1. В Рython не требуется явного объявления переменных.

2. Отсутствуют операторные скобки ((begin..end в pascal или {..}в Си) вместо этого **блоки выделяются отступами**: пробелами или табуляцией, а вход в блок из операторов осуществляется двоеточием.

def get\_fibonacci\_element(n):

fib1 = 0

fib2 = 1

i = 0

while i < n:

fib = fib1

#print(" ",fib," ")

fib1 = fib2

fib2 += fib

i += 1

return fib  
  
Блок кратен 4 «пробелам».

3. Комментарии

Однострочные комментарии начинаются со знака фунта «#», многострочные — начинаются и заканчиваются тремя двойными кавычками «"""». 

>>> myvar -= 1 # это составной оператор присваивания

# Комментарий  может быть, как в начале строки, так в строке после операторов

""Это многострочный комментарий   
Строки заключенные в три двойные кавычки игнорируются""

>>> myvar = 3  
>>> myvar += 2  
>>> myvar -= 1  
""«Это многострочный комментарий   
Строки заключенные в три двойные кавычки игнорируются»""  
>>> mystring = «Hello»  
>>> mystring += " world."  
>>> **print** mystring  
Hello world.  
# Следующая строка меняет   
значения переменных местами. (Всего одна строка!)  
>>> myvar, mystring = mystring, myvar

**Типы данных**

Синтаксис Python подразумевает обязательное определение типа данных для переменных, констант, массивов, списков и т.д. Основные типы ничем не отличаются от других языков с жестко заданной типизацией.

Самые важные типы:

1. Числовые: целые, дробные, вещественные с плавающей точкой, комплексные.
2. Логические: тип для хранения значений алгебры логики – «истина» или «ложь».
3. Строковые: содержат символы Юникода, в том числе, html-код.
4. Списки – упорядоченные массивы переменных.

ПОСЛЕДОВАТЕЛЬНОСТИ:

1. Кортежи – массив упорядоченных констант, т.е. значений, которые не могут изменяться в процессе работы.
2. Множества – массивы неупорядоченных данных.
3. Словари – специализированный массив, состоящий из пары – «ключ» — «значение».
4. Байты, массивы байтов – поименованные области памяти для хранения изображений (jpg, gif и т.д.), pdf-документов и других файлов.

Индексация в последовательностях начинается с 0 до N. Т.е. последний элемент имеет индекс -1.

>>> sample = [1, [«another», «list»], («a», «tuple»)] #Список состоит из целого числа, другого списка и кортежа  
>>> mylist = [«List item 1», 2, 3.14] #Этот список содержит строку, целое и дробное число  
>>> mylist[0] = «List item 1 again» #Изменяем первый (нулевой) элемент листа mylist  
>>> mylist[-1] = 3.14 #Изменяем последний элемент листа  
>>> mydict = {«Key 1»: «Value 1», 2: 3, «pi»: 3.14} #Создаем словарь, с числовыми и целочисленным индексами  
>>> mydict[«pi»] = 3.15 #Изменяем элемент словаря под индексом «pi».  
>>> mytuple = (1, 2, 3) #Задаем кортеж

Можно использовать часть массива, задавая первый и последний индекс через двоеточие «:». В таком случае вы получите часть массива, от первого индекса до второго не включительно. Если не указан первый элемент, то отсчет начинается с начала массива, а если не указан последний — то масив считывается до последнего элемента. Отрицательные значения определяют положение элемента с конца. Например:

>>> mylist = [«List item 1», 2, 3.14]  
>>> **print** mylist[:] #Считываются все элементы массива  
['List item 1', 2, 3.1400000000000001]  
>>> **print** mylist[0:2] #Считываются нулевой и первый элемент массива.  
['List item 1', 2]  
>>> **print** mylist[-3:-1] #Считываются элементы от нулевого (-3) до второго (-1) (не включительно)  
['List item 1', 2]  
>>> **print** mylist[1:] #Считываются элементы от первого, до последнего  
[2, 3.14]

Операторы

Операторы **while**, **if**, **for** составляют операторы перемещения (операторы изменяющие последовательный порядок выполнения программы).

В операторе **for** происходит сравнение **переменной и списка**. Чтобы получить список цифр до числа <number> — используйте функцию range(<number>). Вот пример использования операторов

**for** number in range(1, 20, 2) :

z=number \*number

print (“квадрат ”, number, “равен”, z )

rangelist = range(10) #Получаем список из десяти цифр (от 0 до 9)  
>>> **print** rangelist  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**for** number in rangelist: #Пока переменная number (которая каждый раз увеличивается на единицу) входит в список…   
# Проверяем входит ли переменная  
# numbers в кортеж чисел (3, 4, 7, 9)  
 **if** number in (3, 4, 7, 9): #Если переменная number входит в кортеж (3, 4, 7, 9)...  
# Операция «**break**» обеспечивает  
# выход из цикла в любой момент  
 **break**   
 **else**:  
# «**continue**» осуществляет «прокрутку»  
# цикла. Здесь это не требуется, так как после этой операции   
# в любом случае программа переходит опять к обработке цикла  
 **continue**

**else**:  
# «**else**» указывать необязательно. Условие выполняется  
# если цикл не был прерван при помощи «**break**».  
**pass** # Ничего не делать  
   
**if** rangelist[1] == 2:  
 **print** «The second item (lists are 0-based) is 2»  
**elif** rangelist[1] == 3:  
 **print** «The second item (lists are 0-based) is 3»  
**else**:  
**print** «Dunno»  
   
**while** rangelist[1] == 1:  
**pass**

Функции

Для объявления функции служит **ключевое слово «def»**. Аргументы функции задаются в скобках после названия функции. Можно задавать необязательные аргументы, присваивая им значение по умолчанию. Функции могут возвращать кортежи, в таком случае надо писать возвращаемые значения через запятую. Ключевое слово «**lambda**» служит для объявления элементарных функций .

# arg2 и arg3 — необязательые аргументы, принимают значение объявленное по умолчни,  
# если не задать им другое значение при вызове функци.  
**def** myfunction(arg1, arg2 = 100, arg3 = «test»):  
 **return** arg3, arg2, arg1  
#Функция вызывается со значением первого аргумента — "Argument 1", второго — по умолчанию, и третьего — "Named argument".  
>>>ret1, ret2, ret3 = myfunction(«Argument 1», arg3 = «Named argument»)  
# ret1, ret2 и ret3 принимают значения "Named argument", 100, "Argument 1" соответственно  
>>> **print** ret1, ret2, ret3  
Named argument 100 Argument 1  
   
# Следующая запись эквивалентна **def** f(x): **return** x + 1  
functionvar = **lambda** x: x + 1  
>>> **print** functionvar(1)  
2

Импорт

Внешние библиотеки можно подключить процедурой «**import** [libname]», где [libname] — название подключаемой библиотеки. Вы так же можете использовать команду «**from** [libname] **import**[funcname]», чтобы вы могли использовать функцию [funcname] из библиотеки [libname]

**import** math #Импортируем библиотеку математических функций

**import** random #Импортируем библиотеку «random»  
**from** time **import** clock #И заодно функцию «clock» из библиотеки «time»  
   
randomint = random.randint(1, 100)  
>>> **print** randomint  
64

Особенности

* **Условия могут комбинироваться.** 1 < a < 3 выполняется тогда, когда а больше 1, но меньше 3.
* Используйте операцию «**del**» чтобы **очищать переменные или элементы массива**.
* Python предлагает большие возможности для **работы со списками**. Вы можете использовать операторы объявлении структуры списка. Оператор **for** позволяет задавать элементы списка в определенной последовательности, а **if** — позволяет выбирать элементы по условию.

>>> lst1 = [1, 2, 3]  
>>> lst2 = [3, 4, 5]  
>>> **print** [x \* y **for** x in lst1 **for** y in lst2]  
[3, 4, 5, 6, 8, 10, 9, 12, 15]

>>> **print** [x **for** x in lst1 **if** 4 > x > 1]  
[2, 3]  
# Оператор «any» возвращает true, если хотя   
# бы одно из условий, входящих в него, выполняется.  
>>> any(i % 3 **for** i in [3, 3, 4, 4, 3])  
True  
# Следующая процедура подсчитывает количество   
# подходящих элементов в списке  
>>> sum(1 **for** i in [3, 3, 4, 4, 3] **if** i == 3)  
3  
>>> **del** lst1[0]  
>>> **print** lst1  
[2, 3]  
>>> **del** lst1