ЯЗЫК PYTHON

Лекция 1. Введение. Простейшие программы. Базовые типы данных

Краткий анонс

1. Изучаем Python
2. Как язык программирования
   - Со своим синтаксисом и семантикой
3. И как механизм алгоритмизации

Почему именно Python

1. Потому что популярен
   - В частности - в среде непрофессиональных разработчиков
2. Сравнительно прост
   - Но есть нюансы
   - Просто запустить, сложнее добиться корректной работы

Простейшая программа

```
1 print("Hello, world")
```

Как ее запустить

1. Непосредственно интерпретатором
2. IDE: Integrated Development Environment

Из чего состоит простейшая программа

1. Имя функции: `print`
2. Строковая константа: `"Hello, world"`
3. Вызов функции: `print("hello, world")`

Слегка усложним

```
1 name = input()
2 print("Hello, " + name)
```

Что нового

1. Еще одна функция: `input`
2. Переменная: `name`
3. Строковое выражение: `"Hello, " + name`

Посчитаем что-то минимально полезное

```
1 print("enter number: ")
2 s = input()
3 n = int(s)
4
5 print(n, "^ 2 =", n * n)
```

Что здесь нового

1. Еще одна функция: `int`
2. Несколько переменных: `s, n`
3. Разные типы данных: строка, число
4. Численное выражение: `n * n`

Что такое типы данных

1. В памяти хранятся байты - минимальная единица обмена
2. Байты состоят из 8 битов
3. У байта есть адрес
4. Переменная в Python - именованный адрес области памяти
5. Интерпретировать содержимое памяти можно по-разному

Что такое типы данных

1. Способ интерпретации содержимого памяти - и есть тип данных
2. Пример
   - Целочисленный тип
   - Содержимое байтов интерпретируется как двоичное число
   - В программе представляется в виде последовательности цифр
   - Или является результатом преобразования, вычисления и т.п.

Примеры целочисленных литералов

1. Печать: `print(12345)`
2. Присваивание: `n = 2`
3. Выражение: `v = 123 * 2345`
4. Вперемешку с переменными: `v = v + 3`

Размер и скорость

1. Явного ограничения нет
2. Можно так: `12345678901234567890 * 12345678901234567890`
3. У целых в Python переполнения не бывает
4. Но скорость работы может снижаться для очень больших чисел
5. И для небольших - тоже не идеальная

А почему он популярный, если такой медленный

1. Можно написать программу на Python
2. Работать она будет 5 минут, а писать ее будем 2 часа
3. А можно - на C++
4. Работать будет минуту, а писать ее будем 2 дня
5. Что лучше ?

Что лучше ?

1. Если 1 раз - лучше Python
2. И даже если 100 раз
3. А если 1000 раз ежеминутно - на C++
4. Особенно если клиент заинтересован в быстром ответе
5. У Питона - своя широкая ниша

Операции над целыми числами

1. Сложение: +
2. Вычитание: -
3. Умножение: *
4. Возведение в степень: **
5. Деления - целых два
6. И взятие остатка

Деления и остаток

1. 'Обычное' деление: /
   - Выходим за рамки целых чисел
   - 18 / 4 дает нам 4.5
   - 4.5 - отдельный тип данных (вещественные)
   - О них чуть позже

Деления и остаток

1. Деление нацело: `//`
   - Получаем целое число
   - `22 // 4` дает нам `5`
   - Максимальное целое `n` такое, `n * 4` не превосходит `22`

Деления и остаток

1. Остаток: `%`
   - `21 % 4` дает нам 1
   - Но мы пока смотрели только совсем простые примеры
   - Числитель положительный и знаменатель - тоже

Деления и остаток

1. Деление (или остаток от деления) на 0
   - Будет 'ошибка'
   - А если точнее
     - Будет брошено исключение
     - Если ничего не предпринять, останется непойманным
     - Подробнее про исключения - ближе к концу курса

Деления и остатки с отрицательными

1. С обычным - все просто
2. По школьным правилам
3. С делением нацело и остатком - посложнее
   - `-21 % 4`
   - Сколько это будет ?

Деления и остатки с отрицательными

1. Если сначала посчитать без знака, а потом определить знак
   - То `-21 / 4` дало бы -5
   - Но оно дает -6
   - А почему ?

Деления и остатки с отрицательными

1. Вспомним формулировку для положительных
   - Наибольшее число n такое, что n * 4 не превосходит -21
   - Это и есть -6
   - Осталось еще два случая про деление

Деления и остатки с отрицательными

1. `21 % -4`
2. `-21 % -4`
3. Пусть смена знаков числителя и знаменателя не меняет результата
4. То есть -6 и 5

Деления и остатки с отрицательными

1. Разберем ситуацию с остатками
2. Общая схема
   - Вычисляем неполное частное
   - Умножаем на делитель
   - Полученное значение вычитаем из делимого

Деления и остатки с отрицательными

1. Потренируемся на положительных
2. `37 % 5`
   - Вычисляем неполное частное: 7
   - Умножаем на делитель: 35
   - Полученное значение вычитаем из делимого: 2

Деления и остатки с отрицательными

1. Положительный делитель, отрицательное делимое
2. `-37 % 5`
    1. Вычисляем неполное частное: -8
    2. Умножаем на делитель: -40
    3. Полученное значение вычитаем из делимого: 3

Деления и остатки с отрицательными

1. Положительное делимое, отрицательный делитель
2. `37 % -5`
   - Вычисляем неполное частное: -8
   - Умножаем на делитель: 40
   - Полученное значение вычитаем из делимого: -3

Деления и остатки с отрицательными

1. Отрицательное делимое, отрицательный делитель
2. `-37 % -5`
   - Вычисляем неполное частное: 7
   - Умножаем на делитель: -35
   - Полученное значение вычитаем из делимого: -2

Вещественные числа

1. Немного математики
   - Есть рациональные дроби
   - Их можно преобразовать в десятичные дроби
   - Иногда получается конечная десятичная дробь (1/4, 0.25)
   - Иногда получается бесконечная десятичная дробь (1/3, 0.333333...)

Вещественные числа

1. Немного математики
   - Рациональные дроби могут давать бесконечные периодические
   - Но бывают и непереодические
   - Например, корень из 3
   - Или число PI

Вещественные числа

1. Представление в памяти
   - Память конечна
   - Поэтому если быть совсем честными, мы не храним в ней вещественные числа
   - Мы храним их приближенные значения
   - Для большинства задач это ok
   - Но могут случаться неприятности - о них чуть позже

Символы и строки

1. Представление в памяти
   - В памяти хранятся байты
   - В них - нолики и единички
   - То есть числа
   - А мы хотим хранить буквы, знаки и т.п.

Символы и строки

1. Представление в памяти
   - Пусть каждому значению байта соответсвует какой-то знак
   - Первая широко распространенная кодировка - ASCII
   - American Standard Codes for Informational Interchange
   - https://en.wikipedia.org/wiki/ASCII

Символы и строки

1. Представление в памяти
   - ASCII использует 1 байт на символ
   - Это 256 значений
   - Этого хватало на латиницу + один национальный алфавит (не китайский)
   - Чтобы использовать более богатый набор символов одновременно, придумали Unicode

Символы и строки

1. Представление в памяти
   - Unicode - расширение ASCII
   - Два байта на символ
   - А иногда - 4
   - https://en.wikipedia.org/wiki/List_of_Unicode_cha

Символы и строки

1. Представление в памяти
   - Строка - набор символов
   - Т.е. область памяти, в которой хранятся коды символов
   - И количество символов
   - Количество символов можно узнать с помощью функции len

Строковые литералы

1. Много разных вариантов
2. Пока - два простейших
   - Набор символов в одинарных кавычках: `'hello'`
   - Набор символов в двойных кавычках `"hello"`

Откуда еще берутся строки

1. Ввод данных
   - Функция `input()`
   - Читает символьный ввод до нажатия Enter
   - И возвращает прочитанное

Операции над строками

1. Сложение (конкатенация)

```
1 s = input()
2
3 print('your input is: ' + s)
```

Операции над строками

1. Умножение строки на число
2. Можно в любом порядке

```
1 s = input()
2 print('abc' * 10)
3 print(5 * 'hello ')
```

Преобразования типов данных

1. Поймем разницу
   - Между 1234 и '1234'
   - Между `1234 * 3` и `'1234' * 3`
   - И если хотим ввести через input() какое-то число и умножить на 3, то что нам делать ?

Преобразования типов данных

1. Функции int и str
   - int преобразует строку в целое число
   - Точнее - много всего пытается превратить в целое число
   - Другой пример - число с плавающей точкой
   - Отбрасывая дробную часть

Преобразования типов данных

1. Функции int и str
   - str целое число в строку
   - Точнее - много всего пытается превратить в строку
   - Другой пример - число с плавающей точкой
   - `str(1.23)` превратится в `'1.23'`

Неявные преобразования типов данных

1. Любой параметр print превращается в строку
2. Если контекст требует "вещественного" числа, то целое число превратится в "вещественное"
3. Но при конкатенации - не работает
4. Так нельзя: `'abc' + 123`
5. Так можно: `'abc' + str(123)`

Возведение в степень с вещественными числами

1. Есть операция **
2. С ней можно использовать "вещественные" числа
3. Как слева, так и справа
4. Например, можно возвести в степень 0.5
5. И это квадратный корень