**Функции**

Функции - части программы, которые можно повторно вызывать с разными параметрами, чтобы не писать много раз одно и то же. Функции в программировании немного отличаются от математических функций. В математике функции могут только получить параметры и дать ответ, в программировании же функции умеют делать что-нибудь полезное, например, ничего не возвращать, но что-то печатать.

Функции чрезвычайно полезны, если одни и те же действия нужно выполнять несколько раз. Но некоторые логические блоки работы с программой иногда тоже удобно оформлять в виде функции. Это связано с тем, что человек может одновременно держать в голове ограниченное количество вещей. Когда программа разрастается, отследить все уже очень сложно. В пределах одной небольшой функции запутаться гораздо сложнее - известно, что она получает на вход, что должна выдать, а об остальной программе в это время можно не думать.

В программировании также считается хорошим стилем писать функции, умещающиеся на один экран. Тогда можно одновременно окинуть взглядом всю функцию и не нужно крутить текст туда-сюда. Поэтому, если получилось что-то очень длинное, то нужно нарезать его на кусочки так, чтобы каждый из них был логичным (делал какое-то определенное действие, которое можно назвать) и не превышал при этом 10-15 строк.

Мы уже использовали готовые функции, такие как print, len и некоторые другие. Эти функции описаны в стандартной библиотеке или других подключаемых библиотеках. Сегодня мы научимся создавать свои функции.

**Использование функций**

Рассмотрим, как создать свою функцию, на примере вычисления факториала. Текст программы без функции выглядит так:

|  |  |
| --- | --- |
| 1 | n = int(input()) |
| 2 | fact = 1 |
| 3 | i = 2 |
| 4 | while i <= n: |
| 5 | fact \*= i |
| 6 | i += 1 |
| 7 | print(fact) |

Вычисление факториала можно вынести в функцию, тогда эта же программа будет выглядеть так:

|  |  |
| --- | --- |
| 1 | def factorial(num): |
| 2 | fact = 1 |
| 3 | i = 2 |
| 4 | while i <= num: |
| 5 | fact \*= i |
| 6 | i += 1 |
| 7 | return fact |
| 8 |  |
| 9 | n = int(input()) |
| 10 | print(factorial(n)) |

Описание функции должно идти в начале программы. На самом деле, оно может быть в любом месте, до первого вызова функции factorial.

Определение функции должно начинаться со слова def (сокращение от define, определить). Дальше идет имя функции, после которого в скобках через запятую перечисляются параметры (у нашей функции всего один параметр). После закрытия скобки должно стоять двоеточие.

Команды, выполняемые в функции должны записываться с отступом, как в блоках команд if или while.

В нашей функции num - это параметр, на его место подставляется то значение, с которым функция была вызвана. Действия внутри функции точно такие же, как в обычной программе, кроме дополнительной команды return. Команда return возвращает значение функции (оно должно быть записано через пробел после слова return) и прекращает её работу. Возвращенное значение подставляется на то место, где осуществлялся вызов функции.

Команда return может встречаться в любом месте функции. После того как она выполнится, работа функции будет прекращена. Здесь есть некоторая аналогия с командой break, применяемой для выхода из цикла.

**Вызовы функций из функции**

Функцию подсчета факториала можно использовать для подсчета биномиальных коэффициентов (числа сочетаний). Формула для подсчета числа сочетаний выглядит так: n! / (k! \* (n - k)!).

Если бы мы не пользовались функциями, то нам потребовалось бы три раза записать почти одно и то же. С помощью функций вычисление выглядит намного проще:

|  |  |
| --- | --- |
| 1 | print(factorial(n) // (factorial(k) \* factorial(n - k))) |

Подсчет биномиальных коэффициентов можно также оформить в виде функции с двумя параметрами:

|  |  |
| --- | --- |
| 1 | def binomial(n, k): |
| 2 | return factorial(n) // (factorial(k) \* factorial(n - k)) |

Последнее изменение: Thursday, 6 September 2018, 05:21

**Возврат значений**

Как было сказано выше, выполнение функции прерывается по команде return. Для примера рассмотрим функцию поиска максимума из двух чисел, которые передаются ей в качестве параметров:

|  |  |
| --- | --- |
| 1 | def max2(a, b): |
| 2 | if a > b: |
| 3 | return a |
| 4 | else: |
| 5 | return b |

Её можно было бы записать и по-другому:

|  |  |
| --- | --- |
| 1 | def max2(a, b): |
| 2 | if a > b: |
| 3 | return a |
| 4 | return b |

Если условие в if'е было истинным, то выполнится команда return a и выполнение функции будет прекращено - до команды return b выполнение просто не дойдет.

С помощью функции max2 можно реализовать функцию max3, возвращающую максимум из трех чисел:

|  |  |
| --- | --- |
| 1 | def max3(a, b, c): |
| 2 | return max2(max2(a, b), c) |

Эта функция дважды вызывает max2: сначала для выбора максимума среди чисел a и b, а затем для выбора максимума между найденным значением и оставшимся числом c.

Здесь нужно обратить внимание, что в качестве аргумента функции может передаваться не только переменная или константное значение, но и результат вычисления любого арифметического выражения. Например, результат, возвращенной другой функцией.

Наши функции max2 и max3 будут работать не только для чисел, но и для любых сравнимых объектов, например, для строк.

**Возврат нескольких значений функцией**

Рассмотрим случай, когда функция должна вернуть несколько значений на примере функции, упорядочивающей два числа. Чтобы вернуть несколько значений, достаточно записать их в return через запятую. Аналогично, через запятую должны быть перечислены переменные, в которые будут попадать вычисленные значения.

|  |  |
| --- | --- |
| 1 | def sort2(a, b): |
| 2 | if a < b: |
| 3 | return a, b |
| 4 | else: |
| 5 | return b, a |
| 6 | a = int(input()) |
| 7 | b = int(input()) |
| 8 | minimum, maximum = sort2(a, b) |
| 9 | print(minimum, maximum) |

На самом деле, при перечислении значений через запятую, формируются объекты типа "кортеж", их подробное изучение будет на следующей лекции. Имеющихся знаний достаточно для использования функций, возвращающих несколько значений.

**Возврат логических значений**

Иногда удобно оформлять даже простые вещи в виде функций, чтобы повысить читаемость программы. Например, если нужно проверить число на четность, то гораздо понятнее будет каждый раз вызывать функцию isEven(n), а не писать каждый раз n % 2 == 0.

Такая функция может выглядеть так:

|  |  |
| --- | --- |
| 1 | def isEven(n): |
| 2 | return n % 2 == 0 |

Результатом работы этой функции будет истина или ложь. Теперь функцию очень удобно применять в if'ах:

|  |  |
| --- | --- |
| 1 | if isEven(n): |
| 2 | print("EVEN") |
| 3 | else: |
| 4 | print("ODD") |

Если есть сложное логическое выражение, то лучше оформить его в виде функции с говорящим названием - так программу будет легче читать, а вероятность ошибок в ней резко снизится.

**Использование рекурсии**

Рассмотрим задачу, где действия выполняются как на рекурсивном подъёме, так и на рекурсивном спуске. Пусть дана последовательность, которая оканчивается нулём. Необходимо вывести все чётные члены последовательности в прямом порядке, а затем все нечётные члены последовательности в обратном порядке. Её решение будет выглядеть так:

|  |  |
| --- | --- |
| 1 | def rec(): |
| 2 | n = int(input()) |
| 3 | if n != 0: |
| 4 | if n % 2 == 0: |
| 5 | print(n) |
| 6 | rec() |
| 7 | if n % 2 != 0: |
| 8 | print(n) |
| 9 | rec() |

Каждый экземпляр функции считывает в свою локальную переменную n число, если оно чётное, то сразу выводит его и запускает следующий экземпляр. После того, как все последующие экземпляры функции окончили работу (и вывели последующие нечётные числа в обратном порядке), функция выводит число, если оно было нечетным.

Рассмотрим еще один пример: подсчитать факториал числа, не пользуясь циклами:

|  |  |
| --- | --- |
| 1 | def factorial(n): |
| 2 | if n == 0: |
| 3 | return 1 |
| 4 | return n \* factorial(n - 1) |
| 5 | n = int(input()) |
| 6 | print(factorial(n)) |

Тем, кто знаком с методом математической индукции, будет довольно просто осознать рекурсию. Как и в математической индукции, в рекурсии должна быть база (момент, когда функция не вызывает другую рекурсивную функцию) и переход (правило, по которому считается результат по известному результату для меньшего параметра). Наша функция подсчета факториала делает только свою работу, но пользуется результатами чужого труда. Например, если функция получила на вход параметр 4, то должна вернуть 4 умноженное на 3! (который будет посчитан другими функциями). В случае факториала аналогом "базы индукции" может выступать 0! - по определению он равен единице.

Эти примеры иллюстрируют общую схему написания рекурсивных функций: сначала проверяется условие, когда функция должна закончиться, а дальше делается все остальное. При этом параметр должен сходиться к значению базы. Обычно это означает, что при каждом следующем вызове рекурсии параметр должен уменьшаться.

Последнее изменение: Thursday, 6 September 2018, 06:09

**Рекурсия**

Мы уже пробовали запускать функцию из другой функции и все работало. Ничто не мешает запустить из функции саму себя - тогда просто создастся новый экземпляр функции, которые будет выполнять те же команды. Такой процесс называется рекурсией.

Представим себе, что у нас есть миллиард человек (это будущие экземпляры функции), сидящих в ряд, и у каждого из них есть листочек для записи (это его локальная память). Нам нужно произносить числа и написать инструкцию для людей так, чтобы они в итоге сказали все числа из последовательности в обратном порядке. Пусть каждый из них будет записывать на своем листочке только одно число. Тогда инструкция для человека будет выглядеть так:

1) Запиши названное число

2) Если число не последнее - потереби следующего за тобой человека, пришла его очередь работать

3) Когда следующий за тобой человек сказал, что он закончил – назови записанное число

4) Скажи тому, кто тебя теребил (предыдущий человек), что ты закончил

Формализуем задачу. Пусть задается последовательность натуральных чисел, заканчивающаяся нулем. Необходимо развернуть ее с помощью рекурсии.

|  |  |
| --- | --- |
| 1 | def rec(): |
| 2 | n = int(input()) |
| 3 | if n != 0: |
| 4 | rec() |
| 5 | print(n) |
| 6 | rec() |

Эта функция осуществляет действие (вывод числа) на рекурсивном спуске, т.е. после рекурсивного вызова.

Последнее изменение: Thursday, 6 September 2018, 06:01