Labo d’introduction à l’informatique

Yann Thorimbert

Semaine 5
Fonctions

Un code embarrassant

```
maj = 18 #Situation 1 (Suisse)
age = int(input("Entrez votre âge:"))
if age < maj:
    print("Vous êtes mineur en Suisse.")
else:
    print("Vous êtes majeur en Suisse.")

maj = 21 #Situation 2 (Honduras)
age = int(input("Entrez votre âge:"))
if age < maj:
    print("Vous êtes mineur au Honduras.")
else:
    print("Vous êtes majeur au Honduras.")

maj = 16 #Situation 3 (Allemagne)
age = int(input("Entrez votre âge:"))
if age < maj:
    print("Vous êtes mineur en Allemagne.")
else:
    print("Vous êtes majeur en Allemagne.")
```

Un code embarrassant | **Une première alternative**

```
def check_maj():
    age = int(input("Entrez votre âge:"))
    if age < maj:
        print(f"Vous êtes mineur {fin_phrase}.")
    else:
        print(f"Vous êtes majeur {fin_phrase}.")

maj = 18
fin_phrase = "en Suisse"
check_maj()

maj = 21
fin_phrase = "au Honduras"
check_maj()

maj = 16
fin_phrase = "en Allemagne"
check_maj()
```

On **définit** une fonction (bloc de code délimité par l'indentation)

La fonction se nomme `check_maj`.

Un code embarrassant | **Une première alternative**

```
def check_maj():
    age = int(input("Entrez votre âge:"))
    if age < maj:
        print(f"Vous êtes mineur {fin_phrase}.")
    else:
        print(f"Vous êtes majeur {fin_phrase}.")

maj = 18
fin_phrase = "en Suisse"
check_maj()

maj = 21
fin_phrase = "au Honduras"
check_maj()

maj = 16
fin_phrase = "en Allemagne"
check_maj()
```

On **appelle** (utilise) la fonction.

Un code embarrassant | **Une première alternative**

```
def check_maj():
    age = int(input("Entrez votre âge:"))
    if age < maj:
        print(f"Vous êtes mineur {fin_phrase}.")
    else:
        print(f"Vous êtes majeur {fin_phrase}.")

maj = 18
fin_phrase = "en Suisse"
check_maj()

maj = 21
fin_phrase = "au Honduras"
check_maj()

maj = 16
fin_phrase = "en Allemagne"
check_maj()
```

Ici `age` n'existe qu'au sein de la fonction `check_maj` (portée "locale"), tandis que `maj` et `fin_phrase` sont des variables dites "globales" (déconseillé)

Un code embarrassant | **Une meilleure alternative**

```
def check_maj(maj, fin_phrase):
    age = int(input("Entrez votre âge:"))
    if age < maj:
        print(f"Vous êtes mineur {fin_phrase}.")
    else:
        print(f"Vous êtes majeur {fin_phrase}.")

check_maj(18, "en Suisse")
check_maj(21, "au Honduras")
check_maj(16, "en Allemagne")
```

Cette fois-ci, `check_maj` possède deux **paramètres** : `maj` et `fin_phrase`.

Les paramètres sont traités comme des variables "normales" au sein de la fonction.

Au moment de l'appel, on spécifie la valeur des paramètres : ce sont les **arguments** que l'on passe.

Fonctions sans valeur de retour

- Exemple : fonction uniquement destinée à afficher un graphique, une image, ou a agir sur des variables passées en argument ou par **effet de bord** (modifier une variable globale)**.**
- Dans ne nombreux langages, on parle de **procédure** dans ce cas.
- En Python, une fonction qui n'a pas de return retourne None par défaut (implicite).
- Exemple en Python :

```
def dire_bonjour(x): #Cette fonction retourne None
    print("Bonjour", x)
```

Qu'est-ce que None ?

- Permet d'exprimer "rien" ou un état indéfini.
- Différent de zéro ou d'une liste vide.
- Exemple :

```
prix = None
produit = input("Que voulez-vous acheter ? Eau ou soda ?")
if produit == "eau":
   prix = 1.5
elif produit == "soda":
   prix = 2.5
if prix is None: #None est un objet prédéfini de Python
   print("Le produit indiqué n'existe pas")
else:
   print("Le prix est", prix)
```

Fonctions avec valeur de retour

- "Bout de code" qui peut être réutilisé depuis d'autres scripts.
- Effectue une suite d'instructions sur les paramètres d'entrée.
- Peut **retourner** une valeur.
- Exemple de **déclaration** de fonction :

```
def cube(x):
    return x * x * x
```

- Exemple **d'utilisation** de fonction :
  cube(3) # retourne 27

Fonctions avec valeur de retour | **Exemple**

```
def prix_billet_bus(zone, age): #prix fantaisistes
    if age < 8:
        prix = 0
    elif zone == 1:
        prix = 3
    else:
        prix = 5
    if age < 25:
        prix = prix/2
    return prix

prix = prix_billet_bus(1, 18)
print("Prix de 2 billets:", prix*2)
print("Prix de 15 billets:", prix*15)
```

Commentaires divers

```
def prix_billet_bus(zone, age): #prix fantaisistes
    if age < 8:
        prix = 0
    elif zone == 1:
        prix = 3
    else:
        prix = 5
    if age < 25:
        prix = prix/2
    return prix

prix = prix_billet_bus(1, 18)
print("Prix de 2 billets:", prix*2)
print("Prix de 15 billets:", prix*15)
```

Et si l'on veut vérifier que la zone vaut soit 1, soit 2 (sinon invalide ?)

Fonctions avec valeur de retour

Plusieurs return par fonction sont possibles :

```
def ma_valeur_absolue(x):
    if x < 0:
        return -x
    else:
        return x
```

Fonctions avec valeur de retour

Une fois un return atteint, le flux d'exécution est interrompu et on revient à l'endroit du code où l'appel a été effectué.

```
def ma_fonction():
    print("Salut")
    return 15
    print("Ce texte ne s'affichera jamais")
```

Fonctions avec valeur de retour

Plusieurs return par fonction sont possibles :

```
def ma_valeur_absolue(x):
    if x < 0:
        return -x
    else:
        return x
```

Commentaires finaux

- Une fonction peut s'appeler elle-même (fonction dite "récursive"). Nous y reviendrons dans d'autres TPs.

- Une fonction peut tout à fait appeler d'autres fonctions.

- Vous utilisez des fonctions depuis le début : `input`, `print`, `int`, `float`, ...

Remarque importante | **Mutabilité**

- Nous verrons plus loin dans le cours qu'en Python, certains types sont mutables et d'autres ne le sont pas.
- Une variable d'un type mutable peut être modifiée, tandis qu'une variable immutable ne le peut pas.
- Observez le code suivant :

```
def ajoute_cinq_au_nombre(x):
    x = x + 5

def ajoute_cinq_a_la_liste(x):
    x.append(5)
```

Remarque importante | **Types immutables**

Observez le code suivant :

```
def ajoute_cinq_au_nombre(x):
    x = x + 5
    return x

a = 2
b = ajoute_cinq_au_nombre(a)
print(a, b) #on voit que a n'a pas été modifié !
#Les bool, int, float et str sont tous immutables.
#Nous reviendrons à cela dans un autre TP
```

Remarque importante | **Types mutables**

Observez le code suivant :

```
def ajoute_cinq_a_la_liste(x):
    x.append(5)
    return x

a = [2, 8, 1]
b = ajoute_cinq_a_la_liste(a)
print(a, b) #on voit que cela a modifié a !
#Les listes sont des objets mutables.
#Nous reviendrons à cela dans un autre TP
```

⇒ Pour le moment, évitez de modifier une liste au sein d'une fonction !