**1.Введение в Питон. Режимы работы Питон.**

Python представляет популярный высокоуровневый язык программирования, который предназначен для создания приложений различных типов. Это и веб-приложения, и игры, и настольные программы, и работа с базами данных. Довольно большое распространение питон получил в области машинного обучения и исследований искусственного интеллекта.

Далее приведены некоторые особенности **Python**:

* Простая, разборчивая грамматика.
* И процедурно-ориентированный (procedural-oriented), и объективно-ориентированный (object-oriented)
* Поддерживает module и пакет (package)
* Обработка ошибок с исключением (Exception)
* Наличие динамических типов данных высокого уровня
* Наличие стандартных библиотек и внешних модулей, которые отвечают всем требованиям программирования.
* Возможность взаимодействия с другими модулями, написанными на **C/C++**(или **Java**для **Jython**, или **.Net**для **IronPython**).
* Возможность встроить приложение в качестве интерфейса сценариев (scripting interface).

2 режима работы в питоне:

1)Режим интерпретатора команд

(Интерпретация программ. Для работы со скриптами необходим интерпретатор, который запускает и выполняет скрипт.

Выполнение программы на Python выглядит следующим образом. Сначала мы пишим в текстовом редакторе скрипт с набором выражений на данном языке программирования. Передаем этот скрипт на выполнение интерпретатору. Интерпретатор транслирует код в промежуточный байткод, а затем виртуальная машина переводит полученный байткод в набор инструкций, которые выполняются операционной системой.

Здесь стоит отметить, что хотя формально трансляция интерпретатором исходного кода в байткод и перевод байткода виртуальной машиной в набор машинных команд представляют два разных процесса, но фактически они объединены в самом интерпретаторе.)

2)Режим выполнения скрипта(Скриптовый язык. Код программ определяется в виде скриптов.)

**2.Работы с Питон в режиме калькулятор.**

Написать программу, которая выполняет над двумя вещественными числами одну из четырех арифметических операций (сложение, вычитание, умножение или деление). Программа должна завершаться только по желанию пользователя.

Чтобы программа самопроизвольно не завершалась, в ней надо запустить бесконечный цикл. Выход из него будем осуществлять с помощью оператора break, если пользователь вводит определенный символ вместо знака арифметической операции.

Если пользователь ввел знак, который не является ни знаком арифметической операции, ни символом-"прерывателем" работы программы, то вывести сообщение о некорректном вводе.

Если был введен один из четырех знаков операции, запросить ввод двух чисел.

В зависимости от знака операции выполнить соответствующее арифметическое действие.

Если было выбрано деление, необходимо проверить не является ли нулем второе число. Если это так, то сообщить о невозможности деления.

**print**("Ноль в качестве знака операции завершит работу программы")

**while** True:

s = input("Знак (+,-,\*,/): ")

**if** s == '0': **break**

**if** s **in** ('+','-','\*','/'):

x = float(input("x="))

y = float(input("y="))

**if** s == '+':

**print**("%.2f" % (x+y))

**elif** s == '-':

**print**("%.2f" % (x-y))

**elif** s == '\*':

**print**("%.2f" % (x\*y))

**elif** s == '/':

**if** y != 0:

**print**("%.2f" % (x/y))

**else**:

**print**("Деление на ноль!")

**else**:

**print**("Неверный знак операции!")

Ноль в качестве знака операции завершит работу программы

Знак **(**+,-,**\***,**/)**: h

Неверный знак операции**!**

Знак **(**+,-,**\***,**/)**: +

x=78.456

y=23.567

102.02

Знак **(**+,-,**\***,**/)**: -

x=3.44

y=10.25

-6.81

Знак **(**+,-,**\***,**/)**: **\***

x=0.012

y=-2.15

-0.03

Знак **(**+,-,**\***,**/)**: **/**

x=5

y=0

Деление на ноль**!**

Знак **(**+,-,**\***,**/)**: **/**

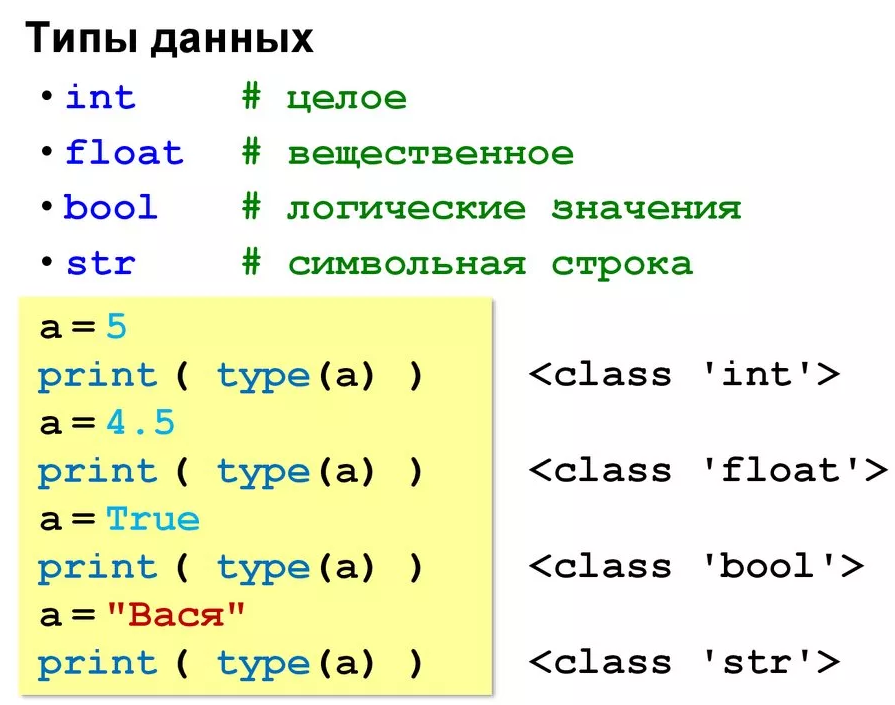
x=-2.3

y=5

-0.46

Знак **(**+,-,**\***,**/)**: 0

**3.Типы данных в Питоне. Встроенные типы данных. Пользовательские типы.**

****

Типы данных в питоне.

* Числа

Целые числа, числа с плавающей запятой и комплексные числа относятся к группе чисел. В Python они представлены классами int, float и complex.

* Списки

Список представляет собой упорядоченную последовательность элементов. Он очень гибкий и является одним из самых используемых типов в Python. Элементы списка не обязательно должны быть одного типа.

Объявить список довольно просто. Внутрь квадратных скобок помещаются элементы списка, разделённые запятой:

>>> a = [1, 2.2, 'python']

* Кортежи

Так же как и список, кортеж (tuple) является упорядоченной последовательностью элементов. Вся разница заключается в том, что кортежи неизменяемы.

Кортежи используются для защиты данных от перезаписи и обычно работают быстрее, чем списки, т.к. их нельзя изменять.

Для создания кортежа нужно поместить внутрь круглых скобок элементы, разделённые запятой:

>>> t = (5,'program', 1+3j)

* Строки

Строка представляет собой последовательность символов. Мы можем использовать одинарные или двойные кавычки для создания строки.  Многострочные строки можно обозначить тройными кавычками, ''' или """:

>>> s = "Простая строка"

>>> s = '''многострочная

строка'''

* Множества

Множество является неупорядоченной уникализированной последовательностью. Объявляется множество с помощью элементов, разделённых запятой, внутри фигурных скобок:

>>> a = {5,2,3,1,4}

# вывод переменной множества

>>> print("a =", a)

a = {1, 2, 3, 4, 5}

# тип данных переменной а

>>> print(type(a))

<class 'set'>



* **Словари**

Словари — неупорядоченные наборы пар ключ-значение.

Они используются, когда нужно сопоставить каждому из ключей значение и иметь возможность быстро получать доступ к значению, зная ключ. В других языках словари обычно называются map, hash или object. Словари оптимизированы для извлечения данных. Чтобы извлечь значение, нужно знать ключ.

Словарь объявляется парами элементов в форме ключ:значение, заключенными в фигурные скобки:

>>> d = {1:'value', 'key':2}

>>> type(d)

<class 'dict'>

* **Преобразование типов данных**

Мы можем преобразовывать значения из одного типа в другой с помощью таких функций, как int(), float(), str() и т.д.

>>> float(5)

5.0

При преобразовании числа с плавающей запятой в целое будет утеряна часть после запятой:

>>> int(10.6)

10

>>> int(-10.6)

-10

Для преобразования из/в строку должны использоваться совместимые значения:

>>> float('2.5')

2.5

>>> str(25)

'25'

>>> int('1p')

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: invalid literal for int() with base 10: '1p'

## Встроенные типы данных в Python:

Информация, сохраненная в памяти может быть разных типов данных. Например, возраст человека может быть числовым значением, а его адрес - буквенно-числовым. В **Python** существует множество стандартных типов данных, которые используются для хранения определенных значений и обладают своими специфическими методами.

К стандартным типам данных в **Python** относят:

* [*Числа (Numbers)*](http://pythonicway.com/python-data-types#num)
* [*Строка (String)*](http://pythonicway.com/python-data-types#str)
* [*Список (List)*](http://pythonicway.com/python-data-types#list)
* [*Кортеж (Tuple)*](http://pythonicway.com/python-data-types#tuple)
* [*Словарь (Dictionary)*](http://pythonicway.com/python-data-types#dict)
* [*Сет (Set)*](http://pythonicway.com/python-data-types#set)

**Пользовательские функции** объявляются с использованием ключевого слова def. Ключевое слово должно сопровождаться именем функции.

**4. Использование словарей, списков и множеств.**

В лекциях: Словарь {1:2, 2:3}

Список {1,2,3}

Множество {3,2,1} – удаляет одинаковые значения

В интернете: смотри 3 вопрос.

**5. Императивное программирование в Питоне. Скрипты Структура программы.**

В лекциях: Императивное программирование – описывает всю программу последовательно.

Декларативное программирование – что представляет задание и решение ее

Функциональное программирование – это функция в которой выводит один и тот же результат.

Логическое программирование.

Интернет: **Императивное программирование** — это парадигма программирования (стиль написания исходного кода компьютерной программы), для которого характерно следующее:

* в исходном коде программы записываются инструкции (команды);
* инструкции должны выполняться последовательно;
* при выполнении инструкции данные, полученные при выполнении предыдущих инструкций, могут читаться из памяти;
* данные, полученные при выполнении инструкции, могут записываться в память.

Императивная программа похожа на приказы (англ. **imperative** — приказ, повелительное наклонение), выражаемые повелительным наклонением в естественных языках, то есть представляют собой последовательность команд, которые должен выполнить компьютер.

Структура Питона

### [Лексическая структура](https://www.yuripetrov.ru/edu/python/ch_02_01.html#id63)

Любая Python-программа состоит из последовательности лексем (допустимых символов), записанных в определенном порядке и по определенным правилам.

Лексемы включают в себя:

* комментарии;
* литералы;

5 *# Целочисленный литерал*

3.4 *# Литерал в виде числа с плавающей точкой*

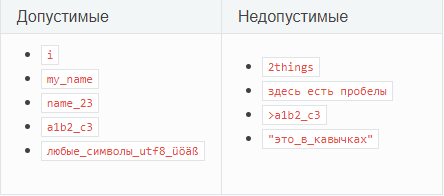
'строка' *# Строковый литерал*

* знаки пунктуации;

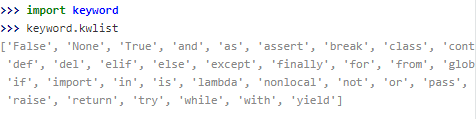
В алфавит Python входит достаточное количество знаков пунктуации, которые используются для различных целей. Например, знаки + или \* могут использоваться для сложения и умножения, а знак запятой , - для разделения параметров функций.

* идентификаторы;

Идентификатор - обычное имя, которое дается ссылке на какой-либо объект.

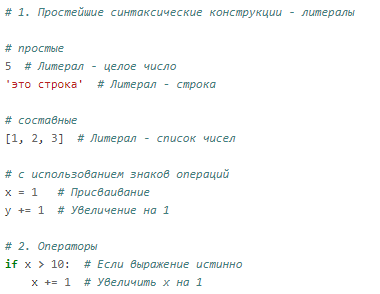


* ключевые слова.



### [Синтаксическая структура](https://www.yuripetrov.ru/edu/python/ch_02_01.html#id69)

Для составления программы необходимо объединить лексемы в синтаксические конструкции.



### [Составные элементы Python-программы](https://www.yuripetrov.ru/edu/python/ch_02_01.html#id70)

Python относится к языкам, разделяющим идею методологии [структурного программирования](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), в основе которой лежит представление программы в виде иерархической структуры блоков.

|  |  |  |
| --- | --- | --- |
| **№** | **Наименование блока** | **Описание** |
| 1 | Последовательность (инструкция) | Любое атомарное действие, например, присваивание |
| 2 | Ветвление (условие) | Выполнение инструкций в зависимости от определенного условия |
| 3 | Цикл | Многократное исполнение набора инструкций |
| 4 | Подпрограмма (процедура/функция) | Часть компьютерной программы, содержащая описание определенного набора инструкций, которая может быть многократно вызвана из разных частей программы. Может содержать (1)-(3) |
| 5 | Класс | Абстрактный тип данных в объектно-ориентированном программировании, задающий общее поведение для группы объектов; модель объекта. Может содержать (1)-(4) |
| 6 | Модуль | Функционально законченный фрагмент программы, оформленный в виде отдельного файла с исходным кодом или поименованной непрерывной ее части. Может содержать (1)-(5) |
| 7 | Пакет | Логически законченная совокупность модулей как единое целое |

|  |
| --- |

### [Структура файла и кодировка программы](https://www.yuripetrov.ru/edu/python/ch_02_01.html#id71)

Программный код на языке Python можно записать в любом простом текстовом редакторе, который способен загружать и сохранять текст либо в кодировке ASCII, либо UTF-8. По умолчанию предполагается, что файлы с программным кодом на языке Python сохраняются в кодировке UTF-8 - надмножестве кодировки ASCII, с помощью которой можно представить практически любой символ любого национального алфавита. Файлы с программным кодом на языке Python обычно имеют расширение .ру

**6Видимость переменных.**

В **Python** переменная доступна только в той области видимости, внутри которой она создана. Эта область называется **областью видимости**. Существует несколько видов областей видимости.

#### Локальная область видимости

Переменная, созданная внутри функции принадлежит локальной области видимости этой функции, другими словами она может использоваться (видна) только внутри этой функции.

#### Вот пример:

#### 

#### Функция внутри функции

В примере выше переменная r доступна только внутри той функции, в которой она определена, т.е. не доступна снаружи, но она будет доступна для любой функции, которая находится внутри данной.

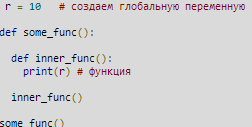
Пример:

#### 

#### Глобальная область видимости:

Если переменная создается вне какой либо функции, то такая переменная попадает в глобальную область видимости и становится глобальной.

Глобальные переменные - это переменные доступные в любой области видимости. Посмотрите на пример далее:



Но использование глобальных переменных может привести к появлению трудно обнаруживаемым ошибкам, в особенности, если одну глобальную переменную изменяют несколько функций.

Поэтому в глобальной области видимости лучше определять неизменяемые переменные - константы, где-то в начале файла.

#### Именование переменных

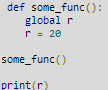
Переменная с одним и тем же именем, но в разных областях видимости в **Python** рассматривается по разному.

Вот пример:

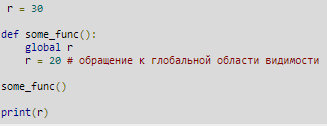
#### 

#### Ключевое слово global в Python

Использование ключевого слова global необходимо тогда, есть потребность сделать локальную переменную глобальной.



Также с помощью данного ключевого слова можно изменить значение глобальной переменной из локальной области. Вот как это можно сделать:

****

**7. Правило оформление кода. PEP8 и его критика.**

Ключевая идея Гуидо такова(PEP 8 создан на основе рекомендаций Гуидо): код читается намного больше раз, чем пишется. Собственно, рекомендации о стиле написания кода направлены на то, чтобы улучшить читаемость кода и сделать его согласованным между большим числом проектов. В идеале, весь код будет написан в едином стиле, и любой сможет легко его прочесть.

Это руководство о согласованности и единстве. Согласованность с этим руководством очень важна. Согласованность внутри одного проекта еще важнее. А согласованность внутри модуля или функции — самое важное. Но важно помнить, что иногда это руководство неприменимо, и понимать, когда можно отойти от рекомендаций. Когда вы сомневаетесь, просто посмотрите на другие примеры и решите, какой выглядит лучше.

Две причины для того, чтобы нарушить данные правила:

1. Когда применение правила сделает код менее читаемым даже для того, кто привык читать код, который следует правилам.
2. Чтобы писать в едином стиле с кодом, который уже есть в проекте и который нарушает правила (возможно, в силу исторических причин) — впрочем, это возможность переписать чужой код.

[**Внешний вид кода**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id2)

**-**[**Отступы**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id3)

**-**[**Табуляция или пробелы?**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id4)

**-**[**Максимальная длина строки**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id5)

Ограничьте длину строки максимум 79 символами.

**-**[**Пустые строки**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id6)

Отделяйте функции верхнего уровня и определения классов двумя пустыми строками.

Определения методов внутри класса разделяются одной пустой строкой.

**-**[**Кодировка исходного файла**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id7)

Кодировка Python должна быть UTF-8

**-**[**Импорты**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id8)

Каждый импорт, как правило, должен быть на отдельной строке.

[**Пробелы в выражениях и инструкциях**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id9)

**-**[**Избегайте использования пробелов в следующих ситуациях:**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id10)

Непосредственно внутри круглых, квадратных или фигурных скобок.

Непосредственно перед запятой, точкой с запятой или двоеточием:

Сразу перед открывающей скобкой, после которой начинается список аргументов при вызове функции

Сразу перед открывающей скобкой, после которой следует индекс или срез

Использование более одного пробела вокруг оператора присваивания (или любого другого) для того, чтобы выровнять его с другим

**-**[**Другие рекомендации**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id11)

Всегда окружайте эти бинарные операторы одним пробелом с каждой стороны: присваивания (=, +=, -= и другие), сравнения (==, <, >, !=, <>, <=, >=, in, not in, is, is not), логические (and, or, not).

Не используйте пробелы вокруг знака =, если он используется для обозначения именованного аргумента или значения параметров по умолчанию.

Не используйте составные инструкции (несколько команд в одной строке).

[**Комментарии**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id12)

**-**[**Блоки комментариев**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id13)

Блок комментариев обычно объясняет код (весь, или только некоторую часть), идущий после блока, и должен иметь тот же отступ, что и сам код. Каждая строчка такого блока должна начинаться с символа # и одного пробела после него (если только сам текст комментария не имеет отступа).

**-**[**"Встрочные" комментарии**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id14)

Такой комментарий находится в той же строке, что и инструкция. "Встрочные" комментарии должны отделяться по крайней мере двумя пробелами от инструкции. Они должны начинаться с символа # и одного пробела.

**-**[**Строки документации**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id15)

Пишите документацию для всех публичных модулей, функций, классов, методов. Строки документации необязательны для приватных методов, но лучше написать, что делает метод. Комментарий нужно писать после строки с def.

**8. объектно-ориентированное программирование на ПИТОНе.**

**Объектно-ориентированное программирование** (**ООП**) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов.

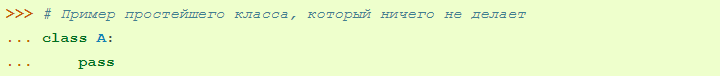
**Класс** — тип, описывающий устройство объектов. **Объект** — это экземпляр класса. Класс можно сравнить с чертежом, по которому создаются объекты.

Python соответствует принципам объектно-ориентированного программирования. В python всё является объектами - и строки, и списки, и словари, и всё остальное.

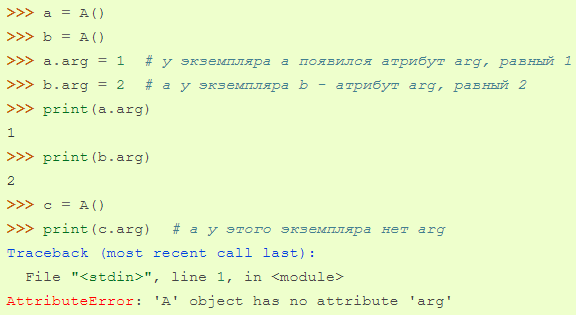
Но возможности ООП в python этим не ограничены. Программист может написать свой тип данных (класс), определить в нём свои методы.

Это не является обязательным - мы можем пользоваться только встроенными объектами. Однако ООП полезно при долгосрочной разработке программы несколькими людьми, так как упрощает понимание кода.

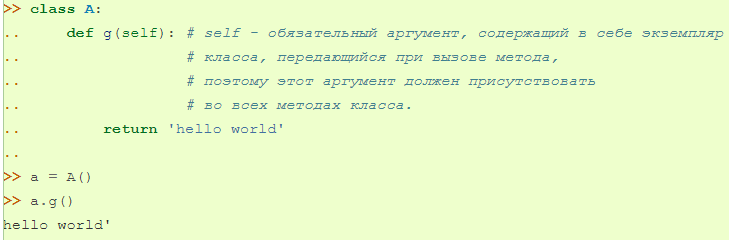
Приступим теперь собственно к написанию своих классов на python. Попробуем определить собственный класс:

****

Теперь мы можем создать несколько экземпляров этого класса:

****

Классу возможно задать собственные методы:

****

**9.Стандартные классы. Пользовательские классы. Имитация функции.**

Стандартные классы -

Пользовательские классы – это создание своего класса(смотри вопрос 8)

Имитация функции - lambda

**10.Декларативное программирование на Питоне. Списковые включения.**

**Декларати́вное** **программи́рование** **это** парадигма **программирования** , в которой задаётся спецификация решения задачи, то есть описывается, что представляет собой проблема и ожидаемый результат.

Для более естественной записи обработки списков в Python 2 была внесена новинка: списковые включения. Фактически это специальный сокращенный синтаксис для вложенных циклов for и условий if, на самом низком уровне которых определенное выражение добавляется к списку, например:

all\_pairs = []

for i in range(5):

for j in range(5):

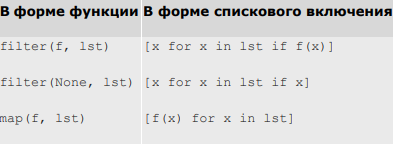
if i <= j: all\_pairs.append((i, j))

Все это можно записать в виде спискового включения так:

all\_pairs = [(i, j) for i in range(5) for j in range(5) if i <= j]

Как легко заметить, списковые включения позволяют заменить map() и filter() на более удобные для прочтения конструкции.

В следующей таблице приведены эквивалентные выражения в разных формах:

****

**11. Декларативное программирование на Питоне. Map.**

Функция map() Для применения некоторой функции ко всем элементам последовательности применяется функция map(f, \*args). Первый параметр этой функции - функция, которая будет применяться ко всем элементам последовательности. Каждый следующий n+1 -й параметр должен быть последовательностью, так как каждый его элемент будет использован в качестве n -го параметра при вызове функции f(). Результатом будет список, составленный из результатов выполнения этой функции.

В следующем примере складываются значения из двух списков:

>>> l1 = [2, 7, 5, 3]

>>> l2 = [-2, 1, 0, 4]

>>> print map(lambda x, y: x+y, l1, l2) [0, 8, 5, 7]

В этом примере применена безымянная функция для получения суммы двух операндов ко всем элементам l1 и l2. В случае если одна из последовательностей короче другой, вместо соответствующего операнда будет None, что, конечно, собьет операцию сложения. В зависимости от решаемой задачи, можно либо видоизменить функцию, либо считать разные по длине последовательности ошибкой, которую нужно обрабатывать как отдельную ветвь алгоритма. Частный случай применения map() - использование None в качестве первого аргумента. В этом случае просто формируется список кортежей из элементов исходных последовательностей:

>>> l1 = [2, 7, 5, 3]

>>> l2 = [-2, 1, 0, 4]

>>> print map(None, l1, l2) [(2, -2), (7, 1), (5, 0), (3, 4)]

**12. Декларативное программирование на Питоне. Lamda.**

С помощью ключевого слова lambda в Python можно определять анонимные функции, которые являются объектами функций, аналогично функциям, объявленными с помощью ключевого слова def.

Синтаксис объявления

|  |  |
| --- | --- |
| 1 | lambda arguments: expression |

arguments - аргументы, expression - выражение, возвращающее значение.

Lambda функции удобно использовать для функций, которые вызываются один раз, а также для функций, которые выполняют одно выражение.

**13. Декларативное программирование на Питоне. Reduce.**

Для организации цепочечных вычислений (вычислений с накоплением результата) можно применять функцию reduce(), которая принимает три аргумента: функцию двух аргументов, последовательность и начальное значение. С помощью этой функции можно, в частности, реализовать функцию sum():

def sum(lst, start): return reduce(lambda x, y: x + y, lst, start)

Совет: Следует помнить, что в качестве передаваемого объекта может оказаться список, который позволит накапливать промежуточные результаты. Тем самым, reduce() может использоваться для генерации последовательностей.

В следующем примере накапливаются промежуточные результаты суммирования:

lst = range(10)

f = lambda x, y: (x[0] + y, x[1]+[x[0] + y])

print reduce(f, lst, (0, []))

В итоге получается: (45, [0, 1, 3, 6, 10, 15, 21, 28, 36, 45])

**14. Декларативное программирование на Питоне. Filter.**

Функция filter()

Другой часто встречающейся операцией является фильтрование исходной последовательности в соответствии с некоторым предикатом (условием). Функция filter(f, seq) принимает два аргумента: функцию с условием и последовательность, из которой берутся значения. В результирующую последовательность попадут только те значения из исходной, для которой f() возвратит истину. Если в качестве f задано значение None, результирующая последовательность будет состоять из тех значений исходной, которые имеют истинностное значение True.

Например, в следующем фрагменте кода можно избавится от символов, которые не являются буквами:

>>> filter(lambda x: x.isalpha(), 'Hi, there! I am eating an apple.')

'HithereIameatinganapple'