**Урок 1. История языков программирования. Компиляция и интерпретация**

**Программа. Язык программирования**

Программу можно представить как набор последовательных команд (алгоритм) для объекта (исполнителя), который должен их выполнить для достижения определенной цели. Так условно можно запрограммировать и человека, если составить ему инструкцию "как приготовить оладьи", а он начнет четко ее исполнять. При этом инструкция (программа) для человека будет написана на так называемом естественном языке (русском, английском или др.).

Обычно принято программировать не людей, а вычислительные машины, используя при этом специальные языки. Использование особых языков вызвано тем, что машины не в состоянии понимать наши (человеческие) языки. Для "инструктирования" машин предназначены разнообразные языки программирования, которые характеризуются синтаксической однозначностью (например, в них нельзя менять местами определенные слова) и ограниченностью (строго определенный набор слов и символов).

**Основные этапы развития языков программирования**

Первые программы писались на машинном языке, т.к. для ЭВМ того времени еще не существовало развитого программного обеспечения, а машинный язык — это единственный способ взаимодействия с аппаратным обеспечением компьютера. Каждую команду машинного языка напрямую выполняет то или иное электронное устройство. Данные и команды программисты записывали в цифровом виде (например, в шестнадцатеричной или двоичной системах счисления). Понять программу на таком языке очень сложно; кроме того, даже небольшая программа получалась состоящей из множества строк кода. Ситуация осложнялась еще и тем, что каждая вычислительная машина понимает лишь свой машинный язык.

Людям, в отличие от машин, более понятны слова, чем наборы цифр. Стремление человека оперировать словами и не цифрами привело к появлению ассемблеров. Это языки, в которых вместо численного обозначения команд и областей памяти используются словесно-буквенные.

При этом появляется проблема: машина не в состоянии понять наборы букв. Необходим какой-нибудь переводчик на ее родной машинный язык. Поэтому, начиная со времен ассемблеров, под каждый язык программирования создаются трансляторы — специальные программы, преобразующие программный код с языка программирования в машинный код. Ассемблеры на сегодняшний день продолжают использоваться (в системном программировании — низкоуровневые интерфейсы операционных систем, части драйверов и др.).

После ассемблеров наступил рассвет языков так называемого высокого уровня. Для этих языков потребовалось разрабатывать более сложные трансляторы, т.к. языки высокого уровня куда больше удобны для человека, чем для вычислительной машины. В отличие от ассемблеров, которые остаются привязанными к своим типам машин, языки высоко уровня обладают переносимостью. Т.е., написав один раз программу, программист мог выполнить ее на любой машине.

Следующим значимым шагом было появление объектно-ориентированных языков программирования. С помощью таких языков программист как бы оперирует виртуальными объектами. На сегодняшний день, реализация больших и сложных проектов осуществляется в основном с помощью объектно-ориентированного программирования.

**Разнообразия языков программирования**

На сегодняшний день существует огромное множество различающихся и похожих между собой языков программирования. Причина такого явления становится понятна, если представить то количество и разнообразие задач, которые на сегодняшний день решается с помощью вычислительной техники. Для решения разных задач требуются разные инструменты (т.е. языки программирования).

Многие программисты старались в прошлом и стараются сейчас придумать свой язык программирования, обладающий теми или иными преимуществами. Хотя подавляющее большинство программистов в настоящее время тратят огромное количество времени на изучение уже существующего арсенала инструментов.

Все существующее многообразие языков можно условно классифицировать по разным критериям. Например, по типу решаемых задач (языки системного или прикладного назначения, языки для web-разработки и др.).

**Трансляция**

Ранее было сказано, что для перевода кода с одного языка программирования (например, языка высокого уровня) на другой (например, машинный язык) требуется специальная программа — транслятор.

Механизм этого перевода весьма сложен, при этом выделяют два основных способа трансляции — компиляция программы или ее интерпретация.

При компиляции весь исходный программный код (тот, который пишет программист) сразу переводится в машинный. Создается так называемый отдельный исполняемый файл, который никак не связан с исходным кодом. Выполнение исполняемого файла обеспечивается операционной системой.

При интерпретации выполнение кода происходит последовательно (можно сказать, строка за строкой). Операционная система взаимодействует с интерпретатором, а не исходным кодом.

Выполнение откомпилированной программы происходит быстрее, т.к. она представляет собой готовый машинный код. Однако на современных компьютерах снижение скорости выполнения при интерпретации обычно не заметно.

**Урок 2. Знакомство с Python и средами программирования. Платформа replit.com**

**История**

Язык программирования Python был создан примерно в 1991 году голландцем Гвидо ван Россумом.

Свое имя - Пайтон (или Питон) - получил от названия телесериала, а не пресмыкающегося.

После того, как Россум разработал язык, он выложил его в Интернет, где уже целое сообщество программистов присоединилось к его улучшению.

Python активно совершенствуется и в настоящее время. Часто выходят его новые версии. Официальный сайт http://python.org.

**Особенности**

Python – это интерпретируемый язык программирования: исходный код частями преобразуется в машинный в процессе выполнения специальной программой — интерпретатором.

Python характеризуется ясным синтаксисом. Читать код на этом языке программирования достаточно легко, т.к. в нем мало вспомогательных элементов, а правила языка заставляют программистов делать отступы. Понятно, что хорошо оформленный текст с малым количеством отвлекающих элементов читать и понимать легче.

Python – это полноценный, можно сказать универсальный, язык программирования. Он поддерживает объектно-ориентированное программирование (на самом деле он и разрабатывался как объектно-ориентированный язык).

Также Python распространяется свободно на основании лицензии подобной GNU General Public License.

**Дзэн Питона**

Если интерпретатору Питона дать команду import this (импортировать "сам объект"), то выведется так называемый "Дзен Питона", иллюстрирующий идеологию и особенности данного языка. Глубокое понимание этого дзена приходит тем, кто сможет освоить язык Python в полной мере и приобретет опыт практического программирования.

**Как писать программы**

**Интерактивный режим**

В основном интерпретатор выполняет команды построчно: пишешь строку, нажимаешь Enter, интерпретатор выполняет ее, наблюдаешь результат.

Это очень удобно, когда человек только изучает программирование или тестирует какую-нибудь небольшую часть кода. Ведь если работать на компилируемом языке, то пришлось бы сначала написать код на исходном языке программирования, затем скомпилировать и уж потом запустить исполняемый файл на выполнение.

Работать в интерактивном режиме в ОС Linux можно в консоли. Для этого следует выполнить команду python. Запустится интерпретатор, где сначала выведется информация об интерпретаторе. Далее, последует приглашение к вводу (>>>).

Запустите интерпретатор Питона.

Поскольку никаких команд мы пока не знаем, то будем использовать Питон как калькулятор (возможности языка это позволяют).

2 +5

3 \* (5 - 8)

2.4 + 3.0 / 2

и т.д.

Наберите подобные примеры в интерактивном режиме (в конце каждого нажимайте Enter).

Ответ выдается сразу после нажатия Enter (завершения ввода команды).

Бывает, что в процессе ввода была допущена ошибка или требуется повторить ранее используемую команду. Чтобы не писать строку сначала, в консоли можно прокручивать список команд, используя для этого стрелки на клавиатуре.

Другой вариант работы в интерактивном режиме — это работа в среде разработки IDLE, у которой есть интерактивный режим работы. В отличие от консольного варианта здесь можно наблюдать подсветку синтаксиса (в зависимости от значения синтаксической единицы она выделяется определенным цветом). Прокручивать список ранее введенных команд можно с помощью комбинаций Alt+N, Alt+P.

Запустите IDLE. Попробуйте решать математические примеры здесь.

**Создание скриптов**

Несмотря на удобства интерактивного режима работы при написании программ на Питоне, обычно требуется сохранять исходный программный код для последующего использования. В таком случае подготавливаются файлы, которые передаются затем интерпретатору на исполнение. По отношению к интерпретируемым языкам программирования часто исходный код называют скриптом. Файлы с кодом на Python обычно имеют расширение py.

Подготовить скрипты можно в той же среде IDLE. Для этого, после запуска программы в меню следует выбрать команду **File → New Window** (Crtl + N), откроется новое окно. Затем желательно сразу сохранить файл (не забываем про расширение py). После того как код будет подготовлен, снова сохраните файл (чтобы обновить сохранение). Ну и наконец, можно запустить скрипт, выполнив команду меню **Run → Run Module** (F5). После этого в первом окне появится результат выполнения кода. (Примечание: если набирать код, не сохранив файл в начале, то подсветка синтаксиса будет отсутствовать.)

Подготовьте скрипт (с примерами). Запустите его на выполнение.

На самом деле скрипты можно готовить в любом текстовом редакторе (желательно, чтобы он поддерживал подсветку синтаксиса языка Python). Кроме того, существуют специальные программы для разработки.

Запускать подготовленные файлы можно не только в IDLE, но и в консоли с помощью команды python адрес/имя\_файла.

В консоли передайте интерпретатору Питона на выполнение подготовленный файл.

Кроме того, существует возможность настроить выполнение скриптов с помощью двойного клика по файлу (в Windows данная возможность присутствует изначально).

***Данные и их типы***

Можно заметить, что все, что мы делаем, мы делаем над чем-то — какими-то предметами или объектами. Мы меняем свойства объектов и их возможности. Программы для компьютеров также манипулируют какими-то объектами (назовем их пока *данными*).

Очевидно, данные бывают разными. Очень часто компьютерной программе приходится работать с числами и строками. Например, на прошлом уроке мы "манипулировали" числами, выполняя над ними арифметические операции. Можно сказать, что операция сложения выполняла изменение первого числа на величину второго, или умножение увеличивало одно число в количество раз, соответствующее второму.

Числа в свою очередь также бывают разными: целыми, дробными, могут иметь огромное значение или очень длинную дробную часть. При знакомстве с языком программирования Python мы точно столкнемся с тремя типами данных:

– *целые числа (integer)* – положительные и отрицательные целые числа, а также 0 (например, 4, 687, -45, 0).

– *числа с плавающей точкой (float point)* – дробные числа (например, 1.45, -3.789654, 0.00453). Примечание: разделителем целой и дробной части служит точка, а не запятая.

– *строки (string)* — набор символов, заключенных в кавычки (например, "ball", "What is your name?", 'dkfjUUv', '6589'). Примечание: кавычки в Python могут быть одинарными или двойными.

***Операции***

Можно сказать, что *операция* — это выполнение каких-нибудь действий над данными (*операндами*). Для выполнения конкретных действий требуются специальные инструменты — *операторы*.



Например, символ "+" по отношению к числам выполняет операцию сложения, а по отношению к строкам — конкатенацию (соединение). Парный знак \*\* возводит первое число в степень второго.

|  |  |
| --- | --- |
| **Выражение** | **Результат выполнения** |
| 34.907 + 320.65 | 355.55699999999996 |
| "Hi, " + "world :)" | 'Hi, world :)' |
| "Hi, " \* 5 | 'Hi, Hi, Hi, Hi, Hi |

***Изменение типа данных***

Что будет, если мы попытаемся выполнить в одном выражении операцию над разными типами данным? Например, сложить целое и дробное число, число и строку. Однозначный ответ дать нельзя: так, при складывании целого числа и числа с плавающей точкой, получается число с плавающей точкой, а если попытаться сложить любое число и строку, то интерпретатор Python выдаст ошибку.

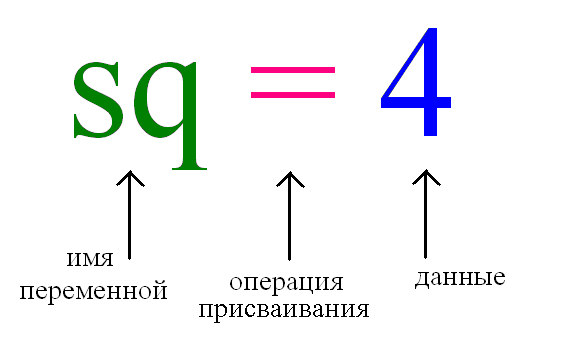
|  |  |
| --- | --- |
| **Выражение** | **Результат выполнения** |
| 1 + 0.65 | 1.6499999999999999 |
| "Hi, " + 15 | O ш и б к а |

Однако, бывают случаи, когда программа получает данные в виде строк, а оперировать должна числами (или наоборот). В таком случае используются специальные функции (особые операторы), позволяющие преобразовать один тип данных в другой. Так функция **int()** преобразует переданную ей строку (или число с плавающей точкой) в целое, функция **str()** преобразует переданный ей аргумент в строку, **float()** - в дробное число.

|  |  |
| --- | --- |
| **Выражение** | **Результат выполнения** |
| int ("56") | 56 |
| int (4.03) | 4 |
| int ("comp 486") | O ш и б к а |
| str (56) | '56' |
| str (4.03) | '4.03' |
| float (56) | 56.0 |
| float ("56") | 56.0 |

Данные хранятся в ячейках памяти компьютера. Когда мы вводим число, оно помещается в память. Но как узнать, куда именно? Как в последствии обращаться к этим данными? Раньше, при написании программ на машинном языке, обращение к ячейкам памяти осуществляли с помощью указания регистров. Но уже с появлением ассемблеров, при обращении к данным стали использовать так называемые *переменные*. Механизм связи между переменными и данными может различаться в зависимости от языка программирования и типа данных. Пока достаточно запомнить, что данные связываются с каким-либо именем и в дальнейшем обращение к ним возможно по этому имени.

В программе на языке Python связь между данными и переменными устанавливается с помощью знака **=**. Такая операция называется присваиванием. Например, выражение sq = 4 означает, что на объект (данные) в определенной области памяти ссылается имя sq и обращаться к ним теперь следует по этому имени.



Имена переменных могут быть любыми. Однако есть несколько общих правил их написания:

1. Желательно давать переменным осмысленные имена, говорящие о назначении данных, на которые они ссылаются.
2. Имя переменной не должно совпадать с командами языка (зарезервированными ключевыми словами).
3. Имя переменной должно начинаться с буквы или символа подчеркивания (\_).

Чтобы узнать значение, на которое ссылается переменная, находясь в режиме интерпретатора, достаточно ее вызвать (написать имя и нажать Enter).

Пример работы с переменными в интерактивном режиме:

>>>  apples = 100

>>>  eat\_day = 5

>>>  day = 7

>>>  apples = apples - eat\_day \* day

>>>  apples

65

>>>

***Практическая работа***

1. Переменной var\_int присвойте значение 10, var\_float - значение 8.4, var\_str - "No".
2. Измените значение, хранимое в переменной var\_int, увеличив его в 3.5 раза, результат свяжите с переменной big\_int.
3. Измените значение, хранимое в переменной var\_float, уменьшив его на единицу, результат свяжите с той же переменной.
4. Разделите var\_int на var\_float, а затем big\_int на var\_float. Результат данных выражений не привязывайте ни к каким переменным.
5. Измените значение переменной var\_str на "NoNoYesYesYes". При формировании нового значения используйте операции конкатенации (+) и повторения строки (\*).
6. Выведите значения всех переменных.

**Результат выполнения практической работы**