**\*args et \*\*kwargs**

**\*args et \*\*kwargs**

**1. Quel est le but de l’astérisque ?**

Python a une syntaxe spéciale, \*(opérateur de décompression) et \*\*(opérateur de décompression), qui vous permet de passer un **nombre variable d'arguments** à une fonction.

Par convention, on les écrit \*argset \*\*kwargs, mais seuls les astérisques sont indispensables ; on pourrait également noter \*varset \*\*varspour obtenir le même résultat.

**2. \*args**

\*argsest approprié lorsque nous ne savons pas à l'avance combien d'arguments nous devons passer dans une fonction.

def check\_arguments(\*args):

print(f"These are the arguments {args}")

check\_arguments(1, 2, 'hey')

>> These are the arguments (1, 2, 'hey') // You get a tuple

\*args: est un tuple d'arguments.

def check\_tuple(a,b):

# Returns the sum of 'a' and 'b'

return sum((a,b))

print(check\_tuple(10,30))

>> 40

# Multiple \*args

def check\_multiple\_arguments(\*args):

return sum(args)

print(check\_multiple\_arguments(10,20,100,30))

>> 160

**3. \*\*kwargs**

\*\*kwargsest utile lorsque nous ne savons pas à l'avance combien d'arguments avec mots-clés nous devons passer dans une fonction.

\*\*kwargs: est un dictionnaire d'arguments (mots-clés).

def check\_keywordedarguments(\*\*kwargs):

print(kwargs)

check\_keywordedarguments(name="Sarah", age=24)

>> {'name': 'Sarah', 'age': 24} // You get a dictionary

def check\_keywordedarguments(\*\*kwargs):

for key, value in kwargs.items():

print(key,":",value)

check\_keywordedarguments(name="Sarah", age=24)

>>

name : Sarah

age : 24

def check\_keywordedarguments(\*\*kwargs):

return kwargs

print(check\_keywordedarguments(fruit='apple', ordered= 2))

>> {'fruit': 'apple', ordered: 2}

**4. \*args et \*\*kwargs ensemble**

Souvent \*args, et \*\*kwargssont utilisés ensemble dans une fonction où nous avons au moins un argument obligatoire.

Dans ces cas, l'ordre est important. \*argset \*\*kwargssont placés après tous les arguments requis.

def check\_arguments\_keywordedarguments (required\_arg, \*args, \*\*kwargs):

print(required\_arg)

if args:

print(args)

if kwargs:

print(kwargs)

check\_arguments\_keywordedarguments("required argument")

check\_arguments\_keywordedarguments("required argument", 1, 2, 'hey')

check\_arguments\_keywordedarguments("required argument", 1, 2, 'hey', name="Sarah", age=24)

**Il faut préserver l'ordre !**

def check\_arguments\_keywordedarguments(\*args,\*\*kwargs):

print('\*args', args)

print('\*\*kwargs', kwargs)

check\_arguments\_keywordedarguments(10,20,30,name='John',surname='Doe')

>>

\*args (10, 20, 30)

\*\*kwargs {'name': 'John', 'surname': 'Doe'}

check\_arguments\_keywordedarguments(10,20,30,name='John',surname='Doe', 2)

>> SyntaxError: positional argument follows keyword argument

Lorsque nous déclarons un paramètre étoilé tel que \*param, tous les arguments positionnels à partir de ce point jusqu'à la fin sont collectés sous forme d'un tuple appelé param.

Lorsque nous déclarons un paramètre à double étoile tel que \*\*param, tous les arguments de mots-clés à partir de ce point sont collectés dans un dictionnaire appelé param.

def check(a, \*numbers, \*\*person):

print('Greetings : ', a)

#iterate through all the items in tuple

for num in numbers:

print('num - ', num)

#iterate through all the items in dictionary

for key, value in person.items():

print(key + ': ' + value)

check("hello", 1,2,3,name="John",surname="Doe")

>> Greetings : hello

num - 1

num - 2

num - 3

name: John

surname: Doe

**5. Passer \*args et \*\*kwargs comme arguments**

def check(a, b, c):

print(a, b, c)

a = [1,2,3]

check(\*a)

>> 1 2 3

a = {'a':'Sarah', 'b': 24}

check(\*\*a)

>> TypeError: check() missing 1 required positional argument: 'c'

a = {'a':'Sarah', 'b':24, 'c': 180}

check(\*\*a)

>> Sarah 24 180