Функции

Функции предназначены для того, чтобы вынести повторяющиеся блоки кода и дать им имя. По этому имени мы сможем обращаться к этой функции, когда она нам понадобится.

Нам уже знакомы несколько функций, например функция `sqrt()` из модуля `math`. Эта функция стандартной библиотеки Python (т.е. нам не нужно писать ее самим), отвечающая за вычисление квадратного корня от заданного числа. Чтобы ей воспользоваться, нам нужно сообщить от какого числа мы вычисляем корень (это называется *фактический параметр функции*) и получить результат ее работы (это называется *возвращаемое значение*).

```
a = math.sqrt(100)
```

Здесь значение 100 является фактическим параметром функции, а возвращаемое значение попадает в переменную `a`.

Фактические параметры могут быть не только обычными значениями, но и значениями других переменных:

```
a = math.sqrt(D)
```

Или даже в фактические параметры можно передавать целые выражения. Они будут предварительно вычислены и результат вычислений попадет в параметр функции:

```
a = math.sqrt(b*b - 4*a*c)
```

Параметров может быть несколько, как, например, в функции `randint()` из модуля `random`. Это тоже функция стандартной библиотеки Python, возвращающая случайное число из заданного диапазона:

```
x = random.randint(-100, 100)
```

У этой функции два параметра: -100 (начало диапазона) и 100 (конец диапазона). Чтобы передать несколько параметров в одну функцию, они разделяются запятой.

У функций может быть только столько параметров, сколько предусмотрел ее создатель, не больше и не меньше. Так, в функцию, принимающую 2 параметра, нельзя передать только 1 или 3.

Некоторые функции, такие как `print()`, позволяют указывать произвольное количество параметров, потому что их авторы специальным образом позаботились об этом.

Возвращаемые значения функций могут присваиваться в переменные, участвовать в дальнейшем вычислении выражений или просто игнорироваться. Присвоение результата в переменную использовалось в примерах выше. Использование результата функции в выражении нам нужно было, например, в задаче по решению квадратного уравнения:

```
x1 = (-b + math.sqrt(D)) / (2 * a)
```

Здесь после вычисления функции корня мы не используем промежуточной переменной для результата, а сразу складываем его со значением `-b`.

Если по какой-то причине нам не нужно возвращаемое значение функции (такое бывает редко, обычно мы ради возвращаемого значения и вызываем функцию!), мы можем его просто проигнорировать:

```
math.sqrt(100)
```

Несмотря на то, что это вполне допустимая конструкция языка, когда эта строчка выполнится, будет вычислено значение квадратного корня от 100 и проигнорировано. То есть мы зря выполнили эту операцию.

Так, если функция возвращает осмысленное значение, его игнорирование является ошибкой!

Тем не менее, бывают ситуации, когда функция не возвращает ничего (точнее в Python функции всегда что-то возвращают, и если мы считаем, что они не возвращают ничего, это значит, что они возвращают специальное значение `None`). В этом случае мы точно знаем, что результата в функции нет и нет смысла его куда-то присваивать. Так мы поступаем, например, с функцией `print()`:

```
print("Hello, World!")
```

В данном случае функция `print()` принимает один параметр и не возвращает ничего.

Создание собственных функций

Помимо использования стандартных функций, можно объявлять свои. Чтобы это сделать, необходимо начать объявление с ключевого слова `def`, а затем указать имя функции.

Имя функции должно отражать то, что она делает. Зачастую, общие имена вроде `f()` или `g()` никак не отражают их назначение и должны избегаться. Вместо них, если функция вычисляет, скажем, площадь прямоугольника, назвать функцию `rectangle_area()` является хорошей идеей. Если вы не знаете английский, лучше воспользоваться онлайн-словарем, чем писать названия транслитом. Несмотря на то, что это возможно, это считается дурным тоном в сообществе программистов.

После имени функции в круглых скобках идет обозначение *формальных параметров*. Формальные параметры это особые переменные, в которые попадают значения фактических параметров при вызове функции.

После закрывающей скобки формальных параметров следует символ двоеточия и со следующей строки с дополнительным отступом указываются действия, которые содержит функция. Эти действия называют *телом функции*.

```
def factorial(x):
    # тело функции
```

Здесь `x` это формальный параметр. Теперь, если функцию вызовут следующим образом:

```
a = factorial(5)
```

в переменную `x` попадет значение 5.

Тело функции может содержать все те же действия, что мы изучали до этого: линейную программу, ветвления циклы и даже другие функции.

Для того, чтобы обозначить результат функции, используют выражение `return`. Это ключевое слово, за которым следует значение, переменная или выражение, которое станет результатом функции:

```
def factorial(x):
    r = 1
    for i in range(1, x+1):
        r *= i
    return r # в r содержится факториал x
```

Как только функция выполняет `return`, она завершается. Это можно использовать для досрочного выхода из функции при проверке корректности:

```
def factorial(x):
    if x < 0:
      # если выполнится эта строка, функция дальше не пойдет
      return 0
    r = 1
    for i in range(1, x+1):
        r *= i
    return r
```

Если вы хотите возвращать значение из функции, лучше позаботиться о явном указании выражения `return` во всех возможных исходах функции. Т.е. если функция использует систему ветвлений, считается плохим тоном оставлять возможность не выполнить `return` вообще при каком-то наборе параметров:

```
def square_equation(a, b, c):
    D = b*b - 4*a*c
    if D >= 0:
        x1 = (-b + math.sqrt(D)) / (2 * a)
        x2 = (-b + math.sqrt(D)) / (2 * a)
        return x1, x2 # вернем оба значения
    # а что вернет функция, если D < 0?
```

В данном примере мы забыли указать возвращаемое значение, если получившийся дискриминант оказался меньше нуля. Python в этом случае найдет выход и вернет `None`, но лучше указать возвращаемое значение явно.

Если же вы не хотите возвращать значение из функции вообще, можно не указывать return совсем:

```
def print_double(x):
    print(x * 2)
```

При этом, если вы хотите досрочно выйти из такой функций, можно использовать return без параметров:

```
def print_double(x):
    if x == 0:
        return # мы не работаем с нулем
    print(x * 2)
```

Стоит различать функции, возвращающие значение, и функции, не возвращающие значение. В функциях первого типа мы значением **возвращаем и не печатаем**. Печать на экран в возвращающих функциях должна рассматриваться только для целей отладки и

должна быть убрана в финальной версии функции. В функциях второго типа мы значение **печатаем и не возвращаем**. Обычно такие функции используются для целей оформления интерфейса программы.

Также стоит помнить, что получение данных от внешнего мира функция осуществляет через параметры. **Не является хорошим тоном использование переменных из основной программы**. Также, если это не является целью написания функции, **ввод с клавиатуры также не осуществляется внутри функции**!

Переменные внутри функции являются собственностью функции и к ним нельзя получить доступ извне. Переменные с одинаковыми именами в разных функциях или функциях и главное программе являются разными переменными, поэтому не стоит ограничивать себя в выборе имени переменной внутри функции, если переменная с таким именем уже существует в другой функции. Опять же хорошим тоном является использование одинаковых имен переменных для одних и тех же сущностей. Не очень хорошо, когда, например, переменная `digit` в одной функции обозначает цифру из некоего числа, а в другой – ее номер.

Структура программы
Несмотря на то, что с точки зрения Python можно организовать программу как угодно, если не придерживаться какой-то структуры, в скором времени вы сами не сможете ничего найти в коде.

Хорошим тоном является следующая структура программы:

импорты

функции

основная программа

Если расположение импортов в начале файла и так является естественным, из-за желания воспользоваться в функциях переменными из главной программы, некоторые начинающие объявляют их выше определения функций. Стоит помнить, что **указание части программы (даже ввода с клавиатуры) выше функций, является плохой практикой**!

Плохо:

```
x = int(input("Введите число: "))

def double():
    return x * 2

print(double())
```

Хорошо:

```
def double(x):
    return x * 2

x = int(input("Введите число: "))
print(double(x))
```