PYTHON

Лекция 1

ПЛАН ПО ЛЕКЦИЯМ

- Будет в течение недели
- После опросов про бекграунд и ожидания
- В целом будем про язык и важные библиотеки
- Надо уточнить варианты

ПРАВИЛА КУРСА

- "Внутренняя" оценка - 100 балльная шкала
- (50, 57] - 4 в 10-бальной
- (57, 64] - 5 в 10-бальной
- (64, 71] - 6 в 10-бальной

ПРАВИЛА КУРСА

- (71, 78] - 7 в 10-бальной
- (78, 85] - 8 в 10-бальной
- (85, 92] - 9 в 10-бальной
- (92, Inf) - 10 в 10-бальной

СОСТАВНЫЕ ЧАСТИ ОЦЕНКИ

СОСТАВНЫЕ ЧАСТИ ОЦЕНКИ

- 5-6 домашек
- Каждая оценивается на 100 баллов
- На балл влияют: полнота решений, качество (по ревью), соблюдение дедлайнов
- Считаем среднюю по домашкам, отбросив худшую
- В итоговую оценку идет с коэффициентом 0.45

СОСТАВНЫЕ ЧАСТИ ОЦЕНКИ

- Почти после каждого семинара - набор автоматизированных тестов
- Каждый набор тестов оценивается в 100 баллов
- Считаем среднюю, отбросив 2 худших
- В итоговую оценку идет с коэффициентом 0.2

СОСТАВНЫЕ ЧАСТИ ОЦЕНКИ

- Экзамен - как беседа
- Что-то спрошу по домашкам, попрошу кусочек кода поревьюить
- Что-то спрошу на понимание
- Пять вопросов, по 20 баллов каждый
- В итоговую оценку идет с коэффициентом 0.35

МОЖНО БЕЗ ЭКЗАМЕНА

- Первый способ - получить высокие баллы за домашки
- Тогда 0 за экзамен не помешает зачету
- Но балл будет 4-5, не исключено, что 6
- Второй способ - получить право на автомат
- Выдается за высокую (и разумную) активность на семинаре
- На фоне высоких показателей по домашкам и тестам

ПОДРОБНЕЕ ПРО ТЕСТЫ

- Короткий дедлайн
- Разбор особо интересных задач на следующем семинаре
- В общем случае запоздалые доделки не предусмотрены
- Если пропускается по уважительной причине, то возможно новое задание
- Новое задание выдается в конце семестра

ПОДРОБНЕЕ ПРО ДОМАШКИ

- Дедлайн никто не меряет на секунды
- По мере пропущенных дней набегает понижающий коэффициент
- 1 день - 0.95
- 2 дня - 0.9
- 14 дней 0.3
- Понижается в день на 0.05

ПОДРОБНЕЕ ПРО ДОМАШКИ

- Через 14 дней фиксируется
- Остается 0.3 навсегда
- В случае ухода на пересдачу - не восстанавливается !!!
- Мораль: не забрасывайте домашки !!!

ПОДРОБНЕЕ ПРО ДОМАШКИ

ПОДРОБНЕЕ ПРО ДОМАШКИ

- Если тесты пройдены или не предусмотрены, то начинается code review
- По итогам review балл может быть понижен
- Как правило, замечания можно исправлять
- Как правило, со скидой
- Как правило, одна итерация на исправление

ПОДРОБНЕЕ ПРО ДОМАШКИ

- Возможны бонусы
- За активность на семинаре, доделки заданий с семинаров (будет объявляться)
- За особо красивые решения в тестах
- Бонусные баллы добавляются к релевантным домашкам, но не выше 100 баллов
- Переноса на другие домашки нет

АНТИПАТТЕРНЫ

- Появиться внезапно в конце семестра в расчете на особые условия
- Заявить задним числом, что "ничего не было понятно"
- Для тех, кто переводится - воспринимать перевод как универсальную индульгенцию
- Аппелировать к правилам других курсов
- Молча исчезнуть в середине семестра - лучше дать обратную связь

ПРАВИЛЬНЫЕ ПАТТЕРНЫ

- Вовлеченно участвовать в семинарах
- Задавать вопросы
- Давать конструктивную обратную связь

МОИ ОБЯЗАТЕЛЬСТВА

- Прислушиваться к обратной связи - в рамках реального
- Отвечать на вопросы и обратную связь - в рамках разумного
- Давать намеки и разумные подсказки по мере необходимости
- Незначительно корректировать критерии в сторону смягчения

ПЛАН НА СЕГОДНЯ

- Обзорная экскурсия по языку Python
- Погружение в язык
- Познакомимся с базовыми понятиями
- Со временем разберем в деталях

ПРОСТЕШИЙ КОД

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

ЧТО ВИДИМ

- Программа выполняется построчно
- Нет специальной точки входа
- Хотя в больших программах лучше код держать в функциях
- Как это сделать - чуть позже

"КАЛЬКУЛЯТОР"

```
1 print(23 * 34)
```

И ДАЖЕ ТАК

```
1 v = 12345678
2 for _ in  range(10):
3     print(v)
4     v = v * v
```

A BOT TAK - HET

```
1 v = 12345678.0
2 for _ in  range(10):
3     print(v)
4     v = v * v
```

И ТАК - ТОЖЕ (НО ПО ДРУГОЙ ПРИЧИНЕ)

```
1 v = 12345678
2 for _ in  range(20):
3     print(v)
4     v = v * v
```

ПОЙМЕМ, ЧТО ПРОИСХОДИТ

- Целые числа в Python - произвольного размера
- Это в чем-то хорошо, в чем-то не очень
- Нет переполнений
- Но дорого по памяти
- И массированная арифметика тормозит
- Есть библиотека numpy - для быстрой и экономией арифметики

ДЕСЯТИЧНЫЕ ДРОБИ

- Процессорная реализация
- Переполнение возможно
- Все равно дорого по памяти и массированная арифметика тормозит
- И спасение - тоже в numpy

СМОТРИМ ДАЛЬШЕ

- for и int сильно отличаются от range и print
- for, int - ключевые слова, элементы синтаксиса
- range, print - функции
- Функция - механизм декомпозиции
- Функции бывают встроенные и определенные в программе (обязательно научимся)

СМОТРИМ ДАЛЬШЕ

- Встроенная функция может быть написана на Python
- И может вызывать другие встроенные функции
- Но print рано или поздно должен что-то напечатать в терминал
- Это умеет делать системная библиотека
- И она написана не на Python

СМОТРИМ ДАЛЬШЕ

- Рано или поздно придется ее вызвать
- Существует механизм вызвать "не-Python" из Python
- В детали не вдаемся, но осознаем, что он есть
- На нем работают Python-библиотеки для работы с сетью, print, input

РАЗБЕРЕМСЯ С RANGE

- range - Python-функция
- Она создает Python-объект, представляющий числовой диапазон
- В нашем случае - от 0 до 10 (НЕ включительно)
- В print тоже есть Python-часть
- Именно она жалуется на слишком длинное число

ПЕРЕМЕННЫЕ

- v - переменная
- Переменная - символическая ссылка на объект
- В Python нет понятия "определение переменной" или "объявление переменной"
- Но есть тип переменной
- Тип переменной - тип объекта, ссылка на который в ней хранится

СДЕЛАЕМ СТРАННОЕ

```
1 print("a")
2 v = "123"
3 print("b")
4 v = v * v
5 print("c")
```

ДЛЯ СРАВНЕНИЯ

```
1 print("a")
2 v = "123"
3 print("b")
4 v = v *
5 print("c")
```

В ЧЕМ РАЗНИЦА

- В первом случае - синтаксис корректен
- Некорретна семантика в момент исполнения
- В переменной v может храниться что угодно
- Целое число, вещественное число, строка, любой другой объект

В ЧЕМ РАЗНИЦА

- Если в момент умножения v * v в ней целое число, то выполнится целочисленное умножение
- Если вещественное число, то будет вещественное умножение
- А если строка, то ошибка - потому что умножение строки на строку не определено
- Но случится это только в момент выполнения

В ЧЕМ РАЗНИЦА

- Во втором случае - синтаксис некорректен
- Это будет понятно до исполнения
- Поэтому ни строчки кода из такого файла не исполнится
- А теперь попробуем определить свою функцию

ПРИМЕР ФУНКЦИИ

```
1 def square(n):
2     return n * n
3
4 for i in range(10):
5     print(i, square(i))
```

РАЗБЕРЕМСЯ

- def - ключевое слово
- square - имя новой функции
- n - параметр функции
- Функция принимает параметры и возвращает значение

РАЗБЕРЕМСЯ

- В нашем случае - один параметр
- В момент вызова создается переменная с именем параметра
  То, что стоит в скобках в момент вызова, присваивается ей
- Тело функции исполняется до return

РАЗБЕРЕМСЯ

- Выполнение функции завершается
- Значение выражения справа от return подставляется на место вызова функции
- В нашем случае - на место второго параметра print
  Переменная n прекращает свое существование

МОДЕЛЬ ИСПОЛНЕНИЯ ПРОГРАММЫ

- Читается исходный файл целиком
- Проводится синтаксический анализ по грамматике Python
- Если текст не соответствует грамматике, сообщается о синтаксической ошибке
- Если соответствует, начинается исполнение

МОДЕЛЬ ИСПОЛНЕНИЯ ПРОГРАММЫ

- Заводится словарь переменных
- Можно представлять его как табличку с двумя колонками
- В левой имя переменной, в правой - значение
- Исполняем код построчно

МОДЕЛЬ ИСПОЛНЕНИЯ ПРОГРАММЫ

- Если встретили присваивание переменной, то определяем ее имя
- И ищем ее в словаре
- Также вычисляем выражение справа от знака присваивания
- Если нашли, то в правую колонку пишем новый адрес значения
- Если не нашли, то создаем и пишем в правую колонку адрес значения

МОДЕЛЬ ИСПОЛНЕНИЯ ПРОГРАММЫ

- Если в выражении встречается имя переменной, ищем ее в словаре
- Если не нашли, это ошибка
- Если нашли, подставляем его ссылку на значение вместо переменной
- А если обнаруживается вызов функции ?

ФУНКЦИЯ КАК ПЕРЕМЕННАЯ

- Функция ничем не отличается по сути от переменной
- Это тоже объект
- "Обычные" переменные могут перемежаться с определениями функций
- И исполняются по очереди

ФУНКЦИЯ КАК ПЕРЕМЕННАЯ

- Пока не дошли до определения функции, ее нет в таблице символов
- Когда дошли - оно там появляется
- В качестве имени - имя из определения функции
- В качестве значения - внутреннее представление функции

ВСТРОЕННЫЕ ФУНКЦИИ

- Если это Python-реализации, то как будто бы их определения исполнились перед исполнением программы
- Если это переход к C-функциям, то внутреннее представление другое
- Но смысл - тот же
- Есть встроенная функция type - возвращает тип объекта

```
1 print(type(1))
2 print(type(1.23))
3 print(type('123'))
4 print(type(print))
```