СОДЕРЖАНИЕ

|  |  |
| --- | --- |
| Задание | 2 |
| Выбор средств для решения задания | 3 |
| Обоснование выбора средств решения задачи | 4 |
| Алгоритм | 5 |
| Примеры | 14 |
| Код | 19 |
| ПРИЛОЖЕНИЕ 1 | 26 |
| ПРИЛОЖЕНИЕ 2 | 27 |

Задание

Произвести синонимизацию повторяющихся слов в предложении следующих частей речи:

* существительных
* глаголов
* прилагательных

Входные данные: предложение на русском языке.

Выходные данные: исходное предложение, в котором слова, повторявшиеся в входных данных, заменены на синонимы.

Выбор средств для решения задачи

Для решения задачи был выбран язык высокого уровня Python(см. ПРИЛОЖЕНИЕ 1) со средой разработки IDLE и использованием библиотеки pymorphy(см. ПРИЛОЖЕНИЕ 1).

Обоснование выбора средств решения задачи

Языком программирования был выбран Python (см. ПРИЛОЖЕНИЕ 1), потому что этот язык ориентирован на повышение производительности разработчика и читаемости кода. Это очень мощный язык, у него много «фанатов», что предоставляет возможность для консультирования и быстрой помощи в случае возникновения трудностей.

Библиотекой была выбрана pymorphy (см. ПРИЛОЖЕНИЕ 1), так как она очень легка в использовании, имеет множество подключаемых словарей, поддерживает разбор слов с дефисами и со сложными префиксами.

Алгоритм

Все прототипы функций взяты с сайта <http://pythonhosted.org/pymorphy> (<http://pythonhosted.org/pymorphy/ref/morph.html> ), <http://docs.python.org/2.7> и <http://pythonworld.ru> .

Вся справочная информация взята с сайта <http://pythonhosted.org/pymorphy> , <http://rd.feb-web.ru/pollex-12-1.html> и <http://habrahabr.ru/post/49421/> .

1. Считать входные данные (предложение)

Считать входные данные лучше всего либо при вводе с консоли, либо из файла. Удобнее сделать считывание с консоли, так как работа с файлом занимает дополнительное время у пользователя и ресурсы у системы, а при необходимости сохранения выходных данных это можно сделать вручную, скопировав с консоли.

Прототип функции, считывающей данные с консоли:

**raw\_input**([prompt])

1. Нахождение нормальной формы слов

Для слов предложения, которые принадлежат к следующим частям речи:

* существительное
* глагол
* прилагательное

нужно найти их нормальную форму. Нормальная форма для существительных представляет собой существительное в именительном падеже и в единственном числе. Для глагола – инфинитив, соответственно для прилагательного это именительный падеж единственного числа мужского рода.

* 1. Разбиение входного текста на токены

Для морфологического анализа необходимо подать в анализатор *слово*.

Так как предложение чаще всего состоит из *нескольких* слов, *знаков* *пунктуации* и *чисел*, необходимо выделить слова.

Прототип функции, разбивающей текст на токены(слова, знаки препинания):

tokenizers.**extract\_tokens**(text)

Пример:

|  |
| --- |
| **from** pymorphy **import** get\_morph *# портируем модули*  **from** pymorphy.contrib **import** tokenizers  morph = get\_morph('C:/Python27/ru.sqlite-json/') *# создаем объект класса pymorph*  text = (raw\_input("Входные данные():  ")).decode('cp1251')  **print** "Выходные данные():",  **for** token **in** tokenizers.extract\_tokens(text): *# выделяем токены*  **print** token |

* 1. Нахождение слов из выделенных токенов

Токены, найденные на втором шаге алгоритма, содержат не только слова, но еще и знаки пунктуации и пробелы, что не позволяет каждый токен подавать в анализатор. Будем считать, что токен содержит слово, если он состоит только из одних букв, не содержит ни пробелов, ни знаков препинания (кроме дефисов), ни цифр.

Приведем прототип функции, которая проверяет символы строки (строка – это тип данных) на содержание букв. Если все символы являются буквами, функция возвращает *True*(значение булево типа), иначе *False*(значение булево типа).

Прототип этой функции:

s.**isalpha**(), где *s* – строка.

Пример:

|  |
| --- |
| s = u‘дерево’ *# строка*  **if** s.isalpha() == True: *# проверка символов строки*  **print** str(‘Строка {0} состоит только из букв’).format(s.encode(‘cp1251’))  **else**:  **print** str(‘Строка {0} состоит не только из букв(букв может не быть)’).format(s.encode(‘cp1251’)) |

* 1. Морфологический анализ слова

После того, как мы выделили токены, которые, предположительно, содержат слова (смысловые слова), мы подаем их на вход анализатора. Функция, анализирующая слово, возвращает грамматическую информацию о нем и его нормальную форму. Работает функция по словарю, для неизвестных слов работает предсказатель, если возможных форм несколько - возвращает несколько форм.

Прототип этой функции:

**get\_graminfo**(word, standard=False, predict=True, \*\*kwargs)

Есть два ограничения на работу этой функции:

* символы слова должны находиться в формате *Unicode*
* символы слова должны быть *в верхнем регистре* (заглавными)

Слово в *Юникоде* сразу после ввода. Прототип ф-ции, возводящий буквы в верхний регистр: s.**upper**(), где *s* – строка.

Пример:

|  |
| --- |
| **from** pymorphy **import** get\_morph *# портируем модули*  **from** pymorphy.contrib **import** tokenizers  morph = get\_morph('C:/Python27/ru.sqlite-json/') *# создаем объект класса pymorph*  text = (raw\_input("Входные данные():  ")).decode('cp1251')  **print** "Выходные данные():",  **for** token **in** tokenizers.extract\_tokens(text): *# выделяем токены*  **if** word.isalpha() == True: *# отсеиваем пунктуацию*        info = morph.get\_graminfo(word.upper()) *# анализ* |

* 1. Проверка на принадлежность к нужной части речи

Синонимизация проводится не для всех слов, а только для существительных, глаголов и прилагательных.

Пример:

|  |
| --- |
| text = (raw\_input("Входные данные:  ")).decode('cp1251')  **for** word **in** tokenizers.extract\_tokens(text): *# выделяем токены, узнаем часть речи*  **if** word.isalpha() == True: *# отсеиваем пунктуацию*          info = morph.get\_graminfo(word.upper()) *# находим грамм. информацию слова*          part = info[0]['class'] *# часть речи слова*  *# учитываем только сущ, глаголы и прил*  **if** part == u'С' **or** part == u'Г' **or** part == u'П' **or** part == u'ИНФИНИТИВ': |

* 1. Проверка на то, что слово не является именем

Над именами синонимизация не проводится.

Пример:

|  |
| --- |
| text = (raw\_input("Входные данные:  ")).decode('cp1251')  **for** word **in** tokenizers.extract\_tokens(text): *# выделяем токены, узнаем часть речи*  **if** word.isalpha() == True: *# отсеиваем пунктуацию*          info = morph.get\_graminfo(word.upper()) *# находим грамм. информацию слова*          part = info[0]['class'] *# часть речи слова*          gram\_info = info[0]['info'] *# грамм. информация слова*  *# учитываем только сущ, глаголы и прил*  **if** part == u'С' **or** part == u'Г' **or** part == u'П' **or** part == u'ИНФИНИТИВ':  *# для имени не подбираем синоним*  **if** u'имя' **in** info[0]['info']:name = 1  **else**:  *# ищем нормальную форму* |

* 1. Нахождение нормальной формы слова

После того, как мы убедились, что слово нужной нам части речи, и что оно не является именем, находим нормальную форму.

Пример:

|  |
| --- |
| nf = info[0]['norm'] *# нормальная форма слова* |

* 1. Сохранение связи нормальной формы и слова

После того, как мы нашли нормальную форму слова, необходимо запомнить его. В данном случае лучшим способом хранения связи нормальной формы и слова является тип данных словарь(см. ПРИЛОЖЕНИЕ 2). Ключом будет являться нормальная форма, а значением список слов, которые принадлежат нормальной форме. Для того, чтобы не затирать старые значения словаря, схожие ключи не должны вставляться. Отследить, что такая нормальная форма уже используется может тип данных множество(см. ПРИЛОЖЕНИЕ 2), который исключает возможность хранения одинаковых элементов.

Пример:

|  |
| --- |
| text = (raw\_input("Входные данные:  ")).decode('cp1251')  d = {} *# словарь для хранения списка слов с одной нормальной формой*  a = set() *# множество, не допускающее повторений нормальных форм*  **for** word **in** tokenizers.extract\_tokens(text): *# выделяем токены, узнаем часть речи*  **if** word.isalpha() == True: *# отсеиваем пунктуацию*          info = morph.get\_graminfo(word.upper())          nf = info[0]['norm'] *# нормальная форма слова*          len\_ = len(a)          a.add(nf)  **if** len\_ == len(a): *# есть ли уже такая нормальная форма?*          l = d[nf]      *# если есть, то добавляем слово в конец списка*          l.append(word)          d[nf] = l |

* + 1. Если нормальная форма уже есть в словаре, просто приписываем в конец списка значения слово.

Пример:

|  |
| --- |
| len\_ = len(a)          a.add(nf)  **if** len\_ == len(a): *# есть ли уже такая нормальная форма?*          l = d[nf]      *# если есть, то добавляем слово в конец списка*          l.append(word)          d[nf] = l |

* + 1. Если нормальной формы нет в словаре, то создаем новую связь

Пример:

|  |
| --- |
| l = [word]  d[nf] = l |

* 1. Сохраняем связь слова с его грамматической формой

Пример:

|  |
| --- |
| len\_ = len(part\_a)  part\_a.add(word)  **if** len\_ != len(part\_a): *# соотносим слово и его грам. инфо*      part\_d[word] = gram\_info |

1. Загрузка словаря синонимов

Считываем словарь синонимов из текстового файла построчно, каждую строку заносим в список.

Прототип функции:

file.**readlines**([sizehint])

Пример:

|  |
| --- |
| t = open('abr2w.txt','r+').readlines() |

1. Нахождение синонимов нормальных форм, содержание слов в которых > 1
   1. Если слов > 1, парсим словарь и находим синоним

Пример:

|  |
| --- |
| **for** key **in** d.keys(): *# проссматриваем все ключи словаря нормальных форм*  **if** len(d[key]) > 1: *# если слов в списке больше одного, то синонимизируем*  **for** s **in** t: *# проходим по строкам текста*  *# парсим строку*              s = str(s)              key\_ = key.encode('cp1251')  **if** s.upper().find(key\_) == 0:  **if** s.upper().find('СМ.') != -1:                      index\_s = s.upper().find('СМ.')  **if** s.upper().find(' ',index\_s+1) != -1:                          index\_f = s.upper().find(' ', index\_s+4)                          synonym = s[index\_s+4:index\_f]  *# и т.д.* |

* 1. Связываем синоним с словами

Синонимы необходимо связать со словами, которые были привязаны к нормальным формам.

Пример:

|  |
| --- |
| l = d[key]  d\_synonym[synonym] = l |

1. Замена слов на синонимы

После того, как мы нашли синонимы, их нужно вставить в текст вместо первоначальных слов.

* 1. Нахождение позиции слова в пердложении

Поиск слова в предложении - это поиск подстроки в строке. Прототип функции, осущ. поиск подстроки:

str.**find**(sub[, start[, end]])

Пример:

|  |
| --- |
| start = text.find(value[i].encode('cp1251')) |

* 1. Удаление слова из предложение

Питон предоставляет очень хорошее средство для таких операций – срезы.

От начала строки до стартовой позиции(позиции начала слова) делаем срез. К этому срезу прибавляем строку от стартовой позиции, прибавленное к длине слова.

Пример:

|  |
| --- |
| text = text[:start] + text[start+len(value[i]):] |

* 1. Преобразование синонима в нужную форму

Синоним находится в нормальной форме. Если мы сразу подставим его в слово, предложение будет выглядеть некрасиво.

Прототип функции, которая преобразует слово в требуемую форму:

**inflect\_ru**(word, gram\_form, gram\_class=None)

Пример:

|  |
| --- |
| sub = morph.inflect\_ru((key).decode('cp1251').upper(),part\_d[value[i]]) |

* 1. Вставка синонима в текст

Используем конкатенацию строк.

Пример:

|  |
| --- |
| text = text[:start] + sub.encode('cp1251') + text[start:] |

1. Вывод предложения

Программа выполнила свою работу, заменив повторяющиеся слова на синонимы, выведем выходную строку.

Прототип функции вывода текста:

**print** expression

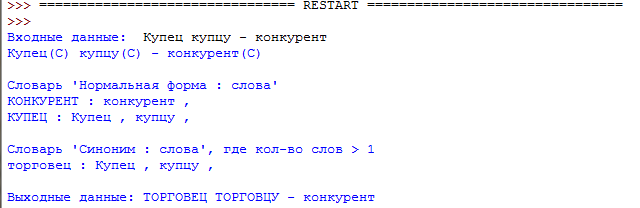
Примеры

Пример №1

Предложение, в котором два раза повторяется одно и тоже существительное.

Входные данные: Купец купцу - конкурент

Выходные данные: ТОРГОВЕЦ ТОРГОВЦУ – конкурент

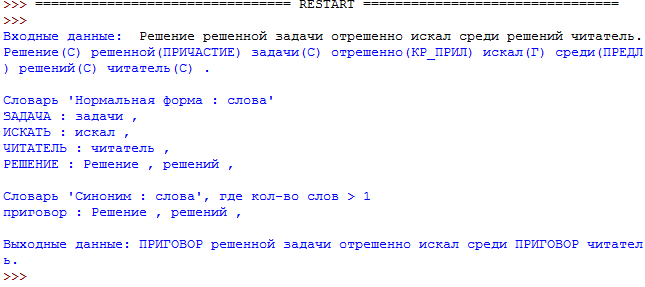


Пример №2

Предложение со словами, содержащими одинаковые корни, но разные части речи.

Входные данные: Решение решенной задачи отрешенно искал среди решений читатель.

Выходные данные: ПРИГОВОР решенной задачи отрешенно искал среди ПРИГОВОР читатель.



Пример №3

Предложение посложней, чем из первого примера.

Входные данные: Из темного леса навстречу ему

Идет вдохновенный кудесник,

...

"Скажи мне, кудесник, любимец богов,

Что сбудется в жизни со мною?"

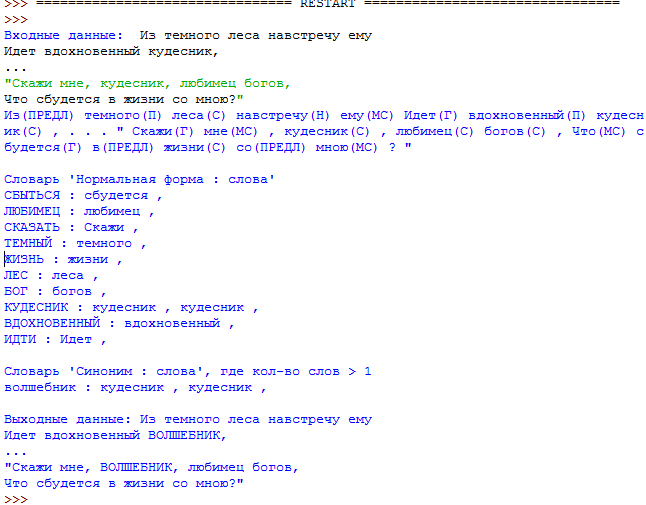
Выходные данные: Из темного леса навстречу ему

Идет вдохновенный ВОЛШЕБНИК,

...

"Скажи мне, ВОЛШЕБНИК, любимец богов,

Что сбудется в жизни со мною?"

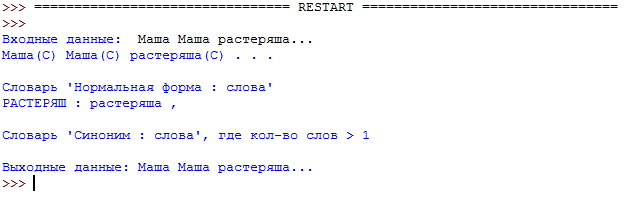


Пример №4.

Реакция программы на ввод имен. Имена не должны синонимизироваться.

Входные данные: Маша Маша растеряша...

Выходные данные: Маша Маша растеряша...

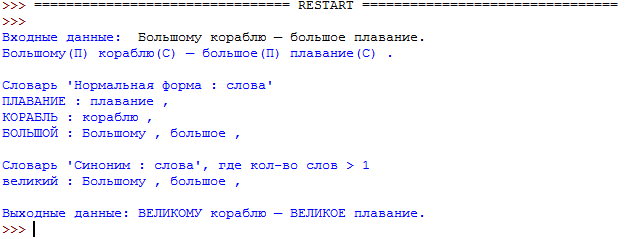


Пример №5

Предложение, состоящее из нескольких прилагательных.

Входные данные: Большому кораблю — большое плавание.

Выходные данные: ВЕЛИКОМУ кораблю — ВЕЛИКОЕ плавание.

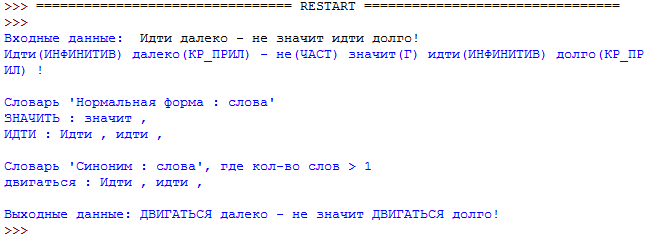


Пример №6

Предложение, состоящие из нескольких глаголов.

Входные данные: Идти далеко - не значит идти долго!

Выходные данные: ДВИГАТЬСЯ далеко - не значит ДВИГАТЬСЯ долго!

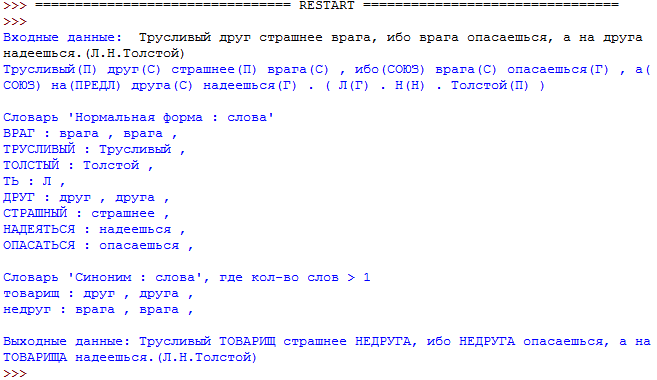


Пример №7

Иллюстрируется случай, когда синонимизируются больше одного слова.

Входные данные: Трусливый друг страшнее врага, ибо врага опасаешься, а на друга надеешься.(Л.Н.Толстой)

Выходные данные: Трусливый ТОВАРИЩ страшнее НЕДРУГА, ибо НЕДРУГА опасаешься, а на ТОВАРИЩА надеешься.(Л.Н.Толстой)

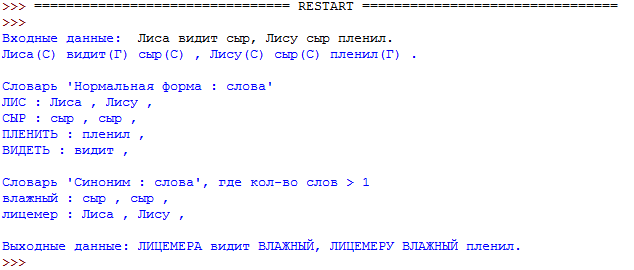


Пример №8

Случай, получающийся из-за того, что нет смыслового разбора предложения.

Входные данные: Лиса видит сыр, Лису сыр пленил.

Выходные данные: ЛИЦЕМЕРА видит ВЛАЖНЫЙ, ЛИЦЕМЕРУ ВЛАЖНЫЙ пленил.



Код

|  |
| --- |
| *# -\*- coding: cp1251 -\*-*  **from** pymorphy **import** get\_morph *# портируем необходимые модули*  **from** pymorphy.contrib **import** tokenizers    morph = get\_morph('C:/Python27/ru.sqlite-json/') *# создаем объект класса pymorphy.Morph*    text = (raw\_input("Входные данные:  ")).decode('cp1251')    d = {} *# словарь для хранения списка слов с одной нормальной формой*  a = set() *# множество, не допускающее повторений нормальных форм*  part\_d = {} *# словарь для грамматической информации слова*  part\_a  = set() *# множество, не допускающее повторений слов*    **for** word **in** tokenizers.extract\_tokens(text): *# выделяем токены, узнаем часть речи*  **if** word.isalpha() == True: *# отсеиваем пунктуацию*          info = morph.get\_graminfo(word.upper()) *# находим грамм. информацию слова*          part = info[0]['class'] *# часть речи слова*          gram\_info = info[0]['info'] *# грамм. информация слова*          nf = info[0]['norm'] *# нормальная форма слова*  **print** str('{0}({1})').format(word.encode('cp1251'),part.encode('cp1251')),  *# учитываем только сущ, глаголы и прил*  **if** part == u'С' **or** part == u'Г' **or** part == u'П' **or** part == u'ИНФИНИТИВ':  *# для имени не подбираем синоним*  **if** u'имя' **in** info[0]['info']:name = 1  **else**:                  len\_ = len(a)                  a.add(nf)  **if** len\_ == len(a): *# есть ли уже такая нормальная форма?*                      l = d[nf]      *# если есть, то добавляем слово в конец списка*                      l.append(word)                      d[nf] = l  **else**:              *# иначе создаем ключ со значением списка*                      l = [word]                      d[nf] = l                    len\_ = len(part\_a)                  part\_a.add(word)  **if** len\_ != len(part\_a): *# соотносим слово и его грам. инфо*                      part\_d[word] = gram\_info  **elif** word.isspace() == False:  **print** word,    *# Выводим словарь:*  **print** "**\n\n**Словарь 'Нормальная форма : слова'"  **for** key, value **in** d.items():  **print** key, ":",  **for** i **in** range(len(value)):  **print** value[i], ',',  **print**    t = open('abr2w.txt','r+').readlines() *# считываем словарь построчно в список*    d\_synonym = {} *# словарь, соотносящий синонимы со списком слов*  a\_synonym = set() *# множество, не пропускающее одинаковые синонимы*    **for** key **in** d.keys(): *# проссматриваем все ключи словаря нормальных форм*  **if** len(d[key]) > 1: *# если слов в списке больше одного, то синонимизируем*  **for** s **in** t: *# проходим по строкам текста*  *# парсим строку*              s = str(s)              key\_ = key.encode('cp1251')  **if** s.upper().find(key\_) == 0:  **if** s.upper().find('СМ.') != -1:                      index\_s = s.upper().find('СМ.')  **if** s.upper().find(' ',index\_s+1) != -1:                          index\_f = s.upper().find(' ', index\_s+4)                          synonym = s[index\_s+4:index\_f]  **elif** s.upper().find(',',index\_s+1) != -1:                          index\_f = s.upper().find(',', index\_s+4)                          synonym = s[index\_s+4:index\_f]  **else**:                          synonym = s[:index\_s+3]  **elif** s.upper().find(', ') != -1:                      index\_s = s.upper().find(', ')                      index\_f = s.upper().find(' ', index\_s+2)                      synonym = s[index\_s+2:index\_f]  **else**:                      synonym = 'синоним не найден'                  synonym = synonym.replace(',', '')                  synonym = synonym.replace('(', '')                  synonym = synonym.replace(')', '')  *# теперь слова принадлежат не нормальной форме, а синониму*                  l = d[key]                  d\_synonym[synonym] = l  **break**    *# Выводим словарь:*  **print** "**\n**Словарь 'Синоним : слова', где кол-во слов > 1"  **for** key, value **in** d\_synonym.items():  **print** key, ":",  **for** i **in** range(len(value)):  **print** value[i], ',',  **print**    text = text.encode('cp1251') *# перегоняем входной текст из юникода в ascii*    **for** key, value **in** d\_synonym.items(): *# получаем ключ и значение по словарю синонимов*  **for** i **in** range(len(value)): *# получаем слова из списка*  *# находим слово и заменяем его на синоним, представленный в той же грам. форме*          start = text.find(value[i].encode('cp1251'))          text = text[:start] + text[start+len(value[i]):]          sub = morph.inflect\_ru((key).decode('cp1251').upper(),part\_d[value[i]])          text = text[:start] + sub.encode('cp1251') + text[start:]    **print** "**\n**Выходные данные:", text |

ПРИЛОЖЕНИЕ 1

Коротко о выбранных средствах:

**Python** ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *python* — [питон](http://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%82%D0%BE%D0%BD); в русском языке распространено название *пито́н*)—[высокоуровневый язык программирования](http://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%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%D1%8F) общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. [Синтаксис](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81_(%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)) ядра Python минималистичен. В то же время [стандартная библиотека](http://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_Python) включает большой объём полезных функций.

Python поддерживает несколько [парадигм программирования](http://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%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%D1%8F), в том числе [структурное](http://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), [объектно-ориентированное](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%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), [функциональное](http://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%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),[императивное](http://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%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) и [аспектно-ориентированное](http://ru.wikipedia.org/wiki/%D0%90%D1%81%D0%BF%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%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). Основные архитектурные черты — [динамическая типизация](http://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [автоматическое управление памятью](http://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0), полная [интроспекция](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D1%80%D0%BE%D1%81%D0%BF%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_(%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)), механизм [обработки исключений](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9), поддержка [многопоточных вычислений](http://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D0%BE%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C) и удобные высокоуровневые [структуры данных](http://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Код в Питоне организовывается в функции и [классы](http://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%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)), которые могут объединяться в [модули](http://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_(%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)) (которые в свою очередь могут быть объединены в пакеты).

(Википедия, <http://ru.wikipedia.org/wiki/Python> )

**pymorphy** - библиотека для морфологического анализа на Python, распространяется по лицензии MIT.

За основу были взяты наработки с сайта aot.ru. Словари (LGPL) для русского и английского, а также идеи - оттуда.

На aot.ru описаны и конкретные алгоритмы реализации, но в терминах теории автоматов. Реализация в pymorphy независимая, не использует конечные автоматы, данные хранятся в key-value хранилище (поддерживаются разные варианты), все алгоритмы переписаны с учетом этого факта.

В pymorphy также есть некоторые возможности, отсутствующие в оригинальной реализации, например:

* поддерживается разбор слов с дефисами, разбор слов со сложными префиксами
* реализовано склонение слов, постановка слов во множественное число

([http://pythonhosted.org/pymorphy/index.html#](http://pythonhosted.org/pymorphy/index.html) )

**IDLE** — это [интегрированная среда разработки](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) на языке [Python](http://ru.wikipedia.org/wiki/Python), созданная с помощью библиотеки [Tkinter](http://ru.wikipedia.org/wiki/Tkinter). Поставляется вместе с Python.

(Википедия, <http://ru.wikipedia.org/wiki/IDLE> )

ПРИЛОЖЕНИЕ 2

Тип данных словарь (dict) :

Словарь — это неупорядоченное множество пар ключ—значение. Когда вы добавляете ключ в словарь, вы также должны добавить и значение для этого ключа. (Значение всегда можно изменить позже.) Словари в Python оптимизированы для получения значения по известному ключу, но не для других целей.

Создание словаря:

|  |
| --- |
| d = {} |

или

|  |
| --- |
| d = {a : a \*\* 2 **for** a **in** range(7)}  >>> d  {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36} |

Подробней <http://pythonworld.ru/tipy-dannyx-v-python/slovari-dict-funkcii-i-metody-slovarej.html>

Тип данных множество (set):

Множество — это «мешок», содержащий неупорядоченные уникальные значения. Одно множество может содержать значения любых типов. Если у вас есть два множества, вы можете совершать над ними любые стандартные операции, например, объединение, пересечение и разность.

Создание множества:

|  |
| --- |
| a = set() |

или

|  |
| --- |
| a = {i \*\* 2 **for** i **in** range(10)} *# генератор множеств*  >>> a  {0, 1, 4, 81, 64, 9, 16, 49, 25, 36} |

Подробней <http://pythonworld.ru/tipy-dannyx-v-python/mnozhestva-set-i-frozenset.html>