Python : Fonctions

1ère NSI

Plan du cours

Introduction

- En programmation, une fonction peut se définir comme une **suite d'instructions pré-enregistrées**, amenées à être exécutées à plusieurs reprises.

Introduction

- En programmation, une fonction peut se définir comme une **suite d'instructions pré-enregistrées**, amenées à être exécutées à plusieurs reprises.

- La plupart des fonctions nécessitent l'utilisation de **paramètres** (ou arguments) servant à modifier leur comportement.

Introduction

- En programmation, une fonction peut se définir comme une **suite d'instructions pré-enregistrées**, amenées à être exécutées à plusieurs reprises.
- La plupart des fonctions nécessitent l'utilisation de **paramètres** (ou arguments) servant à modifier leur comportement.
- L'**appel** à une fonction se fait toujours de la même manière : `nomDeLaFonction(parametres)` .

Afficher avec `print`

- La fonction la plus commune est `print` : elle permet d'afficher une variable, quelque soit son type.

```
1 >>> a = 2*3
2 >>> print(a)
3 6
```

Afficher avec `print`

- La fonction la plus commune est `print` : elle permet d'afficher une variable, quelque soit son type.

```
>>> a = 2*3
>>> print(a)
6
```

- On peut mélanger texte et variables avec la syntaxe particulière suivante (ne pas oublier le `f` !) :

```
>>> prenom = "Alice"
>>> age = 21
>>> print(f"Je m'appelle {prenom} et j'ai {age} ans.")
Je m'appelle Alice et j'ai 21 ans.
```

Type d'une variable avec `type`

- Un doute sur le type d'une variable ? `type` est fait pour ça.

`type(valeur)`

```
1  >>> a = 2*3
2  >>> type(a)
3  <class 'int'>
```

Type d'une variable avec `type`

- Un doute sur le type d'une variable ? `type` est fait pour ça.

`type(valeur)`

```
1  >>> a = 2*3
2  >>> type(a)
3  <class 'int'>
```

- L'argument passé à la fonction peut être une simple valeur :

```
1  >>> type("Bonjour")
2  <class 'str'>
```

Arrondir avec `round`

- La fonction `round` permet d'**arrondir une valeur numérique**.

```
round(valeur, nombreDeDecimales)
```

Arrondir avec `round`

- La fonction `round` permet d'**arrondir une valeur numérique**.

```
round(valeur, nombreDeDecimales)
```

- Exemple : arrondi de $\frac{2}{3}$ à 4 décimales :

```
1 >>> round(2/3, 4)
2 0.6667
```

Arrondir avec `round`

- La fonction `round` permet d'**arrondir une valeur numérique**.

```
round(valeur, nombreDeDecimales)
```

- Exemple : arrondi de $\frac{2}{3}$ à 4 décimales :

```
1 >>> round(2/3, 4)
2 0.6667
```

- Le nombre de décimales est un **paramètre facultatif** : si on ne l'écrit pas, le nombre est arrondi à l'unité.

```
1 >>> round(1.618)
2 2
```

S'informer avec `help`

- La fonction `help` permet d'obtenir des informations sur l'utilisation d'une fonction.

```
help(nomDeLaFonction)
```

S'informer avec `help`

- La fonction `help` permet d'obtenir des informations sur l'utilisation d'une fonction.

```
help(nomDeLaFonction)
```

- Exemple : comment utiliser `round` ?

```
>>> help(round)
Help on built-in function round in module builtins:
round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted
    ↪  or None.  Otherwise
    the return value has the same type as the number.
    ↪  ndigits may be negative.
```

S'informer avec `help`

- La fonction `help` permet d'obtenir des informations sur l'utilisation d'une fonction.

```
help(nomDeLaFonction)
```

- Exemple : comment utiliser `round` ?

```
1  >>> help(round)
2  Help on built-in function round in module builtins:
3  round(number, ndigits=None)
4      Round a number to a given precision in decimal digits.
5
6      The return value is an integer if ndigits is omitted
       ↪  or None.  Otherwise
7      the return value has the same type as the number.
       ↪  ndigits may be negative.
```

- `ndigits=None` signifie que le paramètre `ndigits` possède une valeur par défaut, et est de fait falcutatif.

Autres fonctions

Quelques autres fonctions souvent utilisées :

- `len` : nombre de caractères dans une chaîne

```
len("toto") : 4
```

- `min(a,b)` ; `max(a,b)` : minimum et maximum de deux nombres

```
min(2,7) : 2
max(2,7) : 7
```

- `bin(a)` ; `hex(a)` : conversion en binaire / en héxadécimal

```
bin(42) : '0b101010'
hex(42) : '0x2a'
```

Le module `math`

Certaines fonctions ne sont pas directement disponibles, et nécessitent d'être **importées** depuis un **module**, comme la fonction mathématique *racine carrée*.

Le module `math`

Certaines fonctions ne sont pas directement disponibles, et nécessitent d'être **importées** depuis un **module**, comme la fonction mathématique *racine carrée*.

- On peut importer tout le module :

```
1 >>> import math
2 >>> math.sqrt(2)
3 1.4142135623730951
```

Le module `math`

Certaines fonctions ne sont pas directement disponibles, et nécessitent d'être **importées** depuis un **module**, comme la fonction mathématique *racine carrée*.

- On peut importer tout le module :

```
1 >>> import math
2 >>> math.sqrt(2)
3 1.4142135623730951
```

- Ou bien importer seulement la fonction :

```
1 >>> from math import sqrt
2 >>> sqrt(2)
3 1.4142135623730951
```

Le module `math`

Certaines fonctions ne sont pas directement disponibles, et nécessitent d'être **importées** depuis un **module**, comme la fonction mathématique *racine carrée*.

- On peut importer tout le module :

```
1 >>> import math
2 >>> math.sqrt(2)
3 1.4142135623730951
```

- Ou bien importer seulement la fonction :

```
1 >>> from math import sqrt
2 >>> sqrt(2)
3 1.4142135623730951
```

On peut importer toutes les fonctions d'un module en écrivant

```
from module import *
```

Le module `math`

- Dans le module `math`, on retrouve toutes les fonctions mathématiques connues :
  - `cos`, `sin`, `tan` : cosinus, sinus et tangente
  - `sqrt` : racine carrée
  - `gcd` : PGCD de deux nombres

Le module `math`

- Dans le module `math`, on retrouve toutes les fonctions mathématiques connues :
  - `cos`, `sin`, `tan` : cosinus, sinus et tangente
  - `sqrt` : racine carrée
  - `gcd` : PGCD de deux nombres
- Mais aussi des constantes mathématiques, comme `pi` (le nombre π) ou `e` (le nombre d'Euler)...

Le module `math`

- Dans le module `math`, on retrouve toutes les fonctions mathématiques connues :
  - `cos`, `sin`, `tan` : cosinus, sinus et tangente
  - `sqrt` : racine carrée
  - `gcd` : PGCD de deux nombres
- Mais aussi des constantes mathématiques, comme `pi` (le nombre π) ou `e` (le nombre d'Euler)...
- Liste complète : `https://docs.python.org/3/library/math.html`

Syntaxe de base

Pour définir une fonction, la syntaxe est toujours la même :

```
def nomFonction(parametre1, parametre2, ..., parametreN):
    instruction1
    instruction2
    ...
```

Syntaxe de base

Pour définir une fonction, la syntaxe est toujours la même :

```
def nomFonction(parametre1, parametre2, ..., parametreN):
    instruction1
    instruction2
    ...
```

- Ne pas oublier les deux points `:` à la fin de la première ligne

Syntaxe de base

Pour définir une fonction, la syntaxe est toujours la même :

```
def nomFonction(parametre1, parametre2, ..., parametreN):
    instruction1
    instruction2
    ...
```

- Ne pas oublier les deux points `:` à la fin de la première ligne
- Le **décalage des instructions** sur la droite n'est pas seulement esthétique, c'est `essentiel` pour Python

Quelques exemples

- Somme de deux nombres

```
1  def somme(a,b):
2      return a + b
```

La fonction prend 2 paramètres `a` et `b` et **renvoie** leur somme.

Quelques exemples

- Somme de deux nombres

```
1  def somme(a,b):
2      return a + b
```

La fonction prend 2 paramètres `a` et `b` et **renvoie** leur somme.

- Une fonction qui dit bonjour :

```
1  def direBonjour(prenom):
2      print(f"Bonjour {prenom} !")
```

La fonction prend un paramètre et **affiche** une phrase.

Afficher ou Renvoyer

- Dans la fonction `somme`, le mot clé `return` permet de **renvoyer** une valeur, que l'on peut éventuellement stocker dans une variable.

```
1 >>> S = somme(3,4)
2 >>> S
3 7
```

- La fonction `direBonjour` ne **renvoie rien** et se contente d'afficher quelque chose

```
1 >>> direBonjour("Alice")
2 Bonjour Alice
```

Remarques

- Une fonction peut ne prendre aucun paramètre

```
1  def fonction(): # Une fonction qui renvoie toujours 1
2      return 1
```

Remarques

- Une fonction peut ne prendre aucun paramètre

```
1  def fonction(): # Une fonction qui renvoie toujours 1
2      return 1
```

- Les instructions qui suivent `return` ne sont pas prises en compte

```
1  def fonction(x):
2      y = x**2
3      return y
4      y = y + 3 # Ne sera jamais exécuté !
```