Nom :	Prénom :	Classe :

NSI 1re — Fonction : prototypage, création et utilisation

Objectifs :
- Savoir schématiser (prototyper) une fonction, à partir d'une description donnée.
- Savoir créer une fonction en Python, à partir d'une description donnée.
- Savoir utiliser une fonction Python, dans différents contextes.

Introduction

Une **fonction** est une *portion de code* effectuant une *tâche précise*.
Par exemple, la tâche précise de la fonction `print` est d'afficher une valeur donnée.

Nous avons déjà utilisé de nombreuses fonctions incluses dans Python (*fonctions natives* ou fonctions *built-in*) : `range`, `input`, `print`, `int`, `float`, `bool`, `str`, `len`, etc.

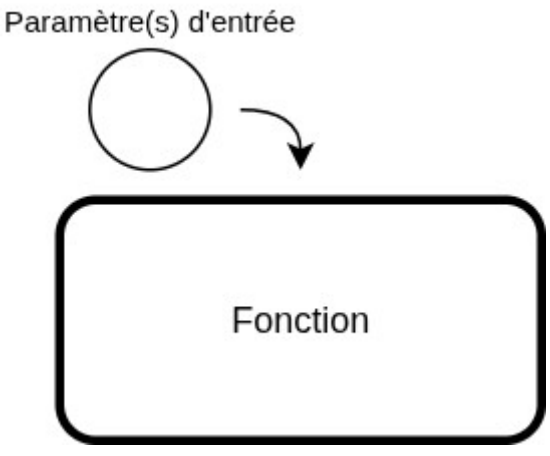
Dans ce cours, nous allons voir qu'il est également possible de se créer des *fonctions personnalisées*.

Paramètre(s) d'entrée

Un **paramètre d'entrée** est une **valeur** *transmise* à une fonction.
La fonction utilise cette valeur dans les différentes instructions qu'elle contient.
Une fonction accepte un ou plusieurs paramètres d'entrée.

Exemples :

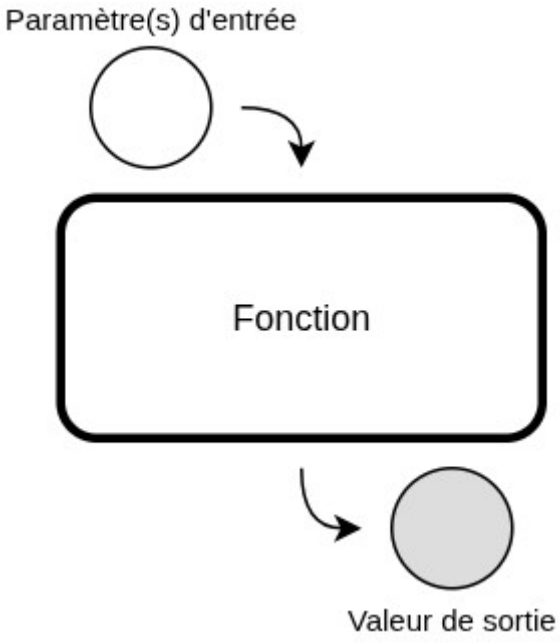
- La fonction `int`[1] accepte un paramètre d'entrée (qui est la valeur à transformer en entier).
- La fonction `range`[2] accepte un ou plusieurs paramètres d'entrée.



1 Voir le cours « Conversion de types + Commande input() ».
2 Voir le cours « Boucle bornée (boucle for) #2 : utilisations de range() ».

Valeur de sortie

La fonction, une fois qu'elle a traité le ou les paramètres d'entrée, parvient à un résultat.
Ce résultat est généralement *renvoyé* par la fonction, c'est ce qu'on appelle la **valeur de sortie**.



Exemple 1 :

```
1  >>> int(3.14)
2  3
```

Dans ce premier exemple, nous avons exécuté (*appelé*) la fonction `int`.
- Le **paramètre d'entrée** est la valeur `3.14`.
- La **valeur de sortie** est `3`.

Exemple 2 :

```
1  >>> a = float(18)
2  >>> print(a)
3  18.0
```

Dans ce deuxième exemple, nous avons appelé la fonction `float`.
- Le **paramètre d'entrée** est la valeur `18`.
- La **valeur de sortie** est stockée dans une variable `a` (ligne 1).

Puis nous avons appelé la fonction `print`.
- Le paramètre d'entrée est la valeur stockée dans `a`.

La fonction print nous a permis d'afficher la valeur stockée dans `a`.

➤ **Exercice 1** — Inventorier

Inventorier le ou les paramètres d'entrée et la valeur de sortie dans les appels suivants :

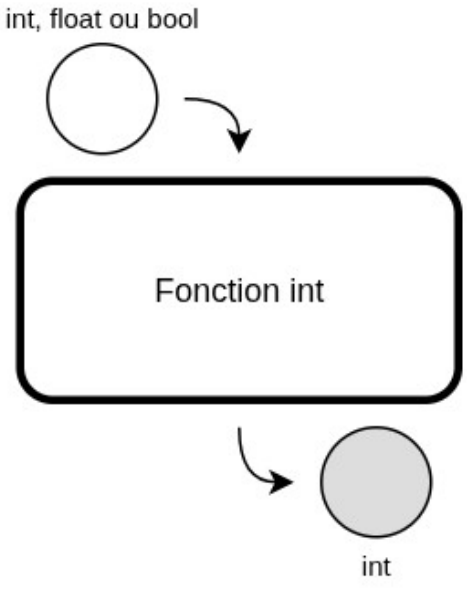
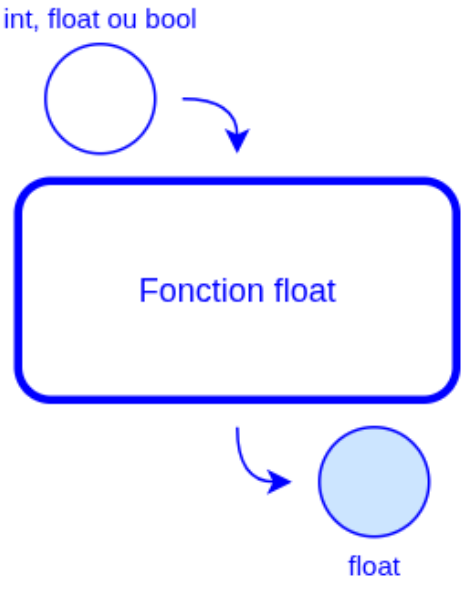
N°	Code Python	Paramètre(s) d'entrée	Valeur de sortie
1	`>>> bool(0)` `False`	`0`	`False`
2	`>>> str(18)` `'18'`	`18`	`'18'`
3	`>>> len([18, 23, 72])` `3`	`[18, 23, 72]`	`3`

Prototypage

Prototyper une fonction, c'est schématiser à l'avance le ou les paramètres d'entrée acceptés et la valeur de sortie, en précisant — si possible — leurs types.

➤ **Exercice 2** — Schématiser

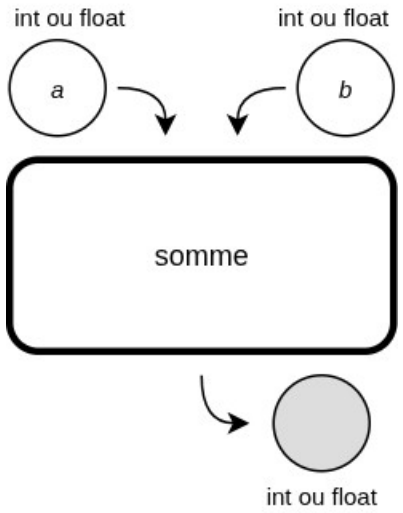
À partir de l'exemple donné pour la fonction `int`, schématiser le prototype de la fonction `float`.

Descriptif	La fonction `int` prend comme paramètre d'entrée un entier, un flottant ou un booléen, et renvoie un entier.	La fonction `float` prend comme paramètre d'entrée un entier, un flottant ou un booléen, et renvoie un flottant.
Prototype		

Plusieurs paramètres d'entrée

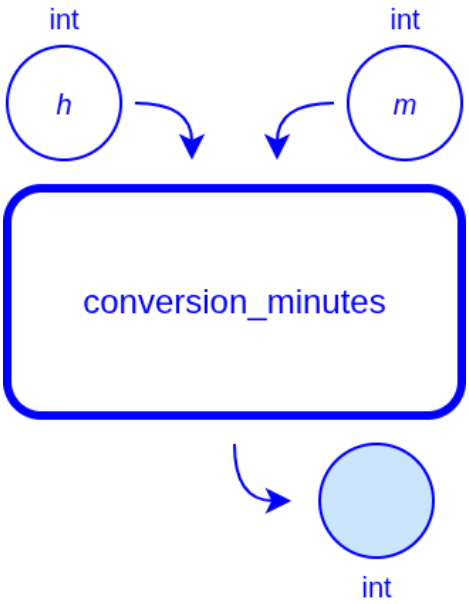
Notons qu'il est également possible de schématiser le prototype d'une fonction acceptant plusieurs paramètres d'entrée.

L'exemple ci-dessous illustre cette possibilité, en réalisant le schéma d'une fonction, à partir d'un descriptif :

Descriptif	Prototype
Soit `somme` la fonction qui prend comme paramètres deux entiers ou flottants `a` et `b`. Cette fonction renvoie le résultat entier ou flottant de l'addition de `a` et `b`.	

➤ **Exercice 3** — Schématiser

Schématiser le prototype de la fonction décrite ci-dessous :

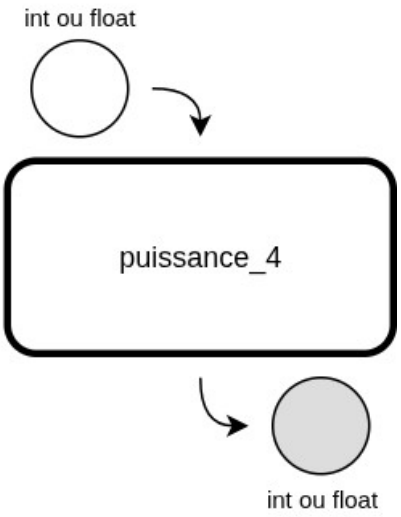
Descriptif	Prototype
Soit `conversion_minutes` la fonction qui prend comme paramètres deux entiers `h` et `m`. Cette fonction renvoie le résultat entier de la conversion de `h` (nombre d'heures) et `m` (nombre de minutes) en minutes.	

Code Python d'une fonction

La création d'une fonction en Python possède une structure *relativement proche* de celles déjà étudiées pour la création d'une condition ou d'une boucle.

Penchons-nous par exemple sur la création d'une nouvelle fonction nommée « puissance_4 ».

Voici le **descriptif** et le **schéma** de cette fonction :

Descriptif	Prototype
Soit `puissance_4` la fonction qui prend comme paramètre un entier ou flottant `a`. Cette fonction renvoie le résultat entier ou flottant correspondant au calcul de la valeur de `a` élevée à la puissance 4.	

Et en voici le **code Python** :

```
1  def puissance_4(a):
2      c = a ** 4
3      return c
```

Que pouvons-nous observer à propos de la structure du code ci-dessus ?

- La création (on peut aussi dire *la définition*) d'une nouvelle fonction repose sur l'utilisation du **mot-clé** `def` suivi du **nom de la fonction** à créer. *Ici le nom de la fonction est « puissance_4 ».*
- Le nom de la fonction ainsi définie est **toujours suivi de parenthèses**.
  Ces parenthèses contiennent le ou les noms des **paramètres d'entrée**. *Ici son nom est « a ».*
- Les parenthèses sont **toujours suivies du symbole** `:`
- Puis, tout le **bloc de code** de la fonction est indenté d'un niveau.
- Le bloc de code se termine généralement par une instruction `return` qui permet de "faire sortir" (on dit aussi *renvoyer*) une valeur de sortie de la fonction. *Ici on renvoie la valeur stockée dans* `c`.

Création et appel de fonction Python

Définir une fonction permet de l'appeler **à tout moment** et **autant de fois que désiré** dans un programme.

Par exemple, imaginons que nous ayons défini la fonction `ajoute_tva` suivante :

```
1  def ajoute_tva(prix_ht):
2      c = prix_ht + prix_ht * 20 / 100
3      return c
```

Une fois définie, nous pouvons l'appeler autant de fois que nous le désirons :

```
..
5  console_switch2 = ajoute_tva(320)
6  console_ps5 = ajoute_tva(350)
```

La création et l'appel de fonction permet de créer des programmes plus flexibles, en les découpant en « morceaux réutilisables ».

➤ **Exercice 4** — Appeler

Écrire le code correspondant aux descriptions données :

N°	Description	Code Python
1	Appeler la fonction `ajoute_tva` avec `536` comme paramètre d'entrée. Stocker le résultat dans une variable `iphone_15`.	`iphone_15 = ajoute_tva(536)`
2	Appeler la fonction `ajoute_tva` avec `463` comme paramètre d'entrée. Stocker le résultat dans une variable `drone_neo2`.	`drone_neo2 = ajoute_tva(463)`
3	Appeler la fonction `ajoute_tva` avec `320` comme paramètre d'entrée. Stocker le résultat dans une variable `pixel_9`.	`pixel_9 = ajoute_tva(320)`

➤ **Exercice 5** — Créer

Écrire le code Python correspondant aux différentes situations décrites :

N°	Description de situation	Code Python

1	Une entreprise d'origine suédoise spécialisée dans la conception et la vente de mobilier et objets de décoration vous a sollicité pour créer une fonction `volume_pave` permettant de calculer le volume d'un semi-remorque. Cette fonction prend les paramètres `longueur`, `largeur` et `hauteur` et renvoie le volume correspondant.
	
	Pour rappel, le volume d'un pavé se calcule en faisant longueur × largeur × hauteur.
		`def volume_pave(longueur, largeur, hauteur):`
		`  res = longueur * largeur * hauteur`
		`  return res`

2	Un camarade qui n'a jamais écrit de Python aimerait savoir comment créer une fonction `perimetre_cercle` qui prend `r` comme paramètre et qui renvoie le résultat du calcul correspondant au périmètre d'un cercle de rayon `r`.
	
	Pour rappel, le périmètre d'un cercle se calcule en faisant 2 × 3.14 × rayon.
		`def perimetre_cercle(r):`
		`  p = 2 * 3.14 * r`
		`  return p`

3	Une enseigne spécialisée dans la commercialisation de biens et loisirs culturels et créatifs vous a contacté pour créer une fonction `ajoute_tva` permettant de calculer des prix « toutes taxes comprises » (TTC). Cette fonction prend les paramètres `prix_ht` et `tva` et renvoie le résultat du calcul correspondant à l'ajout du taux de TVA `tva` à `prix_ht`.	`def ajoute_tva(prix_ht, tva):`
		`  ttc = prix_ht + prix_ht * tva / 100`
		`  return ttc`