Функции

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
| **def findV (d,sh,vus):**  **y=d\*sh\*vus**  **return y** | Обявление функции и переменные какие она принимает  Тело функции  Возвращаемое значение из функции |
| **V=findV(a,b,c)** | Вызов функции. Сама функция должна быть в коде выше чем ее вызов |
| **def nalogi(\*args):**  **c=sum(args)**  **c=c\*1.2**  **return c** | Принимает любое количество переменных. \*\*kvargs Принимает пару переменных, значение/ключь |
| **m = map(nalogi,[200,300,400,500,600,700])** | Продвинутый вызов функции. Выполняет по очереди с каждым элеменом. |
| **V= lambda x,y,z: x\*y\*z**  V(a,b,c)  list1=[1,2,3,4,5,6,7,8,9,10]  x= list(map((lambda y: y+50), list1))  print(x) | Однострочная функция |
| **a=True**  **b=True**  **c=False**  **print(all([a,b,c]))**  **print(any([a,b,c]))** | Встроенная функция проверяет все ли значения Тру.  Проверяет на тру одно любое из значений списка |
| **a = ("hi:", "lo:", "op:", "cl:")**  **b = [2.00, 1.50, 1.70, 1.80]**  **skleit = list(zip(a,b))** | Склейка двух списков в пару список либо словарь. Выполняется в момент обращения |
|  |  |
|  |  |
| **def maker(N):**  **def action(X):**  **return X \*\* N**  **return action**  **f = maker(2)** # Запишет 2 в N  **f** возвращает ссылку на вложенную функцию  <function action at 0x014720B0>  >>> **f(3)**  9  >>> **f(4)**  16 | *# Создать и вернуть функцию*  *# Функция action запоминает значение N в объемлющей области видимости*  *# Запишет 3 в X, в N по-прежнему хранится число 2*  *# 4 \*\* 2*  Данный пример вложенной функции удобно использовать если разово переменной мы присвоили первый множитель из вложенной функции а далее через атрибут присваиваем второй множитель |
| **def tester(start):**  **state = start**  **def nested(label):**  **nonlocal state**  **print(label, state)**  **state += 1**  **return nested**  **F = tester(0)**  >>> **F(‘ham’)**  ham 1  >>> **F(‘eggs’)**  eggs 2 | *# В каждом вызове сохраняется свое значение state*  *# Объект state находится в объемлющей области видимости*  *# Изменит значение переменной, объявленной как nonlocal*  *# Будет увеличивать значение state при каждом вызове* |
|  |  |
| def add\_fun(\*kvarg):      x=kvarg      y=sum(kvarg)      print(x, y)  a=10; b=20; c=30  List=[40,50,60]  add\_fun(\*List)  add\_fun(a,b,c)  add\_fun(10,20,30)  (40, 50, 60) 150  (10, 20, 30) 60  (10, 20, 30) 60 | Передает любое количество аргументов. В функции возвращает кортеж. Список указываем со \*.  **def f(\*args): print(args)**  **f(1,2,3,4)**  (1, 2, 3, 4) |
| def add\_fun(\*\*kvargs):      x=kvargs      #y=sum(kvarg)      print(x)  dict\_={"1":"Pobedy", "2":"1"}  add\_fun(\*\*dict\_) | Применимо для словарей.  **def f(\*\*args): print(args)**  **f(a=1, b=2)**  {‘a’: 1, ‘b’: 2} |
| **def f(a, \*pargs, \*\*kargs):**  **print(a, pargs, kargs)**  **f(1, 2, 3, x=1, y=2)**  1 (2, 3) {‘y’: 2, ‘x’: 1} | Смешивание типов сопоставления аргументов в функции.  A – позиционный,  \*pargs – последовательность,  \*\*kargs- словарь |
| **f(c=3, b=2, a=1)**  1 2 3  >>> **f(1, c=3, b=2)**  1 2 3 | Сопоставление по именам при вызове.  1 - Позиционный, c b- именованные |
| **def f(a, b=2, c=3): print a, b, c** | а - Обязательный аргумент, b и c – не обязательные так как уазаны значения по умолчанию. |
| **Не путайте синтаксические конструкции name=value в заголовке функции и в вызове функции – в вызове она означает использование режима сопоставления значения с именованным аргументом, а в заголовке определяет значение по умолчанию для необязательного аргумента.** | |
|  |  |
| def sum\_list(L):      print(L)      return 0 if not L else L[0]+ sum\_list(L[1:])  list=[1,2,3,4,5,6,7,8,9,10]  print(sum\_list(list)) | Рекурсия. Вызов функции на саму себя.  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  [2, 3, 4, 5, 6, 7, 8, 9, 10]  [3, 4, 5, 6, 7, 8, 9, 10]  [4, 5, 6, 7, 8, 9, 10]  [5, 6, 7, 8, 9, 10]  [6, 7, 8, 9, 10]  [7, 8, 9, 10]  [8, 9, 10]  [9, 10]  [10]  []  55 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

*Таблица 16.1. Инструкции и выражения, имеющие отношение к функциям*

|  |  |
| --- | --- |
| **Инструкция** | **Примеры** |
| **Вызов** | myfunc(‘spam’, ‘eggs’, meat=ham) |
| **def, return** | def adder(a, b=1, \*c):  return a+b+c[0] |
| **global** | def changer():  global x; x = ‘new’ |
| **nonlocal** | def changer():  nonlocal x; x = ‘new’ |
| **yield** | def squares(x):  for i in range(x): yield i \*\* 2 |
| **lambda** | funcs = [lambda x: x\*\*2, lambda x: x\*3] |

**Таблица 18.1. Формы сопоставления аргументов функций**

|  |  |  |
| --- | --- | --- |
| **Синтаксис** | **Мест. –жение** | **Интерпретация** |
| **func*{значение)*** | Вызывающий код | Нормальный аргумент: сопоставляется по позиции |
| **func*{имя=значение)*** | Вызывающий код | Ключевой аргумент: сопоставляется по имени |
| **func(\**итерируемый объект)*** | Вызывающий код | Передает все объекты в *итерируемом объекте* как отдельные позиционные аргументы |
| **func*{★★словарь)*** | Вызывающий код | Передает все пары ключ/значение в *словаре* как отдельные ключевые аргументы |
| **def func(имя)** | Функция | Нормальный аргумент: сопоставляется с любым переданным значением по позиции или по имени |
| **def func*{имя=\* значение)*** | Функция | Стандартное значение аргумента, если значение в вызове не передавалось |
| **def func(\**имя)*** | Функция | Сопоставляет и собирает оставшиеся позиционные аргументы в кортеж |
| **def func(\*\*m)** | Функция | Сопоставляет и собирает оставшиеся ключевые аргументы в словарь |
| **def func (\**остальные, имя)*** | Функция | Аргументы, которые должны передаваться в вызовах только по ключевому слову (Python З.Х) |
| **def func(\*,*имя=значение)*** | Функция | Аргументы, которые должны передаваться в вызовах только по ключевому слову (Python З.Х) |

**Если вы решили использовать и комбинировать специальные режимы сопоставления аргументов, тогда Python потребует соблюдать описанные далее правила упорядочения для необязательных компонентов.**

• В вызове функции аргументы должны указываться в следующем порядке: любые позиционные аргументы (значение), за ними комбинация любых ключевых аргументов (имя=значение) и формы \*итерируемый\_объект, а затем форма \*\*словарь.

**funk\_name((Позиционные), (По ключам именованные),**

**(Последовательности), (Словари)**

• В заголовке функции аргументы должны указываться в следующем порядке: любые нормальные аргументы (имя), за ними любые стандартные аргументы (имя=значение), далее форма \*имя (или \* в Python З.Х), затем аргументы с передачей только по ключевым словам имя или имя=значение (в Python З.Х) и, наконец, форма \*\*имя.

**def funk\_name((Позиционные), (Со стандартным значением),**

**(Последовательности), (По ключам именованные),**

**(Словари)**

**\*- вместо последовательностей говорит о том что принимаемые последовательности отсуцтвуют но еще ожидается аргументы по ключам**

**Действия, которые выполняет интерпретатор при сопоставлении аргументов перед присваивани­ем, грубо можно описать так:**

**1. Присваивание неключевых аргументов по позиции.**

**2. Присваивание ключевых аргументов по совпадающим именам.**

**3. Присваивание добавочных неключевых аргументов кортежу \*имя.**

**4. Присваивание добавочных ключевых аргументов словарю \*\*имя.**

**5. Присваивание стандартных значений неприсвоенным аргументам в заголовке.**