Title

Введение:

Сейчас многие отрасли испытывают дефицит it-специалистов. Это открывает множество возможностей для людей для изучения программирования. Для этого в качестве первого языка многие выбирают именно Python3 из-за его изначальной простоты в освоении, широкой поддержки разработчиками и сообществом и востребованности во многих направлениях разработки ПО. Однако, на мой взгляд при изучении любого языка программирования кроме синтаксиса и умения работать с прикладными библиотеками необходимо также знание базовых (внутренних) принципов работы выбранного вами инструмента. В случае с Python3 речь идёт об интерпретаторе кода.

Цель: подготовить актуальный и структурированный доклад о принципе работы интерпретатора языка программирования Python3.

Задачи:

* Найти информацию об устройстве интерпретатора и его отличиях от компилятора.
* Изучить этапы исполнения программы (сценария) на языке Python3
* Продемонстрировать процесс исполнения программы на реальном примере и сделать выводы

**Глава 1.** Python - что такое интерпретатор и его отличия от компилятора.

Следует начать с того, что Python — мультипарадигменный высокоуровневый интерпретируемый язык программирования. Он был разработан в 1991 году Гвидо ван Россумом. Язык испытал влияние C и Lisp, что заметно из его синтаксиса и реализации поддержки функционального стиля программирования. Программа на языке Python может рассматриваться ещё и как сценарий — кода, решающего конкретную задачу «здесь и сейчас» (благодаря чему язык нашёл широкое применение в сфере автоматизации и тестирования ПО). Этот язык обладает внушительной базой кода — сейчас сайт pypi.org насчитывает более 400 тысяч библиотек.

Как было упомянуто ранее, Python является интерпретируемым языком. Интерпретатор — это программа, выполняющая интерпретацию. Данный процесс включает в себя построчный анализ, обработку и исполнение исходного кода. Также существуют компилируемые языки, такие как C, Java, Go и другие. Компилятор — программа, реализующая в целом тот же функционал, что и интерпретатор, однако вся обработка производится статически. Это значит, что все необходимые приведения типов и объем памяти для выполнения программы известны ещё до её выполнения. Таким образом, компилятор выявляет все возможные ошибки при потенциальном исполнении программы и требует их исправить до запуска. Интерпретатор же проверяет лишь синтаксическую правильность написанного кода и в таком случае позволит его запустить. Принято считать, что в своём составе интерпретатор также содержит компилятор, который отвечает за отсутствие синтаксических ошибок в исполняемой программе, при этом не проверяя корректность операций связанных с типами данных и необходимый объём памяти. Основное преимущество компилируемого подхода — заранее известные условия выполнения программы, а интерпретируемого — отсутствие необходимости тратить время на проверку кода, особенно в крупных проектах с большой базой исходного кода.

**Глава 2.** Процесс компиляции исходного кода.

Как было упомянуто ранее, процесс обработки кода на языке Python включает в себя компиляцию и интерпретацию на виртуальной машине Python Virtual Machine (PVM). Это показано на рисунке 1.

На протяжении всего процесса рассмотрения будет использоваться реализация CPython — каноничная реализация интерпретатора на языке C. По сути, любая программа на Python в процессе компиляции так или иначе превращается в программу на языке C. Подробнее процесс компиляции показан на рисунке 2. Как видим, процесс можно разбить на следущие этапы:

1. Разбор кода с помощью синтаксического дерева (Parse Tree);
2. Токенизация и генерация абстрактного синтаксического дерева (AST);
3. Генерация объектов кода;
4. Генерация и опптимизация байт-кода;

Для наглядности будет использована программа, содержащая в себе функцию, принимающую один числовой аргумент и увеличивающий его на единицу, затем принимающая с консоли число и выводящая результат выполнения функции на нём (рисунок 3).

2.1 Дерево разбора (Parse Tree)

Первый этап обработки программы — синтаксическое дерево (дерево разбора). Процесс создания синтаксического дерева представляет собой сопоставление лексем языка с его формальной грамматикой: учитывает порядок слов, их связь в предложении и т. д. На рисунке 4 в качестве примера приведено типичное для английского языка синтаксическое дерево. Процесс создания дерева разбора происходит при помощи парсера. В Python используется LL-парсер — это значит, что разбор кода происходит слева направо и вывод результата также производится слева. В листинге 5 приведена часть актуальной грамматики Python — набора правил, по которым парсер производит разбор кода. Полная версия грамматики также находится в материалах к проекту. После создания дерева разбора интерпретатор переходит к генерации абстрактного синтаксического дерева. В языке есть модуль parse, являющийся интерфейсом для обращения к синтаксическому дереву программы, его изменению и компиляции. Однако в рамках проекта он не представляет интереса, т. к. не даёт никаких наглядных представлений передаваемого ему кода.

2.2 Токенизация и абстрактное синтаксическое дерево (AST)

Вся программа на языке Python состоит из токенов. Например, return — токен ключевого слова, литерал 2 — численный токен и т. п. Все токены относятся к какой-либо группе из следущего списка:

1. Индентификаторы — имена, задаваемые пользователем;
2. Операторы — например, +,-,\* и другие.
3. Разделители — например, скобки, операторы присваивания и др.
4. Литералы — значения переменных, такие как «spam», 2, 0xAFFA и подобные.
5. Комментарии
6. Специальный литерал новой строки — так как Python не использует фигурные скобки, имеет особое значение
7. Литералы отступа для показания вложенности

В процессе генерации дерева разбора каждой лексеме присваивается свой уникальный номер в зависимости от того, к какой категории токенов она принадлежит и какую функцию выполняет.

Далее дерево разбора переводится в AST — представление кода, лишённое особенностей синтаксиса языка. В файле res.txt приведён вывод скрипта decompile.py для рассматриаемой в докладе программы example.py. В этой строке следует обратить внимание на 3 самых важных её части — блоки FunctionDef, Assign и Expr. Из названий параметров этих блоков понятно, за что отвечает каждый из них

В блоке FunctionDef происходит задание функции f. Также описывается операция сложения (Add), в ходе которой аргумент функции увеличивается на константу (единицу). В конце блока указано, какая локальная переменная возвращается. В этом блоке и далее каждый упоминаемый объект AST имеет большое количество не интересующих нас аргументов, таких как ключевые слова, или, например, список декораторов функции.

В блоке Assign создаётся переменная а, которой присвается значение из консольного ввода. Следует помнить, что всякая переменная в языке Python — ссылка на область памяти, в которой хранится объект. Объекты бывают неизменяемые (числа, строки) и изменяемые (списки, отображения, кортежи). Так, при изменении объекта первого типа значения всех переменных, ссылающихся на него создают новые объекты и перезаписываются, тогда как при изменении объектов второго типа переменные продолжают ссылаться на исходный объект.

В блоке Expr находится описание функции print, выводящей результат выполнения функции f на консоль.

После создания абстрактного синтаксического дерева интерпретатор пере

Приложение

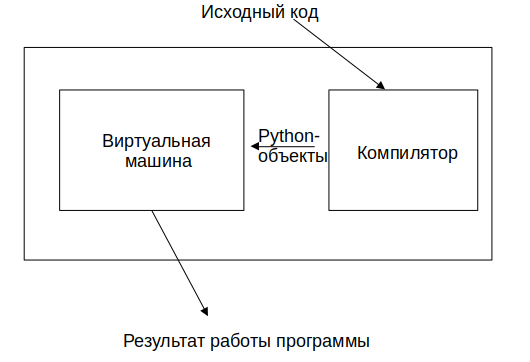


Рис. 1 Процесс обработки Python-кода

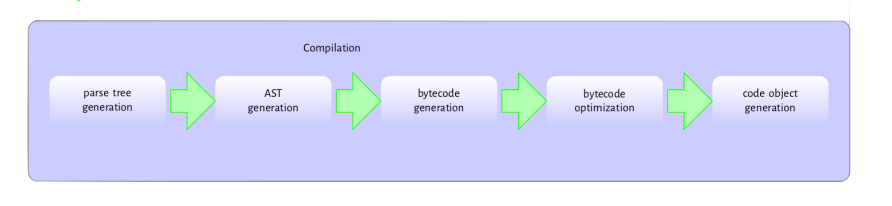
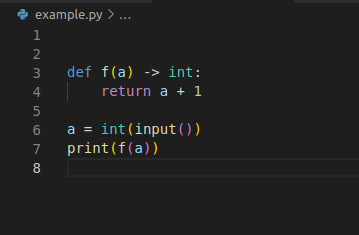
Рис. 2 Компиляция Python-кода

Рис.3 Используемая программа (example.py)

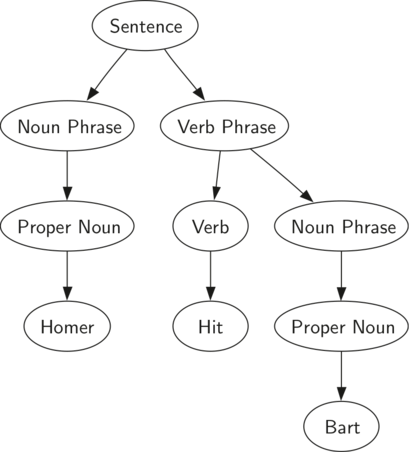
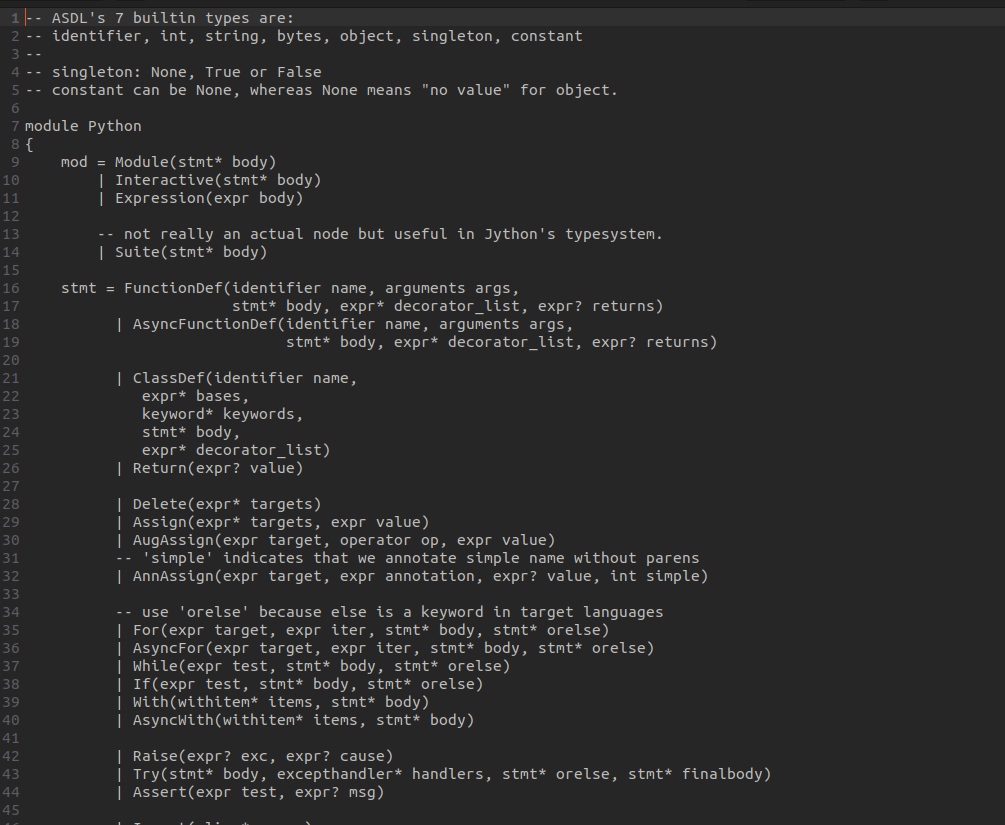


Рис 4. Синтаксическое дерево английского языка



Листинг 5. Грамматика Python

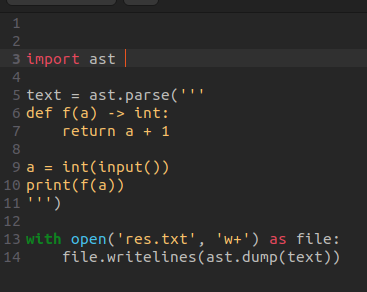


Рис.6 Декомпилятор исходного кода к AST