**\*Args and \*Kwargs**

**1. What is the point of the asterisk?**

Python has a special syntax, \* (unpacking operator) and \*\* (unpacking operator), that lets you pass a **variable number of arguments** to a function.

By convention, these are written as \*args and \*\*kwargs, but only the asterisks are essential; you could equally note \*vars and \*\*vars to achieve the same result.

**2. \*args**

\*args is proper when we don’t know in advance how many arguments we need to pass in a function.

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: is a tuple of args.

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**

\*\*kwargs is useful when we don’t know in advance how many keyworded arguments we need to pass in a function.

\*\*kwargs: is a dictionary of args (keywords).

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 and \*\*kwargs together**

Often \*args and \*\*kwargs are used together in a function where we have at least one required argument.

In these cases, order matters.\*args and \*\*kwargs are placed after any required arguments.

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)

**You have to preserve the order!**

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

When we declare a starred parameter such as \*param, then all the positional arguments from that point till the end are collected as a tuple called param.

When we declare a double-starred parameter such as \*\*param, all the keyword arguments from that point are collected as a dictionary called 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. Passing \*args and \*\*kwargs as 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