Функции на PYTHON

Урок 7

Актау, 2018

Функции

python

Весь шаблон объявления функции выглядит следующим образом (жирным

шрифтом выделены ключевые элементы):

def имя_ функции (аргументы) :

команды

- Для функции не указывается тип результата (хотя функция, разумеется , можетвозвращать результат).

Функции

python

Листинг 3.1. Объявление функций

```
# Функция без аргументов
def your_name():
    # Отображается сообщение
    print("Добрый день!")
    # Запоминается введенный пользователем текст
    name=input("Как Вас зовут? ")
    # Результат функции
    return name
# Функция с одним аргументом
def say_hello(txt):
    # Отображается сообщение
    print("Здравствуйте,",txt+"!")
# Вызываем функцию и результат записываем в переменную
my_name=your_name()
# Вызываем функцию с аргументом
say_hello(my_name)
```

В результате выполнения этого программного кода получаем следующий результат (жирным шрифтом выделено введенное пользователем значение):

Функции

python

Результат выполнения программы (из листинга 3.1)

```
Добрый день!
Как Вас зовут? Алексей Васильев
Здравствуйте, Алексей Васильев!
```

Код достаточно простой, но мы его все же прокомментируем. В программе объявляется две функции. Функция `your_name()` не имеет аргументов. При выполнении этой функции сначала командой `print("Добрый день!")` отображается приветствие. Затем пользователю предлагается ввести свое имя. Введенное пользователем текстовое значение запоминается в переменной `name`. Вся команда выглядит как `name=input("Как Вас зовут? ")`.

После этого с помощью инструкции `return name` значение переменной `name` возвращается в качестве результата функции `your_name()`. Таким образом, у этой функции нет аргументов, но зато она возвращает результат. И ее результат - это то, что вводит пользователь (предполагается, что имя пользователя).

Еще одна функция `say_hello()` нужна для отображения приветствия. У функции один аргумент (обозначен как `txt`). Текст сообщения, которое отображается при вызове функции, формируется с учетом значения аргумента `txt`, переданного функции. Результат функция не возвращает. В теле функции всего одна команда `print("Здравствуйте,",txt+"!")`, которой в консольное окно выводится сообщение.

Функции

python

При вычислении выражения `txt+"!"` мы неявно предполагаем, что аргумент `txt` ссылается на текстовое значение. Если такой уверенности нет, то лучше перестраховаться и воспользоваться выражением `str(txt)+"!"`, в котором выполняется явное приведение аргумента `txt` к текстовому типу.

При вычислениях мы используем следующее выражение (основанное на ряде Тейлора для соответствующей функции): $exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} + \cdots$.
Восклицательный знак означает вычисление факториала: по определению $n!$ означает произведение натуральных чисел от 1 до n, то есть $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$

Функции

python

Листинг 3.2. Математические функции

```
# Функция для вычисления экспоненты
def my_exp(x,n):
    s=0 # Начальное значение суммы ряда
    q=1 # Начальное значение добавки
    # Оператор цикла для вычисления ряда
    for k in range(n+1):
        s+=q        # Добавка к сумме
        q*=x/(k+1) # Новая добавка
    # Результат функции
    return s
# Проверяем результат вызова функции
x=1 # Аргумент для экспоненты
# Оператор цикла для многократного
# вызова функции вычисления экспоненты
for n in range(11):
    # Отображаем результат вызова
    # функции экспоненты
    print("n =",n,"->",my_exp(x,n))
```

Функции

python

Результат выполнения программы (из листинга 3.2)

```
n = 1 -> 1
n = 2 -> 2.0
n = 3 -> 2.5
n = 4 -> 2.666666666666665
n = 5 -> 2.708333333333333
n = 6 -> 2.716666666666663
n = 7 -> 2.718055555555554
n = 8 -> 2.718253968253968 4
n = 9 -> 2.71827876984127
n = 10 -> 2.718281525573192 2
```

Как и следовало ожидать, с увеличением количества слагаемых точность вычислений возрастает (напомним, "точное" значение равняется $e \equiv \exp(1) \approx 2.718281828459045$).

Функции

python

Листинг 3.3. Значения аргументов по умолчанию

```
# 1-я функция с одним аргументом.
# У аргумента есть значение по умолчанию
def print_text(txt="Значение аргумента по умолчанию."):
    print(txt)
# 2-я функция с двумя аргументами.
# У второго аргумента есть значение по умолчанию
def show_args(a,b="Второй аргумент не указан."):
    print(a,b)
# 3-я функция с двумя аргументами.
# У аргументов есть значения по умолчанию
def my_func(x="1-й аргумент x.",y="2-й аргумент y.",):
    print(x,y)
# Проверяем работу 1-й функции.
# Функции передан один аргумент
print_text("Аргумент указан явно.")
# Функции аргументы не передаются
print_text()
# Проверяем работу 2-й функции.
# Функции переданы два аргумента
show_args("Первый аргумент.","Второй аргумент.")
# Функции передан один аргумент
show_args("Первый аргумент.")
# Проверяем работу 3-й функции.
# Функции аргументы не передаются
my_func()
# Функции передан один аргумент
my_func("Один из аргументов.")
# Функции передан один аргумент.
# Переданный аргумент идентифицирован явно
my_func(y="Один из аргументов.")
```

Функции

python

В результате выполнения программного кода получаем следующий результат:

Результат выполнения программы (из листинга 3.3)

```
Аргумент указан явно.
Значение аргумента по умолчанию.
Первый аргумент. Второй аргумент.
Первый аргумент. Второй аргумент не указан.
1-й аргумент x. 2-й аргумент y.
Один из аргументов. 2-й аргумент y.
1-й аргумент x. Один из аргументов.
```

Мы объявляем три функции. Все три функции действуют по одной схеме: значения аргументов выводятся на экран. Мы подразумеваем, что все три функции оперируют с текстовыми аргументами. Некоторые из этих аргументов имеют значения по умолчанию. Так, у функции `print_text()` один аргумент, и у этого аргумента есть значение по умолчанию. Мы вызываем эту функцию без аргумента (в этом случае используется значение аргумента по умолчанию), а также вызываем эту функцию с одним аргументом (в этом случае используется переданное аргументом значение).

Функции

При присваивании
переменной в качестве значения имени функции эта переменная
становится ссылкой на функцию (наравне с именем функции). Небольшой
пример, иллюстрирующий эту возможность, представлен в листинге 3.4.

Листинг 3.4. Ссылка на функцию

```
# Исходная функция
def my_func(txt):
    print("Функция my_func:",txt)
# Переменной присваивается имя функции
new_func=my_func
# Вызываем функцию через переменную
new_func("вызов через new_func.")
```

Результат выполнения этого программного кода тако

Результат выполнения программы (из листинга 3.4)

```
Функция my_func: вызов через new_func.
```