Урок 2. Возврат значений из функций. Оператор return

В прошлом модуле мы обсуждали, что можно делать с передаваемыми аргументами внутри функции. Но, конечно же, обсудили мы далеко не все интересные особенности при работе с ними. И сейчас мы это исправим.

Задача 1

Давайте рассмотрим простой пример. Пользователь вводит цену товара и величину налога. Причём цена — это вещественное число, а налог — целое, в виде процентов. Например, товар стоит 25 рублей 0 копеек, а налог будет 6%. Нам нужно написать функцию, которая рассчитывает цену с учётом налога.

main.py

```
1   def calculate_tax(price, tax):
2       total = price + (price * tax / 100)
3       print(total)
4
5   myPrice = float(input('Введите цену: '))
6   myTax = int(input('Введите налог (в %): '))
7
8   calculate_tax(myPrice, myTax)
9
```

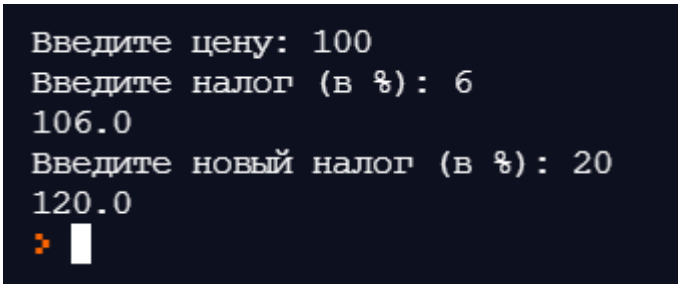
Реализация такой функции и программы в целом довольно простая: сначала мы запрашиваем у пользователя нужные значения, затем вызываем функцию, куда передаём эти значения в виде аргументов, а в самой функции всё это считается и выводится. Однако в реальном коде всё обычно бывает не так просто. Немного дополним нашу задачу: теперь нужно запросить у пользователя ещё один дополнительный налог и прибавить его к новому значению. Что ж, запросим и снова вызовем функцию.

```
10  newTax = int(input('Введите новый налог (в %): '))
11
12  calculate_tax()
13
```

Вот только что теперь писать в качестве первого аргумента? Давайте попробуем передать в качестве параметров переменную цены и нового налога.

```
12    calculate_tax(myPrice, newTax)
```

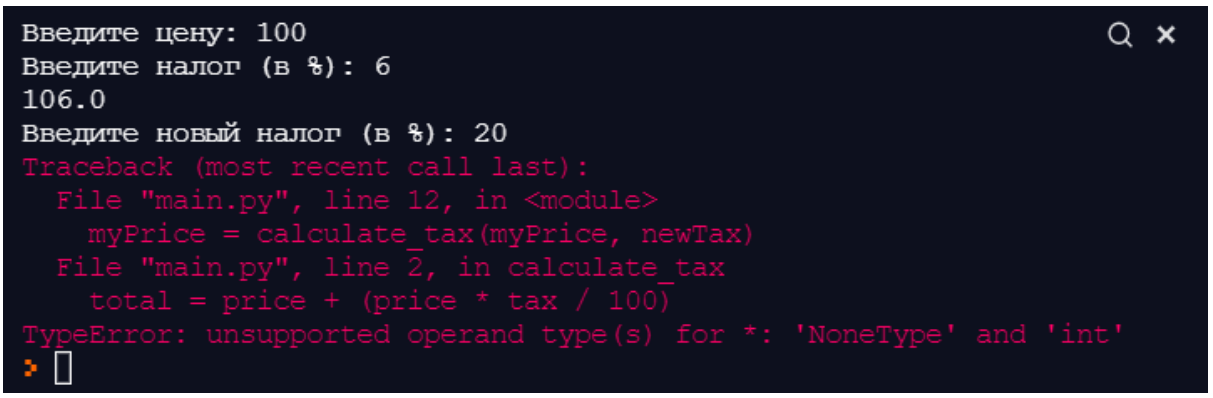
Тест:



Как мы видим, результат у нас получается некорректный, так как 20 процентов берётся не от нового значения, а от старого. Переменная myPrice отвечает за начальную цену товара, сама переменная при этом никак не меняется, а внутри функции мы просто выводим на экран новую цену с учётом налога. Значит, по идее, при вызове функции нам нужно как-то забрать значение переменной total и закинуть его в myPrice. Давайте попробуем просто переприсвоить переменную:

```
 8    myPrice = calculate_tax(myPrice, myTax)
 9
10    newTax = int(input('Введите новый налог (в %): '))
11
12    myPrice = calculate_tax(myPrice, newTax)
```

Тест:



Посмотрим, с чего вдруг возникает ошибка. На самом деле всё довольно просто: внутри функции командой print мы выводим значение переменной total на экран. А нам нужно забрать его отсюда и перекинуть в myPrice. Значит, вместо команды вывода нам понадобится новая команда — return:

main.py

```
1   def calculate_tax(price, tax):
2       total = price + (price * tax / 100)
3       return total
```

С английского return как раз переводится как «вернуть». Работает это довольно просто: мы указываем переменную (myPrice), в которую нужно вернуть подсчитанное значение, затем присваиваем ей вызов нашей функции с нужными аргументами (myPrice = calculate_tax(myPrice, myTax)). Внутри функции мы производим различные действия с этими аргументами и записываем результат в переменную (total). А затем забираем значение этой переменной в основную программу и передаём это значение в myPrice. В программировании это так и называется: функция возвращает значение.

Вообще, кстати, как-то некрасиво получается, что итоговая цена хранится там же, куда мы её вводим изначально. Давайте возьмём новую переменную и назовём её соответствующе и заодно проверим программу:

main.py

```
1   def calculate_tax(price, tax):
2       total = price + (price * tax / 100)
3       return total
4
5   myPrice = float(input('Введите цену: '))
6   myTax = int(input('Введите налог (в %): '))
7
8   totalPrice = calculate_tax(myPrice, myTax)
9
10  newTax = int(input('Введите новый налог (в %): '))
11
12  totalPrice = calculate_tax(myPrice, newTax)
13
14  print('Итоговая цена:', totalPrice)
15
```

Тест:

```
Введите цену: 100
Введите налог (в %): 6
Введите новый налог (в %): 20
Итоговая цена: 120.0
>
```

Кажется, ответ у нас получился неверный. Посмотрим ещё раз: мы считаем новую цену товара и записываем её в переменную. Затем запрашиваем новый налог. А потом мы вызываем функцию с новыми аргументами. Там всё считается, вот только куда возвращается новое значение? Правильно, в пустоту. Это как если бы мы просто написали команду input и никуда не записывали наше значение. И тогда эти данные

просто потерялись бы, хотя синтаксических ошибок тут нет. Значит, здесь нам нужно переприсвоить переменную:

```
12    totalPrice = calculate_tax(totalPrice, newTax)
```

Вот теперь всё будет считаться верно.

Особенность работы с функциями

Теперь давайте предположим, что аргумент tax ещё меняется внутри самой функции. Пусть, например, налог увеличивается на 10.

main.py

```
1   def calculate_tax(price, tax):
2       tax += 10
3       print(tax)
4       total = price + (price * tax / 100)
5       print(total)
6       return total
```

Тест:

```
Введите цену: 100
Введите налог (в %): 6
16
116.0
Введите новый налог (в %):
```

Всё довольно ожидаемо, мы передали налог 6 и получили 16, а затем к цене 100 прибавили налог 16%. Так как мы с помощью переменной myTax передали число 6, то у нас получилось 16. А теперь вопрос: если мы передали переменную, а потом её изменили, то изменилась ли начальная переменная, то есть переменная myTax? Проверим.

main.py

```
1   def calculate_tax(price, tax):
2       tax += 10
3       total = price + (price * tax / 100)
4       print(total)
5       return total
6
7   myPrice = float(input('Введите цену: '))
8   myTax = int(input('Введите налог (в %): '))
9
10  totalPrice = calculate_tax(myPrice, myTax)
11  print(myTax)
```

Тест:

```
Введите цену: 100
Введите налог (в %): 6
116.0
6
Введите новый налог (в %):
```

Как мы видим, значение переменной осталось прежним. Внутри функции мы можем делать с передаваемыми аргументами что угодно, однако в основной программе значения тех переменных, которые мы передаём, меняться не будут.

Задача 2

Для закрепления давайте решим ещё один небольшой пример. В одном дата-центре ресурсы распределены так, что сначала обрабатываются крупные задачи, а затем уже идут небольшие. Каждая из этих задач — это, по сути, просто огромный поток цифр. Наша задача как программиста этого центра — написать программу, которая поможет определять, какую из двух задач нужно решать в первую очередь. То есть у нас вводится два числа — это две задачи. Нам нужно составить программу, определяющую, в каком из данных двух чисел больше цифр:

main.py

```
1   firstTask = int(input('Введите первое число: '))
2   secondTask = int(input('Введите второе число: '))
3
```

Итак, первым делом мы запросили эти самые два числа. Теперь нам нужно сравнить количество цифр этих двух чисел. Для этого нам понадобится пока что несуществующая функция, которая будет принимать число, считать количество цифр в нём и возвращать это значение в основную программу. Давайте возьмём новые переменные, куда закинем полученные результаты:

main.py

```
firstTask = int(input('Введите первое число: '))
secondTask = int(input('Введите второе число: '))

firstNumeral = numeral_count(firstTask)
secondNumeral = numeral_count(secondTask)

```

Пока мы не пошли дальше, необходимо эту самую функцию написать. Принимать у нас она будет всего один аргумент — число:

main.py

```
def numeral_count(number):


firstTask = int(input('Введите первое число: '))
secondTask = int(input('Введите второе число: '))

firstNumeral = numeral_count(firstTask)
secondNumeral = numeral_count(secondTask)

```

Дальше у нас будет алгоритм, который считает количество цифр. Такое мы уже проходили — просто берём новую переменную count, а затем делим число на 10 и увеличиваем счётчик.

main.py

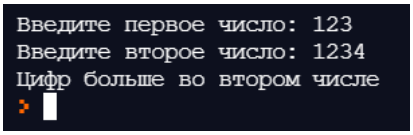
```
def numeral_count(number):
    count = 0
    while number > 0:
        number //= 10
        count += 1

    return count
```

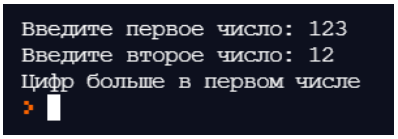
Так как нужная нам информация, которую нужно использовать в основной программе, хранится в переменной count, то именно её мы и возвращаем.

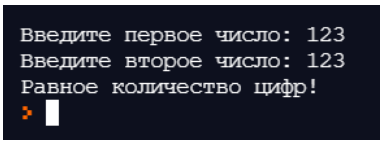
Теперь можно вернуться к основной программе. Здесь нам осталось написать проверки для новых переменных и вывести нужные сообщения:

```
firstTask = int(input('Введите первое число: '))
secondTask = int(input('Введите второе число: '))

firstNumeral = numeral_count(firstTask)
secondNumeral = numeral_count(secondTask)

if firstNumeral > secondNumeral:
    print('Цифр больше в первом числе')
elif firstNumeral < secondNumeral:
    print('Цифр больше во втором числе')
else:
    print('Равное количество цифр!')

```

Тесты:







Всё работает так, как задумано.

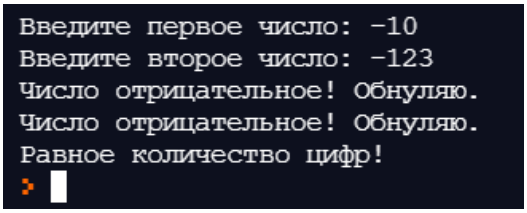
Использование нескольких операторов return

Допустим, наша функция numeral_count получает в качестве параметра отрицательное число и нам нужно, чтобы в таком случае выводилось соответствующее сообщение и число занулялось. Давайте так и запишем в начале функции:

main.py

```
def numeral_count(number):
    if number < 0:
        print('Число отрицательное! Обнуляю.')
        return 0

    count = 0
    while number > 0:
        number //= 10
        count += 1

    return count
```

Тест:



Как мы видим, в результате два раза подряд вывелось сообщение о занулении, и в итоге оба числа оказались равными. Таким образом, мы увидели, что в функции может быть несколько операторов return. Однако всегда выполняется только один из них, а именно тот, до которого мы дошли первым. У нас -10 меньше нуля, поэтому мы здесь вывели сообщение, а затем здесь вернули 0, и на этом выполнение функции сразу же закончилось. Помните о такой особенности при работе с функциями, ведь это можно использовать как во благо, так и во вред.

Итоги урока

На этом самое время подвести небольшие итоги урока. Итак, что мы узнали:

- При вызове функции и передаче ей аргументов можно взять нужное внутри неё значение и использовать его дальше в основной программе. Для этого используется оператор return, что переводится как «вернуть».
- Возвращаемое значение можно присваивать новым переменным или переприсваивать старым.
- В функции может содержаться несколько операторов return, но функция выполнит только первый попавшийся, и поток выполнения вернётся в основную программу.