Nom :	Prénom :	Classe :

SNT — Fonction : récapitulatif

Objectifs :
- Comprendre le rôle d'une fonction.
- Savoir schématiser une fonction, sous la forme d'un prototype.
- Savoir définir une fonction, en Python.
- Savoir appeler une fonction, en Python.

Rôle d'une fonction

Une fonction n'est ni plus ni moins qu'un morceau de code. Mais c'est un **morceau de code** que l'on peut **utiliser** et **réutiliser** à tout moment dans un programme.

Création d'une fonction

Voici par exemple une fonction définie sous le nom de `premiere_fonction` :

```
1  def premiere_fonction():
2      return "Je suis une fonction"
```

Voilà, nous venons de **créer** (définir) une fonction dont le nom est `premiere_fonction`.

Utilisation d'une fonction

Une fois qu'une fonction est définie — avec le mot-clé `def` — nous pouvons, **après**, l'utiliser.

Pour utiliser une fonction, il nous suffit juste d'en taper le nom suivi des parenthèses :

```
3  premiere_fonction()
```

Voilà, nous venons **d'utiliser** (appeler) une fonction dont le nom est `premiere_fonction`.

Mais il ne s'est rien passé ! Rien n'est affiché !
Pourtant notre fonction contient un « return » ...

En fait, la fonction a bien été utilisée (appelée), mais nous n'avons rien fait de la valeur qu'elle renvoie.

... Alors comment **afficher** ce que la fonction **renvoie** lorsqu'elle est **appelée** ?

Comme ceci :

```
4  print( premiere_fonction() )
```

Ce qui affiche bien, cette fois : `Je suis une fonction`

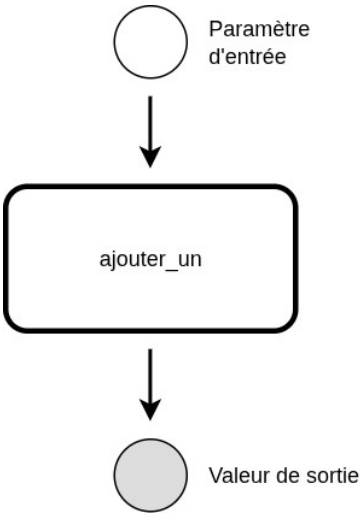
Création d'une fonction prenant un paramètre

Une fonction est surtout intéressante lorsque nous pouvons lui transmettre une valeur, comme un chiffre par exemple. Pour pouvoir lui transmettre une valeur, il faut, lorsque nous définissons notre fonction, lui ajouter un **paramètre**.

Prenons un exemple de création de fonction prenant un paramètre, à partir de **l'énoncé** suivant :

> Nous souhaitons créer une fonction nommée `ajouter_un` qui permet d'ajouter 1 à toutes les valeurs que nous pouvons lui donner.
>
> Par exemple, en lui donnant la valeur 100, cette fonction nous renverrait 101.
> Ou encore, en lui donnant une valeur 179, cette fonction nous renverrait 180.

Essayons d'imaginer à quoi pourrait ressembler cette fonction en la schématisant (en la **prototypant**) :



Donc, lorsque le paramètre d'entrée est 100, la fonction aurait 101 comme valeur de sortie...

Ou encore, lorsque le paramètre d'entrée est 179, la fonction aurait 180 comme valeur de sortie...

Essayons maintenant de **définir cette fonction en Python** :

```
1  def ajouter_un(a):
2      return a + 1
```

Voilà, nous venons de créer (définir) une fonction dont le nom est `ajouter_un`.
Cette fonction prend un paramètre. Paramètre que nous avons nommé ici `a`.

Cette fonction renvoie la valeur qui sera stockée dans `a`, additionnée de `1`.

Pour info nous aurions pu nommer notre paramètre comme bon nous semble, par exemple :

```
1  def ajouter_un(zaza):
2      return zaza + 1
```

Cette fonction fait exactement la même chose que la précédente. On a juste changé le nom du paramètre.

Bon, maintenant que notre fonction `ajouter_un` est définie, nous pouvons l'utiliser.

Pour utiliser une fonction, il nous suffit juste d'en taper le nom suivi des parenthèses :

```
3  ajouter_un()
```

Sauf que cette fois-ci, Python signale une erreur !

```
TypeError: ajouter_un() missing 1 required positional argument: 'a'
```

Qu'avons-nous oublié ? ...

Nous avons **défini une fonction** qui **prend un paramètre**.
Donc lorsque nous souhaitons l'appeler (l'exécuter),
il nous faut lui donner une valeur pour ce paramètre !

Réessayons, mais cette fois en donnant une valeur au paramètre :

```
4  ajouter_un(100)
```

Ouf, cette fois pas d'erreur signalée par Python.

Par contre, rien ne s'affiche, normal ?

En fait, la fonction a bien été utilisée (appelée), mais nous n'avons rien fait de la valeur qu'elle renvoie.

... Alors comment afficher ce que la fonction renvoie lorsqu'elle est appelée ? Comme ceci :

```
4  print( ajouter_un(100) )
```

Ce qui affiche bien, cette fois : `101`

Super, essayons d'appeler à nouveau notre fonction, mais en donnant une autre valeur au paramètre :

```
4  print( ajouter_un(179) )
```

Ce qui affiche : `180`

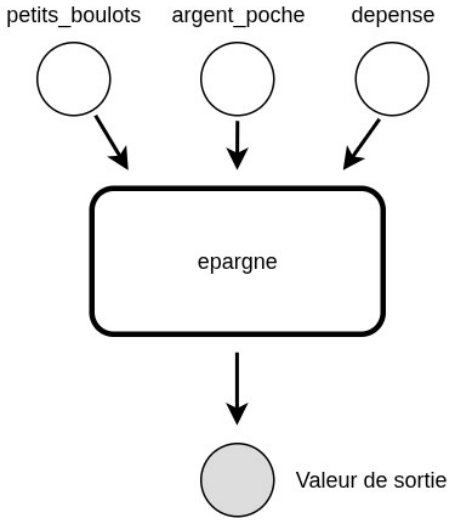
Voilà comment créer (définir) puis utiliser (appeler) une fonction qui prend un paramètre !

Création d'une fonction prenant plusieurs paramètres

Penchons-nous à présent sur un exemple de fonction prenant plusieurs paramètres, à partir de l'énoncé suivant :

> Nous souhaitons créer une fonction nommée `epargne` qui prend `petits_boulots`, `argent_poche` et `depense` comme paramètres.
>
> *Cette fonction renvoie une valeur de sortie correspondant au résultat du calcul suivant :*
> valeur de `petits_boulots` + valeur de `argent_poche` - valeur de `depense`.
>
> Par exemple, en lui donnant les valeurs 70, 30, 10, cette fonction nous renverrait 90.
> Ou encore, en lui donnant les valeurs 130, 10, 40, cette fonction nous renverrait 100.

Essayons d'imaginer à quoi pourrait ressembler cette fonction en la prototypant :



Essayons maintenant de définir cette fonction en Python :

```
1  def epargne(petits_boulots, argent_poche, depense):
2      return petits_boulots + argent_poche - depense
```

Maintenant que cette fonction est définie, nous pouvons l'utiliser :

```
3  epargne(70, 30, 10)
```

Zut, j'ai oublié de *print* ce que la fonction renvoie. Je modifie ma ligne :

```
3  print( epargne(70, 30, 10) )
```

Ce qui affiche `90`.

Voilà comment créer (définir) puis utiliser (appeler) une fonction qui prend plusieurs paramètres !

Et si on s'amusait à créer une fonction utile ?

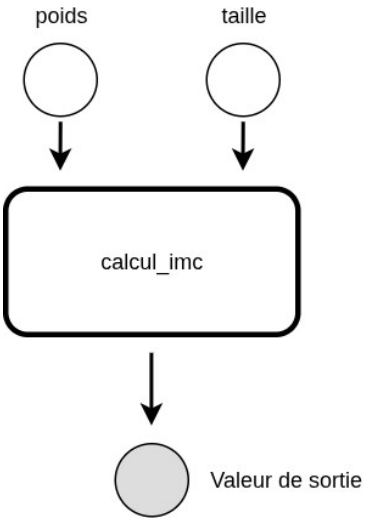
Essayons à présent de créer une fonction qui pourrait être utile dans la vie de tous les jours …

L'indice de masse corporelle (abrégé en IMC) est un calcul servant à classifier rapidement le poids d'une personne par rapport à sa taille.
Selon l'Organisation mondiale de la santé : indice IMC = Poids (en kilogrammes) ÷ Taille² (en mètres).

Énoncé :

> Nous souhaitons créer une fonction nommée `calcul_imc` qui prend `poids` et `taille` comme paramètres.
>
> Cette fonction renvoie une valeur d'IMC, c'est à dire le résultat de l'opération `poids` ÷ `taille`²

Essayons d'imaginer à quoi pourrait ressembler cette fonction en la prototypant :



Essayons maintenant de définir cette fonction en Python :

```
1  def calcul_imc(poids, taille):
2      return poids / taille**2
```

Maintenant que cette fonction est définie, nous pouvons l'utiliser :

Judith pèse par exemple 63 kg et mesure 1,74 m :

```
3  judith = calcul_imc(63, 1.74)
   print(judith)  # Ce qui affiche « 20.808561236623067 »
```

Donc l'indice IMC de Judith est d'environ 20,81 ; Super, alors, qu'en dit l'OMS[1] ?

Et vous, quel est **votre indice** IMC ?
Pour le savoir, vous pouvez utiliser ce bac à sable Python.

1 Voir le chapitre « Interprétation » de l'article « Indice de masse corporelle » sur Wikipédia.

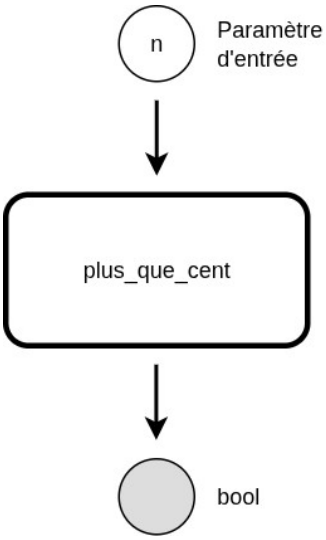
Dans sa définition, une fonction peut contenir tout type de code

Pour terminer, voyons qu'il est possible d'utiliser *tout type de code* lors de la définition d'une fonction. C'est à dire que nous mettre dans son *bloc de code* des *conditions*, des *boucles bornées*, des *boucles non bornées*, etc.

Prenons un exemple de création de fonction qui nécessite l'usage d'une condition, à partir de l'énoncé suivant :

> Nous souhaitons créer une fonction `plus_que_cent` qui permet de savoir si un entier donné est strictement supérieur à 100, ou pas.
>
> Cette fonction prend un paramètre `n`, et retourne `True` si `n` est strictement supérieur à 100, ou `False` sinon.
>
> Par exemple, en lui donnant la valeur 200, cette fonction nous renverrait True.
> Ou encore, en lui donnant la valeur 50, cette fonction nous renverrait False.

Essayons d'imaginer à quoi pourrait ressembler cette fonction en la prototypant :



Essayons maintenant de définir cette fonction en Python :

```
1  def plus_que_cent(n):
2      if n > 100:
3          return True
4      else:
5          return False
```

Maintenant que cette fonction est définie, nous pouvons l'utiliser (l'appeler)
sans oublier de « print » ce qu'elle renvoie :

```
6  print( plus_que_cent(200) )
```

Ce qui affiche `True`.
... Et si nous l'avions utilisée en faisant `print( plus_que_cent(50) )` cela aurait affiché `False`.