

Технологии програм-ия. Python как второй язык.
DevOps – новая метафора Жизненного Цикла ПО.
Python3 (основы), IDE(Pycharm), VCS(git).
Web-django-framework, [СУБД(PostgreSQL)]
[CLI интерфейс, shell-оболочка, утилиты, 4 "фишки"
баша: 1)\$, 2)` , 3)|, 4)>, bash-скрипты.] [Докеры.]

Лекция 1/13

DevOps как новая культура в создании ПО, модели и платформы.
Python. Стиль кода, сравнение с Си. Автор и его время. Развитие
языка, его философия и место среди... Вопросы для экз. (sic!)
(v. 0.8)

И.Ф.Травов

igor.travov@gmail.com

Оргвопросы

Требования по курсу:

- **Лингва франка курса:** Python3.6, оболочка — python3 (можно ipython3 или bpython3). Оболочка Питона, командная строка OS Linux внутри Pycharm.
- **Требуется установить:**
 - **дистрибутив языка** с родного сайта или анакондовский дистрибутив (см. ниже)
 - **среду разработки** – IDE-Pycharm (community edition не ниже 2017.2.2),
 - **настроить IDE** на Питон3.6, Питон оболочку типа ipython3, терминал для командной строки,
 - **либо** в ваш наркоманский виндовоз поставить окружение от ubuntu (для докерности в след семестре),
 - **либо** в ваш наркоманский виндовоз поставить Cygwin (от Red Hat, см Википедию) (для докерности в след семестре).
- **Вопрос:** кто имеет пусть минимал опыт с ОС Linux, или macOS?
- **Самое время начинать новую жизнь и слезать с виндовозовской иглы!!!** иначе не сможете гарантировать себе свою безопасность!!!

Источники: откуда загрузить и что почитать

<https://www.python.org/downloads/> официальный сайт для скачивания дистрибутива, для изучения, с инфор-ей по всему из мира Питон;
(Anaconda) <https://www.continuum.io/downloads> Дистрибутив Питона для всех ОС. (с акцентом на научный питон – обработку всякой математики и вывода графики налету в веб интерфейсе – проект джупитер со своей ide и док/презентац-ой системой);
<https://docs.python.org/3/download.html>): (пакет pdf - документов);
docs.python.org/3.6/ раздел с документацией по языку версии 3.6.XX и больше/меньше;
docs.python.org/3.6/library/index.html Питон стандартная биб-ка (держат под подушкой), PSL 37 разделов, 1950 pdf страниц;
pythonz.net русскоязычный сайт в помощь (справочник, перевод полезных видео);
pythonworld.ru Python-3 для начинающих;
<http://ru.stackoverflow.com/questions/tagged/python+python-3.x> ясно;
Книги: Любые по Python-3, поновее и с адекватным переводом.

Вопрос: кто в чём собирается специализироваться?

1. Разработчик ПО:

(джуниор|синьор|тимлидер|архитектор ПО|рук-ль проекта|freelancer).

2. Системный администратор (или DevOps-инж, спец по CD,CI):

(сист аналитик|админ сетей/систем/БазД-ых|автоматизац операций по деливери|менеджер|главн конструктор по компьютеростроению).

3. Другой ИТ специалист:


(маркетолог|консультант|tec-writer|спец-ст тестир-щик|менеджер|прЕпод|офисн планктон|шоумэн(напр, ИТпроф-ный радиожурналист.[†]

4. **Хочу быть кем-то по ИТ**, но пока не знаю (*полустраусовая политика, "школьный синдром"* не преодолен, пока).

5. **Всё пОфиг, не знаю, чего-то жду**, буду на шее у родителей или родственников пока (*страусовая политика иждивенца-инфантила*).

(по специализации см. профессиональные стандарты (около 20) по различным ИТ специальностям, почти по всему компьютерингу!) [‡]

[†]<https://talkpython.fm/episodes/show/44> (про [ipython](#) или др очень интересные эпизоды про тот же Питон3.6 3.7...)

[‡]<http://www.apkit.ru/committees/education/meetings/standarts.php> 

4 направления программирования (условно).

Если для "нечистых" айтишников язык и ОС – это инструменты, а прикладная область основная проф. специализация, то для айтишников **хард и софт – это всё, их надо знать и уметь.**

Научное: общецелевое ПО, биг-датовая обработка, разработка всяких CAD/CAM/CAE пакетов физической среды моделирования, машин-ленинг, инструментальное ПО, параллельное ПО и...;

Вебовское: сетевое, http-сервисное, либо клиент-серверное, фронт-бэк-эндное, либо пиринговое (peer-to-peer) децентрализованное, глобально компьютерное;

Реал-таймовое: контроллеры промышленности технологий, рОботы, дроны, боевые/бытовые встраиваемые системы, там, где обработка прерываний от внешних всяких датчиков и исполнительных механизмов...;

“Игрушечное”: всякие игры для всяких гаджетов. Ну и Вы, куда? **Кодить будете что?, уже определились?, чего ждёте?**

(1) домзаготовка про бигдатовые возможности языков, 2) реалтайм особ

DevOps как новая культура в
создании ПО, модели и
платформы.

Какие проблемы решает DevOps1.0 (2009 – 2015)?

”Гармонизировать отношения между девелопментом, качеством (QA-quality assurance), и внедрением с эксплуатацией (operations), или иначе Институализация Continuous Delivery” (CD - непрерывная выдача). Тогда отсюда:

- описание инфраструктуры как кода;
- ускорение деплоя (учащение и автоматизация процесса релиза).

Какие проблемы решает DevOps2.0 (2016 и далее на сейчас)?

- релиз менеджмент – ключевой фактор успеха (как новые фичИ выложить на серверы, – от деливериз календарей как раньше, до деливериз пайплайн с автоматическим тестированием как сейчас).
- фич-тим-ы сходятся на том, что тикет готов, когда он вышел в продакшн, а не как раньше после девелопинга или тестинга;
- отсюда – архиважность формирования и взаимодействие команды – типа ответственного советского коллектива!!

Как бы генезис ДевОпс

- По сути это объединение системных администраторов с разработчиками прог обеспечения и частично с тестировщиками.
- Пограничная область между программистами и админами – т.е. появились **люди - девопсеры**, которые являются либо админами со знанием программирования, либо программистами со знанием администрирования
- Одновременный процесс разработки, тестирования и эксплуатации. Пересечение: разработки, тестирования и эксплуатации (как в диаграммах Венна?)
- Это также фИчер свичес (переключение фич). Когда вы управляете возможностями какие фичи присутствуют и какие отсутствуют в вашем приложении. Можете разворачивать не все приложение а на уровне отдельных фич!!!
- Continuous Delivery(CD): **git → build → test → operate**

Модели архитектур разработки/исполнения софтвера:

- клиент-серверная, фронт-энд/бэк-энд
 - ▶ Front-end (клиентская сторона, инструменты): JavaScript (+биб-а jQuery), Jupyter (питоновская интерактивная оболочка, вместе с др языками делает её незаменимой средой для научников), html, CSS
 - ▶ Back-end (серв сторона, инструменты): Java, Python, Ruby, php, C++, SQL ...
- сервис-ориентированная, Микросервисы, возможно через докеры. Подход, при котором единое приложение строится как набор небольших сервисов, каждый из которых работает в собственном процессе и коммуницирует с остальными используя легковесные механизмы, как правило HTTP. Централизованное управление этими сервисами минимально. Сами по себе сервисы могут быть написаны на разных языках, в т.ч. на Питоне.
- фреймвёрковая – каркасная. "заточенная" под IDE среда разработки узкоприкладных задач, (всякие SCADA-сис-мы, среды для технол языков (в т.ч. стандарта МЭК 61131), финанс/торгов среды, там, где приклад напр-е "рулит")

Python. Рейтинги. Стил ь кода,
сравнение с Си...
(см внимательно, прочувствуйте).

Рейтинги языков на июль-август 2017 по 3-м индексам:
1) индекс TIOBE, 2) индекс PYPL (PopularitY of Programming Language), 3) индекс IEEE Top Programming Languages

Python либо на 5-м, либо на 2-м,
либо на 1-м местах

(понятно, разные методики подсчета)[†]

[†] не для "крутизны" языка привел, хватает у него и сторонников и недоброжелателей, а для объективной его распространенности, универсальности и оптимальности как инструмента для большинства задач

Самые поверхностные сравнения кода на С и на Питоне.

(найти хотя бы 5 отличий, работаем в командной строке в оболочке или создаем файлы и запускаем обычным способом).

```
1 //на Си. Программа генерирует первые 15 чисел Фибоначчи.
2 include <stdio.h>
3 int main(void)
4 {
5     int Fibonacci[15], i;
6     Fibonacci[0] = 0;    // по умолчанию
7     Fibonacci[1] = 1;    // по умолчанию
8
9     for ( i = 2; i < 15; ++i )
10         Fibonacci[i] = Fibonacci[i-2] + Fibonacci[i-1];
11     for ( i = 0; i <15; ++i )
12         printf ("%i ", Fibonacci[i]);
13     printf ("\n");
14     return 0;
15 }

cc fib15.c    //# запускаем гниС, по классике, без ключей
./a.out      //# запускаем получившийся исполняемый файл

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

```

1 def fib(n): #теперь Питон, объяв-ие(define) функции fib
2     """ Программа генерирует первые 15 чисел Фибоначчи. """
3     a, b, k = 0, 1, 0      #внутриблочные отступы обязательны
4     while k < n:           # это требование синтаксиса
5         print(a, end=" ")
6         a, b = b, a + b    # множественное присваивание
7         k += 1             # дополненное(augmented) присв-ие
8     print()               # пустой принт для конечного перевода строки
9
10 fib(15) #последняя строчка файла, вызов функции с аргум-ом
Запускаем созданный файл и получаем строку с числами Фибоначчи
python fib15.py
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

```

- *a, b, k* – не переменные, в них нет их значений, а ссылки на объекты, они просто имена. Их типизация – динамическая, т.е. во время исполнения.
- докстрока (№2, в тройных ") встраивается в код и из нее автоматом делается *help()*.

- gcd.py – greatest common divisor

```
1 a = int(input("Vvedite a: ")) # input вводит строковое знач.
2 b = int(input("Vvedite b: ")) # через int делаем целое
3 while b:                       # пока b имеет значение True циклим
4     a, b = b, a % b #множественное присваивание справа налево
5     print ("a = ", a, "b = ", b)
6 print ("NOD = ", a)
```

```
python gcd.py      # запускаем программу
Vvedite a: 22      # вводим первое число
Vvedite b: 4       # второе
a = 4 b = 2
a = 2 b = 0
NOD = 2            # получаем ответ
```


- `geron.py`, здесь импортируем модуль из библиотеки и вызываем его как имя_модуля.имя_нужного_метода (`dot` – точка).

```
1 import math      # нам нужен метод math.sqrt (от square root)
2 a = int(input("введите сторону a "))
3 b = int(input("введите сторону b "))
4 c = int(input("введите сторону c "))
5 P = (a + b + c) / 2
6 Geron = math.sqrt(P * (P - a) * (P - b) * (P - c))
7 print("Площадь треугольника по Герону будет: ", Geron)
```

`python geron.py`

введите сторону a 3

введите сторону b 4

введите сторону c 5

Площадь треугольника по Герону будет: 6.0

- Функция *diving* выводит через так называемое **генераторное выражение** (в `()` внутри `for`) **списки полных имен** файлов (полных, начиная с корня, здесь от `./AAA`) во всех директориях от указанной вглубь рекурсивно. Ген-выражение экономит память, т.к. отдает элементы по одному.

```
import os      #см. PSL16.1 (Питон стандарт лайблари раздел 16.1)
```

```
def diving(dir):  
    acc = []    # пустой список для наращивания (extend)  
    for root, dirs, files in os.walk(dir): # итерация по трём  
        acc.extend(os.path.join(root, file) for file in files)  
        print(acc) # печатаем список acc на каждой итерации  
    return acc  
diving("./AAA")
```

```
python diving.py  
['./AAA/a2', './AAA/c3', './AAA/q1'] #вAAA лежат файлы a2, c3, q  
['./AAA/BBB/rr3', './AAA/BBB/eee43', './AAA/BBB/rr1'] #еще BBB
```

- Определение високосного года через создание длинных списков не литералами заранее, а программным способом (налету) через *list comprehension* – включение списков (в квадрат скобках на 2-ух строках).
- Работаем в оболочке, символы > > > как приглашение.
- встроенная функция *range()* дает диапазон упорядоченных целых от указанных границ.
- фильтр-условие задает алгоритм какие года являются или не явл-ся високосными (делится на 400 – да, на 100 – нет, остальные, если на 4 – тоже да)

```
>>> leaps = [y for y in range(1900, 1940)
              if (y % 4 == 0 and y % 100 != 0) or (y % 400 == 0)]
>>> leaps
[1904, 1908, 1912, 1916, 1920, 1924, 1928, 1932, 1936]
>>>
```

“Байка” от Д.Кнута про полный алгоритм с учетом 2000 и 10000 гг

- Варианты примеров обнуления intОвого массива на С (через указатель) и на Питоне (2 способа).

//Вариант на С

```
int a[10];    // физ скобки, определение функции и return
int *pa;      // не показаны (здесь всего 4 строки)
for(pa = a; pa < a + 10; pa++)
*pa = 0;
```

Вариант на Питоне:

```
>>> ll = [0 for i in range(10)] #опять через список-е включение
>>> ll
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
# всего одна строка, а можно еще более простой способ
# через умножение списка на целое
>>> ll = [0] * 10
>>> ll
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>>
```

Python – RAD язык (rapid application development) – плюсы и минусы:

- **RAD** – ответ на "неуклюжие" методы разработки программ 70-х 80-х гг, таких как «**модель водопада**» (Waterfall model);
- **быстрое проектирование и кодирование** (конёк Питона), 1 строчка на Питоне \approx 10 строк на С;
- но, если быстроедействие **исполнения** критично, необходимо либо переписывать участки кода, на С, либо **включать** уже готовый код от компилируемых (быстрых) языков (**питоновская обёртка.**) Вставка кода возможна с обеих сторон;
- Скорее всего, в 90% случаев это не нужно. Но бывает особенно нужно;
- **Один из примеров**-продуктов такого подхода – опенсорс система **Sage** компьютерной алгебры, покрывающая много областей математики, включая алгебру, комбинаторику, вычислительную математику, матанализ, теорию чисел, криптографию, дискретную математику и др. Любима как исследователями, так и студентами всего мира.

Некоторые новые Пайтоновские термины

(нумерация произвольная)

Пусть пока непонятные, но потом вы их поймете и полюбите как сам язык!

- 1 объекты, модель данных, фабрика (фабричная функция);
- 2 продакшн, деплоймент, контрибутор, непрерывная интеграция/деливеринг, трансляция в байт-код;
- 3 контейнерная последовательность(3), плоская послед-ть(5);
- 4 из/неиз-меняемая последовательность (mutable/immutable) (2 и 4). Литерал (для представления значений некот встроенных типов);
- 5 итерируемый объект, итератор, генератор (от функции с yield)
- 6 множе-во, строка, список, кортеж, словарь, послед-ть, коллекция
- 7 хешируемый(неизмен-й), списковое включение(list comprehension);
- 8 специальный метод(double underline), специальный атрибут;
- 9 динамическая (утиная) типизация, Инстансы (экземпляры);
- 10 scope, namespace (область видимости, пространство имён);
- 11 callable– оператор (), инструкция (statement), (in, (), [], in, is, !=, ==, >, ... – операторы, а for, =, while, if – инструкции);
- 12 ABC (нельзя "экземплярить", а только наследовать);
- 13 Юникод, CamelCase, snake_case, dunder (от double underline);
- 14 идентификаторы, литералы(2, 1/4), ссылки, имена(не переменные, их по классике в Питоне нет).

Кто же главный герой?
и как он “дошел” до этого?

Автор языка Python Гвидо ван Россум (1956 г.р.)
“великодушный пожизненный диктатор”, награждён
разными айти премиями, сотрудник Dropbox (с 2012)



АйТи опыт Гвидо к концу 1989 года (начало разработки)

Как он говорил про себя, про свои школу и унивЕр:

*был интровертом, нердом и увлекался в школе
“паяльнико-конструкторскими” игрушками, но в универе, в
Амстердаме (1974), “запАл” на программирование и даже
бросил унивЕр, уйдя полностью в кодинг, но потом, под
влиянием шефа, восстановился и закончил магистратуру
когда ему было уже 26 (на дворе был 1982).*

Первый проект, куда входил как исследователь — разработка языка программирования **АВС**. Но проект “не взлетел”.

Вторым была ОС Амёба, (лидер – тот самый Таненбаум).

Распределенная ОС на микроядре – не Юникс и нужен был системный язык (на Си писать не хотели). В конце концов уговорил отпустить его на инди-проект с периодическими отчетами. Учли, видно, мотив и его опыт АВС-конструирования. Был декабрь 1989.

Чем руководствовался Гвидо при разработке Python (декабрь 89 – февраль 91)

- Заимствовать идеи отовсюду, откуда это имеет смысл.
- “Вещи должны быть настолько простыми, насколько это возможно, но не проще этого.” (Эйнштейн)
- Если делать одну вещь — то делать хорошо («Философия UNIX»).
- Не заморачиваться по поводу производительности — можно будет оптимизировать в дальнейшем, если понадобится.
- Не бороться с железом, а просто плыть по течению.
- Не пытаться достичь совершенства, ибо “достаточно хорошо” — зачастую именно то, что нужно. . . .

Язык нужен был интерпретируемый, с синтаксисом похожим на английский, вместо фигур. скобок отступы, быть лёгким в обучении, быть RAD языком.

- типизация данных в языке должна быть динамическая, т.е. не требующая предварительной декларации;
- в отличие от Си модель данных (набор типов) должна быть абстрактнее (шире), но в отличие от ABC легкий и гладкий доступ на системный уровень (к файлам, процессам ОС);
- **встроенные типы** должны легко расширяться пользовательскими с помощью специальной организации и подстыковки библиотек (намек на импортируемые модули в языке Modula2 Н.Вирта);
- программисты должны выработать стиль кодирования узнаваемый, максимально одинаковый, легко читаемый, по не раз заявляемому Гвидо принципу:

"...код читается намного больше раз, чем пишется..."

(см. **PEP8** – главный документ по стилю кодирования)

Примеры идей для нового языка (лёгкая бесскобочная читаемость и обработка исключений)

//из языка ABC (много чего)

//например, как вернуть список слов в документе:

```
PUT {} IN collection
```

```
FOR line IN document:
```

```
    FOR word IN split line:
```

```
        IF word not in collection:
```

```
            INSERT word IN collection
```

```
RETURN collection
```

//из языка Modula-3 (из Modula-2 модули, Modula-3 - исключения)

```
TRY
```

```
DO.Something()
```

```
EXCEPT
```

```
    IO.Error => IO.Put('An I/O error occurred.')
```

```
END;
```

IT(80-е):экономика, образ-ие, общество(покол-е X→Y)

- hardware:** Обновилась компьютер-ная классификация (новая линейка №5 - встроенные контроллеры). IBM PC(1981). i386(1985), i486(1989). Сот телефоны (Motorola). Полноценный графический MAC(1984). Компьютеры пятого поколения (Япония, проект "не взлетел"). Архитектура VAX(DEC), в т.ч. клоны (+СЭВ), ЕС ЭВМ (РЯДЗ, СЭВ). Эльбрус-3 (ИТМиВТ, см Эль8С сегодня).
- software:** C++(1983). POSIX стандарт(1984). Продукты Borland. БД (реляц). LaTeX(1984). ADA, LabVIEW, Perl, Oberon, Tcl, Модула3.
- network:** протоколы TCP/IP(1983), DNS(1984). Довебовский медленный Internet (имейл, телеконф), Ethernet, LAN.
- education:** На ОС UNIX в Зап университетах (CompScience) проводят исслед-ия, обучение). Профессура, в основном, вся на MAC-ах. Начало преподавания информатики в СССР (1985).
- society:** Индустриал-й экономический уклад начинает переходить в информационный (США 80-ые, Зап Евр 90-е, РФ 10-е). Движение свободного ПО от R.Stallman, (GPL, 1984). В чём критерий эконом уклада? Почему "технологичность" важна?

В чем мейнстрим сегодня? А в чем он будет завтра?

- что такое *iot*, на какой элементной базе (SOC)...(по планам к 21г. на каждого из нас д.б. по 6 иот-вещей, а к 24г. – 100mbit Inet в кажд доме (97%)),
- а что есть блокчейн или эфириум?...
- что такое открытый лэджер (*hyperledger*)? в чем его преимущества?
- транзакционные издержки где? их надо уменьшать или?
- говорят, мейнстрим не в IT, а в DT (*data technology*), и это повлияет на экономику?, но как?
- говорят "основным драйвером новых рабочих мест станет сервисная индустрия" – это про что? фигня, или что-то адекватное?
- если *Web3.0* (эра социальных сетей и мессенджеров), а *Web4.0* (интернет всего, не только людей, но и механизмов), то через какое образование к этому можно подготовиться? (к 24г. число выпускников ИТ-бакалавров в стране д.б. 180 тыс. в год)

Развитие Питона – первые месяцы, годы, когда уже сложилась команда, давит ответственность, постоянное упорное выбивание грантов, переезд в США, переходы из фирмы в фирму, некоммерческий питон-Фонд - PSF

Строк Си кода в интерпретаторе Python 0.9 \approx 15000 (1991) (в феврале 1991 Гвидо продеплоил Питон в комьюнити, напомним, ядро Linux от Торвальдса – в августе 1991, дистрибутивы с середины 90-х)

Строк Си кода в интерпретаторе Python 1.5.2 \approx 23000 (1999)

Строк Си кода в интерпретаторе Python 2.2 \approx 30000 (2002)

Строк Си кода в интерпретаторе Python 3.5.3 \approx 66000 (2016)

Основные реализации:

CPython, Jython, IronPython, PyPy, Stackless

CPython (эталонная реализация языка) является **интерпретатором байт-кода**, написан на Си (первый шаг – компиляция в байт код, второй – интерпретация).

Богатая стандартная библиотека является одной из привлекательных сторон Python (батареек до фига)

Почему проект “выстрелил”

Проект “выстреливает” когда:

Возраст проекта — мин 10 лет и он активно развивается.

Опен сорс -- для всех гарантия, что не умрёт (всегда кто-то поддержит и понесет).

Сплоченное, целенаправленное комьюнити, core-team высокой квалификации (Питон комьюнити считается одной из самых эффективных в айти. Это как бы своя культура.)

Мощная финансовая поддержка заинтересованных спонсоров.

Своя философия программирования (дзен Питона)

Упертый лидер (голландец) с “поражительным эстетическим чувством дизайна языка”

...

Стоит заметить, что в кор-тим органично вписались несколько русскоязычных разработчиков (в частности, модули, связанные с *asyncio* для версии 3.5, 3.6 PEP492, PEP525, PEP530)

ABC, Python2 и Python3

Три попытки Гвидо создания языка, и, похоже, может так сложиться, что третья самая удачная, несмотря на рискованное создание несовместимой ветки языка.

Дихотомия языка, риски перехода на Pyt3 (2018? 2020?)

...

Python 3.6.2	2017-07-17	в 2018 новые проекты в обеих ветках
Python 3.5.0	2015-09-13	<i>сравниваются по числу</i> , а к 2020 Pyth3
Python 2.7.10	2015-05-23	догонит Pyth2 по производительности
Python 2.7.10	2015-05-23	по всем <u>бенчмаркам</u> и 2-ю ветку прекратят
Python 2.7.10	2015-05-23	поддерживать.
Python 3.4.3	2015-02-25	Сегодня <u>Pyth3.6</u> быстрее <u>Pyth3.5</u> и от этого
Python 2.7.9	2014-12-10	многие уже выигрывают (напр, в “числ-ных”
Python 3.4.2	2014-10-13	фреймворках: numpy, scipy, sympy, jupyter).
Python 3.3.6	2014-10-12	(Pyth3.7 уже быстрее Pyth3.6 см Pycon2017)

...

Python 2.5.4 2008-12-23
Python 2.5.3 2008-12-19
Python 2.4.6 2008-12-19
Python 2.6.1 2008-12-04
Python 3.0.0 2008-12-03
Python 2.6.0 2008-10-02

Назначение языка

Философия языка

По словам разработчиков, лучше всего использовать
Питон **в следующих областях:**

Веб-разработка (Web Development): Django, Pyramid, Bottle, Tornado, Flask, web2py...

Разработка графич_интерфейса (GUI Development): tkInter, PyGObject, PyQt, PySide, Kivy, wxPython...

Вычислительное и научное программирование (Scientific and Numeric): SciPy, Pandas, IPython Считается, что 40% в своё развитие Питон получает от мирового сайнтифик сообщества

Разработка ПО (Software Development): Buildbot, Trac, Roundup...

Системное администрирование (System Administration): Ansible, Salt, OpenStack...

... lets you work quickly and integrate systems more effectively

... powerful... and fast;

plays well with others;

runs everywhere;

is friendly & easy to learn.

На сегодня сложилось, что почти по всем прикладным направлениям более универсального инструмента, чем язык Питон не существует (imho).

Уже видно, что чем больше у комьюнити растут запросы по развитию Питон3, тем больше оно пополняется новыми сторонниками уже с новыми требованиями для новых версий языка (пусть в угоду моде, или под влиянием других языков) и так далее по циклу. Имеющийся задел и переписанный с нуля интерпретатор (к 2009) обеспечили достаточную гладкость такой цикличности, как и непрекращающиеся работы по улучшению производительности.

Философия (дзен) Питона

1. Красивое лучше, чем уродливое.
2. Явное лучше, чем неявное.
3. Простое лучше, чем сложное.
4. Сложное лучше, чем запутанное.
5. Плоское лучше, чем вложенное.
6. Разреженное лучше, чем плотное.
7. Читаемость имеет значение.
8. Особые случаи не настолько особые, чтобы нарушать правила.
9. При этом практичность важнее безупречности.
10. Ошибки никогда не должны замалчиваться.
11. Если не замалчиваются явно.
12. Встретив двусмысленность, отбрось искушение угадать.
13. Должен существовать один — и, желательно, только один — очевидный способ сделать это.
14. Хотя он поначалу может быть и не очевиден, если вы не голландец (шут. намёк на Гвидо).
15. Сейчас лучше, чем никогда.
16. Хотя никогда зачастую лучше, чем прямо сейчас.
17. Если реализацию сложно объяснить — идея плоха.
18. Если реализацию легко объяснить — идея, возможно, хороша.
19. Пространства имён — отличная штука! Будем делать их побольше!

Zen Питона встроен в библиотечный модуль *this* и его можно вызвать через импорт модуля

```
>>> import this
```

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one – and preferably only one – obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea – let's do more of those!

Итак! (выводы)

Python: высокоуровневый универсальный язык программирования.

Обеспечивает:

- повышение производительности разработчика,
- высокую читаемость кода. Имеет:
- богатую стандартную библиотеку и
- “очень” богатую стороннюю библиотеку.

построен и исполняется как интерпретируемый, с динамической, сильной типизацией.

лицензия, платформа: PSF License, (опенсорс), кроссплатформенное ПО.

уровень зрелости и перспективы развития: версия Python 3.6.2 (07.2017), Python 3.5.3 и 3.4.6 (01.2017), Python2.7.14rc1(08.2017) версия Python 3.7.0 (ожидается к 07.2018), версии Python 2.8 (не будет никогда)

Комьюнити, лидер: сложившееся данное сочетание на сегодня можно охарактеризовать как: удачное, целеустремленное, прагматичное, результативное.

the End по Лекции 1, вопросы?

Далее

Вопросы(для экзамена)/упражнения(для лабораторных)

- ❶ Чем сисадмин отличается от дев-опс инженера? Покажите на пальцах динамику решения проблем в девопсе за последние 10 лет. Если девопс набор операций, то какая Вам понятней и ближе всего?
- ❷ В приведенных на 6 слайде, пусть условных, направлениях программирования вам понятней и ближе всего какое, и почему? Как оно связано с тем, в чём Вы собираетесь специализироваться? Если ничего пока Вас не цепляет, то обоснуйте свою мотивацию?
- ❸ Если **бигдатовые возможности** языков опираются на работу с большими массивами - последовательностями то, исходя из определения последовательности(массива) из школьной алгебры, языка C, языка Питон, скажите главную (фишечную) часть каждого из этих трёх определений? Эта часть реализована в специальном очень распространенном протоколе во многих языках программирования (а уж, тем более, в Питоне), отрефлексируйте, плиз?

- 4 Питон язык молодой, старый или зрелый и что значит зрелый или старый?, Питон как проект, состоявшийся или еще нет? Какие критерии у состоявшегося проекта, а какие у несост..., и насколько они достоверные по-вашему? И, если состоявшийся, то так ли должно быть у любого стартапа или не обязательно?
- 5 если этот язык любят ученые всего Мира, то почему он не конкурент C++ или Джава?, за какие фИчи языка его они любят?
- 6 сильная (строгая) и динамическая типизация в языке возможна?
- 7 если язык интерпретируемый, то у него всегда ли должна быть одноэтапность в исполнении исходного кода: сделал шаг – исполнил, сделал другой – исполнил, так или не совсем так?
- 8 если в *Python3* кроме прочего, *print* из команды превратился а функцию, то, какой из встроенных типов, по-вашему, подвергся основной переработке при переписывании языка с нуля?
- 9 дзен Питона – 19 умозаключений, Вам какое ближе всего? Дайте не простой ответ, плиз!

- 10 Комьюнити ждет Питон3.7, судя по трендам его развития, чего можно ожидать? (любители PyCharm, а также самых последних версий Django должны догадаться... Какая-то отгадка (не вся) у докладчика с русскоязычной фамилией на PyCon2017-portland)
- 11 Представьте, что Вы девелопер ПО. Тогда скажите какая из моделей архитектур разработки/исполнения софтвера вам ближе всего и почему? (см какой-то там слайд выше). У каждой есть свои плюсы и минусы – назовите, плиз?
- 12 Если Питон называют RAD языком, то перечислите хотя бы 4 фишки, показывающие его 'RAD-ность'?
- 13 В чём **критерий** того или иного экономического уклада? Почему **"технологичность"** важна? И какая супер технология сейчас переворачивает нынешний эконом уклад и почему нынешнему молодому поколению (поколению Z) надо это знать и уметь? (вопрос немного из макроэкономики, но на уровне школьника).