Использование языка Python в задачах робототехники

Кирсанов К.Б.

ИПМ им. Келдыша РАН, Международная лаборатория “Сенсорика“

05.10.09 / Школа молодых учёных Graphicon‘2009

Введение

Python — высокоуровневый интерпретируемый динамичский язык общего назначения.

- Основные особенности: минималистский синтаксис, динамическая типизация, автоматическое управление памятью, интроспекция, элементы функционального программирования, высокоуровневые структуры данных

- Автор языка, голландец Гвидо ван Россум (Guido van Rossum), создал его как адаптацию учебного языка ABC для нужд unix/C программистов

Введение

Результат исполения “import this“

- Красивое лучше уродливого.
- Явное лучше неявного.
- Простое лучше сложного.
- Сложное лучше усложнённого.
- Единое лучше сцепленного.
- Рассеянное лучше плотного.
- Чёткость важна.
- Частные случаи не настолько существенны, чтобы нарушать правила.
- Хотя практичность важнее чистоты.
- Ошибки никогда не должны умалчиваться.
- Кроме тех случаев, когда они явны.

Введение

Результат исполения “import this“. Продожение.

- Если что-то не ясно, сопротивляйтесь искушению угадать.
- Должен существовать один — и, желательно, только один — очевидный способ сделать это.
- Хотя поначалу он может быть не очевиден, если только ты не голландец.
- Сейчас лучше, чем никогда.
- Хотя никогда часто бывает лучше, чем прямо сейчас.
- Если реализацию сложно объяснить — это плохая идея.
- Если реализацию легко объяснить — это может быть хорошей идеей.
- Пространства имён — великолепная идея. Давайте придумаем больше таких!

Введение

Влияние других языков

Python не содержит ни одной новой идеи, но является удачной комбинацией уже существующих:

- ABC - отступы для группировки операторов
- Haskell - операции над списками
- Modula-3 - пакеты, модули, именованные аргументы функций
- Smalltalk - объектно-ориентированное программирование
- Lisp – элементы функционального программирования.
- Java – часть стандартной библиотеки

Введение

Реализации языка

Язык не стандартизируется ISO. В качестве стандарта выступает реализация CPython.

- Интерпретатор:
  - CPython – стандартная реализация (X86, ARM, MIPS)
  - TinyPy - минималистский реализация (64K)
  - Python for S60 - Symian(Nokia)
- Компилятор в байткод виртуальных машин
  - Jython – Java
  - Iron Python – Microsoft .NET
  - Unladen Swallow – LLVM (Low Level Virtual Machine)
- Другие реализации:
  - PyPy – Интерпретатор Python, написанный на Python

Язык Python

Пример простейшей программы. Отсупы как операторные скобки

```
1  a=1
2  b=2
3  c=a+b
4  c+=2
5
6  a="123"
7  b=a+"'asd'"
8  if a == "123":
9      print a
10 else:
11     pass
12
13 a = "1"+2 #cannot concatenate 'str' and 'int'
```

Язык Python

Модули

Модули Python должны находится в текущем каталоги или в одной из системных каталогов из sys.path. Модули могут быть упакованы в ZIP.

Подключение модуля осуществляется п мощью команты import

```
1 import time
2 import time as t
3
4 from time import sleep
5 from time import sleep as s
6
7 from math import *
```

При импортировании модули кешируются и их повторное импортировние игнорируется

Язык Python

Типы данных

```
1  a = 1 #int
2  b = "123" #string
3  c = u"123" #UTF string
4  d = 99999999999999999999999999 #long
5  e = 123.321 #double
6  f = 1 + 2j #complex
7  g = True #bool
8  d = None #None
9
10 def F(): pass
11 e = F # function
12
13 class C:pass
14 f = C # class
15 g = C() #C
```

Язык Python

Типы данных - Списки

```
1  a = [] # empty list
2  b = [1, 2, 3]
3  c = [1, "asd", b]
4  d = b + c
5  c.append(1)
6  print c[2] # asd
7  del c[0]
8
9  if 2 in b:
10     print "2 in ", b
11 if "zxc" not in b:
12     print "'zxc' not in", b
13 if a:
14     print "not empty"
```

Язык Python

Операции над списками: срезы

Пусть дан список A=[1,2,3,4,5,6,7,8,9]

- A или A[:] - все элементы
- A[0] – первый элемент
- A[-1] – последний элемент
- A[0:3] – с первого по третий
- A[:-2] – все элементы, кроме двух последних
- A[2:] – все элементы, кроме двух первых
- A[0:8:2] – элементы с 1-го по 8-й с шагом 2

Дополнительные операции

- len(A), max(A), min(A), sum(A), sort(A)
- распаковка - a,b = [1,2]

Язык Python

Операции над списками. Пример: Контроль длинны истории измерений датчика

```
1  maxLen = 1000
2  measureHistory = []
3
4  while True:
5      measureHistory.append(GetNewMeasure())
6      if len(measureHistory) > maxLen:
7          measureHistory = measureHistory[-maxLen:]
```

Язык Python

Списковые операции

```
1  L = range(10)#[0,1,2,3,4,5,6,7,8,9]
2  a = [x + 1 for x in L]#[1,2,3,4,5,6,7,8,9,10]
3  b = [x for x in L if x % 2 == 0]#[0, 2, 4, 6, 8]
4
5  L = [[1, 2], [3, 4], [5, 6]]
6  a = [a for a, b in L] #[1, 3, 5]
```

Язык Python

Типы данных - Словари

Словарь (ассоциативный массив) - осуществляет к элементу произвольного типа по ключу произвольного типа.

```
1  d = {} # empty dict
2  d = {1:2, 2:3, "asd":3, 4:"dsa"}
3  d[6] = "asd"
4
5  if d.has_key(1):
6      print d[1]
```

Язык Python

Типы данных - Котрежи и строки

Кортеж (tuple) - неизменяемая(immutable) последовательность объектов.

```
1  a = (1, 2)
2  a += (3, 4) # (1,2,3,4)
3  print a[2] #3
4  #a[2] =5#'tuple' object does not support item assignment
5  a = (1,) #(1)
6
7  st = "as df gh jk"
8  if "as" in st:
9      print "Bingo"
10 l = st.split(" ")# ['as', 'df', 'gh', 'jk']
```

Язык Python

Организация циклов

В языке Python приняты циклы, перебирающие элементы:

```
1  for x in [1, 2, 3]:
2      print x
3
4  d = {}
5  for key, value in d.items():
6      print key, "->", value
7
8  i=0
9  while i<10:
10     print i
11     i+=1
```

Язык Python

Функции и их аргументы

Значения передаются “по ссылке“. Возможно использование именованных аргументов.

```
1  def f(x, y=1, z=3):
2      return [x, y, z]
3  f(1)
4  f(1, z=3)
5  f(z=3, y=2, x=1)
6
7  def Adder(x):
8      def add_x(val):
9          return val+x
10     return add_x
11
12 add2 = Adder(2)
13 print add2(2) # 4
```

Из за принятной типизации невозможно создать перегруженные функции

Язык Python

Функции и их аргументы. Списковые аргументы

```
1  def GetPixcelColor(x, y):
2      return (1, 2, 3)
3  pos = (1, 2)
4
5  color = GetPixcelColor(pos[0], pos[1])
6  color = GetPixcelColor(*pos)
7  r, g, b = GetPixcelColor(*pos)
8
9  pos = {'x':1, 'y':2}
10 r, g, b = GetPixcelColor(**pos)
11
12 def F(*args, **kwargs):
13     print args #(1, 2, 3)
14     print kwargs #{'z': 4}
15
16 F(1,2,3,z=4)
```

Язык Python

Замыкания - функции, ссылающиеся на свободные переменные в своём лексическом контексте

Пример: Добавление временных меток к сенсорным данным

```
1  import time, random
2  def AddTime(fn, *args, **kwargs):
3      #generates new new_fn every time
4      def new_fn(*args, **kwargs):
5          data = fn(*args, **kwargs)
6          return (time.time(), data)
7      return new_fn
8
9  @AddTime #decorator
10 def GetMeasure():
11     return random.random()
12 print GetMeasure() # (1254785016.7783949, 0.46029099903771442
```

Язык Python

Продолжения(generators, continuation) - представляет состояние программы в определённый момент, которое может быть сохранено и использовано для перехода в это состояние

```
1  data = range(100)
2  def CalcPow2(a):
3      for x in a:
4          yield x*x
5
6  for x in CalcPow2(data):
7      print x
8
9  d = (x*x for x in data)
```

Язык Python

Функции высших порядков - функции принимающие в ачестве аргументов другиме функции

```
1  inc = lambda x:x+1
2  def inc(x): return x + 1
3
4  a = [1, 2, 3]
5
6  incs = map (inc, a)
7  incs = map (lambda x:x + 1, a)#2,3,4
8  f = filter (lambda x:x<2, a) #1
9
10 measureTimes = [1,2,3]
11 m = min(measureTimes)
12 measureTimes = map(lambda x:x-m, measureTimes)
```

Язык Python

Классы

```
1  class A:
2      a = 1
3      def SetA(self, a):
4          self.a = a
5          self.c = self.a
6      def __init__(self):
7          pass
8  class B(C):
9      def __init__(self):
10         C.__init__(self)
11
12 b=B(1)
13 b.d = 123
14 B.e=321
```

Проблемы

Низкая производительность

Интерпретатор Python, хотя и является одним из самых быстрых (в 2-3 раза быстрее tk) но, тем не мение, заметно отстает от C (до 1000 раз на некоторых тестах)
Пути решения:

- psyco - ускорение в 2-100 раз
- Выявить вычислительно-сложные функции и переписать их на C
- Воспользоваться математическими библиотеками для Python
- Реализовывать на Python лишь общую логику программы, а все частности решать в других языках

Проблемы:Низкая производительность

Решение: psyco

http://psyco.sourceforge.net/
Осуществляет частичную специализацию программы.

- Существенный расход памяти.
- Автор покинул проект и перешел в PyPy
- Дает стократное ускорение лишь на специально подготовленных тестах.

Проблемы:Низкая производительность

psyco: пример использования

```
1  from math import sin, cos
2  from time import time
3  import psyco
4  a,b,c,d = 1,2,3,4
5
6  def F(x):
7      return x*a/b+3-4/5*6-sin(c+d*2)/5*6+cos(x*x)
8
9  def Calc():
10     t0 = time()
11     for x in xrange(1, 100000):
12         F(x);F(x);F(x);F(x);F(x)
13     print time() - t0
14
15 Calc() # 1.07147097588
16 psyco.full()
17 Calc() # 0.17466211319
```

Проблемы

Невозможность организации параллельных вычичлений в CPython

Глобальная блокировка интерпретатора (GIL) - CPython не потокобезопасен. Все потоки реализуются через коллективную многозадачность. Переключение задач раз в 100 инструкций Это не является проблемой т.к.:

- Можно использовать MPI
- На бортовой ЭВМ всё равно 1 энергоэффективный прцессор и истинная многопоточность не нужна
- GIL на распространяется на написанные на C фрагменты, т.е. на операции ввода-вывода, функции библиотек и т.п.

Это удобно:

- Не нужно думать о примитивах синхронизации
- Упрощается отладка
- Экономия памяти

WEB сервер Tornado, облсуживающий Facebook, написан на

Проблемы

Склонность к ошибкам - Динамическая типизация лишает программиста контроля типов со стороны компилятора

Пути решения:

- Использовать PyChecker или pylint (анализаторы исходного кода, обнаруживают большинство ошибок)
- Регулярное регрессионное тестирование
- Реализовывать на Python лишь общую логику программы, а все частности решать в других языках

Библиотеки

В репозитарии Lunux Ubutu более 1400 Python программ и библиотек

Pyhon чрезвычайно просто интегрируется с существующими программами на C. В результате для него существует огромное количество сторонних библиотек:

- psyco - специализирующий компилятор
- numpy - быстрые вычисления а массивах
- mathplotlib - графопостроитель
- scipy - научные расчеты (включает numpy)
- rpyc - распределенные вычисления
- pygame - создание комп.игр (джойстик, клавиатура, мышь, 2D и звуковые библиотеки)
- pyserial - взаимодействие по RS-232
- pyCUDA
- OpenCV
- OpenGl, BluetTooth, ...

Интеграция

Встраивание и расширение

Python предлагает 2 способа интеграции с уже существующим ПО:

- Встраивание(embedding): интерпретатор Python встраивается в целевое ПО
- Расширение(extending): целевое ПО реализуется как библиотека для Python

Т.к. CPython реализован на C, то обе эти операции выполняются довольно легко. Суеществует ряд библиотек и программ, ещё больше упрощающих этот процесс: Boost::Python, Swing, Blitz, Pyrex, f2py, Weave

Интеграция

Встраивание с использовнием boost::python

```
1  #include <Python.h>
2  #include <boost/python.hpp>
3
4  int add_five(int x) {return x + 5;}
5  BOOST_PYTHON_MODULE(Test){
6      def("add_five", add_five);
7  }
8
9  int main(){
10     Py_Initialize();
11     initTest();
12     PyRun_SimpleString("import Test");
13     PyRun_SimpleString("print Test.add_five(4)");
14     return 0;
15 }
```

Интеграция

Расширение с использовнием scipy.wave

Компилирует C++ вставки в Python коде “на лету“, автоматически сосздавя подключаемые модули и повторно используя их при необходимости:

```
1 import scipy.weave as wave
2 from time import time
3 data = 1
4 code = "return_val = data+1;"
5 result = wave.inline(code, ['data'], compiler='gcc')
```