
#Appel de la fonction mon_hypot avec a = 3 et b = 4
mon_hypot(3,4)
```

Puis on fait `run`. Que se passe-t-il?

3.0.5 Fonction sans valeur de retour (sans *return*)

Dans certains cas, on fera des fonctions sans retour. Celles-ci feront une tâche sans retourner de valeur. Ce sera le cas, par exemple, si on veut afficher un graphe, afficher des résultats dans la console, mesurer le temps d'exécution d'une fonction, enregistrer des données dans un fichier, afficher des erreurs à l'utilisateur, etc. Dans ce cas, on ne met pas le mot clé `return`.

Exemple:



The screenshot shows the Thonny Python IDE interface. The top menu bar includes 'Fichier', 'Édition', 'Affichage', 'Exécuter', 'Outils', and 'Aide'. Below the menu is a toolbar with icons for file operations, running, and debugging. The main editor window, titled '<sans nom> * x', contains the following Python code:

```
1 def affichage_donnees(nom, prenom, taille_cm):
2     print(prenom, nom, 'mesure', taille_cm, 'cm.' )
3
4
5
6 affichage_donnees('Bélanger', 'Marie-Christine', 178)|
```

The bottom panel, titled 'Console x', shows the output of the code execution:

```
>>> %Run -c $EDITOR_CONTENT
Marie-Christine Bélanger mesure 178 cm.
>>>
```

La ligne 2 fait partie de la fonction, mais la ligne 6 n'est plus dans la fonction. On le détermine à cause de l'indentation.

3.0.6 Import math

Pour accéder à plus d'opérateurs mathématiques, nous devons importer une librairie nommée `math` qui contient, entre autres, les fonctions ou valeurs suivantes: - `pi` - `max()` ou `min()` d'une série de nombres - `abs()` pour valeur absolue - `pow()` qui donne la puissance d'une base et de son exposant - `sqrt()` la racine carrée - `sin()`, `cos()`, `tan()`, etc. - toutes les autres ici: <https://docs.python.org/3/library/math.html>

Pour que ça fonctionne, il faut entrer la commande

```
#Dans les première lignes du fichier, avant les appels qui en ont besoin  
import math
```

dans le haut de votre fenêtre principale (avant l'endroit où vous en aurez besoin).

3.0.7 Exercice 2:

Créer une fonction nommée *volume* qui prend en **paramètre** le rayon d'un cercle et qui retourne le volume de la sphère qu'il engendre. Le volume d'une sphère est:

$$V = \frac{4}{3}\pi r^3$$

Vous pouvez faire [la question 1](#) des exercices du cours 2.

4 Structures conditionnelles

Les structures conditionnelles, ou structure de contrôle de flux, permettent de contrôler l'exécution du code. Elles permettent de faire des choix selon des conditions. Elles permettent aussi de répéter des instructions.

En français, on utilise les mots **si**, **sinon** et **sinon si** pour faire des choix. En programmation, on utilise les mots **if**, **else** et **elif** (*contraction de else if*).

On utilise parfois un diagramme à flux de données pour représenter les structures de contrôle de flux. Voici un exemple :

Voici la **structure générale** de contrôle de flux (blocs conditionnels) en Python:

```
if condition 1:  
    # Instructions si condition 1 vrai  
  
# Optionnel  
elif condition N:  
    # Instructions si condition N vrai  
  
# Optionnel  
else:  
    # Instructions si toutes les conditions sont fausses
```

Seul le **if** est obligatoire. On peut avoir plusieurs **elif** et un seul **else**. On peut aussi avoir seulement un **if** et un **else**.**

Exemple:

```

# Variables
age = 18

# Structure conditionnelle
if age < 5:
    print("Gratuit")
elif age < 12:
    print("Tarif enfant")
elif age < 65:
    print("Tarif adulte")
else:
    print("Tarif aîné")

```

Dans un `if`, il y a une condition. La condition est une expression booléenne. Une expression booléenne est une expression qui retourne `True` ou `False`. Si la condition est `True`, les instructions dans le `if` sont exécutées. Si la condition est `False`, les instructions dans le `if` sont ignorées.

```

# Variables
age = 18

if (age >= 18):
    print("Vous êtes un adulte")
else:
    print("Vous êtes mineur")

```

Voici le code qui représente le diagramme de flux montré plus haut :

```

# Variables
note = 70

if note <= 65:
    print ("Attention, risque d'échec!")

```

On peut mettre des structures conditionnelles à l'intérieur des fonctions.

Voici un exemple :

```

# Fonction indiquant si le pH est acide, neutre ou basique
def type_pH(pH):
    if pH < 7:
        print("Acide")
    elif pH > 7:
        print("Basique")
    else:
        print("Neutre")

# Appel de la fonction
type_pH(7.5)

```

Note : On remarque toujours l'indentation dans les structures de contrôle de flux. Cela facilite la lecture en bloc. On rappelle que l'indentation est obligatoire en Python.

4.0.1 Opérateurs de comparaison

Les expressions conditionnelles utilisent des opérateurs de comparaison. Voici un tableau des opérateurs de comparaison de base :

Opérateur	Description	Exemple
<code>==</code>	Égalité	<code>a == b</code>
<code>!=</code>	Différence	<code>a != b</code>
<code>></code>	Plus grand que	<code>a > b</code>
<code><</code>	Plus petit que	<code>a < b</code>
<code>>=</code>	Plus grand ou égal à	<code>a >= b</code>
<code><=</code>	Plus petit ou égal à	<code>a <= b</code>
<code>not</code>	Négation	<code>not a</code>
<code>and</code>	ET logique	<code>a and b</code>
<code>or</code>	OU logique	<code>a or b</code>

Dans tous les cas, l'expression retourne `True` ou `False`. `a` ou `b` peuvent être des variables ou des expressions.

Voici un exemple :

```
# Variables
age = 18
ami = True

# Structure de contrôle de flux
if age >= 18 and ami:
    print("On va au Trou du Diable!")
else:
    print("On va dans ton sous-sol!")
```

Attention erreur fréquente! Dans les expressions conditionnelles, on utilise `==` pour vérifier l'égalité. On utilise `=` pour assigner une valeur à une variable. Si votre code ne fonctionne pas, vérifiez que vous n'avez pas utilisé `=` au lieu de `==`.

Table de vérité Voici un petit rappel de la table de vérité pour les opérateurs logiques `and` et `or` :

a	b	a and b	a or b
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

En résumé pour le ET, il faut que les deux conditions soient vraies pour que l'expression soit vraie. Pour le OU, il faut qu'au moins une des conditions soit vraie pour que l'expression soit vraie.

4.1 La valeur None

En Python, il y a une valeur spéciale qui est `None`. Cette valeur est utilisée pour indiquer qu'une variable n'a pas de valeur. On peut l'utiliser pour initialiser une variable. On peut aussi l'utiliser pour indiquer qu'une fonction ne retourne pas de valeur.

Elle peut être utilisée pour initialiser une variable que l'on va utiliser plus tard. Par exemple, on peut utiliser `None` pour initialiser une variable qui va contenir le résultat d'un calcul.

Exemple d'utilisation du None:

Voici un exemple dans lequel on a une fonction $PV = nRT$ dépendant des données d'entrée :

```
# Variables
R = 0.0821 # Constante des gaz parfaits (L.atm/mol.K)

# Essayez de remplacer une valeur par None pour
# voir le résultat
n = 2.5      # Moles de gaz
T = 300      # Température en Kelvin
V = 10       # Volume en litres
P = None     # Pression en atm

msg = ''

# Structure de contrôle de flux
if n == None:
    # Cas où n est l'inconnue
    n = (P * V) / (R * T)
    msg = "Moles de gaz (n) :" + str(n)
elif T == None:
    # Cas où T est l'inconnue
    T = (P * V) / (R * n)
    msg = "Température (T) :" + str(T) + " K"
elif V == None:
    # Cas où V est l'inconnue
    V = (n * R * T) / P
    msg = "Volume (V) :" + str(V) + " L"
elif P == None:
    # Cas où P est l'inconnue
    P = (n * R * T) / V
    msg = "Pression (P) :" + str(P) + " atm"
else:
    # Cas où il y a une erreur
    msg = "Revérifiez vos données d'entrée. Il y a une erreur, car toutes les variables sont d

# Affichage du résultat
print ("Voici le résultat :")
print (msg)
```


5 Blocs conditionnels imbriqués

Il est possible de faire un bloc conditionnel dans un autre bloc conditionnel. C'est une bonne façon de réduire les répétitions de code

Exemple: Reprenons la structure de décision de la sortie à faire entre amis:

```
# Variables
age = 18
ami = True

# Structure de contrôle de flux
if ami:
    if age >= 18:
        print("On va au Trou du Diable!")
    else:
        print("On va dans ton sous-sol!")
else:
    print("On apprendra à se connaître avant de faire une activité")
```

On met donc des instructions et vérifications supplémentaires si les personnes sont amies ou non. Il y a une alternative si les personnes ne sont pas amies. Les if doivent avoir la même **indentation** que leurs else (ou elif) respectifs.

5.1 Quelques [exercices](#) pour se pratiquer

[]: