# Урок 17.03. Именованные аргументы. Значения по умолчанию. Аргументы \*args и \*\*kwargs

Давайте вернемся к функциям, а именно к её аргументам. До этого мы передавали в функцию исключительно позиционные аргументы: в строго заданном порядке и количестве. Однако, не редко случаются ситуации, когда в функцию нужно передать, либо много аргументов, либо часть из них.

Рассмотрим такую задачу «работа с файлом».

Напишем небольшое приложение для работы с файлами. Для этого приложения мы хотим реализовать три базовые возможности: запись в файл, удаление файла и выход из приложения. Мы задаем соответствующий вопрос пользователю, и совершаем выбранное действие. При этом количество попыток на ответ ограничено, для того чтобы пользователь мог просто вернуться в меню. Пока мы сделаем без самой работы с файлами (об этом мы скоро с вами узнаем). Приступим.

def ask\_user(question, complaint, retries):

while True:

answer = input(question).lower()

if answer == 'да':

return 1

elif answer == 'нет':

return 0

retries -= 1

if retries == 0:

print('Количество попыток истекло')

break

print(complaint)

print(f'Осталось попыток: {retries}')

Это универсальная функция, которая поможет нам задавать вопросы несколько раз. Можем протестировать нашу функцию, и добавим вопросы

def ask\_user(question, complaint, retries):

while True:

answer = input(question).lower()

if answer == 'да':

return 1

elif answer == 'нет':

return 0

retries -= 1

if retries == 0:

print('Количество попыток истекло')

break

print(complaint)

print(f'Осталось попыток: {retries}')

ask\_user('Вы действительно хотите выйти из программы? ',

'Ошибка ввода. Пожалуйста, введите да или нет.',

4)

ask\_user('Удалить файл? ',

'Ошибка ввода. Пожалуйста, введите да или нет.',

4)

ask\_user('Сохранить файл? ',

'Ошибка ввода. Пожалуйста, введите да или нет.',

4)

Запустим. Всё работает. Однако, одно из первых на что обращаешь внимание при написании такой программы это то, что приходится копировать текст ошибки, а программисты не любят повторяться. Можно сообщение об ошибки вставить внутрь самой функции и проверять переданное сообщение. И как вы уже догадались решение и тут существует. Для этого придуман такой способ, как задание значения аргументов по умолчанию. Это реализуется путем добавления необходимого значения при объявлении аргументов функции:

def ask\_user(question,

complaint='Ошибка ввода. Пожалуйста, введите да или нет.',

retries=4):

while True:

answer = input(question).lower()

if answer == 'да':

return 1

elif answer == 'нет':

return 0

retries -= 1

if retries == 0:

print('Количество попыток истекло')

break

print(complaint)

print(f'Осталось попыток: {retries}')

ask\_user('Вы действительно хотите выйти из программы? ')

ask\_user('Удалить файл? ')

ask\_user('Сохранить файл? ')

Вот так просто можно задать значение аргументов по умолчанию. Однако, как их теперь задавать? Тут тоже всё просто. Есть два способа: либо задавать как и раньше по порядку, либо задавать указывая имя параметра который мы задаем. Во втором случае мы теперь может задавать любые аргументы функции, которые перечислены в её заголовке.

def ask\_user(question,

complaint='Ошибка ввода. Пожалуйста, введите да или нет.',

retries=4):

while True:

answer = input(question).lower()

if answer == 'да':

return 1

elif answer == 'нет':

return 0

retries -= 1

if retries == 0:

print('Количество попыток истекло')

break

print(complaint)

print(f'Осталось попыток: {retries}')

ask\_user('Вы действительно хотите выйти из программы? ',

retries=1)

ask\_user('Удалить файл? ',

'Введите только да или нет.')

ask\_user('Сохранить файл? ')

Запустим и проверим. Теперь нам понятно как используются не позиционные аргументы.

Тут важно отметить следующие условия: именованные аргументы идут только после позиционных, одному аргументу нельзя присваивать два значения сразу.

# Аргументы \*args и \*\*kwargs.

При написании функция иногда мы можем знать, что будет передаваться, но не знаем в каком количестве. Например, функция print. И как это работает мы сегодня и познакомимся. И как всегда на примере.

Задача «Финансовый документ».

Нам нужно написать небольшую функцию для бухгалтерии. Эта функция принимает налог и цены на товары, а также должна принимать дополнительную информацию о документе. Фишка в том, что цен, как и дополнительной информации может быть сколько угодно. Внутри функции считается цена товара с учетом налога, и выводится их сумма, затем выводится дополнительная информация о документе, если таковая имелась. Начнем.

def print\_tax\_document(tax):

...

my\_tax = int(input('Введите налог: '))

print\_tax\_document(my\_tax, 1000, 950, 880, 920, 990)

И когда мы переходим к написанию функции, то мы не знаем как же нам принять неизвестное заранее количество аргументов. Для таких случаев придуман специальный инструмент.

def print\_tax\_document(tax, \*args):

price\_sum = 0

for i\_price in args:

price\_sum += i\_price \* tax / 100

print(f'Сумма цен с учетом налога: {price\_sum}')

my\_tax = int(input('Введите налог: '))

print\_tax\_document(my\_tax, 1000, 950, 880, 920, 990)

Вот так это работает. Вообще, args это кортеж и соответственно с ним нужно работать как с кортежем. А звездочка обозначает, что производится передача нескольких аргументов.

Теперь осталось разобраться с дополнительной информацией. Например, мы передаем год документа, тип и id документа. Но тогда все эти дополнения пойдут в args, и функция начнет умножать и делить эти данные, что вызовет ошибку. Но можно передать эти аргументы как именованные.

def print\_tax\_document(tax, \*args, \*\*kwargs):

price\_sum = 0

for i\_price in args:

price\_sum += i\_price \* tax / 100

print(f'Сумма цен с учетом налога: {price\_sum}')

my\_tax = int(input('Введите налог: '))

print\_tax\_document(my\_tax, 1000, 950, 880, 920, 990,

year=2024, title='Отчет', id\_doc=12345)

Получается, что для неименованных аргументов передача идет без указаний аргумента, а для передачи определенных данных используются именованные аргументы. И если args это кортеж, то kwargs, я думаю вы уже догадались, будет словарем.

def print\_tax\_document(tax, \*args, \*\*kwargs):

price\_sum = 0

for i\_price in args:

price\_sum += i\_price \* tax / 100

print(f'Сумма цен с учетом налога: {price\_sum}')

for i\_info, i\_value in kwargs.items():

print(f'{i\_info}: {i\_value}')

my\_tax = int(input('Введите налог: '))

print\_tax\_document(my\_tax, 1000, 950, 880, 920, 990,

year=2024, title='Отчет', id\_doc=12345)

Запустим. Всё получилось. Такой способ позволяет не задумываться о передаваемых аргументах, однако, это может привести к непониманию, что нужно передавать и что происходит уже внутри функции. Так что не стоит злоупотреблять таким способом, и использовать только в исключительных случаях, там где это действительно нужно.