Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Реферат

по курсу «Практикум на ЭВМ»

II семестр

«Интерпретируемые языки программирования на примере python»

Студент	**Сыроежкин Кирилл Геннадьевич**
Группа	**М8О-104Б-18**
Руководитель	**Доцент кафедры 806 Никулин С.П**
Оценка	
Дата	

Москва 2019.

Вступление

Мы полагаемся на такие инструменты, как компиляция и интерпретация, чтобы преобразовать наш код в форму, понятную компьютеру. Код может быть исполнен нативно, в операционной системе после конвертации в машинный (путём компиляции) или же исполняться построчно другой программой, которая делает это вместо ОС (интерпретатор).

Компилируемый язык — это такой язык, что программа, будучи скомпилированной, содержит инструкции целевой машины; этот машинный код непонятен людям. Интерпретируемый же язык — это такой, в котором инструкции не исполняются целевой машиной, а считываются и исполняются другой программой (которая обычно написана на языке целевой машины).

Прежде чем мы продолжим, стоит отметить, что многие языки программирования имеют как компилируемую, так и интерпретируемую версии, поэтому классифицировать их затруднительно. Тем не менее, чтобы не усложнять, в дальнейшем я буду разделять компилируемые и интерпретируемые языки.

Содержание

1.Интерпретируемость

Интерпретируемый язык программирования — язык программирования, исходный код на котором выполняется методом интерпретации. Классифицируя языки программирования по способу исполнения, к группе интерпретируемых относят языки, в которых операторы программы друг за другом отдельно транслируются и сразу выполняются (интерпретируются) с помощью специальной программы-интерпретатора (что противопоставляется компилируемым языкам, в которых все операторы программы заранее оттранслированы в объектный код). Такой язык может реализовывать конструкции, позволяющие динамические изменения на этапе времени выполнения (модификация существовавших или создание новых подпрограмм). Эти конструкции затрудняют компиляцию и трансляцию на компилируемый язык.

В общем случае, любой язык может быть компилируемым и интерпретируемым. В предельном случае такой язык можно реализовать только при помощи интерпретаторов. Также встречается название interpretative language - интерпретативный язык, interpretable (букв. поддающийся интерпретации) programming language - интерпретируемый язык программирования и interpreted language - интерпретируемый язык.

При этом для многих языков существует различие в производительности между компилируемой и интерпретируемой реализацией.

1.1. Краткая историческая справка

В ранние годы развития программирования на языки сильно влиял выбор способа выполнения. Например, компилируемые языки требовали задания типа данных переменной в момент её описания или первого использования. В то время как интерпретируемые языки в силу своей динамической природы позволяли отказаться от этого требования, что давало больше гибкости и ускоряло разработку.

Изначально интерпретируемые языки преобразовывались в машинный код построчно, то есть каждая логическая строка компилировалась непосредственно перед выполнением. В результате каждая инструкция,

заключенная в тело цикла и исполняемая несколько раз, столько же раз обрабатывалась транслятором. В настоящее время такие эффекты редки. Большинство интерпретируемых языков предварительно транслируются в промежуточное представление. Оно представляет собой байт-код или шитый код (threaded code). Это набор инструкций по вызову небольших фрагментов более низкоуровневого кода, эквивалентный нескольким командам ассемблера или командам виртуальной машины соответственно. Уже этот код исполняется интерпретатором или виртуальной машиной.

Например, такую схему используют следующие языки:

- Java
- Python
- Ruby (использует представление кода в виде абстрактного синтаксического дерева)

Промежуточный код может создаваться как явной процедурой компиляции всего проекта (Java), так и скрытой трансляцией каждый раз перед началом выполнения программы (Perl, Ruby) и при изменении исходного кода (Python).

1.2. Преимущества

Есть ряд возможностей, которые значительно легче реализовать в интерпретаторе, чем в компиляторе:

1. Кроссплатформенность.
2. Рефлексия и интроспекция.
3. Динамическая типизация.
4. Использование динамической области видимости и замыканий.
5. Пошаговое отслеживание выполнения программы.
6. Модификация программы во время исполнения.
7. Меньшие затраты времени на разработку и отладку.
8. Простой способ создания переносимых программ.
9. Не требует затрат на компиляцию небольших программ.

Кроме того, принципы и стиль программирования часто не требуют создания и описания специальных конструкций, оформляющих программу (манифестов, классов, типов данных). Это позволяет разрабатывать и тестировать код постепенно, что удобно как для написания небольших программ, так и для изолированной разработки модулей для сложных

систем. В силу своей универсальности их удобно применять в качестве скриптовых языков.

1.3. Недостатки

Основным недостатком является более медленное выполнение программы по сравнению с выполнением программы, предварительно скомпилированной в машинный код. Например выполнение PHP и Python может оказаться в более чем 100 раз медленнее чем C++. Трансляция в байт-код и JIT-компиляция не решают этой проблемы полностью. Дополнительный слой интерпретатора или виртуальной машины замедляет выполнение программы и может требовать больше ресурсов. Во время выполнения интерпретатор всегда должен быть загружен в память (а это могут быть и большие программы, как браузер для JS или Office для VBA). Комментарии могут снижать производительность и для обхода этого создают две версии кода - готовую для использования (с удалёнными комментариями) и разрабатываемую.

В результате в среднем интерпретируемый код следует тестировать тщательнее компилируемого, строже придерживаться соглашений по оформлению программ и использовать дополнительные анализаторы качества кода. Последний недостаток выражен несильно, так как при серьезной разработке на компилируемых языках также необходимо применение этих средств.

2. Python

Хорошим примером интерпретируемого языка python - это простой в освоении и мощный язык программирования. Он предоставляет эффективные высокоуровневые структуры данных, а также простой, но эффективный подход к объектно-ориентированному программированию. Его элегантный синтаксис и динамическая типизация наряду с тем, что он является интерпретируемым, делают его идеальным языком для написания сценариев и быстрой разработки приложений в различных областях и на большинстве платформ.

2.1. История

История языка программирования Python началась в конце 1980-х. Гвидо ван Россум задумал Python в 1980-х годах, а приступил к его созданию в декабре 1989 года в центре математики и информатики в Нидерландах. Язык Python был задуман как потомок языка программирования ABC, способный к обработке исключений и взаимодействию с операционной

системой Амёба. Ван Россум является основным автором Python и продолжал выполнять центральную роль в принятии решений относительно развития языка вплоть до 12 июля 2018 года.

Версия Python 2.0 была выпущена 16 октября 2000 года и включала в себя много новых крупных функций — таких как полный сборщик мусора и поддержка Unicode. Однако наиболее важным из всех изменений было изменение самого процесса развития языка и переход на более прозрачный процесс его создания.

Первая обратно-несовместимая версия Python 3.0 была выпущена 3 декабря 2008 года после длительного периода тестирования. Многие её функции были портированы в обратно совместимые Python 2.6 и Python 2.7.

Появившись сравнительно поздно, Python создавался под влиянием множества языков программирования:

- ABC — отступы для группировки операторов, высокоуровневые структуры данных (map) (Python фактически создавался как попытка исправить ошибки, допущенные при проектировании ABC);
- Modula-3 — пакеты, модули, использование else совместно с try и except, именованные аргументы функций (на это также повлиял Common Lisp);
- C, C++ — некоторые синтаксические конструкции (как пишет сам Гвидо ван Россум — он использовал наиболее непротиворечивые конструкции из C, чтобы не вызвать неприязнь у C-программистов к Python);
- Smalltalk — объектно-ориентированное программирование;
- Lisp — отдельные черты функционального программирования (lambda, map, reduce, filter и другие);
- Fortran — срезы массивов, комплексная арифметика;
- Miranda — списочные выражения;
- Java — модули logging, unittest, threading (часть возможностей оригинального модуля не реализована), xml.sax стандартной библиотеки, совместное использование finally и except при обработке исключений, использование @ для декораторов;
- Icon — генераторы.

Большая часть других возможностей Python (например, байт-компиляция исходного кода) также была реализована ранее в других языках.

2.2. Особенности Python

2.2.1. Простой

Python – простой и минималистичный язык. Чтение хорошей программы на Python очень напоминает чтение английского текста, хотя и достаточно строгого! Такая псевдокодовая природа Python является одной из его самых сильных сторон. Она позволяет вам сосредоточиться на решении задачи, а не на самом языке.

2.2.2. Свободный и открытый

Python – это пример свободного и открытого программного обеспечения – FLOSS (Free/Libré and Open Source Software). Проще говоря, вы имеете право свободно распространять копии этого программного обеспечения, читать его исходные тексты, вносить изменения, а также использовать его части в своих программах. В основе свободного ПО лежит идея сообщества, которое делится своими знаниями. Это одна из причин, по которым Python так хорош: он был создан и постоянно улучшается сообществом, которое просто хочет сделать его лучше.

2.2.3. Язык высокого уровня

При написании программына Python вам никогда не придётся отвлекаться на такие низкоуровневые детали, как управление памятью, используемой вашей программой, и т.п.

2.2.4. Портируемый

Благодаря своей открытой природе, Python был портирован на много платформ (т.е. изменён таким образом, чтобы работать на них). Все ваши программы смогут запускатьсяна любой из этих платформ без каких-либо изменений, если только вы избегали использования системно-зависимых функций. Python можно использовать в GNU/Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga,AROS, AS/400, BeOS, OS/390, z/OS, PalmOS,QNX, VMS, Psion, Acorn RISCOS, VxWorks, PlayStation, Sharp Zaurus, Windows CE и даже на PocketPC! Вы можете даже использовать такую платформу, как Kivy для создания игр для iOS (iPhone, iPad) и Android.

2.2.5. Объектно-ориентированный

Python поддерживает объектно-ориентированное программирование. В процедурно ориентированных языках программы строятся на основе процедур или функций, которые представляют собой просто-напросто многократно используемые фрагменты программы. В объектно-

ориентированных языках программирования программы строятся на основе объектов, объединяющих в себе данные и функционал. Python предоставляет простые, но мощные средства для ООП, особенно в сравнении с такими большими языками программирования, как C++ или Java.

2.2.6. Расширяемый

Если вам нужно, чтобы некоторая критическая часть программы работала очень быстро или вы вынуждены скрыть часть алгоритма, вы можете написать эту часть программы на C или C++, а затем вызывать её из программы на Python.

2.2.7 Встраиваемый

Python можно встраивать в программы на C/C++, чтобы предоставлять возможности написания сценариев их пользователям.

2.2.8 Обширные библиотеки

Стандартная библиотека Python просто огромна. Она может помочь в решении самых разнообразных задач, связанных с использованием регулярных выражений, генерированием документации, проверкой блоков кода, распараллеливанием процессов, базами данных, веб-браузерами, CGI, FTP, электронной почтой, XML, XML-RPC, HTML, WAV файлами, криптографией, GUI (графическим интерфейсом пользователя) и другими системно-зависимыми вещами. Помните, что всё это доступно абсолютно везде, где установлен Python. В этом заключается философия Python “Всё включено”. Кроме стандартной библиотеки, существует множество других высококачественных библиотек, которые можно найти в Каталоге пакетов Python

2.3. Основной синтаксис

2.3.1. Общая структура

Во-первых, стоит отметить интересную особенность Python. Он не содержит операторных скобок (begin..end в pascal или {..}в Си), вместо этого блоки выделяются отступами: пробелами или табуляцией, а вход в блок из операторов осуществляется двоеточием. Однострочные комментарии начинаются со знака фунта «#», многострочные — начинаются и заканчиваются тремя двойными кавычками «"""».
Чтобы присвоить значение пременной используется знак «=», а для сравнения — «==». Для увеличения значения переменной, или

добавления к строке используется оператор «+=», а для уменьшения — «-=». Все эти операции могут взаимодействовать с большинством типов, в том числе со строками. Например

```
>>> 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
```

2.3.2. Структуры данных

Python содержит такие структуры данных как списки (lists), кортежи (tuples) и словари (dictionaries). Списки — похожи на одномерные массивы (но вы можете использовать Список включающий списки — многомерный массив), кортежи — неизменяемые списки, словари — тоже списки, но индексы могут быть любого типа, а не только числовыми. "Массивы" в Python могут содержать данные любого типа, то есть в одном массиве может могут находиться числовые, строковые и другие типы данных. Массивы начинаются с индекса 0, а последний элемент можно получить по индексу -1 Вы можете присваивать переменным функции и использовать их соответственно.

```
>>> 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
>>> myfunction = len #Python позволяет таким образом объявлять синонимы функции
>>> print myfunction(list)
3
```

Вы можете 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]
```

2.3.3. Строки

Строки в Python обособляются кавычками двойными «"» или одинарными «'». Внутри двойных ковычек могут присутствовать одинарные или наоборот. К примеру строка «Он сказал 'привет'!» будет выведена на экран как «Он сказал 'привет'!». Если нужно использовать строку из несколько строчек, то эту строку надо начинать и заканчивать тремя двойными кавычками «"""». Вы можете подставить в шаблон строки элементы из кортежа или словаря. Знак процента «%» между строкой и кортежем, заменяет в строке символы «%s» на элемент кортежа. Словари позволяют вставлять в строку элемент под заданным индексом. Для этого надо использовать в строке конструкцию «%(индекс)s». В этом случае вместо «%(индекс)s» будет подставлено значение словаря под заданным индексом.

```
>>>print «Name: %s\nNumber: %s\nString: %s» % (myclass.name, 3, 3 * "-")
Name: Poromenos
Number: 3
String: —
strString = """«Этот текст расположен
на нескольких строках»"""

>>> print «This %(verb)s a %(noun)s.» % {«noun»: «test», «verb»: «is»}
This is a test.
```

2.3.4. Операторы

Операторы while, if, for составляют операторы перемещения. Здесь нет аналога оператора select, так что придется обходиться if. В операторе for происходит сравнение переменной и списка. Чтобы получить список цифр до числа <number> — используйте функцию range(<number>). Вот пример использования операторов

```
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
```

2.3.5. 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 — необязательные 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
```

2.3.6. Классы

Язык Python ограничен в множественном наследовании в классах. Внутренние переменные и внутренние методы классов начинаются с двух знаков нижнего подчеркивания «__» (например «__myprivatevar»). Мы можем также присвоить значение переменной класса извне.

2.3.7. Исключения

Исключения в Python имеют структуру try-except [exceptionname]:

```
def somefunction():
try:
# Деление на ноль вызывает ошибку
10 / 0
except ZeroDivisionError:
# Но программа не "Выполняет недопустимую операцию"
# А обрабатывает блок исключения соответствующий ошибке «ZeroDivisionError»
print «Oops, invalid.»

>>> fnexcept()
Oops, invalid.
```

2.3.8. 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 random #Импортируем библиотеку «random»
from time import clock #И заодно функцию «clock» из библиотеки «time»

randomint = random.randint(1, 100)
>>> print randomint
64
```

2.3.9. Работа с файловой системой

Python имеет много встроенных библиотек. В этом примере мы попробуем сохранить в бинарном файле структуру списка, прочитать ее и сохраним строку в текстовом файле. Для преобразования структуры данных мы будем использовать стандартную библиотеку «pickle»

```
import pickle
mylist = [«This», «is», 4, 13327]
# Откроем файл C:\binary.dat для записи. Символ «r»
# предотвращает замену специальных сиволов (таких как \n, \t, \b и др.).
myfile = file(r«C:\binary.dat», «w»)
pickle.dump(mylist, myfile)
myfile.close()

myfile = file(r«C:\text.txt», «w»)
myfile.write(«This is a sample string»)
myfile.close()

myfile = file(r«C:\text.txt»)
>>> print myfile.read()
'This is a sample string'
myfile.close()

# Открываем файл для чтения
myfile = file(r«C:\binary.dat»)
loadedlist = pickle.load(myfile)
myfile.close()
>>> print loadedlist
['This', 'is', 4, 13327]
```

2.3.10. Особенности

- Условия могут комбинироваться. 1 < a < 3 выполняется тогда, когда a 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.

- Глобальные переменные объявляются вне функций и могут быть прочитанны без каких либо объявлений. Но если вам необходимо изменить значение глобальной переменной из функции, то вам необходимо объявить ее в начале функции ключевым словом «global», если вы этого не сделаете, то Python объявит переменную, доступную только для этой функции.

2.4. Основные применения

2.4.1. Веб-разработка

Фреймворки, основанные на Python, такие как Django и Flask, в последнее время приобрели широкую популярность среди веб-разработчиков. Эти фреймворки позволяют создавать серверный код (backend-код) на Python, который выполняется на сервере, в отличие от frontend-кода, исполняемого на пользовательских устройствах и в браузерах.

2.4.2. Обработка данных (включая машинное обучение, анализ и визуализацию данных)

Прежде всего, следует разобраться, что такое машинное обучение.

Предположим, что вы хотите разработать программу, которая будет автоматически определять, что изображено на картинке.

Например, предлагая ей изображение, вы хотите, чтобы программа опознала собаку или стол. Возможно, вы думаете, что для решения этой задачи можно просто написать код анализа изображения. Например, если на картинке много светло-коричневых пикселей, делаем вывод, что это собака.

Или вы можете научиться определять на изображении края и границы. Тогда картинка с большим количеством прямых границ, вероятно, окажется столом.

Однако это довольно сложный и непродуманный подход. Что делать, если на фотографии изображена белая собака без коричневых пятен? Или если на картинке круглый стол?

Здесь вступает в игру машинное обучение. Обычно оно реализует некоторый алгоритм, который позволяет автоматически обнаруживать знакомый шаблон среди входных данных.

Вы можете предложить алгоритму машинного обучения, скажем, 1000 изображений собаки и 1000 снимков столов. Он выучит разницу между этими объектами. Затем, когда вы дадите ему новую картинку со столом или собакой, он сможет определить, что именно на ней изображено.

Это очень похоже на то, как учатся маленькие дети. Каким именно образом они узнают, что одна вещь похожа на стол, а другая – на собаку? Из большого количества примеров.

Вы ведь не даете ребенку четкую инструкцию: «Если нечто пушистое и светло-каштановое, значит, это собака». Напротив, вы говорите: «Это собака. Это тоже собака. И это. А это стол. И это тоже стол».

Алгоритмы машинного обучения в основном работают сходным образом.

Эта технология может применяться:

- в рекомендательных сервисах (вспомните, например, YouTube, Amazon и Netflix);
- в системах распознавания лиц и голосов.

Среди самых популярных алгоритмов машинного обучения, можно выделить:

- нейронные сети;
- глубокое обучение;
- метод опорных векторов;
- random forest.

Любой из вышеперечисленных алгоритмов может быть использован для решения задачи с собаками и столами на изображениях.

2.4.3. Написание скриптов

Обычно под этим понимают создание небольших программ для автоматизации простых задач. Например, компании используют различные системы поддержки клиентов по электронной почте. Чтобы анализировать полученные сообщения, компаниям нужно подсчитать, какой их количество содержит определённые ключевые слова.

Это можно либо делать вручную, либо написать незамысловатую программу (скрипт) для автоматической обработки сообщений. Для подобных задач отлично подходит Python, главным образом благодаря относительно простому синтаксису и потому, что на нём можно легко и быстро писать и тестировать небольшие проекты.

Заключение

Итак, программа, написанная на компилируемом языке программирования, как например, C или C++, преобразуется из исходного языка (т.е. C или C++) в язык, понятный компьютеру (бинарный код, т.е. нули и единицы) при помощи компилятора с применением разнообразных флагов и параметров. Когда вы запускаете такую программу, компоновщик/загрузчик копирует программу с диска в оперативную память и запускает её. Python же, напротив, не требует компиляции в бинарный код. Программа просто выполняется из исходного текста. Python сам преобразует этот исходный текст в некоторую промежуточную форму, называемую байткодом, а затем переводит его на машинный язык и запускает. Всё это заметно облегчает использование Python, поскольку нет необходимости заботиться о компиляции программы, подключении и загрузке нужных библиотек и т.д. Вместе с тем, это делает программы на Python намного более переносимыми, так как достаточно их просто скопировать на другой компьютер, и они работают!

Литература

1. “A Byte of Python” Swaroop C. H.
2. https://habr.com/ru/hub/python/
3. https://www.mvoronin.pro/en/blog/post-75
4. https://proglib.io/p/python-applications/
5. https://habr.com/ru/post/340894/
6. https://itproger.com/news/132
7. https://wiki2.org/ru/Интерпретируемый_язык_программирования
8. https://ru.wikipedia.org/wiki/Python